

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NGOẠI NGỮ – TIN HỌC THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN



Chuyên ngành: An ninh mạng
MÔN HỌC : Điện toán đám mây
Đề Tài: Triển khai Cloud, Jenkins và Docker
GIẢNG VIÊN HƯỚNG DẪN: Ths.Cao Tiến Thành

Thành viên nhóm: Nhóm 24

Huỳnh Gia Hòa – 22DH114543

Nguyễn Lê Văn Quyền – 22DH113040

TP.HCM, tháng 7/ 2025

🚦 Điểm phần trình bày – Điểm hệ 10

	CBCT1	CBCT2
Họ tên CBCT Chữ ký: Chữ ký:
Điểm Bằng chữ: Bằng chữ:
Nhận xét	Quyển báo cáo: (...) điểm.....	Quyển báo cáo: (...) điểm.....
Báo cáo: 2đ	Vấn đáp: (...) điểm.....	Vấn đáp: (...) điểm.....
Vấn đáp: 2đ	Chức năng: (...) điểm.....	Chức năng: (...) điểm.....
	Mở rộng: (...) điểm.....	Mở rộng: (...) điểm.....
Chức năng và demo: 5đ
Mở rộng và ứng dụng thực tiễn: 1đ

🚦 Điểm quá trình – Điểm hệ 10

Họ tên CBCT:

🚦 Điểm tổng kết:(Bằng chữ:.....

LỜI NÓI ĐẦU

Trong thời đại số hóa, việc triển khai hệ thống trên nền tảng điện toán đám mây (Cloud Computing) đã trở thành xu hướng tất yếu giúp doanh nghiệp và cá nhân tối ưu hóa chi phí, nâng cao hiệu suất và khả năng mở rộng hệ thống. Hai công cụ nổi bật hỗ trợ mạnh mẽ trong quá trình tự động hóa triển khai dịch vụ trên Cloud là Jenkins và Docker.

Jenkins là công cụ mã nguồn mở hỗ trợ CI/CD (Continuous Integration/Continuous Deployment), cho phép tự động hóa quy trình build, test và triển khai phần mềm. Trong khi đó, Docker là nền tảng container hóa phổ biến giúp đóng gói ứng dụng cùng các thành phần phụ thuộc thành một khối duy nhất có thể chạy ở mọi môi trường.

Việc kết hợp Jenkins và Docker trong triển khai hệ thống Cloud không chỉ giúp quá trình phát triển và vận hành phần mềm trở nên nhanh chóng, linh hoạt mà còn đảm bảo tính ổn định và khả năng tái sử dụng cao.

Với đề tài “Triển khai Cloud trên Jenkins và Docker”, nhóm chúng em mong muốn nghiên cứu, ứng dụng và thực hành triển khai dịch vụ trên nền tảng Cloud một cách tự động hóa, qua đó nắm vững kiến thức và kỹ năng quan trọng trong xu thế phát triển công nghệ hiện đại.

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành và sâu sắc đến thầy Cao Tiến Thành, người đã tận tình hướng dẫn, hỗ trợ và đồng hành cùng chúng em trong suốt quá trình thực hiện đồ án môn học Cloud Computing, với đề tài “Triển khai Cloud trên Jenkins và Docker”.

Trong suốt quá trình làm việc, sự nhiệt huyết, kiến thức chuyên môn sâu rộng và tinh thần tận tâm của thầy đã giúp chúng em hiểu rõ hơn về những khái niệm quan trọng trong lĩnh vực điện toán đám mây, cũng như cách ứng dụng các công cụ thực tế như Jenkins và Docker vào quy trình triển khai tự động.

Những lời khuyên, góp ý và định hướng chuyên môn từ thầy không chỉ giúp chúng em hoàn thiện tốt bài báo cáo, mà còn mang lại nhiều giá trị thực tiễn, tạo nền tảng vững chắc cho việc học tập và nghiên cứu sau này.

Một lần nữa, chúng em xin chân thành cảm ơn thầy vì đã luôn đồng hành và hỗ trợ chúng em trong hành trình học tập và phát triển.

NHẬN XÉT CỦA GIẢNG VIÊN

MỤC LỤC

MỤC LỤC HÌNH ẢNH.....	8
MỤC LỤC BẢNG.....	9
I. Giới Thiệu.....	10
1.1 Lý do chọn đề tài.....	10
1.1.1 Xu thế tất yếu của Cloud Computing:.....	10
1.1.2 Nhu cầu tự động hóa trong phát triển phần mềm:	10
1.1.3 Docker – công nghệ container hóa hiện đại:.....	10
1.1.4 Jenkins – công cụ tự động hóa mạnh mẽ:	11
1.1.5 Tính thực tiễn cao trong doanh nghiệp:	11
1.2 Mục tiêu báo cáo	11
1.2.1 Tìm hiểu lý thuyết và kiến trúc	11
1.2.2 Cài đặt và cấu hình môi trường.....	11
1.2.3 Xây dựng quy trình CI/CD thực tế.....	11
1.2.4 Đánh giá tính bảo mật và khả năng backup	12
II. Cơ sở lý thuyết	12
2.1 Tổng quan về DevOps và CI/CD	12
2.1.1 Văn hóa DevOps	12
2.1.2 Continuous Integration và Continuous Deployment / Delivery (CI/CD)	14
2.2 Containerisation – Đóng gói container	15
2.3 Tổng quan về Docker.....	16
2.3.1 Dockerfiles.....	18
2.3.2 Docker Compose.....	18
2.3.3 Lợi ích của Docker.....	19
2.4 Tổng quan về Jenkins.....	20
2.4.1 Cách hoạt động của Jenkins	21
2.4.2 Jenkins Pipeline	21
2.4.3 Các plugin của Jenkins.....	22
2.5 Kiến trúc tổng thể của hệ thống triển khai	23
2.5.1 Vai trò của từng thành phần.....	24
2.5.2 Quy trình tự động (CI/CD)	25
2.6 Tổng quan về nền tảng Cloud.	25
III. Phương pháp thực hiện	26

3.1 Mục tiêu	26
3.2 Kiến trúc tổng thể.....	26
3.3 Quy trình triển khai.....	27
3.4 Công cụ và nền tảng sử dụng.....	27
3.5 Cấu hình Jenkins Pipeline	27
IV. Triển Khai (Demo)	29
4.1 Triển khai Jenkins trên Docker	29
4.2 Tạo CI/CD triển khai webhook với github	34
4.3 Tạo ECW2 để tiến hành đẩy lên cloud	38
4.4 Cấu hình Let's encrypt và revers proxy để cấp https.....	46
4.5 Tạo backup trên EC2	49
V. Đánh giá và kết luận.....	55
Tài Liệu Tham Khảo	56

MỤC LỤC HÌNH ẢNH

Hình 1: Mục tiêu của DevOps.....	13
Hình 2: Continuous Integration and ContinuousDeployment / Delivery	14
Hình 3: Containerisation	15
Hình 4: Tổng quan Docker.....	17
Hình 5 Tổng quan Jenkins.....	20
Hình 6: Kiến trúc triển khai	23
Hình 7: Vai trò các thành phần.....	24
Hình 8: Giao diện Getting Started của Jenkins	29
Hình 9: Lấy mật khẩu lần đầu.....	30
Hình 10: Tạo Job	30
Hình 11: Cấu hình Job.....	31
Hình 12: Tạo Port.....	32
Hình 13: Tạo MongoDBURI	33
Hình 14: Tạo JWT.....	33
Hình 15: Tạo Session	34
Hình 16: Credentials Jenkins	34
Hình 17: Tải Ngrok	35
Hình 18: Lấy Auth Token Ngrok	35
Hình 19: Add authtoken Ngrok.....	36
Hình 20: Giao diện ngrok.....	36
Hình 21: Cấu hình Webhook Repo	37
Hình 22: Jenkins Trigger webhook	38
Hình 23: Liên kết thành công pipeline.....	38
Hình 24: Tạo instance	39
Hình 25: Mở port ec2	39
Hình 26: SSH EC2	40
Hình 27: Kiểm tra quyền key ssh.....	41
Hình 28: SSH Jenkins tới EC2.....	41
Hình 29: Credentials cho SSH	42
Hình 30: Cấu hình domain cho EC2	45

Hình 31: Trang chủ web với domain.....	46
Hình 32: Mở port 80 và 443 EC2.....	46
Hình 33: Cấu hình revers proxy.....	47
Hình 34: An toàn hơn khi không cần truy cập port của web trên docker.....	47
Hình 35: Yêu cầu cấp SSL.....	48
Hình 36: Thành công cấp SSL.....	48
Hình 37: Giao diện chứng chỉ trên web.....	49
Hình 38: Tự động gia hạn cho SSL.....	49
Hình 39: Script backup.....	52
Hình 40: Script Restore.....	54

MỤC LỤC BẢNG

Bảng 1 : Container and Virtual Machine	16
Bảng 2: Các plugin Jenkins.....	22
Bảng 3: Nền tảng Cloud phổ biến.....	26
Bảng 4: Công cụ nền tảng sử dụng	27
Bảng 5: File cần backup.....	50
Bảng 6: Đường dẫn cần cho backup	50
Bảng 7: Cách sử dụng script	54

I. Giới Thiệu

1.1 Lý do chọn đề tài

1.1.1 Xu thế tất yếu của Cloud Computing:

Trong bối cảnh công nghệ phát triển nhanh chóng, mô hình truyền thống với máy chủ vật lý và triển khai thủ công không còn đáp ứng được nhu cầu linh hoạt, mở rộng và tiết kiệm chi phí. Cloud Computing (điện toán đám mây) mang lại khả năng mở rộng tài nguyên theo nhu cầu, dễ dàng triển khai và quản lý, cũng như tối ưu chi phí vận hành. Việc chuyển dịch hạ tầng và dịch vụ lên cloud đã trở thành xu hướng chủ đạo tại các doanh nghiệp, tổ chức lớn nhỏ trên toàn cầu.

1.1.2 Nhu cầu tự động hóa trong phát triển phần mềm:

Khi quy mô dự án và nhóm phát triển tăng lên, việc đảm bảo tính ổn định và liên tục trong quá trình phát triển – kiểm thử – triển khai trở nên quan trọng hơn bao giờ hết. CI/CD (Continuous Integration / Continuous Deployment) giúp loại bỏ các thao tác thủ công, tăng tốc độ release phần mềm, giảm rủi ro lỗi, đồng thời tăng tính nhất quán và độ tin cậy.

1.1.3 Docker – công nghệ container hóa hiện đại:

Docker là nền tảng container mạnh mẽ cho phép đóng gói ứng dụng kèm theo toàn bộ môi trường cần thiết. Nhờ đó, ứng dụng có thể chạy nhất quán ở mọi môi trường từ máy tính cá nhân, server vật lý đến cloud. Tính nhẹ, khả năng chia sẻ và quản lý tài nguyên hiệu quả đã giúp Docker trở thành công cụ không thể thiếu trong DevOps hiện đại.

1.1.4 Jenkins – công cụ tự động hóa mạnh mẽ:

Jenkins là một trong những công cụ phổ biến nhất dùng để xây dựng pipeline CI/CD. Với khả năng tích hợp linh hoạt qua các plugin, Jenkins cho phép thực hiện hàng loạt tác vụ tự động như build code, test, tạo image Docker, đẩy lên registry, triển khai lên máy chủ, ...

1.1.5 Tính thực tiễn cao trong doanh nghiệp:

Việc kết hợp Jenkins, Docker và Cloud (EC2 của AWS) tạo thành một hệ sinh thái hoàn chỉnh cho phát triển và triển khai phần mềm hiện đại. Đây chính là mô hình được sử dụng thực tế tại nhiều công ty phần mềm và tổ chức DevOps chuyên nghiệp. Việc nghiên cứu, cài đặt và thực hành mô hình này sẽ giúp sinh viên tiếp cận gần hơn với môi trường làm việc thực tế, rèn luyện kỹ năng chuyên sâu về DevOps và triển khai hệ thống.

