ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA TOÁN - TIN

o0o

ĐỒ ÁN II

NGHIÊN CỨU VÀ TRIỂN KHAI QUY TRÌNH CI/CD TRÊN NHIỀU NỀN TẢNG

Chuyên ngành: TOÁN TIN

Giảng viên hướng dẫn: **TS. NGÔ QUỐC HOÀN** _____
Chữ kí của GVHD

Sinh viên thực hiện: **TRẦN XUÂN SƠN**

MSSV: **20227149**

HÀ NỘI, 10/2025

ĐỒ ÁN II

Chuyên ngành: Toán Tin

NGHIÊN CỨU VÀ TRIỂN KHAI QUY TRÌNH CI/CD TRÊN NHIỀU NỀN TẢNG

Giảng viên hướng dẫn:	**TS. NGÔ QUỐC HOÀN**
Họ và tên sinh viên:	**TRẦN XUÂN SƠN**
Mã số sinh viên:	**20227149**
Lớp:	**Toán - Tin 02 - K67**

HÀ NỘI, 10/2025

Nhận xét của giảng viên hướng dẫn

1. Mục tiêu và nội dung của đồ án

(a) Mục tiêu:

(b) Nội dung:

2. Kết quả đạt được

3. Ý thức làm việc của sinh viên:

Hà Nội, ngày .... tháng .... năm ....

Giảng viên hướng dẫn

Lời cảm ơn

Mục lục

Danh sách bảng

Danh sách hình vẽ

Ký hiệu và chữ viết tắt

CI	Continuous Integration
CD	Continuous Delivery / Continuous Deployment
DevOps	Development Operations
MTTR	Mean Time To Resolution
RSA	Rivest-Shamir-Adleman
VMs	Virtual Machines

Tóm tắt nội dung

Mở đầu

Trong kỷ nguyên chuyển đổi số, nơi tốc độ phát triển và chất lượng sản phẩm là yếu tố sống còn, khả năng triển khai phần mềm nhanh chóng, liên tục và đáng tin cậy đã trở thành ưu tiên hàng đầu của mọi tổ chức. Phương pháp phát triển truyền thống với các quy trình tích hợp và phân phối thủ công không còn đáp ứng được yêu cầu về tốc độ thị trường. Chính vì vậy, Tích hợp Liên tục và Phân phối/Triển khai Liên tục (CI/CD) đã nổi lên như một giải pháp kỹ thuật cốt lõi.

Về mặt lịch sử, khái niệm Tích hợp Liên tục (CI) được giới thiệu lần đầu tiên bởi Grady Booch vào năm 1991, nhưng đã được phổ biến rộng rãi và đưa vào thực tiễn phát triển phần mềm hiện đại bởi Kent Beck vào cuối những năm 1990. Tiếp nối CI, khái niệm Phân phối Liên tục (Continuous Delivery – CD) đã được hệ thống hóa và làm rõ trong cuốn sách cùng tên (xuất bản năm 2010) của Jez Humble và David Farley, trở thành nền tảng quan trọng trong văn hóa DevOps ngày nay.

CI/CD tự động hóa toàn bộ vòng đời phát triển phần mềm, từ khi các lập trình viên tích hợp mã nguồn (CI) đến khi ứng dụng được chuyển giao hoặc triển khai tới môi trường sản xuất (CD). Tầm quan trọng của CI/CD được chứng minh rõ rệt qua các số liệu: theo khảo sát của GitLab (2024), các nhóm phát triển phần mềm áp dụng CI/CD cho thấy tỷ lệ thành công khi triển khai tăng 42% so với các nhóm không áp dụng phương pháp này, đồng thời giúp giảm thiểu đáng kể thời gian xử lý lỗi (MTTR), một chỉ số then chốt về khả năng bảo trì hệ thống. Báo cáo đồ án 2 sẽ đi sâu vào nghiên cứu và triển khai quy trình CI/CD trên nhiều nền tảng.

Nội dung đồ án 2 được tổ chức thành các chương sau:

- Chương 1. Giới thiệu chung: Trình bày tổng quát về CI/CD (Tích hợp và Phân

phối/Triển khai Liên tục), bao gồm khái niệm, ưu điểm, hạn chế, và cách thức hoạt động cơ bản của quy trình.

