Giới thiệu ASP.NET Core (Web API)

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](https://github.com/dotnet/core)

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

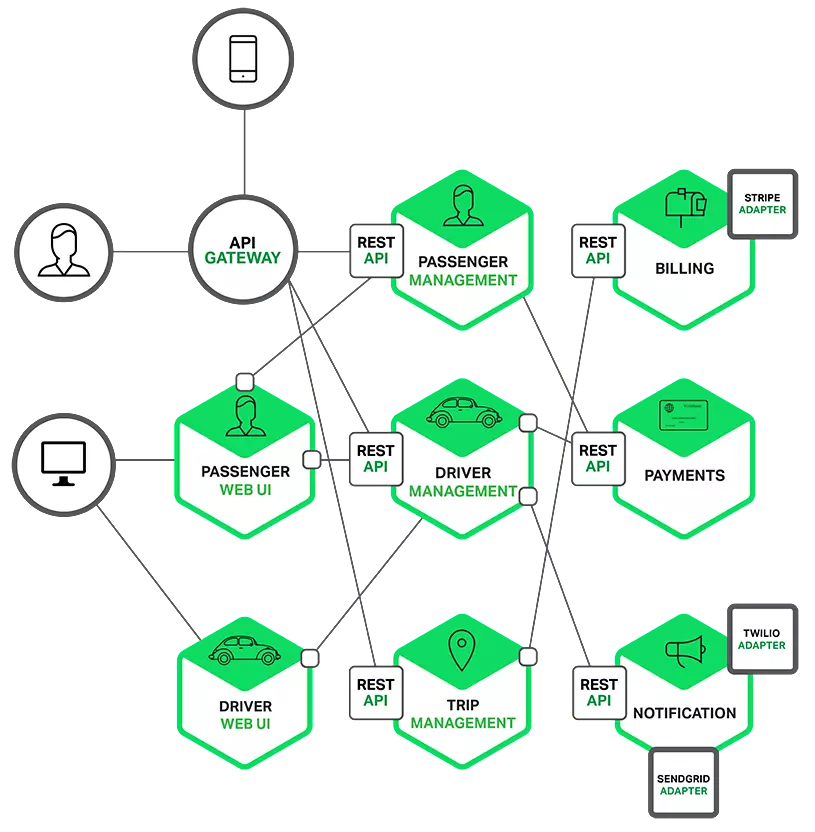
Để sử dụng hiệu quả cần có kiến thức chuyên sâu, có kinh nghiệm backend tốt

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 Microservices Architecture

**Nền tảng cả kiến trúc microservices là xây dựng một ứng dụng mà ứng dụng này là tổng hợp của nhiều services nhỏ và độc lập có thể chạy riêng biệt, phát triển và triển khai độc lập.**  
Ta có thể giải quyết các vấn đề của ứng dụng một khối bằng kiến trúc microservices (nhiều dịch vụ nhỏ). Ý tưởng là chia nhỏ ứng dụng lớn ra thành các dịch vụ nhỏ kết nối với nhau.  
Mỗi dịch vụ nhỏ thực hiện một tập các chức năng chuyên biệt như quản lý đơn hàng, quản lý khách hàng. Mỗi dịch vụ là một ứng dụng nhỏ có kiến trúc đa diện lõi là business logic kết nối ra các adapter khác nhau. Một số dịch vụ nhỏ lộ ra giao tiếp lập trình API cho dịch vụ nhỏ khác hay ứng dụng client gọi tới. Khi vận hành, mỗi dịch vụ nhỏ được chạy trong một máy ảo hoặc Docker container.

