ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM

BÁO CÁO ĐỒ ÁN

MÔN SEMINAR VÀ CÁC VẤN ĐỀ HIỆN ĐẠI

Đề tài: Phát Triển Môi Trường CI/CD Tự Động Hóa với Jenkins

Giảng viên hướng dẫn: Đinh Nguyễn Anh Dũng

Lớp: SE400.P11.PMCL

Sinh viên thực hiện:

- Nguyễn Thiện - 21521461
- Vũ Đức Minh - 21522348

TP. Hồ Chí Minh, Ngày 11 Tháng 11 Năm 2024

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

……, ngày…….....tháng……năm 20…

Người nhận xét

(Ký tên và ghi rõ họ tên)

BẢNG PHÂN CÔNG, ĐÁNH GIÁ THÀNH VIÊN:

Họ và tên	MSSV	Phân công	Đánh giá
Nguyễn Thiện	21521461		
Vũ Đức Minh	21522348		

Mục lục

Chương 1 Tổng quan
1. Giới thiệu..... 7
1.1 Đặt vấn đề..... 7
1.2 Mục tiêu báo cáo..... 7
1.3 Phạm vi nghiên cứu..... 7
2. Tổng quan về CI/CD và Jenkins..... 8
2.1 Khái niệm CI/CD..... 8
2.2 Tại sao phải quan tâm đến CI/CD?..... 8
2.3 So sánh tổng quan về các framework..... 8
2.4 CI/CD Pipeline chi tiết cho từng môi trường..... 9
2.5 Mô hình quản lý code (Git Flow)..... 12
Chương 2: Quy trình thực hiện
1. Mô tả tổng quan:..... 13
2. Các bước chi tiết:..... 13
a. Thiết lập Git Flow..... 13
b. Thiết lập Webhook để Github kích hoạt Pipeline khi có sự kiện:..... 15
c. Cài đặt AWS phục vụ pipeline..... 15
d. Cài đặt Jenkins trên AWS..... 15
e. Cấu hình Docker..... 16
f. Cấu hình SonarQube..... 16
g. Xây dựng pipeline Jenkins..... 17
h. Tích hợp thông báo qua Email..... 19
Chương 3: Demo
1. Giới thiệu demo (Introduction to the Demo)..... 21
2. Thực hiện demo (Demo Execution)..... 21
a. Kích hoạt pipeline..... 21
b. Hiện trạng ban đầu của ứng dụng (Hello World):..... 22
c. Quy trình pipeline hoạt động..... 22
d. Ứng dụng được deploy lên docker hub..... 25
e. Thông báo được gửi về cho người dùng..... 26
f. Ứng dụng sau khi được deploy lên docker hub..... 26
g. Hiện trạng ứng dụng sau khi pipeline chạy thành công (Hello World Minh)..... 27
Chương 4: Ứng dụng/ Kết quả thực nghiệm/So sánh-Đánh giá
1. Kết quả thực nghiệm..... 28
2. So sánh-Đánh giá..... 28
Chương 5: Kết luận
1. Ưu điểm..... 29
2. Khuyết điểm..... 29
3. Hướng phát triển..... 29

Chương 1: Tổng quan

1. Giới thiệu

1.1 Đặt vấn đề

Trong thời đại công nghệ số hiện nay, nhu cầu sử dụng các hệ thống quản lý học sinh ngày càng tăng cao, kéo theo yêu cầu về hiệu quả, tính linh hoạt và khả năng mở rộng của các giải pháp công nghệ trong ngành giáo dục. Việc phát triển hệ thống quản lý học sinh không chỉ dừng lại ở việc lưu trữ và quản lý dữ liệu, mà còn phải đảm bảo khả năng truy cập nhanh chóng, tính bảo mật cao và đáp ứng trải nghiệm người dùng tốt nhất trên nhiều môi trường khác nhau. Trong bối cảnh đó, tích hợp Jenkins CI/CD với kiến trúc đa môi trường đã nổi lên như một giải pháp tối ưu, giúp các nhà phát triển hệ thống dễ dàng triển khai, kiểm thử và duy trì hệ thống một cách hiệu quả, từ đó đáp ứng nhu cầu đa dạng của người dùng một cách nhanh chóng và linh hoạt

1.2 Mục tiêu báo cáo

Mục tiêu của báo cáo này là trình bày rõ ràng về kiến trúc đa môi trường và quy trình tích hợp Jenkins CI/CD trong phát triển hệ thống quản lý học sinh, đồng thời làm rõ các lợi ích của việc áp dụng các giải pháp này. Báo cáo cũng sẽ phân tích các thách thức và đề xuất các giải pháp khi triển khai Jenkins CI/CD trong hệ thống quản lý học sinh, cũng như đưa ra các ví dụ thực tiễn về những tổ chức đã thành công trong việc ứng dụng kiến trúc đa môi trường và Jenkins CI/CD vào quy trình quản lý và phát triển hệ thống của họ

1.3 Phạm vi nghiên cứu

Báo cáo sẽ tập trung vào việc phân tích và đánh giá kiến trúc đa môi trường cùng quy trình tích hợp Jenkins CI/CD trong bối cảnh phát triển hệ thống quản lý học sinh. Các vấn đề liên quan đến quá trình phát triển, triển khai, bảo trì và nâng cấp hệ thống cũng sẽ được đề cập. Đồng thời, báo cáo sẽ giới thiệu những xu hướng mới trong lĩnh vực công nghệ phần mềm có liên quan đến kiến trúc đa môi trường và tự động hóa CI/CD nhằm tối ưu hóa hiệu quả quản lý và vận hành hệ thống quản lý học sinh..

2. Tổng quan về CI/CD và Jenkins

2.1 Khái niệm CI/CD

CI/CD là một bộ đôi công việc, bao gồm CI (Continuous Integration) và CD (Continuous Delivery) hoặc Continuous Deployment (Triển khai liên tục), ý nói là quá trình tích hợp (integration) thường xuyên, nhanh chóng hơn khi code cũng như thường xuyên cập nhật phiên bản mới (delivery)

2.2 Tại sao phải quan tâm đến CI/CD?

Ngày nay, với xu hướng agile/lean dẫn đến việc phát triển tính năng là điều bình thường, quan trọng là phải nhanh. Nếu một tính năng mà mất 2, 3 tháng mới release thì dẫn đến nhiều hệ lụy như làm không phù hợp nhu cầu khách hàng, mất đi cái lợi thế dẫn đầu. Do đó, việc làm ra một sản phẩm, tính năng đòi hỏi thần tốc là ưu tiên số một hiện nay.

