# **BÁO CÁO**

# **GIỚI THIỆU GIT**

## **GIT là gì?**

Là một hệ thống quản lý phiên bản phân tán(Distributed Version Control System-DVCS)ra đời vào năm 2005 và hiện được dùng rất phổ biến.So với các hệ thống quản lý phiên bản tập trung khi tất cả các mã nguồn và lịch sử thay đổi chỉ được lưu một nơi là máy chủ thì trong hệ thống phân tán, các máy không chỉ "check out" phiên bản mới nhất của các tập tin mà là sao chép(mirror)toàn bộ kho mã nguồn(repository).Như vậy, nếu như máy chủ ngừng hoạt động , thì bạn hoàn toàn có thể lấy kho chứa từ bắt kì máy khách nào để sao chép ngược trở lại máy chủ để khôi phục lại toàn bộ hệ thống.Mỗi checkout thực sự là một bản sao đầy đủ của tất cả dữ liệu của kho chưá từ máy chủ.

## **GIT có tác dụng như thế nào?**

1. Lưu lại lịch sử các version của bất kỳ thay đổi nào của dự án. Giúp xem lại các sự thay đổi hoặc khôi phục (revert) lại sau này.
2. Việc chia sẻ code trở nên dễ dàng hơn, lập trình viên có thể để public cho bất kỳ ai, hoặc private chỉ cho một số người có thẩm quyền có thể truy cập và lấy code về.

Vốn là một VCS nên Git cũng ghi nhớ lại toàn bộ lịch sử thay đổi của source code trong dự án. Lập trình sửa file, thêm dòng code tại đâu, xóa dòng code ở hàng nào…đều được Git ghi nhận và lưu trữ lại.

## **GIT hoạt động như thế nào?**

Sự khác biệt chính giữa Git và bất kỳ VCS nào khác (bao gồm Subversion…) là cách Git nghĩ về dữ liệu của nó.

Về mặt khái niệm, hầu hết các hệ thống khác đều lưu trữ thông tin dưới dạng danh sách các thay đổi dựa trên file. Các hệ thống này (CVS, Subversion, Perforce, Bazaar, v.v.) coi thông tin chúng lưu giữ dưới dạng một tập hợp các file và những thay đổi được thực hiện đối với mỗi file theo thời gian.

[](https://user-images.githubusercontent.com/107382675/173315605-66111018-514a-482e-ba2a-d06e81973cad.png)

## **Ưu điểm của GIT là:**

* Dễ sử dụng, thao tác nhanh, gọn, lẹ và rất an toàn.
* Sẽ dàng kết hợp các phân nhánh (branch), có thể giúp quy trình làm việc code theo nhóm đơn giản hơn rất nhiều.
* Chỉ cần clone mã nguồn từ kho chứa hoặc clone một phiên bản thay đổi nào đó từ kho chứa, hoặc một nhánh nào đó từ kho chứa là bạn có thể làm việc ở mọi lúc mọi nơi.
* Deployment sản phẩm của bạn một cách không thể nào dễ dàng hơn.

## **Hạn chế của GIT là:**

* Thuật toán SHA1 sự va chạm giá trị băm làm cho các pc thông thường làm hư hỏng một kho git.
* Sử dụng GIT trên hệ điều hành Microsoft Windows hơi phức tạp.
* Các tập tin không liên quan mà luôn luôn bị thay đổi, Git có thể chịu thiệt thòi hơn các hệ thống khác bởi vì các tập tin không được giữ dấu viết từng cái riêng lẻ.

## **Các thuật ngữ GIT quan trọng là:**

* Branch - Commit - Checkout - Fetch - Fork - Head - Index - Master - Merge - Origin - Pull - Push - Rebase - Remote - Repository - Stash - Tags - Upstream

## **Các lệnh GIT cơ bản là:**

1. git config

Tác dụng : Để set user name và email của bạn trong main configuration file.

Cách xài : Để kiểm tra tên và kiểu email trong cấu hình dùng git config -- global user.name và git config -- global user.email. Để set email hoặc tên mới git config -- global user.name = “Hải Nguyễn” và git config -- global user.email = “[hainguyen@gmail.com](mailto:hainguyen@gmail.com)”

1. git init

Tác dụng : Khởi tạo 1 git repository 1 project mới hoặc đã có.

