**A) CONTENT**

[1. Jhipster 3](#_heading=h.gjdgxs)

[1.1. Jhipster là gì? 3](#_heading=h.30j0zll)

[1.2. Xây dựng app với Jhipster 3](#_heading=h.1fob9te)

[1.3. Microservice với Jhipster 10](#_heading=h.1fob9te)

[1.3.1. Microservice là gì 10](#_heading=h.3znysh7)

[1.3.2. So sánh với kiến trúc Monolith 11](#_heading=h.2et92p0)

[1.3.3. Cấu trúc Microservice 13](#_heading=h.tyjcwt)

[1.3.4. API Gateway 13](#_heading=h.3dy6vkm)

[1.3.5. Service Discovery với Jhipster Resgitry 16](#_heading=h.1t3h5sf)

[1.3.6. Service Discovery với Consul 16](#_heading=h.4d34og8)

1.3.7. Ưu và nhược điểm của Microservice 16

[2. Elasticsearch 17](#_heading=h.2s8eyo1)

[2.1. Elasticsearch là gì? 17](#_heading=h.17dp8vu)

[2.2. Cấu trúc của elasticsearch 17](#_heading=h.3rdcrjn)

[2.3. Cài đặt elastic stack và ví dụ với elasticsearch 18](#_heading=h.26in1rg)

[2.4. Elasticsearch APIs 23](#_heading=h.lnxbz9)

[2.4.1. Elasticsearch APIs 23](#_heading=h.35nkun2)

[2.4.2. ES Document APIs 23](#_heading=h.1ksv4uv)

[2.4.3. ES Search APIs 27](#_heading=h.2grqrue)

[2.4.4. ES Aggregation APIs 27](#_heading=h.44sinio)

[2.4.5. ES Index APIs 28](#_heading=h.2jxsxqh)

[2.4.6. ES Cluster APIs 29](#_heading=h.2jxsxqh)

[3. Kafka 30](#_heading=h.z337ya)

[3.1. Kafka là gì? 30](#_heading=h.3j2qqm3)

[3.2. Kafka core concept](#_heading=h.4i7ojhp) 32

[3.2.1. Kafka topics](#_heading=h.2xcytpi) 32

[3.2.2. Topic replication](#_heading=h.1ci93xb) 35

[3.2.3. Kafka producer](#_heading=h.3whwml4) 35

[3.2.4. Consumer & Comsumer Groups](#_heading=h.3whwml4) 38

[3.2.5. Advantage & Disadvantage](#_heading=h.3whwml4) 39

[3.2.6. Kafka Architecture](#_heading=h.3whwml4) 41

[4. Redis Cache](#_heading=h.41mghml) 41

[4.1. Giới thiệu về Redis](#_heading=h.4i7ojhp) 41

[4.2. Redis Commands](#_heading=h.4i7ojhp) 43

[4.3. Publishser và Subcriber với Redis](#_heading=h.4i7ojhp) 56

[5. Nguồn tham khảo](#_heading=h.41mghml) 58

**A) CONTENT**

# Jhipster

## Jhipster là gì?

1. Giới thiệu

Jhipster là một trong những dự án Open-source được tạo bởi Julien Dubois vào 10/2013. Và Jhipter được public lần đầu ra công chúng (phiên bản 0.3.1) vào 7/12/2013. Hiện tại jhipster được phát triển bởi 19 lập trình viên và hơn 430 đóng góp viên.

Jhipster là sự kết hợp hoàn hảo giữa các frameworks đã thành công khi lập trình web như: Bootstrap, Angular, Spring boot, React JS.

