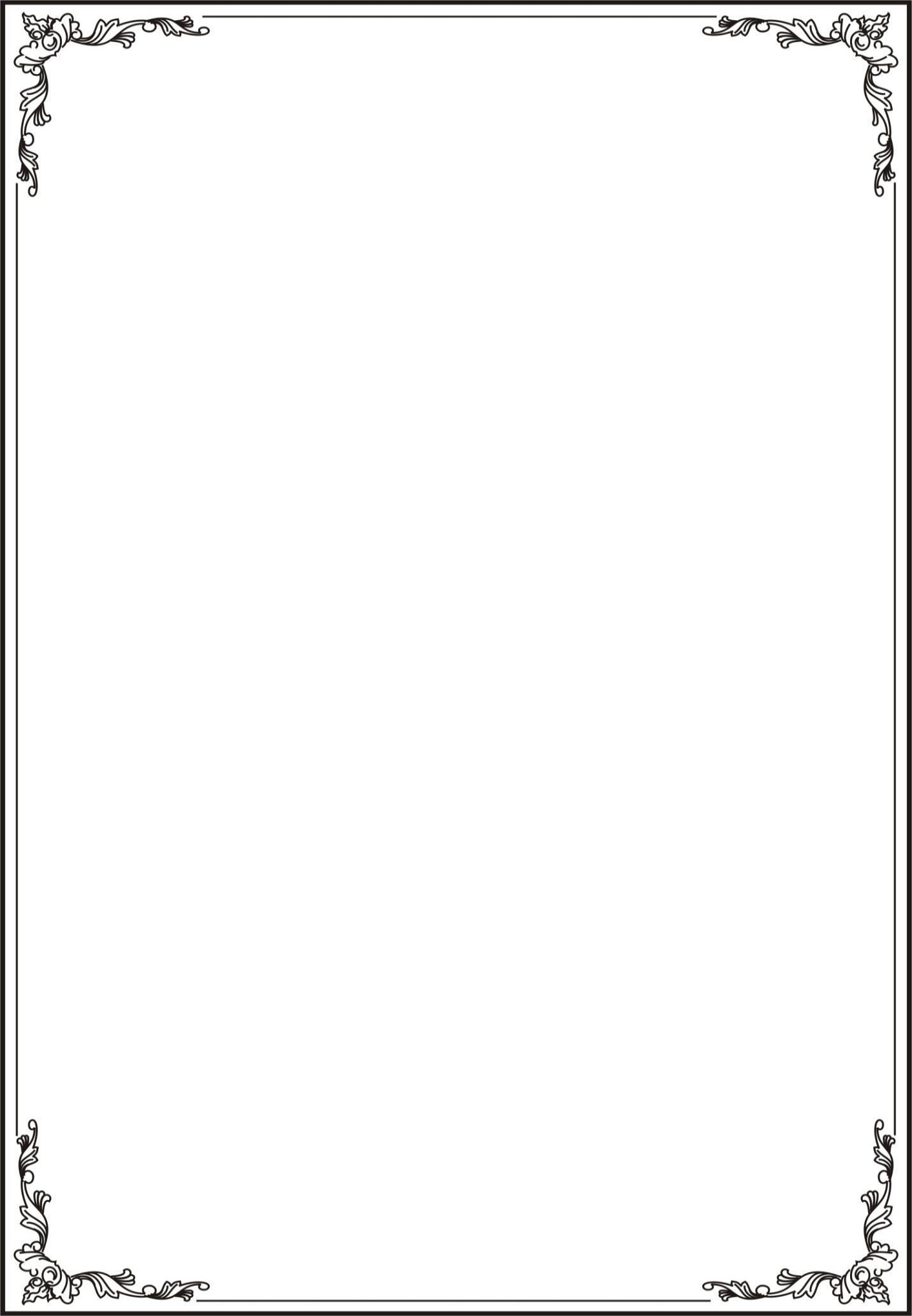
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

\_\_\_\_\_\_\_\_\_\_\_\_\_🙠⬥🙢\_\_\_\_\_\_\_\_\_\_\_\_\_



**BÁO CÁO BÀI TẬP LỚN**

**ĐỀ TÀI:**

**Docker for DevOps and Developer**

Giảng viên hướng dẫn : **TS.Quách Xuân Trưởng**

**Sinh viên thực hiện: Nhóm 2**

***1. Hà Quốc Hữu (nhóm trưởng)***

***2. Đinh Trọng Giáp***

***3. Nguyễn Bá Tùng Lâm***

**Lớp: *N02***

***Thái Nguyên ngày….tháng…., Năm 2024***

MỤC LỤC

[CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI 3](#_1fob9te)

[**1.1 Tầm quan trọng của Docker trong quy trình DevOps and Devevoper 3**](#_ncze9ugy5am6)

[1.2 Lý do chọn đề tài 4](#_3znysh7)

[1.3 Phạm vi chủ đề 5](#_2et92p0)

[1.4 Mục tiêu đề tài 5](#_tyjcwt)

[1.5 Tổng quan về quy trinh DevOps 6](#_3dy6vkm)

[1.6 Tầm quan trọng của giai đoạn Đóng gói và triển khai trong DevOps 8](#_1t3h5sf)

[1.7 Các khó khăn trong giai đoạn đóng gói và triển khai 8](#_1kqzkbct725)

[CHƯƠNG 2: CÔNG CỤ ĐÓNG GÓI VÀ TRIỂN KHAI CHƯƠNG TRÌNH 9](#_4d34og8)

[2.1. Đóng gói dự án 9](#_2s8eyo1)

[2.2. Triển khai dự án 9](#_17dp8vu)

[2.3 Lý do chọn Docker tròn giai đoạn đóng gói và triển khai 9](#_1ci93xb)

[CHƯƠNG 3: TÌM HIỂU VỀ Docker 9](#_3whwml4)

[3.1 9](#_2bn6wsx)

[3.1.1 Sơ lược về Docker 9](#_qsh70q)

[3.1.2 Lịch sử Docker 10](#_3as4poj)

[3.1.3 Các khái niệm quan trọng của Docker 12](#_1pxezwc)

[1. Container 12](#_z77q4nm02hpp)

[2. Image 12](#_xef9gtc01zon)

[3.1.4 Cách hoạt động của Docker 14](#_49x2ik5)

[a, Xây dựng. 14](#_f77igujlv24m)

[b, Đẩy. 14](#_uob403z888ej)

[c, Kéo, Chạy 15](#_t7ksekyci9rm)

[3.1.5 Quy trình hệ thống dùng Docker 15](#_2p2csry)

[1. Xây dựng Docker Image 15](#_ey5tpj5xwows)

[2. Chạy Container từ Docker Image 15](#_x417qw8mdyxz)

[3. Quản lý Nhiều Container bằng Docker Compose 15](#_7ubvy4g77jnt)

[4. Thử nghiệm và Phát triển trong Docker 16](#_i8a5tk80se80)

[5. Triển khai Docker Container lên Server 16](#_2dj4id6nllre)

[6. Sử dụng Docker Swarm hoặc Kubernetes cho triển khai lớn 16](#_6w3sttfpq697)

[7. Giám sát và Bảo trì Container 16](#_dzgid1ubuxj)

[\*Ưu nhược và nhược điểm của Docker 16](#_av73jvpblmzb)

[Ưu điểm của Docker 16](#_4n8a63tpxshd)

[Nhược điểm của Docker 17](#_4ir3z9lf3jpz)

[3.2 Công cụ kết hợp với Docker 18](#_147n2zr)

[3.2.1 Docker Compose 18](#_3o7alnk)

[Docker Compose là gì? 18](#_sxu3u7uvynk2)

[Các thành phần chính của Docker Compose 18](#_t7d64do06r9w)

[Các tính năng chính của Docker Compose 19](#_c9z11kljkakr)

[3.2.2 DockerHub. 19](#_fgoxtaerv9as)

[Docker Hub là gì? 19](#_1w8y2n5guiac)

[Các tính năng chính của Docker Hub 19](#_y8curodfvy8g)

[3. 2.3 Kubernetes 20](#_23ckvvd)

[1.Kubernetes và Docker 20](#_yygidyz8kajz)

[2.Cách Kubernetes và Docker hoạt động cùng nhau 21](#_5xj3tj6ok8gr)

[3.Các thành phần tương tự và khác biệt giữa Kubernetes và Docker 21](#_1ecpkeiipc9h)

[4.Kết luận 22](#_a0vorcy54ckw)

[3.2.4 Jenkins 22](#_ty61ay6syv93)

[● 22](#_3gax934lzmpc)

[3.3. Các câu lệnh của Docker 22](#_rfn6ykekku8)

[● cài đặt Docker trên Linux, Window, MacOs 23](#_g56nibn625d1)

[CHƯƠNG 4 TRIỂN KHAI DEMO 24](#_nmf14n)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 35](#_37m2jsg)

[TÀI LIỆU THAM KHẢO 38](#_1mrcu09)

# CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

# 1.1 [Tầm quan trọng của Docker trong quy trình DevOps and Delevoper](#_46r0co2)

- Docker đóng vai trò quan trọng trong đóng gói và triển khai dự án phần mềm. Dưới đây là một số điểm quan trọng về tầm quan trọng của Docker trong đóng gói và triển khai:

+ Đóng gói chương trình với các phụ thuộc: Docker là công cụ cung cấp khả năng đóng gói và chạy chương trình dưới môi trường lỏng lẻo độc lập với nhau gọi là Container.

+ Phân phối chương trình nhanh và nhất quán: Docker cho phép nhà phát triển làm việc trong môi trường tiêu chuẩn hóa sử dụng container cục bộ, nó cung cấp các dịch vụ. Sử dụng container tốt cho quy trình làm việc CI/CD.

