HỌC VIỆN KỸ THUẬT QUÂN SỰ

KHOA CÔNG NGHỆ THÔNG TIN

Tên học viên: Nguyễn Văn Hiến

Lớp: TN\_CNTT17

**ĐỀ CƯƠNG**

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC**

**Xây dựng phát triển và triển khai website bán quần áo**

Hà Nội, 01/2023

HỌC VIỆN KỸ THUẬT QUÂN SỰ

KHOA CÔNG NGHỆ THÔNG TIN

Tên học viên: Nguyễn Văn Hiến

Lớp: TN\_CNTT17

**ĐỀ CƯƠNG**

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC**

**Xây dựng phát triển và triển khai website bán quần áo**

GVHD: Đại tá, GVC.TS, Nguyễn Mạnh Hùng

Hà Nội, 01/2023

# LỜI NÓI ĐẦU

Trong thời đại công nghệ số hiện nay, việc mua sắm trực tuyến đã trở thành một xu hướng phổ biến và tiện lợi. Để đáp ứng nhu cầu của khách hàng, các trang web bán hàng trực tuyến ngày càng được phát triển và nâng cấp với mong muốn mang đến cho khách hàng trải nghiệm mua sắm tốt nhất. Trong bài báo cáo đồ án tốt nghiệp này, chúng tôi xin giới thiệu về website bán quần áo của chúng tôi.

Website bán quần áo này sẽ cung cấp một loạt các sản phẩm thời trang cho cả nam và nữ với nhiều kiểu dáng và phong cách khác nhau. Khách hàng có thể dễ dàng tìm kiếm sản phẩm mình muốn bằng cách sử dụng các công cụ tìm kiếm thông minh trên trang web của mà khách hàng mong muốn trên website đều có. Bên cạnh đó, chúng tôi cũng cung cấp các thông tin chi tiết về sản phẩm như kích thước, màu sắc, để giúp khách hàng có được sự lựa chọn tốt nhất.

# LỜI CẢM ƠN

Lời đầu tiên cho phép em gửi lời cảm ơn sâu sắc tới toàn thể các thầy cô giáo trong Khoa Công Nghệ Thông Tin – Trường Học Viện Kỹ Thuật Quân Sự, những người đã hết mình truyền đạt và chỉ dẫn cho chúng em những kiến thức, những bài học quý báu và bổ ích trong suốt 5 năm học vừa qua.

Để hoàn thành được đồ án tốt nghiệp này, đặc biệt em xin được bày tỏ sự tri ân và xin chân thành cảm ơn thầy Nguyễn Mạnh Hùng người trực tiếp hướng dẫn, chỉ bảo em trong quá trình học tập và nghiên cứu để hoàn thành nghiên cứu khoa học này.

Sau nữa, em xin gửi lời ơn sâu sắc tới những người bạn, anh chị em đồng nghiệp của em vì đã luôn bên cạnh khuyến khích, động viên, giúp đỡ cả về vật chất lẫn tinh thần em trong suốt quá trình học tập để em hoàn thành tốt công việc của mình.

Trong quá làm đồ án tốt nghiệp và làm báo cáo này do năng lực, kiến thức, trình độ bản thân còn hạn hẹp nên không tránh khỏi những thiếu sót. Em kính mong nhận được sự thông cảm và những ý kiến đóng góp của quý thầy cô và các bạn.

**Em xin chân thành cảm ơn!**

Hà Nội, ngày 06 tháng 02 năm 2023

Sinh viên thực hiện

Nguyễn Văn Hiến

# DANH MỤC TỪ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **STT** | **Từ viết tắt** | **Ý Nghĩa** |
|  |  |  |

# DANH MỤC CÁCH HÌNH ẢNH

|  |  |  |
| --- | --- | --- |
| **STT** | **Nội dung** | **Số trang** |
|  |  |  |

# CHƯƠNG I. MỞ ĐẦU

## 1. Giới thiệu

Tên đề tài: Xây dựng và phát triển website bán quần áo

### 1.1 Thông tin sinh viên thực hiện

**Họ và Tên**: Nguyễn Văn Hiến

**Lớp**: CNDL\_17

### 1.2 Thông tin giáo viên hướng dẫn

**Họ và Tên của GVHD:** Nguyễn Mạnh Hùng

**Học hàm của GVHD:** Đại tá

**Học vị của GVHD:** GVC.TS

### 1.3 Giới thiệu về đề tài

Ngày này xã hội ngày càng phát triển, cũng nhờ đó mà thấy được nhu cầu mua sắm trực tuyến đang ngày càng tăng cao. Nhưng với sự phát triển của thị trường, khách hàng ngày càng đòi hỏi sự tiện lợi, tốc độ giao hàng nhanh chóng, chất lượng sản phẩm tốt và giá cả hợp lý. Tôi muốn phát triển một website bán quần áo giúp cho khách hàng dễ dàng lựa chọn sản phẩm, thanh toán và nhận hàng tại nhà. Đồng thời, tôi cũng muốn cung cấp cho khách hàng các thông tin về sản phẩm, chính sách bảo hành và đổi trả để khách hàng có thể tin tưởng và sử dụng dịch vụ của mình.

## 2. Mục tiêu đề tài

**Mục đích đề tài:** Xây dựng phát triển và triển khai được một website bán quần áo

## 3. Phạm vi đề tài

**Về mặt phát triển**:

Đáp ứng như cầu mua bán, xem thông tin sản phẩm nhanh chóng tiện lợi, hiệu quả, an toàn về mặt mua sắm trực tuyến.

Website đáp ứng được các yêu cầu:

**Đối với khách hàng:**

* Xem thông tin sản phẩm, giá cả, loại sản phẩm, kiểu dáng, kích cỡ size, màu sắc…..
* Tìm kiếm sản phẩm nhanh chóng theo yêu cầu
* Mua sắm sản phẩm bằng cách đặt hàng qua mạng một cách nhanh chóng và tiện lợi
* Xem thông tin bài viết về các sản phẩm

**Đối với quản trị:**

* Quản lý thông tin người dùng (khách hàng)
* Quản lý sản phẩm, loại sản phẩm
* Quản lý các đơn hàng của khách hàng
* Quản lý các bài viết, loại bài viết

**Về mặt triển khai:**

* Triển khai hệ thống website bán quần áo thực tế trên mạng người dùng (mọi người) có thể mua sắm được qua mạng

# CHƯƠNG II. CƠ SỞ LÝ THUYẾT

## 1. Giới thiệu công nghệ và công cụ sử dụng

Các công nghệ sử dụng để phát triển

* Công nghệ làm Fontend: Framework Vuejs.
* Công nghệ làm Backend: Asp.net core.
* Một số công nghệ khác như: Redis sử dụng để lưu cached, MinIO hệ thống lưu trữ file, RabbitMQ.
* Các công nghệ sử dụng để triển khai
* Docker: sử dụng quản lý đóng gói các image, container.
* Jenkins: CI/CD - Continuous Intergration/Continuous Deliver.
* Công cụ sử dụng: Visual studio 2019, Visual studio code, dbeaver…..

## 2. Cơ sở lý thuyết

### 2.1 Giới thiệu công nghệ phát triển về Fontend Vue js

Gọi tắt là Vue (phát âm là /vjuː/, giống như view trong tiếng Anh).

Vue.js là một framework linh động (nguyên bản tiếng Anh: progressive – tiệm tiến) dùng để xây dựng giao diện người dùng (user interfaces). Khác với các framework nguyên khối (monolithic), Vue được thiết kế từ đầu theo hướng cho phép và khuyến khích việc phát triển ứng dụng theo từng bước. Khi phát triển lớp giao diện (view layer), người dùng chỉ cần dùng thư viện lõi (core library) của Vue, vốn rất dễ học và tích hợp với các thư viện hoặc dự án có sẵn. Cùng lúc đó, nếu kết hợp với những kĩ thuật hiện đại như SFC (single file components) và các thư viện hỗ trợ, Vue cũng đáp ứng được dễ dàng nhu cầu xây dựng những ứng dụng một trang (SPA - Single-Page Applications) với độ phức tạp cao hơn nhiều.