- Chương 2. Giới thiệu các nền tảng và công cụ triển khai CI/CD: Giới thiệu và so sánh các nền tảng CI/CD cốt lõi được sử dụng: GitHub, GitLab, và Jenkins. Đồng thời, giới thiệu các công cụ hỗ trợ cần thiết khác như VMware Workstation Pro, Nginx, và SonarQube.
- Chương 3. Xây dựng hệ thống server ảo: Trình bày chi tiết việc xây dựng môi trường server ảo (VMs) và mô tả Kiến trúc hệ thống cụ thể cho việc triển khai CI/CD trên từng nền tảng (GitHub, GitLab và Jenkins).
- Chương 4. Xây dựng và triển khai quy trình CI/CD: Tập trung vào các bước thực hiện trọng tâm: Xây dựng quy trình CI/CD cho ứng dụng mẫu và tiến hành triển khai quy trình trên các môi trường đã thiết lập.
- Chương 5. Xây dựng các trường hợp kiểm thử và trình bày kết quả: Trình bày việc xây dựng các trường hợp kiểm thử và phân tích kết quả kiểm thử tự động, đánh giá hiệu quả của quy trình CI/CD đa nền tảng.
- Kết luận và hướng nghiên cứu: Tổng kết kết quả đạt được và đề xuất hướng cải thiện hoặc mở rộng nghiên cứu trong tương lai.

Chương 1

Giới thiệu chung

1.1 Đặt vấn đề

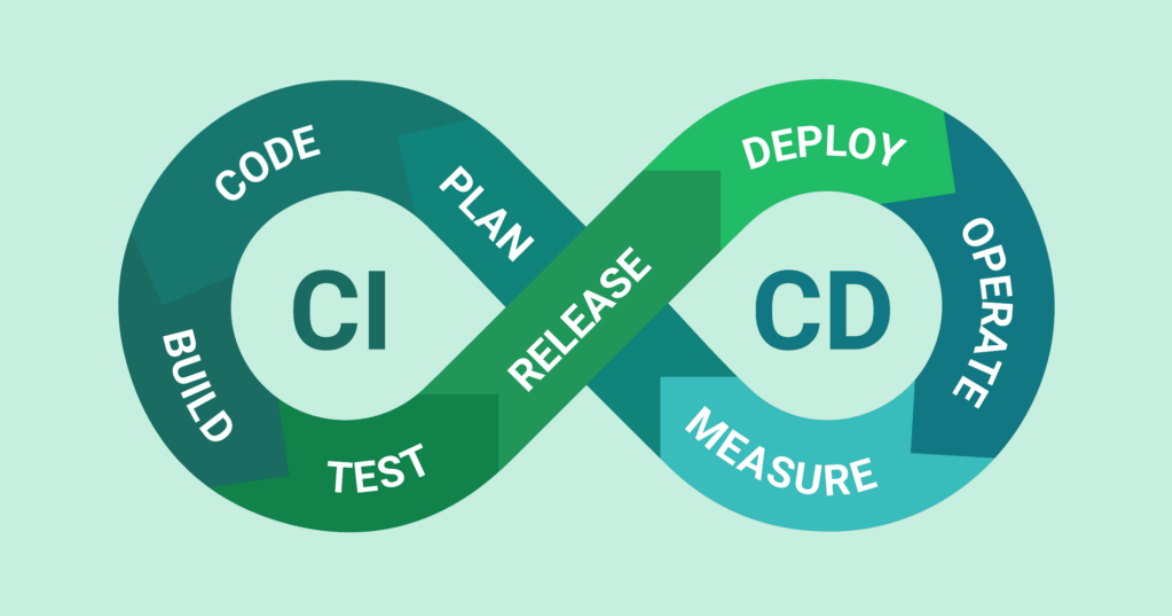
Trong các doanh nghiệp, các các lập trình viên chịu trách nhiệm viết code cho dự án, còn các kỹ sư vận hàng và triển khai hạ tầng đảm nhận việc lấy mã nguồn và triển khai. Khi chỉ có một hoặc hai mã nguồn, việc triển khai chưa quá phức tạp. Tuy nhiên, thực tế một doanh nghiệp có thể có hàng chục, thậm chí hàng trăm mã nguồn. Chỉ riêng với 20 dự án trên **Git**, đã đặt ra câu hỏi: cần bao nhiêu kỹ sư vận hàng và triển khai hạ tầng để triển khai kịp tiến độ? Quá trình phát triển dự án thường diễn ra trên ba môi trường chính:

- **Development**: Môi trường để lập trình viên cập nhật các đoạn mã nguồn mới
- **Staging**: Môi trường kiểm thử, đảm bảo mọi thứ chạy ổn trước khi lên production và thường được các tester sử dụng.
- **Production**: Môi trường phục vụ người dùng cuối.