Cách xài: git init trong thư mục gốc của dự án.

1. git clone

Tác dụng: Copy 1 git repository từ remote source.

Cách xài: git clone <:clone git url:>

1. git status

Tác dụng: Để check trạng thái của những file bạn đã thay đổi trong thư mục làm việc. VD: Tất cả các thay đổi cuối cùng từ lần commit cuối cùng.

Cách xài: git status trong thư mục làm việc.

1. git add

Tác dụng: Thêm thay đổi đến stage/index trong thư mục làm việc.

Cách xài: git add

1. git commit

Tác dụng: commit nghĩa là một action để Git lưu lại một snapshot của các sự thay đổi trong thư mục làm việc. Và các tập tin, thư mục được thay đổi đã phải nằm trong Staging Area. Mỗi lần commit nó sẽ được lưu lại lịch sử chỉnh sửa của code kèm theo tên và địa chỉ email của người commit. Ngoài ra trong Git bạn cũng có thể khôi phục lại tập tin trong lịch sử commit của nó để chia cho một branch khác, vì vậy bạn sẽ dễ dàng khôi phục lại các thay đổi trước đó.

Cách dùng: git commit -m ”Đây là message, bạn dùng để note những thay đổi để sau này dễ dò lại”

1. git push/ git pull

Tác dụng: Push hoặc Pull các thay đổi đến remote. Nếu bạn đã added và committed các thay đổi và bạn muốn đẩy nó lên hoặc remote của bạn đã update và bạn apply tất cả thay đổi đó trên code của mình.

Cách dùng: git pull <:remote:> <:branch:> and git push <:remote:> <:branch:>

1. git branch

Tác dụng: liệt kê tất cả các branch (nhánh).

Cách dùng: git branch hoặc git branch -a

1. git checkout

Tác dụng: Chuyển sang branch khác

Cách dùng: git checkout <: branch:> hoặc \*\* \_ git checkout -b <: branch:> nếu bạn muốn tạo và chuyển sang một chi nhánh mới.

1. git stash

Tác dụng: Lưu thay đổi mà bạn không muốn commit ngay lập tức.

Cách dùng: git stash trong thư mục làm việc của bạn.

1. git merge

Tác dụng: Merge 2 branch lại với nahu.

Cách dùng: Chuyển tới branch bạn muốn merge rồi dùng git merge <:branch\_ban\_muon\_merge:>

1. git reset

Tác dụng: Bạn đã đưa một tập tin nào đó vào Staging Area nhưng bây giờ bạn muốn loại bỏ nó ra khỏi đây để không phải bị commit theo.

Cách dùng: git reset HEAD tên\_file

1. git remote

Tác dụng: Để check remote/source bạn có hoặc add thêm remote

Cách dùng: git remote để kiểm tra và liệt kê. Và git remote add <: remote\_url:> để thêm.

1. git add

Tác dụng: Để đưa một tập tin vào Staging Area

Cách dùng: git add tên\_file hoặc muốn thêm hết file của thư mục thì git add all

## **So sánh GIT và các phần mềm khác**

Đây là điểm khác biệt quan trọng giữa Git và gần như tất cả các VCS khác. Nó khiến Git phải xem xét lại hầu hết mọi khía cạnh của kiểm soát phiên bản mà hầu hết các hệ thống khác đã sao chép từ thế hệ trước. Điều này làm cho Git giống như một hệ thống tệp nhỏ với một số công cụ cực kỳ mạnh mẽ được xây dựng trên nó, thay vì chỉ đơn giản là một VCS.

## **Khi sử dụng GIT**

* Lưu lại được các phiên bản khác nhau của mã nguồn dự án phần mềm.
* Khôi phục lại mã nguồn từ 1 phiên bản bất kỳ.
* Dễ dàng so sánh của các phiên bản.
* Phát hiện được ai đã sửa phần nào làm phát sinh lỗi.
* Khôi phục lại tập tin bị mất.
* Dễ dàng thử nghiệm, mở rộng tính năng của dự án mà không làm ảnh hưởng đến phiên bản chính (master branch).
* Giúp phối hợp thực hiện dự án trong một nhóm 1 cách hiệu quả.