 Mỗi vùng chức năng giờ được thực thị bởi một dịch vụ nhỏ. Ứng dụng web cũng có thể chia nhỏ hơn chuyên cho từng đối tượng người dùng (hành khách/tài xế). Thiết kế giao diện cho từng đối tượng người dùng giúp tối ưu trải nghiệm tốt hơn, tốc độ nhanh hơn, dễ tương thích hơn trong khi chức năng tối giản hơn.  
Một số khái niệm về microservices nói về quá trình chia tách ứng dụng monolithic thành nhóm các services độc lập. Tuy nhiên, theo quan điểm của tôi, microservices không chỉ về chia tách các servcies sẵn có trong monolithic.  
Điều quan trọng chính là nhìn vào các tính năng trong một ứng dụng monolithic, ta có thể nhận biết, xác định các yêu cầu và khả năng cần thiết để đáp ứng một nghiệp vụ. Sau đó từng nghiệp vụ này sẽ được xây dựng thành những service nhỏ, độc lập. Những services này có thể sử dụng các nền tảng công nghệ khác nhau và phục vụ một mục đích cụ thể và có giới hạn.

**Ưu nhược điểm của Microservices**

**Ưu điểm**

Cho phép dễ dàng continuous delivery và deployment các ứng dựng lớn, phức tạp:

Cải thiện khả năng bảo trì - mỗi service tương đối nhỏ do đó dễ hiểu và thay đổi hơn

Khả năng testing dễ dàng hơn - các services nhỏ hơn và nhanh hơn để test

Khả năng triển khai tốt hơn - các services có thể được triển khai độc lập

Cho phép các services được phát triển bởi những team khác nhau. Mỗi team có thể phát triển, thử nghiệm, triển khai và mở rộng quy mô dịch vụ của mình một cách độc lập với tất cả các team khác.

Giảm thiểu rủi ro: Nếu có lỗi trong một service thì chỉ có service đó bị ảnh hưởng. Các services khác sẽ tiếp tục xử lý các yêu cầu. Trong khi đó, một thành phần hoạt động sai của kiến trúc một khối có thể làm ảnh hưởng toàn bộ hệ thống.

Dễ dàng thay đổi sử dụng các công nghệ mới: Khi triển khai các services bạn có thể lựa chọn nhiều công nghệ mới. Tương tự khi có thay đổi lớn đối với các services hiện có bạn có thể dễ dàng thay đổi công nghệ.

**Nhược điểm**

Các nhà phát triển phải đối phó với sự phức tạp của việc tạo ra một hệ thống phân tán:

Cần implement việc communication giữa các inter-services

Handle partial failure là rất phức tạp vì một luồng xử lý cần đi qua nhiều services