1.2 Mục tiêu báo cáo

1.2.1 Tìm hiểu lý thuyết và kiến trúc

- Nắm vững các khái niệm, nguyên lý hoạt động của **Jenkins**, **Docker**, và **EC2** trong hệ sinh thái CI/CD.
- Hiểu rõ **pipeline CI/CD** hoạt động ra sao, các bước thực thi từ khi developer commit code đến khi phần mềm được triển khai.

Phân tích cách Jenkins tương tác với Docker và cách Jenkins kiểm soát quá trình build – deploy ứng dụng.

1.2.2 Cài đặt và cấu hình môi trường

- Cài đặt Jenkins dưới dạng container Docker trên máy cục bộ.
- Thiết lập pipeline thực tế bao gồm tích hợp GitHub, Docker Hub và máy chủ EC2.
- Cấu hình Jenkins để sử dụng SSH truy cập và triển khai ứng dụng từ xa lên EC2.

1.2.3 Xây dựng quy trình CI/CD thực tế

Thiết kế một file Jenkinsfile mô tả toàn bộ quy trình:

- Lấy mã nguồn từ GitHub

- Tạo file .env từ các biến bí mật
- Build Docker image ứng dụng
- Push lên Docker Hub (hoặc backup image)
- SSH vào EC2, pull image và khởi chạy container

Tự động hóa các bước kiểm thử đơn vị, kiểm tra công, kiểm tra dịch vụ hoạt động sau khi triển khai.

1.2.4 Đánh giá tính bảo mật và khả năng backup

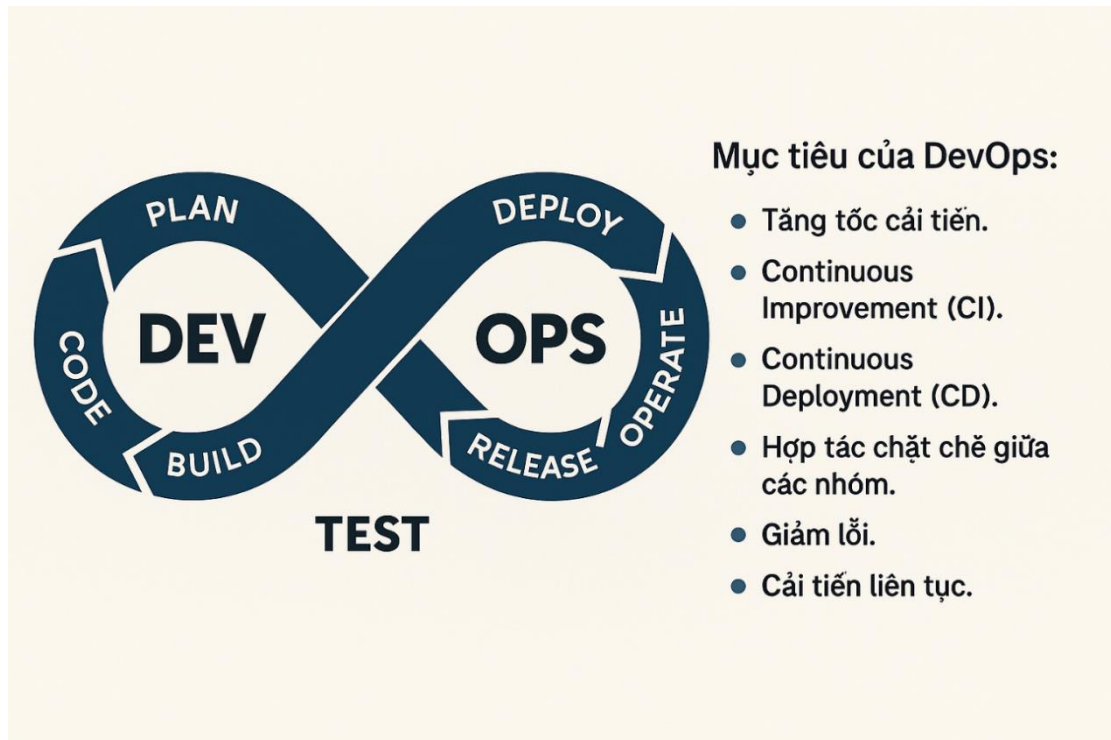
- Sử dụng Docker .env và Jenkins Credentials để bảo mật biến môi trường.
- Thiết lập cơ chế backup/restore image và source code định kỳ nhằm đảm bảo dữ liệu không bị mất.
- Áp dụng HTTPS và các cấu hình firewall cơ bản trong EC2 để tăng độ an toàn.

II. Cơ sở lý thuyết

2.1 Tổng quan về DevOps và CI/CD

2.1.1 Văn hóa DevOps

- Quy trình kỹ thuật truyền thống thường gặp nhiều vấn đề khiến quá trình làm việc và cải tiến sản phẩm trì trệ, tốn thời gian hay lo sợ vì mỗi lần triển khai dễ lỗi, rollback chậm. Các nhóm kỹ thuật làm việc rời rạc không liên kết khiến quá trình test và production khác nhau thậm chí tốn thời gian do triển khai thủ công. Vì thế DevOps ra đời.



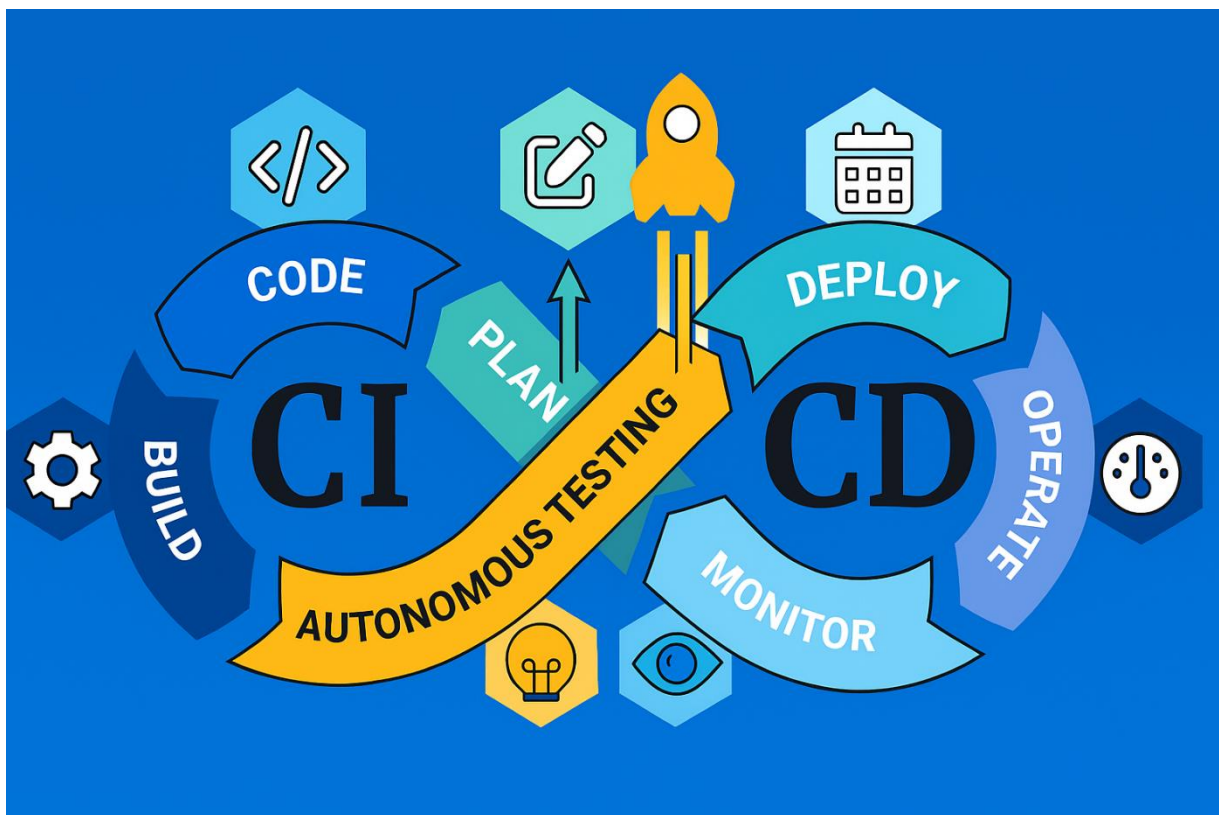
Hình 1: Mục tiêu của DevOps

- DevOps là triết lý, văn hóa kỹ thuật tập trung vào sự hợp tác và học hỏi liên tục vào mục tiêu tăng tốc vòng đời phát triển phần mềm đưa vào sản xuất. DevOps ưu tiên con người hơn quy trình, quy trình hơn công cụ, xây dựng môi trường nơi mà Development, Operation, Security và infrastructure cùng nhau tin tưởng và cải tiến mang lại sản phẩm tốt cho người dùng. Mục tiêu của DevOps:

- + Tăng tốc cải tiến.
- + Continuous Improvement (CI).
- + Continuous Deployment (CD).
- + Hợp tác chặt chẽ giữa các nhóm.
- + Giảm lỗi.
- + Cải tiến liên tục.

2.1.2 Continuous Integration và Continuous Deployment / Delivery (CI/CD)

- CI/CD là nền tảng kỹ thuật cốt lõi của DevOps, giúp việc phát hành phần mềm nhanh chóng, ổn định và an toàn. Đây không chỉ là công nghệ mà còn là cách tiếp cận quy trình làm việc của tổ chức hiệu quả hơn.



Hình 2: Continuous Integration and Continuous Deployment / Delivery

* **Continuous Integration (CI) – Tích hợp liên tục:** Quá trình tích hợp mã nguồn từ các lập trình viên vào branch main. Mỗi lần push code sẽ tự động kiểm tra, build và test để phát hiện lỗi càng sớm càng tốt. Giúp hệ thống đảm bảo mọi thay đổi ổn định, tăng độ tin cậy cho hệ thống.

* **Continuous Deployment / Delivery (CD) – Triển khai liên tục / Phân phối liên tục:** Giai đoạn sau của CI, nơi phần mềm tự động triển khai đến môi trường staging hoặc production nếu vượt qua kiểm thử.

+ Continuous Delivery cần có sự cho phép của người.

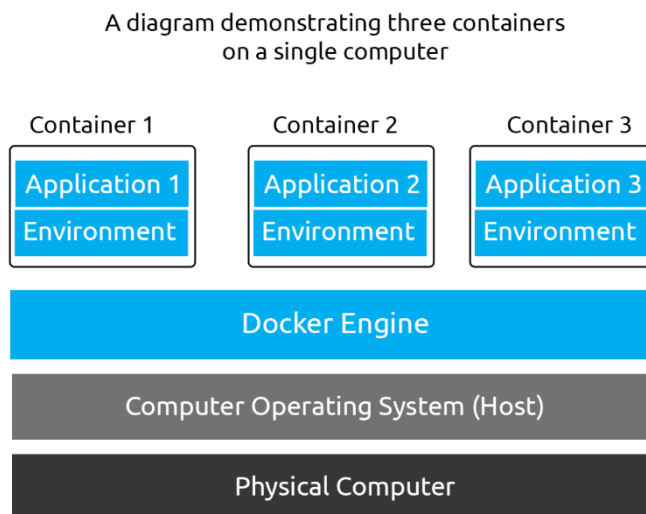
+ Continuous Deployment diễn ra hoàn toàn tự động không cần can thiệp thủ công.

- **Lợi ích từ CI / CD :**

- + Giảm thiểu lỗi tác động thủ công.
- + Tăng tốc độ phát hành các tính năng.
- + Dễ dàng rollback khi lỗi.
- + Đảm bảo môi trường test và production đồng nhất.

2.2 Containerisation – Đóng gói container

- Containerisation là quá trình đóng gói toàn bộ ứng dụng cùng các thư viện (libraries) hay các gói (packages) thành một gọi là container. Việc đóng gói thành một container giúp cho phần mềm dễ dàng di chuyển và khởi động ở môi trường máy mới.
- Các ứng dụng hiện đại thường phụ thuộc nhiều vào thư viện và framework gây ra các vấn đề như:
 - + Phức tạp trong việc cài đặt môi trường khác (Máy, hệ điều hành khác).
 - + Khó chuẩn đoán lỗi vì có thể lỗi ở môi trường.
 - + Xung đột thư viện.



