

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 hệ thống quản lý học sinh với kiến trúc
đa môi trường, tích hợp Jenkins CI/CD

GVHD: Đinh Nguyễn Anh Dũng

Nhóm sinh viên thực hiện:

1. Nguyễn Thị Hiền
2. Vũ Đức Minh

- MSSV: 21521461
MSSV: 21522348

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

....., ngày.....tháng.....năm 20...

Người nhận xét

(Ký tên và ghi rõ họ tên)

LỜI MỞ ĐẦU

Trong bối cảnh công nghệ thông tin ngày càng phát triển, việc áp dụng các công nghệ tiên tiến vào quy trình quản lý học sinh trở nên vô cùng quan trọng để nâng cao hiệu quả công việc và tiết kiệm thời gian. Việc xây dựng hệ thống quản lý học sinh không chỉ giúp theo dõi quá trình học tập mà còn mang lại sự tiện lợi trong việc quản lý dữ liệu, tối ưu hóa quy trình công việc, và đảm bảo tính minh bạch trong việc quản lý.

Ứng dụng này trình bày quá trình phát triển hệ thống quản lý học sinh với kiến trúc đa môi trường, tích hợp Jenkins CI/CD nhằm tự động hóa quy trình kiểm thử và triển khai phần mềm. Hệ thống được thiết kế với mục tiêu cung cấp một nền tảng mạnh mẽ, dễ sử dụng, hỗ trợ các tính năng như quản lý thông tin học sinh, theo dõi điểm số, và báo cáo kết quả học tập. Đặc biệt, việc áp dụng Jenkins giúp tự động hóa quy trình xây dựng, kiểm thử và triển khai, nâng cao tính ổn định và hiệu quả của hệ thống.

MỤC LỤC

I. Tổng quan	6
1. Giới thiệu	6
2. Mục tiêu và lợi ích	6
3. Nội dung thực hiện	6
4. So sánh tổng quan về các framework	7
5. CI/CD Pipeline chi tiết cho từng môi trường	7
6. Mô hình quản lý code (Git Flow)	9
II. Khái niệm về DevOps, DevSecOps và CI/CD	10
1 DevOps, DevSecOps	10
1.1 DevOps	10
1.2 DevSecOps	11
2. Continuous Integration và Continuous Delivery (CI/CD)	12
2.1 Khái niệm về CI/CD	12
2.2 Tầm quan trọng của Jenkins CI/CD Docker trong DevOps	14
III. Các công cụ và công nghệ sử dụng trong quy trình	17
1 Docker	17
1.1 Khái niệm về Docker và containerization	17
1.2 Tối ưu docker image	18
1.3 Docker trong quy trình CI/CD	19
2. Gitlab, Gitflow	20
2.1 Khái niệm về Gitlab	20
2.2 Gitlab CI/CD	22
2.3 Gitflow	23
3. Kubernetes	24
3.1 Tổng quan về Kubernetes	24
3.2 Các thành phần chính của Kubernetes	25
IV. Các kiến trúc xây dựng trong xây dựng CICD pipeline với Jenkins	30
1. Sơ đồ kiến trúc sử dụng các công cụ	30
2. Sơ đồ pipeline trong gitflow	31
3. Kiến trúc của k8s:	33
V. Demo ứng dụng	34
1. Ứng dụng demo với Jenkins CICD pipeline	34
VI. Quy trình thực hiện	35
1. Commit code tính năng	36
VII. Ứng dụng/ Kết quả thực nghiệm/So sánh-Đánh giá	39
1. Kết quả thực nghiệm	40
2. So sánh-Đánh giá	40
VIII. Kết luận	40
1. Ưu điểm	41
2. Khuyết điểm	41
3. Hướng phát triển	41
TÀI LIỆU THAM KHẢO	42

Mục lục ảnh

Hình 1: Quy trình của DevOps	11
Hình 2: Quy trình của DevSecOps	13
Hình 3: Quy trình của Continuous delivery và continuous deployment	14
Hình 4: Quy trình CI/CD	15
Hình 5: Sơ đồ tổng quan DevOps	16
Hình 6: Docker	18
Hình 7: Kiến trúc của Docker	19
Hình 8: Gitlab	22
Hình 9: Gitlab CI/CD	23
Hình 10: Tổng quan về Gitflow	24
Hình 11: Kubernetes	25
Hình 12: Kiến trúc của Kubernetes	26
Hình 13: Kubernetes Pod	27
Hình 14: Kubernetes Deployment	28
Hình 15: Kubernetes Service	29
Hình 16: Kubernetes Namespaces	30
Hình 17: Kubernetes ConfigMap và Secret	30
Hình 18: Sơ đồ các công cụ sử dụng	31
Hình 19: Sơ đồ quy trình Gitflow	32
Hình 20: Chi tiết sơ đồ quy trình Gitflow	32
Hình 21: Kiến trúc k8s	34
Hình 22: Ứng dụng demo	35
Hình 23: Hiện trạng ban đầu của ứng dụng	36

I. Tổng quan

1. Giới thiệu

Trong lĩnh vực phát triển phần mềm hiện đại, CI/CD đóng vai trò thiết yếu trong việc tự động hóa quá trình phát triển và triển khai ứng dụng. Việc triển khai CI/CD giúp cải thiện tốc độ ra mắt sản phẩm, tăng tính ổn định và chất lượng của các bản phát hành, cũng như giảm thiểu lỗi do quá trình triển khai thủ công gây ra. Trong đề tài này, chúng em sẽ nghiên cứu và xây dựng một CI/CD pipeline hoàn chỉnh, sử dụng Jenkins và GitLab, cho một ứng dụng demo đơn giản. Ứng dụng bao gồm hai thành phần chính: backend được xây dựng bằng Spring Boot và frontend bằng React, giúp quản lý các quy trình CI/CD cho cả hai phần này.

2. Mục tiêu và lợi ích

Mục tiêu của đề tài bao gồm:

- Xây dựng một CI/CD pipeline hoàn chỉnh: Tạo ra một quy trình CI/CD từ đầu đến cuối cho ứng dụng, bao gồm các bước build, kiểm thử, và triển khai.
- Sử dụng Jenkins cho tự động hóa: Jenkins sẽ được sử dụng để tự động hóa toàn bộ quy trình phát triển, từ xây dựng mã nguồn đến kiểm thử, đảm bảo các thay đổi được kiểm tra liên tục trước khi triển khai.
- Triển khai ứng dụng trên các môi trường khác nhau: Sử dụng GitLab CI/CD để triển khai ứng dụng vào các môi trường khác nhau như staging và production, giúp quá trình phát triển và kiểm thử diễn ra thuận lợi.
- Đảm bảo CI/CD đạt chuẩn enterprise: Đề tài sẽ tập trung vào việc tuân thủ các tiêu chuẩn doanh nghiệp, bao gồm bảo mật, tính ổn định và khả năng mở rộng của pipeline.

3. Nội dung thực hiện

Ứng dụng demo là một hệ thống quản lý học sinh, có các chức năng cơ bản như

- CRUD (Create, Read, Update, Delete) học sinh: Cho phép người dùng thêm mới, xem, chỉnh sửa và xóa thông tin học sinh.
- UI: Giao diện được tối ưu để phản hồi nhanh và thân thiện với người dùng.
- Xử lý lỗi: Hệ thống sẽ xử lý và ghi nhận các lỗi có thể xảy ra trong quá trình sử dụng.
- Ghi log backend: Lưu lại các hành động trong backend, hỗ trợ việc theo dõi và kiểm tra lỗi khi cần thiết.

4. 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 toán nặng	Tốt cho I/O, không mạnh về CPU	Hiệu năng cao, ngôn ngữ biên dịch	Hiệu năng cao với .NET tối 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 Jenkins, 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

5. 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.

6. Mô hình quản lý code (Git Flow)

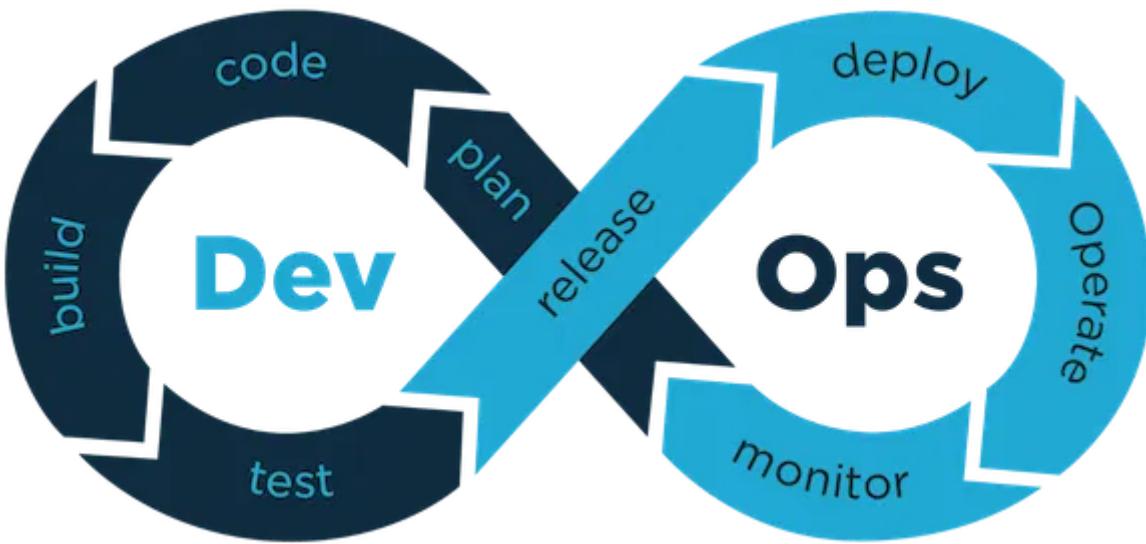
- Main: Đây là nhánh chứa mã nguồn đã qua kiểm duyệt và sẵn sàng để triển khai lên Production.
- Develop: Nhánh phát triển chính, nơi chứa mã nguồn đang được phát triển và thử nghiệm, nhưng chưa sẵn sàng cho Production.
- Feature branches: Dành cho các tính năng hoặc cải tiến mới. Mỗi tính năng sẽ có 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.

II. Khái niệm về DevOps, DevSecOps và CI/CD

1 DevOps, DevSecOps

1.1 DevOps

DevOps là sự kết hợp giữa Phát triển (Development) và Vận hành (Operations) nhằm mục đích nâng cao hiệu quả, tốc độ và bảo mật trong quá trình phát triển và phân phối phần mềm.



Hình 1: Quy trình của DevOps

DevOps có thể coi là một phương pháp hoặc văn hóa hoặc chức danh công việc dành cho những người có khả năng kết hợp tốt với kiến thức tốt, khả năng giao tiếp và thực hành quản lý tốt 2 giai đoạn phát triển phần mềm phía trên.

Cách thức hoạt động của DevOps gồm:

a. Lập kế hoạch và thiết kế

- Các nhóm Dev và Ops phối hợp để xác định mục tiêu, yêu cầu và kiến trúc của hệ thống phần mềm.
- Lập kế hoạch cho các giai đoạn phát triển, thử nghiệm, triển khai và vận hành.
- Thiết kế hệ thống phần mềm để đáp ứng nhu cầu của người dùng, đảm bảo dễ dàng phát triển và vận hành.

b. Phát triển và tích hợp

Các nhóm Dev sử dụng các phương pháp Agile để phát triển phần mềm theo từng phần nhỏ, liên tục tích hợp và kiểm tra. Việc tích hợp liên tục giúp đảm bảo các phần mềm được phát triển bởi các nhóm khác nhau có thể hoạt động đồng bộ.

Song song với đó, các nhóm Ops sẽ tham gia vào quá trình phát triển để đảm bảo khả năng triển khai và vận hành.

c. Triển khai

Quá trình này sẽ sử dụng các phương pháp tự động hóa để đảm bảo triển khai phần mềm nhanh chóng, hiệu quả. Các phương pháp triển khai phổ biến bao gồm Continuous Integration (CI) và Continuous Delivery (CD). Trong đó:

- CI tự động hóa việc xây dựng, kiểm tra và tích hợp phần mềm.
- CD tự động hóa việc triển khai phần mềm từ môi trường phát triển sang môi trường sản xuất.

d. Vận hành và giám sát

Các nhóm Ops sử dụng các công cụ giám sát để theo dõi hiệu suất, bảo mật và khả năng phục hồi của hệ thống. Đồng thời với đó, cần phân tích dữ liệu thu thập được để xác định các vấn đề tiềm ẩn, cải thiện hiệu quả vận hành cũng như cập nhật phần mềm và sửa lỗi khi cần thiết.

e. Phản hồi và cải tiến

Giai đoạn này bao gồm:

- Thu thập phản hồi từ người dùng và các bên liên quan để liên tục cải thiện phần mềm và quy trình DevOps.
- Sử dụng các phương pháp Agile để liên tục học hỏi và cải tiến quy trình làm việc.
- Tạo môi trường học tập và phát triển cho các nhóm Dev và Ops.

1.2 DevSecOps

DevSecOps là phương pháp tích hợp kiểm thử bảo mật ở mọi giai đoạn của quy trình phát triển phần mềm. Phương pháp này bao gồm các công cụ và quy trình khuyến khích cộng tác giữa các nhà phát triển, chuyên gia bảo mật và đội ngũ vận hành nhằm xây dựng phần mềm vừa hiệu quả vừa an toàn. DevSecOps mang đến sự chuyển đổi văn hóa, biến việc bảo mật trở thành trách nhiệm chung đối với tất cả các cá nhân xây dựng phần mềm.