Nếu bạn muốn tìm hiểu thêm về Vue, chúng tôi đã tạo một video clip về những nguyên tắc cốt lõi và một dự án mẫu.

Nếu bạn là một lập trình viên front-end giàu kinh nghiệm và muốn hiểu hơn về tương quan giữa vue và các thư viện hay framework khác, hãy xem phần So sánh với các framework khác.

Vue là một framework Javascript được tạo bởi Evan You, giúp chúng ta xây dựng giao diện người dùng cũng như xây dựng Single Page Application thân thiện với người dùng, chúng xây dựng từ các thư viện, cách triển khai component, các chức năng đặc trưng của nó như SFC (Single File Component).

**Vuex là gì?**  
Vuex là thư viện giúp quản lý trạng thái các component trong Vue.js, nó là nơi lưu trữ tập trung cho tất cả các component trong một ứng dụng, với nguyên tắc trạng thái chỉ có thể được thay đổi theo kiểu có thể dự đoán.

Vuex hoạt động theo mô hình "Luồng dữ liệu một chiều" với các thành phần sau:

State: Trạng thái, là nơi khởi nguồn để thực hiện ứng dụng.

View: Khung nhìn, là các khai báo ánh xạ với trạng thái.

Action: Hành động, là những cách thức làm trạng thái thay đổi phản ứng lại các nhập liệu của người dùng từ View.

Vuex nhìn thấy tại sao không đưa các trạng thái được chia sẻ của các component ra và quản lý chúng trong một bộ máy toàn cục, và đó chính là lý do cho sự ra đời của Vuex. Trong đó, các component trở thành các view và các component có thể truy xuất trạng thái hoặc trigger các hành động. Với cách thức này, mã nguồn có cấu trúc và dễ dàng duy trì.

Giới thiệu Ant design

Ant là tập hợp các components của Vue được xây dựng theo chuẩn thiết kế của Ant UED Team. Tương tự như chuẩn Material Design, Ant cung cấp hầu hết các component thông dụng trong ứng dụng web hiện đại, như Layout, Button, Icon, DatePicket, v.v…Bên cạnh đó Ant cũng có những component riêng thú vị, như LocaleProvider cho phép bạn thay đổi ngôn ngữ trên toàn ứng dụng.

Có thể coi Ant là tập hợp của hầu hết các thư viện về Vue. Nó đáp ứng được hầu hết các yêu cầu của project của bạn mà ban không phải cài thêm bất cứ thư viện nào nữa. Dưới đây là danh sách các component mà nó cung cấp:

General: Button, Icon

Layout: Grid, Layout

Navigation: Affix, Breadcrumb, Dropdown, Menu, Pagination, Steps

Data Entry: AutoComplete, Checkbox, Cascader, DatePicker, Form, InputNumber, Input, Mention, Rate, Radio, Switch, Slider, Select, TreeSelect, Transfer, TimePicker, Upload

Data Display: Avatar, Badge, Collapse, Carousel, Card, Calendar, List, Popover, Tree, Tooltip, Timeline, Tag, Tabs, Table

Feedback: Alert, Drawer, Modal, Message, Notification, Progress, Popconfirm, Spin, Skeleton

Other: Anchor, BackTop, Divider, LocaleProvider

### 2.2 Giới thiệu công nghệ phát triển về Backend Asp.net core

**ASP.NET Core là gì?**

ASP.NET Core là một tập hợp các thư viện chuẩn như một framework để xây dựng ứng dụng web. ASP.NET Core không phải là phiên bản tiếp theo của ASP.NET. Nó là một cái tên mới được xây dựng từ đầu. Nó có một sự thay đổi lớn về kiến trúc và kết quả là nó gọn hơn, phân chia module tốt hơn. ASP.NET Core có thể chạy trên cả .NET Core hoặc full .NET Framework.

**.Net Core core là gì?**

.NET Core là môi trường thực thi. Nó được thiết kế lại hoàn toàn của .NET Framework. Mục tiêu chính của .NET Core là hỗ trợ phát triển ứng dụng đa nền tảng cho ứng dụng .NET. Nó được hỗ trợ trên Windows, Mac OS và Linux. .NET Core là một framework mã nguồn mở được xây dựng và phát triển bởi Microsoft và cộng đồng .NET trên Github

NET Core là một tập con của Full .NET Framwork. WebForms, Windows Forms, WPF không phải là một phần của .NET Core. Nó cũng triển khai đặc điểm của .NET Standard.

**.NET Standard là gì?**

.NET Standard là một đặc tả chuẩn của .NET API hướng tới hỗ trợ trên tất cả các triển khai của nền tảng .NET. Nó định nghĩa một tập các quy tắc thống nhất cần thiết để hỗ trợ tất cả các ứng dụng trên nền .NET.

Sự khác nhau giữa .NET Core và .NET Framework

.NET Framwork là môi trường cũ hơn và đã tồn tại trên Windows nhiều năm nay. .NET Core hỗ trợ các tính năng trong tập con của .NET Framwork. Các tính năng như WebForms, Windows Forms, WPF chưa được đưa lên .NET Core. .NET Framwork chỉ chạy trên Windows trong khi .NET Core có thể chạy trên bất cứ nền tảng nào.

Từ Core trong ASP.NET Core rất dễ nhầm. Tên đó gợi nhớ việc ứng dụng ASP.NET Core chỉ có thể được build trên .NET Core, điều đó là sai.Ứng dụng ASP.NET Core có thể được phát triển sử dụng .NET Core hoặc .NET Framework. Ứng dụng ASP.NET Core được xây dựng sử dụng .NET Core có thể chạy trên bất cứ hệ điều hành nào trong khi nếu được xây dựng trên .NET Framework chỉ có thể chạy trên Windows.

Lựa chọn công nghệ: Các giải pháp công nghệ được lựa chọn để thực hiện bài toán: Hệ điều hành (nếu có đặc thù), hạ tầng mạng, ngôn ngữ lập trình, Hệ quản trị CSDL, các công cụ hỗ trợ lập trình (front-end/back-end) và quản lý tài nguyên, giao tiếp nhóm….

**Các đặc tính quan trọng của ASP.NET Core**

Bạn có thể xây dựng và chạy ứng dụng ASP.NET đa nền tảng trên Windows, Mac và Linux (mã nguồn mở và cộng đồng phát triển)

ASP.NET Core hợp nhất ASP.NET MVC và ASP.NET Web API.

Có thể host trên IIS hoặc tự host.

Có sẵn Dependency Injection.

Dễ dàng tích hợp với các framework frontend như Angular, Knockout...

Hỗ trợ cấu hình cho nhiều môi trường.

Cơ chế HTTP Request pipeline mới.

Hỗ trợ quản lý phiên bản

Dùng chung toàn bộ Nuget Package.

**Các nhánh của ASP.NET**

Có hai nhánh của ASP.NET cho đến hiện tại

ASP.NET Đây là phiên bản hiện tại của ASP.NET và nó cần .NET Framwork để chạy

ASP.NET Core. Là cách mới để xây dựng ứng dụng web. Nó có thể chạy cả trên .NET Framework và .NET Core.

**Web API là gì?**

Web API là một phương thức dùng để cho phép các ứng dụng khác nhau có thể giao tiếp, trao đổi dữ liệu qua lại. Dữ liệu được Web API trả lại thường ở dạng JSON hoặc XML thông qua giao thức HTTP hoặc HTTPS.

Những điểm nổi bật của Web API Web API hỗ trợ restful đầy đủ các phương thức: Get/Post/put/delete dữ liệu. Nó giúp bạn xây dựng các HTTP service một cách rất đơn giản và nhanh chóng. Nó cũng có khả năng hỗ trợ đầy đủ các thành phần HTTP: URI, request/response headers, caching, versioning, content format.