Hình 3: Containerisation

- Các Containerisation Platform giúp tạo ra các container mỗi ứng dụng và cô lập chúng. Cơ chế hoạt động dựa trên namespace của kernel, giúp các tài nguyên được phân bổ cho từng process, khiến các container không thể nhìn thấy và tương tác với nhau.

Ví dụ: Nếu một container bị RCE (Remote Code Execution), thì các container khác không bị ảnh hưởng nếu không dùng chung namespace.

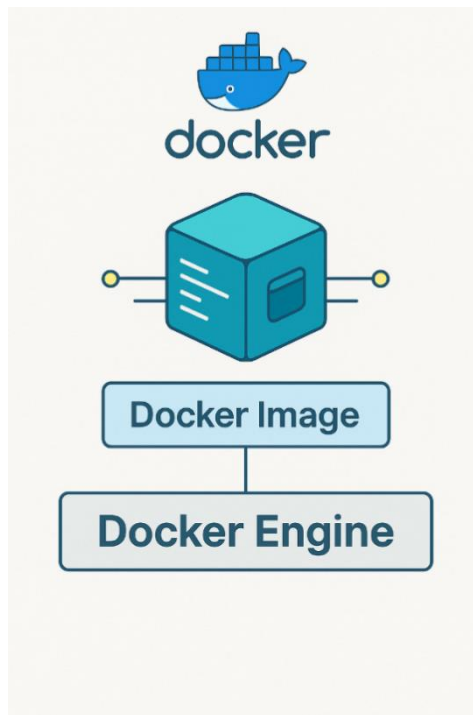
So sánh Container và Virtual Machine:

Tiêu chí	Container	Virtual Machine (VM)
Khái niệm	Đóng gói ứng dụng cùng môi trường, chia sẻ kernel hệ điều hành.	Mô phỏng toàn bộ hệ điều hành, chạy trên hypervisor.
Cấu trúc	App → Bin/Lib → Container Engine (Docker) → OS	App → Bin/Lib → Guest OS → Hypervisor → Host OS
Kích thước	Nhẹ (tính bằng MB).	Nặng (thường tính bằng GB).
Khởi động	Rất nhanh (vài giây).	Chậm (vài phút).
Sử dụng tài nguyên	Tối ưu, chia sẻ tài nguyên.	Tốn nhiều RAM, CPU vì chạy OS riêng.
Tính di động (portable)	Cao – chạy được mọi nơi có Docker.	Thấp hơn – phụ thuộc hypervisor và OS.
Cách cô lập	Dùng namespace và cgroup (cô lập nhẹ).	Cô lập chặt chẽ, có kernel riêng biệt.
Dễ deploy CI/CD	Rất phù hợp.	Phức tạp hơn.
Khả năng scale (mở rộng)	Dễ dàng, nhanh chóng.	Khó scale nhanh.
Quản lý & Bảo trì	Dễ – xóa & tạo lại nhanh.	Khó – OS cần bảo trì riêng.

Bảng 1 : Container and Virtual Machine

2.3 Tổng quan về Docker

- Docker là Containerisation Platform mã nguồn mở, mạnh mẽ và dễ sử dụng. Cho phép các ứng dụng triển khai, quản lý và chia sẻ dễ dàng. Hoạt động trên Linux, Windows



Hình 4: Tổng quan Docker

và MacOS. Các ứng dụng được đóng gói thành Docker image và có thể chia sẻ image đó cho người khác. Người nhận chỉ cần tải (pull) image về và chạy trên docker.

- Docker sử dụng công nghệ Containerisation để cô lập các ứng dụng vào container thông qua Docker Engine. Docker Engine là API chạy ở Máy chủ hệ điều hành (host operating system) giúp giao tiếp giữa hệ điều hành và container, từ đó cấp quyền truy cập và các hardware resource như CPU, RAM, mạng và Hard Disk.
- Nhờ Docker Engine, cho phép thực hiện các việc như :
 - + Kết nối container với nhau.
 - + Xuất nhập ứng dụng bằng image.
 - + Chuyển file giữ hệ điều hành và container.
- Docker sử dụng YAML để định nghĩa container được xây dựng và cách chạy. Chỉ cần chia sẻ file YAML, hệ thống có thể build và chạy nếu có Docker Engine.
- Docker Engine còn cho phép orchestration (Điều phối container), xây dựng và chạy nhiều container, cho phép chúng giao tiếp nội bộ. Ví dụ: một web server trên container này có thể kết nối với container khác chạy database.

- Có hai đối tượng chính :

- + Docker image : Chứa toàn bộ source code, môi trường và hướng dẫn chạy ứng dụng.
- + Docker Container: Quá trình thực thi từ image, hoạt động độc lập và có thể bị xóa sau khi chạy.

2.3.1 Dockerfiles

- Dockerfiles là một tệp văn bản có định dạng đặc biệt, chứa các lệnh hướng dẫn (manual instructions) để xây dựng Docker image. Nói cách khác, Dockerfiles chính là công thức mà Docker sử dụng để tạo ra môi trường container.

Công dụng:

Tự động hóa quá trình cài đặt phần mềm, copy file, cấu hình, thiết lập biến môi trường,...

Mỗi lệnh trong Dockerfile sẽ tạo ra một layer trong image.

Ví dụ:

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "index.js"]
```

2.3.2 Docker Compose

Docker Compose là một công cụ giúp triển khai nhiều container cùng lúc bằng cách định nghĩa toàn bộ hệ thống trong một file YAML duy nhất (docker-compose.yml).

Nó đặc biệt hữu ích khi cần vận hành các hệ thống microservices có nhiều thành phần như web, cơ sở dữ liệu, Redis, ...

Công dụng:

Đơn giản hóa quản lý nhiều container: chỉ cần 1 lệnh để build, start, stop, remove.

Tự động hóa việc liên kết các container với nhau thông qua mạng nội bộ riêng (network bridge).

Dễ chia sẻ: chỉ cần gửi file docker-compose.yml là người khác có thể chạy toàn bộ hệ thống.

Cấu trúc cơ bản của một file docker-compose.yml:

```
version: '3'
services:
  web:
    build: .
    ports:
      - "8080:80"
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: example
```

Giải thích:

- version: xác định phiên bản Compose đang dùng.
- services: định nghĩa các dịch vụ (mỗi dịch vụ là một container).
- build: chỉ định Dockerfile trong thư mục hiện tại.
- image: tên image được pull từ Docker Hub.
- ports: ánh xạ cổng container → máy chủ.
- environment: thiết lập biến môi trường cho container.

2.3.3 Lợi ích của Docker

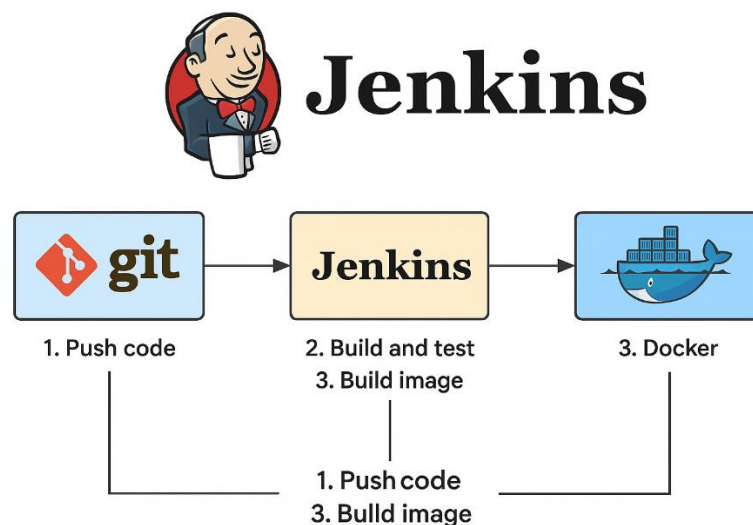
- Tính di động cao: Docker image chạy được trên mọi hệ điều hành có cài Docker Engine (Windows, Linux, macOS).
- Triển khai nhanh chóng: từ một image, có thể tạo ra hàng loạt container chạy ứng dụng chỉ với một dòng lệnh.
- Cô lập môi trường: mỗi container là một thế giới riêng biệt, không ảnh hưởng đến hệ thống gốc hoặc container khác.
- Tối ưu tài nguyên: dùng chung kernel với hệ điều hành, nhẹ và nhanh hơn máy ảo (VM).
- Hỗ trợ CI/CD: dễ tích hợp vào pipeline DevOps để tự động hóa build, test, deploy.
- Tái sử dụng và chia sẻ dễ dàng: thông qua Docker Hub hoặc file cấu hình (Dockerfile / docker-compose.yml).

- Quản lý linh hoạt: dễ dàng nâng cấp, rollback, và mở rộng hệ thống bằng cách sửa Dockerfile hoặc YAML.

2.4 Tổng quan về Jenkins

Jenkins là một công cụ mã nguồn mở giúp tự động hóa các quy trình trong phát triển phần mềm, đặc biệt là quy trình CI/CD (Continuous Integration/Continuous Deployment). Jenkins hỗ trợ việc tích hợp liên tục và triển khai nhanh chóng các ứng dụng bằng cách lấy mã nguồn từ hệ thống quản lý phiên bản như Git, sau đó tự động build, kiểm thử, và triển khai ứng dụng.

Quy trình hoạt động của Jenkins bao gồm các bước chính: khi nhà phát triển push mã nguồn lên Git, Jenkins sẽ kích hoạt một Job hoặc Pipeline được định nghĩa sẵn để tự động thực hiện build và kiểm thử. Nếu không phát sinh lỗi, ứng dụng có thể được triển khai lên môi trường staging hoặc production.



Hình 5 Tổng quan Jenkins

Jenkins hỗ trợ khái niệm Pipeline – một cách mô tả quy trình CI/CD dưới dạng mã, thông qua tệp Jenkinsfile. Jenkinsfile cho phép định nghĩa rõ ràng các stage như: build, test, deploy. Điều này giúp quy trình trở nên minh bạch, dễ kiểm soát và có thể version-control cùng với mã nguồn.

Ngoài ra, Jenkins còn có hệ thống plugin phong phú, hỗ trợ tích hợp với nhiều công cụ khác như: Git, Docker, SSH Agent, Slack, Kubernetes, v.v. Những plugin quan trọng thường dùng bao gồm:

- Git Plugin: kết nối Jenkins với GitHub hoặc GitLab.
- Docker Plugin: hỗ trợ build và chạy container.
- Pipeline Plugin: định nghĩa quy trình CI/CD.
- SSH Agent Plugin: dùng để deploy ứng dụng lên server thông qua SSH.

2.4.1 Cách hoạt động của Jenkins

Quy trình hoạt động cơ bản:

- Lấy mã nguồn từ Git hoặc GitHub.
- Build (biên dịch) mã nguồn.
- Test (kiểm thử): chạy unit test, integration test,...
- Deploy (triển khai): đẩy lên server hoặc cloud (EC2, Docker,...)

→ Tự động hóa toàn bộ quá trình phát triển và triển khai phần mềm.

2.4.2 Jenkins Pipeline

Jenkins Pipeline là cách mô tả quy trình CI/CD bằng mã (as code) thông qua file Jenkinsfile.

Cho phép tái sử dụng, kiểm soát phiên bản và mở rộng dễ dàng quy trình tự động hóa.

Ví dụ Jenkinsfile:

```
pipeline {
  agent any
  stages {
    stage('Clone') {
      steps {
        git 'https://github.com/user/project.git'
      }
    }
    stage('Build') {
      steps {
        sh 'npm install'
      }
    }
  }
}
```