DevSecOps là viết tắt của development (phát triển), security (bảo mật) và operations (vận hành). Đây là một phần mở rộng của phương pháp DevOps. Mỗi thuật ngữ đều xác định vai trò và trách nhiệm khác nhau của các đội ngũ phần mềm khi xây dựng ứng dụng phần mềm.

Phát triển

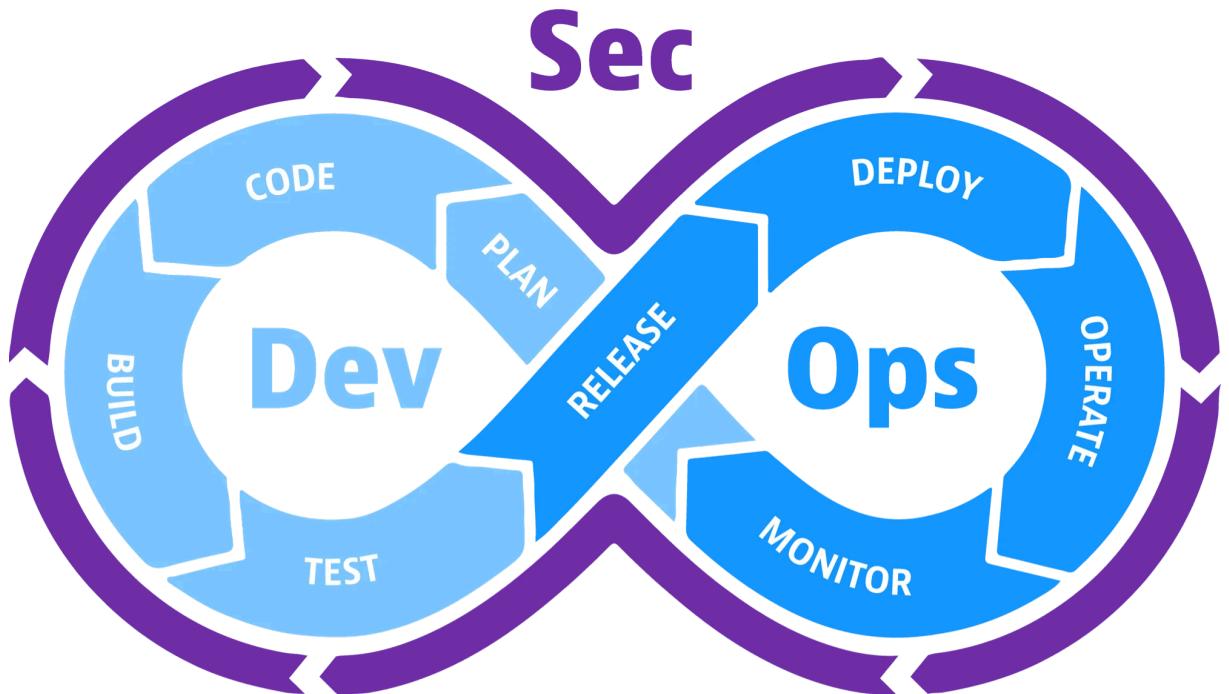
- Phát triển là quá trình lập kế hoạch, mã hóa, xây dựng và thử nghiệm ứng dụng.

Bảo mật

- Bảo mật có nghĩa là giới thiệu bảo mật sớm hơn trong chu trình phát triển phần mềm.

Hoạt động

- Nhóm hoạt động phát hành, giám sát và sửa chữa bất kỳ vấn đề phát sinh từ phần mềm.



Hình 2: Quy trình của DevSecOps

Một mô hình DevSecOps cơ bản bao gồm các thành phần:

- CI/CD Pipeline: thành phần phân phối các sản phẩm, dịch vụ một cách nhanh chóng và an toàn
- Infrastructure as code: thành phần giúp tài nguyên hệ thống có khả năng thay đổi đáp ứng co giãn khi cần thiết
- Monitoring: thành phần giám sát chặt chẽ khía cạnh an ninh trong từng giai đoạn
- Logging: ghi lại nhật ký các sự kiện bảo mật
- Microservice: chia nhỏ hệ thống lớn thành các thành phần nhỏ hơn để quản lý
- Communication: thành phần giao tiếp, liên lạc các nhóm kết hợp

2. Continuous Integration và Continuous Delivery (CI/CD)

2.1 Khái niệm về CI/CD

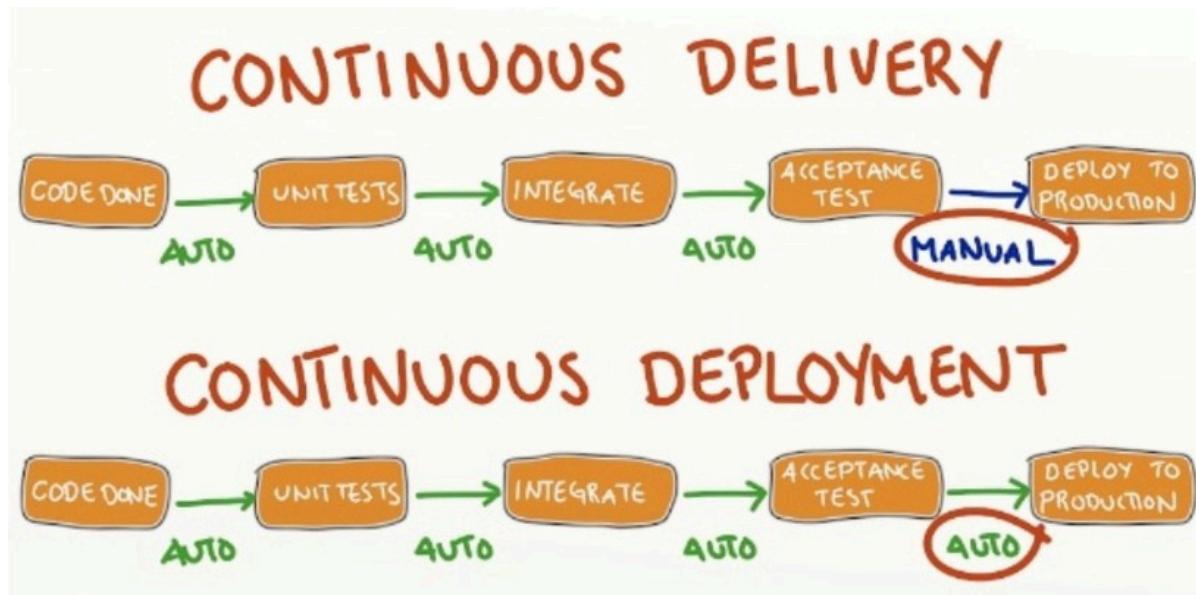
Continuous Integration (CI) là một quy trình tự động hóa việc kết hợp mã nguồn của các thành viên trong nhóm phát triển. Mỗi khi có thay đổi được đưa vào kho mã nguồn, hệ thống CI sẽ tự động thực hiện các bước như biên dịch, kiểm thử, và đóng gói phần mềm. Điều này giúp giảm thiểu lỗi do con người gây ra và đảm bảo rằng phần mềm luôn ở trong trạng thái ổn định và sẵn sàng để triển khai.

Lợi ích của việc sử dụng CI

- Giảm rủi ro tích hợp: Càng nhiều người làm thì sự tích hợp càng nguy hiểm. Tùy thuộc vào vấn đề thực sự tồi tệ như thế nào, việc sửa lỗi và giải quyết vấn đề có thể thực sự gây ra phiền phức và có thể có nghĩa là có nhiều thay đổi đối với mã nguồn. Thực hiện tích hợp hàng ngày hoặc thậm chí thường xuyên hơn có thể giúp giảm thiểu các loại vấn đề này ở mức tối thiểu.
- Chất lượng code cao hơn: Không cần phải lo lắng về các vấn đề xảy ra và tập trung nhiều hơn vào các tính năng của hệ thống giúp ta viết ra sản phẩm có chất lượng cao hơn.
- Code trên version control luôn hoạt động: Nếu commit phần nào đó làm hỏng việc build, các thành viên sẽ nhận ra điều này ngay lập tức, và vấn đề sẽ được giải quyết trước khi ai đó kéo code lỗi về

Khác với khái niệm CI - Continuous Integration - tích hợp liên tục là quy trình build và test tự động. CD - Continuous Delivery - nâng cao hơn một chút, bằng cách triển khai tất cả thay đổi về code (đã được build và test) đến môi trường testing hoặc staging. Continuous Delivery cho phép developer tự động hóa phần testing bên cạnh việc sử dụng unit test, kiểm tra phần mềm qua nhiều thước đo trước khi triển khai cho khách hàng (production). Những bài test này bao gồm UI testing, load testing, integration testing, API testing... Nó tự động hoàn toàn quy trình release phần mềm.

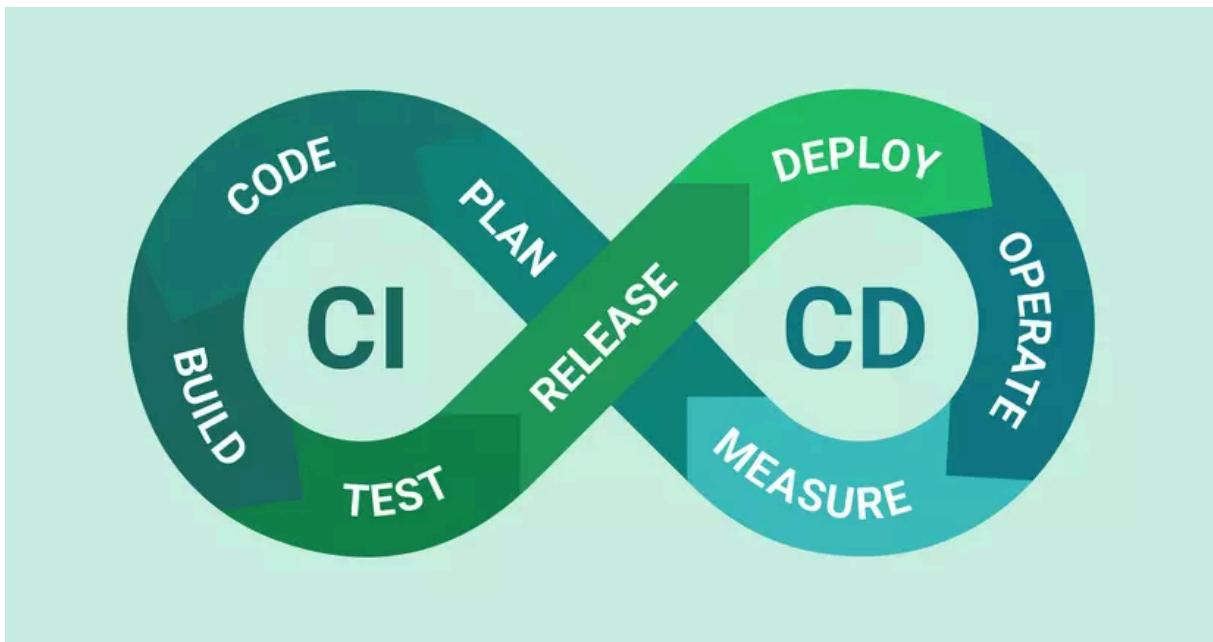
Có một khái niệm nữa là Continuous Deployment (cũng viết tắt là CD), và hai khái niệm này thường hay bị nhầm lẫn với nhau. Nếu Continuous Delivery là triển khai code lên môi trường staging, và deploy thủ công lên môi trường production, thì Continuous Deployment lại là kỹ thuật để triển khai code lên môi trường production một cách tự động.



Hình 3: Quy trình của Continuous delivery và continuous deployment

Mối quan hệ giữa CI và CD

CICD là một phương pháp thường xuyên cung cấp ứng dụng cho khách hàng bằng cách đưa tự động hóa vào các giai đoạn phát triển ứng dụng. Các khái niệm chính được gán cho CICD là tích hợp liên tục, phân phối liên tục và triển khai liên tục. CICD là một giải pháp cho các vấn đề tích hợp mã mới có thể gây ra cho các nhóm phát triển và hoạt động.