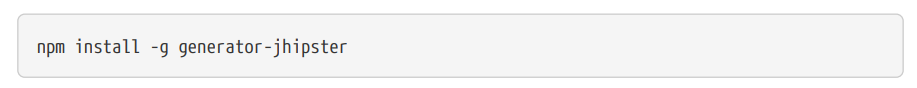
Jhipster là một công cụ tạo tự động core mã nguần giúp loại bỏ thời gian tạo ra các đoạn mã cơ bản từ đó tăng thời gian lập trình và cũng như tập trung vào logic của dự án

1. Cài đặt Genarator Jhipter

B1: Trước hết bạn cần cài đặt java, maven và git trước khi cài đặt Jhipster

B2: Sau đó cài Node.js từ Nodejs.org or Yarn

B3: Run dòng lệnh này trên command line:



## Xây dựng app với Jhipster

1. Genarating project

B1: Bạn cần tạo một folder để chứa dự án của bạn

B2:

Cách 1: Run dòng lệnh này trên command line:



Sau khi bạn chạy xong dòng code kia, thì bạn sẽ được hỏi các câu hỏi tương ứng với những gì bạn muốn cài trên dự án của bạn.



Giải thích các câu hỏi trên:

1: Which “type” of application would you like to create: Là kiểu kiến trúc bạn muốn trong dự án của bạn. Ở ví dụ này là Monolithic application

2: What is the base name of you application: Tên dự án của bạn. Ở ví dụ này là TwentyOnePoints

3: What is your default Java package name: Tên package của dụ án của bạn. Ở ví dụ này là org.jhipster.health

4: Do you want to use the Jhipster Registry to configure, monitor and scale your application: Bạn có muốn sử dụng Jhipter Registry để cài đặt và quản lý dự án của bạn hay không. Ở ví dụ này là No do nó là kiến trúc Monolith

5: Which “type” of authentication would you like to use: Kiểu xác thực mà bạn muốn sử dụng trong dự án là gì. Ở ví dụ này là JWT authentication (stateless, with a token)

6: Which “type” of database would you like to use: Loại Database mà bạn muốn sử dụng. Ở ví dụ này là SQL

7: Which ”production” database would you like to use: Database nào mà bạn muốn sử dụng trong chế độ sản phẩm. Ở ví dụ này là PostgreSQL

8: Which ”development” database would you like to use: Database nào mà bạn muốn sử dụng trong khi phát triển. Ở ví dụ này là H2 with disk-based persistence

9: Do you want to use the Spring cache abstraction: Bạn có muốn sử dụng cache cho dự án này không. Ở ví dụ này là Yes

10: Do you want to use Hiberate 2nd level cache: bạn có muốn sử dụng Hibernate 2nd không. Ở ví dụ này là Yes

11: Would you like to use Maven or Gradle for building the backend: Bạn thích sử dụng Maven hay Gradle cho xây dựng backend. Ở ví dụ này là Gradle

12: Which other technologies would you like to use: Công nghệ khác bạn muốn sử dụng trong dự án. Ở ví dụ này là Search engine using Elasticsearch

13: Which “framework” would you like to use for the client: Framework frontend bạn muốn sử dụng cho dự án. Ở ví dụ này là Anglar 6

14: Would you like to enable “SASS” support using the LibSass stylesheet preprocessor: Bạn có muốn kích hoạt hỗ trợ SASS cho tiền xử lý không. Ở ví dụ này là Yes.

15: Would you like to enable internationalization support: Bạn có muốn kích hoạt hỗ trợ đã ngôn ngữ không. Ở ví dụ này là Yes