```
stage('Test') {
    steps {
        sh 'npm test'
    }
}
stage('Deploy') {
    steps {
        sh './deploy.sh'
    }
}
}
```

Ưu điểm của Jenkinsfile:

- Quản lý được trong Git (pipeline as code).
- Tự động hóa và dễ debug.
- Hỗ trợ logic điều kiện, phân nhánh, song song.

2.4.3 Các plugin của Jenkins

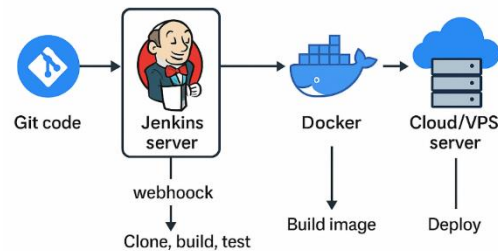
Plugin	Mục đích sử dụng
Git Plugin	Kết nối và clone code từ Git / GitHub
Docker Plugin	Build / run container, đẩy lên Docker Hub
SSH Agent	Kết nối SSH đến server EC2, VPS,... để deploy
Pipeline	Cho phép dùng Jenkinsfile định nghĩa pipeline
Credentials	Lưu và bảo mật thông tin đăng nhập SSH, Git
Email Ext	Gửi email khi build thất bại hoặc thành công
Blue Ocean	Giao diện trực quan cho quản lý pipeline

Bảng 2: Các plugin Jenkins

2.5 Kiến trúc tổng thể của hệ thống triển khai

Một hệ thống CI/CD điển hình được triển khai với các thành phần chính như sau:

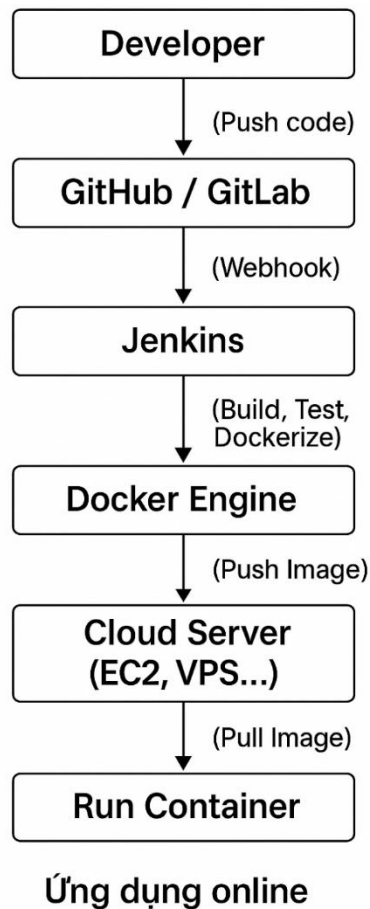
Git repository → Jenkins → Docker → Cloud server.



Hình 6: Kiến trúc triển khai

- **Git repository (ví dụ: GitHub, GitLab):** là nơi lưu trữ mã nguồn. Mỗi khi nhà phát triển thực hiện thao tác push, một webhook sẽ kích hoạt Jenkins.
- **Jenkins server:** nhận tín hiệu từ Git, lấy mã nguồn, build project, kiểm thử và đóng gói ứng dụng (thành Docker Image).
- **Docker:** được Jenkins sử dụng để tạo image và chạy container. Docker giúp đảm bảo tính đồng nhất giữa các môi trường.
- **Cloud/VPS server:** nơi chứa môi trường production. Jenkins có thể deploy image này lên cloud thông qua SSH hoặc thông qua dịch vụ container orchestration như Kubernetes.

2.5.1 Vai trò của từng thành phần



Hình 7: Vai trò các thành phần

Developer:

Viết code, commit và push lên GitHub.

GitHub / GitLab:

Lưu trữ mã nguồn và gửi tín hiệu (webhook) tới Jenkins khi có cập nhật.

Jenkins:

Nhận webhook → chạy pipeline tự động hóa quá trình build, test và tạo Docker image.

Docker Engine:

Xây dựng image từ mã nguồn → đóng gói ứng dụng vào container.

Docker Hub / Registry:

Lưu trữ Docker image để sử dụng cho triển khai.

Cloud Server (EC2, VPS, v.v.):

Jenkins SSH vào server để pull image và chạy container mới.

Ứng dụng online:

Sau khi container chạy thành công, ứng dụng sẽ có thể truy cập qua trình duyệt.

2.5.2 Quy trình tự động (CI/CD)

- Developer push code lên GitHub/GitLab.
- Webhook tự động gửi tín hiệu đến Jenkins khi có thay đổi trong repository.
- Jenkins nhận webhook, thực thi file Jenkinsfile với các bước:
- Clone mã nguồn mới.
- Build ứng dụng (nếu cần).
- Chạy test tự động.
- Tạo Docker image.
- Push image lên Docker Hub.
- Jenkins dùng SSH Agent để kết nối đến Cloud Server (EC2, VPS,...).
- Cloud Server pull Docker image từ Docker Hub.
- Dừng container cũ (nếu có) và khởi chạy container mới từ image vừa cập nhật.
- Ứng dụng được triển khai online, có thể truy cập qua domain hoặc IP.

2.6 Tổng quan về nền tảng Cloud.

Cloud là một mô hình cung cấp tài nguyên điện toán như máy chủ (server), lưu trữ (storage), cơ sở dữ liệu (database) và các dịch vụ phần mềm thông qua Internet. Thay vì phải đầu tư máy chủ vật lý, người dùng có thể thuê tài nguyên từ các nhà cung cấp cloud và chỉ trả tiền theo mức sử dụng.

Tài nguyên bao gồm:

- Máy chủ ảo (virtual server – VPS).
- Lưu trữ (storage).
- Cơ sở dữ liệu, mạng, dịch vụ AI,...

Người dùng không cần đầu tư phần cứng, chỉ cần thuê theo nhu cầu sử dụng.

Các nền tảng cloud phổ biến hiện nay bao gồm:

Nền tảng	Mô tả ngắn gọn
AWS	Amazon Web Services – nền tảng cloud phổ biến nhất

Nền tảng	Mô tả ngắn gọn
GCP	Google Cloud Platform – mạnh về AI/Big Data
Azure	Cloud của Microsoft – tích hợp với Windows tốt
Linode	Dịch vụ VPS giá rẻ, dễ sử dụng
DigitalOcean	Cloud thân thiện với developer nhỏ/lẻ
Vultr	VPS phổ biến, dễ đăng ký, giá hợp lý

Bảng 3: Nền tảng Cloud phổ biến

Tại sao nên triển khai ứng dụng trên Cloud?

- Có thể truy cập từ mọi nơi thông qua IP Public.
- Dễ mở rộng và cập nhật ứng dụng.
- Không tốn chi phí bảo trì phần cứng.
- Tính linh hoạt và độ sẵn sàng cao.

Nên chọn nền tảng nào?

- Nếu là sinh viên hoặc người mới bắt đầu, có thể chọn các VPS giá rẻ như Linode, Vultr, hoặc tận dụng các chương trình Free Tier của AWS và GCP.
- Với các dự án lớn hơn, nên chọn nền tảng mạnh như AWS hoặc GCP để tận dụng đầy đủ các dịch vụ nâng cao.

III. Phương pháp thực hiện

3.1 Mục tiêu

- Triển khai một hệ thống CI/CD hoàn chỉnh sử dụng Jenkins, Docker, và Cloud server.
- Tự động hóa toàn bộ quy trình từ khi lập trình viên push code đến khi ứng dụng hoạt động online.

3.2 Kiến trúc tổng thể

Hệ thống bao gồm các thành phần chính:

- GitHub: lưu trữ mã nguồn.
- Jenkins: công cụ CI/CD tự động build, test, tạo Docker image và deploy.
- Docker Engine: đóng gói ứng dụng thành container.
- Docker Hub: lưu trữ và chia sẻ Docker image.
- Cloud Server (EC2/VPS): nơi chạy ứng dụng container hóa.

3.3 Quy trình triển khai

- Viết mã nguồn và đẩy lên GitHub.
- GitHub gửi webhook tới Jenkins khi có cập nhật.
- Jenkins thực hiện các bước:
 - Clone source code.
 - Cài đặt dependencies, build và test ứng dụng.
 - Tạo Docker image và đẩy lên Docker Hub.
 - Jenkins dùng SSH truy cập Cloud Server.
 - Cloud Server pull image và chạy container mới.

3.4 Công cụ và nền tảng sử dụng

Tên công cụ	Mục đích
Windows 11	Máy host
Jenkins	Tự động hóa build, test, deploy
Docker	Đóng gói và chạy ứng dụng bằng container
Docker Hub	Lưu trữ image
GitHub	Lưu trữ mã nguồn, webhook
AWS EC2 / VPS	Chạy ứng dụng online
Ubuntu 20.04	Hệ điều hành máy chủ

Bảng 4: Công cụ nền tảng sử dụng

3.5 Cấu hình Jenkins Pipeline

Sử dụng file Jenkinsfile để định nghĩa các stage trong pipeline.

Mẫu:

groovy

Copy code

```
pipeline {
  agent any
  stages {
    stage('Clone') {
      steps { git 'https://github.com/user/project.git' }
    }
    stage('Build') {
      steps { sh 'npm install' }
    }
    stage('Test') {
      steps { sh 'npm test' }
    }
    stage('Docker Build & Push') {
      steps {
        sh 'docker build -t user/app:latest .'
        sh 'docker push user/app:latest'
      }
    }
    stage('Deploy') {
      steps {
        sshagent(['ssh-key-id']) {
          sh 'ssh ubuntu@your-server "docker pull user/app:latest && docker run -d -p 80:80 user/app:latest"'
        }
      }
    }
  }
}
```

IV. Triển Khai (Demo)

4.1 Triển khai Jenkins trên Docker

Lệnh khởi chạy:

```
docker run -d --name jenkins -u root --privileged -p 8080:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock jenkins/jenkins:lts
```

Truy cập Jenkins bash

```
docker exec -u root -it jenkins bash
```

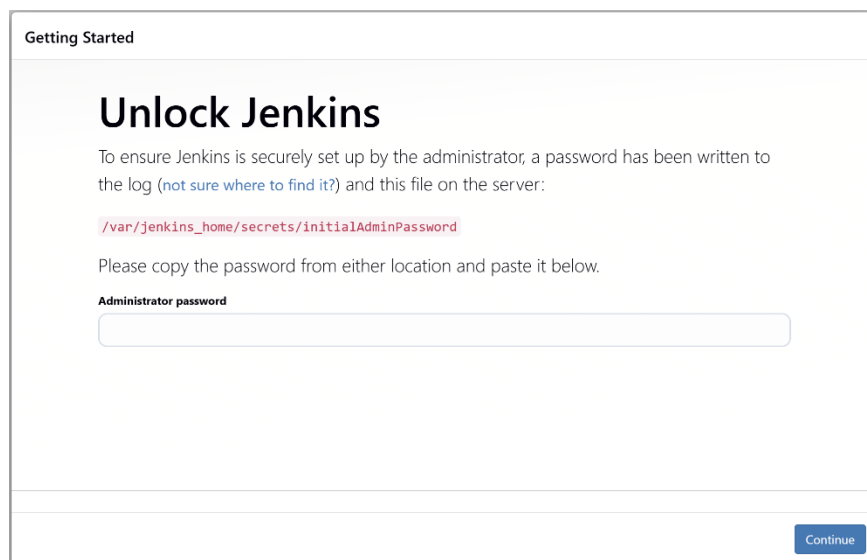
Cài Docker CLI trong container:

```
apt-get update  
apt-get install -y docker.io
```

Kiểm tra:

```
docker --version
```

Truy cập Jenkin : localhost:8080



Hình 8: Giao diện Getting Started của Jenkins

Lấy mật khẩu lần đầu:

```
docker exec jenkins cat /var/jenkins_homes cat  
/var/jenkins_home/secrets/initialAdminPassword
```