+ Thuận lợi triển khai và mở rộng quy mô: Nền tảng dựa trên container của Docker cho phép khối lượng công việc có tính linh động cao. Container của Docker có thể chạy trên laptop, máy vật lí hay máy ảo trên hệ thống dữ liệu, trên nhà cung cấp dịch vụ đám mây, hay trên hỗn hợp môi trường. Bản chất linh động và nhẹ nhàng của Docker cũng khiến nó trỏe nên dễ dàng năng động để quản lý khối lượng công việc, mở rộng hoặc loại bỏ các ứng dụng và dịch vụ theo nhu cầu kinh doanh, trong thời gian gần như thực.

+ Chạy nhiều khối lượng công việc hơn trên cùng phần cứng: Docker nhẹ mà nhanh. Nó cung cấp tính khả thi, giải pháp thay thế hiệu quả về mặt chi phí cho các máy ảo dựa trên bộ ảo hóa, nhờ đó bạn có thể sử dụng nhiều dung lnợn máy chủ hơn để đạt được mục tiêu kinh doanh của mình. Docker hoàn hảo cho môi trường mật độ cao và sự triển khai vừa và nhỏ khi bạn cần nhiều tài nguyên với ít tài nguyên hơn.

Why ?

## 1.2 Lý do chọn đề tài

- Việc chúng tôi chọn đề tài về “Docker cho vận hành - bảo trì và nhà phatr triển” có thể được lý giải bằng một số lý do sau:

* Khó khăn gặp phải tron quá trình đóng gói và triển khai: Các chương trình cần cài đặt các phụ thuộc khác và triển khai trên môi trường host.

+ Phổ biến và quan trọng: Docker phổ biến và quan trọng cho lập trình viên và kĩ sư vận hành. Nắm vững kiến thức về chúng là một yêu cầu cơ bản đối với các kĩ sư phát triển phần mềm.

+ Tính thực tiễn: Việc nghiên cứu và hiểu rõ về Docker mang lại lợi ích lớn cho việc phát triển phần mềm thực tế. Các kỹ thuật và công cụ liên quan đến Docker thường được áp dụng trong các dự án phần mềm thực tế, giúp tăng hiệu quả làm việc với khâu đóng gói và triển khai.

+ Cơ hội nghề nghiệp: Hiểu biết về Docker là một kỹ năng cộng cho các cho nhà phát triển phần mềm, và là mảng kiến thức quan trọng với các kĩ sư vận hành. Việc nắm vững kiến thức về chúng có thể tạo ra cơ hội nghề nghiệp tốt trong ngành công nghiệp phần mềm.

+ Hỗ trợ cộng đồng: Docker có một cộng đồng phát triển rất lớn và tích cực. Việc nghiên cứu và chia sẻ kiến thức về chúng có thể góp phần vào việc phát triển và thúc đẩy sự chia sẻ kiến thức trong cộng đồng.

+ Tính hệ thống và linh hoạt: Việc hiểu biết về cách sử dụng chúng một cách hiệu quả có thể giúp tăng tính linh hoạt và hiệu quả của quá trình phát triển phần mềm.

## 1.3 Phạm vi chủ đề

* Lý thuy ết Quy trình DevOPs tổng quan, Pha đóng gói và Triển khai.

- Giới thiệu và cơ bản về Docker: Điều này bao gồm giới thiệu về Docker là gì, lịch sử phát triển của nó, cách thức hoạt động, cùng với các khái niệm cơ bản như Dockerfile ,Image, Container,...

- Cài đặt và cấu hình Docker: Hướng dẫn cài đặt Docker trên các hệ điều hành khác nhau và cách cấu hình Docker.

- Các lệnh cơ bản của Docker: Các lệnh quan trọng như: build, run, pull, push,...

- Đóng gói với Docker: Hướng dẫn cách sử dụng file Dockercompose để tạo Image.

- Quản lý dự án trên DockerHub: Giới thiệu về các tính năng của DockerHub.

- .Các công cụ Docker đi kèm.

- Tích hợp Docker với quy trình DevOps: Hướng dẫn tích hợp Docker vào quy trình CI/CD.

## 1.4 Mục tiêu đề tài

* Cung cấp kiến thức tổng quan về Quy trình DevOps, pha đóng gói và triển khai trong quy trình DevOps

-  Hiểu biết sâu sắc về Docker: Mục tiêu chính là hiểu rõ về cách hoạt động của Docker, từ các khái niệm cơ bản như Container, Image, DockerCompose đến các khái niệm phức tạp như …

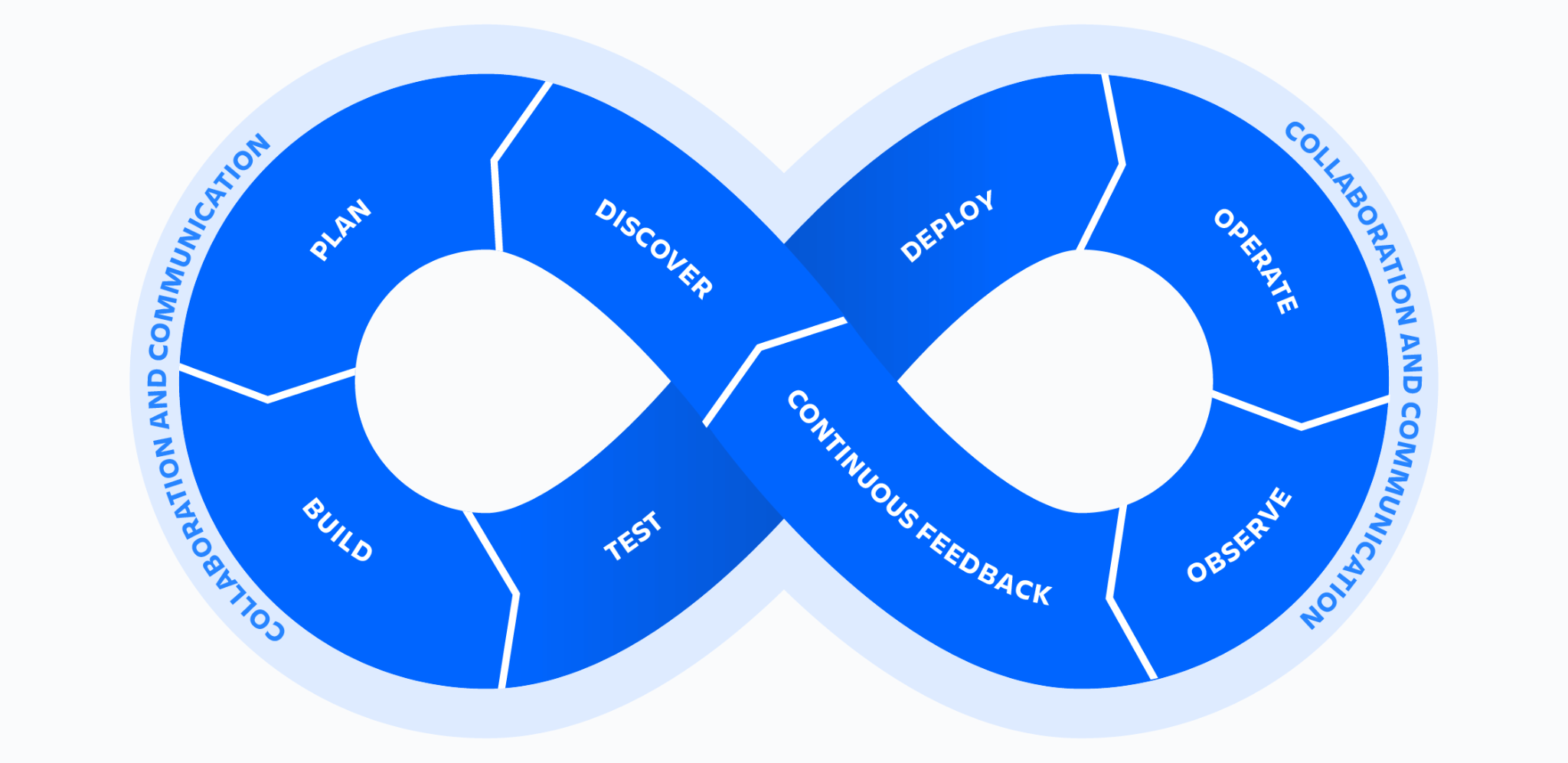
- Cài đặt Docker trên các hệ điều hành khác nhau.

- Áp dụng Docker trong dự án thực tế: Mục tiêu này đặt ra là áp dụng kiến thức về Docker vào một dự án phần mềm thực tế, từ việc đóng gói một chương trình cho đến việc triển khai lên môi trường host.

- Tối ưu hóa quy trình làm việc: Cách tích hợp Docker với các công cụ và quy trình DevOps.

- Nâng cao kỹ năng cá nhân: Mục tiêu cuối cùng là nâng cao kỹ năng của các thành viên trong nhóm nghiên cứu về việc sử dụng Docker, và đóng góp vào việc chia sẻ kiến thức.

## 1.5 Tổng quan về quy trinh DevOps



DevOps .. là tập hợp các phương pháp, công cụ và triết lý văn hóa mà tự động hóa và tích hợp các quy trình giữa nhóm phát triển phần mềm và nhóm vận hành. Nó nhấn mạnh việc trao quyền cho nhóm, giao tiếp và cộng tác giữa các nhóm cũng như tự động hóa công nghệ.

Vòng đời devOps: Vì tính chất liên tục của DevOps, Người thực hành sửa dụng vòng lặp vô cực để hiển thị các giai đoạn của vòng đời DevÓp liên quan với nhau ntn. Mặc dù có vẻ như diễn ra tuần tự nhưng vòng lặp tượng trưng cho nhu cầu hợp tác liên tục và cải tiến lặp đi lặp lại trong toàn bộ vòng đời:

Discover: Xây dựng phần mềm là một môn thể thao đồng đội. Để chuẩn bị cho sprint sắp tới, các nhóm phải hội thảo để khám phá, sắp xếp và ưu tiên các ý tưởng. Ý tưởng phải phù hợp với mục tiêu chiến lược và mang lại tác động cho khách hàng. Agile có thể giúp hướng dẫn các nhóm DevOps.