Khả năng tích hợp linh động API cho phép lấy nội dung từ bất kỳ website hoặc ứng dụng nào một cách dễ dàng nếu được cho phép, tăng trải nghiệm người dùng. API hoạt động như một chiếc cổng, cho phép các công ty chia sẻ thông tin được chọn nhưng vẫn tránh được những yêu cầu không mong muốn.

Cập nhật thông tin thời gian thực API có chức năng thay đổi và cập nhật thay đổi theo thời gian thực. Với công nghệ này, dữ liệu sẽ được truyền đi tốt hơn, thông tin chính xác hơn, dịch vụ cung cấp linh hoạt hơn.

Có tiêu chuẩn chung dễ sử dụng bất kỳ người dùng, công ty nào sử dụng cũng có thể điều chỉnh nội dung, dịch vụ mà họ sử dụng.Hỗ trợ đầy đủ các thành phần MVC như: routing, controller, action result, filter, model binder, IoC container, dependency injection, unit test.

**Web API hoạt động như thế nào?**

Đầu tiên là xây dựng URL API để bên thứ ba có thể gửi request dữ liệu đến máy chủ cung cấp nội dung, dịch vụ thông qua giao thức HTTP hoặc HTTPS.

Tại web server cung cấp nội dung, các ứng dụng nguồn sẽ thực hiện kiểm tra xác thực nếu có và tìm đến tài nguyên thích hợp để tạo nội dung trả về kết quả.

Server trả về kết quả theo định dạng JSON hoặc XML thông qua giao thức HTTP/HTTPS.Tại nơi yêu cầu ban đầu là ứng dụng web hoặc ứng dụng di động , dữ liệu JSON/XML sẽ được parse để lấy data. Sau khi có được data thì thực hiện tiếp các hoạt động như lưu dữ liệu xuống Cơ sở dữ liệu, hiển thị dữ liệu…

**Ưu và nhược điểm của Web API**

Mỗi một ứng dụng bất kỳ đều có những ưu nhược điểm riêng, hỗ trợ tốt cho các ứng dụng. Vì vậy mà web API cũng không ngoại lệ:

Ưu điểm:

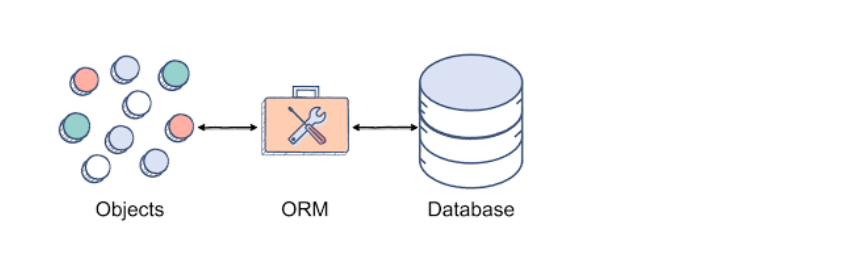
* Web API được sử dụng hầu hết trên các ứng dụng desktop, ứng dụng mobile và ứng dụng website.
* Linh hoạt với các định dạng dữ liệu khi trả về client: Json, XML hay định dạng khác.
* Nhanh chóng xây dựng HTTP service: URI, request/response headers, caching, versioning, content formats và có thể host trong ứng dụng hoặc trên IIS.
* Mã nguồn mở, hỗ trợ chức năng RESTful đầy đủ, sử dụng bởi bất kì client nào hỗ trợ XML, Json.
* Hỗ trợ đầy đủ các thành phần MVC như: routing, controller, action result, filter, model binder, IoC container, dependency injection, unit test.
* Giao tiếp hai chiều được xác nhận trong các giao dịch, đảm bảo độ tin cậy cao.

Nhược điểm

* Web API chưa hoàn toàn phải là RESTful service, mới chỉ hỗ trợ mặc định GET, POST
* Để sử dụng hiệu quả cần có kiến thức chuyên sâu, có kinh nghiệm backend tốt
* Tốn thời gian và chi phí cho việc phát triển, nâng cấp và vận hành
* Có thể gặp vấn đề về bảo mật khi hệ thống bị tấn công nếu không giới hạn điều kiện kỹ.

**Giới thiệu về Entity framework core**

**Entity framework core là gì?**

  
**Entity Framework Core** là một Object Relational Mapping (ORM) framework giúp dễ dàng truy xuất và lưu trữ dữ liệu trong database thông qua việc ánh xạ database thành các objects tương ứng trong code.

**Một số đặc điểm của Entity framework core**

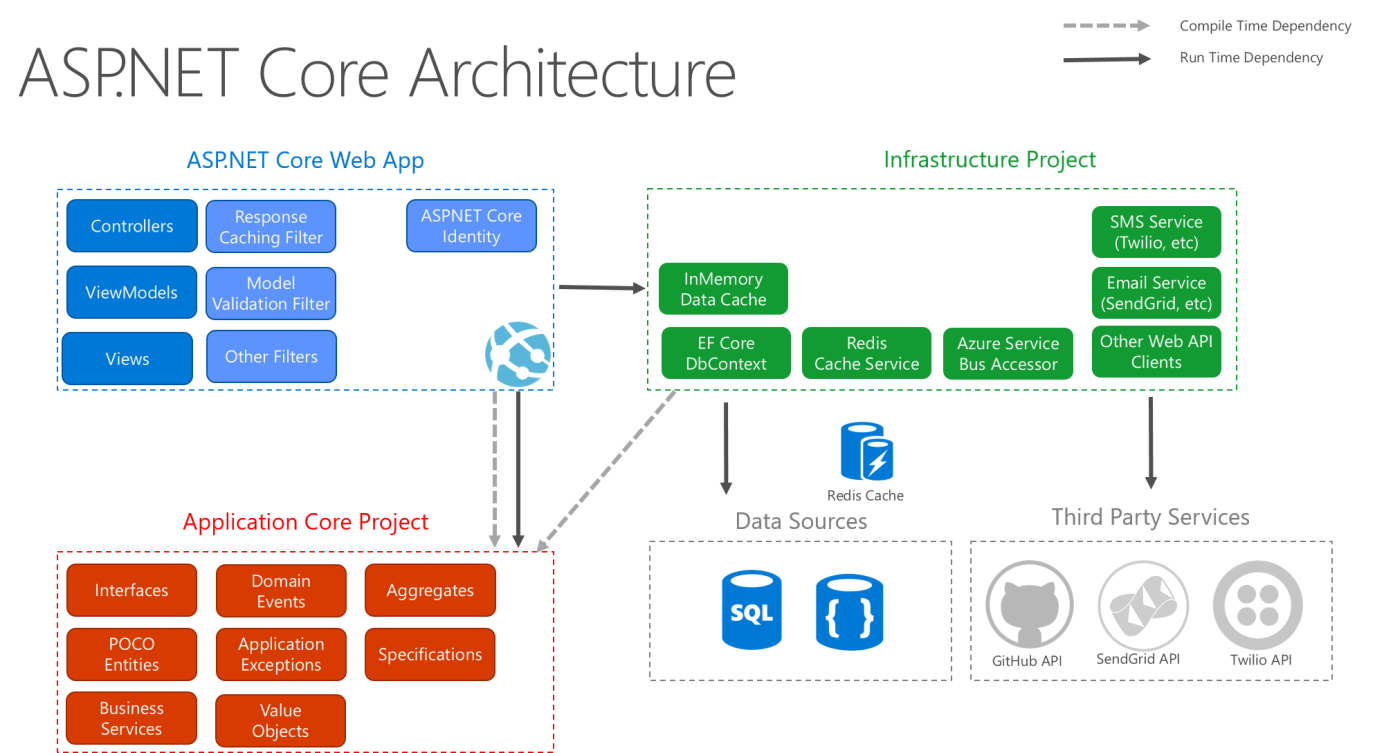
* Là một ORM
* Là phiên bản mới của Entity Framework 6.x (https://docs.microsoft.com/en-us/ef/efcore-and-ef6/)
* Open source
* Nhẹ hơn Entity Framework => tốc độ nhanh hơn (performance)

**Entity framework core hỗ trợ các database nào?**

Entity framework core hỗ trợ cho phép làm việc với nhiều loại database. Để làm việc với từng loại ta phải instal các database provider qua NuGet packages.