```
C:\Users\2ien>docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword  
19b57e680a224150948f516875efb595
```

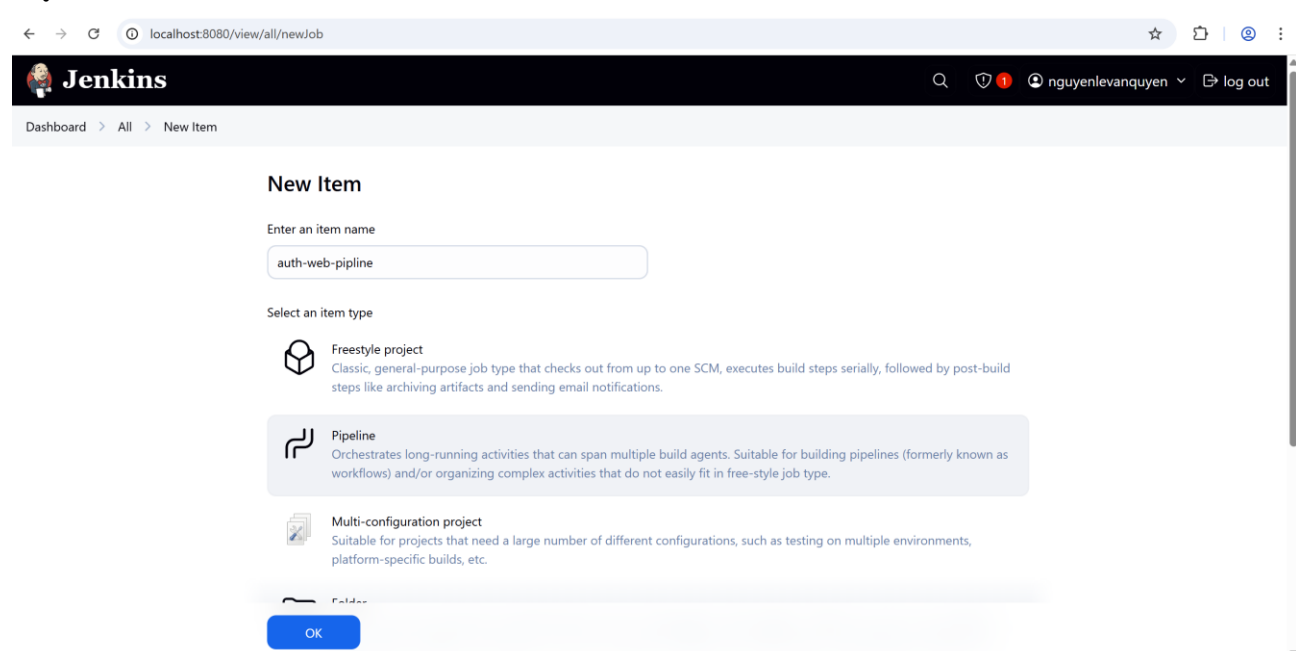
Hình 9: Lấy mật khẩu lần đầu

Chọn Install suggested plugins

→ Jenkins sẽ tự động cài các plugin cơ bản như:

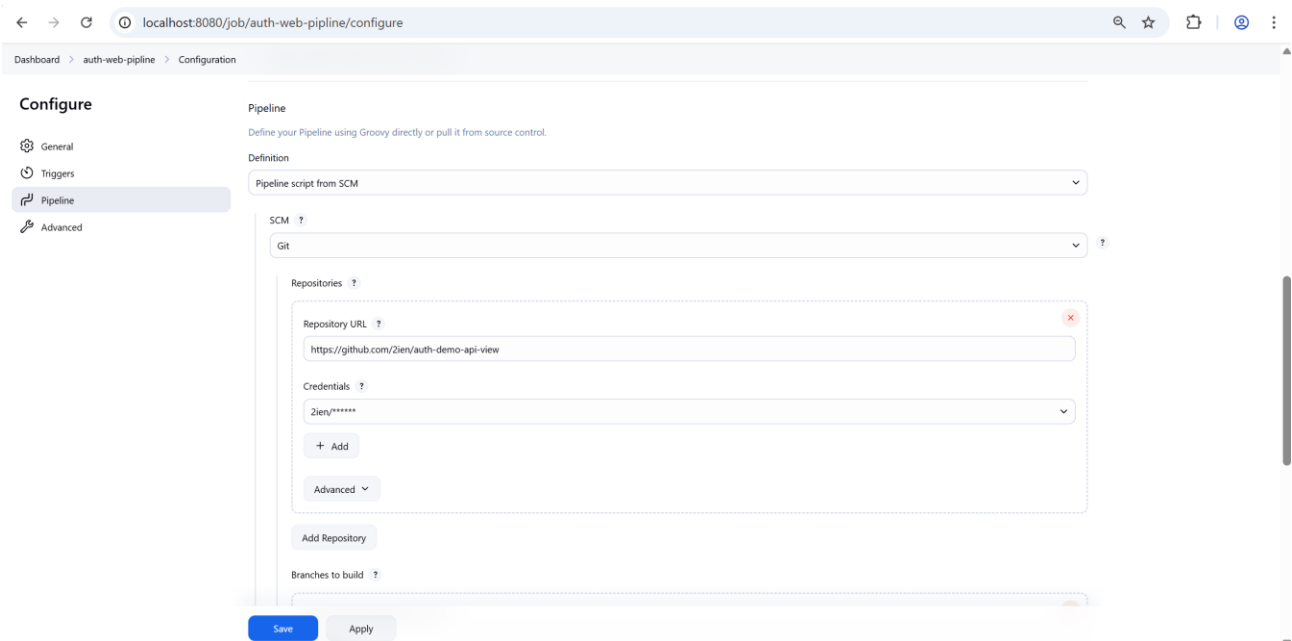
- Git
- Pipeline
- Script Security, ...

Tạo Job



Hình 10: Tạo Job

Cấu hình Job



Hình 11: Cấu hình Job

SCM: Git

Repository URL: Đường dẫn Repo git

Credentials:

BƯỚC 1: Tạo GitHub Token

- Truy cập GitHub: <https://github.com/settings/tokens>
- Nhấn: Generate new token (classic)
- Chọn các scope:
repo (để Jenkins có quyền clone repo private)
(Không cần chọn quá nhiều)
- Nhấn Generate token
- Copy token ngay (chỉ hiển thị 1 lần)

BƯỚC 2: Thêm token vào Jenkins Credentials

Quay lại Jenkins → từ giao diện chính:

Vào Manage Jenkins

→ Credentials

→ (global) → Add Credentials

Chọn:

Kind: Username with password

Username: GitHub username của bạn (VD: your-user)

Password: Dán GitHub token đã tạo ở bước 1

ID: github-token (tùy ý, nhưng bạn nên nhớ tên này)

File .env mẫu:

```
PORT=
MONGO_URI=
JWT_SECRET=
SESSION_SECRET=
```

Tạo port

New credentials

Kind

Secret text

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Secret

....

ID ?

port

Description ?

PORT

Create

Hình 12: Tạo Port

MongoURI:

New credentials

Kind
Secret text

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

Secret
.....

ID ?
mongo-uri

Description ?
MongoDB connection string

Create

Hình 13: Tạo MongoURI

JWT:

Kind
Secret text

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

Secret
.....

ID ?
jwt-secret

Description ?
JWT secret

Create

Hình 14: Tạo JWT

Session:

Kind
Secret text

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

Secret

ID ?
session-secret

Description ?
Session secret

Create






Hình 15: Tạo Session

Kết quả:

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

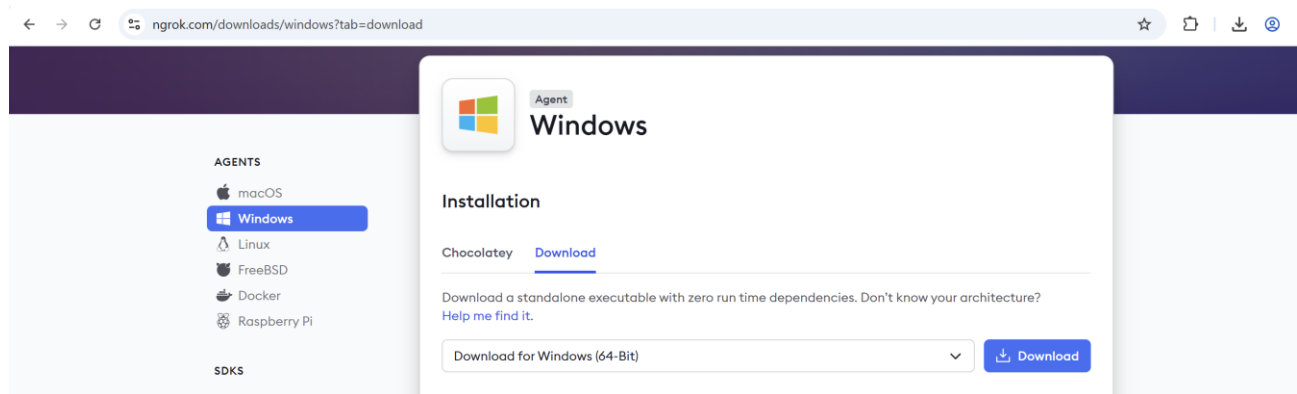
ID	Name	Kind	Description
 jenkins	zlen/*****	Username with password	
 port	PORT	Secret text	PORT
 mongo-uri	MongoDB connection string	Secret text	MongoDB connection string
 jwt-secret	JWT secret	Secret text	JWT secret
 session-secret	Session secret	Secret text	Session secret

Icon: S M L

Hình 16: Credentials Jenkins

4.2 Tạo CI/CD triển khai webhook với github

Bước 1: Cài ngrok để Forwarding tới URL Jenkins



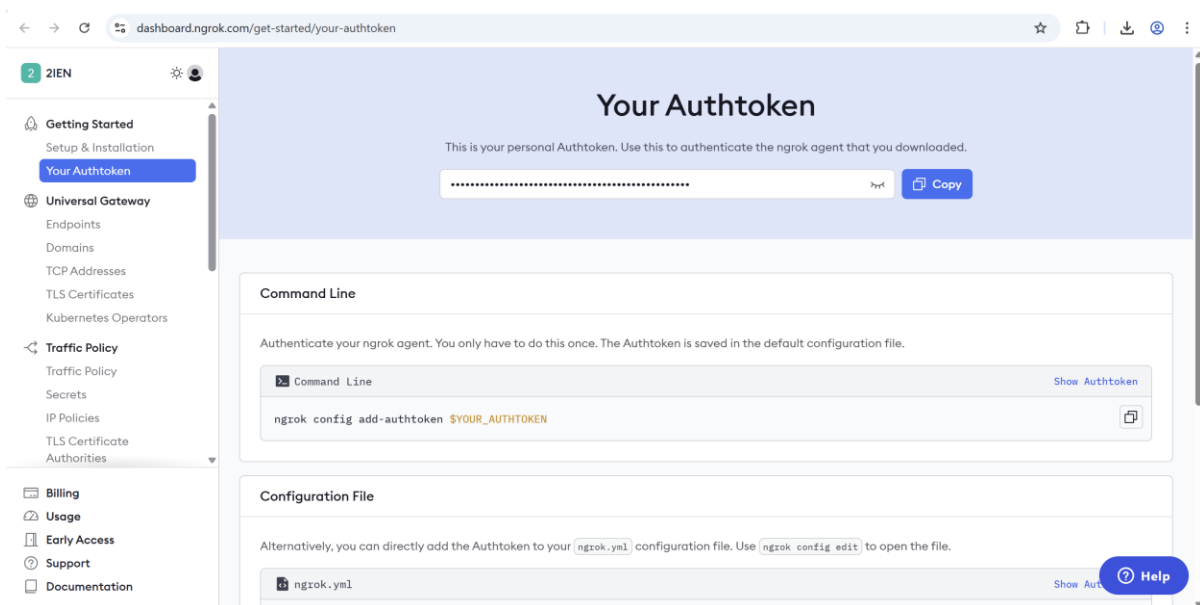
Hình 17: Tải Ngrok

Bước 2: Đăng ký tài khoản ngrok

Truy cập: <https://ngrok.com/signup>

Đăng ký và login

Lấy Auth Token tại: <https://dashboard.ngrok.com/get-started/your-authtoken>



Hình 18: Lấy Auth Token Ngrok

Bước 3: Thêm Auth Token