Plan: Các nhóm DevOps nên áp dụng các phương pháp thực hành linh hoạt để cải thiện tốc độ và chất lượng. Agile là một cách tiếp cận lặp đi lặp lại để quản lý dự án và phát triển phần mềm, giúp các nhóm chia công việc thành những phần nhỏ hơn để mang lại giá trị gia tăng.

Build: Git là một hệ thống kiểm soát phiên bản mã nguồn mở và miễn phí. Nó cung cấp hỗ trợ tuyệt vời cho việc phân nhánh, hợp nhất và viết lại lịch sử kho lưu trữ, dẫn đến nhiều quy trình làm việc và công cụ sáng tạo và mạnh mẽ cho quy trình xây dựng phát triển.

Test: Tích hợp liên tục (CI) cho phép nhiều nhà phát triển đóng góp vào một kho lưu trữ chung duy nhất. Khi các thay đổi mã được hợp nhất, các thử nghiệm tự động được chạy để đảm bảo tính chính xác trước khi tích hợp. Việc hợp nhất và thử nghiệm mã thường giúp các nhóm phát triển có được sự đảm bảo về chất lượng và khả năng dự đoán của mã sau khi triển khai.

Deploy: Triển khai Triển khai liên tục (CD) cho phép các nhóm phát hành các tính năng thường xuyên vào sản xuất theo cách tự động. Các nhóm cũng có tùy chọn triển khai với cờ tính năng, cung cấp mã mới cho người dùng một cách đều đặn và có phương pháp thay vì tất cả cùng một lúc. Cách tiếp cận này cải thiện tốc độ, năng suất và tính bền vững của các nhóm phát triển phần mềm.

Operate: Quản lý việc cung cấp dịch vụ CNTT từ đầu đến cuối cho khách hàng. Điều này bao gồm các hoạt động liên quan đến thiết kế, triển khai, cấu hình, triển khai và bảo trì tất cả cơ sở hạ tầng CNTT hỗ trợ các dịch vụ của tổ chức.

Observe: Nhanh chóng xác định và giải quyết các vấn đề ảnh hưởng đến thời gian hoạt động, tốc độ và chức năng của sản phẩm. Tự động thông báo cho nhóm của bạn về những thay đổi, hành động có rủi ro cao hoặc lỗi để bạn có thể tiếp tục duy trì dịch vụ.

Continuous feedback: Các nhóm DevOps nên đánh giá từng bản phát hành và tạo báo cáo để cải thiện các bản phát hành trong tương lai. Bằng cách thu thập phản hồi liên tục, các nhóm có thể cải thiện quy trình của mình và kết hợp phản hồi của khách hàng để cải thiện bản phát hành tiếp theo.

## 1.6 Tầm quan trọng của giai đoạn Đóng gói và triển khai trong DevOps

* Giai đoạn đóng gói quan trọng vì:
* Quản lí và đóng gói các phụ thuộc của dự án, sẵn sàng cho chạy hoàn thiện dự án.
* sản phẩm sau quá trình đóng gói là quan trọng cho quá trình test và sẵn sàng cho quá trình triển khai.
* thường có phạm vi nhỏ hơn, giảm nguy cơ đưa các lỗi hoặc sự cố lớn vào phần mềm. Điều này cho phép các nhóm phát hiện và giải quyết các vấn đề sớm hơn trong quá trình phát triển, trước khi chúng trở thành các vấn đề lớn hơn;
* khuyến khích hợp tác giữa các nhóm trong quá trình DevOps.

Trong giai đoạn DevOps, triển khai là rất quan trọng vì các lý do sau:

* cho phép các nhóm nhanh chóng cung cấp các tính năng và cải tiến mới cho người dùng, giảm thời gian đưa sản phẩm và bản cập nhật mới ra thị trường;
* cho phép các nhóm nhận phản hồi từ người dùng và các bên liên quan nhanh hơn, cho phép họ thực hiện các điều chỉnh cần thiết cho phần mềm và cải thiện chất lượng của phần mềm;
* yêu cầu sự hợp tác chặt chẽ giữa các nhóm phát triển, thử nghiệm và vận hành, thúc đẩy giao tiếp và hợp tác tốt hơn giữa các phòng ban;
* tạo ra văn hóa cải tiến liên tục, khuyến khích các nhóm liên tục lặp lại và cải thiện phần mềm dựa trên phản hồi từ người dùng và các bên liên quan.

## 1.7 Các khó khăn trong giai đoạn đóng gói và triển khai

TRong giai đoạn đóng gói:

Cài đặt và liên kết giữa các thành phần

Quản lí các phụ thuộc, phiên bản, thư viện khác nhau, cấu hình dự án

Giai đoạn triển khai: Cài đặt trên máy chủ, m áy khách

Cài đặt trên các h ệ điều hành khác

* Lựa chọn dịch vụ triển khai
* Cấu hình: Thiết lập sản phẩm mới đòi hỏi phải cấu hình các thông số cẩn thận để phù hợp với các yêu cầu của sản phẩm.
* Cài đặt và liên k ết giữa các thành phần

# CHƯƠNG 2: CÔNG CỤ ĐÓNG GÓI VÀ TRIỂN KHAI CHƯƠNG TRÌNH

# 2.1. Đóng gói dự án

Khái niệm cơ bản về đóng gói ( các phụ thuộc, các phiên bản, các lỗi khi đóng gói dự án,..)

Công cụ đóng gói dự án (...

# 2.2. Triển khai dự án

Khái niệm cơ bản (ác lỗi gặp phải khi triển khai, công cụ triển khai, lưu trữ,... )

Công cụ triển khai dự án(...

# 2.3 Lý do chọn Docker tròn giai đoạn đóng gói và triển khai

# CHƯƠNG 3: TÌM HIỂU VỀ Docker

## 3.1

## 3.1.1 Sơ lược về Docker

Docker là nền tảng cung cấp cho các công cụ, service để các developers, adminsystems có thể phát triển, thực thi, chạy các ứng dụng với containers. Hay nói một cách khác nó là một nền tảng để cung cấp cách để building, deploy và run các ứng dụng một cách dễ dàng trên nền tảng ảo hóa - "Build once, run anywhere". Hay nói một cách dễ hiểu như sau: Khi chúng ta muốn chạy app thì chúng ta phải thiết lập môi trường chạy cho nó. Thay vì chúng ta sẽ đi cài môi trường chạy cho nó thì chúng ta sẽ chạy docker.

Ứng dụng Docker chạy trong vùng chứa (container) có thể được sử dụng trên bất kỳ hệ thống nào: máy tính xách tay của nhà phát triển, hệ thống trên cơ sở hoặc trong hệ thống đám mây. Và là một công cụ tạo môi trường được "đóng gói" (còn gọi là Container) trên máy tính mà không làm tác động tới môi trường hiện tại của máy, môi trường trong Docker sẽ chạy độc lập.

Docker có thể làm việc trên nhiều nền tảng như Linux, Microsoft Windows và Apple OS X

## 3.1.2 Lịch sử Docker

--Lich sử docker

Docker là một nền tảng mã nguồn mở được phát triển để tự động hóa việc triển khai ứng dụng trong các container. Dưới đây là một cái nhìn tổng quan về lịch sử của Docker:

**1. Khởi nguồn (2010-2013)**

* 2010: Solomon Hykes bắt đầu phát triển một dự án có tên "dotCloud", một dịch vụ Platform as a Service (PaaS). Đây là nền tảng gốc của Docker.
* 2013: Docker chính thức được ra mắt vào tháng 3. Nó được công bố tại hội nghị PyCon và nhanh chóng thu hút sự chú ý từ cộng đồng lập trình viên.

**2. Phát triển nhanh chóng (2013-2014)**

* Docker trở thành một dự án mã nguồn mở, thu hút sự đóng góp từ nhiều lập trình viên.
* Các tính năng như Docker Hub (nơi lưu trữ và chia sẻ các container) được giới thiệu.
* Docker đã chứng minh được sự hiệu quả trong việc triển khai, mở rộng và quản lý ứng dụng.

**3. Tăng trưởng và phổ biến (2014-2015)**

* Docker trở thành một phần quan trọng trong DevOps và kiến trúc microservices.
* Nhiều công ty lớn bắt đầu sử dụng Docker để cải thiện quy trình phát triển và triển khai ứng dụng.
* Docker Inc. được thành lập để phát triển và hỗ trợ Docker.

**4. Tiến hóa và cộng đồng (2016-2019)**

* Các công cụ như Docker Compose, Docker Swarm và Kubernetes (mặc dù không phải của Docker) bắt đầu phát triển mạnh mẽ để quản lý container.
* Docker tiếp tục cập nhật và cải tiến, với các phiên bản mới mang lại nhiều tính năng và cải thiện hiệu suất

## 3.1.3 Các khái niệm quan trọng của Docker

Docker là một nền tảng mã nguồn mở giúp phát triển, vận chuyển và chạy ứng dụng trong các container. Dưới đây là một số khái niệm quan trọng của Docker:

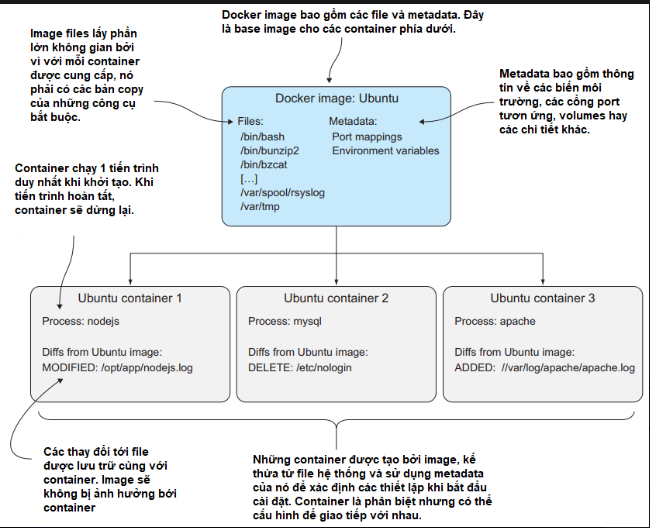
### 1. Container

Container là một đơn vị tiêu chuẩn hóa mà trong đó các ứng dụng và các phụ thuộc của chúng được đóng gói. Container chạy trên cùng một hệ điều hành nhưng tách biệt về môi trường, giúp ứng dụng chạy nhất quán trên nhiều môi trường khác nhau.