## **Khi không sử dụng GIT**

* Không khôi phục được mã code lỡ xóa gây ra lỗi.
* Không quản lý được những người đã sửa code làm phát sinh lỗi
* Không khôi phục được tập tin đã bị mất.
* Khả năng phối hợp dự án trong nhóm bị hạn chế.

## **Cách sử dụng GIT**

#### **B1: Tạo tài khoản GitHub**

[](https://user-images.githubusercontent.com/107382675/173311849-7d286b02-3547-407d-a44a-b53d2495219f.png)

Để tạo tài khoản của bạn, bạn cần truy cập trang web của GitHub và điền vào mẫu đăng ký.

#### **B2: Cài đặt Git**

Bây giờ chúng ta cần cài đặt các công cụ của Git trên máy tính. Chúng ta sẽ sử dụng CLI để liên lạc với GitHub.

Đối với Ubuntu:

Đầu tiên, cập nhật các gói của bạn.

* sudo apt update

Tiếp theo, cài đặt Git và GitHub với apt-get

* sudo apt-get install git

Cuối cùng, xác minh rằng Git đã được cài đặt đúng

* git –version

Chạy các lệnh sau với thông tin của bạn để đặt tên người dùng và email mặc định khi bạn sẽ lưu công việc của mình.

* git config –global user.name “MV Thanoshan”
* git config –global user.email “[example@mail.com](mailto:example@mail.com)”

#### **B3: Sử dụng Git theo hai cách**

1. Cách thứ 1: Tạo kho lưu trữ, sao chép nó vào pc của bạn và làm việc với nó.

Tạo một kho lưu trữ mới bằng cách nhấp vào nút Kho lưu trữ mới của Wikipedia trên trang web GitHub.

[](https://user-images.githubusercontent.com/107382675/173312121-da4ea5a0-24ff-43d0-9941-954f247d446d.png)

Chọn tên cho kho lưu trữ đầu tiên của bạn, thêm một mô tả nhỏ, đánh dấu vào ô ‘Khởi tạo kho lưu trữ này với README’ và nhấp vào nút Tạo kho lưu trữ.

[](https://user-images.githubusercontent.com/107382675/173312279-72bcd828-7ad6-4e1d-a339-b8bf801a04d0.png)

Kho GitHub đầu tiên của bạn được tạo.

Nhiệm vụ đầu tiên của bạn là lấy một bản sao của kho lưu trữ vào máy tính của mình. Để làm được điều đó, bạn cần phải sao chép lại kho lưu trữ trên máy tính của bạn.

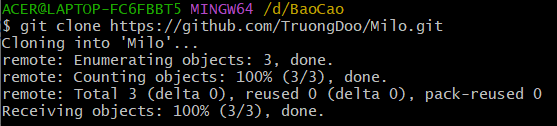
Để sao chép một kho lưu trữ có nghĩa là bạn đang lấy một kho lưu trữ trên máy chủ và sao chép nó vào máy tính của bạn – giống như tải xuống. Trên trang kho lưu trữ, bạn cần lấy địa chỉ HTTPS.

[](https://user-images.githubusercontent.com/107382675/173312470-1204e5b4-324c-4c3c-9177-e0bb07166734.png)

Khi bạn có địa chỉ của kho lưu trữ, bạn cần sử dụng terminal của mình. Sử dụng lệnh sau trên terminal của bạn. Khi bạn đã sẵn sàng, bạn có thể nhập lệnh này:

- git clone [HTTPS ADDRESS]

Lệnh này sẽ tạo một bản sao cục bộ của kho lưu trữ tại địa chỉ đã cho.

[](https://user-images.githubusercontent.com/107389856/173522116-dd2c1d13-6691-430a-b504-1a8a36a80247.png)Thông báo đầu ra của lệnh git clone bản sao

Bây giờ, kho lưu trữ của bạn là trên máy tính của bạn. Bạn cần di chuyển trong nó bằng lệnh sau.