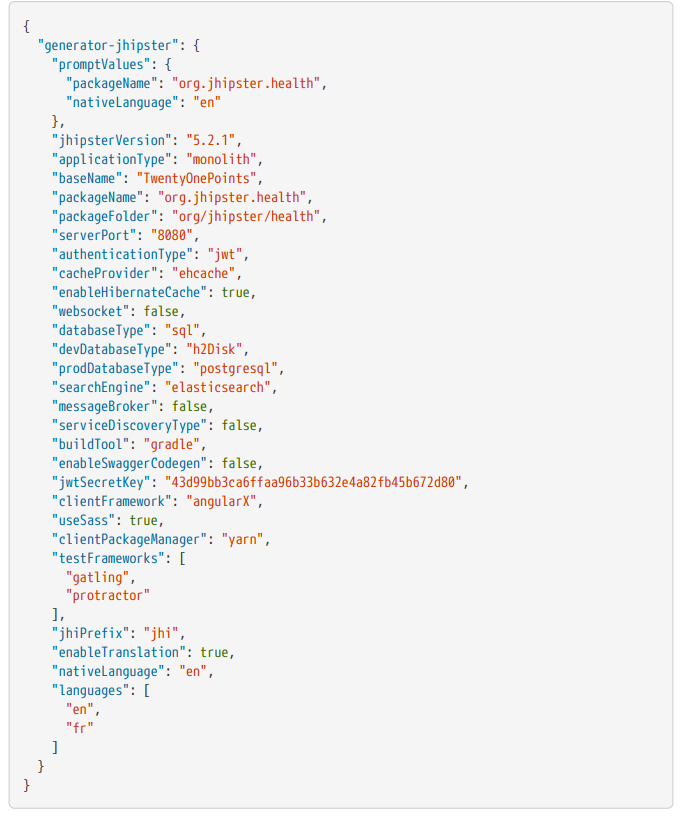
16: Please choose the native language of the application: Chọn ngôn ngữ mặc định cho dự án. Ở ví dụ này là English

17: Please choose additional languages to install: Chọn thêm ngôn ngữ. Ở ví dụ này là French

18: Resides Junit and jest, which testing frameworks would you like to use: Loại test nào bạn muốn sử dụng. Ở ví dụ này là Gatling, Protractor

19: Would you like to install other generators from the Jhipster Marketplace: Bạn có muốn cài đặt các trình tạo khác từ Jhipster Marketplace không. Ở ví dụ này là No

Cách 2: Cách khác để bạn tạo ra project đó là sử dụng file json (.yo-rc.json)



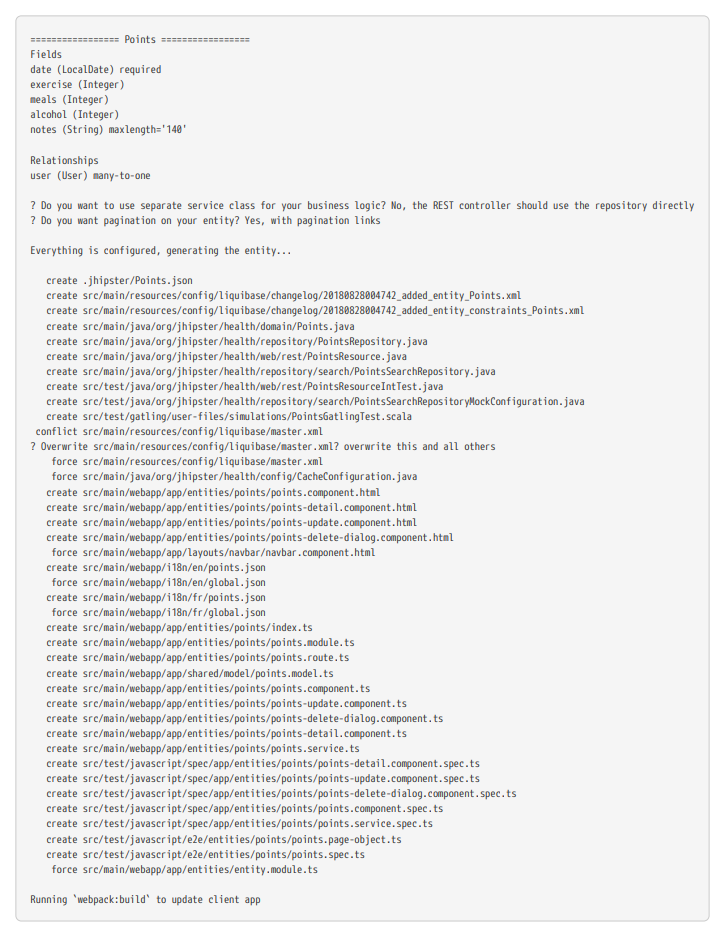
1. Genarating entities

Cho mỗi entity bạn thường muốn tạo ra như sau:

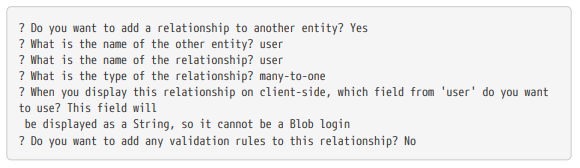
* A database table
* A liquibase change set
* A JPA entity class
* A Spring Data JpaRepository interface
* A Spring MVC RestController class
* An Angular router, controller, and service
* A HTML page

JDL-Studio là browser-based tool cho đĩnh nghĩa model của bạn với Jhipster Domain Language (JDL)

Cách tạo ra Entity: run command line: jhipster entiry [Tên entity]



Cách tạo relationship với bảng khác



## Microservice với Jhipster

### Microservice là gì?

1. Lịch sử của microservice

Theo Wikipedia, Microservice lần đầu tiên được sử dụng như một phong cách kiến trúc phổ biến tại một hội thảo của các kiến trúc sư phần mềm gần Venice vào tháng 2011 năm XNUMX. Vào tháng 2012 năm XNUMX, cùng một nhóm đã quyết định "Microservice" là một cái tên thích hợp hơn

1. Tại sao lại chọn microservice

Cho hầu hết các lập trình viên, nhóm dev và các tổ chức, thật dễ dàng để làm việc với project nhỏ và làm một dịch vụ rất là tốt. Không một service program đại diện cho cả ứng dụng vì vậy service có thể thay đổi frameworks với không mất giá cả cao. Các service khác nhau có thể viết bởi nhiều ngôn ngữ lập trình khác nhau như: Java, Ruby, Node….

Các nhà cung cấp Platform-as-a-service (PaaS) và chứa đã làm cho nó dễ dàng hơn để triển khai microservice . Tất cả các công nghệ cần để hỗ trợ cho kiến trúc monolith (Load balancing, discovery, process monitoring) được cung cấp bởi PaaS, bên ngoài container của bạn và nỗ lực triển khải trở nên dẽ dàng hơn.

1. Tự định nghĩa

Kiến trúc Microservice là kiểu kiến trúc ứng dụng mà ở đó ứng dụng được phát triển bằng một tập các service. Nó cung cấp các framework cho lập trình và triển khai và duy trì, bảo trì dự án. Và các service hoạt động một các độc lập với nhau

1. Nên sử dụng kiến trúc Microservice ở đâu?

Một số ví dụ có thể sử dụng kiến trúc microservice:

* **Website migration:** Một trang web phức tạp được lưu trữ trên nền tảng nguyên khối có thể được di chuyển sang nền tảng microservice dữa trên đám mây và dữa trên container
* **Media content:** Sử dụng kiến trúc Microservice, hình ảnh và tải sản video có thể được lưu trữ trong hệ thống lưu trữ đối tượng có thể mở rộng và được phục vụ trực tiếp cho web hoặc thiết bị di động.
* **Transactions and invoices:** Xử lý và đặt hàng thanh toán có thể được phân tách dưới dạng các đơn vị dịch vụ độc lập để các khoản thanh toán tiếp tục được chấp nhận và hóa đơn không hoạt động.
* **Data processing:** Nền tảng Microservice có thể mở rộng hỗ trợ đám mây cho các dịch vụ xử lý dữ liệu modul hiện có.