### 2. Image

Image là một mẫu (template) không thay đổi dùng để tạo ra container. Mỗi image chứa tất cả các tệp cần thiết để chạy ứng dụng, bao gồm mã nguồn, thư viện và các tệp cấu hình.

Nãy giờ chúng ta cứ đá qua đá lại 2 khái niệm này mà chưa hiểu rõ lắm, giờ chúng ta sẽ tìm hiểu qua chút về nó.

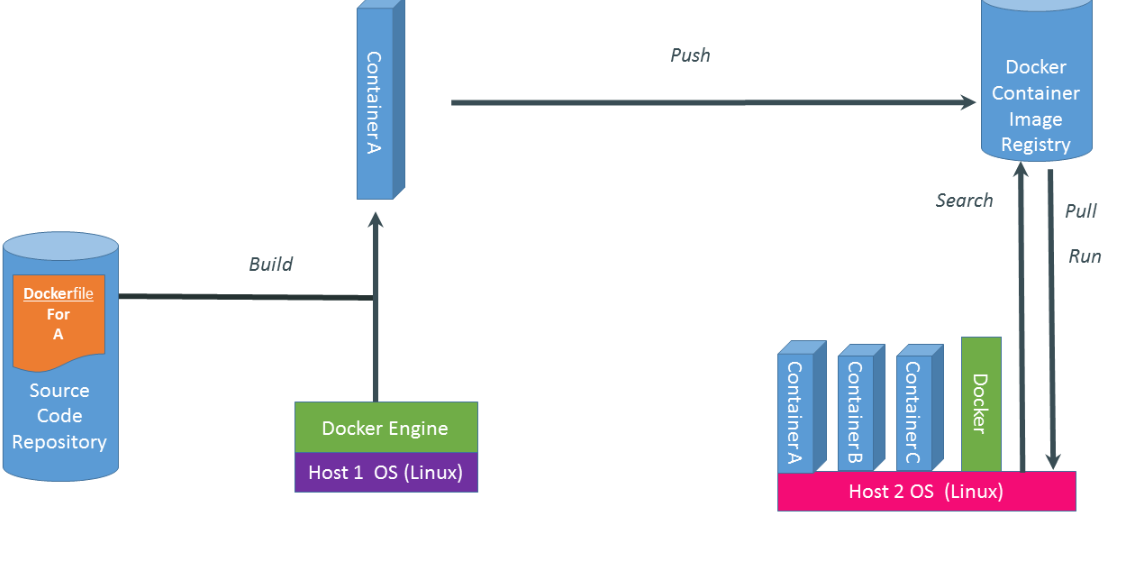


Hình vẽ giải thích khá rõ về Image và Container.

Nếu bạn đã quen với hướng đối tượng, chúng ta có thể hiểu images như những class còn cointaner là những object. Tương tự như việc các object được khởi tạo từ những class, container cũng được khởi tạo từ image. Bạn có thể tạo ra nhiều container từ 1 image, và chúng đều bị cô lập như cách các object khởi tạo. Bất kỳ thay đổi nào trên object, nó sẽ không ảnh hưởng đến định nghĩa và function của class.

## 3.1.4 Cách hoạt động của Docker

Docker sử dụng kiến ​​trúc client-server. Docker client sẽ liên lạc với các Docker daemon, các Docker daemon sẽ thực hiện các tác vụ build, run và phân phối các Docker container. Cả Docker client và Docker daemon đều có thể chạy trên cùng 1 máy hoặc có thể kết nối loại Dockẻ client điều khiển các docker daemon như hình dưới đây. Docker client và daemon giao tiếp với nhau thông qua socker hoặc RESTful API.



### a, Xây dựng.

Đầu tiên chúng ta sẽ tạo một dockerfile, trong dockerfile này chính là mã của chúng ta.

Dockerfile này sẽ được Build tại một máy tính đã cài đặt Docker Engine.

Sau khi xây dựng ta sẽ thu được Container, trong Container này chứa thư viện và ứng dụng của chúng ta.

### b, Đẩy.

Sau khi có được Container, chúng tôi thực hiện đẩy Container này lên đám mây và lưu trữ ở đó.

Việc đẩy này có thể thực thi qua môi trường mạng Internet.

### c, Kéo, Chạy

Giả sử một máy tính muốn sử dụng Container mà chúng tôi đã đẩy lên đám mây (máy đã cài đặt Docker Engine) thì bắt buộc máy phải thực hiện Kéo container này về máy. Sau đó thực hiện Run Container này.

Đó chính là quy trình 3 bước mô tả hoạt động của một hệ thống sử dụng Docker. Rất đơn giản và rõ ràng.

## 3.1.5 Quy trình hệ thống dùng Docker

Việc xây dựng và triển khai một hệ thống sử dụng Docker giúp tối ưu hóa quy trình phát triển, thử nghiệm và triển khai ứng dụng. Docker cho phép đóng gói ứng dụng và các phụ thuộc thành một container duy nhất, giúp việc di chuyển ứng dụng giữa các môi trường dễ dàng hơn.

### 1. Xây dựng Docker Image

Docker image chứa toàn bộ mã nguồn, thư viện, và các cấu hình cần thiết cho ứng dụng.

### 2. Chạy Container từ Docker Image

Sau khi tạo được image, bạn có thể chạy container từ image đó. Container là instance thực thi của image.

### 3. Quản lý Nhiều Container bằng Docker Compose

Docker Compose cho phép bạn chạy và quản lý nhiều container trong một ứng dụng phức tạp, ví dụ như khi bạn cần một container cho ứng dụng và một container cho cơ sở dữ liệu.

### 4. Thử nghiệm và Phát triển trong Docker

Trong quá trình phát triển, bạn có thể tạo môi trường Docker riêng để chạy thử nghiệm. Điều này giúp đảm bảo ứng dụng hoạt động như mong đợi trước khi triển khai thực tế.

### 5. Triển khai Docker Container lên Server

Khi phát triển xong, bạn cần triển khai ứng dụng của mình lên server (production environment)

### 6. Sử dụng Docker Swarm hoặc Kubernetes cho triển khai lớn

Đối với các ứng dụng cần triển khai trên nhiều máy chủ, bạn có thể dùng Docker Swarm hoặc Kubernetes để quản lý các container một cách linh hoạt.

### 7. Giám sát và Bảo trì Container

Trong môi trường production, bạn cần giám sát các container để đảm bảo hệ thống hoạt động ổn định. Có thể sử dụng các công cụ như:

* **Prometheus**: Giám sát và thu thập số liệu từ các container.
* **Grafana**: Hiển thị số liệu từ Prometheus hoặc các nguồn dữ liệu khác.
* **ELK Stack**: Để quản lý logs của container.

## \*Ưu nhược và nhược điểm của Docker

### Ưu điểm của Docker

**1.Tính di động (Portability)**: Docker giúp bạn đóng gói ứng dụng và các phụ thuộc của nó vào một container. Container có thể chạy trên bất kỳ môi trường nào (máy tính cá nhân, server, hay đám mây) mà không cần lo lắng về khác biệt hệ thống. Điều này giúp ứng dụng có thể dễ dàng di chuyển và triển khai trên nhiều môi trường khác nhau.

**2.Nhẹ và hiệu quả (Lightweight and Efficient)**: Docker container chỉ chia sẻ kernel của hệ điều hành với máy chủ nên chúng nhẹ hơn nhiều so với máy ảo (VM), và việc khởi chạy, tắt container nhanh chóng, tiêu tốn ít tài nguyên hệ thống hơn.

**3.Tính nhất quán và tái sử dụng (Consistency and Reproducibility)**: Docker đảm bảo rằng môi trường phát triển, thử nghiệm và triển khai luôn giống nhau, tránh các vấn đề “hoạt động trên máy tôi” (works on my machine) do sự khác biệt về cấu hình hệ thống.

**4.Tốc độ triển khai nhanh chóng (Fast Deployment)**: Việc khởi chạy container rất nhanh chóng, giúp giảm thời gian khởi động các dịch vụ và ứng dụng, đồng thời dễ dàng mở rộng quy mô bằng cách khởi chạy nhiều container cùng lúc.

**5. Hỗ trợ DevOps và CI/CD**: Docker hỗ trợ tốt các quy trình tích hợp liên tục (CI) và triển khai liên tục (CD). Các ứng dụng đóng gói bằng Docker có thể dễ dàng được tích hợp vào quy trình tự động, giúp tăng tính hiệu quả của việc phát triển và triển khai phần mềm.

**6.Hệ sinh thái phong phú**: Docker có một hệ sinh thái mạnh mẽ với Docker Hub, nơi bạn có thể tìm thấy hàng nghìn image đã được tạo sẵn từ cộng đồng, giúp tiết kiệm thời gian cài đặt và cấu hình các dịch vụ thông dụng (như database, web server, etc.).