Trong môi trường **Development**, các lập trình viên cập nhật mã nguồn hàng giờ, thậm chí hàng phút, và họ cần kiểm tra mã nguồn trên môi trường kiểm thử trước khi chuyển sang các môi trường quan trọng hơn. Nếu phải chờ các kỹ sư triển khai thủ công, điều này sẽ rất tốn thời gian và ảnh hưởng đến hiệu suất dự án. Không chỉ lập trình viên gặp khó khăn, đội ngũ vận hàng và triển khai cũng gặp nhiều trở ngại khi triển khai thủ công: sao chép mã nguồn giữa các máy chủ có thể dẫn đến sai sót; họ phải truy cập từng máy chủ, **clone** mã nguồn, **build**, **run**, rồi kiểm tra **logs**... Ngoài

ra, triển khai dự án còn bao gồm nhiều công việc khác như cài đặt hệ thống, giám sát, bảo trì và nâng cấp, làm quá trình triển khai trở nên phức tạp và tốn thời gian.

1.2 Khái niệm CI/CD



Hình 1.1: Mô Hình CI/CD

Để giải quyết các khó khăn trong phần trên, **CI/CD** ra đời. **CI/CD** gồm hai phần chính:

CI (Continuous Integration)

Bước CI bao gồm: clone mã nguồn, build dự án, và tích hợp các bước kiểm thử như:

1. Kiểm tra hiệu năng
2. Kiểm tra chất lượng mã nguồn
3. Kiểm tra bảo mật
4. Và nhiều loại kiểm thử khác

Mục tiêu là đảm bảo chất lượng mã nguồn trước khi triển khai.

CD (Continuous Deployment / Continuous Delivery)

- **Continuous Deployment**: triển khai hoàn toàn tự động. Lập trình viên chỉ cần commit code, chức năng mới sẽ được triển khai ngay mà không cần thao tác thủ công.
- **Continuous Delivery**: triển khai vẫn yêu cầu bước xác nhận thủ công. Code sau khi **build** và kiểm thử chỉ được triển khai khi có sự đồng ý từ team.

Mỗi chiến lược **CD** có ưu nhược điểm riêng:

– Triển khai tự động tối ưu hiệu suất và rút ngắn thời gian ra môi trường thực tế.

– Triển khai thủ công giúp tăng khả năng kiểm soát, giảm rủi ro lỗi khi đưa code vào môi trường quan trọng. Tùy theo nhu cầu doanh nghiệp, có thể kết hợp cả hai để xây dựng quy trình **CI/CD** tối ưu, vừa nhanh, vừa an toàn.

1.3 Ưu điểm và hạn chế của quy trình CI/CD

Ưu điểm

Việc triển khai quy trình **CI/CD** mang lại nhiều lợi ích rõ rệt cho đội ngũ phát triển phần mềm:

1. **Giảm thiểu lỗi không đáng có:** Nhờ có CI/CD, các lỗi như lỗi biên dịch hoặc lỗi phát sinh do khác biệt môi trường build được phát hiện sớm. Ví dụ: cùng một source code nhưng khi bạn A build trên máy của mình và bạn B build trên máy khác có thể cho ra kết quả khác nhau. Với **CI/CD**, tất cả đều được build trong môi trường thống nhất, giúp loại bỏ rủi ro này.
2. **Đảm bảo tính ổn định và logic của hệ thống:** Các bài kiểm thử tự động giúp bảo đảm rằng việc thêm tính năng mới không ảnh hưởng đến các chức năng hiện có.
3. **Tăng hiệu quả làm việc:** Kỹ sư vận hành và hạ tầng không cần tự build hay deploy ứng dụng nữa, vì mọi thao tác đã được tự động hóa. Điều này giúp họ tập trung hơn vào việc phát triển và cải thiện chất lượng phần mềm.