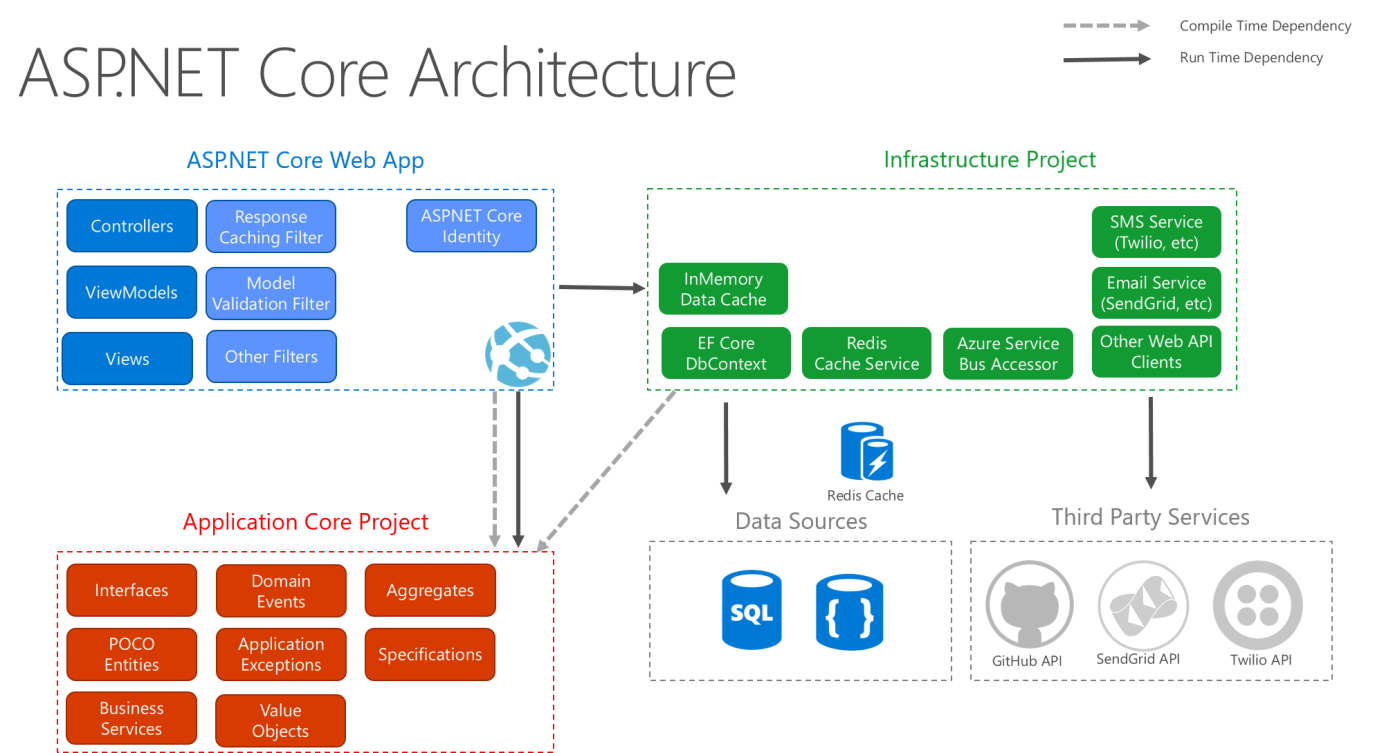
Việc thực hiện các requests trải rộng trên nhiều services khó khăn hơn, điều này cũng đòi hỏi sự phối hợp cẩn thận giữa các teams

Khó khăn trong việc đảm bảo toàn vẹn CSDL nếu triển khai theo kiến trúc cơ sở dữ liệu phân vùng

Triển khai và quản lý microservices nếu làm thủ công theo cách đã làm với ứng dụng một khối phức tạp hơn rất nhiều

Phải xử lý sự cố khi kết nối chậm, lỗi khi thông điệp không gửi được hoặc thông điệp gửi đến nhiều đích đến vào các thời điểm khác nhau

**Clean Architectue**

****

Giới thiệu công nghệ làm fontend

**Giới thiệu 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?**

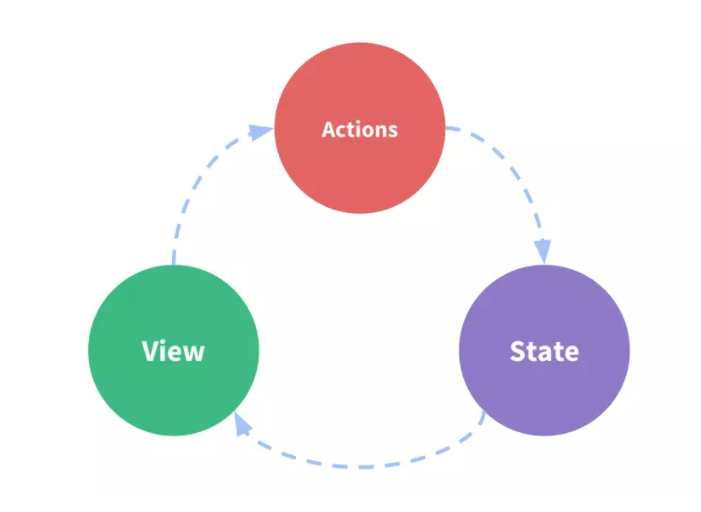
**Vuex** là thư viện giúp quản lý trạng thái các component trong Vue.js, nó là nơi lưu trữ tập trung cho tất cả các component trong một ứng dụng, với nguyên tắc trạng thái chỉ có thể được thay đổi theo kiểu có thể dự đoán.