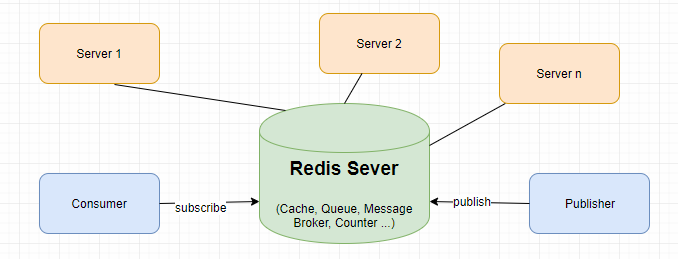
* SQL Server (Microsoft.EntityFrameworkCore.SqlServer)
* MySQL (MySql.Data.EntityFrameworkCore)
* PostgreSQL
* Oracle
* SQLite (mobile)
* SQL Server Compact
* DB2
* Firebird
* Jet (Microsoft Access)
* Azure Cosmos DB
* In-memory (for testing) (Microsoft.EntityFrameworkCore.InMemory)

**Clean Architectue**



### 2.3 Giới thiệu về redis

**Redis là gì? – Redis (REmote DIctionary Server)** là một mã nguồn mở được dùng để lưu trữ dữ liệu có cấu trúc, có thể sử dụng như một database, bộ nhớ cache hay một message broker.Nó là hệ thống lưu trữ dữ liệu với dạng KEY-VALUE rất mạnh mẽ và phổ biến hiện nay. **Redis** nổi bật bởi việc hỗ trợ nhiều cấu trúc dữ liệu cơ bản như:hash, list, set, sorted set, string… Tất cả dữ liệu được ghi và lưu trên ram, do đó tốc độ đọc ghi dữ liệu rất là nhanh.



**Các ứng dụng của Redis**

Sau khái niệm redis là gì thì chúng ta hãy đi đến ứng dụng của Redis ngoài tính năng lưu trữ KEY-VALUE trên RAM thì Redis còn hỗ trợ tính năng xắp xếp, query, backup dữ liệu trên đĩa cứng cho phép bạn có thể phục hồi dữ liệu khi hệ thống gặp sự cố…và có thể nhân bản (Chạy nhiều Server Redis cùng lúc).

**Caching:**Sử dụng làm bộ nhớ đệm. Chính tốc độ đọc ghi nhanh mà Redis có thể làm bộ nhớ đệm, nơi chia sẻ dữ liệu giữa các ứng dụng hoặc làm database tạm thời. Ngoài ra Redis có thể sử dụng để làm Full Page Cache cho website. Cũng vì tính nhất quán của Redis, cho dù restart Redis thì người dùng cũng không có cảm nhận chậm khi tải trang.

**Counter:** Sử dụng làm bộ đếm. Với thuộc tính tăng giảm thông số rất nhanh trong khi dữ liệu được lưu trên RAM, sets và sorted sets được sử dụng thực hiện đếm lượt view của một website, các bảng xếp hạng trong game chẳng hạng. Redis hỗ trợ thread safe do đó nó có thể đồng bộ dữ liệu giữa các request.

**Publish/Suscribe (Pub/Sub):** Tạo kênh chia sẻ dữ liệu. Redis hỗ trợ tạo các channel để trao đổi dữ liệu giữa publisher và subscriber giống như channel trong **Socket Cluster** hay topic trong **Apache Kafka. Ví dụ:**Pub/Sub được sử dụng theo dõi các kết nối trong mạng xã hội hoặc các hệ thống chat.

**Queues:** Tạo hàng đợi để xử lý lần lượt các request. Redis cho phép lưu trữ theo list và cung cấp rất nhiều thao tác với các phần tử trong list, vì vậy nó còn được sử dụng như một message queue.

**Các kiểu dữ liệu trong Redis**

Khác với RDMS như MySQL, hay PostgreSQL, Redis không có table (bảng). Redis lưu trữ data dưới dạng key-value. Thực tế thì memcache cũng làm vậy, nhưng kiểu dữ liệu của memcache bị hạn chế, không đa dạng được như Redis, do đó không hỗ trợ được nhiều thao tác từ phía người dùng. Dưới đây là sơ lược về các kiểu dữ liệu Redis dùng để lưu value.

**STRING**: string, integer hoặc float. Redis có thể làm việc với cả string, từng phần của string, cũng như tăng/giảm giá trị của integer, float.

**LIST**: List là một danh sách của strings, sắp xếp theo thứ tự insert. Redis có thể thêm một phần tử vào đầu hoặc cuối list. List phù hợp cho các bài toán cần thao tác với các phần tử gần đầu và cuối vì việc truy xuất này là cực nhanh, cho dù insert cả triệu phần tử. Tuy nhiên nhược điểm là việc truy cập vào các phần tử ở giữa list rất chậm.

**SET**: tập hợp các string (không được sắp xếp). Redis hỗ trợ các thao tác thêm, đọc, xóa từng phần tử, kiểm tra sự xuất hiện của phần tử trong tập hợp. Ngoài ra Redis còn hỗ trợ các phép toán tập hợp, gồm intersect/union/difference.

**HASH**: lưu trữ hash table của các cặp key-value, trong đó key được sắp xếp ngẫu nhiên, không theo thứ tự nào cả. Redis hỗ trợ các thao tác thêm, đọc, xóa từng phần tử, cũng như đọc tất cả giá trị.

**SORTED SET (ZSET)**: là 1 danh sách, trong đó mỗi phần tử là map của 1 string (member) và 1 floating-point number (score), danh sách được sắp xếp theo score này. Các phần tử của zset được sắp xếp theo thứ tự từ score nhỏ tới lớn.

Ngoài ra, Redis còn hỗ trợ các data types khác như: Bit arrays, HyperLogLogs, Streams.

**Persistent redis là gì**

Bên cạnh việc lưu key-value trên bộ nhớ RAM, Redis có 2 background threads chuyên làm nhiệm vụ định kỳ ghi dữ liệu lên đĩa cứng.

Có 2 loại file được ghi xuống đĩa cứng: **RDB (Redis DataBase file) và AOF (Append Only File)**

**RDB (Redis DataBase file)**

RDB thực hiện tạo và sao lưu snapshot của DB vào ổ cứng sau mỗi khoảng thời gian nhất định.

**Ưu điểm**

RDB cho phép người dùng lưu các version khác nhau của DB, rất thuận tiện khi có sự cố xảy ra.

Bằng việc lưu trữ data vào 1 file cố định, người dùng có thể dễ dàng chuyển data đến các data centers, máy chủ khác nhau.

RDB giúp tối ưu hóa hiệu năng của Redis. Tiến trình Redis chính sẽ chỉ làm các công việc trên RAM, bao gồm các thao tác cơ bản được yêu cầu từ phía client như thêm/đọc/xóa, trong khi đó 1 tiến trình con sẽ đảm nhiệm các thao tác disk I/O. Cách tổ chức này giúp tối đa hiệu năng của Redis.

Khi restart server, dùng RDB làm việc với lượng data lớn sẽ có tốc độ cao hơn là dùng AOF.

**Nhược điểm**

RDB không phải là lựa chọn tốt nếu bạn muốn giảm thiểu tối đa nguy cơ mất mát dữ liệu.

Thông thường người dùng sẽ set up để tạo RDB snapshot 5 phút 1 lần (hoặc nhiều hơn). Do vậy, trong trường hợp có sự cố, Redis không thể hoạt động, dữ liệu trong những phút cuối sẽ bị mất.