### Cấu trúc Microservice

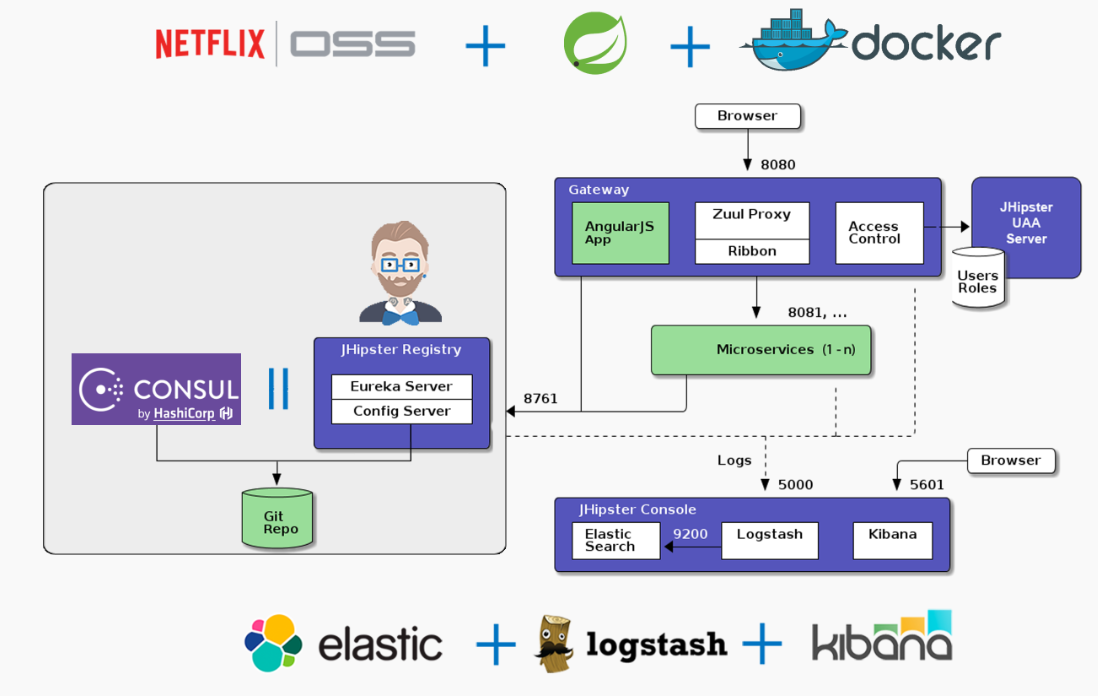
1. Cấu trúc chung



+ Management/orchestration: Thành phần này chịu trách nhiệm đặt các dịch vụ trên các nút, xác định lỗi, các dịch vụ cân bằng lại trên các nút… Thông thường thành phần này là một công nghệ ngoài luồng như Kubernetes, thay vì một cái gì đó được xây dựng tùy chỉnh.

+ API Gateway: API Gateway là điểm vào cho client. Thay vì gọi trực tiếp service, khách hàng gọi qua API Gateway, chuyển tiếp service đó đến các dịch vụ thích hợp ở phía sau.

1. Cấu trúc microservice trong jhipster



Kiến trúc Jhipster Microservice hoạt động gồm các thành phần sau:

+ **A Gateway**: Gateway là một ứng dụng do Jhipster tạo ra (Sử dụng microservice gateway) xử lý lưu lượng web và phục vụ ứng dụng Angular/React. Có thể có một số cộng khác nhau, nếu bạn muốn theo dõi các phụ trợ cho mẫu frontends, nhưng nó không phải bắt buộc. (Xem chi tiết trong phần Jhipster Gateway )

+ **The Jhipster Registry**: là một ứng dụng runtime trạy trên đó và tất cả các ứng dụng cần đăng ký và nhận cấu hình của chúng. Nó cũng cung cấp bảng điều kiển giám giát thời gian chạy

+ **Consul**: Consul là một dịch vụ khám phá dịch vụ, cũng như một của hàng key/value. Nó có thể được sử dụng như một thay thế cho **Jhipster Registry**