Bên cạnh đó, để nhanh chóng ra mắt một tính năng, phiên bản mới nếu theo cách cổ điển sẽ mất nhiều thời gian bởi công việc chân tay khá nhiều và mỗi lần release cũng huy động một cơ số người không nhỏ để cập nhật một thay đổi dù là nhỏ nhất. Bởi vậy, xu hướng CI/CD giúp cung cấp các framework, workflow giúp tiết kiệm thời gian, nguồn lực của quá trình release

2.3 So sánh tổng quan về các framework

Tiêu chí	Spring Boot (Java)	Node.js (Express)	Django (Python)	Go (Gin/Gorilla)	ASP.NET Core (C#)
Hiệu năng	Cao, ổn định với JVM	Tốt cho I/O, hạn chế về tính	Tốt cho I/O, không mạnh về	Hiệu năng cao, ngôn ngữ biên	Hiệu năng cao với .NET tối

		toán nặng	CPU	dịch	ưu hóa
Dễ sử dụng	Phức tạp hơn, cần hiểu về Spring	Dễ học, ít cấu hình	Dễ học, Python đơn giản	Hơi phức tạp hơn với Goroutines	Dễ cho người quen .NET
Mở rộng	Rất tốt cho hệ thống lớn, doanh nghiệp	Tốt với microservices nhỏ	Tốt cho ứng dụng vừa và nhỏ	Rất tốt, phù hợp hệ thống lớn	Mở rộng tốt, tích hợp dịch vụ Microsoft

Lý do sử dụng Spring Boot:

- Tính mở rộng và ổn định: Spring Boot dễ mở rộng và phù hợp cho các hệ thống doanh nghiệp.
- Tích hợp dễ dàng: Hỗ trợ nhiều công cụ và dịch vụ phổ biến như Jenkin, Docker và các hệ thống cơ sở dữ liệu.
- Cộng đồng lớn và tài liệu phong phú: Nhiều tài liệu và ví dụ thực tiễn giúp quá trình học tập và phát triển dễ dàng hơn.
- Tích hợp bảo mật: Hỗ trợ mạnh mẽ các giải pháp bảo mật như OAuth2, JWT, giúp xây dựng hệ thống bảo mật hiệu quả.

2.4 CI/CD Pipeline chi tiết cho từng môi trường

Môi trường Development (Dev)

Pipeline chi tiết:

Kiểm tra chất lượng mã nguồn:

- Chạy các công cụ kiểm tra mã như ESLint (cho React) và Checkstyle (cho Java/Spring Boot).
- Đảm bảo mã không có lỗi cú pháp và tuân thủ coding standard.

Build ứng dụng:

- React: Sử dụng npm hoặc yarn để build ứng dụng React.

- Spring Boot: Sử dụng Maven hoặc Gradle để build backend Spring Boot.

Kiểm thử đơn vị (Unit Tests):

- Chạy các bài kiểm thử đơn vị (các function/method riêng lẻ) cho cả React và Spring Boot.

Triển khai lên môi trường Development:

- Sau khi các bước trên thành công, ứng dụng được tự động triển khai lên server Development bằng Docker Compose.

Thông báo:

- Gửi thông báo qua email hoặc Slack cho nhóm phát triển để biết rằng quá trình đã thành công hay thất bại.

Môi trường Staging

Pipeline chi tiết:

- Trigger: Mỗi khi có merge vào branch staging.

Build ứng dụng với cấu hình Production:

- Sử dụng các biến môi trường và cấu hình Production (ví dụ, sử dụng API keys cho production).

Kiểm thử smoke (Smoke Tests):

- Chạy kiểm thử nhanh để đảm bảo các chức năng chính hoạt động bình thường (chẳng hạn, trang chủ của ứng dụng phải load được).

Triển khai lên Production:

- Triển khai ứng dụng lên môi trường Production bằng Helm.
- Sử dụng chiến lược Canary Deployment hoặc Blue-Green Deployment để giảm thiểu rủi ro.

Giám sát và cảnh báo:

- Kết nối với các hệ thống giám sát như Prometheus và Grafana để theo dõi hiệu

suất và tình trạng của hệ thống.

Thông báo:

- Gửi thông báo đến các bên liên quan về việc triển khai thành công hoặc thất bại

Môi trường Production

Mục tiêu:

- Đảm bảo các thay đổi đã được kiểm tra kỹ lưỡng, có thể triển khai an toàn mà không làm gián đoạn hệ thống.

Pipeline chi tiết:

Trigger:

- Mỗi khi có merge vào branch master hoặc main.

Build ứng dụng với cấu hình Production:

- Sử dụng các biến môi trường và cấu hình Production (ví dụ, sử dụng API keys cho production).

Kiểm thử smoke (Smoke Tests):

- Chạy kiểm thử nhanh để đảm bảo các chức năng chính hoạt động bình thường (chẳng hạn, trang chủ của ứng dụng phải load được).

Triển khai lên Production:

- Triển khai ứng dụng lên môi trường Production bằng Kubernetes hoặc Helm.
- Có thể sử dụng chiến lược Canary Deployment hoặc Blue-Green Deployment để giảm thiểu rủi ro.

Giám sát và cảnh báo:

- Kết nối với các hệ thống giám sát như Prometheus và Grafana để theo dõi hiệu suất và tình trạng của hệ thống.

Thông báo:

- Gửi thông báo đến các bên liên quan về việc triển khai thành công hoặc thất bại.

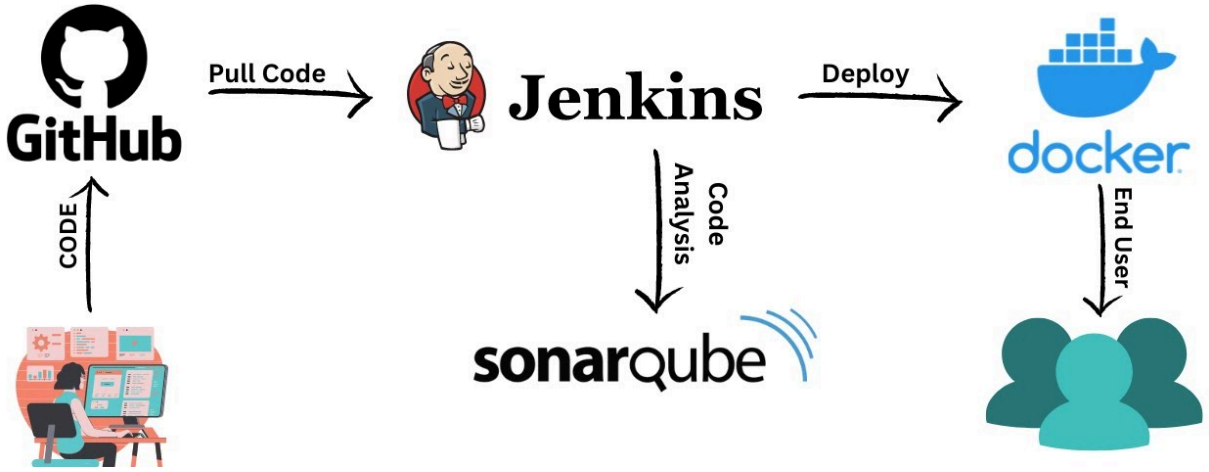
2.5 Mô hình quản lý code (Git Flow)