- cd [NAME OF REPOSITORY]

Như bạn có thể thấy trong hình trên, tên kho lưu trữ của tôi là tên My- GitHub-Project, và lệnh này đã đưa tôi đến thư mục cụ thể đó.

\*LƯU Ý: Khi bạn sao chép, Git sẽ tạo một kho lưu trữ trên máy tính của bạn. Nếu bạn muốn, bạn có thể truy cập dự án của mình bằng giao diện người dùng máy tính thay vì sử dụng lệnh ‘cd’ trên terminal.

Bây giờ, trong thư mục đó, chúng ta có thể tạo các tệp, làm việc với chúng và lưu chúng cục bộ. Để lưu chúng ở một nơi xa xôi – như GitHub – chúng ta đã thực hiện một quy trình gọi là commit. Để làm điều này, quay trở lại terminal của bạn. Nếu bạn đã đóng nó, như tôi đã nói trước đây, hãy sử dụng lệnh ‘cd’.

- cd [NAME OF REPOSITORY]

Bây giờ, trong terminal, bạn đang ở trong thư mục kho lưu trữ của bạn. Có 4 bước trong một commit: ‘status’, ‘add’, ‘commit’ và ‘push’. Tất cả các bước sau đây phải được thực hiện trong dự án của bạn. Chúng ta hãy đi qua từng cái một.

‘status’: Điều đầu tiên bạn cần làm là kiểm tra các tập tin bạn đã sửa đổi. Để làm điều này, bạn có thể gõ lệnh sau để làm cho một danh sách các thay đổi xuất hiện.

- git status

[](https://user-images.githubusercontent.com/107389856/173522836-8503e5e4-ba38-4f27-902a-0bf982c40b22.png)

‘add’: Với sự trợ giúp của danh sách thay đổi, bạn có thể thêm tất cả các tệp bạn muốn tải lên bằng lệnh sau,

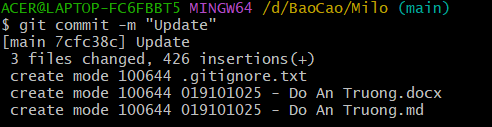
- git add [FILENAME] [FILENAME] […]

Trong trường hợp của chúng ta, chúng ta sẽ thêm một tệp HTML đơn giản.

- git add sample.html

‘commit’: Bây giờ chúng ta đã thêm các tệp mà chúng ta chọn, chúng ta cần viết một thông điệp để giải thích những gì chúng ta đã làm. Thông báo này có thể hữu ích sau này nếu chúng ta muốn kiểm tra lịch sử thay đổi. Dưới đây là một ví dụ về những gì chúng ta có thể đặt trong trường hợp của chúng ta.

- git commit -m “Added sample HTML file that contain basic syntax”

[](https://user-images.githubusercontent.com/107389856/173523002-618b4b71-cdd0-4c99-a4b7-fd28e6d7c355.png)

‘push’: Để làm điều đó, chúng ta phải ‘đẩy’ các tệp của mình lên Remote. Remote là một bản sao trùng lặp của kho lưu trữ ở một nơi khác trên máy chủ từ xa. Để làm điều này, chúng ta phải biết tên của Remote (Chủ yếu là từ xa được đặt tên gốc). Để tìm ra tên đó, gõ lệnh sau.

- git remote

Như bạn có thể thấy trong hình trên, nó nói rằng tên từ là origin. Bây giờ chúng ta có thể ‘đẩy’ công việc của mình một cách an toàn bằng lệnh sau.

- git push origin master

Bây giờ, nếu chúng ta truy cập kho lưu trữ của mình trên trang web GitHub, chúng ta có thể thấy tệp sample.html mà chúng ta đã đẩy đến từ xa – GitHub!

[](https://user-images.githubusercontent.com/107382675/173312470-1204e5b4-324c-4c3c-9177-e0bb07166734.png)

\*LƯU Ý : Đôi khi, khi bạn đang sử dụng các lệnh Git trong terminal, nó có thể dẫn bạn đến trình soạn thảo văn bản VIM (trình soạn thảo văn bản dựa trên CLI). Vì vậy, để thoát khỏi nó, bạn phải gõ

- :q

và ENTER.