**+ Microservices:** Nó được tạo ra bằng Jhipster (microservice application) để xử lý các Rest request.

### So sánh với kiến trúc Monolith

Chart, bubble chart

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Key** | **Monolithic Architecture** | **Microservice Architecture** |
| Basic | Monolithic Architecture được xây dựng như một hệ thống lớn và thưởng có 1 code base | Mocroservices Architecture được xây dựng lên từ các module nhỏ độc lập với nhau dựa trên business functionality |
| Scale | Nó khó để mở rộng trên các lệnh | Dễ dàng mở rộng và maintain |
| Database | Nó chia sẻ database với nhau | Mỗi project và module có thể có các database của chính nó |
| Deployment | Một lượng lớn code base làm cho IDE và thời gian build tăng lên | Mỗi project hoạt động độc lập với nhau và với một lượng nhỏ code nên tổng thời gian build và deployment giảm đáng kể |
| Tightly coupled and loosely coupled | Nó cực kì khó để thay đổi công nghệ hoặc ngôn ngữ khác bởi vì nó có quan hệ chặt chẽ với nhau và phụ thuộc lẫn nhau | Dễ dàng thay đổi về công nghệ hoặc framework bời vì mỗi module và project hoạt động độc lập với nhau |

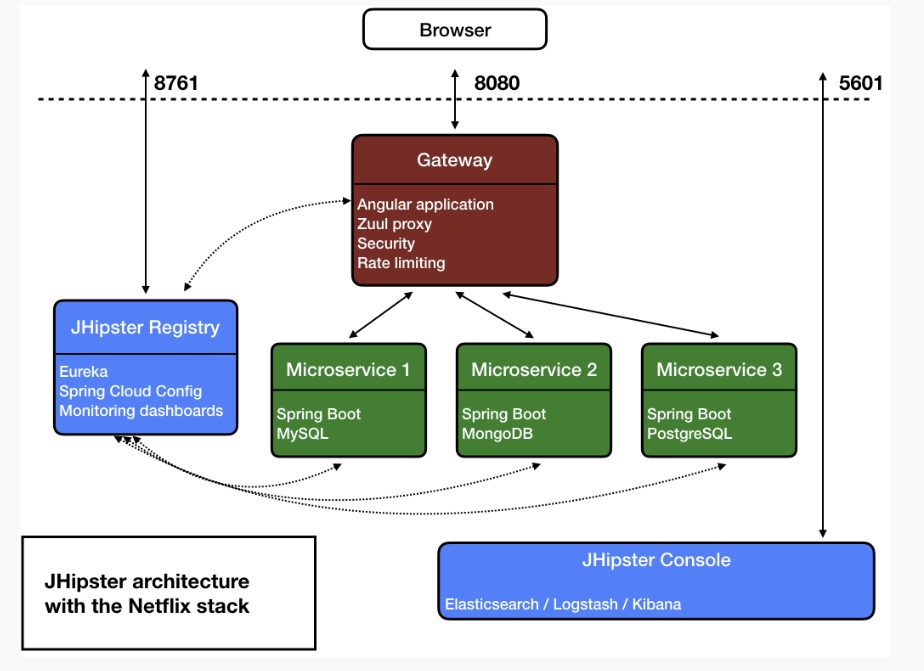
### API Gateway

1. Giới thiệu:

Jhipster có thể tạo ra API gateway. A gateway là một ứng dụng Jhipster thông thường, vì vậy bạn có thể sử dụng các lựa chọn của jhipster và phát triển workflows cho dự án nhưng nó cũng hoạt động như một cánh cổng để các request có thể đến được microservice của bạn.

Đặc biệt hơn, Gateway cung cấp HTTP routing và load balancing, bảo mật, tài liệu API ... cho tất cả microservice.