- Main: Đây là nhánh chứa mã nguồn đã qua kiểm duyệt và sẵn sàng để triển khai lên Production.
- Develop: Nhánh phát triển chính, nơi chứa mã nguồn đang được phát triển và thử nghiệm, nhưng chưa sẵn sàng cho Production.
- Feature branches: Dành cho các tính năng hoặc cải tiến mới. Mỗi tính năng sẽ có một nhánh riêng biệt, giúp tránh xung đột và dễ dàng quản lý.
- Release branches: Dùng để chuẩn bị cho quá trình phát hành. Nhánh này được tạo khi mã nguồn trên nhánh Develop đã hoàn thiện và sẵn sàng cho các bước kiểm tra cuối cùng.
- Hotfix branches: Được sử dụng để sửa lỗi khẩn cấp trên môi trường Production, giúp đội ngũ nhanh chóng phát hiện và khắc phục lỗi nghiêm trọng khi cần

Chương 2: Quy trình thực hiện

1. **Mô tả tổng quan:**

- Mục tiêu dự án: Mục tiêu của dự án này là xây dựng một pipeline CI/CD tự động sử dụng Jenkins, giúp tối ưu hóa quy trình phát triển và triển khai phần mềm trong môi trường doanh nghiệp trên nền tảng AWS. Việc này giúp cải thiện tốc độ triển khai và chất lượng phần mềm nhờ việc kiểm thử và kiểm tra liên tục.
- Sơ đồ quy trình: Pipeline bao gồm các giai đoạn chính như kiểm tra mã nguồn (checkout code), build dự án bằng Maven, phân tích mã nguồn với SonarQube, tạo Docker image, và đẩy image lên Docker Hub.

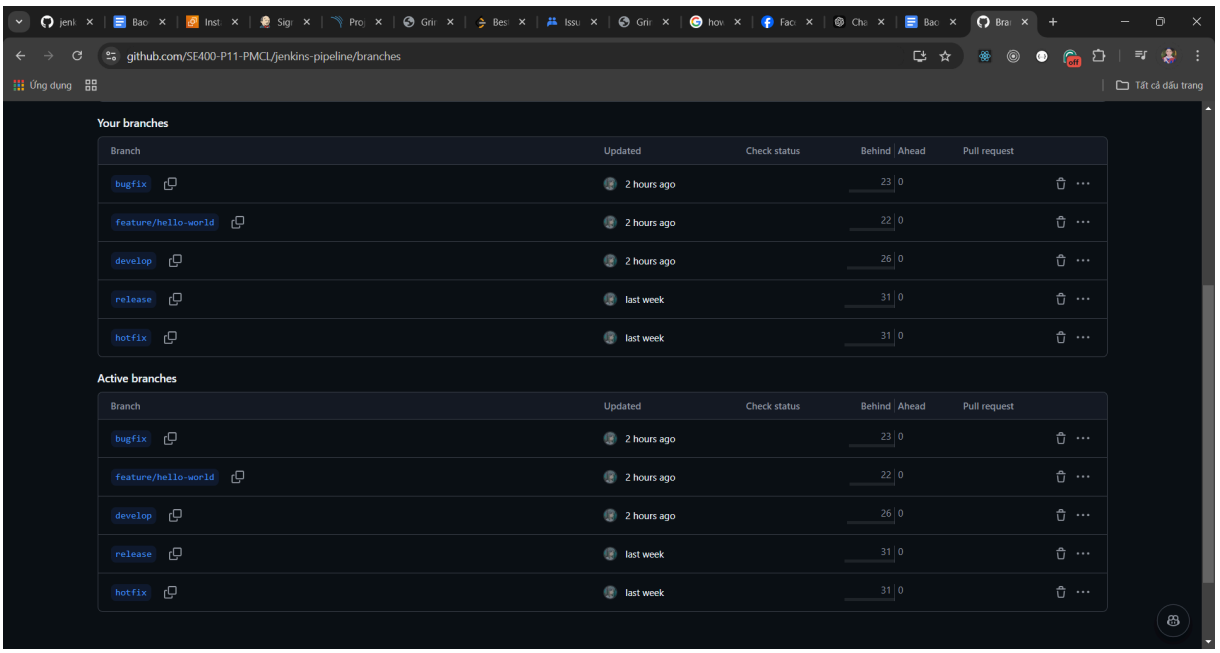


2. **Các bước chi tiết:**

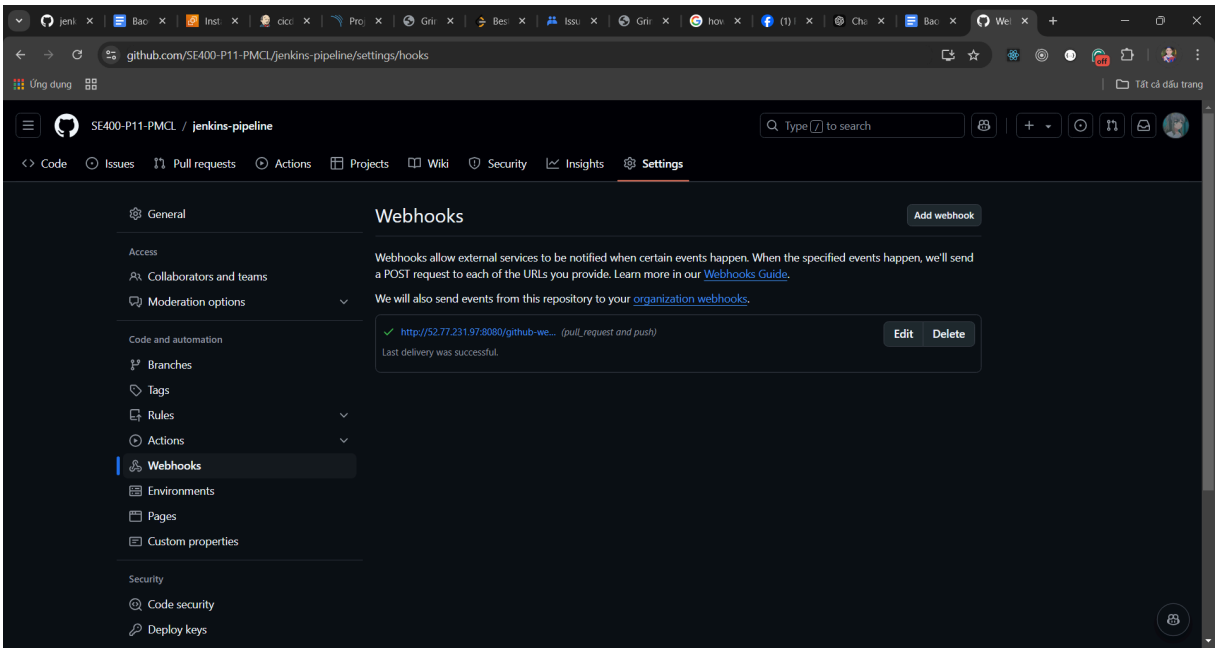
a. Thiết lập Git Flow

Quá trình: Git Flow được thiết lập để quản lý việc phát triển các nhánh mã nguồn chính và nhánh chức năng, giúp dễ dàng theo dõi sự phát triển và tích hợp của các tính năng mới.