**7.Cô lập ứng dụng (Isolation)**: Mỗi container chạy trong một không gian riêng biệt, giúp cô lập ứng dụng về tài nguyên và môi trường, đảm bảo các ứng dụng không ảnh hưởng lẫn nhau, đặc biệt trong trường hợp chạy nhiều ứng dụng trên cùng một máy chủ.

**8.Dễ mở rộng (Scalability)**: Docker giúp dễ dàng mở rộng ứng dụng thông qua việc chạy thêm các container. Đặc biệt trong các hệ thống phân tán lớn, Docker kết hợp với Kubernetes hoặc Docker Swarm giúp quản lý việc mở rộng theo yêu cầu một cách hiệu quả.

### Nhược điểm của Docker

**1.Hiệu suất không bằng máy chủ vật lý (Performance Overhead)**: Dù Docker container nhẹ hơn so với máy ảo, nhưng vì chia sẻ kernel với máy chủ nên Docker không phù hợp cho các ứng dụng yêu cầu hiệu suất cực kỳ cao và đòi hỏi nhiều tài nguyên. Máy chủ vật lý vẫn hiệu quả hơn cho các ứng dụng nặng.

**2.Không hoàn toàn thay thế máy ảo**: Docker không phải là giải pháp thay thế hoàn toàn cho máy ảo. Với những môi trường cần mức độ cô lập mạnh mẽ về kernel và tài nguyên, hoặc cần chạy nhiều hệ điều hành khác nhau trên cùng một máy chủ, Docker không thể thay thế hoàn toàn máy ảo.

3.Hỗ trợ hệ điều hành hạn chế: Docker chủ yếu hoạt động trên nhân Linux, và dù có thể chạy trên Windows và macOS, nhưng bản chất Docker trên các nền tảng này vẫn phải sử dụng một lớp ảo hóa để chạy nhân Linux bên dưới. Điều này có thể gây một số hạn chế và sự phức tạp trong cấu hình.

**4.Bảo mật (Security)**: Docker container chạy trên cùng một kernel với máy chủ, do đó một số lỗ hổng bảo mật có thể xảy ra nếu có lỗi trong kernel. Việc chạy nhiều container trên cùng một máy chủ có thể tạo ra rủi ro nếu một container bị xâm nhập.

**5.Quản lý container phức tạp**: Khi số lượng container tăng lên, việc quản lý và giám sát chúng trở nên phức tạp. Việc theo dõi logs, tài nguyên và đảm bảo tính ổn định cho toàn bộ hệ thống phân tán đòi hỏi cần có các công cụ hỗ trợ bổ sung như Kubernetes hoặc Docker Swarm.

**6.Yêu cầu hiểu biết về cấu trúc hệ thống**: Docker dễ sử dụng cho các ứng dụng nhỏ, nhưng đối với các hệ thống lớn và phức tạp, việc sử dụng Docker đòi hỏi hiểu biết sâu về hạ tầng hệ thống, mạng, và bảo mật để thiết lập và quản lý một cách hiệu quả.

**7.Công nghệ còn tương đối mới**: Docker là một công nghệ tương đối mới (so với các giải pháp ảo hóa truyền thống), nên có thể gặp phải những vấn đề về hỗ trợ kỹ thuật, hoặc không tương thích với một số công nghệ cũ hơn.

## 3.2 Công cụ kết hợp với Docker

### 3.2.1 Docker Compose

### Docker Compose là gì?

Docker Compose là một công cụ hỗ trợ xác định và chạy các ứng dụng multi-container . Docker Compose có thể xử lý đồng thời multi-container trong sản xuất, staging, phát triển, thử nghiệm và CI

### Các thành phần chính của Docker Compose

1. **docker-compose.yml**:
   * Đây là file cấu hình chính cho Docker Compose, nơi bạn mô tả các dịch vụ (services) cần thiết cho ứng dụng. Mỗi service đại diện cho một container.
   * Bạn có thể định nghĩa môi trường, volume, mạng, và các thông số khác trong file này.
2. **Lệnh Docker Compose**:
   * **docker-compose up**: Khởi chạy tất cả các container được định nghĩa trong file docker-compose.yml.
   * **docker-compose down**: Dừng và xóa tất cả container, network, volume, v.v. được tạo bởi Docker Compose.
   * **docker-compose start**: Khởi động lại các container đã dừng.
   * **docker-compose stop**: Dừng tất cả các container mà không xóa chúng.
   * **docker-compose build**: Build lại các image được định nghĩa trong docker-compose.yml mà không cần chạy container.
3. **Mạng và Volume**:
   * Docker Compose cho phép bạn thiết lập mạng và volume để các container có thể giao tiếp với nhau và lưu trữ dữ liệu một cách dễ dàng.

### Các tính năng chính của Docker Compose

1. **Đa container**: Docker Compose giúp bạn dễ dàng quản lý và chạy nhiều container cùng một lúc. Bạn chỉ cần định nghĩa các container trong file docker-compose.yml.
2. **Môi trường tích hợp**: Bạn có thể định nghĩa các biến môi trường cho từng container hoặc dùng file .env để giữ các biến này bên ngoài file cấu hình chính.
3. **Mạng tự động**: Docker Compose tự động tạo các network để các container trong cùng một hệ thống có thể giao tiếp với nhau mà không cần cấu hình phức tạp.
4. **Volume**: Docker Compose hỗ trợ việc sử dụng volume để giữ dữ liệu ngay cả khi container bị xóa hoặc khởi động lại.
5. **Dependencies**: Bạn có thể định nghĩa thứ tự phụ thuộc giữa các container (sử dụng depends\_on), để đảm bảo container này chỉ khởi chạy khi container kia đã sẵn sàng.

### 3.2.2 DockerHub.

### Docker Hub là gì?

Docker Hub là một nền tảng lưu trữ và chia sẻ các Docker images, tương tự như một "kho lưu trữ" cho các images Docker. Nó là dịch vụ chính thức từ Docker để lưu trữ và phân phối các images. Docker Hub cho phép bạn tìm kiếm, tải xuống, và đẩy (push) Docker images lên một repository (repo). Bạn có thể sử dụng Docker Hub để chia sẻ Docker images với cộng đồng hoặc giữa các nhóm phát triển, cũng như lưu trữ riêng các images của mình để sử dụng cho việc phát triển và triển khai.

### Các tính năng chính của Docker Hub

1. **Public và Private Repositories**:
   * **Public repositories**: Các repo công khai cho phép bất kỳ ai cũng có thể tải xuống và sử dụng các images của bạn.
   * **Private repositories**: Repo riêng tư chỉ có thể được truy cập bởi bạn hoặc những người mà bạn cấp quyền.
2. **Official Images**:
   * Docker Hub cung cấp một danh sách các **Official Images** được duy trì bởi Docker và các nhà phát triển phần mềm lớn. Đây là các images chuẩn của các ứng dụng phổ biến như MySQL, Redis, Node.js, Python, Nginx,... Các official images thường được tối ưu hóa và kiểm tra để đảm bảo an toàn và hiệu suất.
3. **Automated Builds**:
   * Docker Hub cung cấp tính năng **Automated Builds**, cho phép bạn tự động build Docker image mỗi khi mã nguồn của bạn được cập nhật trong một repository như GitHub hoặc Bitbucket. Điều này giúp đảm bảo image luôn được cập nhật với mã mới nhất.
4. **Docker Hub Registry**:
   * Docker Hub đóng vai trò là một **registry** nơi bạn có thể lưu trữ các image Docker. Bạn có thể kéo (pull) các image từ Docker Hub về máy local hoặc đẩy (push) các image từ máy local lên Docker Hub để chia sẻ.
5. **Webhooks**:
   * Docker Hub hỗ trợ **webhooks**, cho phép bạn tự động kích hoạt các hành động khi một image mới được đẩy lên repo. Ví dụ, bạn có thể thiết lập một webhook để tự động triển khai image mới nhất lên server.
6. **Tích hợp CI/CD**:
   * Docker Hub có thể tích hợp dễ dàng với các công cụ CI/CD như Jenkins, Travis CI, GitLab CI, v.v., giúp tự động hóa quy trình build và triển khai container.

### 3. 2.3 Kubernetes

### 1.Kubernetes và Docker

Kubernetes và Docker là hai công nghệ liên quan chặt chẽ trong lĩnh vực quản lý container, nhưng chúng không hoàn toàn giống nhau và phục vụ các mục đích khác nhau. Docker được sử dụng để **container hóa ứng dụng**, còn Kubernetes là một công cụ **orchestration** dùng để quản lý các container (bao gồm cả Docker containers) ở quy mô lớn.

### 2.Cách Kubernetes và Docker hoạt động cùng nhau

* **Docker** là nền tảng dùng để xây dựng, đóng gói và chạy các container. Trong môi trường Kubernetes, Docker là một trong những **Container Runtime** (môi trường chạy container) được hỗ trợ. Điều này có nghĩa là Kubernetes sử dụng Docker để chạy các container, nhưng nó không phụ thuộc duy nhất vào Docker, vì Kubernetes có thể làm việc với các container runtime khác như **containerd** hoặc **CRI-O**.
* **Kubernetes quản lý Docker containers**: Khi bạn triển khai một ứng dụng lên Kubernetes, Kubernetes sẽ sử dụng Docker để khởi động các container, sau đó quản lý vòng đời, khả năng mở rộng, và đảm bảo rằng các container này luôn hoạt động đúng cách.