1. Architecture diagram



1. HTTP routing

Khi gateway và các ứng dụng microservice của bạn chạy, Nó sẽ đăng kí lên Registry (thông qua config eureka.client.serviceUrl.defaultZone) trong file application.yml or application.properties

Gateway sẽ tự động xử lý tất cả các request để chuyển đến các microservice tương ứng sử dụng tên của ứng dụng microservice của bạn.

Ví dụ:

+ Tên ứng dụng microservice là: app1

+ Gateway chạy ở port 8080

+ Truy nhập apit của app1: /api/foos

* Url: http://localhost:8080/services/app1/api/foos

1. Automatic documentation

Gateway sẽ hiển thị các định nghĩa API Swagger về các service mà nó hướng tới để bạn có tất cả các công cụ hưu ích để sử dụng

1. Rate limiting

Rate limiting là một chức năng nâng cao nó sử dụng Bucket4j và Hazelcast để cung cấp số lượng các serice trong microservice

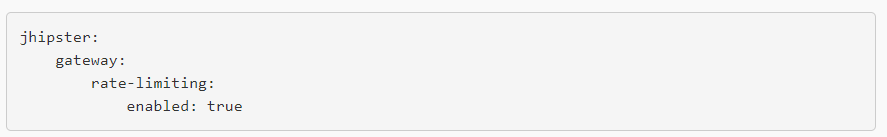
Gateway cung cấp chức năng rate-limiting, vì vậy Rest request có thể giới hạn:

* + - * Bởi IP address (cho user ẩn danh)
      * Bởi User login (cho user đã login)

Jhipster sẽ sử dụng Bucket4j và Hazelcast để tính toán số lượng các request và sẽ gửi HTTP 429 (QUÁ NHIỀU REQUEST) lỗi khi vượt quá giới hạn.

Mặc định giới hạn cho mỗi user là 100000 API calls cho mỗi giờ.

Kích hoạt nó bằng cách config trong file application.yml or application.properties



1. Access control policy

Mặc định tất cả các đăng kí bởi microservice sẽ tồn tại và có sẵn trong Gateway. Nhưng nếu bạn muốn chỉ một số API có thể truy cập bởi API Gateway thì bạn có thể config ở file file application.yml or application.properties



### Service Discovery với Jhipster Registry

* Jhipster Registry là một runtime application được cung cấp với nhóm của Jhipster.
* Nó là một mã nguần mở, Apache 2-licensed application và source code của nó có sẵn trên github: <https://github.com/jhipster/jhipster-registry>
* Jhipster Registry có 3 chức năng chính:
  + Eureka server: cung cấp như một máy chủ khám phá cho các ứng dụng. Máy chủ này duy trì và phân phối một danh sách động các trường hợp ứng dụng có sẵn sau đó được sử dụng bởi các dịch vụ microservice để thực hiện các yêu cầu HTTP định tuyến và cân bằng tải
  + Spring cloud config server: Nó cung cấp runtime config cho tất cả các ứng dụng
  + Đây là một máy chủ quản trị, với bẳng điều khiển để giám sát và quản lý các ứng dụng

### Service Discovery với Consul

### Ưu và nhược điểm của Microservice

1. Ưu điểm

* Microservice giúp dễ dàng continuos delivery và deployment các ứng dụng lớn phức tạp
  + Cải thiện về khả năng bảo trì, mỗi service tương đối nhỏ vì vậy dễ hiểu và thay đổi hơn
  + Khả năng testing dễ dàng hơn, vì những services nhỏ hơn và nhanh hơn để test
  + Khả năng triển khai tốt hơn, các service có thể được triển khai độc lập với nhau
* Giảm thiểu được rủi ro: Ví dụ như có lỗi trong một service thì chỉ có service đó bị ảnh hưởng. Các services khác sẽ vẫn tiếp tục xử lý các request khác
* Dễ dàng thay đổi để sử dụng các công nghệ mới, Khi thực hiện triển khai các services bạn có thể lựa chọn nhiều công nghệ mới phù hợp hơn.
* Agility (Nhanh nhẹn) Microservice hỗ trợ phát triển theo mô hình Agile