RDB cần dùng fork() để tạo tiến trình con phục vụ cho thao tác disk I/O. Trong trường hợp dữ liệu quá lớn, quá trình fork() có thể tốn thời gian và server sẽ không thể đáp ứng được request từ client trong vài milisecond hoặc thậm chí là 1 second tùy thuộc vào lượng data và hiệu năng CPU.

**AOF (Append Only File)**

AOF lưu lại tất cả các thao tác write mà server nhận được, các thao tác này sẽ được chạy lại khi restart server hoặc tái thiết lập dataset ban đầu.

**Ưu điểm**

Sử dụng AOF sẽ giúp đảm bảo dataset được bền vững hơn so với dùng RDB. Người dùng có thể config để Redis ghi log theo từng câu query hoặc mỗi giây 1 lần.

Redis ghi log AOF theo kiểu thêm vào cuối file sẵn có, do đó tiến trình seek trên file có sẵn là không cần thiết. Ngoài ra, kể cả khi chỉ 1 nửa câu lệnh được ghi trong file log (có thể do ổ đĩa bị full), Redis vẫn có cơ chế quản lý và sửa chữa lối đó (redis-check-aof).

Redis cung cấp tiến trình chạy nền, cho phép ghi lại file AOF khi dung lượng file quá lớn.

Trong khi server vẫn thực hiện thao tác trên file cũ, 1 file hoàn toàn mới được tạo ra với số lượng tối thiểu operation phục vụ cho việc tạo dataset hiện tại. Và 1 khi file mới được ghi xong, Redis sẽ chuyển sang thực hiện thao tác ghi log trên file mới.

**Nhược điểm**

File AOF thường lớn hơn file RDB với cùng 1 dataset.

AOF có thể chậm hơn RDB tùy theo cách thức thiết lập khoảng thời gian cho việc sao lưu vào ổ cứng. Tuy nhiên, nếu thiết lập log 1 giây 1 lần có thể đạt hiệu năng tương đương với RDB.

Developer của Redis đã từng gặp phải bug với AOF (mặc dù là rất hiếm), đó là lỗi AOF không thể tái tạo lại chính xác dataset khi restart Redis. Lỗi này chưa gặp phải khi làm việc với RDB bao giờ.

**Kết luận redis là gì?**

**Redis** là một sự lựa chọn tuyệt vời khi ta cần đến một server lưu trữ dữ liệu đòi hỏi tính mở rộng cao (scaleable) và chia sẻ bởi nhiều tiến trình, nhiều ứng dụng và nhiều server khác nhau.

### 2.4 Giới thiệu về Rabbit MQ

**RabbitMQ là gì?**  
**RabbitMQ** là một AMQP message broker hay còn gọi là phần mềm quản lý hàng đợi message. Nói đơn giản, đây là phần mềm định nghĩa hàng đợi một ứng dụng khác có thể kết nối đến để bỏ message vào và gửi message dựa trên nó.Có thể coi nó như một hộp thư nơi bạn xếp chồng các bức thư của mình. RabbitMQ sau đó lấy từng bức thư và đưa nó đến đích của nó.

**Message broker là gì?**

**Message broker** là một chương trình trung gian được thiết kế để validating, transforming và routing messages. Chúng phục vụ các nhu cầu giao tiếp giữa các ứng dụng với nhau.

Với Message broker, ứng dụng nguồn (producer) gửi một message đến một server process mà nó có thể cung cấp việc sắp xếp dữ liệu, routing (Định tuyến), message translation, persistence và delivery tất cả các điểm đến thích hợp (consumer).

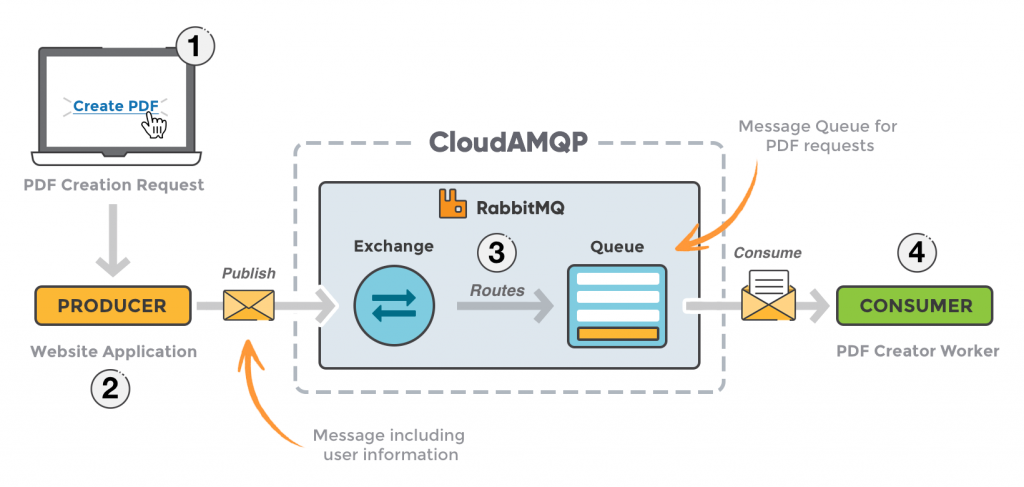
Có 2 hình thức giao tiếp cơ bản với một Message Broker:

Publish và Subscribe (Topics)

Point-to-Point (Queues)

Khi nào và tại sao dùng RabbitMQ

RabbitMQ giúp các web server gửi các reponse cho các request rất nhanh thay vì bị ép buộc chạy một procedure ngốn tài nguyên trên một hệ thống. Việc đưa message vào hàng đợi là một giải pháp hay khi ta muốn phân tán message cho nhiều người nhận giúp giảm tải cho các worker xử lý.



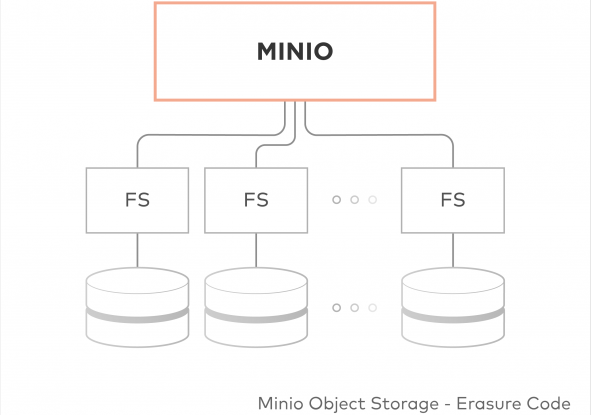
**Khi nào và tại sao dùng RabbitMQ**

RabbitMQ giúp các web server gửi các reponse cho các request rất nhanh thay vì bị ép buộc chạy một procedure ngốn tài nguyên trên một hệ thống. Việc đưa message vào hàng đợi là một giải pháp hay khi ta muốn phân tán message cho nhiều người nhận giúp giảm tải cho các worker xử lý.

VD như user được phép tạo file PDF cho mẫu cv it từ phần mềm tạo CV Online của TopDev, bài toán là khi hàng nghìn user cùng nhấp vào nút tạo PDF, lúc này server nhận rất nhiều request sẽ gây ra vài vấn đề như chậm, quá tải, thậm chí không tạo được file PDF do nghẽn…lúc này chúng ta cần dùng RabbitMQ để đẩy các request này vào hàng chờ. Cơ chế như sau:

Một consumer lấy message từ hàng đợi và bắt đầu xử lý PDF trong lúc với một producer đang bỏ thêm những message mới vào trong hàng đợi. Một request có thể được tạo bằng ngôn ngữ này và xử lý bằng một ngôn ngữ khác. Hai ứng dụng trao đổi với nhau qua các message. Do đó, hai ứng dụng gửi và nhận sẽ có độ ràng buộc thấp.