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

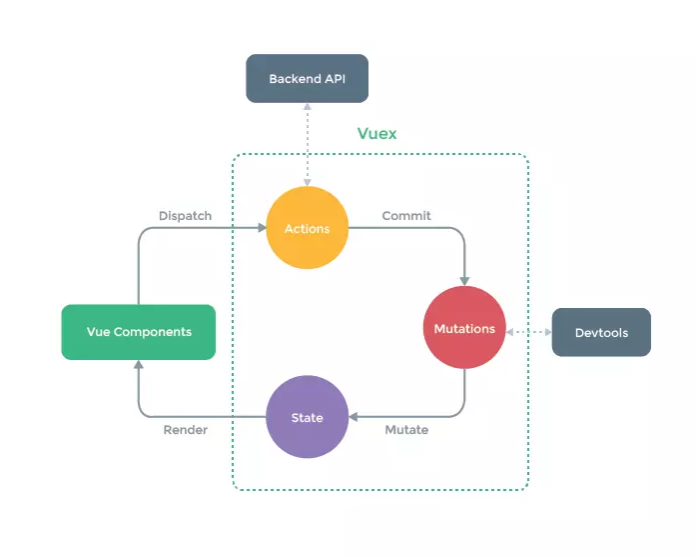
**State**: Trạng thái, là nơi khởi nguồn để thực hiện ứng dụng.

**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

**Hệ quản trị cơ sở dữ liệu MongoDB(NoSQL)**

**NoSql là gì?**

NoSQL là một dạng CSDL mã nguồn mở và được viết tắt bởi: None-Relational SQL hay có nơi thường gọi là Not-Only SQL.

NoSQL được phát triển trên Javascript Framework với kiểu dữ liệu là JSON và dạng dữ liệu theo kiểu key và value.

NoSQL ra đời như là một mảnh vá cho những khuyết điểm và thiếu xót cũng như hạn chế của mô hình dữ liệu quan hệ RDBMS (Relational Database Management System - Hệ quản trị cơ sở dữ liệu quan hệ) về tốc độ, tính năng, khả năng mở rộng,...

Với NoSQL bạn có thể mở rộng dữ liệu mà không lo tới những việc như tạo khóa ngoại, khóa chính, kiểm tra ràng buộc.

NoSQL bỏ qua tính toàn vẹn của dữ liệu và transaction để đổi lấy hiệu suất nhanh và khả năng mở rộng.

NoSQL được sử dụng ở rất nhiều công ty, tập đoàn lớn, ví dụ như FaceBook sử dụng Cassandra do FaceBook phát triển, Google phát triển và sử dụng BigTable,...

**MongoDB là gì?**



**MongoDB** là một hệ quản trị cơ sở dữ liệu mã nguồn mở, là CSDL thuộc NoSql và được hàng triệu người sử dụng.

MongoDB là một database hướng tài liệu (document), các dữ liệu được lưu trữ trong document kiểu JSON thay vì dạng bảng như CSDL quan hệ nên truy vấn sẽ rất nhanh.

Với CSDL quan hệ chúng ta có khái niệm bảng, các cơ sở dữ liệu quan hệ (như MySQL hay SQL Server...) sử dụng các bảng để lưu dữ liệu thì với MongoDB chúng ta sẽ dùng khái niệm là **collection** thay vì bảng

So với RDBMS thì trong MongoDB **collection** ứng với **table**, còn **document** sẽ ứng với **row** , MongoDB sẽ dùng các document thay cho row trong RDBMS.

Các collection trong MongoDB được cấu trúc rất linh hoạt, cho phép các dữ liệu lưu trữ không cần tuân theo một cấu trúc nhất định.