Mô tả cách pull và push làm việc:

[](https://user-images.githubusercontent.com/107382675/173314720-36352134-acfd-4d8d-bb88-a9086a1a4474.png)

Pull là hành động nhận từ GitHub.

Push là hành động gửi đến GitHub.

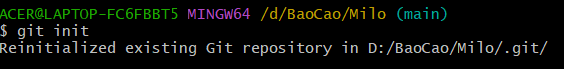
1. Cách thứ 2: Làm việc trên dự án của bạn cục bộ sau đó tạo kho lưu trữ trên github và đẩy nó vào remote.

Loại 2 cho phép bạn tạo một kho lưu trữ mới từ một thư mục hiện có trên máy tính và gửi nó đến GitHub. Trong rất nhiều trường hợp, bạn có thể đã thực sự tạo ra một cái gì đó trên máy tính mà bạn muốn đột nhiên biến thành một kho lưu trữ trên GitHub.

khi thực hiện bất kỳ lệnh Git nào, chúng ta phải đảm bảo rằng chúng ta đang ở đúng thư mục trong terminal.

Theo mặc định, bất kỳ thư mục nào trên máy tính không phải là kho lưu trữ Git – nhưng chúng ta có thể biến nó thành kho lưu trữ Git bằng cách thực hiện lệnh sau trong terminal.

- git init

[](https://user-images.githubusercontent.com/107389856/173523170-b961588b-30c8-4489-9315-2d4386f522f6.png)Sau khi chuyển đổi thư mục của chúng ta sang kho lưu trữ Git, điều đầu tiên chúng ta cần làm là kiểm tra các tệp chúng ta có bằng cách sử dụng lệnh sau.

- git status

[](https://user-images.githubusercontent.com/107389856/173522836-8503e5e4-ba38-4f27-902a-0bf982c40b22.png)

Vì vậy, có hai tập tin trong thư mục đó mà chúng ta cần để thêm vào Repo của chúng ta.

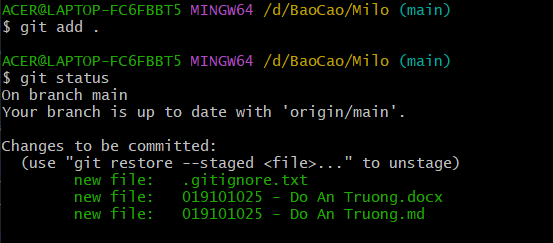
- git add [FILENAME] [FILENAME] […]

\*LƯU Ý: Để thêm vào tất cả các tệp trong Kho lưu trữ của chúng ta, chúng ta có thể sử dụng lệnh sau:

- git add .

Sau khi khu vực tổ chức (quá trình thêm) hoàn tất, chúng ta có thể kiểm tra xem các tệp có được thêm thành công hay không bằng cách thực hiện git status

Nếu những tệp cụ thể đó có màu xanh lục như hình dưới đây, bạn đã hoàn thành công việc của mình!

[](https://user-images.githubusercontent.com/107389856/173523453-865e14b2-6dce-4d38-a259-59e005960ddb.png)

Sau đó, chúng ta phải commit với một mô tả trong đó.

- git commit -m “Adding web Survey form”

Nếu kho lưu trữ của tôi bắt đầu trên GitHub và tôi đã đưa nó xuống máy tính của mình, một Remote đã được gắn vào nó (Loại 1). Nhưng nếu tôi đang khởi động kho lưu trữ của mình trên máy tính thì nó không có Remote, vì vậy tôi cần thêm Remote đó (Loại 2).

Vì vậy, để thêm Remote, chúng ta phải vào GitHub trước. Tạo một kho lưu trữ mới và đặt tên cho nó bất cứ điều gì bạn muốn lưu trữ trong GitHub. Sau đó nhấp vào nút Tạo kho lưu trữ.

\*LƯU Ý: Trong Loại 2, Vui lòng không khởi tạo kho lưu trữ với tệp README khi tạo kho lưu trữ mới trên trang web GitHub.