Hình 4: Quy trình CI/CD

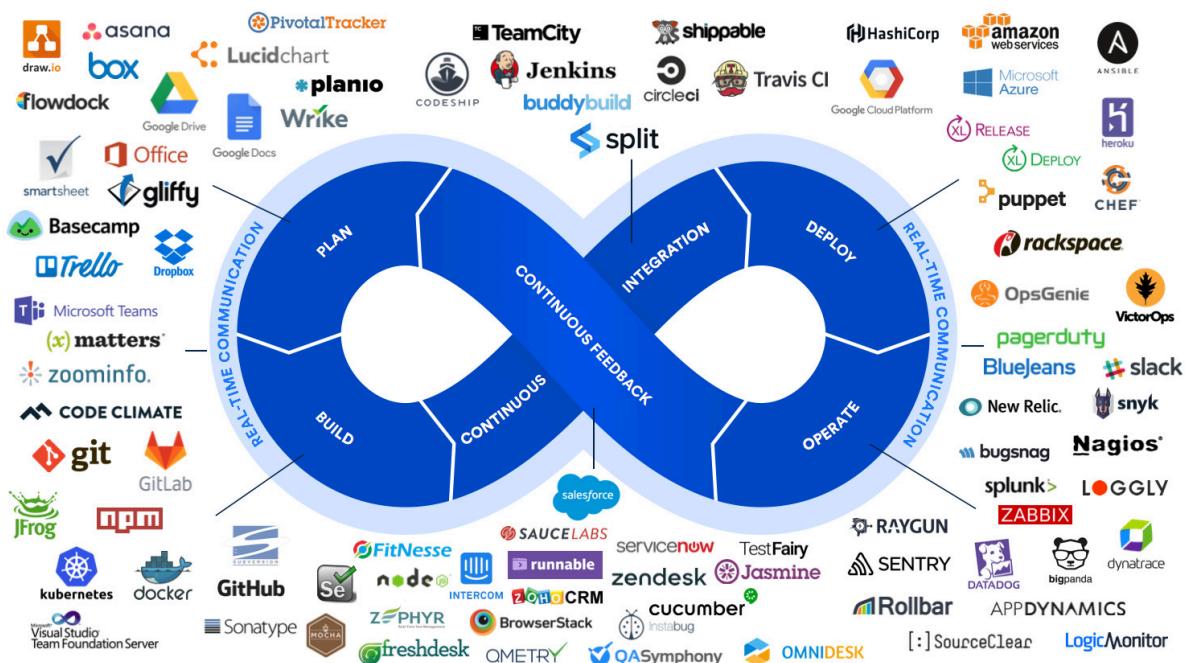
Cách thức hoạt động:

CI/CD được kết hợp để tạo ra một quy trình DevOps linh hoạt, tập trung vào việc phân phối phần mềm thường xuyên và đáng tin cậy. Đây là một phương pháp lặp đi lặp lại, hỗ trợ DevOps viết mã, tích hợp, chạy thử nghiệm. Đồng thời qua đó cung cấp các bản triển khai những tính năng mới theo thời gian thực.

Đặc điểm của quy trình CI/CD chính là tự động hóa để đảm bảo chất lượng code. Khi có sự thay đổi trong phần mềm, test automation sẽ xác định các vấn đề khác trước đó, push code lên nhiều môi trường ứng dụng khác nhau để thử nghiệm. Qua đó có thể đánh giá, kiểm soát chất lượng, hiệu suất, khả năng sử dụng và bảo mật.

2.2 Tầm quan trọng của Jenkin CI/CD Docker trong DevOps

Trong thời đại công nghệ số hiện nay, DevOps trở thành phương pháp quan trọng trong phát triển phần mềm, giúp kết hợp chặt chẽ giữa các đội phát triển (Development) và vận hành (Operations). Mục tiêu của DevOps là tối ưu hóa quy trình phát triển và triển khai, đảm bảo tính linh hoạt, hiệu suất và chất lượng cao cho các sản phẩm phần mềm.



Hình 5: Sơ đồ tổng quan DevOps

Jenkins CI/CD trong DevOps

Jenkins là một công cụ CI/CD mạnh mẽ, cho phép tự động hóa các quy trình từ tích hợp mã nguồn đến triển khai. Khi được áp dụng trong DevOps, Jenkins giúp tối ưu hóa từng giai đoạn của pipeline CI/CD, bao gồm:

- Tích hợp mã nguồn liên tục: Jenkins tự động hóa việc tích hợp mã nguồn từ nhiều lập trình viên, giúp giảm thiểu lỗi và xung đột.
- Tự động hóa kiểm thử: Jenkins có thể tích hợp với các công cụ kiểm thử để đảm bảo rằng mọi thay đổi mã nguồn đều được kiểm tra kỹ lưỡng trước khi triển khai.
- Triển khai liên tục: Jenkins hỗ trợ triển khai tự động lên môi trường production, giảm thời gian đưa sản phẩm ra thị trường.

Docker trong DevOps

Docker, với khả năng container hóa, cung cấp môi trường triển khai đồng nhất và linh hoạt. Trong DevOps, Docker giúp:

- Đồng nhất môi trường phát triển và triển khai: Docker đảm bảo ứng dụng hoạt động ổn định từ giai đoạn phát triển đến môi trường production.
- Tăng cường tính linh hoạt: Docker cho phép dễ dàng đóng gói và triển khai các ứng dụng trên nhiều nền tảng khác nhau.
- Tích hợp với các công cụ DevOps: Docker có thể kết hợp với Jenkins và các công cụ khác để xây dựng các pipeline DevOps hiệu quả hơn.

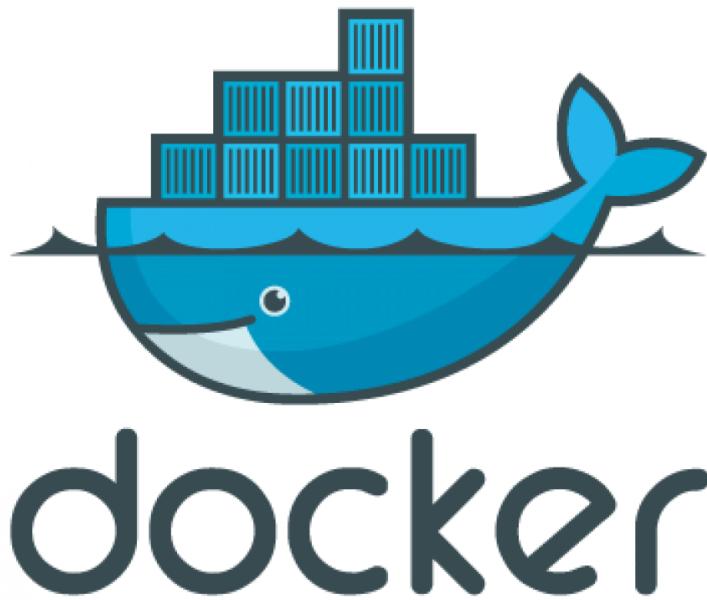
Lợi ích của DevOps với Jenkins CI/CD và Docker

- Tăng tốc độ phát triển và triển khai: Jenkins CI/CD tự động hóa các quy trình, trong khi Docker đảm bảo môi trường triển khai nhất quán, giúp tiết kiệm thời gian.
- Cải thiện chất lượng phần mềm: Sự kết hợp của Jenkins và Docker giúp giảm thiểu lỗi, tăng cường kiểm thử và đảm bảo chất lượng sản phẩm.
- Giảm chi phí và rủi ro: Tự động hóa kiểm thử và triển khai giúp giảm chi phí vận hành, đồng thời hạn chế rủi ro trong sản xuất.
- Hỗ trợ chuyển đổi số: Các tổ chức dễ dàng áp dụng công nghệ container hóa và tự động hóa để nhanh chóng thích nghi với thị trường.

III. Các công cụ và công nghệ sử dụng trong quy trình

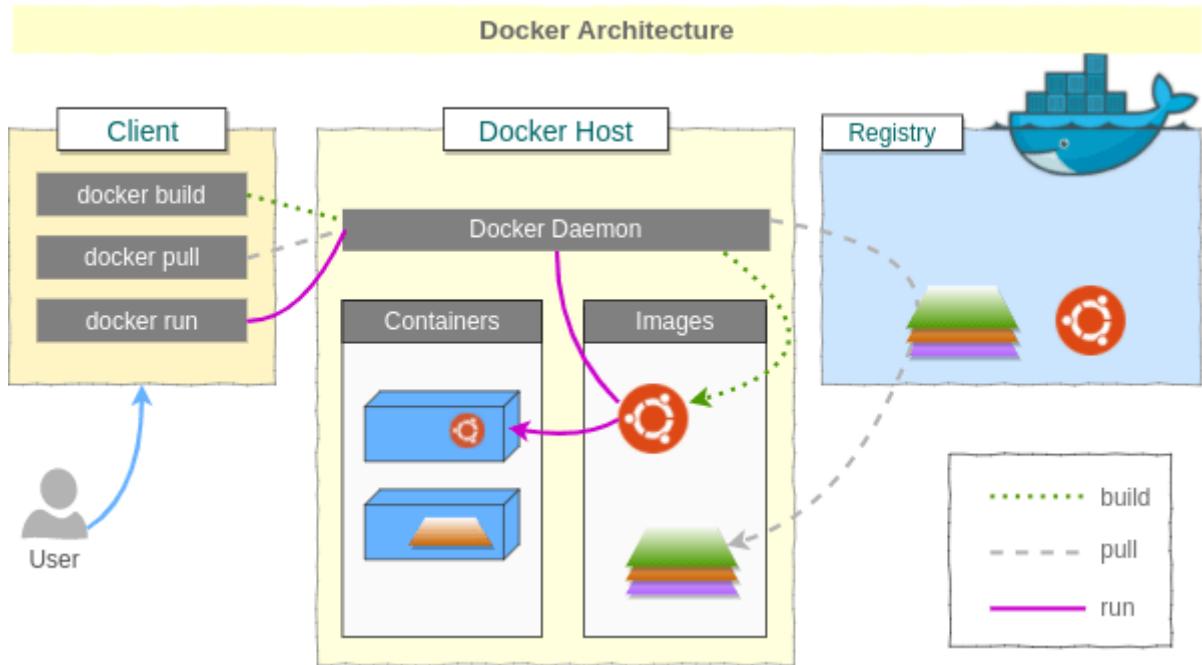
1 Docker

1.1 Khái niệm về Docker và containerization



Hình 6: Docker

Docker là một nền tảng cho developers và sysadmin để develop, deploy và run application với container. Nó cho phép tạo các môi trường độc lập và tách biệt để khởi chạy và phát triển ứng dụng và môi trường này được gọi là container. Khi cần deploy lên bất kỳ server nào chỉ cần run container của Docker thì application của chúng ta sẽ được khởi chạy ngay lập tức.



Hình 7: Kiến trúc của Docker

Các khái niệm liên quan

- Docker Client: là cách mà chúng ta tương tác với docker thông qua command trong terminal. Docker Client sẽ sử dụng API gửi lệnh tới Docker Daemon.
- Docker Daemon: là server Docker cho yêu cầu từ Docker API. Nó quản lý images, containers, networks và volume.
- Docker Volumes: là cách tốt nhất để lưu trữ dữ liệu liên tục cho việc sử dụng và tạo apps.
- Docker Registry: là nơi lưu trữ riêng của Docker Images. Images được push vào registry và client sẽ pull images từ registry. Có thể sử dụng registry của riêng bạn hoặc registry của nhà cung cấp như : AWS, Google Cloud, Microsoft Azure.
- Docker Hub: là Registry lớn nhất của Docker Images (mặc định). Có thể tìm thấy images và lưu trữ images của riêng mình trên Docker Hub.
- Docker Repository: là tập hợp các Docker Images cùng tên nhưng khác tags.
- Docker Networking: cho phép kết nối các container lại với nhau. Kết nối này có thể trên 1 host hoặc nhiều host.
- Docker Compose: là công cụ cho phép run app với nhiều Docker containers 1 cách dễ dàng hơn. Docker Compose cho phép chúng ta config các command trong file docker-compose.yml để sử dụng lại.
- Docker Swarm: để phối hợp triển khai container.
- Docker Services: là các containers trong production. 1 service chỉ run 1 image nhưng nó mã hoá cách thức để run image — sử dụng port nào, bao nhiêu bản sao container run để service có hiệu năng cần thiết và ngay lập tức.

Containerization là một công nghệ ảo hoá, giúp giải quyết vấn đề này bằng cách đóng gói ứng dụng và tất cả các phụ thuộc (dependencies) của nó (như thư viện, cấu hình) vào một đơn vị độc lập gọi là container. Container này đảm bảo rằng phần mềm sẽ

chạy ổn định và nhất quán trên bất kỳ môi trường nào, dù là trên máy tính cá nhân của Developer, Test Server, hay môi trường Production trên cloud.

1.2 Tối ưu docker image

Khi xây dựng Docker image, việc tối ưu hóa Dockerfile đóng vai trò quan trọng trong việc giảm dung lượng image và cải thiện hiệu suất.

- Lựa chọn base image nhỏ gọn: Ưu tiên sử dụng các base image nhỏ như Alpine thay vì các image lớn như Ubuntu, giúp giảm kích thước container đáng kể.
- Tối ưu hóa số lượng layer: Mỗi lệnh trong Dockerfile như RUN, COPY, hoặc ADD sẽ tạo ra một layer mới trong Docker image. Do đó, việc kết hợp các lệnh lại với nhau là cách hiệu quả để giảm số lượng layer.
- Tận dụng caching hiệu quả: Docker lưu trữ cache của các layer đã được tạo trước đó, giúp tăng tốc quá trình build khi không có thay đổi. Sắp xếp các lệnh trong Dockerfile theo thứ tự ít thay đổi trước sẽ tối ưu hóa khả năng tái sử dụng cache.

Ngoài ra, bảo mật cũng là một yếu tố không thể bỏ qua khi sử dụng Docker, vì container có thể trở thành mục tiêu của các cuộc tấn công nếu không được quản lý cẩn thận.

- Cập nhật định kỳ: Đảm bảo các Docker image luôn được cập nhật thường xuyên để xử lý các lỗ hổng bảo mật tiềm ẩn.
- Hạn chế quyền truy cập: Tránh chạy container với quyền root. Thay vào đó, hãy cấu hình để container hoạt động với một user có quyền hạn thấp nhằm giảm thiểu rủi ro bảo mật.
- Quản lý thông tin nhạy cảm: Sử dụng tính năng Docker secrets để lưu trữ các thông tin quan trọng như mật khẩu hoặc API key một cách an toàn.