### 2.5 Giới thiệu về MinIO



**Minio là gì ?**

Minio là một object storage server được implement những public API giống như AWS S3. Điều đó có nghĩa là những ứng dụng có thể config để giao tiếp với Minio thì cũng có thể giao tiếp với AWS S3. Là một server lưu trữ object nên có thể được sử dụng để lưu trữ những unstructured data như ảnh, video, log files, backups và container/VM images. Dung lượng của 1 object có thể dao động từ một vài KB tới tối đa là 5TB. File cũng được gom lại trong 1 buckets, nó là được chỉ cùng với access key khi dùng app.

**Xử lý khối lượng lớn dự liệu**

Giả sử bạn phải lưu trữ một khối dữ liệu, bạn không muốn những dữ liệu này (ảnh, mailboxes,...) lưu trữ trên cùng một app. Sẽ là một vấn đề lớn nếu lưu trữ ở cùng một server vì lượng dữ liệu này khá lớn, ví dụ như mailboxes chúng ta cần dung lượng có thể mở rộng và sao lưu được để tránh việc mất dữ liệu.

Minio là công cụ tốt để handle những điều trên. Nó tách những dữ liệu lưu trữ khỏi app của bạn và có thể truy cập thông qua HTTP.

### 2.6 Giới thiệu về Docker

Docker là gì?

Docker là một nền tảng cho developers và [system admin](https://topdev.vn/viec-lam-it/system-admin-kt31) để develop, deploy và run application với container. Nó cho phép tạo các môi trường độc lập và tách biệt để khởi chạy và phát triển ứng dụng và môi trường này được gọi là container. Khi cần deploy lên bất kỳ server nào chỉ cần run container của Docker thì application của bạn sẽ được khởi chạy ngay lập tức.

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.

**Virtualization là gì ?**

**Virtualization host là gì ?**

Khi nói về Virtualization, nó đề cập đến việc nhập hệ điều hành Guest trên hệ điều hành máy chủ, cho phép các nhà phát triển chạy nhiều HĐH trên các máy ảo khác nhau trong khi tất cả chúng chạy trên cùng một máy chủ, do đó loại bỏ nhu cầu cung cấp thêm tài nguyên phần cứng.

**Ưu điểm :**

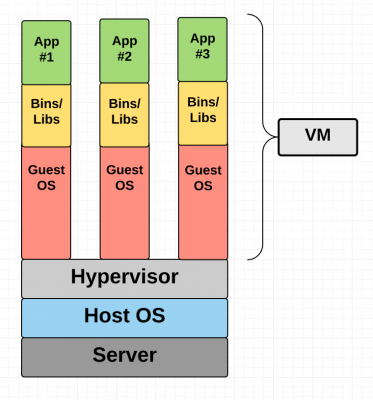
Kích hoạt nhiều hệ điều hành trên cùng một máy.

Nó rẻ hơn so với các phương pháp trước đây, do thiết lập cơ sở hạ tầng ít hơn / nhỏ gọn.

Nếu có bất kỳ trạng thái thất bại nào, thật dễ dàng để phục hồi và bảo trì.

Cung cấp nhanh hơn các ứng dụng và tài nguyên cần thiết cho các nhiệm vụ.

Tăng năng suất, hiệu quả và đáp ứng CNTT.



Từ kiến trúc VM ở trên, chúng ta có thể hình dung ra rằng 3 hệ điều hành Guest hoạt động như các máy ảo đang chạy trên một hệ điều hành máy chủ. Trong **Virtualization**, quá trình cấu hình lại phần cứng, phần sụn thủ công, cài đặt HĐH mới, cài đặt hệ điều hành mới có thể hoàn toàn tự động, tất cả các bước này được lưu trữ dưới dạng dữ liệu trong bất kỳ tệp nào của đĩa.

Trong **Virtualization**, mỗi ứng dụng và hệ điều hành sống trong một thùng chứa phần mềm riêng biệt có tên Virtural Machine (VM) , trong đó VM hoàn toàn tách biệt, tất cả các tài nguyên điện toán như CPU, lưu trữ và kết nối mạng được gộp chung với nhau. Virtural Machine (VM) về bản chất là một giả lập của một máy tính để thực thi các ứng dụng giống như một máy tính thật. VMs chạy trên một máy vật lý sử dụng một thứ gọi là “hypervisor”. Hypervisor có thể là phần cứng, phần mềm hoặc là một bản firmware nào đó có thể chạy trực tiếp trên máy thật (host machine) có chức năng cho nhiều máy ảo chạy trên nó. Host machine sẽ cung cấp cho VMs những tài nguyên như là RAM, CPU. Những tài nguyên đó sẽ được phân bổ giữa các VMs theo cách mà bạn cho là hợp lý. Nếu một VM chạy nhiều ứng dụng hoặc nặng thì bạn phải cung cấp tài nguyên cho nó nhiều tài nguyên hơn những VMs khác trên cùng một host machine.

Những VMs chạy trên host machine thường được gọi là guest machine. Guest machine này sẽ chứa tất cả những thứ mà hệ thống cần để chạy ứng dụng như hệ điều hành (OS), system binaries và libraries. VMs chạy trên hệ điều hành của host machine và không thể truy cập trực tiếp đến phần cứng mà phải thông qua hệ điều hành.

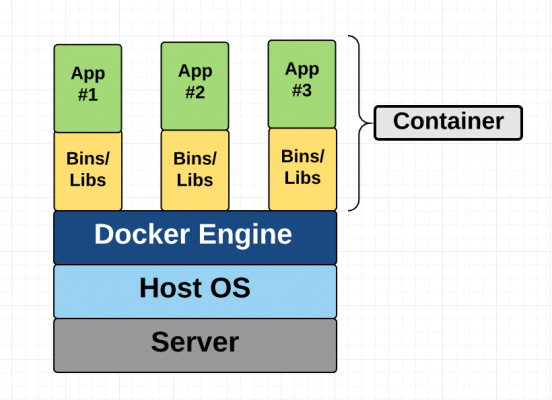
Để khắc phục những hạn chế trên của **Virtualization**, **Containerization** ra đời.

**Containerization là gì ?**

Containerization là một kỹ thuật trong đó virtualization được đưa containerization lên mức hệ điều hành. Trong containerization, ta virtualization tài nguyên hệ điều hành, sẽ hiệu quả hơn vì không có hệ điều hành khách nào tiêu thụ tài nguyên máy chủ, vì container chỉ sử dụng hệ điều hành của máy chủ và chỉ chia sẻ thư viện & tài nguyên có liên quan khi được yêu cầu. Các nhị phân và thư viện cần thiết của container chạy trên kernel host dẫn đến việc xử lý và thực thi nhanh hơn.

**Container là gì ?**

Container không giống như VMs, Container không cung cấp sự ảo hóa về phần cứng. Một Container cung cấp ảo hóa ở cấp hệ điều hành bằng một khái niệm trừu tượng là “user space”. Sự khác nhau lớn nhất của Container và VMs là Container có thể chia sẻ host system’s kernel với các container khác. Cùng xem mô hình bên dưới để hiểu rõ hơn về Container.



Dựa vào sơ đồ bên trên. Các bạn có thể thấy các gói container chỉ là một user space bao gồm ứng dụng, system binaries và libraries mà không cần guest OS hoặc ảo hóa phần cứng như VMs. Đây là cái mà làm cho các container nhẹ hơn (lightweight). Các container sẽ chạy trên công nghệ cụ thể ở đây là Docker Engine.

**Ưu điểm :**

Các container nhỏ và nhẹ hơn vì chúng có chung nhân hệ điều hành.

Không mất nhiều thời gian để khởi động (trong một vài giây).