4. **Nâng cao chất lượng mã nguồn:** CI/CD cho phép thiết lập các quy tắc ngay từ đầu, chẳng hạn như giới hạn kích thước của pull request hoặc số lượng thay đổi tối đa. Nhờ vậy, quy trình đánh giá mã nguồn trở nên rõ ràng và chất lượng mã nguồn được cải thiện.

5. **Phát triển kỹ năng viết Unit Test:** Khi pipeline có chỉ số ràng buộc về *code coverage*, Developer cần đảm bảo phần code mới có đủ kiểm thử để không làm giảm tỷ lệ bao phủ. Điều này giúp các developer hình thành thói quen viết Unit Test và nâng cao kỹ năng kiểm thử tự động.

6. **Tối ưu tốc độ phát triển sản phẩm:** CI/CD cho phép theo dõi chi tiết thời gian build, test hay lint check, giúp nhóm phát triển dễ dàng phát hiện và tối ưu các khâu còn chậm.

Hạn chế

Bên cạnh những ưu điểm nổi bật, CI/CD cũng có một số hạn chế cần lưu ý:

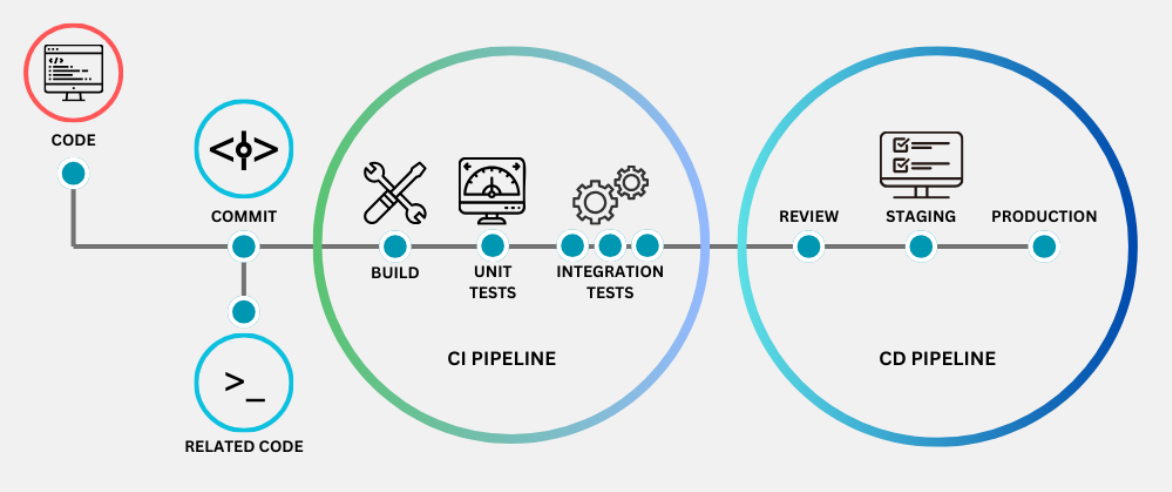
1. **Xung đột khi nhiều lập trình viên cùng làm việc:** Khi có nhiều pull request cần merge vào branch, pipeline có thể bị tắc nghẽn. Các thành viên phải chờ người khác merge xong, sau đó cập nhật lại code nếu có conflict và chạy test lại từ đầu, gây gián đoạn quá trình phát triển.

2. **Phụ thuộc vào dịch vụ bên thứ ba:** Nếu tổ chức sử dụng nền tảng CI/CD của bên ngoài như Jenkins Cloud, GitLab CI hoặc GitHub Actions, khi dịch vụ gặp sự cố hoặc ngừng hoạt động, toàn bộ pipeline sẽ bị ảnh hưởng, làm gián đoạn quá trình phát triển phần mềm.

1.4 Cách thức hoạt động của quy trình CI/CD

Quy trình CI/CD được vận hành thông qua sự phối hợp giữa hai thành phần chính:

1. **Git Repository** – nơi lưu trữ mã nguồn và quản lý toàn bộ thay đổi trong dự án.