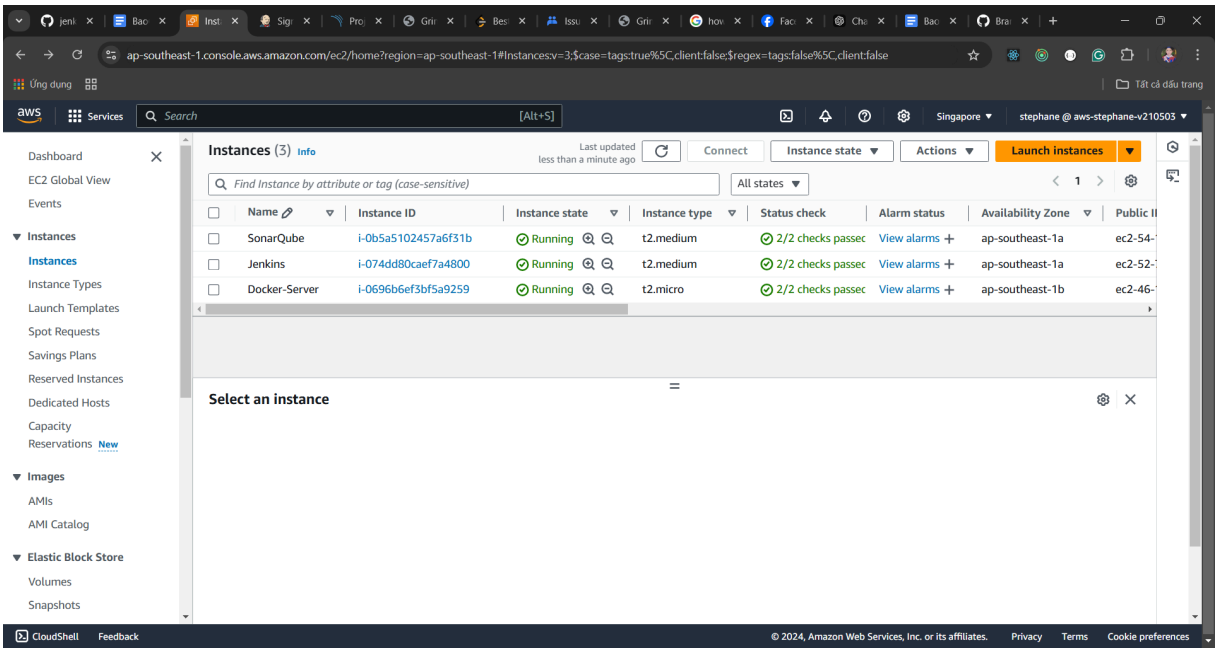
Lợi ích: Dễ dàng tích hợp với Jenkins để tự động phát hiện các thay đổi mã nguồn và khởi động pipeline.


github.com/SE400-P11-PMCL/jenkins-pipeline/branches
Your branches
Branch
Updated
Check status
Behind
Ahead
Pull request
bugfix
2 hours ago
23
0
feature/hello-world
2 hours ago
22
0
develop
2 hours ago
26
0
release
last week
31
0
hotfix
last week
31
0
Active branches
Branch
Updated
Check status
Behind
Ahead
Pull request
bugfix
2 hours ago
23
0
feature/hello-world
2 hours ago
22
0
develop
2 hours ago
26
0
release
last week
31
0
hotfix
last week
31
0

b. Thiết lập Webhook để Github kích hoạt Pipeline khi có sự kiện:



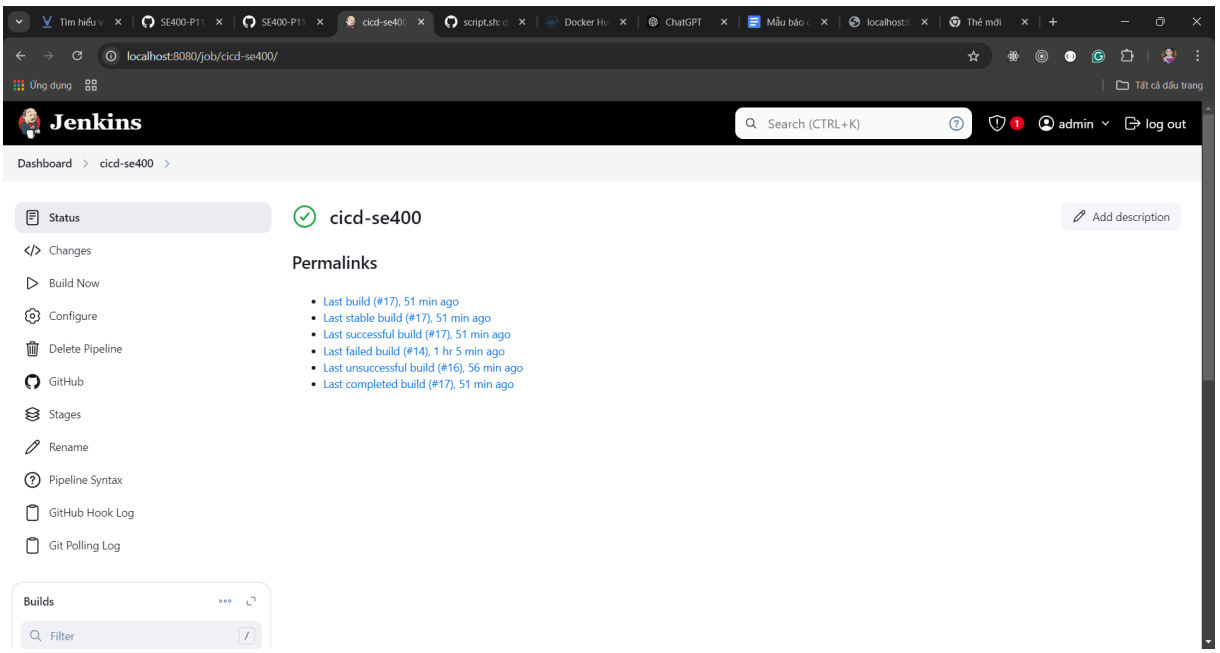
c. Cài đặt AWS phục vụ pipeline



d. Cài đặt Jenkins trên AWS

Các bước thực hiện:

- Chọn một instance EC2 thích hợp và cài đặt hệ điều hành Ubuntu.
- Cài đặt Java (yêu cầu phiên bản 17 trở lên) và Jenkins.
- Cấu hình Jenkins để truy cập thông qua trình duyệt và cài đặt các plugin cần thiết như Git, Maven Integration, và Docker Pipeline.



e. Cấu hình Docker

Các bước thực hiện:

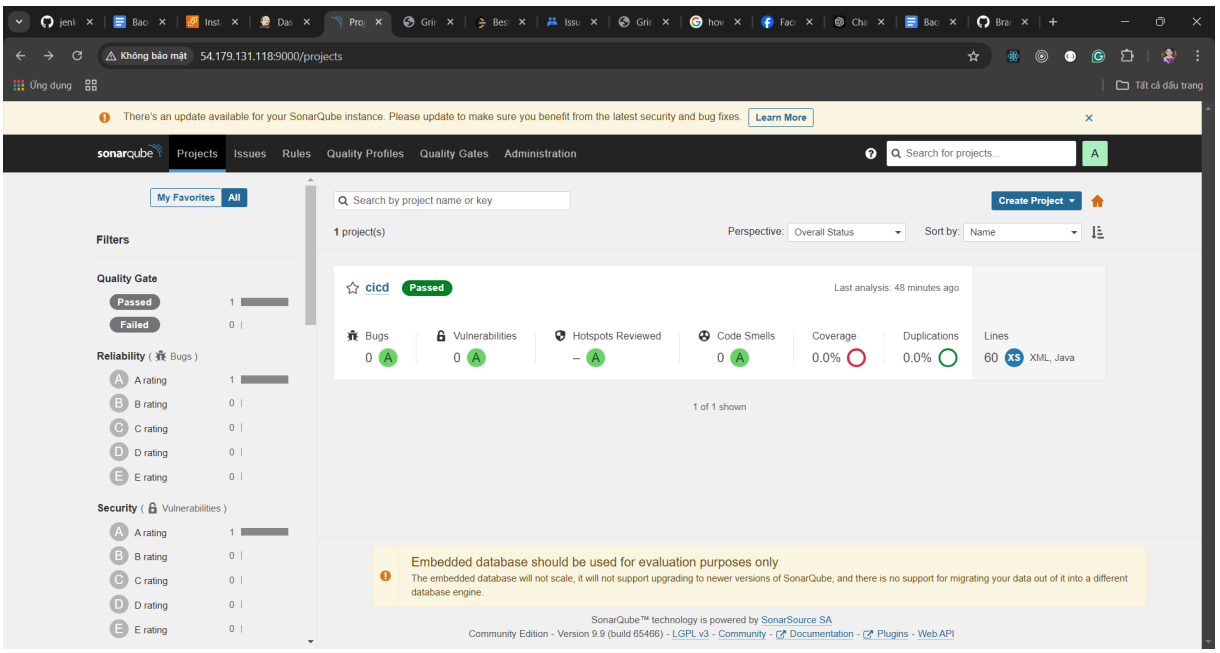
Cài đặt Java 17 và Docker

Câu hình Docker để kết nối tới Jenkins server

f. Cấu hình SonarQube

Cài đặt SonarQube: Cài đặt SonarQube trên một instance EC2 riêng biệt và cấu hình để kết nối với Jenkins.

Tích hợp với Jenkins: Tạo và sử dụng token bảo mật cho việc phân tích mã nguồn. Cấu hình trong Jenkins để sử dụng plugin SonarQube Scanner.



g. Xây dựng pipeline Jenkins

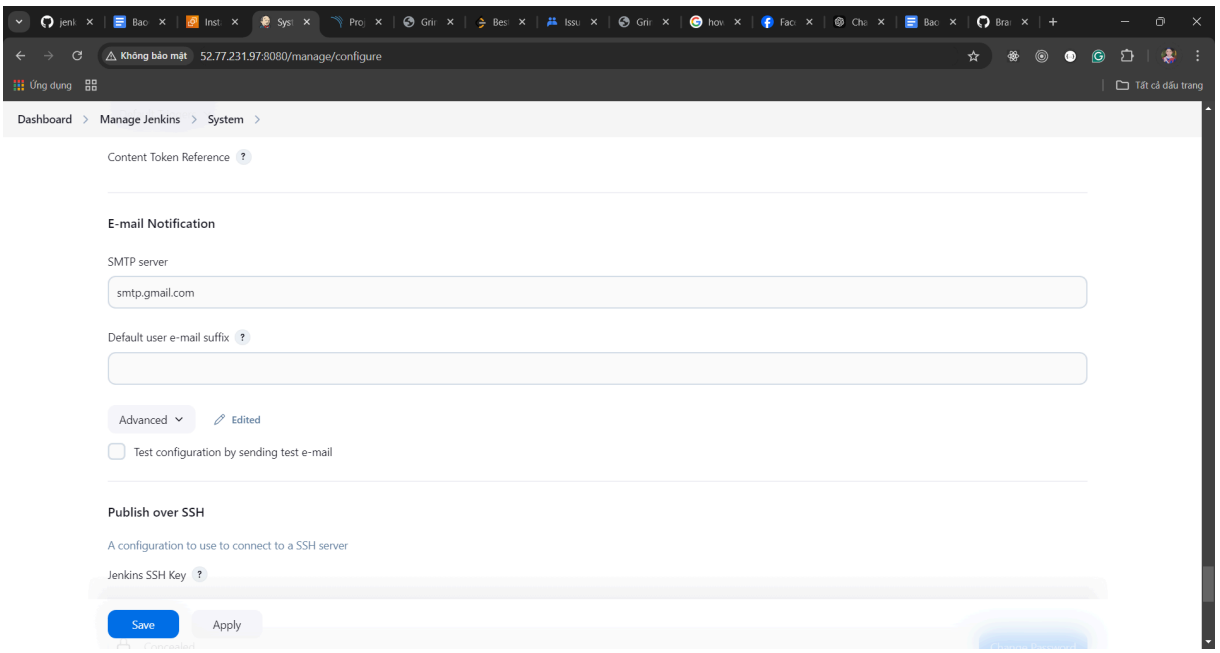
```
pipeline {

    agent any

    environment {

        SONAR_TOKEN = credentials('sonartoken')

    }

    tools {

        maven 'maven_tool'

    }

    stages {

        stage('Checkout Code') {

            steps {

                checkout scmGit(branches: [[name: '*/main']], extensions: [],
userRemoteConfigs: [[url:
'https://github.com/SE400-P11-PMCL/jenkins-pipeline']])

            }

        }

        stage('Build Maven') {

            steps {
```

```
                sh 'mvn clean install'
            }
        }
        stage('SonarQube Analysis') {
            steps {
                script {
                    sh """
                        mvn sonar:sonar \
                        -Dsonar.projectKey=cicd-se400 \
                        -Dsonar.host.url=http://54.179.131.118:9000 \
                        -Dsonar.login=${SONAR_TOKEN}
                    """
                }
            }
        }
        stage('Build docker image') {
            steps {
                sh 'docker build -t ducminh210503/cicd-se400 .'
            }
        }
        stage('Push image to Hub') {
            steps {
                withCredentials([string(credentialsId: 'dockerhub-pwd',
variable: 'dockerhubpwd')]) {
                    sh 'docker login -u ducminh210503 -p ${dockerhubpwd}'
                }
                sh 'docker tag ducminh210503/cicd-se400
ducminh210503/cicd-se400'
                sh 'docker push ducminh210503/cicd-se400'
            }
```

```
        }
    }
    post {
        success {
            emailext(
                subject: 'Pipeline Succeeded',
                body: 'Pipeline has succeeded.',
                to: 'vuducminh210503@gmail.com'
            )
        }
        failure {
            emailext(
                subject: 'Pipeline Failed',
                body: 'Pipeline has failed.',
                to: 'vuducminh210503@gmail.com'
            )
        }
    }
}
```

h. Tích hợp thông báo qua Email

Cách cấu hình: Sử dụng plugin Email Extension để gửi thông báo về kết quả chạy pipeline đến hộp thư đã định sẵn.