1.3 Docker trong quy trình CI/CD

Docker đóng vai trò quan trọng trong quy trình CI/CD (Continuous Integration/Continuous Deployment) nhờ khả năng đóng gói ứng dụng và môi trường chạy vào các container nhất quán.

Docker trong Continuous Integration (CI)

- Đóng gói ứng dụng vào container:

Docker cho phép đóng gói ứng dụng cùng với tất cả các phụ thuộc vào một container. Điều này đảm bảo ứng dụng luôn chạy ổn định trên mọi môi trường (dev, test, production).

- Tích hợp với các công cụ CI:

Docker tích hợp dễ dàng với các công cụ CI như Jenkins, GitLab CI/CD, Travis CI, hoặc CircleCI để build và test tự động.

- Parallel Testing:

Sử dụng container để chạy các test case song song trên nhiều môi trường hoặc cấu hình khác nhau, giảm thời gian kiểm thử.

Docker trong Continuous Deployment (CD):

- Build và đẩy image lên registry:

Sau khi kiểm thử thành công, Docker image được build và push lên container registry (như Docker Hub, AWS ECR, hoặc Azure Container Registry). Điều này cho phép các môi trường khác nhau truy cập và sử dụng image mới nhất.

- Triển khai container tự động:

Các công cụ triển khai như Kubernetes, Docker Swarm, hoặc các nền tảng CI/CD (GitLab, Jenkins) có thể tự động triển khai container mới từ image được cập nhật.

- Rollback dễ dàng:

Docker giúp dễ dàng quay lại phiên bản trước bằng cách sử dụng các tag cụ thể hoặc image được lưu trữ trước đó trong registry.

Lợi ích của Docker trong CI/CD:

- Nhát quán trong môi trường:

Docker đảm bảo rằng ứng dụng luôn chạy trong cùng một môi trường từ giai đoạn phát triển, kiểm thử đến sản xuất. Nhờ việc đóng gói toàn bộ phụ thuộc, thư viện và cấu hình hệ điều hành trong container, chúng ta không còn phải lo lắng về sự khác biệt giữa các môi trường, giúp giảm thiểu lỗi do môi trường gây ra.

- Tăng tốc quá trình build và kiểm thử

Docker container khởi động nhanh hơn máy ảo, giúp tiết kiệm thời gian khi chạy các bước kiểm thử và triển khai. Ngoài ra, việc chạy các container song song để kiểm thử nhiều cấu hình hoặc môi trường khác nhau cũng giúp đẩy nhanh tốc độ kiểm thử.

- Dễ dàng đóng gói và tái sử dụng

Docker cho phép đóng gói ứng dụng cùng với toàn bộ phụ thuộc vào Docker image. Những image này có thể được chia sẻ và tái sử dụng trong các pipeline CI/CD hoặc trên các môi trường khác nhau, đảm bảo tính nhất quán và giảm thiểu lỗi phát sinh.

- Tối ưu hóa tài nguyên

Docker container sử dụng tài nguyên hệ thống một cách hiệu quả hơn so với máy ảo, giúp giảm chi phí hạ tầng trong quá trình phát triển, kiểm thử và triển

khai. Điều này đặc biệt hữu ích khi bạn phải xử lý khối lượng công việc lớn trên hạ tầng hạn chế.

- Bảo mật nâng cao

Docker cung cấp khả năng cách ly ứng dụng trong từng container, giảm nguy cơ ảnh hưởng lẫn nhau giữa các ứng dụng. Ngoài ra, Docker secrets hỗ trợ lưu trữ các thông tin nhạy cảm như mật khẩu, API key một cách an toàn, giúp nâng cao bảo mật cho hệ thống.

2. Gitlab, Gitflow

2.1 Khái niệm về Gitlab



Hình 8: Gitlab

GitLab là nền tảng DevOps tích hợp, cung cấp một loạt công cụ hỗ trợ quá trình phát triển phần mềm từ giai đoạn lập kế hoạch, phát triển mã nguồn, kiểm thử, triển khai cho đến bảo trì. Nhờ các tính năng như quản lý mã nguồn (Git), theo dõi vấn đề, CI/CD và nhiều công cụ hỗ trợ khác, GitLab giúp các nhóm phát triển phần mềm làm việc hiệu quả và liền mạch.

GitLab là hệ thống quản lý mã nguồn phân tán, sử dụng Git, giúp các lập trình viên dễ dàng quản lý mã, hợp tác và theo dõi các thay đổi trong dự án. Ngoài việc quản lý mã nguồn, GitLab còn cung cấp các tính năng hỗ trợ quy trình DevOps, nổi bật là GitLab CI/CD, cho phép tự động hóa các công việc như kiểm thử, xây dựng, triển khai và giám sát ứng dụng.

GitLab tích hợp tất cả các công cụ DevOps trong một nền tảng duy nhất, giúp các nhóm phát triển phần mềm làm việc hiệu quả hơn với quy trình CI/CD (Continuous

Integration và Continuous Deployment). Quy trình này giúp tự động kiểm tra mã nguồn, xây dựng ứng dụng và triển khai sản phẩm lên các môi trường.

- Continuous Integration (CI): Là quá trình tự động tích hợp các thay đổi mã nguồn vào một repository chung, giúp GitLab CI kiểm tra và xây dựng ứng dụng mỗi khi có thay đổi mới từ lập trình viên.
- Continuous Delivery/Deployment (CD): Sau khi mã được tích hợp, GitLab tự động triển khai ứng dụng lên môi trường kiểm thử, staging hoặc sản xuất mà không cần sự can thiệp thủ công.

Với GitLab, việc xây dựng, kiểm thử và triển khai ứng dụng trở nên đơn giản và tự động, giúp tăng năng suất và tính ổn định của quá trình phát triển phần mềm.

2.2 Gitlab CI/CD



Hình 9: Gitlab CI/CD

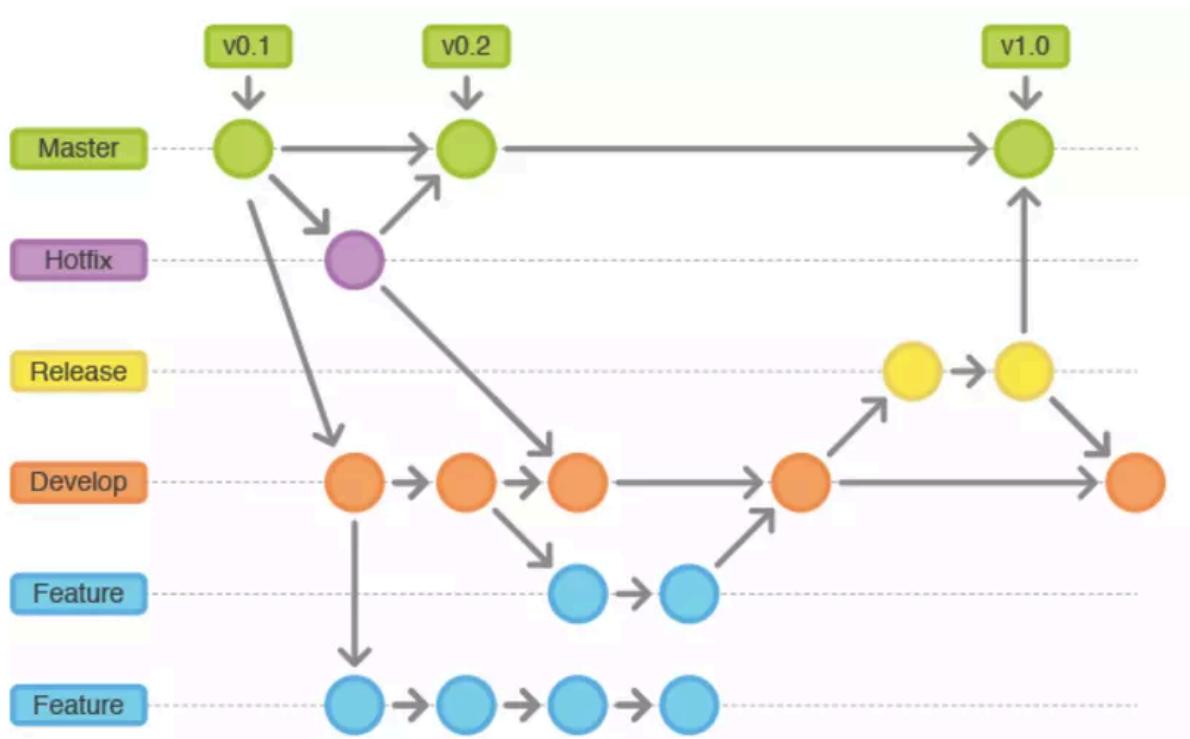
GitLab CI/CD (Continuous Integration and Continuous Deployment/Delivery) là một phần của GitLab, một nền tảng DevOps mã nguồn mở tích hợp cung cấp các công cụ để quản lý toàn bộ vòng đời của một dự án phần mềm (software development life cycle). GitLab CI/CD giúp tự động hóa quy trình xây dựng (Build), kiểm thử (Test) và triển khai mã nguồn (Deploy), giúp cải thiện chất lượng phần mềm và tăng cường hiệu quả làm việc.

Các thành phần chính của GitLab CICD

- Pipeline: Là một tập hợp các công việc được thực thi theo một thứ tự xác định. Mỗi pipeline có thể có nhiều giai đoạn, và mỗi giai đoạn có thể có nhiều công việc.
- Job: Là một đơn vị công việc cụ thể, ví dụ như biên dịch mã nguồn, chạy kiểm thử, hoặc triển khai ứng dụng. Các công việc trong cùng một giai đoạn được thực thi song song, còn các giai đoạn được thực thi tuần tự.
- Runner: Là các agent chịu trách nhiệm thực thi các công việc. GitLab Runner có thể được cài đặt trên máy chủ riêng hoặc sử dụng các runners được cung cấp bởi GitLab.
- .gitlab-ci.yml: Là tệp cấu hình được lưu trữ trong kho mã nguồn (repository) của bạn. Tệp này xác định các pipelines, stages và jobs cần thực thi. Đây là nơi chúng ta định nghĩa logic CI/CD cho dự án của mình.

2.3 Gitflow

Gitflow là khái niệm chỉ cách phân nhánh và phối hợp phát triển (development), phát hành (release) tính năng bằng cách sử dụng git.



Hotfix: Được base trên nhánh master để sửa nhanh những lỗi trên UIT hoặc sửa những cấu hình đặc biệt chỉ có trên môi trường productions.

Release: Trước khi Release một phần mềm dev team cần được tạo ra để kiểm tra lại lần cuối trước đi release sản phẩm để người dùng có thể sử dụng (Thông thường mã nguồn tại thời điểm này sẽ tạo ra bản build để test và kiểm tra lại business).

Develop: Được khởi tạo từ master branches để lưu lại tất cả lịch sử thay đổi của mã nguồn. Develop branch là merge code của tất cả các branches feature.

Feature: Được base trên branches Develop. Mỗi khi phát triển một feature mới chúng ta cần tạo một branches để việt mã nguồn cho từng feature.

3. Kubernetes

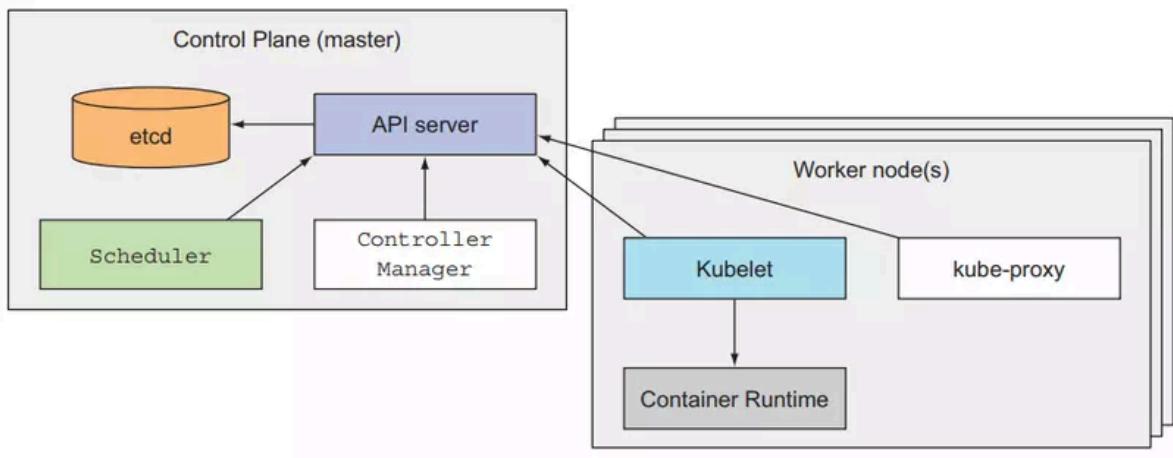
3.1 Tổng quan về Kubernetes



kubernetes

Hình 11: Kubernetes

Kubernetes là một nền tảng nguồn mở, khả chuyển, có thể mở rộng để quản lý các ứng dụng được đóng gói và các service, giúp thuận lợi trong việc cấu hình và tự động hóa việc triển khai ứng dụng. Kubernetes là một hệ sinh thái lớn và phát triển nhanh chóng. Các dịch vụ, sự hỗ trợ và công cụ có sẵn rộng rãi.