Hiệu suất cao với việc sử dụng tài nguyên thấp hơn

**Tại sao nên sử dụng docker**

Tiện lợi: Khi bạn được phân bổ vào dự án mới, bạn đọc file README, thấy dự án hiện tại cần cài rất nhiều thứ, nào là ruby, rồi rails, redis, mysql, nginx, ... mỗi thứ lại phải kèm theo version bao nhiêu, .... Bạn lên google search cách cài đặt, config những thứ này sẽ mất một khoảng thời gian khá lâu. Chưa hết, có thể chưa cài xong đã conflic tùm lum, cái nọ xung đột cái kia chẳng hạn, lại còn ảnh hưởng tới những chương trình cũ đã cài đặt trong máy nữa chứ, thôi cài lại luôn cả hệ điều hành cho máy. Thế là mất thời gian, mệt mỏi với nhưng thao tác phụ mà chưa tập trung được vào việc chính. Nhưng với docker mọi thứ đơn gian hơn nhiều, chỉ vài dòng lệnh thôi, bạn sẽ có thể nhanh chóng tạo được môi trường ảo hóa chứa đầy đủ những cài đặt cần thiết cho project rồi.

Dễ dàng sử dụng: Docker rất dễ cho mọi người sử dụng từ developers, systems admins, architects…v…v.. nó tận dụng lợi thế của container để build, test nhanh chóng. Có thể đóng gói ứng dụng trên laptop của họ và chạy trên public cloud, private cloud..v.v… “Build once, run anywhere”.

Tốc độ: Docker container rất nhẹ và nhanh, bạn có thể tạo và chạy docker container trong vài giây so sánh với VMs thì mỗi lần chạy VMs cần rất nhiều thời gian khởi động.

Khả năng di động: môi trường develop được dựng lên bằng docker có thể chuyển từ người này sang người khác mà không làm thay đổi cấu hình ở trong.

Chia sẻ: DockerHub là một “app store for docker images”. Trên DockerHub có hàng ngàn public images được tạo bởi cộng đồng. Dễ dàng tìm thấy những image mà bạn cần và chỉ cần pull về và sử dụng với một số sửa đổi nhỏ.

Môi trường chạy và khả năng mở rộng: Bạn có thể chia nhỏ những chức năng của ứng dụng thành các container riêng lẻ. Ví dụng Database chạy trên một container và Redis cache có thể chạy trên một container khác trong khi ứng dụng Node.js lại chạy trên một cái khác nữa. Với Docker, rất dễ để liên kết các container với nhau để tạo thành một ứng dụng, làm cho nó dễ dàng scale, update các thành phần độc lập với nhau.

**Khi nào sử dụng Docker?**

Khi bạn cần triển khai kiến trúc Mircoservices.

Khi ứng dụng bạn cần scale một cách linh hoạt.

Khi bạn cần build 1 lần và chạy ở nhiều máy khác nhau mà không cần quan tâm đến config.

Khi bạn cần một cách tiếp cận mới về building, shipping, running ứng dụng một cách nhanh chóng dễ dàng.

**Một số khái niệm cơ bản:**

**Docker Client:** là cách mà bạn tương tác với docker thông qua command trong terminal. Docker Client sẽ sử dụng API gửi lệnh tới Docker Daemon.

**Docker Daemon:** là server Docker cho yêu cầu từ Docker API. Nó quản lý images, containers, networks và volume.

**Docker Volumes:** là cách tốt nhất để lưu trữ dữ liệu liên tục cho việc sử dụng và tạo apps.

**Docker Registry:** là nơi lưu trữ riêng của Docker Images. Images được push vào registry và client sẽ pull images từ registry. Có thể sử dụng registry của riêng bạn hoặc registry của nhà cung cấp như : AWS, Google Cloud, Microsoft Azure.

**Docker Hub:** là Registry lớn nhất của Docker Images ( mặc định). Có thể tìm thấy images và lưu trữ images của riêng bạn trên Docker Hub ( miễn phí).

**Docker Repository:** là tập hợp các Docker Images cùng tên nhưng khác tags. VD: golang:1.11-alpine.

**Docker Networking:** cho phép kết nối các container lại với nhau. Kết nối này có thể trên 1 host hoặc nhiều host.

**Docker Compose:** là công cụ cho phép run app với nhiều Docker containers 1 cách dễ dàng hơn. Docker Compose cho phép bạn config các command trong file docker-compose.yml để sử dụng lại. Có sẵn khi cài Docker.

**Docker Swarm:** để phối hợp triển khai container.

**Docker Services:** là các containers trong production. 1 service chỉ run 1 image nhưng nó mã hoá cách thức để run image — sử dụng port nào, bao nhiêu bản sao container run để service có hiệu năng cần thiết và ngay lập tức.

### 2.7 Giới thiệu về Jenkins

**Jenkins là gì?**

**Jenkins** là một máy chủ tự động hóa nguồn mở, độc lập, có thể được sử dụng để tự động hóa tất cả các loại tác vụ liên quan đến xây dựng, thử nghiệm và phân phối hoặc triển khai phần mềm.Jenkins có thể được cài đặt thông qua các gói hệ thống gốc, Docker hoặc thậm chí chạy độc lập bởi bất kỳ máy nào có cài đặt Môi trường thời gian chạy Java (JRE).  
Nó tích hợp các source code của các members trong team lại nhanh chóng một cách liên tục, theo dõi sự thực thi và trạng thái thông qua các bước kiểm thử (**Integration test**, **units test**). Tất nhiên là nhằm giúp sản phẩm chạy ổn định.



**Quá trình hình thành**

Năm 2004, Kohsuke Kawaguchi – một lập trình viên Java tại Sun vào một ngày cảm thấy mệt mỏi vì các mã nguồn của mình thường xuyên bị fail.

Ông ta muốn tìm cách biết là trước khi đưa code vào deploy, liệu code có hoạt động hay không.

Vì vậy, Kawaguchi đã xây dựng một hệ thống server tự động hóa cho chính server và cho Java có thể kiểm tra tình trạng code trước khi deploy, được gọi là **Hudson**. **Hudson** trở nên phổ biến tại Sun và thịnh hành sang các công ty khác dưới dạng opensource.

Đến năm 2011, với việc Oracle mua lại Sun gây nên sự tranh chấp với cộng đồng nguồn mở, Jenkins được forked từ Hudson sau đó. Cả Hudson và Jenkins đều tiếp tục tồn tại, mặc dù Jenkins có phần phổ biến hơn.

Năm 2014 Kawaguchi trở thành CTO của CloudBees, một công ty cung cấp các sản phẩm dựa trên nền tảng Jenkins.

**CI là gì? CD là gì?**



**CI là viết tắt của Continuous Integration**

Là tích hợp liên tục, nhằm liên tục tích hợp các source code của các thành viên trong team lại một cách nhanh chóng.

Chu trình làm việc

* Bước đầu tiên, các thành viên trong team dev sẽ bắt đầu pull code mới nhất từ repo về branch để thực hiện các yêu cầu chức năng nhất định.
* Tiếp đó là quá trình lập trình và test code để đảm bảo chất lượng của chức năng cũng như toàn bộ source code.
* Thành viên code xong thì sẵn sàng cho việc commit vào branch develop của team.
* Thành viên cập nhật code mới từ repo về local repo
* Merge code và giải quyết conflict.
* Build và đảm bảo code pass qua các tests dưới local.
* Commit code lên repo
* Máy chủ CI lắng nghe các thay đổi code từ repository và có thể tự động build/test, sau đó đưa ra các thông báo (pass/failure) cho các thành viên.

**CD là viết tắt của Continuous Delivery**

**Continuous Delivery** là chuyển giao liên tục, là 1 tập hợp các kỹ thuật để triển khai tích hợp souce code trên môi trường **staging** ( một môi trường rất giống với môi trường **production**).

### 2.8 Giới thiệu về hệ quản trị cơ sở dữ liệu PostgreSQL

**PostgreSQL là gì?**

PostgreSQL là một hệ thống quản trị cơ sở dữ liệu quan hệ và đối tượng (object-relational database management system) miễn phí và nguồn mở (RDBMS) tiên tiến nhất hiện nay. khả năng mở rộng cao và tuân thủ các tiêu chuẩn kỹ thuật. Nó được thiết kế để xử lý một loạt các khối lượng công việc lớn, từ các máy tính cá nhân đến kho dữ liệu hoặc dịch vụ Web có nhiều người dùng đồng thời.