Không bảo mật
52.77.231.97:8080/manage/configure
Ứng dụng
Tất cả dấu trang
Dashboard
Manage Jenkins
System
Content Token Reference
E-mail Notification
SMTP server
smtp.gmail.com
Default user e-mail suffix
Advanced
Edited
Test configuration by sending test e-mail
Publish over SSH
A configuration to use to connect to a SSH server
Jenkins SSH Key
Save
Apply

Chương 3: Demo

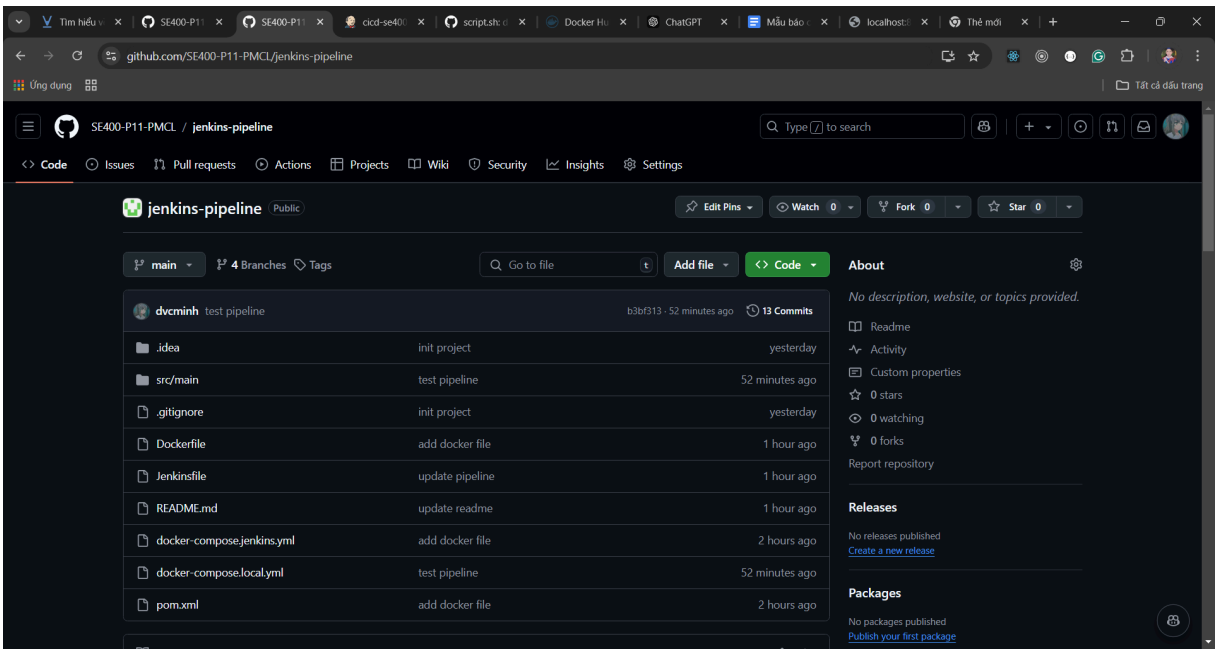
1. Giới thiệu demo (Introduction to the Demo)

1. Mục tiêu: Minh họa cách pipeline hoạt động từ đầu đến cuối để có cái nhìn trực quan về quy trình CI/CD.
2. Tóm tắt pipeline:
    a. Pipeline hoạt động khi có sự kiện xảy ra trên main branch
    b. Chạy pipeline: Bắt đầu chạy pipeline và giải thích từng giai đoạn khi chạy. Thể hiện giao diện Jenkins và các bước như:
        i. Checkout Code: Kiểm tra mã nguồn từ GitHub.
        ii. Build Maven: Thực hiện lệnh build, test và kiểm tra kết quả.
        iii. SonarQube Analysis: Hiển thị kết quả phân tích mã nguồn trên giao diện SonarQube.
    c. Build và đẩy Docker image: Minh họa quá trình tạo Docker image và đẩy lên Docker Hub.
    d. Kiểm tra thông báo qua email: Trình bày email thông báo đã gửi khi pipeline hoàn thành.

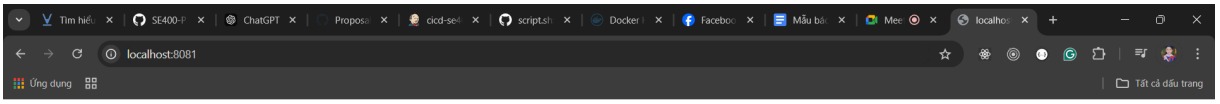
2. Thực hiện demo (Demo Execution)

a. Kích hoạt pipeline

Jenkins tự động kiểm tra mã nguồn từ kho lưu trữ (repository) và thực hiện quá trình build. Khi có sự kiện được kích hoạt trên branch main, pipeline sẽ được gọi bởi Github Webhooks



b. Hiện trạng ban đầu của ứng dụng (Hello World):



Hello World

c. Quy trình pipeline hoạt động

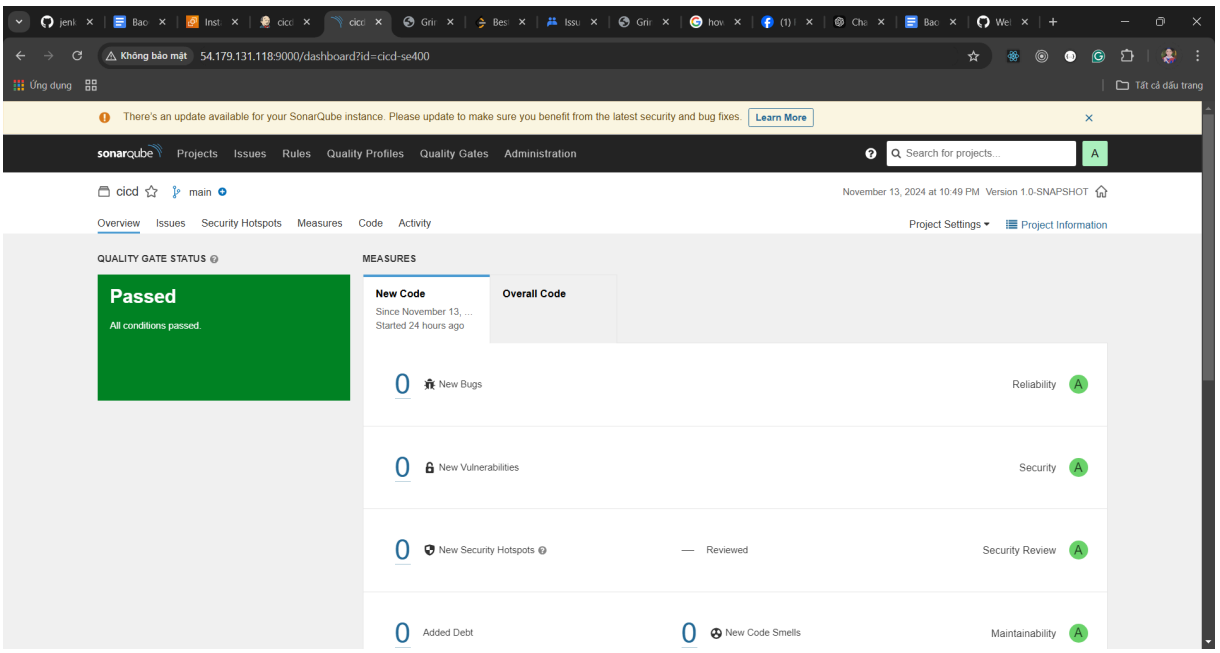
Sau khi được kích hoạt, đầu tiên pipeline sẽ checkout code từ github

```
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/cicd-se400/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/SE400-P11-PMCL/jenkins-pipeline # timeout=10
Fetching upstream changes from https://github.com/SE400-P11-PMCL/jenkins-pipeline
 > git --version # timeout=10
 > git --version # 'git version 2.34.1'
 > git fetch --tags --force --progress -- https://github.com/SE400-P11-PMCL/jenkins-pipeline +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 7ecb47a9b65b72f33dc9dc45bb421fe4419c609b (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 7ecb47a9b65b72f33dc9dc45bb421fe4419c609b # timeout=10
Commit message: "Update README.md"
```