Hình 1.2: Quy trình CI/CD

2. **CI/CD Tool** – công cụ đảm nhiệm việc tự động hóa các giai đoạn như build, test, deploy và theo dõi kết quả.

Cụ thể, khi một Developer thực hiện thay đổi trong Git repository (ví dụ: tạo *pull request* hoặc *commit* mới), Git sẽ gửi thông báo đến CI/CD tool rằng có cập nhật mới được tạo. Ngay sau đó, CI/CD tool tự động kích hoạt pipeline và chạy tuần tự các bước đã được cấu hình sẵn, thường bao gồm:

- **Biên dịch mã nguồn (Build):** Toàn bộ mã nguồn được biên dịch để kiểm tra tính hợp lệ, đảm bảo không có lỗi cú pháp hoặc lỗi phụ thuộc thư viện.
- **Kiểm thử tự động (Testing):** Hệ thống thực hiện các bài kiểm thử đơn vị (unit test), kiểm thử tích hợp (integration test), hoặc kiểm thử giao diện (UI test) để đảm bảo logic chương trình hoạt động đúng.
- **Phân tích chất lượng mã (Static Code Analysis):** Mã nguồn được kiểm tra bằng các công cụ như SonarQube, ESLint hoặc Checkstyle để phát hiện lỗi tiềm ẩn, vi phạm quy tắc code, hoặc các rủi ro bảo mật.
- **Kiểm tra quy tắc (Code Policy Check):** Pipeline xác minh các quy định nội bộ như kích thước *pull request*, quy ước đặt tên, hay tiêu chuẩn lập trình.

Giai đoạn sau kiểm thử (Post-testing stage)

Khi toàn bộ các bài kiểm thử và kiểm tra chất lượng mã đều đạt yêu cầu, CI/CD tool sẽ tự động chuyển sang các bước xử lý tiếp theo:

1. **Đóng gói và build artifact:** Mã nguồn sau khi được xác nhận hợp lệ sẽ được đóng gói thành một *artifact* (ví dụ: file `.jar`, `.war`, `.zip` hoặc Docker image). Artifact này được lưu trữ trong kho lưu trữ riêng (như Artifactory, Nexus hoặc Docker Registry) để phục vụ cho việc triển khai ở các môi trường tiếp theo.

2. **Triển khai tự động (Deploy):** Hệ thống tự động triển khai artifact vừa build lên môi trường *staging* hoặc *production* tùy theo cấu hình. Việc deploy có thể được thực hiện thông qua các công cụ như Jenkins, GitLab CI/CD, ArgoCD hoặc Kubernetes. Ở giai đoạn này, pipeline có thể tích hợp thêm các bài kiểm thử sau triển khai (post-deployment tests) nhằm xác nhận ứng dụng hoạt động đúng trên môi trường thực tế.

3. **Khả năng rollback:** Trong trường hợp phiên bản mới gây lỗi hoặc ảnh hưởng đến hệ thống, pipeline hỗ trợ cơ chế **rollback** để quay lại phiên bản ổn định trước đó. Rollback có thể được thực hiện thủ công hoặc tự động (auto-rollback) nếu hệ thống phát hiện các chỉ số bất thường như downtime, lỗi 5xx, hoặc giảm hiệu năng.

Tổng kết

Khi toàn bộ quá trình hoàn tất, CI/CD tool phản hồi kết quả ngược lại cho Git repository, thể hiện trạng thái của *pull request* hoặc *commit* là thành công hoặc thất bại. Dựa vào đó, tester có thể dễ dàng đánh giá sơ bộ chất lượng code trước khi thực hiện đánh giá chi tiết.

Tuy nhiên, quy trình CI/CD chỉ có thể tự động hóa và kiểm soát được một phần logic kỹ thuật. Việc đánh code thủ công từ lập trình viên hoặc tester vẫn là bước cần thiết nhằm đảm bảo mã nguồn đáp ứng tiêu chuẩn của nhóm và xử lý được những trường hợp logic phức tạp mà hệ thống tự động chưa thể bao quát hết.