Hình 12: Kiến trúc của Kubernetes

Kubernetes cluster (một cụm bao gồm một master và một hoặc nhiều worker) bao gồm 2 thành phần (component) chính:

- Master nodes (control plane)
- Worker nodes

Master nodes bao gồm 4 thành phần chính là API server, controller manager, Scheduler, Etcd:

- API server: thành phần chính để giao tiếp với các thành phần khác
- Controller manager: gồm nhiều controller riêng cụ thể cho từng resource và thực hiện các chức năng cụ thể cho từng thằng resource trong kube như create pod, create deployment, v...v...
- Scheduler: schedules ứng dụng tới node nào
- Etcd: là một database để lưu giữ trạng thái và resource của cluster

Ứng dụng của chúng ta sẽ được chạy trên worker node. Worker node gồm 3 thành phần chính:

- Container runtime (docker, rkt hoặc nền tảng khác): chạy container
- Kubelet: giao tiếp với API server và quản lý container trong một worker node
- Kubernetes Service Proxy (kube-proxy): quản lý network và traffic của các ứng dụng trong worker node

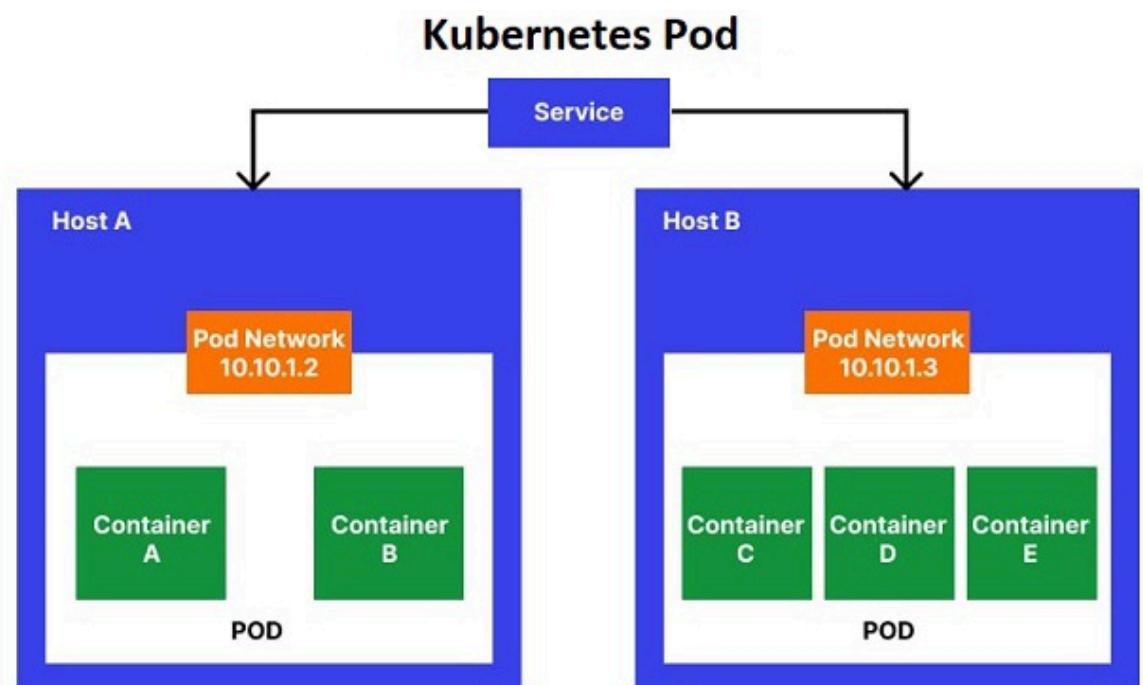
Kubernetes giúp chúng ta quản lý container trên một cụm máy tính (cluster). Trong một cluster Kubernetes, bạn có thể triển khai các ứng dụng phức tạp, quản lý số lượng

lớn container, đảm bảo ứng dụng hoạt động một cách ổn định và có thể mở rộng linh hoạt.

3.2 Các thành phần chính của Kubernetes

Pod:

- Pod là đơn vị cơ bản nhất trong Kubernetes, đại diện cho một hoặc nhiều container được chạy cùng nhau trên một máy chủ. Các container trong cùng một pod chia sẻ mạng, storage và có thể giao tiếp với nhau một cách trực tiếp.
- Một pod có thể chứa một hoặc nhiều container, và các container trong một pod thường làm việc với nhau để thực hiện một chức năng chung.

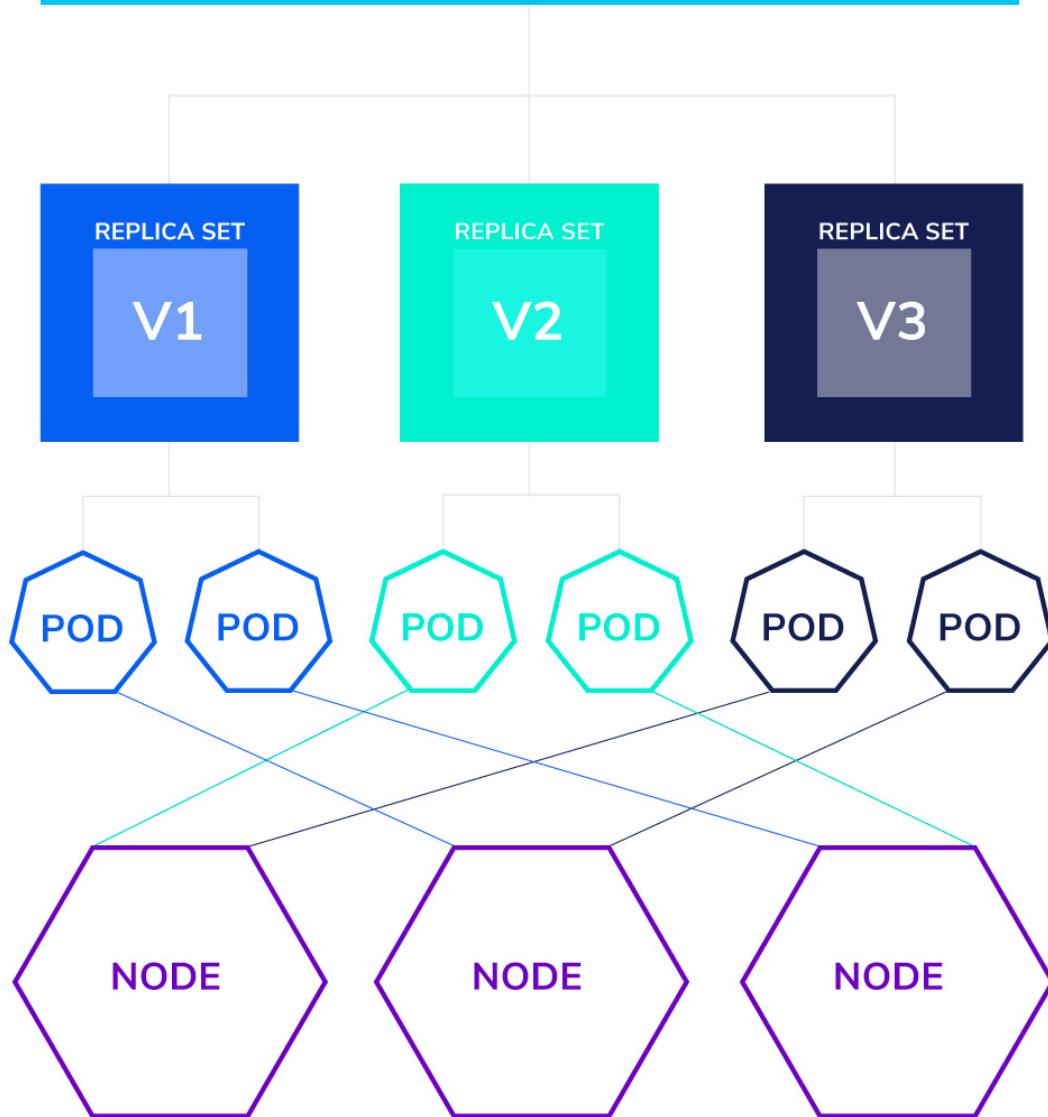


Hình 13: Kubernetes Pod

Deployment

- Deployment giúp quản lý và triển khai các pod. Nó đảm bảo rằng một số lượng pod nhất định luôn được chạy và sẵn sàng tiếp nhận lưu lượng truy cập.
- Deployment cũng hỗ trợ việc cập nhật ứng dụng một cách mượt mà, đảm bảo rằng các phiên bản mới không gây gián đoạn dịch vụ.

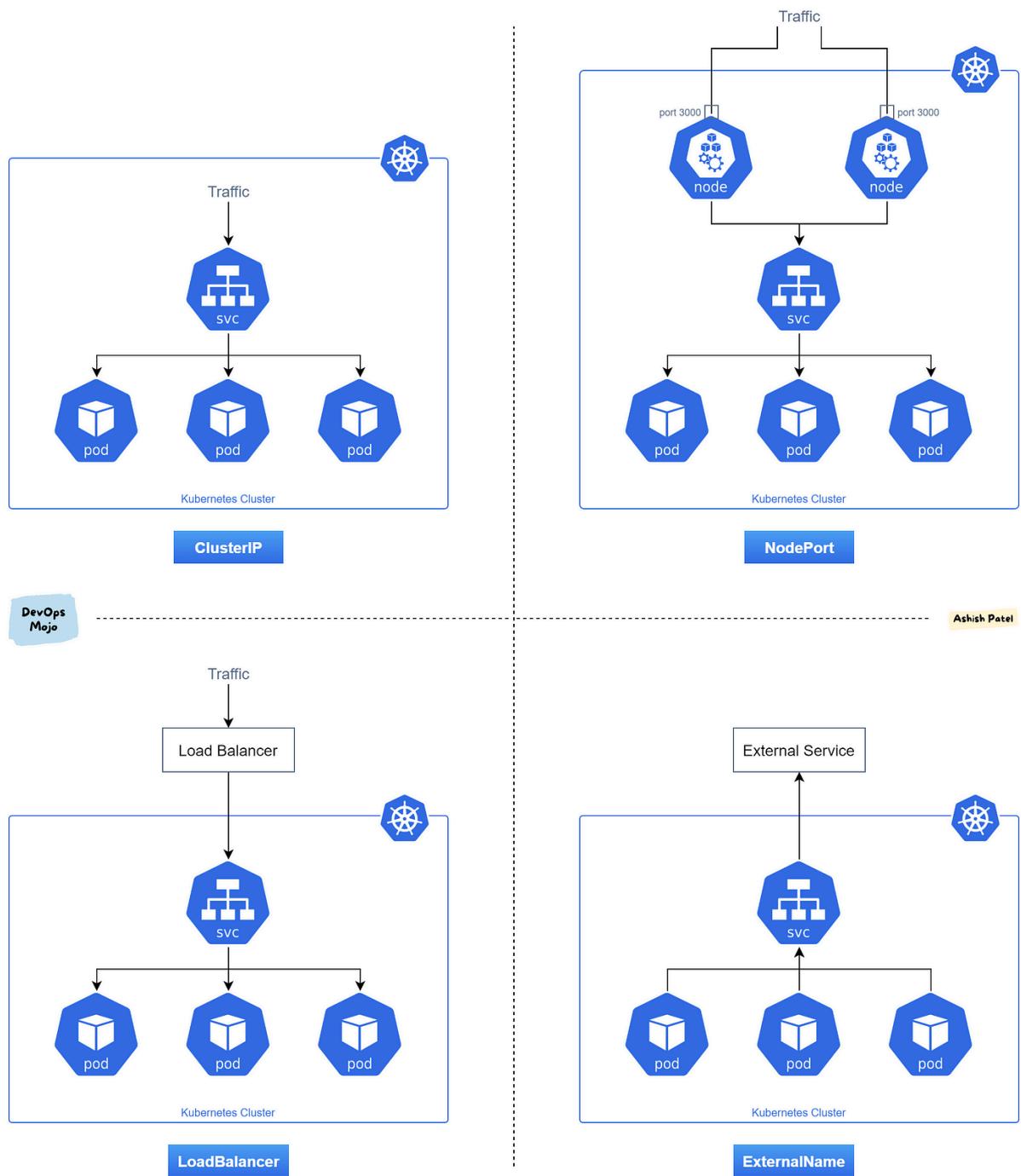
DEPLOYMENT



Hình 14: Kubernetes Deployment

Service

- Service là một đối tượng trong Kubernetes để định nghĩa cách các pod giao tiếp với nhau. Service giúp bạn truy cập vào các pod mà không cần phải biết địa chỉ IP cụ thể của chúng.
- Kubernetes cung cấp nhiều loại Service như ClusterIP (mặc định, cho phép truy cập trong nội bộ cluster), NodePort (truy cập từ bên ngoài cluster) và LoadBalancer (sử dụng Load Balancer từ cloud provider).