Build Maven: Thực hiện lệnh build và test.

```
+ mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------------------< org.minh:cicd >----------------------------
[INFO] Building cicd 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.3.2:clean (default-clean) @ cicd ---
[INFO] Deleting /var/lib/jenkins/workspace/cicd-se400/target
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ cicd ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ cicd ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 2 source files with javac [debug parameters release 17] to target/classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ cicd ---
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/cicd-se400/src/test/resources
```

SonarQube Analysis: Hiển thị kết quả phân tích mã nguồn trên giao diện SonarQube.



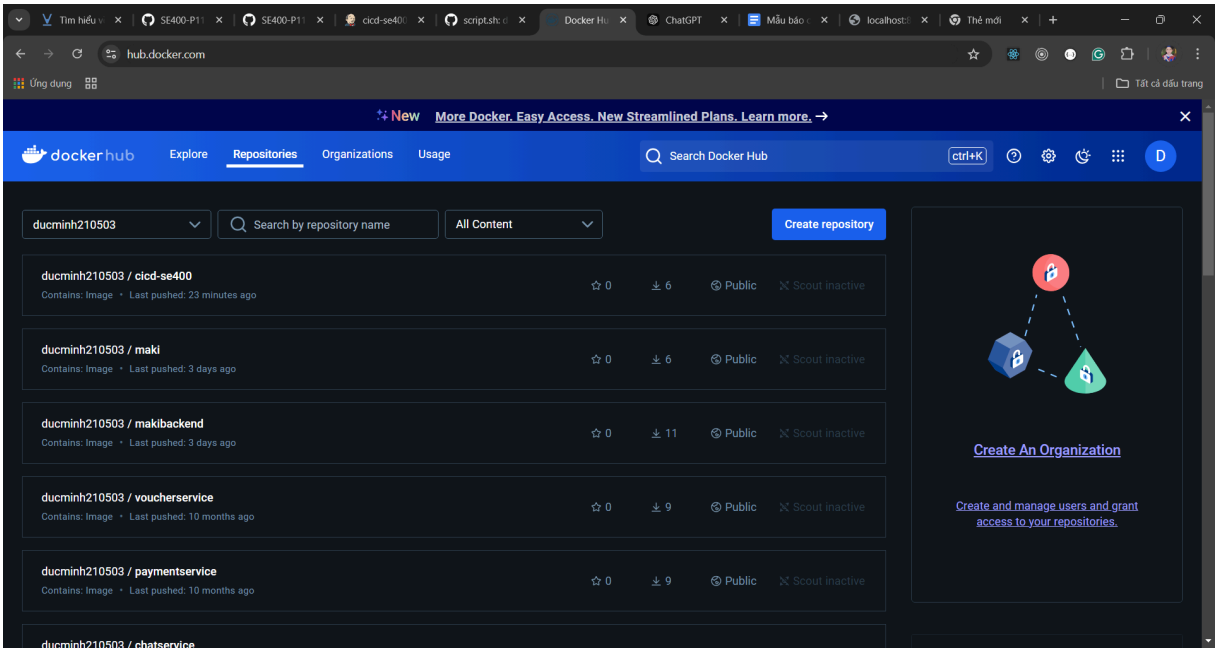
Xây dựng và tag container Docker

Sau khi build mã nguồn thành công, Jenkins sẽ tạo container Docker cho ứng dụng.

Pipeline hoạt động để đẩy container đã được tag theo tên tài khoản và tên image lên Docker Hub để deploy.

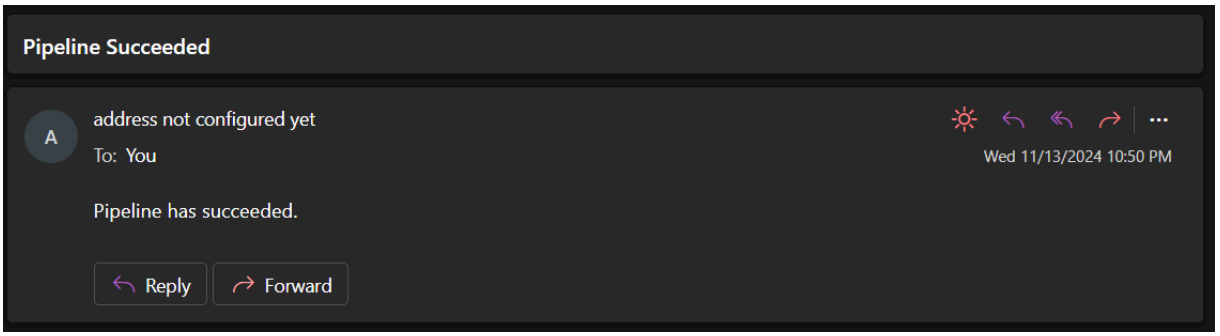
```
+ docker build -t ducminh210503/cicd-se400 .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  20.65MB

Step 1/5 : FROM openjdk:17-jdk-slim
 ---> 37cb44321d04
Step 2/5 : WORKDIR /app
 ---> Using cache
 ---> 11e2917aac39
Step 3/5 : COPY target/cicd-se400.jar .
 ---> f99c2ef8de93
Step 4/5 : EXPOSE 8081
 ---> Running in 86325ae1ccc3
Removing intermediate container 86325ae1ccc3
 ---> 8f9ba4a2b161
Step 5/5 : ENTRYPOINT ["java","-jar","cicd-se400.jar"]
 ---> Running in 0cd8597b80c7
Removing intermediate container 0cd8597b80c7
 ---> 00b7386f4d20
Successfully built 00b7386f4d20
Successfully tagged ducminh210503/cicd-se400:latest
```