[](https://user-images.githubusercontent.com/107382675/173312279-72bcd828-7ad6-4e1d-a339-b8bf801a04d0.png)

Sau khi nhấp vào nút Tạo kho lưu trữ, bạn sẽ tìm thấy dạng trang web.

[](https://user-images.githubusercontent.com/107382675/173313680-0926ac3f-ab44-43cd-a21c-83769318aefb.png)

Sao chép địa chỉ HTTPS. Bây giờ chúng ta sẽ tạo Remote cho kho lưu trữ của mình.

- git remote add origin [HTTPS ADDRESS]

Sau khi thực hiện lệnh này, chúng ta có thể kiểm tra xem chúng ta đã thêm thành công Remote hay chưa bằng lệnh sau

- git remote

Và nếu nó xuất ra Origin thì bạn đã thêm Remote vào dự án của mình.

\*LƯU Ý: Chỉ cần nhớ rằng chúng ta có thể nêu bất kỳ tên nào cho Remote bằng cách thay đổi tên xuất xứ. Ví dụ:

- git remote add [REMOTE NAME] [HTTPS ADDRESS]

Bây giờ, chúng ta có thể đẩy dự án của mình lên GitHub mà không gặp vấn đề gì!

- git push origin master

Sau khi hoàn thành từng bước một, nếu bạn truy cập GitHub, bạn có thể tìm thấy kho lưu trữ của mình với các tệp!

[](https://user-images.githubusercontent.com/107382675/173312470-1204e5b4-324c-4c3c-9177-e0bb07166734.png)

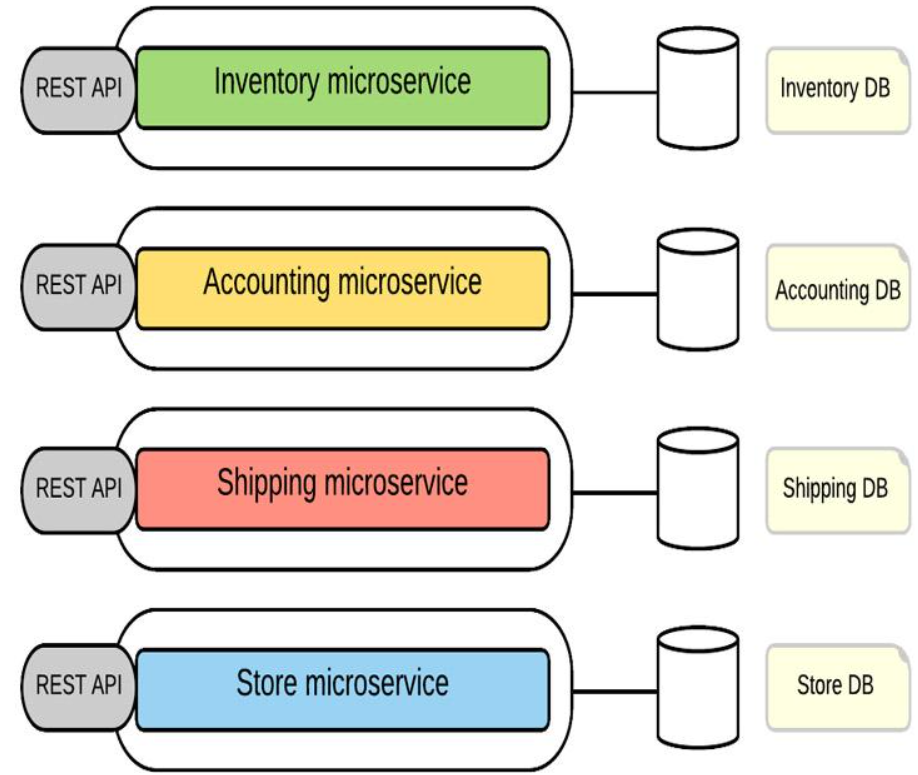
# **Microservices**

## **Microservices là gì?**

Microservices là tên gọi của các dịch vụ nhỏ thuộc dạng tách biệt đại diện cho 1 phần nhỏ tương ứng bên trong các Business domain của lập trình viên. Với kiến thức Monolithic thì bạn sẽ sở hữu một server lớn với khả năng chịu mọi trách nhiệm giải quyết hầu hết các requests. Và việc này sẽ gây ra khá nhiều khó khăn trên các phương tiện đối với tất cả requests.