### 3.Các thành phần tương tự và khác biệt giữa Kubernetes và Docker

| **Thành phần** | **Docker** | **Kubernetes** |
| --- | --- | --- |
| **Orchestration** | Docker Compose (nhỏ, không có khả năng mở rộng lớn) | Kubernetes là hệ thống orchestration mạnh mẽ và phức tạp, hỗ trợ scaling tự động, cân bằng tải, tự phục hồi. |
| **Container Runtime** | Docker Engine | Kubernetes có thể sử dụng nhiều container runtime khác nhau (bao gồm Docker Engine, containerd, CRI-O). |
| **Deployment** | Docker sử dụng Dockerfiles và docker-compose.yml để mô tả cách xây dựng và chạy container. | Kubernetes sử dụng các đối tượng như Pod, Deployment, Service để quản lý các ứng dụng container. |
| **Scaling** | Docker Compose không có khả năng scaling tự động. | Kubernetes có khả năng scaling tự động các Pod (horizontal scaling) dựa trên nhu cầu tài nguyên. |
| **Networking** | Docker sử dụng mạng mặc định hoặc do người dùng cấu hình thủ công (bridge, host, overlay). | Kubernetes có hệ thống mạng phức tạp, tự động tạo ra các mạng nội bộ giữa các Pod và Service. |
| **State Management** | Docker không có khả năng quản lý trạng thái của container nếu bị lỗi. | Kubernetes có khả năng tự động khởi động lại các container bị lỗi, duy trì trạng thái mong muốn cho ứng dụng. |
| **Service Discovery** | Docker có thể cần công cụ bổ sung để thực hiện việc tìm kiếm dịch vụ (ví dụ: Consul, etcd). | Kubernetes có sẵn dịch vụ tự động cho việc khám phá các Pod và Service thông qua DNS. |

### 4.Kết luận

* **Docker** cung cấp nền tảng để xây dựng, đóng gói và chạy container, giúp đảm bảo ứng dụng hoạt động đồng nhất trong mọi môi trường.
* **Kubernetes** giúp quản lý, triển khai và điều phối hàng loạt container, đảm bảo khả năng mở rộng và tính sẵn sàng cao trong các hệ thống phân tán.

### 3.2.4 Jenkins

## 

## 3.3. Các câu lệnh của Docker

| **Lệnh** | **Mô tả** |
| --- | --- |
| **docker images** | **Liệt kê các images hiện có** |
| **docker pull image\_name** | **Pull một image từ Docker Hub** |
| **docker rmi {image\_id/name}** | **Xóa một image** |
| **docker run -v <forder\_in\_computer>:<forder\_in\_container> -p <port\_in\_computer> <port\_in\_container> -it <image\_name> /bin/bash** | **Tạo mới container bằng cách chạy image (kèm theo các tùy chọn)** |
| **docker ps** | **Liệt kê các container đang chạy** |
| **docker ps -a** | **Liệt kê các container đã tắt** |
| **docker rm -f {container\_id/name}** | **Xóa một container** |
| **docker start {new\_container\_name}** | **Khởi động một container** |
| **docker exec -it {new\_container\_name} /bin/bash** | **Truy cập vào container đang chạy** |
| **uname -a** | **Hiển thị thông tin của Kernel ubuntu** |

### 

### 

### 

### 

### 

### 

### 

## cài đặt Docker trên Linux, Window, MacOs

# CHƯƠNG 4 TRIỂN KHAI DEMO

**4.1 Mô hình và Kỹ thuật phân nhánh được sử dụng.**

**4.1.1 Mô hình phân nhánh GitFlow**

Mô hình phân nhánh Gitflow là một phương pháp quản lý mã nguồn dựa trên Git, được phát triển bởi Vincent Driessen. Mô hình này cung cấp một cách cấu trúc hóa và quản lý quy trình phát triển phần mềm, bao gồm các quy tắc và quy trình rõ ràng cho việc tạo, quản lý và hợp nhất các nhánh. Dưới đây là một sơ đồ và mô tả chi tiết về các nhánh và quy trình trong mô hình phân nhánh Gitflow:

**1. Nhánh Master:**

Nhánh master là nhánh chính của dự án, chứa mã nguồn ổn định đã được kiểm tra và sẵn sàng triển khai.

Mọi thay đổi từ nhánh release và hotfix được hợp nhất vào nhánh này sau khi kiểm tra và hoàn thành.

**2. Nhánh Develop:**

Nhánh develop là nơi phát triển chính của dự án, chứa mã nguồn mới nhất và các tính năng đang được phát triển.

Mọi tính năng mới được phát triển từ nhánh này.

**3. Nhánh Feature:**

Mỗi tính năng mới hoặc thay đổi lớn được phát triển trên một nhánh tính năng riêng.

Nhánh tính năng được tạo từ develop và sau khi hoàn thành, được hợp nhất trở lại vào develop.

**4. Nhánh Release:**

Khi chuẩn bị cho một bản phát hành mới, một nhánh release được tạo từ develop.

Trên nhánh release, các công việc như kiểm tra, sửa lỗi và chuẩn bị cho bản phát hành được thực hiện.

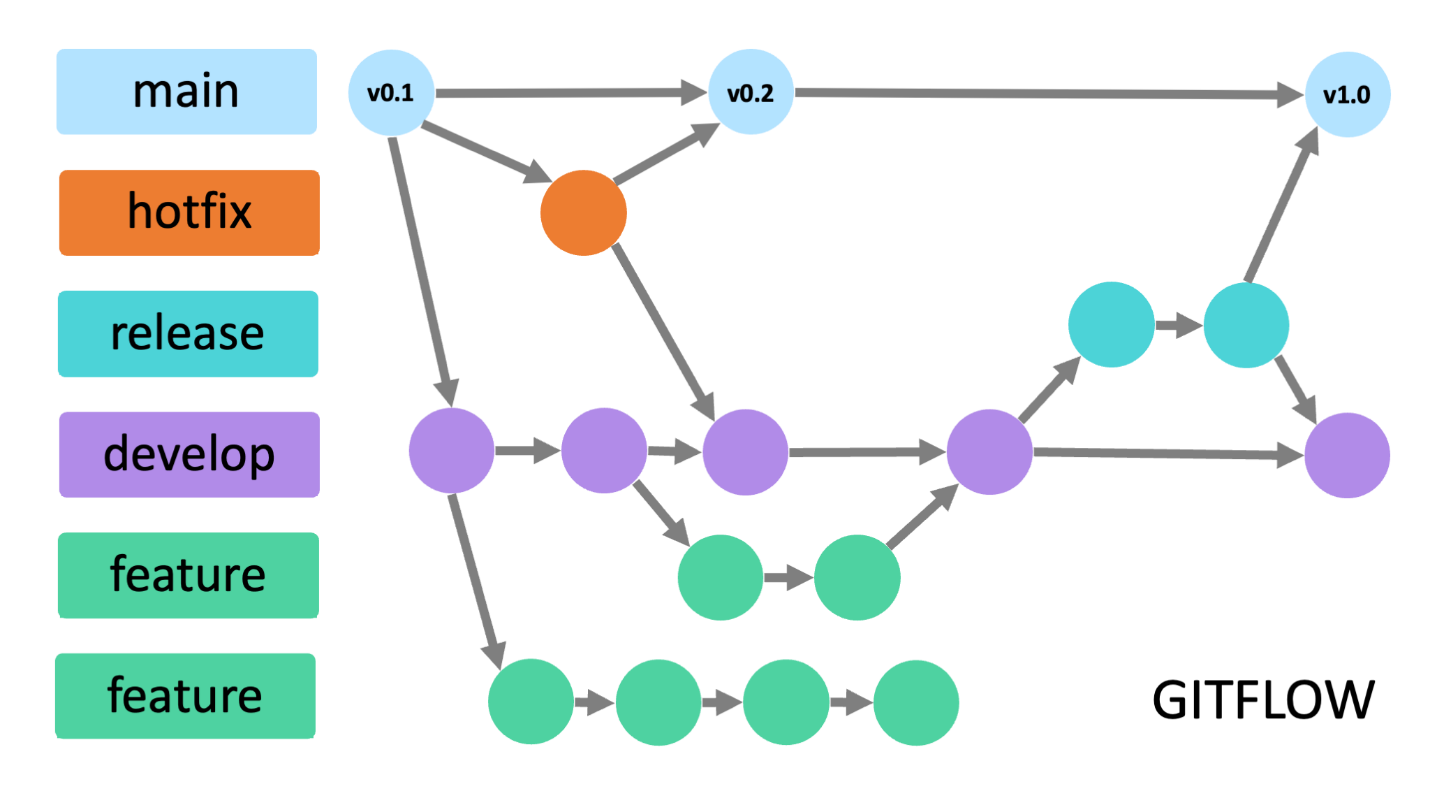
Sau khi hoàn thành, nhánh release được hợp nhất vào cả master và develop.

**5. Nhánh Hotfix:**

Để xử lý các lỗi gấp, một nhánh hotfix được tạo từ nhánh master.

Trên nhánh hotfix, các sửa đổi cần thiết được thực hiện và sau đó hợp nhất vào cả master và develop.

**Sơ đồ mô hình GitFlow :**



**Lợi Ích:**

Quản lý Mã nguồn Tốt hơn: Mô hình Gitflow cung cấp cách cấu trúc và tổ chức mã nguồn rõ ràng, giúp quản lý dự án hiệu quả hơn.

Tách Biệt Công việc: Các nhánh riêng biệt cho tính năng, bản phát hành và sửa lỗi giúp tách biệt các công việc và giảm xung đột.

Kiểm soát Quy trình Phát triển: Mỗi bước trong quy trình phát triển (phát triển, kiểm tra, phát hành, sửa lỗi) được xác định rõ ràng và kiểm soát chặt chẽ.

**4.1.2 Chi tiết kỹ thuật phân nhánh GitFlow**

**1. Tạo Nhánh (Branching):**

Tạo Nhánh: Sử dụng lệnh git branch <tên\_nhánh> để tạo một nhánh mới từ nhánh hiện tại. Ví dụ: git branch feature/new-feature.

Chuyển Nhánh: Sử dụng lệnh git checkout <tên\_nhánh> để chuyển đổi giữa các nhánh.

Ví dụ: git checkout feature/new-feature.