* PostgreSQL được phát triển bởi PostgreSQL Global Development Group, Phát hành lần đầu: 08/07/1996
* PostgreSQL linh động có thể chạy được trên nhiều nền tảng khác nhau như Mac OS X, Solaris và Windows.
* PostgreSQL là một phần mềm mã nguồn mở miễn phí bởi vậy PostgreSQL có thể được dùng, sửa đổi và phổ biến bởi bất kỳ ai cho bất kỳ mục đích nào.
* PostgreSQL có tính ổn định cao.
* PostgreSQL là hệ thống quản lý cơ sở dữ liệu đầu tiên triển khai tính năng kiểm soát đồng thời nhiều phiên bản (MVCC).

**Tính năng PostgreSQL**

PostgreSQL tích hợp nhiều tính năng tuyệt vời giúp hỗ trợ nhà phát triển xây dựng app đáp ứng các chức năng phức tạp, truy vấn nhanh chóng và bảo mật duy trì tính toàn vẹn và độ tin cậy. Để đáng tin cậy hơn, Postgresql cung cấp các tùy chọn bảo mật, xác thực và khôi phục thảm họa khác nhau. PostgreSQL được chứng minh là có khả năng mở rộng cao cả về số lượng dữ liệu và số lượng người dùng có thể thao tác cùng lúc.

**Một số tính năng nổi bật của PostgreSQL**

* Câu truy vấn phức hợp (complex query)
* Thủ tục sự kiện (trigger)
* Các khung nhìn (view)
* Tính toàn vẹn của các giao dịch (integrity transactions)
* Việc kiểm tra truy cập đồng thời đa phiên bản (multiversion concurrency control)
* Truy vấn xử lý song song (parallel query)
* Sao chép dữ liệu dạng luồng (Streaming replication)

**Kiểu dữ liệu**

* Nguyên hàm: Số nguyên, số, chuỗi, Boolean
* Cấu trúc: Date/Time, Array, Phạm vi, UUID
* Document: JSON/JSONB, XML, Key-value (Hstore)
* Hình học: Điểm, Đường thẳng, Vòng tròn, Đa giác
* Tùy chỉnh: Composite, Các kiểu tùy chỉnh

**Toàn vẹn dữ liệu**

* UNIQUE, NOT NULL
* Primary Keys
* Foreign Keys
* Ràng buộc loại trừ
* Khóa hàm số, Khóa khuyến nghị
* Đồng quy, hiệu suất
* Lập danh mục: B-tree, Multicolumn, Expressions, Partial
* Lập danh mục nâng cao: GiST, SP-Gist, KNN Gist, GIN, BRIN, Bloom filters
* Trình lập kế hoạch / trình tối ưu hóa truy vấn phức tạp, quét index-only, thống kê số liệu trên nhiều cột.
* Giao tác, Giao tác dạng nest (thông qua lưu điểm)
* Điều khiển đồng thời nhiều phiên bản (MVCC)
* Truy vấn đọc song song
* Phân vùng bảng
* Tất cả các mức độ giao dịch độc lập được xác định trong tiêu chuẩn SQL, bao gồm cả Serializable
* Độ tin cậy, phục hồi sau thảm hoạ
* Ghi nhật ký ghi trước (Write-ahead Logging - WAL)
* Replication: Không đồng bộ, Đồng bộ, Logical
* Khôi phục điểm-theo-thời gian (Point-in-time-recovery - PITR), active standbys
* Không gian bảng

**Bảo mật**

* Xác thực: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate và các hình thức khác
* Hệ thống kiểm soát truy cập mạnh mẽ
* Bảo mật cấp độ cột và hàng

**Khả năng mở rộng**

* Phương pháp lưu trữ
* Ngôn ngữ thủ tục: PL / PGSQL, Perl, Python (và nhiều ngôn ngữ khác)
* Trình wrapper dữ liệu ngoài: kết nối với các cơ sở dữ liệu hoặc luồng khác với giao diện SQL chuẩn
* Và nhiều tiện ích mở rộng cung cấp chức năng bổ sung, bao gồm cả PostGIS
* Tìm kiếm văn bản:
* Hỗ trợ các bộ ký tự quốc tế, ví dụ: thông qua ICU collations
* Tìm kiếm văn bản đầy đủ

# CHƯƠNG III. KHẢO SÁT ĐỀ TÀI

## 1. Khảo sát đề tài

1.1 Nhiệm vụ cơ bản

1.2 Cơ cấu tổ chức và nhiệm vụ

1.3 Quy trình xử lý

## 2. Mô tả hệ thống

**2.1 Mô hình tiến trình nghiệp vụ**

**2.2 Biểu đồ hoạt động và giải thích ký hiệu**

# CHƯƠNG IV. PHÂN TÍCH HỆ THỐNG

## 1. Phân tích chức năng nghiệp vụ

**1.1 Mô hình hóa chức năng nghiệp vụ**

**1.2 Mô hình hóa tiến trình nghiệp vụ**

**1.3 Đặc tả tiến trình nghiệp vụ**

## 2. Phân tích dữ liệu nghiệp vụ

**2.1 Mô hình dữ liệu ban đầu**

**2.2 Chuẩn hóa dữ liệu**

**2.3 Đặc tả dữ liệu**

# CHƯƠNG V. THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG

## 1. Thiết kế tổng thể

### 1.1 Xác định tiến trình hệ thống

### 1.2 Xác định kho dữ liệu hệ thống

### 1.3 DFD hệ thống

## 2. Thiết kế kiểm soát

### 2.1 Xác định nhóm người

### 2.2 Phân định quyền hạn nhóm người dùng

## 3. Thiết kế cơ sở dữ liệu

### 3.1 Đánh giá nhu cầu bảo mật

### 3.2 Đánh giá nhu cầu cải thiện tính hiệu quả

### 3.3 Mô hình dữ liệu hệ thống

### 3.4 Đặc tả bảng dữ liệu

## 4. Thiết kế giao diện người dùng

### 4.1 Thiết kế giao diện người dùng

### 4.2 Xây dựng giao diện người dùng

## 5. Thiết kế xử lý dữ liệu người dùng

### 5.1 Thiết kế các đầu service API của hệ thống

### 5.2 Xây dựng luồng xử lý hệ thống

## 6. Thiết kế triển khai

# CHƯƠNG VI. KẾT LUẬN VÀ TÀI LIỆU THAM KHẢO

**1. Kết luận đề tài**

**2. Tài liệu tham khảo**

[1] <https://vuejs.org/guide/introduction.html#api-styles>

[2] <https://viblo.asia/p/gioi-thieu-ant-design-L4x5xwzblBM>

[3] <https://viblo.asia/p/phan-1-tong-quan-ve-entity-framework-core-4dbZNQNaKYM>

[4] https://vietnix.vn/api-la-gi/

[5] <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures>

[6] https://topdev.vn/blog/redis-la-gi/

[7] https://topdev.vn/blog/rabbitmq-la-gi/

[8] <https://viblo.asia/p/minio-object-storage-server-nhu-aws-s3-LzD5d0AW5jY>

[9] https://viblo.asia/p/docker-chua-biet-gi-den-biet-dung-phan-1-lich-su-ByEZkWrEZQ0

[10] <https://topdev.vn/blog/docker-la-gi-kien-thuc-co-ban-ve-docker/>

[11] https://www.jenkins.io/doc/

[12] <https://topdev.vn/blog/jenkins-la-gi/>

[13] <https://viblo.asia/p/tim-hieu-he-quan-tri-co-so-du-lieu-postgresql-m68Z0eLdlkG>

[14] <https://en.wikipedia.org/wiki/PostgreSQL>

[15] https://quantrimang.com/cong-nghe/co-so-du-lieu-phi-quan-he-nosql-160708