Chương 2

Giới thiệu các nền tảng và ông cụ triển khai CI/CD

2.1 Các nền tảng được sử dụng triển khai CI/CD

Để triển khai một quy trình CI/CD hiệu quả thì có rất nhiều yếu tố phải quan tâm như là: Những tài nguyên hiện có của dự án, kinh nghiệm sử dụng các công cụ của các kỹ sư trong công ty và chi phí để phát triển dự án, ... Và từ những yếu tố khác nhau đó thì mỗi dữ án, mỗi công ty sẽ phải lựa chọn những nền tảng và công cụ phù hợp để xây dựng quy trình CI/CD của mình. Hiện nay, ba nền tảng phổ biến thường được sử dụng là GitHub, GitLab và Jenkins, mỗi nền tảng có đặc điểm và cách tiếp cận riêng, phù hợp với những nhu cầu khác nhau của dự án.

2.1.1 GitHub

GitHub là nền tảng quản lý mã nguồn phổ biến, nổi bật với sự đơn giản, dễ sử dụng và tích hợp mạnh mẽ với nhiều công cụ bên thứ ba. GitHub hỗ trợ quản lý phiên bản, theo dõi lịch sử thay đổi, và cung cấp môi trường cộng tác trực tuyến, giúp các nhóm phát triển dễ dàng làm việc cùng nhau.

GitHub Cloud

GitHub Cloud là dịch vụ lưu trữ mã nguồn trên đám mây, giúp người dùng truy cập dự án từ bất kỳ đâu mà không cần cài đặt và quản lý server riêng. Nhờ GitHub Cloud,

các dự án nhỏ hoặc các nhóm làm việc phân tán có thể triển khai CI/CD nhanh chóng, tận dụng hạ tầng sẵn có của GitHub. Ngoài ra, GitHub Cloud luôn được cập nhật và bảo mật bởi chính đội ngũ GitHub, giúp giảm gánh nặng vận hành cho doanh nghiệp.

GitHub Actions

GitHub Actions là công cụ tích hợp sẵn cho phép tự động hóa workflow CI/CD trực tiếp trên GitHub. Actions cho phép cấu hình các pipeline để tự động build, test, deploy mỗi khi có thay đổi trong kho mã nguồn. GitHub Actions hỗ trợ nhiều môi trường, từ Linux, Windows, macOS đến các container, đồng thời có thư viện hành động (actions) phong phú giúp triển khai nhanh chóng mà không cần viết script phức tạp. Nhờ đó, GitHub trở thành nền tảng CI/CD "all-in-one" tiện lợi cho các dự án vừa và nhỏ.

2.1.2 GitLab

GitLab là nền tảng quản lý mã nguồn toàn diện, nổi bật với khả năng kết hợp quản lý dự án, issue tracking và CI/CD trong cùng một hệ sinh thái. GitLab phù hợp với các tổ chức muốn kiểm soát toàn bộ vòng đời phát triển phần mềm từ một nơi, từ lập kế hoạch, phát triển, kiểm thử đến triển khai.

GitLab Server

GitLab Server có thể được cài đặt trên máy chủ riêng hoặc trên cloud do tổ chức quản lý. Nó cung cấp toàn quyền kiểm soát về bảo mật, truy cập và cấu hình pipeline CI/CD, đồng thời hỗ trợ quản lý các dự án lớn với nhiều nhóm phát triển. GitLab Server đặc biệt phù hợp cho các doanh nghiệp có yêu cầu cao về bảo mật hoặc muốn duy trì dữ liệu nội bộ.

GitLab Runner

GitLab Runner là thành phần thực thi các công việc CI/CD, chịu trách nhiệm build, test và deploy theo pipeline đã cấu hình. Runner có thể được cài đặt trên nhiều máy, giúp phân tán tải và tăng tốc độ xử lý pipeline. GitLab Runner cũng linh hoạt về môi trường triển khai, có thể chạy trong container, máy vật lý hoặc cloud, cho phép tùy

chỉnh theo nhu cầu cụ thể của từng dự án. Nhờ vậy, GitLab vừa cung cấp khả năng kiểm soát nội bộ, vừa hỗ trợ tự động hóa CI/CD hiệu quả.

2.1.3 Jenkins

Jenkins là một công cụ tự động hóa mã nguồn mở, nổi bật với khả năng linh hoạt, plugin phong phú và hỗ trợ tích hợp với hầu hết các hệ thống hiện có. Jenkins phù hợp với các dự án cần pipeline CI/CD phức tạp, với nhiều bước kiểm thử, build, deploy và các quy trình tùy chỉnh đặc thù.