Tạo và Chuyển Nhánh: Bạn cũng có thể sử dụng lệnh git checkout -b <tên\_nhánh> để tạo và chuyển đến một nhánh mới trong một lệnh duy nhất.

**2. Làm việc trên Nhánh (Working on Branch):**

Thêm và Commit: Thực hiện các thay đổi trong mã nguồn của bạn trên nhánh của bạn. Sử dụng git add để thêm các tệp tin đã thay đổi và git commit để lưu các thay đổi đó vào lịch sử commit.

Push Nhánh: Khi bạn đã thực hiện các commit trên nhánh của mình, bạn có thể đẩy nhánh đó lên repository từ xa bằng cách sử dụng git push origin <tên\_nhánh>.

**3. Hợp Nhất (Merging) và Tạo Pull Request:**

Hợp Nhất Nhánh: Khi bạn đã hoàn thành công việc trên nhánh của mình, bạn có thể hợp nhất nhánh đó vào nhánh chính (ví dụ: master) bằng cách sử dụng git merge.

Tạo Pull Request (PR): Trên GitHub, bạn có thể tạo một Pull Request từ nhánh của mình vào nhánh chính của dự án. Điều này cho phép các thành viên khác trong nhóm xem xét và đánh giá công việc của bạn trước khi hợp nhất vào nhánh chính.

**4. Xử lý xung đột (Handling Conflicts):**

Xung đột: Nếu có xung đột giữa các thay đổi trên nhánh của bạn và nhánh mục tiêu, Git sẽ báo lỗi. Bạn cần giải quyết xung đột này bằng cách chỉnh sửa tệp tin và sau đó thêm và commit lại các thay đổi.

**5. Xóa Nhánh (Deleting Branch):**

Xóa Nhánh Cục Bộ: Sử dụng lệnh git branch -d <tên\_nhánh> để xóa một nhánh cục bộ sau khi đã hợp nhất.

Xóa Nhánh Từ Xa: Trên GitHub, sau khi đã hợp nhất một Pull Request, bạn có thể chọn xóa nhánh từ xa mà Pull Request đó đến từ.

**4.2 Áp dụng mô hình GitFlow vào dự án**

**1. Khởi Tạo Repository và Thiết Lập Cấu Trúc Nhánh**

**1.1. Khởi Tạo Repository:**

* Tạo repository trên GitHub cho dự án của bạn.
* Sao chép URL của repository để sử dụng trong việc clone dự án xuống máy tính cá nhân hoặc máy chủ.

**1.2. Clone Repository:**

* Sử dụng câu lệnh git clone <repository\_url> để sao chép repository xuống máy tính cá nhân.

**1.3. Thiết Lập Cấu Trúc Nhánh:**

* Tạo nhánh master: Nhánh chứa mã nguồn ổn định, sẵn sàng cho việc triển khai.
* Tạo nhánh develop: Nhánh phát triển chứa mã nguồn mới nhất, được tích hợp từ các nhánh tính năng.
* Sử dụng câu lệnh git checkout -b develop để tạo và chuyển sang nhánh develop.

**2. Phát Triển Tính Năng**

**2.1. Tạo Nhánh Tính Năng:**

* Tạo một nhánh mới từ nhánh develop cho mỗi tính năng cụ thể bằng câu lệnh git checkout -b feature/<feature\_name>.

**2.2. Phát Triển Tính Năng:**

* Phát triển tính năng trên nhánh tính năng vừa tạo.
* Thực hiện commit thường xuyên và đồng bộ hóa với nhánh develop bằng câu lệnh git push origin feature/<feature\_name>.

**2.3. Hoàn Thiện Tính Năng:**

* Khi tính năng hoàn thành, thực hiện kiểm tra và kiểm thử kỹ lưỡng.

**3. Chuẩn Bị Cho Phát Hành**

**3.1. Tạo Nhánh Release:**

* Tạo một nhánh từ develop gọi là release/\* bằng câu lệnh git checkout -b release/<version\_number>.

**3.2. Chuẩn Bị Phiên Bản:**

* Tiến hành kiểm tra tổng thể, cập nhật tài liệu, và chuẩn bị cho việc phát hành.

**4. Phát Hành Phiên Bản**

**4.1. Merge Nhánh Release vào Master và Develop:**

* Merge nhánh release/\* vào master và develop bằng các câu lệnh git merge release/<version\_number>.

**4.2. Triển Khai Phiên Bản:**

* Triển khai phiên bản đã phát hành lên môi trường sản phẩm.

**4.3. Đặt Tag Cho Phiên Bản:**

* Đặt tag cho phiên bản phát hành trên nhánh master bằng câu lệnh git tag -a <version\_number> -m "Release <version\_number>".

**5. Sửa Lỗi và Bảo Trì**

**5.1. Tạo Nhánh Hotfix:**

* Nếu có sự cố hoặc lỗi phát sinh, tạo một nhánh từ master gọi là hotfix/\* bằng câu lệnh git checkout -b hotfix/<fix\_name>.

**5.2. Sửa Lỗi:**

* Sửa lỗi trong nhánh hotfix/\* và sau đó merge vào cả master và develop để đồng bộ hóa.

**6. Quản Lý Thay Đổi**

* Đảm bảo sử dụng message commit có ý nghĩa và theo chuẩn của Gitflow.
* Sử dụng các công cụ quản lý mã nguồn như GitHub để theo dõi và kiểm soát quy trình làm việc theo Gitflow.

**4.3 Kế hoạch bảo trì phần mềm**

Mục tiêu:

Mục tiêu của kế hoạch này là duy trì tính ổn định và cải thiện chức năng của phần mềm qua các phiên bản bảo trì liên tục. Các bản cập nhật sẽ tập trung vào việc sửa lỗi, cải thiện tính năng và hiệu suất, đồng thời đáp ứng nhu cầu của người dùng.

Thành viên nhóm:

+ Nhóm phát triển Giao diện(FontEnd): Tuấn, Diệp, Hội, Dương, Vũ, Tuyến.

+ Nhóm phát triển BackEnd: Tuấn.

Phiên bản 1.0 (Tháng 3, 2024)

Mô tả:

Cập nhật giao diện người dùng để tăng tính tương tác và trải nghiệm người dùng.

Sửa lỗi liên quan đến hiệu suất và tính năng không hoạt động đúng cách.

Chi tiết công việc:

Phát triển giao diện người dùng mới:

Mô tả: Cải thiện giao diện người dùng để tăng tính tương tác và thu hút người dùng.

Phân công: Nhóm phát triển giao diện - Amanda và John.

Thời gian: Dự kiến 2 tuần.

Nhánh: feature/ui-update.

Sửa lỗi hiệu suất:

Mô tả: Tìm kiếm và sửa lỗi liên quan đến hiệu suất của ứng dụng để tăng tốc độ và đáp ứng.

Phân công: Nhóm phát triển backend - David và Emily.

Thời gian: Dự kiến 1 tuần.

Nhánh: hotfix/performance-issue.

*Phiên bản 1.1 (Tháng 4, 2024)*

· **Mô tả:**

· Cập nhật giao diện người dùng để tăng tính tương tác và trải nghiệm người dùng.

· Sửa lỗi liên quan đến hiệu suất và tính năng không hoạt động đúng cách.

· **Chi tiết công việc:**

· Phát triển giao diện người dùng mới:

· Mô tả: Cải thiện giao diện người dùng để tăng tính tương tác và thu hút người dùng.

· Phân công: Nhóm phát triển giao diện - Amanda và John.

· Thời gian: Dự kiến 2 tuần.

· Nhánh: feature/ui-update.

· Sửa lỗi hiệu suất:

· Mô tả: Tìm kiếm và sửa lỗi liên quan đến hiệu suất của ứng dụng để tăng tốc độ và đáp ứng.

· Phân công: Nhóm phát triển backend - David và Emily.

· Thời gian: Dự kiến 1 tuần.

· Nhánh: hotfix/performance-issue.

*Phiên bản 1.2 (Tháng 6, 2024)*

· **Mô tả:**

· Thêm tính năng đăng nhập bằng các tài khoản mạng xã hội (Facebook, Google).

· Cải thiện tính năng tìm kiếm.

· **Chi tiết công việc:**

· Phát triển tính năng đăng nhập bằng mạng xã hội:

· Mô tả: Thêm tính năng cho phép người dùng đăng nhập bằng tài khoản mạng xã hội để tăng tính tiện ích.

· Phân công: Nhóm phát triển backend - David và Emily.

· Thời gian: Dự kiến 2 tuần.

· Nhánh: feature/social-login.

· Cải thiện tính năng tìm kiếm:

· Mô tả: Tối ưu hóa tính năng tìm kiếm để cung cấp kết quả chính xác và nhanh chóng hơn cho người dùng.

· Phân công: Nhóm phát triển frontend - Amanda và John.

· Thời gian: Dự kiến 1 tuần.

· Nhánh: feature/search-improvement.

*Phiên bản 1.3 (Tháng 8, 2024)*

· **Mô tả:**

· Tối ưu hóa dữ liệu và cơ sở dữ liệu để tăng hiệu suất.

· Sửa lỗi và cải thiện tính ổn định.

· **Chi tiết công việc:**

· Tối ưu hóa cơ sở dữ liệu:

· Mô tả: Kiểm tra và tối ưu hóa cơ sở dữ liệu để tăng hiệu suất và sự ổn định của hệ thống.

· Phân công: Nhóm DevOps - Michael và Sarah.

· Thời gian: Dự kiến 2 tuần.

· Nhánh: feature/database-optimization.

· Kiểm tra và sửa lỗi tự động:

· Mô tả: Thiết lập hệ thống kiểm tra tự động để phát hiện và sửa lỗi tự động.

· Phân công: Nhóm kiểm thử - Daniel và Lisa.

· Thời gian: Dự kiến 1 tuần.