Thông tin liên quan được lưu trữ cùng nhau để truy cập truy vấn nhanh thông qua ngôn ngữ truy vấn MongoDB

**Ưu điểm của mongoDB.**

* Schema linh hoạt: Do MongoDB sử dụng lưu trữ dữ liệu dưới dạng Document JSON nên mỗi một collection sẽ các các kích cỡ và các document khác nhau.
* Cấu trúc đối tượng rõ ràng: Tuy rằng cấu trúc của dữ liệu là linh hoạt nhưng đối tượng của nó được xác định rất rõ ràng.
* Sử dụng bộ nhớ nội tại, nên truy vấn sẽ rất nhanh.
* MongoDB rất dễ mở rộng.
* Không có các join: Điều này cũng góp phần tạo nên tốc độ truy vấn cực nhanh trên mongoDB.
* MongoDB phù hợp cho các ứng dụng realtime.

**Nhược điểm của mongoDB**

* Điều đầu tiên phải kể đến ở đây là MongoDB không có các tính chất ràng buộc như trong RDBMS nên khi thao tác với mongoDB thì phải hết sức cẩn thận.
* MongoDB sử dụng sẽ hao tốn tài nguyên của hệ thống nhiều hơn RDBMS. Nhưng đến thời điểm hiện tại thì vấn đề này không còn là điều lo ngại nữa (Máy giờ trâu bò lắm rồi).

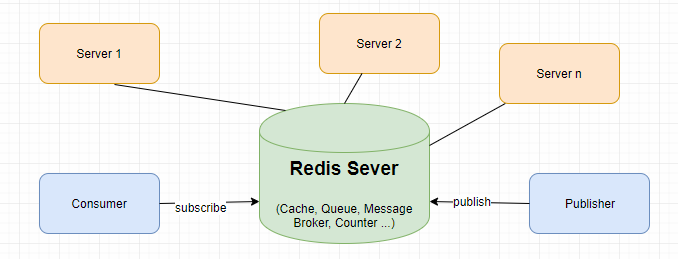
Một vài sự khác nhau cơ bản giữa MongDB và SQL DB

|  |  |
| --- | --- |
| **SQL DB** | **Mongo DB** |
| Table | Collection |
| Row | Document |
| Column | Field |
| Joins | Embeded documents, linking |

**Giới thiệu Redis**

**Redis là gì?**

**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.

**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).

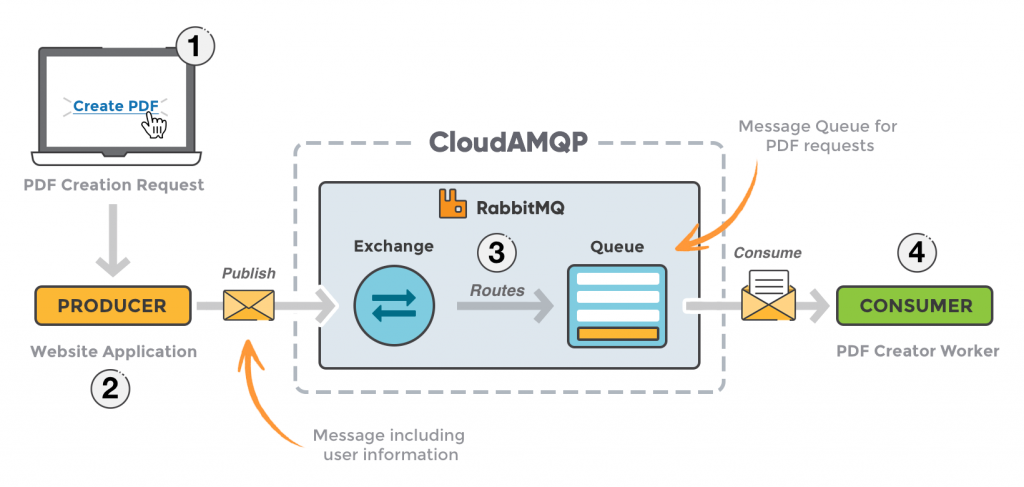
Có 2 hình thức giao tiếp cơ bản với một Message Broker:

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.



**Giới thiệu docker**

**Giới thiệu về K8s (Kubernetes)**

**Giới thiệu về Redis**

**Giới thiệu về MinIO**