```
ngrok config add-authtoken <YOUR_TOKEN>
```

```
C:\Users\2ien\Downloads\ngrok-v3-stable-windows-amd64>ngrok config add-authtoken 2zu9B9bkIs65cR4HNOVyV8T0guR_84Jm7XbnW9e2Eq
Authtoken saved to configuration file: C:\Users\2ien\AppData\Local\ngrok\ngrok.yml
```

Hình 19: Add authtoken Ngrok

Bước 4: Mở port Jenkins bằng ngrok

Jenkins chạy ở port 8080:

```
ngrok http 8080
```

```
ngrok
🔖 Call internal services from your gateway: https://ngrok.com/r/http-request

Session Status      online
Account             2IEN (Plan: Free)
Version             3.23.3
Region              Asia Pacific (ap)
Web Interface        http://127.0.0.1:4040
Forwarding           https://941be4642387.ngrok-free.app -> http://localhost:8080

Connections          ttl    opn    rt1    rt5    p50    p90
0                   0      0      0.00   0.00   0.00   0.00
```

Hình 20: Giao diện ngrok

URL Forwarding: <https://941be4642387.ngrok-free.app>

Bước 5: Thêm webhook trong GitHub

Vào GitHub repo → Settings > Webhooks

Click Add Webhook

Điền:

- Payload URL: <https://abc123.ngrok.io/github-webhook/>
- Content type: application/json
- Secret: (tùy chọn)
- Events: chọn "Just the push event"
- Add

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, etc). More information can be found in [our developer documentation](#).

Payload URL *

Content type *

Secret

SSL verification

 By default, we verify SSL certificates when delivering payloads.

☒ Enable SSL verification ☐ Disable (not recommended)

Which events would you like to trigger this webhook?

- ☒ Just the push event.
- ☐ Send me **everything**.
- ☐ Let me select individual events.

Hình 21: Cấu hình Webhook Repo

Bước 7: Tạo Jenkins job Freestyle hoặc Pipeline

- Chọn "Git" làm SCM
- Dán URL GitHub repo
- Trong "Build Triggers", tick vào:
- Check GitHub hook trigger for GITScm polling

Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?

Hình 22: Jenkins Trigger webhook

Check Terminal của ngrok:

```
HTTP Requests
-----
16:06:37.459 +07 POST /github-webhook/ 200 OK
```

Hình 23: Liên kết thành công pipeline

Ghi chú

- Ngrok URL sẽ thay đổi mỗi lần restart, trừ khi bạn dùng gói trả phí → phải cập nhật lại webhook nếu restart ngrok

4.3 Tạo ECW2 để tiến hành đẩy lên cloud

Tạo instance

ap-southeast-1.console.aws.amazon.com/ec2/home?region=ap-southeast-1#LaunchInstances:

EC2 > Instances > Launch an instance

Name and tags Info

Name: auth-demo Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Recents Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux Debian

Amazon Machine Image (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type Free tier eligible

Summary

Number of instances: 1

Software Image (AMI): Canonical, Ubuntu, 24.04, amd64...read more

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year of running an instance, you can run t2.micro instances for free.

Cancel Launch instance Preview code

Hình 24: Tạo instance

Mở port web : 8000

Security group rule 2 (TCP, 8000, 0.0.0.0/0, Web) Remove

Type: Custom TCP

Protocol: TCP

Port range: 8000

Source type: Anywhere

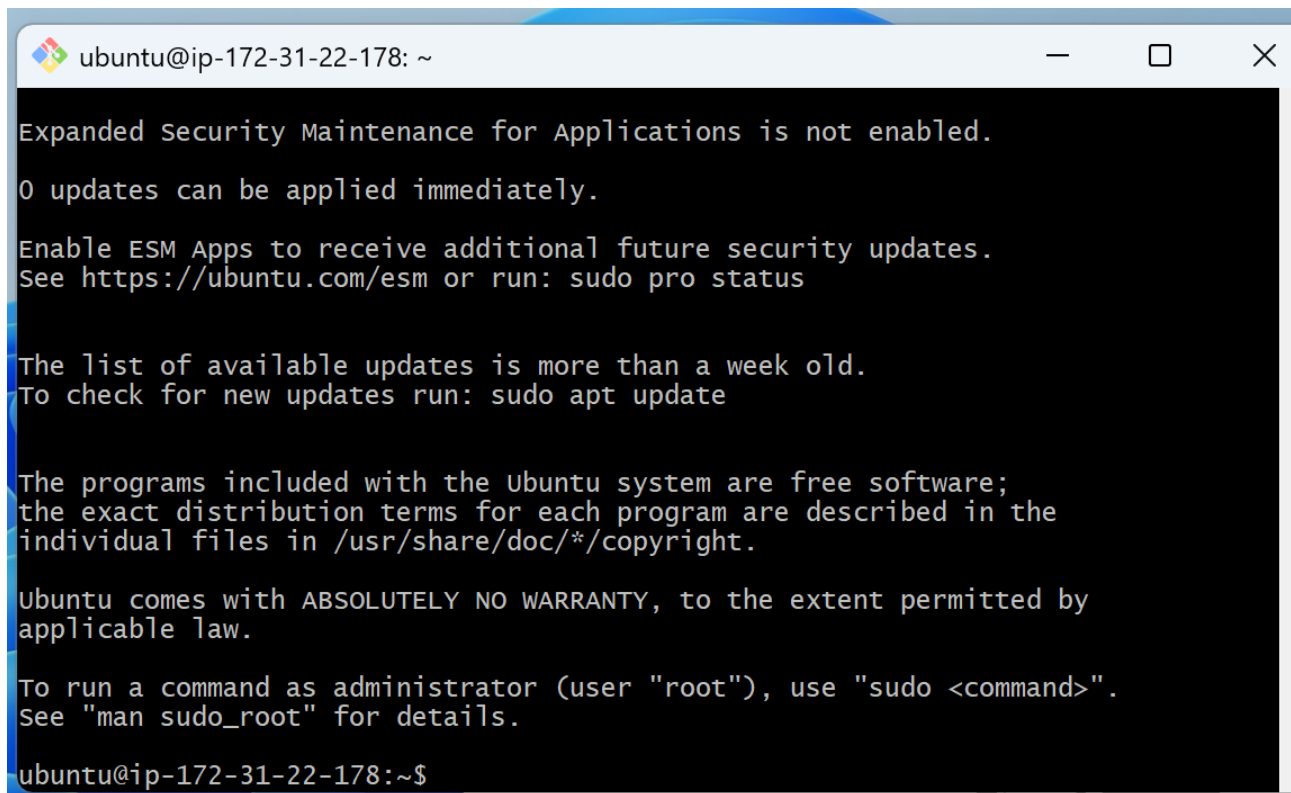
Source: 0.0.0.0/0

Description - optional: Web

Hình 25: Mở port ec2

SSH VÀO EC2, CÀI ĐẶT DOCKER

```
ssh -i "your-key.pem" ubuntu@<EC2-PUBLIC-IP>
```

A screenshot of a terminal window titled 'ubuntu@ip-172-31-22-178: ~'. The terminal displays several messages: 'Expanded Security Maintenance for Applications is not enabled.', '0 updates can be applied immediately.', 'Enable ESM Apps to receive additional future security updates. See https://ubuntu.com/esm or run: sudo pro status', 'The list of available updates is more than a week old. To check for new updates run: sudo apt update', 'The programs included with the Ubuntu system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright.', 'Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.', 'To run a command as administrator (user "root"), use "sudo <command>". See "man sudo_root" for details.', and the prompt 'ubuntu@ip-172-31-22-178:~\$'.

Hình 26: SSH EC2

Sau đó cài Docker

```
sudo apt update
sudo apt install docker.io -y
sudo systemctl enable docker
sudo usermod -aG docker ubuntu
```

KẾT NỐI JENKINS VỚI EC2 (DÙNG SSH KEY .pem TỪ AWS)

- Cài đặt plugin SSH Agent Plugin
- Vào Jenkins Web UI
- Vào Manage Jenkins → Plugins
- Chuyển sang tab Available
- Tìm: SSH Agent
- Tick chọn và bấm Install without restart

Lưu trong folder ở volume

Docker cp

```
"C:\Users\2ien\Desktop\NGUYENLEVANQUYEN.pem"
```

```
jenkins:/home/NGUYENLEVANQUYEN.pem
```

cấp quyền cho file:

```
docker exec -u root -it jenkins chmod 600 /home/NGUYENLEVANQUYEN.pem
```

Kiểm tra:

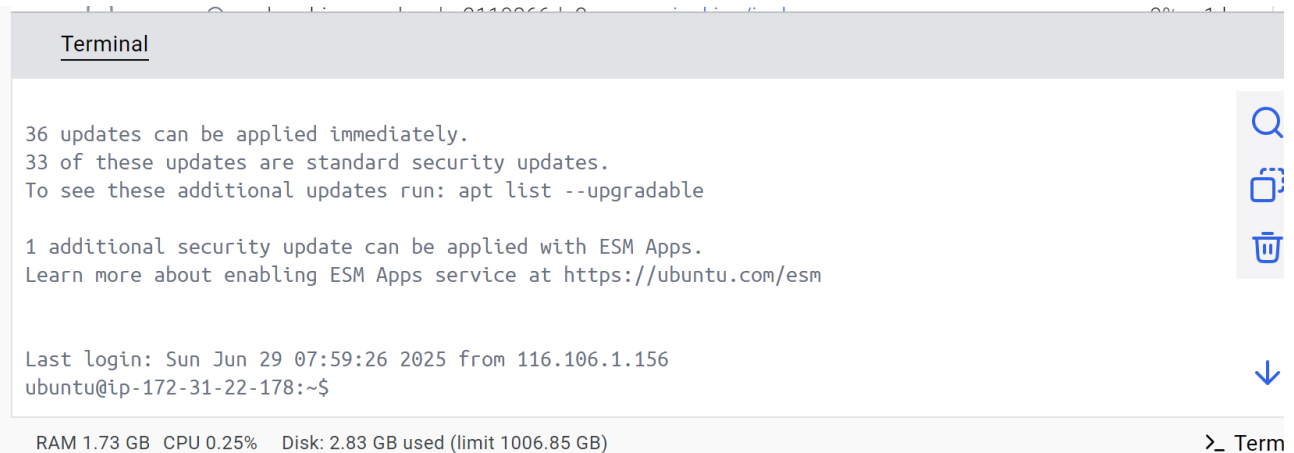
```
C:\Users\2ien>docker exec -u root -it jenkins ls -l /home
total 4
-rwxr-xr-x 1 root root 1678 Jun 29 07:51 NGUYENLEVANQUYEN.pem
```

Hình 27: Kiểm tra quyền key ssh

SSH từ Jenkins sử dụng PEM trong /home

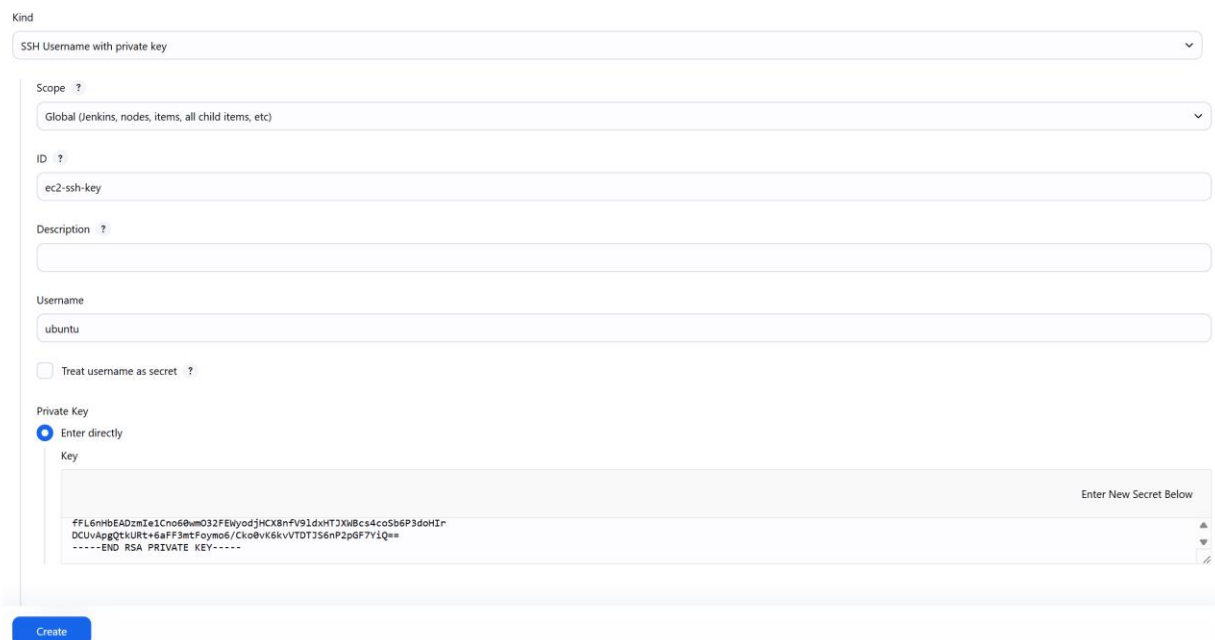
Trong Jenkinsfile hoặc script, bạn có thể dùng:

```
ssh -i /home/NGUYENLEVANQUYEN.pem ubuntu@<ec2-ip>
```



Hình 28: SSH Jenkins tới EC2

Tạo credentials cho SSH:



The screenshot shows the Jenkins 'SSH Credentials' configuration page. The 'Kind' dropdown is set to 'SSH Username with private key'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'ID' field contains 'ec2-ssh-key'. The 'Description' field is empty. The 'Username' field contains 'ubuntu'. The 'Treat username as secret' checkbox is unchecked. The 'Private Key' section has 'Enter directly' selected. The 'Key' text area contains a base64-encoded private key for an RSA key, starting with 'FFLEnhbEADzmTe1Cno60wm032FEkyodjHCK8nFv91dxHT3Xh8cs4coSb6P3dohIr' and ending with '-----END RSA PRIVATE KEY-----'. A 'Create' button is at the bottom left.

Hình 29: Credentials cho SSH

Thêm Jenkinsfile vào github:

```
pipeline {
  agent any

  environment {
    IMAGE_NAME = 'auth-demo-api-view'
    TAG = 'latest'
    DOCKERHUB_USER = 'nguyenlevanquyen'
  }
}
```

```

stages {
  stage('Load Secrets') {
    steps {
      withCredentials([
        string(credentialsId: 'port', variable: 'PORT'),
        string(credentialsId: 'mongo-uri', variable: 'MONGO_URI'),
        string(credentialsId: 'jwt-secret', variable: 'JWT_SECRET'),
        string(credentialsId: 'session-secret', variable: 'SESSION_SECRET')
      ]) {
        script {
          env.PORT = PORT
          env.MONGO_URI = MONGO_URI
          env.JWT_SECRET = JWT_SECRET
          env.SESSION_SECRET = SESSION_SECRET
        }
      }
    }
  }

  stage('Create .env') {
    steps {
      writeFile file: '.env', text: """
PORT=${PORT}
MONGO_URI=${MONGO_URI}
JWT_SECRET=${JWT_SECRET}
SESSION_SECRET=${SESSION_SECRET}
"""
    }
  }
}

```

```

stage('Build Docker Image') {
    steps {
        sh 'docker build -t $DOCKERHUB_USER/$IMAGE_NAME:$TAG .'
    }
}

/*
stage('Run Container') {
    steps {
        sh """
            docker stop webapp || true
            docker rm webapp || true
            docker run -d --env-file .env -p 8000:$PORT --name webapp
$DOCKERHUB_USER/$IMAGE_NAME:$TAG
        """
    }
}

*/

stage('Deploy to EC2') {
    steps {
        sshagent (credentials: ['ec2-ssh-key']) {
            sh """
ssh -o StrictHostKeyChecking=no ubuntu@54.255.40.151 <<EOF
docker stop webapp || true
docker rm webapp || true
docker pull ${DOCKERHUB_USER}/${IMAGE_NAME}:${TAG}

echo "PORT=${PORT}" > /home/ubuntu/.env
echo "MONGO_URI=${MONGO_URI}" >> /home/ubuntu/.env
echo "JWT_SECRET=${JWT_SECRET}" >> /home/ubuntu/.env
echo "SESSION_SECRET=${SESSION_SECRET}" >> /home/ubuntu/.env


```

```
docker run -d --env-file /home/ubuntu/.env -p 127.0.0.1:8000:${PORT} --name webapp ${DOCKERHUB_USER}/${IMAGE_NAME}:${TAG}
```

EOF

```
        """"
    }
}
}
}
}
```

Cấu hình domain bằng No – ip cho ec2

 **Modify Hostname : authdemoxxx.zapto.org**

IPv4 Address ⓘ

Last Update ⓘ
Jul 7, 2025
05:31 PDT

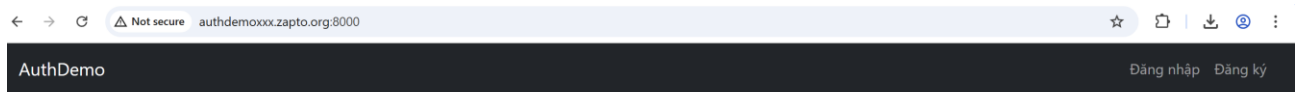
☐ Offline ⓘ **Upgrade to Enhanced** to enable offline settings.

MX Records
[+ Add MX Records](#)

Cancel

Update Hostname

Hình 30: Cấu hình domain cho EC2



Trang chủ

Bạn chưa đăng nhập. Vui lòng [Đăng nhập](#) hoặc [Đăng ký](#).

Hình 31: Trang chủ web với domain

4.4 Cấu hình Let's encrypt và revers proxy để cấp https

Reverse Proxy là một máy chủ trung gian (ví dụ: Nginx) đứng trước ứng dụng web của bạn, nhận toàn bộ request từ client, rồi chuyển tiếp (proxy) vào phía sau cho ứng dụng xử lý.

Client (trình duyệt)



https://authdemo.zapto.org (port 443)



[Nginx] ← Reverse Proxy trên EC2 (port 443, 80)



localhost:8000 → [Express app (Docker container)]

Cài đặt Nginx

```
sudo apt install nginx -y
```

Mở port 80 và 443 trong ec2

Edit inbound rules [Info](#)
Inbound rules control the incoming traffic that's allowed to reach the instance.

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info	
sgr-03848a31d383dc0a2	HTTP	TCP	80	Custom	0.0.0.0/0	Delete
sgr-0b77c9b74fc4919ca	Custom TCP	TCP	8000	Custom	0.0.0.0/0	Delete
sgr-090d96f5686e5c2cf	SSH	TCP	22	Custom	0.0.0.0/0	Delete
-	Custom TCP	TCP	443	Any...	0.0.0.0/0	Delete

[Add rule](#)

Hình 32: Mở port 80 và 443 EC2

Tạo file cấu hình reverse proxy

```
sudo nano /etc/nginx/sites-available/express
```

```
GNU nano 7.2 /etc/nginx/sites-available/express *
server {
    listen 80;
    server_name authdemoxxx.zapto.org;

    location / {
        proxy_pass http://localhost:8000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Hình 33: Cấu hình reverse proxy

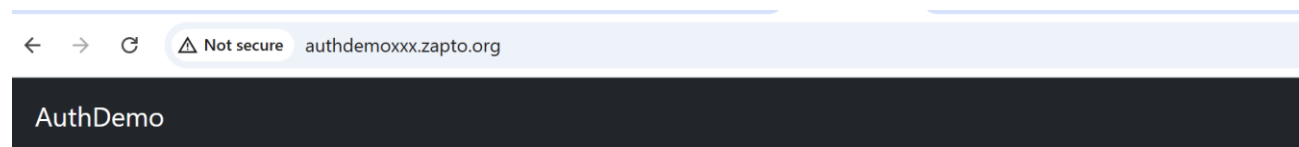
Kích hoạt site và reload Nginx

```
sudo ln -s /etc/nginx/sites-available/express /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
sửa lại jenkins file
```

Chạy lại jenkins

- Giờ không cần phải chỉ cập vào port 8000 của docker nữa, bảo mật hơn.

Client truy cập trực tiếp vào port 80 của nginx



Trang chủ

Bạn chưa đăng nhập. Vui lòng [Đăng nhập](#) hoặc [Đăng ký](#).

Hình 34: An toàn hơn khi không cần truy cập port của web trên docker

Cấu hình let's crypt Cấp HTTPS (port 443) với Let's Encrypt

Cài Certbot và plugin Nginx

```
sudo apt install certbot python3-certbot-nginx -y
```

Chạy Certbot để cấp SSL

```
sudo certbot --nginx -d authdemoxxx.zapto.org
```

Điền yêu cầu

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-22-178:~$ sudo nano /etc/nginx/sites-available/express
ubuntu@ip-172-31-22-178:~$ sudo certbot --nginx -d authdemoxxx.zapto.org
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Enter email address (used for urgent renewal and security notices)
(Enter 'c' to cancel): nguyenlevanquyen8844@gmail.com

-----
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.5-February-24-2025.pdf. You must
agree in order to register with the ACME server. Do you agree?
-----
(Y)es/(N)o: y

-----
Would you be willing, once your first certificate is successfully issued, to
share your email address with the Electronic Frontier Foundation, a founding
partner of the Let's Encrypt project and the non-profit organization that
develops Certbot? We'd like to send you email about our work encrypting the web,
EFF news, campaigns, and ways to support digital freedom.
-----
(Y)es/(N)o: y
Account registered.
Requesting a certificate for authdemoxxx.zapto.org
█
```

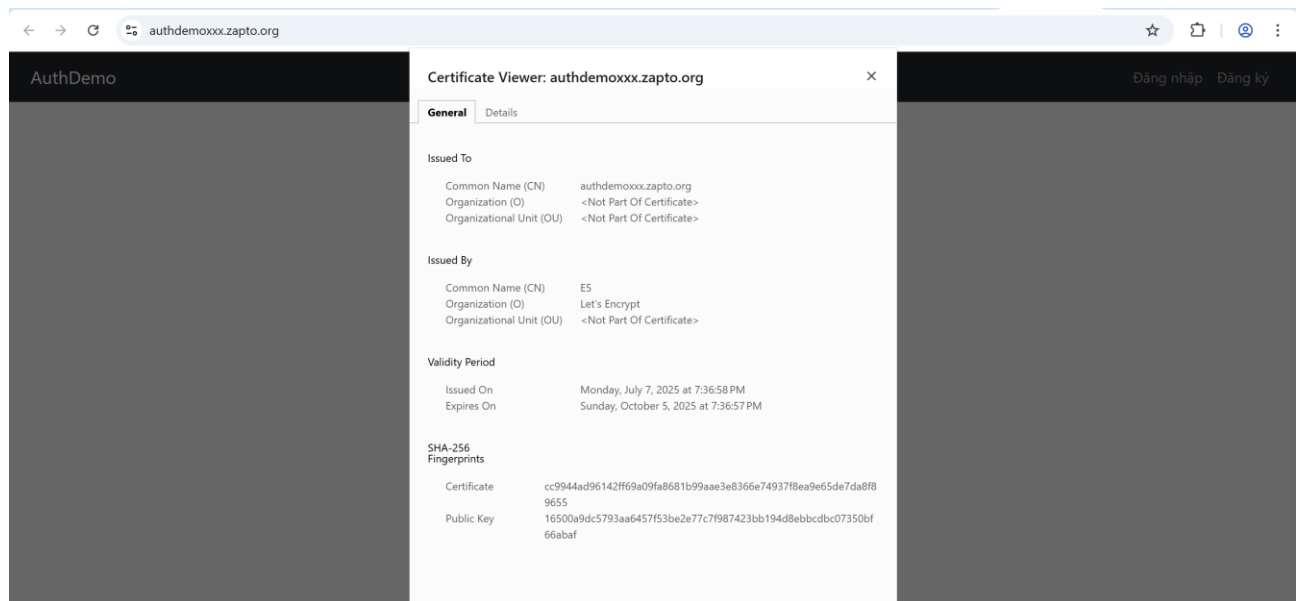
Hình 35: Yêu cầu cấp SSL