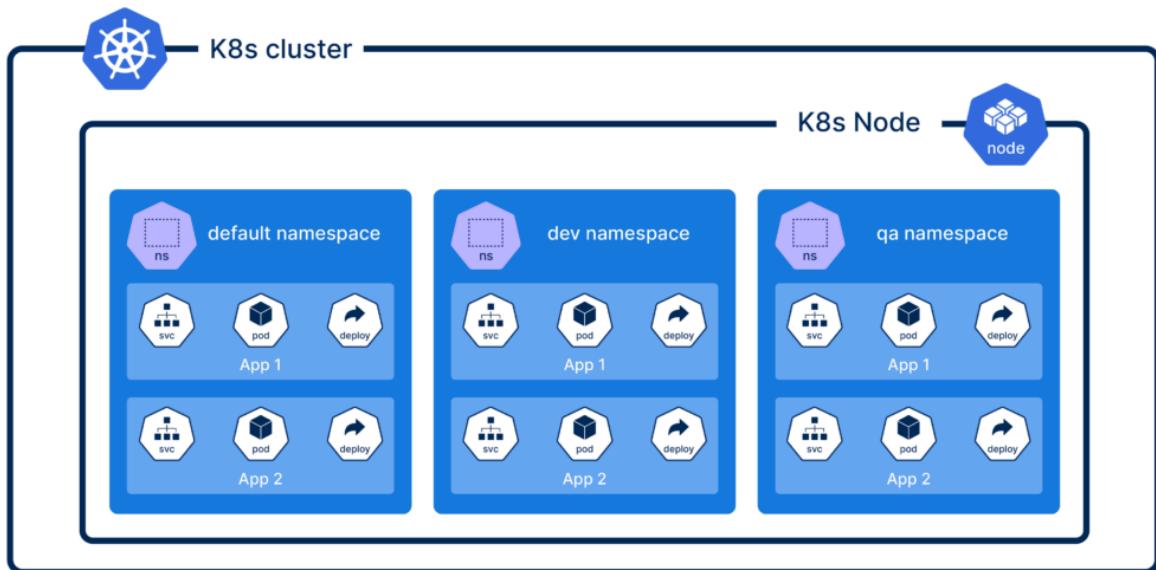


Hình 15: Kubernetes Service

Namespace

- Namespace là một cách để phân tách các tài nguyên trong Kubernetes thành các không gian khác nhau. Điều này rất hữu ích khi bạn có nhiều môi trường (như development, staging, production) hoặc nhiều nhóm làm việc khác nhau trong cùng một cluster.

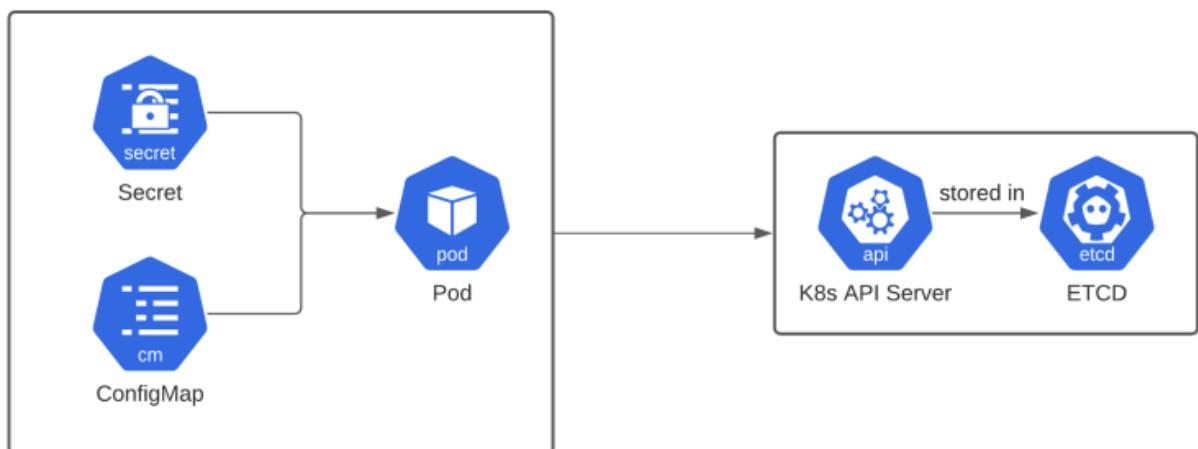
Kubernetes - Namespaces



Hình 16: Kubernetes Namespaces

ConfigMap và Secret

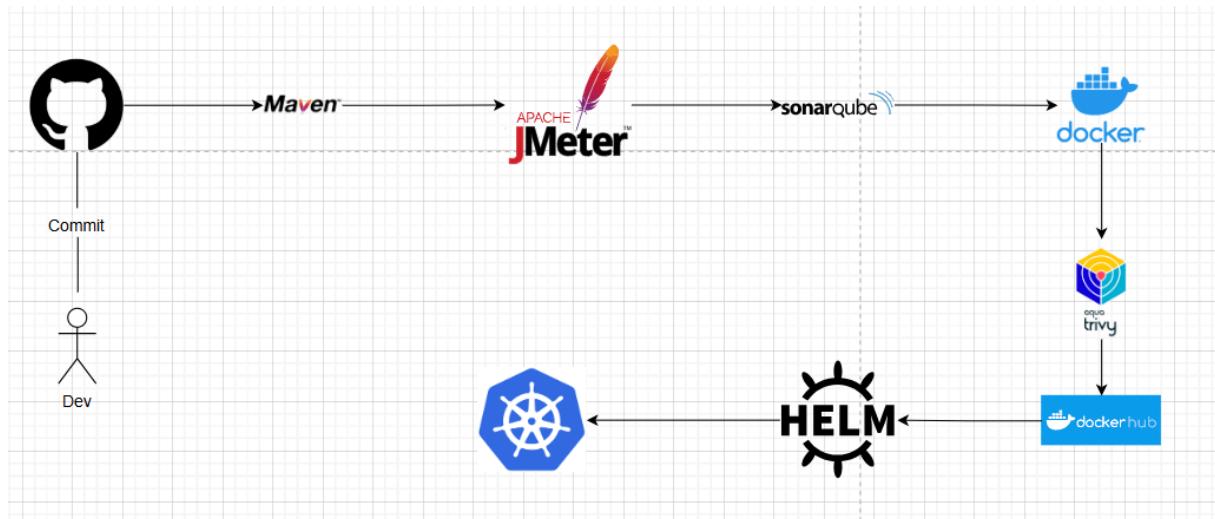
- ConfigMap cho phép bạn lưu trữ các cấu hình dưới dạng các key-value pairs mà không cần phải xây dựng lại image khi thay đổi cấu hình.
- Secret là đối tượng bảo mật giúp lưu trữ các thông tin nhạy cảm như mật khẩu, token hoặc chứng chỉ SSL. Secret được mã hóa và có thể sử dụng trong pod mà không cần phải đưa các thông tin nhạy cảm vào trong mã nguồn.



Hình 17: Kubernetes ConfigMap và Secret

IV. Các kiến trúc xây dựng trong xây dựng CICD pipeline với Jenkins

1. Sơ đồ kiến trúc sử dụng các công cụ

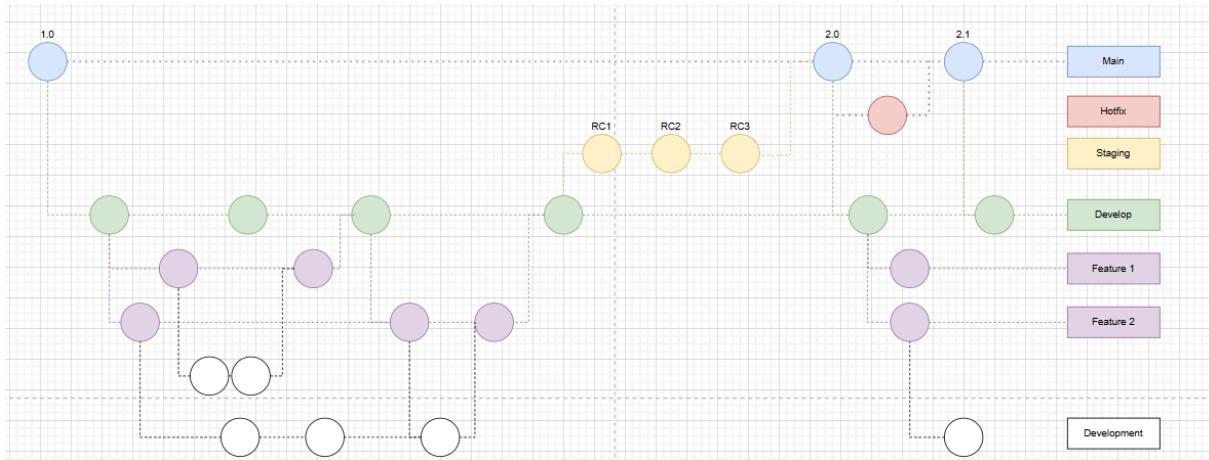


Hình 18: Sơ đồ các công cụ sử dụng

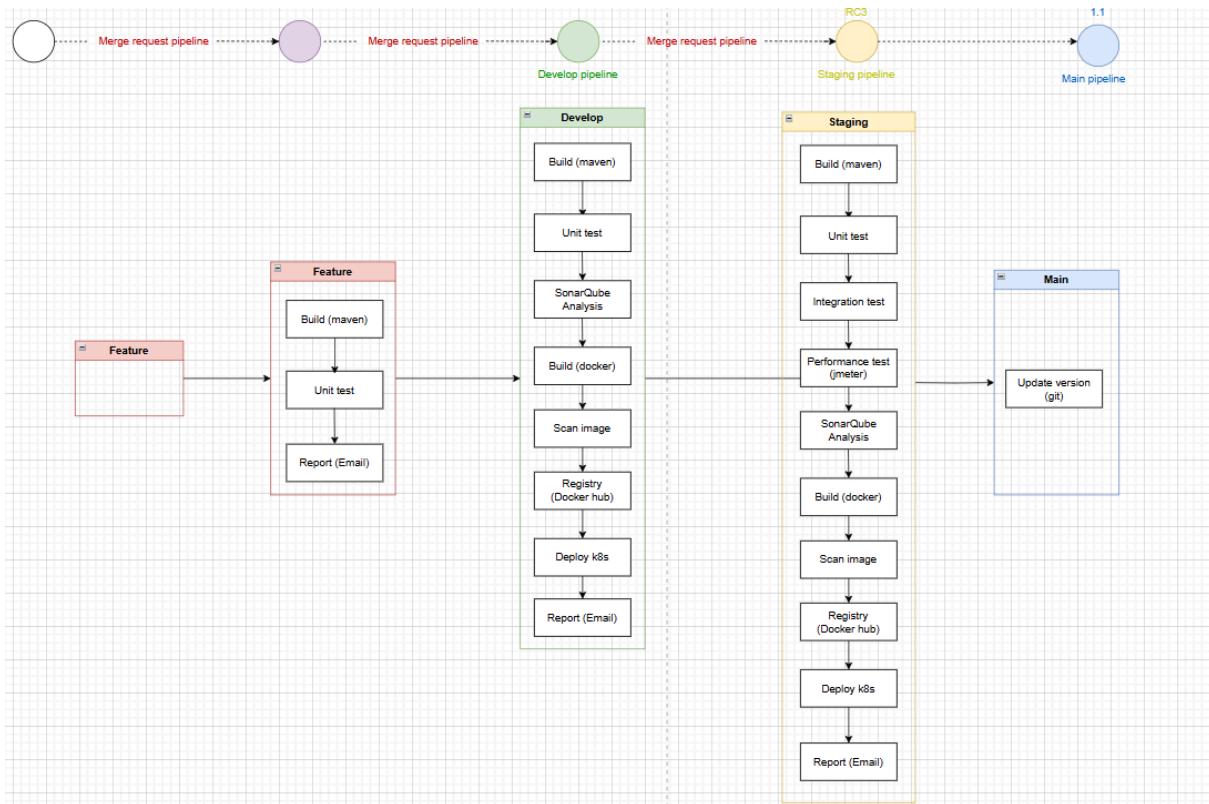
Quy trình thực hiện:

- Pipeline được kích hoạt khi mà Dev commit lên Github
- Code sẽ được chạy vào file packet cũng như run các loại test
- Sau đó chạy tới performance test và sử dụng JMeter
- Sau đó code sẽ được scan qua sonarqube để kiểm tra các lỗi trong code
- Code sẽ được đóng gói thành docker container
- Sau đó được trivy scan để kiểm tra bảo mật của image
- Image sẽ được push lên dockerhub
- Cuối cùng sử dụng Helm để upgrade job