Jenkins Server

Jenkins Server là trung tâm quản lý, nơi lưu trữ cấu hình pipeline, trigger build khi có thay đổi trong kho mã nguồn và tổng hợp kết quả thực thi từ các agent. Server chịu trách nhiệm điều phối workflow CI/CD, cung cấp giao diện quản lý trực quan và báo cáo kết quả các pipeline. Jenkins Server thường được cài đặt trên máy chủ riêng hoặc máy chủ cloud, và có thể mở rộng quy mô bằng cách thêm nhiều agent để thực thi pipeline.

Jenkins Agent

Jenkins Agent thực hiện các công việc build, test, deploy theo chỉ định của Jenkins Server. Agent giúp phân tán tải, chạy pipeline trên nhiều môi trường khác nhau, từ Linux, Windows, macOS đến các container. Nhờ Agent, Jenkins có thể xử lý các pipeline phức tạp, hỗ trợ nhiều dự án cùng lúc và tối ưu hiệu suất hệ thống. Cấu trúc Server-Agent của Jenkins cũng cho phép mở rộng linh hoạt, thích hợp cho các dự án lớn hoặc doanh nghiệp có nhiều nhóm phát triển.

2.1.4 So sánh ba nền tảng Github, Gitlab, Jenkins

- GitHub phù hợp với các dự án nhỏ và vừa, ưu điểm là dễ sử dụng, nhanh chóng triển khai CI/CD mà không cần hạ tầng riêng. Tuy nhiên đối với các dự án đòi hỏi tính bảo mật cao thì việc công khai mã nguồn dụ án của mình trên Github cloud là điều thể. Nếu sử dụng các dịch vụ cho doanh nghiệp của Github để triển

khai dự án thì chi phí quá cao. Vì vậy, Github phù hợp các dự án nhỏ và không đòi hỏi bảo mật cao.

- GitLab thích hợp với doanh nghiệp muốn kiểm soát toàn bộ vòng đời phát triển, đặc biệt khi cần cài đặt trên server nội bộ. Gitlab hỗ trợ cài đặt Gitlab server trên máy chủ của công ty tăng tính bảo mật, so với Github thì gitlab cung cấp nhiều bộ công cụ hơn, nhiều tính năng cho dự án lớn. Đây là cũng là công cụ thường được các doanh nghiệp sử dụng.

- Jenkins mạnh về linh hoạt và khả năng mở rộng, phù hợp với các pipeline phức tạp và dự án lớn, nhưng yêu cầu vận hành nhiều hơn. So với 2 nền tảng trước Jenkins là nên tảng mạnh mã nhất về triển khai tự động, cho kỹ sư vận hàng những khả năng tùy biến cao. Tuy không trực tiếp lưu trữ mã nguồn, nhưng với việc tích hợp được nhiều tiện ích đã khiến Jenkins được nhiều công ty lớn tin dùng. Tuy vậy, Jenkins là nền tảng phức tạp đòi hỏi đội ngũ vận hành phải học thêm kỹ năng để sử dụng.

Nhờ những đặc điểm này, việc lựa chọn nền tảng CI/CD phụ thuộc vào quy mô dự án, yêu cầu bảo mật, ngân sách và mức độ phức tạp của pipeline.

2.2 Các công cụ và ứng dụng cần thiết khác

Ngoài các nền tảng CI/CD chính, quá trình triển khai phần mềm còn cần một số công cụ hỗ trợ khác để đảm bảo môi trường phát triển, quản lý server và kiểm soát chất lượng mã nguồn. Các công cụ này giúp tăng tính ổn định, tối ưu hóa hiệu suất và đảm bảo phần mềm luôn đạt chất lượng cao trước khi triển khai.

2.2.1 VMware Workstation Pro

VMware Workstation Pro là phần mềm ảo hóa mạnh mẽ, cho phép tạo và quản lý nhiều máy ảo trên cùng một máy tính vật lý. Trong quá trình phát triển và triển khai CI/CD, VMware Workstation Pro giúp:

- Tạo môi trường thử nghiệm giống hệt môi trường thực tế mà không ảnh hưởng đến máy chủ chính.
- Chạy nhiều hệ điều hành khác nhau song song, thuận tiện cho việc kiểm thử đa nền tảng.
- Dễ dàng sao lưu, khôi phục máy ảo, giảm thiểu rủi ro khi triển khai thử nghiệm.