```
Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/authdemoxxx.zapto.org/fullchain.pem
Key is saved at: /etc/letsencrypt/live/authdemoxxx.zapto.org/privkey.pem
This certificate expires on 2025-10-05.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

Deploying certificate
Successfully deployed certificate for authdemoxxx.zapto.org to /etc/nginx/sites-enabled/express
Congratulations! You have successfully enabled HTTPS on https://authdemoxxx.zapto.org

-----
If you like Certbot, please consider supporting our work by:
 * Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate
 * Donating to EFF: https://eff.org/donate-le
-----
ubuntu@ip-172-31-22-178:~$ █
```

Hình 36: Thành công cấp SSL



Hình 37: Giao diện chứng chỉ trên web

Thêm tự động gia hạn cho SSL

```
sudo certbot renew --dry-run
```

```
ubuntu@ip-172-31-22-178:~$ sudo certbot renew --dry-run
Saving debug log to /var/log/letsencrypt/letsencrypt.log

-----
Processing /etc/letsencrypt/renewal/authdemoxxx.zapto.org.conf
-----
Account registered.
Simulating renewal of an existing certificate for authdemoxxx.zapto.org

-----
Congratulations, all simulated renewals succeeded:
  /etc/letsencrypt/live/authdemoxxx.zapto.org/fullchain.pem (success)
-----
```

Hình 38: Tự động gia hạn cho SSL

4.5 Tạo backup trên EC2

Thư mục lưu backup : /home/ubuntu/backups/

File được tạo trong thư mục backup

Loại file	Định dạng	Nội dung
.env.<timestamp>	/home/ubuntu/backups/.env.YYYYMMDD_HHMMSS	Bản sao cấu hình môi trường ứng dụng
image_<timestamp>.tar	/home/ubuntu/backups/image_YYMMDD_HHMMSS.tar	File Docker image đã được docker save

Bảng 5: File cần backup

Ví dụ cây thư mục sau vài lần backup:

/home/ubuntu/backups/
├── .env.20250714_030000
├── image_20250714_030000.tar
├── .env.20250715_030000
└── image_20250715_030000.tar

Tóm tắt đường dẫn chính:

Mục	Đường dẫn
Thư mục backup chính	/home/ubuntu/backups
Bản sao .env	/home/ubuntu/backups/.env.<timestamp>
Docker image .tar	/home/ubuntu/backups/image_<timestamp>.tar
File .env đang dùng để chạy container	/home/ubuntu/.env
File script backup	/home/ubuntu/backup_webapp.sh
File script restore	/home/ubuntu/restore_webapp.sh

Bảng 6: Đường dẫn cần cho backup

Tạo Script:

- **Script backup:** lưu .env + image .tar

- **Script restore:**

+ Khôi phục bản backup mới nhất (*mặc định*)

+ Chỉ định bản backup theo YYYYMMDD_HHMMSS

Script Backup (backup_webapp.sh):

```
#!/bin/bash

TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
BACKUP_DIR="/home/ubuntu/backups"
IMAGE_NAME="auth-demo-api-view"
TAG="latest"

mkdir -p $BACKUP_DIR

# Backup .env
cp /home/ubuntu/.env $BACKUP_DIR/.env.$TIMESTAMP

# Backup Docker image
docker save $IMAGE_NAME:$TAG $BACKUP_DIR/image_${TIMESTAMP}.tar -o

echo "Backup hoàn tất:"
echo "ENV: $BACKUP_DIR/.env.$TIMESTAMP"
echo "TAR: $BACKUP_DIR/image_${TIMESTAMP}.tar"
```

```

GNU nano 7.2 /home/ubuntu/backup_webapp.sh *
#!/bin/bash

TIMESTAMP=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/home/ubuntu/backups"
IMAGE_NAME="auth-demo-api-view"
TAG="latest"

mkdir -p $BACKUP_DIR

# Backup .env
cp /home/ubuntu/.env $BACKUP_DIR/.env.$TIMESTAMP

# Backup Docker image
docker save $IMAGE_NAME:$TAG -o $BACKUP_DIR/image_${TIMESTAMP}.tar

echo "Backup hoàn tất:"
echo "ENV: $BACKUP_DIR/.env.$TIMESTAMP"
echo "TAR: $BACKUP_DIR/image_${TIMESTAMP}.tar"

```

Hình 39: Script backup

Cấp quyền:

```
chmod +x /home/ubuntu/backup_webapp.sh
```

Tự động dùng mỗi ngày:

```
crontab -e
```

```
0 3 * * * /home/ubuntu/backup_webapp.sh >> /home/ubuntu/backup.log 2>&1
```

Script Restore (restore_webapp.sh):

```

#!/bin/bash

BACKUP_DIR="/home/ubuntu/backups"
IMAGE_NAME="auth-demo-api-view"
TAG="latest"
CONTAINER_NAME="webapp"

# Nhận tham số ngày giờ (nếu có)
TIMESTAMP="$1"

if [ -z "$TIMESTAMP" ]; then
    # Khôi phục bản mới nhất
    ENV_FILE=$(ls -t $BACKUP_DIR/.env.* 2>/dev/null | head -n 1)

```

```
TAR_FILE=$(ls -t $BACKUP_DIR/image_*.tar 2>/dev/null | head -n 1)
else
    ENV_FILE="$BACKUP_DIR/.env.$TIMESTAMP"
    TAR_FILE="$BACKUP_DIR/image_${TIMESTAMP}.tar"
fi

# Kiểm tra tồn tại
if [[ ! -f "$ENV_FILE" || ! -f "$TAR_FILE" ]]; then
    echo "Không tìm thấy file backup với thời gian: $TIMESTAMP"
    echo "ENV: $ENV_FILE"
    echo "TAR: $TAR_FILE"
    exit 1
fi

echo "Khôi phục bản backup:"
echo "ENV: $ENV_FILE"
echo "TAR: $TAR_FILE"

# Dừng & xóa container cũ
docker stop $CONTAINER_NAME || true
docker rm $CONTAINER_NAME || true

# Load image và copy .env
docker load -i "$TAR_FILE"
cp "$ENV_FILE" /home/ubuntu/.env

# Đọc PORT từ .env
PORT=$(grep PORT /home/ubuntu/.env | cut -d '=' -f2)

# Chạy lại container
```

```
docker run -d --env-file /home/ubuntu/.env -p 127.0.0.1:8000:$PORT --name $CONTAINER_NAME $IMAGE_NAME:$TAG
```

```
echo "Khôi phục hoàn tất!"
```

```
GNU nano 7.2 /home/ubuntu/restore_webapp.sh *
r1
echo "Khôi phục bản backup:"
echo "ENV: $ENV_FILE"
echo "TAR: $TAR_FILE"

# Dừng & xóa container cũ
docker stop $CONTAINER_NAME || true
docker rm $CONTAINER_NAME || true

# Load image và copy .env
docker load -i "$TAR_FILE"
cp "$ENV_FILE" /home/ubuntu/.env

# Đọc PORT từ .env
PORT=$(grep PORT /home/ubuntu/.env | cut -d '=' -f2)

# Chạy lại container
docker run -d --env-file /home/ubuntu/.env -p 127.0.0.1:8000:$PORT --name $CONTAINER_NAME $IMAGE_NAME:$TAG

echo "Khôi phục hoàn tất!"

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^E Execute    ^G Location   M-U Undo      M-A Set Mark  M-I To Bracket
^X Exit      ^R Read File  ^N Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-B Redo      M-C Copy      ^Q Where Was
```

Hình 40: Script Restore

Cấp quyền cho restore file

```
chmod +x /home/ubuntu/restore_webapp.sh
```

Cách sử dụng:

Mục đích	Lệnh
Khôi phục bản mới nhất	./restore_webapp.sh
Khôi phục bản cụ thể	./restore_webapp.sh 20250715_030000
Xem danh sách bản backup	ls /home/ubuntu/backups
Xem log container	docker logs -f webapp

Bảng 7: Cách sử dụng script

Kiểm tra sử dụng:

```

ubuntu@ip-172-31-22-178:~$ ./backup_webapp.sh
Backup hoàn tất:
ENV: /home/ubuntu/backups/.env.20250715_100136
TAR: /home/ubuntu/backups/image_20250715_100136.tar
ubuntu@ip-172-31-22-178:~$ ls /home/ubuntu/backups
image_20250715_100136.tar
ubuntu@ip-172-31-22-178:~$ ./restore_webapp.sh
Khôi phục bản backup:
ENV: /home/ubuntu/backups/.env.20250715_100136
TAR: /home/ubuntu/backups/image_20250715_100136.tar
webapp
webapp
Loaded image: auth-demo-api-view:latest
7e0252cebc54b36925b889a43d43ecb7ca9d958f34b237aebc5fe147e629266b
Khôi phục hoàn tất!
ubuntu@ip-172-31-22-178:~$ docker logs -f webapp

> auth-project@1.0.0 start
> node --env-file=.env index.js

Server is running
Database connected

```

Hình 41: Backup

URL Demo: <https://youtu.be/oO4LxJiEF2U>

URL Demo web động: <https://youtu.be/zvFqw3MPVAQ>

V. Đánh giá và kết luận

Trong quá trình triển khai hệ thống CI/CD sử dụng Jenkins, Docker và EC2, đồ án đã đạt được các mục tiêu chính:

- **Tự động hóa quy trình build và deploy** ứng dụng từ GitHub khi có commit mới, thông qua Webhook.
- **Đóng gói ứng dụng bằng Docker**, đảm bảo sự nhất quán giữa môi trường phát triển và môi trường triển khai.
- **Triển khai thành công trên EC2**, với hệ thống tự tạo .env, chạy container độc lập và bảo mật.

- **Quản lý biến môi trường bằng Jenkins Credentials**, đảm bảo không lộ thông tin nhạy cảm trong pipeline.

Ngoài ra, hệ thống **sao lưu và khôi phục ứng dụng** (backup & restore) cũng được thiết lập, giúp dễ dàng phục hồi trạng thái hệ thống nếu xảy ra sự cố.

Tuy nhiên, vẫn còn một số điểm cần cải thiện:

- Jenkins hiện đang chạy qua ngrok, chưa thật sự ổn định lâu dài cho production.
- Docker Hub private yêu cầu đăng nhập khi pull trên EC2, cần tối ưu cơ chế cache hoặc chuyển sang registry nội bộ.
- Chưa tích hợp kiểm thử tự động (unit test) trong pipeline.

Kết luận

Đồ án đã giúp sinh viên tiếp cận và triển khai thành công một quy trình DevOps cơ bản nhưng hiệu quả, áp dụng các công cụ thực tế như:

- Jenkins (tự động hóa build/deploy)
- Docker (đóng gói ứng dụng)
- GitHub Webhook (trigger CI)
- Amazon EC2 (hạ tầng triển khai)
- Ngrok (mở truy cập Jenkins nội bộ)
- Backup/Restore Shell Script (quản lý trạng thái hệ thống)

Thông qua đồ án này, người thực hiện đã hiểu rõ hơn về cách hoạt động của một pipeline CI/CD, cách phối hợp nhiều công cụ lại với nhau, và khả năng xử lý sự cố trong môi trường thật. Đây là nền tảng vững chắc để phát triển các hệ thống DevOps chuyên nghiệp hơn trong tương lai.

Tài Liệu Tham Khảo

- [1] Docker Documentation, <https://docs.docker.com>
- [2] Jenkins Official Documentation, <https://www.jenkins.io/doc/>
- [3] GitHub Webhooks Documentation, <https://docs.github.com/en/webhooks>
- [4] Ngrok Documentation, <https://ngrok.com/docs>
- [5] Stack Overflow – How to deploy Docker image to EC2, <https://stackoverflow.com>
- [6] Emily Freeman, *DevOps For Dummies*, Wiley, 2020.
- [7] URL Repository: <https://github.com/2ien/auth-demo-api-view.git>

- [8] URL Demo: <https://youtu.be/oO4LxJiEF2U>
- [9] URL Demo: <https://youtu.be/zvFqw3MPVAQ>