2. Sơ đồ pipeline trong gitflow



Hình 19: Sơ đồ quy trình Gitflow



Hình 20: Chi tiết sơ đồ quy trình Gitflow

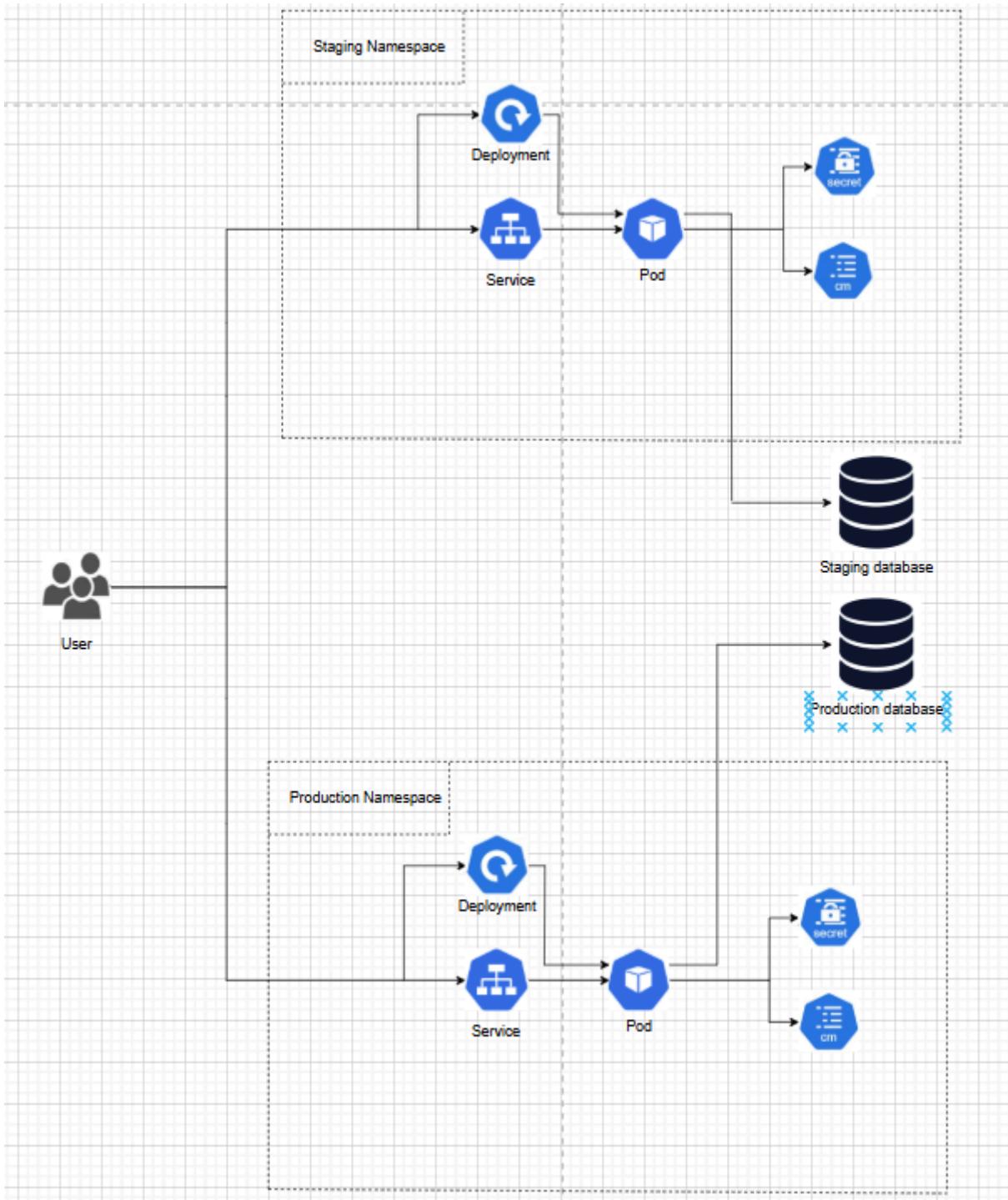
Chi tiết về pipeline được thực hiện như sau:

- Giữa các branch feature 1 và feature 2, khi có commit giữa các branch này với nhau thì code sẽ được build sử dụng maven và chạy unit test
- Sau khi feature thành công hoạt động và merge vào developer thì code sẽ được build và chạy unit test. Sau đó sẽ chạy SonarQube để scan code. Build docker sau đó scan image bằng cách sử dụng Trivy. Sau đó push lên dockerhub, được

deploy lên môi trường Staging. Toàn bộ kết quả sẽ được report kết quả của toàn bộ kết quả cho developer.

- Khi qua môi trường Staging thì tương tự code sẽ được build bởi maven sau đó chạy unit test và integration test, sau đó sẽ được chạy performance test bởi JMeter. Sử dụng SonarQube Analyst để phân tích source code, sau đó thì code sẽ được đóng gói thành docker image. Scan image bằng cách sử dụng Trivy sau đó push lên Docker hub. Sau đó deploy lên k8s, khác với môi trường develop thì môi trường Staging sẽ được deploy bằng tay.

3. Kiến trúc của k8s:



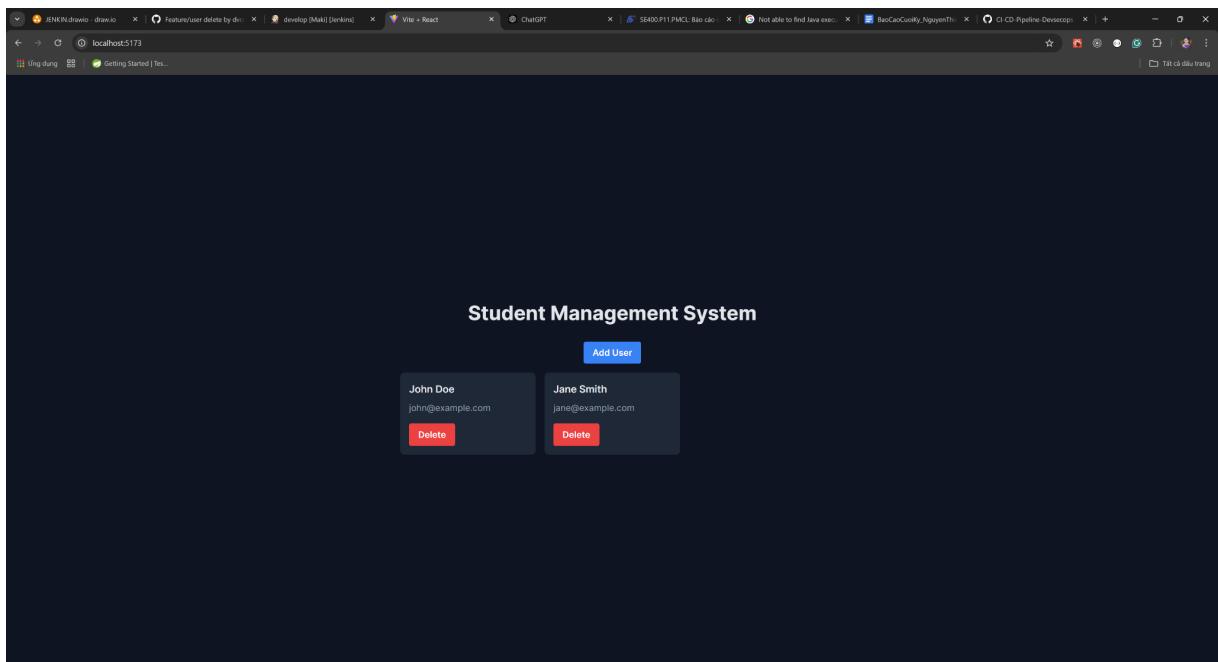
Hình 21: Kiến trúc k8s

Trong môi trường K8s thì sẽ có các môi trường như môi trường Staging và môi trường Production. Mỗi môi trường sẽ được phân biệt giữa các namespace khác nhau.

Các pod của ứng dụng sẽ sử dụng deployment và service. Mỗi môi trường sẽ có các database riêng

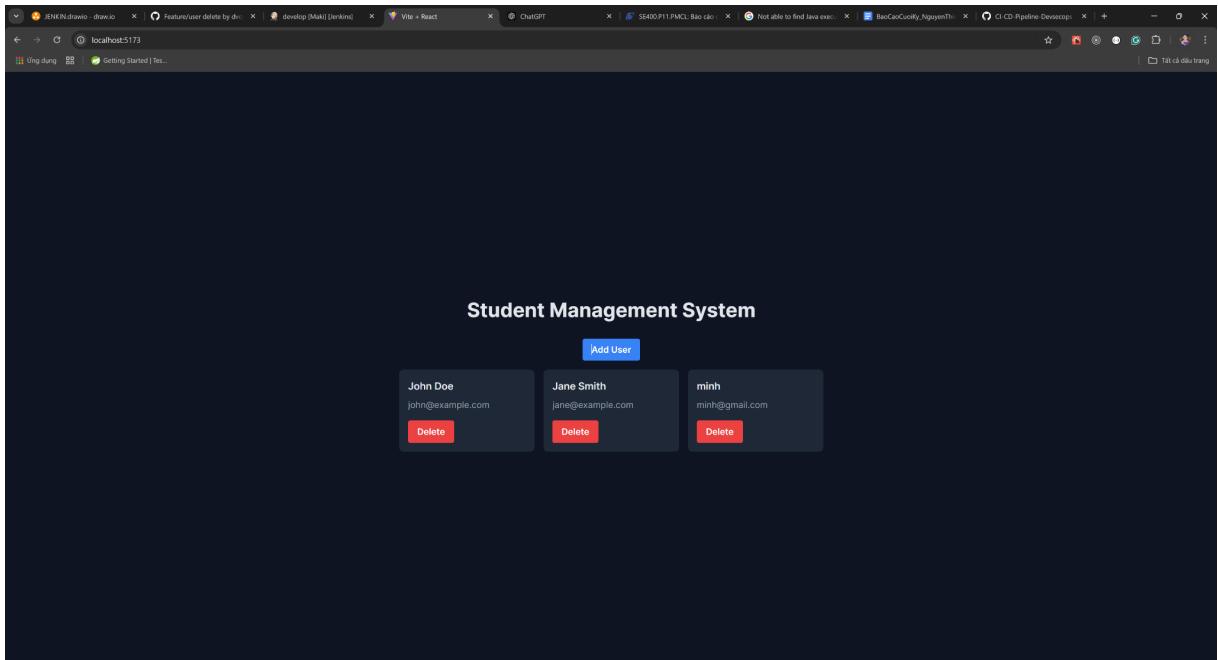
V. Demo ứng dụng

1. Ứng dụng demo với Jenkins CICD pipeline



Hình 22: Ứng dụng demo

Ứng dụng demo với mục đích trình bày tính năng delete học sinh được tự động đưa lên môi trường production với Jenkins CICD pipeline.



Hình 23: Hiện trạng ban đầu của Ứng dụng

Hiện trạng ban đầu của ứng dụng có thể hoạt động bình thường với các chức năng như xem danh sách học sinh, thêm học sinh. Tuy nhiên chức năng xóa học sinh vẫn chưa phát triển nên hiện tại ứng dụng chưa thể thực hiện chức năng này

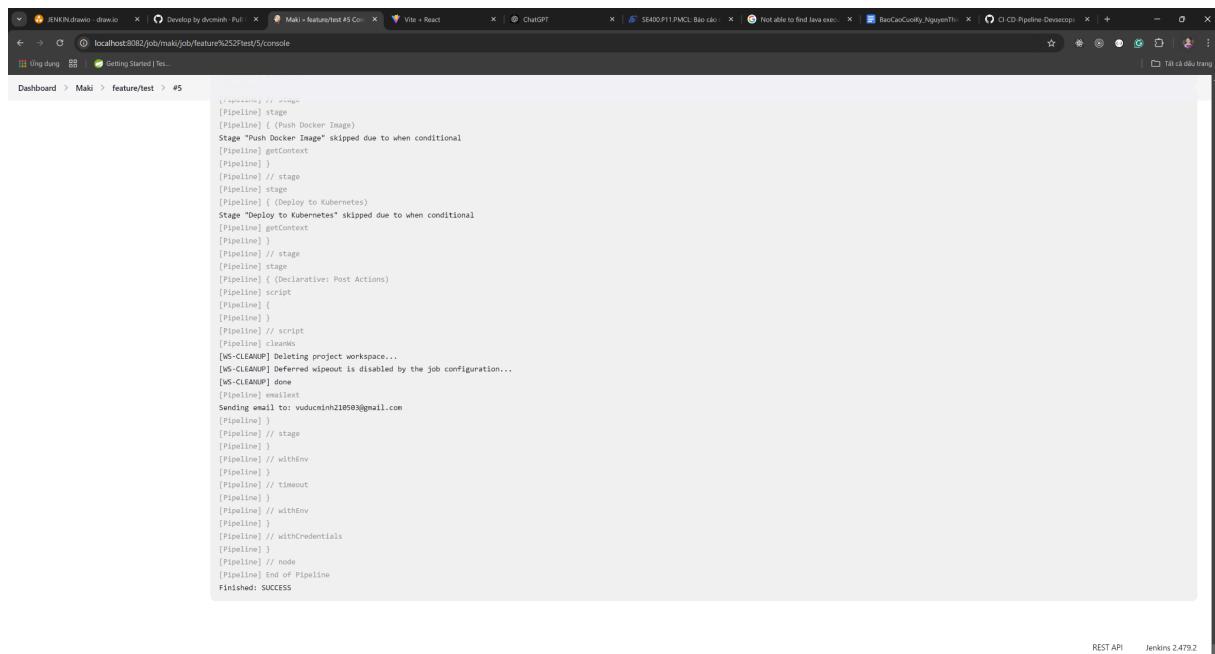
VI. Quy trình thực hiện

1. Commit code tính năng

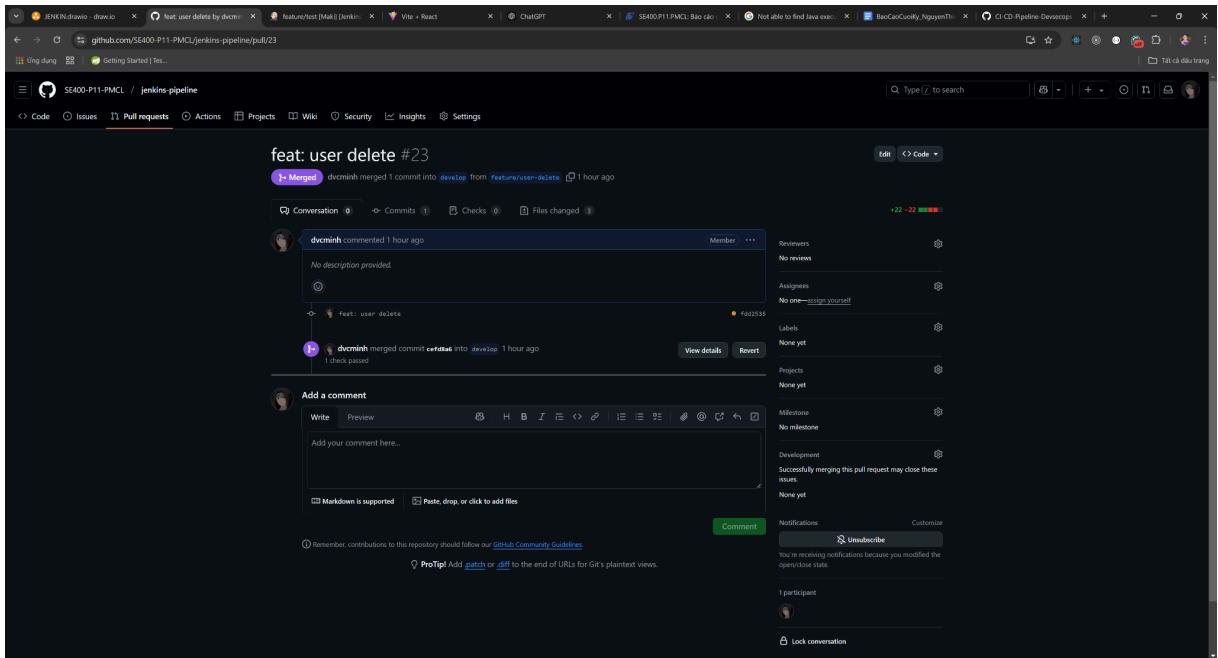
```
jenkins_key.pub
jmeter.log

PS D:\Workspace\Reference\cicd> git commit -m "feat: user delete"
[feature/user-delete fdd2535] feat: user delete
  3 files changed, 22 insertions(+), 22 deletions(-)
PS D:\Workspace\Reference\cicd> git push
Enumerating objects: 32, done.
Counting objects: 100% (32/32), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (18/18), 1.30 KiB | 1.30 MiB/s, done.
Total 18 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 5 local objects.
To https://github.com/SE400-P11-PMCL/jenkins-pipeline.git
  49d2c0e..fdd2535  feature/user-delete -> feature/user-delete
```