Nhờ VMware Workstation Pro, các nhóm phát triển có thể mô phỏng môi trường triển khai, kiểm tra tính tương thích và đảm bảo các pipeline CI/CD hoạt động ổn định trước khi đưa phần mềm lên server thật.

2.2.2 Nginx

Nginx là một web server và reverse proxy phổ biến, nổi bật với khả năng xử lý lượng lớn kết nối đồng thời và hiệu suất cao. Trong quy trình CI/CD, Nginx thường được sử dụng để:

- Phục vụ ứng dụng web sau khi deploy, đảm bảo tốc độ tải trang nhanh và ổn định.
- Reverse proxy để điều phối các request đến nhiều server backend khác nhau, hỗ trợ triển khai microservices.
- Cấu hình SSL/TLS để bảo mật kết nối giữa client và server.

Nginx giúp đảm bảo rằng các ứng dụng được triển khai thông qua pipeline CI/CD có thể hoạt động mượt mà trong môi trường sản xuất, đồng thời hỗ trợ các chiến lược deploy như blue-green hoặc canary deployment.

2.2.3 SonarQube

SonarQube là công cụ phân tích chất lượng mã nguồn tự động, hỗ trợ phát hiện lỗi, code smells và các vấn đề về bảo mật ngay trong giai đoạn phát triển. Khi tích hợp vào CI/CD, SonarQube giúp:

- Phân tích mã nguồn sau mỗi lần commit hoặc build, đảm bảo chất lượng code được duy trì.

- Đưa ra báo cáo chi tiết về lỗi tiềm ẩn, các tiêu chuẩn coding và khả năng bảo mật.
- Hỗ trợ nhiều ngôn ngữ lập trình, dễ dàng tích hợp với các nền tảng CI/CD như GitHub Actions, GitLab CI/CD hay Jenkins.

Việc sử dụng SonarQube trong pipeline CI/CD giúp nâng cao chất lượng phần mềm, giảm thiểu lỗi sản phẩm và đảm bảo code luôn tuân thủ các chuẩn mực kỹ thuật.

2.2.4 Docker

Docker là nền tảng mã nguồn mở cho phép đóng gói, triển khai và chạy ứng dụng trong các container — môi trường nhẹ, độc lập và nhất quán giữa các hệ thống. Khi tích hợp vào quy trình CI/CD, Docker mang lại nhiều lợi ích quan trọng như:

- Giúp tạo môi trường triển khai đồng nhất giữa các giai đoạn (phát triển, kiểm thử, sản xuất), hạn chế lỗi do khác biệt cấu hình.
- Tăng tốc độ triển khai nhờ cơ chế container hóa, cho phép khởi tạo và hủy môi trường chỉ trong vài giây.
- Dễ dàng tích hợp với các hệ thống CI/CD như Jenkins, GitHub Actions hay GitLab CI/CD để tự động hóa quá trình build và deploy ứng dụng.
- Cho phép quản lý phiên bản của môi trường chạy thông qua Dockerfile, giúp đảm bảo tính tái lập và kiểm soát thay đổi.

Việc sử dụng Docker trong pipeline CI/CD giúp đảm bảo quá trình triển khai nhanh chóng, ổn định và dễ mở rộng, đồng thời giảm thiểu các rủi ro khi triển khai ứng dụng trên nhiều môi trường khác nhau.

Chương 3

Xây dựng hệ thống server ảo

3.1 Giới thiệu kiến trúc hệ thống

3.1.1 Kiến trúc sử dụng cho nền tảng Github và Gitlab

3.1.2 Kiến trúc sử dụng cho nền tảng Jenkins

3.2 Cài đặt hệ thống server ảo

Chương 4

Xây dựng và triển khai quy trình CI/CD

4.1 Xây dựng quy trình CI/CD

4.2 Triển khai CI/CD

Chương 5

Xây dựng các trường hợp kiểm thử và trình bày kết quả

5.1 Viết Unit Test và Integration Test

5.2 Trình bày kết quả

Kết luận và hướng phát triển

Tài liệu tham khảo