## **Kiến trúc của microservices là gì?**

Bên trong kiến trúc của Microservices thì các services sẽ tồn tại độc lập nhau về xử lý, lưu trữ và cả request. Và cấu trúc cụ thể của nó sẽ như hình sau:

[](https://user-images.githubusercontent.com/107389856/173475379-a9a370fc-aac8-45fb-b170-4dd6dc637998.png)

Ví dụ như sau: nếu như bạn cần xây dựng một hệ thống để bán hàng dựa trên Microservices thì giả giả sử đơn hàng sẽ cần tối thiểu 4 service với database độc lập như sau:

* Employee service (sử dụng table tc-employee)
* Store service (sử dụng table tc-store)
* Inventory service (sử dụng table tc-warehouse)
* Order service (sử dụng table tc-order)

## **Ưu điểm của microservices:**

* Khi cần thay đổi một thành phần, thì chỉ cần sửa đổi, cập nhật và triển khai lại thành phần đó chứ không cần triển khai lại toàn bộ hệ thống.
* Dễ dàng mở rộng và tích hợp với các dịch vụ của bên thứ ba.
* Các microservice khởi động nhanh giúp quá trình phát triển, kiểm thử cũng nhanh hơn.
* Một microservice có thể được phát triển bởi một team nhỏ. Do vậy việc quản lý sẽ dễ dàng hơn.
* Dễ dàng thực hiện tự động tích hợp và tự động triển khai (CI-CD) bằng cách sử dụng một số công cụ như Jenkins, Hudson …
* Cho phép lập trình viên linh động hơn trong việc lựa chọn ngôn ngữ, công cụ và nền tảng để phát triển và triển khai các microservice (tuy nhiên trong một hệ thống, việc lựa chọn các ngôn ngữ khác nhau để phát triển các microservice không được khuyến khích)
* Mỗi microservice có kích thước nhỏ, giúp cho các lập trình viên dễ tiếp cận, đọc hiểu source code. Do vậy các thành viên mới tham gia team sẽ hòa nhập và đóng góp cho team nhanh hơn.
* Các microservice khởi động nhanh giúp quá trình phát triển, kiểm thử cũng nhanh hơn.
* Cô lập lỗi tốt hơn, khi một microservice bị lỗi và ngừng hoạt động thì các microservice khác vẫn có thể hoạt động bình thường. Với mô hình nguyên khối, một lỗi nhỏ có thể làm cả hệ thống ngừng hoạt động.