d. Ứng dụng được deploy lên docker hub



e. Thông báo được gửi về cho người dùng

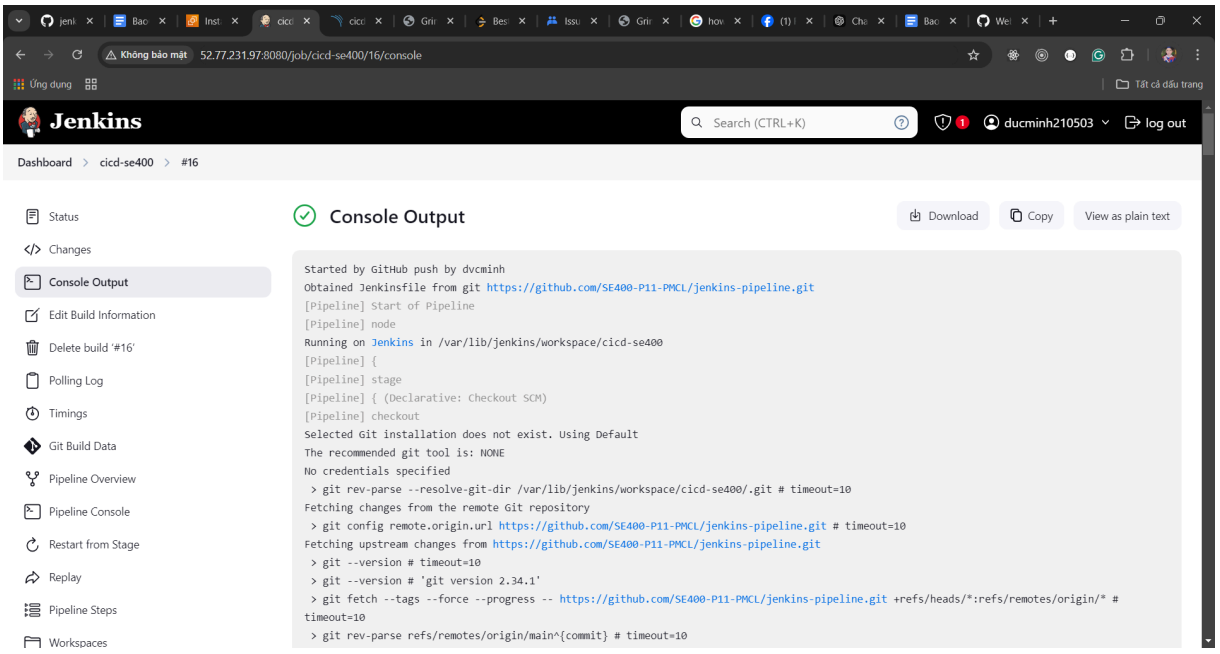
Kết quả của pipeline sẽ được gửi về thông qua email của người dùng



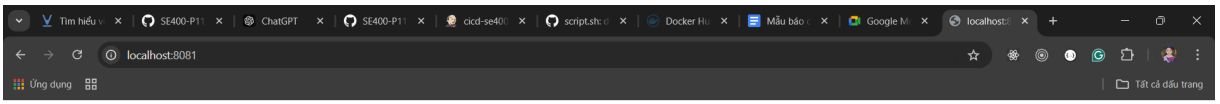
f. Ứng dụng sau khi được deploy lên docker hub

Image được deploy lên Dockerhub sẽ được sử dụng bởi mọi người bằng cách chạy file docker compose

Kết quả pipeline chạy thành công



g. Hiện trạng ứng dụng sau khi pipeline chạy thành công (Hello World Minh)



Hello World Minh

Chương 4: Ứng dụng/ Kết quả thực nghiệm/So sánh-Đánh giá

1. Kết quả thực nghiệm

Trong quá trình thực hiện và thử nghiệm pipeline CI/CD, hệ thống đã vận hành đúng như mong đợi với các kết quả sau:

- Tự động hóa hoàn toàn: Quy trình build, kiểm thử, tạo container, và đẩy container lên Docker Hub được thực hiện tự động mỗi khi có thay đổi được đẩy lên kho lưu trữ.
- Thời gian phản hồi nhanh: Toàn bộ quá trình pipeline diễn ra trong thời gian ngắn, giúp phát hiện lỗi và cung cấp phản hồi nhanh chóng đến nhóm.
- Thông báo kịp thời: Tích hợp gửi email giúp nhóm phát triển nhận được thông báo ngay khi pipeline hoàn tất, tạo điều kiện xử lý kịp thời trong trường hợp thất bại.

2. So sánh-Đánh giá

So sánh với quy trình truyền thống (build và triển khai thủ công):

- Tốc độ: Quy trình tự động hóa nhanh hơn đáng kể so với quy trình thủ công nhờ giảm thiểu thao tác lặp đi lặp lại.
- Tính chính xác: Giảm thiểu lỗi do thao tác thủ công, đảm bảo tính nhất quán trong các bước triển khai.
- Khả năng mở rộng: Pipeline dễ dàng mở rộng và thêm các bước mới như kiểm thử tự động, kiểm tra bảo mật, triển khai tự động.

Chương 5: Kết luận

1. Ưu điểm

Tự động hóa toàn diện: Giúp tiết kiệm thời gian và công sức, giảm thiểu các thao tác thủ công.

Thông báo kịp thời: Tích hợp thông báo qua email giúp nhóm phát triển luôn cập nhật tình trạng build.

Khả năng tích hợp mở rộng: Dễ dàng tích hợp thêm các công cụ và plugin khác để tăng cường khả năng của pipeline.

Quản lý version tốt hơn: Nhờ vào việc tự động đẩy container lên Docker Hub, việc quản lý phiên bản trở nên dễ dàng hơn.

2. Khuyết điểm

Cấu hình phức tạp: Cần kiến thức về Jenkins, Docker, và các công cụ liên quan để cấu hình ban đầu, gây khó khăn.

Phụ thuộc vào hạ tầng: Cần đảm bảo Jenkins server và các container chạy ổn định, tránh tình trạng downtime ảnh hưởng đến pipeline.

Bảo mật: Cần chú ý bảo mật các thông tin nhạy cảm như thông tin đăng nhập SMTP và thông tin Docker Hub, để tránh rủi ro rò rỉ dữ liệu.

3. Hướng phát triển

Kế hoạch bổ sung các tính năng như kiểm thử tự động chi tiết hơn hoặc tích hợp công cụ giám sát.

Nêu rõ kế hoạch cải thiện tính bảo mật của hệ thống Jenkins.

Tích hợp thêm bước triển khai tự động vào môi trường staging/production.

Nâng cấp pipeline để chạy các bài kiểm thử tích hợp và hiệu năng trước khi được chuyển qua các môi trường khác

TÀI LIỆU THAM KHẢO

1. https://docs.docker.com/
2. https://www.jenkins.io/
3. https://viblo.asia/p/ci-cd-va-devops-07LKXYXDZV4
4. https://topdev.vn/blog/trien-khai-ci-cd-voi-gitlab/
5. https://www.sonarsource.com/products/sonarqube/
6. https://www.youtube.com/watch?v=361bfIvXMBI&t=225s
7. https://www.youtube.com/watch?v=PKcGy9oPVXg&t=1119s

Link github: **https://github.com/SE400-P11-PMCL/jenkins-pipeline**