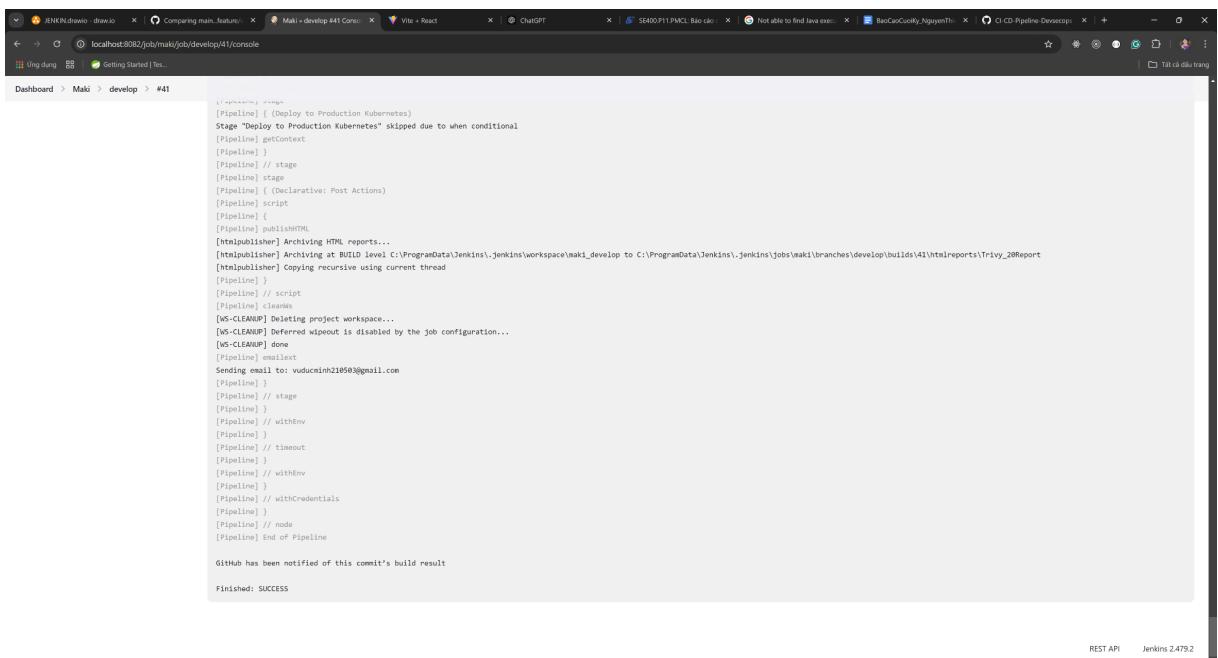
Sau khi đã phát triển xong tính năng, developer sẽ tiến hành commit code trên branch feature/user-delete



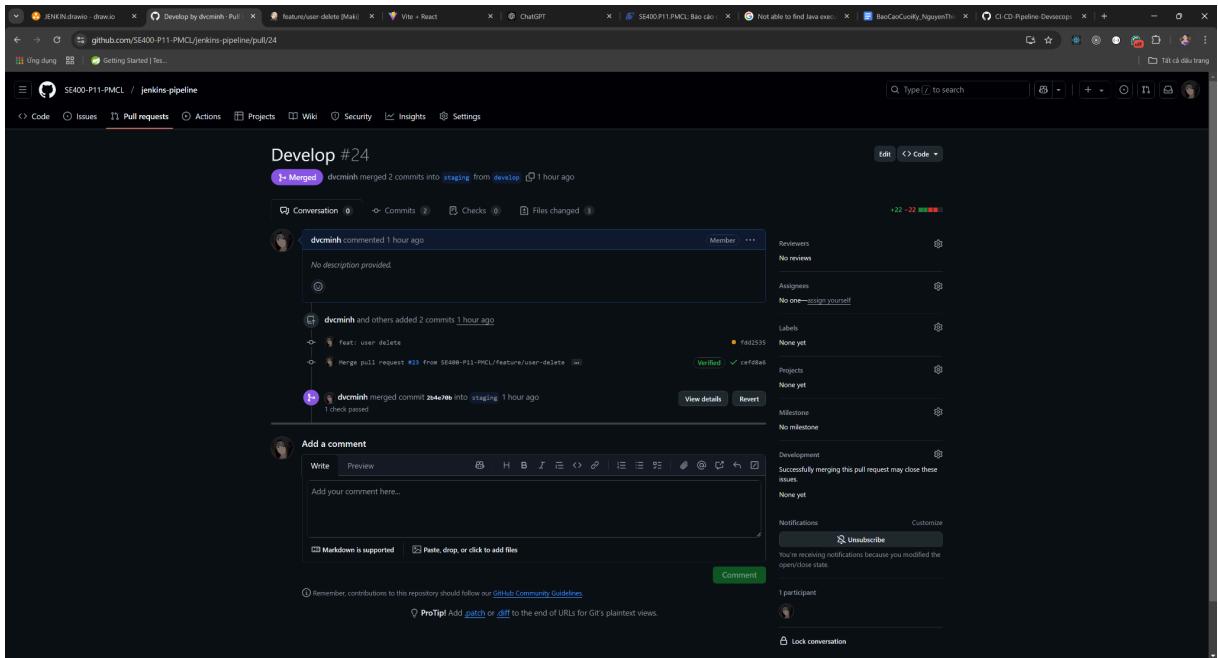
Pipeline của branch feature đã được kích hoạt để tự động build, test và kiểm tra code



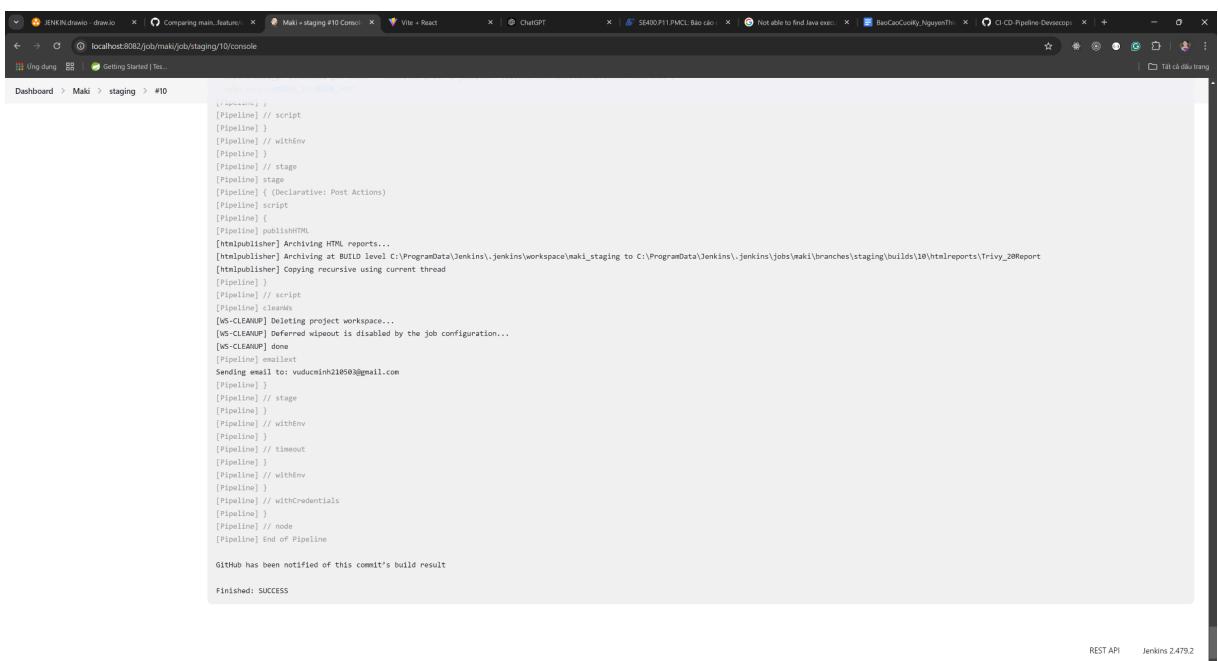
Sau khi ứng dụng trên branch develop được phát triển bởi developer thành công, ta sẽ tiến hành merge code lên staging để thực hiện các bước test xa hơn như integration test, performance test



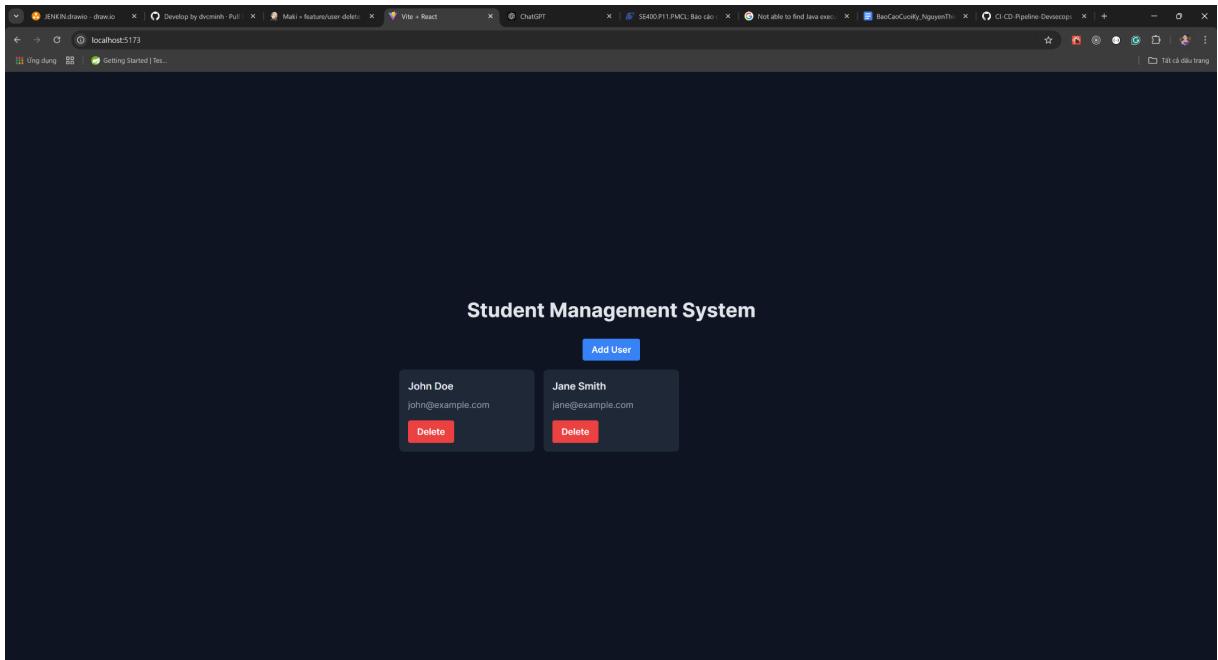
Hiện trạng pipeline đã được thực thi thành công và code đã được tự động đưa lên môi trường staging



Tiến hành tạo pull request để deploy tính năng lên môi trường production cũng như chạy các test khác



Hiện trạng pipeline đã được chạy các bài test tự động và bằng tay thành công, code sẽ được tiến hành deploy lên môi trường production



Lúc này ứng dụng trên production sau khi trải qua quá trình build, test và kiểm tra nghiêm ngặt đã có thể thực hiện chức năng xóa học sinh trên môi trường production

VII. 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.

VII. 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

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

Thêm bước kiểm tra bảo mật

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>

Link github: [**https://github.com/SE400-P11-PMCL**](https://github.com/SE400-P11-PMCL)