## **Nhược điểm của microservices:**

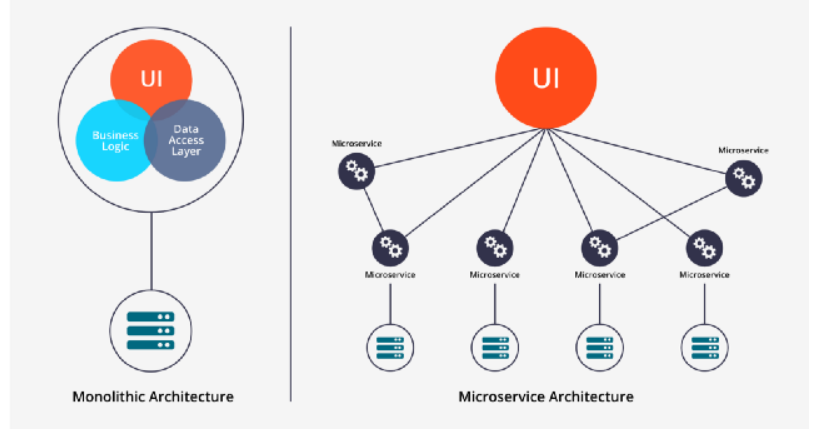
* Nhược điểm của kiến trúc microservice đến từ bản chất của hệ thống phân tán.
* Việc triển khai hệ thống microservice phức tạp hơn nhiều so với việc triển khai hệ thống nguyên khối.
* Các lập trình viên phải tốn nhiều công sức hơn để thực hiện phần giao tiếp giữa các microservice, với kiến trúc nguyên khối có khi họ chỉ cần gọi hàm để thực hiện việc này.
* Các microservice thường (nên) được triển khai bên trong docker container và giao tiếp với nhau qua REST API. Việc này làm hiệu năng của toàn bộ chương trình ứng dụng giảm xuống đáng kể do giới hạn tốc độ truyền tải của các giao thức và tốc độ mạng. Hơn nữa việc giao tiếp giữa các microservice có thể bị lỗi khi các kết nối bị lỗi.
* Cần tính toán kích cỡ của một microservice. Nếu một microservice quá lớn, bản thân nó trở thành một ứng dụng theo kiến trúc nguyên khối. Nếu một microservice quá nhỏ thì độ phức tạp của hệ thống tăng lên rất nhiều, làm cho hệ thống trở lên khó hiểu, lúc này việc quản lý giám sát và triển khai hệ thống sẽ khó khăn hơn.
* Khi ứng dụng ngày càng lớn lên, số lượng microservice ngày càng nhiều, các lập trình viên thường có xu hướng sử dụng sự hỗ trợ từ các công cụ mã nguồn mở, hoặc của bên thứ 3, việc sử dụng, tích hợp các công cụ này làm cho hệ thống khó kiểm soát và có thể bị dính các mã độc làm cho hệ thống kém an toàn.

## **Lợi ích của việc dùng microservices:**

* Source code rất tinh gọn: Bởi vì hệ thống được cấu thành từ các dự án nhỏ, và mỗi dự án đều rất đơn giản cũng như tập trung vào 1 hoặc 1 vài nghiệp vụ chính. Vì vậy, các code base và độ phức tạp của chúng đều không cao. Nhờ vậy, nó sẽ giúp mang lại tính gọn nhẹ, dễ bảo trì cũng như mở rộng hơn.
* Bảo mật tối ưu cho source code: Khi nhân viên làm việc ở các dự án thì chỉ truy cập được vào một source code của dự án đó.
* Được tồn tại độc lập: Bởi vì đây là 4 dự án khác nhau và chúng có thể có cách deploy riêng biệt và một service nào đó chết thì các service khác vẫn sẽ hoạt động một cách bình thường.
* Scale hoàn toàn độc lập: Tùy thuộc vào nhu cầu sử dụng của hệ thống mà bạn có thể scale riêng cho service đó. Có thể như service đơn hàng mà sử dụng thường xuyên nên chạy từ 2 đến 3 server để gia tăng performance.

## **Sự khác biệt của kiến trúc microservices đối với các kiến trúc khác như nào?**

#### **Sự khác biệt giữa Microservices và Monolithic**

[](https://user-images.githubusercontent.com/107389856/173479161-4e645c2e-8dc4-4c1e-bfc7-d42783903312.png)

Biểu đồ ở trên giải thích sự khác biệt giữa kiến trúc Monolithic và kiến trúc Microservice rất rất đơn giản. Với kiến trúc Monolithic, bạn sẽ có 1 server lớn chịu trách nhiệm giải quyết tất cả các requests. Việc này sẽ gây khó khăn rất nhiều trên phương diện scale. Tuy nhiên, Microservices có thể cân bằng traffic theo nhu cầu của doanh nghiệp. Nếu đang nhận được 1 lượng lớn thanh toán, bạn có thể scale up thiết bị thanh toán và giữ các dịch vụ khác ở mức sử dụng 1 lượng nhỏ hơn các services. Đây được gọi là scaling theo chiều ngang.

#### **Sự khác biệt giữa Microservices và API**

* API là tập hợp các phương thức giao tiếp, truyền thông được xác định một cách rõ ràng các thành phần khác nhau.
* Microservices là một kiến trúc có khả năng phân tách các thành phần của một ứng dụng (nguyên khối) thành các dịch vụ nhỏ hơn và có khả năng tự vận hành.