· Nhánh: feature/automated-testing.

Tổng Kết

Kế hoạch bảo trì phần mềm được thiết kế để cải thiện và bổ sung các tính năng của trang web giới thiệu việc làm một cách có tổ chức và hiệu quả.

Công việc đã được phân công cho từng nhóm nhân viên với sự phân chia công việc rõ ràng, đảm bảo tiến độ và chất lượng của từng phiên bản.

Mỗi tính năng mới và sửa đổi được thực hiện trên các nhánh riêng biệt để quản lý mã nguồn dễ dàng và giảm thiểu xung đột.

**4.3 Thống kê, theo dõi quy trình bảo trì qua lịch sử commit**

Báo cáo Tổng kết Quá trình Bảo trì - Website Mua sắm Trực tuyến "Shoppe"

Thời gian: Tháng 3 - Tháng 4, 2024

Tổng quan:

Trong khoảng thời gian bảo trì này, chúng tôi đã tiến hành nhiều sửa đổi và cải thiện trên trang web mua sắm trực tuyến "Shoppe". Dưới đây là một tóm tắt về những gì đã được thực hiện:

Các Thay Đổi:

Tính năng mới:

Tổng số tính năng mới được thêm vào: 3

Mô tả tóm tắt về các tính năng mới:

Tính năng "Đánh giá và Nhận xét Sản phẩm": Cho phép người dùng đánh giá và viết nhận xét về sản phẩm mà họ đã mua.

Tính năng "Gợi ý Mua Hàng Dựa trên Lịch Sử": Gợi ý sản phẩm dựa trên lịch sử mua hàng của người dùng.

Tính năng "Thanh toán bằng Ví Điện Tử": Hỗ trợ thanh toán nhanh chóng và thuận tiện bằng ví điện tử.

Sửa lỗi:

Tổng số lỗi đã sửa: 10

Mô tả tóm tắt về các lỗi đã sửa: Bao gồm sửa lỗi hiển thị sản phẩm không đồng nhất, lỗi tương tác với các nút chức năng, và sửa lỗi liên quan đến thanh toán.

Cải thiện hiệu suất:

Tổng số cải thiện về hiệu suất: 5

Mô tả tóm tắt về các cải thiện hiệu suất: Bao gồm tối ưu hóa tốc độ tải trang, cải thiện thời gian phản hồi và tăng cường bảo mật.

Số lượng Commit:

Tổng số lượng commit: 50

Phân loại commit:

Tính năng mới: 20 commit

Sửa lỗi: 20 commit

Cải thiện hiệu suất: 10 commit

Đánh giá Tổng quan:

Quá trình bảo trì đã đạt được một số kết quả tích cực, bao gồm việc thêm tính năng mới, sửa lỗi và cải thiện hiệu suất. Số lượng commit cho thấy sự hoạt động tích cực và cam kết của nhóm trong việc nâng cấp và bảo trì trang web mua sắm trực tuyến "Shoppe".

Đề Xuất Tiếp Theo:

Tiếp tục theo dõi và xử lý các lỗi và yêu cầu từ người dùng.

Tiếp tục phát triển tính năng mới để cải thiện trải nghiệm người dùng.

Tổ chức các phiên bản bảo trì định kỳ để duy trì tính ổn định và an toàn của trang web.

**4.4 Danh sách các lệnh git được sử dụng**

Dưới đây là một danh sách các lệnh Git mà bạn có thể sử dụng trong quá trình demo để thực hiện các tác vụ cơ bản như tạo nhánh, hợp nhất các nhánh, xem lịch sử commit và xem trạng thái hiện tại của dự án:

* git init: Khởi tạo một kho chứa Git mới trong thư mục hiện tại.
* git clone <url>: Sao chép một kho chứa Git từ một URL.
* git status: Xem trạng thái hiện tại của thư mục làm việc và các tệp tin đã thay đổi.
* git add <tên\_tệp>: Thêm các thay đổi trong tệp tin cụ thể vào chỉ mục chuẩn bị.
* git commit -m "Thông điệp commit": Tạo một commit mới với các thay đổi trong chỉ mục chuẩn bị và một thông điệp mô tả.
* git branch: Liệt kê tất cả các nhánh có trong kho chứa.
* git checkout -b <tên\_nhánh>: Tạo và chuyển đến một nhánh mới.
* git checkout <tên\_nhánh>: Chuyển đến một nhánh tồn tại.
* git merge <tên\_nhánh>: Hợp nhất nhánh hiện tại với nhánh đã chỉ định.
* git log: Hiển thị lịch sử commit trên nhánh hiện tại.
* git show <commit\_id>: Hiển thị thông tin chi tiết về một commit cụ thể.
* git pull: Cập nhật các thay đổi từ remote repository về local repository.
* git push: Đẩy các thay đổi từ local repository lên remote repository.
* git remote -v: Hiển thị thông tin về các remote repository được liên kết với local repository.
* git fetch: Lấy tất cả các thay đổi từ remote repository mà không thực hiện hợp nhất.
* git tag: Quản lý các tag (nhãn) trên các commit.

**4.5 Kết quả**

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Git và GitHub là hai công cụ mạnh mẽ giúp chúng em quản lý và theo dõi mã nguồn trong dự án phần mềm của mình một cách hiệu quả. Bằng cách sử dụng các tính năng như quản lý phiên bản, hợp nhất và theo dõi vấn đề, chúng em đã có thể cải thiện quá trình phát triển và hợp tác trong dự án của mình.

Sử dụng Git và GitHub trong dự án của chúng em đã đem lại nhiều lợi ích đáng kể và kết quả tích cực:

* Quản Lý Phiên Bản Hiệu Quả: Sử dụng Git cho phép chúng em quản lý lịch sử thay đổi của mã nguồn một cách linh hoạt và an toàn. Điều này giúp chúng em dễ dàng theo dõi sự thay đổi trong mã nguồn, quay lại phiên bản trước đó, và giải quyết xung đột một cách dễ dàng.
* Hợp Nhất Tính Năng Một Cách Cẩn Thận: Sử dụng GitHub, các pull request được sử dụng để kiểm tra và xác nhận tính năng mới trước khi hợp nhất vào nhánh chính. Quy trình này giúp đảm bảo tính ổn định và chất lượng của mã nguồn.
* Theo Dõi Và Giải Quyết Vấn Đề: Sử dụng tính năng theo dõi vấn đề trên GitHub giúp chúng em ghi lại và giải quyết các vấn đề hoặc yêu cầu tính năng một cách có tổ chức. Điều này tăng cường sự linh hoạt và phản hồi trong quá trình phát triển.

**Kết Quả Đạt Được**

Sử dụng Git và GitHub đã mang lại những kết quả tích cực cho dự án của chúng em:

* Tăng Cường Hợp Tác Trong Nhóm: Quy trình làm việc phân tán trên Git và GitHub đã tăng cường sự hợp tác và giao tiếp giữa các thành viên trong nhóm phát triển. Điều này đã giúp cải thiện hiệu suất làm việc và chất lượng mã nguồn.
* Kiểm Soát Và Quản Lý Mã Nguồn Hiệu Quả: Sử dụng các tính năng của Git và GitHub, chúng em đã có thể duy trì một mã nguồn ổn định và dễ quản lý. Cơ chế kiểm soát phiên bản và quản lý pull request giúp đảm bảo rằng chỉ những thay đổi chất lượng được hợp nhất vào mã nguồn chính.
* Tăng Hiệu Suất Phát Triển: Quy trình làm việc linh hoạt và hiệu quả trên Git và GitHub đã giúp tăng cường hiệu suất phát triển của dự án. Việc giảm thiểu xung đột và tối ưu hóa quy trình làm việc đã dẫn đến việc hoàn thành các tính năng nhanh chóng hơn và giảm thiểu lỗi phát sinh.

**Hướng Phát Triển**

Dựa trên kết quả và kinh nghiệm tích lũy từ việc sử dụng Git và GitHub trong dự án hiện tại, chúng em đề xuất một số hướng phát triển tiếp theo:

* Tối Ưu Hóa Quy Trình Làm Việc: Tiếp tục đánh giá và tối ưu hóa quy trình làm việc trên Git và GitHub để đảm bảo sự linh hoạt và hiệu quả cao nhất.
* Đào Sâu Vào Các Tính Năng Của GitHub: Nắm vững các tính năng nâng cao của GitHub như Projects, Wiki, và Actions để tối ưu hóa quy trình làm việc và tăng cường hiệu suất.
* Học Hỏi Và Chia Sẻ Kinh Nghiệm: Tổ chức các buổi workshop hoặc buổi đào tạo để chia sẻ kinh nghiệm sử dụng Git và GitHub với các thành viên mới trong nhóm và tổ chức.
* Mở Rộng Sử Dụng Cho Các Dự Án Khác: Áp dụng kiến thức và kinh nghiệm từ dự án hiện tại để mở rộng việc sử dụng Git và GitHub cho các dự án khác trong tổ chức, từ đó tăng cường sự phát triển và hiệu suất làm việc.

# TÀI LIỆU THAM KHẢO

[1] Giáo Trình Và Slide Môn Vận Hành và Bảo Trì Phần Mềm – Phạm Thị Thương

[2] <https://git-scm.com/doc>

[3] <https://rogerdudler.github.io/git-guide/>

[4] <https://chat.openai.com/>

[5] <https://www.youtube.com/watch?v=1EmOFN0oyks>

[6]<https://memart.vn/tin-tuc/blog/tim-hieu-github-la-gi-cach-su-dung-de-quan-ly-ma-nguon-va-cac-du-an-phat-trien-vi-cb.html>

[7] <https://vietjack.com/git/tai_lieu_tham_khao_ve_git.jsp>

[8] <https://freetuts.net/hoc-git>

[9] <https://codelearn.io/sharing/git-github-tu-co-ban-den-nang-cao-p1>