1. Nhược điểm

* Các nhà phát triển phần mềm phải đối mặt với sự phức tạp của việc tạo ra một hệ thống phân tán:
  + Cần phải implement việc communication giữa các inter-services
  + Việc Handle parial failture là rất phức tạp vì một luồng xử lý cần đi qua nhiều service
  + Việc thực hiện những requests ở nhiều services khó khăn hơn, nó cũng đòi hỏi sự phối hợp cẩn thận giữa các teams
  + Những khó khăn trong việc đảm bảo toàn vẹn CSDL nếu triển khai theo kiến trúc cơ sở dữ liệu phân vùng
* Việc triển khai và quản lý microservice nếu làm theo các thủ công so với ứng dụng một khối phức tạp hơn nhiều
* Phải xử lý sự cố khi có kết nối chậm

# Elasticsearch

## Elasticsearch là gì?

Elasticsearch là một công cụ phân tích và tìm kiếm dựa trên engine **Apache Lucene,** được viết bằng ngôn ngữ Java và cũng là kho dữ liệu không theo cấu trúc như SQL.

An index, search engine, analytics database, big data solution, document-oriented database, …

Elasticsearch là cơ sở dữ liệu NoSQL (Not-only SQL).

Mục đích của elasticsearch là để làm giảm thời gian tìm kiếm dữ liệu

## Cấu trúc của elasticsearch

Dưới đây là liệt kê về cấu trúc và cách tổ chức dữ liệu của elasticsearch:

* **Index** = cơ sở dữ liệu (a logical namespace that points to one or more **shards**)
* Mỗi Index có nhiều **Type** (giống table trong SQL dành cho phiên bản trước elasticsearch 6.0)

“**Thực tế 1 index chỉ chứa 1 type**”

Thông tin chi tiết: https://www.elastic.co/blog/removal-of-mapping-types-elasticsearch

* Mỗi 1 Index có 1 hoặc nhiều **Shard**
* Các Shard tồn tại trên các **Node** khác nhau = phần quan trọng giúp thực hiện công việc một cách nhanh chóng
* Shard có: **Primary shard và Replica shard** tồn tại trên các Node khác nhau. Số lượng “Primary shard” cho 1 index không thể thay đổi sau khi index được tạo, chính vì vậy nên cần cân nhắc kĩ trước khi tạo index.
* Elasticsearch hoạt động dựa trên **Lucene của Apache**
* 1 shard thực chất là một **Lucene index**, là nơi lưu trữ dữ liệu và là **search engine** (mỗi một Shard là một search engine riêng)
* Vì là Lucene index => Shardtạo từ nhiều **Segment** và mỗi segment là **Inverted Index** với đầy đủ chức năng
* **Mỗi segment hoạt động độc lập => yêu cầu sử dụng CPU và bộ nhớ**
* Nhiều **Node** hợp lại tạo thành **Cluster**
* Có 3 loại **Node**: **Master, Data** và **Client**.
  + **Master:** Cluster sẽ tự động chọn ra một Node làm Master; có nhiệm vụ phân phối shard, tạo/xóa các index và cập nhật trạng thái cluster
  + **Data:** Được phân bố cho các Shard bởi Master; có nhiệm vụ index và tìm kiếm dữ liệu
  + **Client:** = Load Balancer
* Trong mỗi Node sẽ thường có 1 **Primary shard** riêng và nhiều **Replica shard** sẽ được tạo ra từ bản sao của **Primary shard** giúp tăng tốc độ đọc và đề phòng shard chính dừng hoạt động hoặc gặp trục trặc vấn đề gì đấy.

Những từ khóa cần lưu ý: Index, Type, Shard, Node, Primary shard và Replica shard, Lucence của Apache, Lucene index, Segment và Inverted Index, Node, Cluster

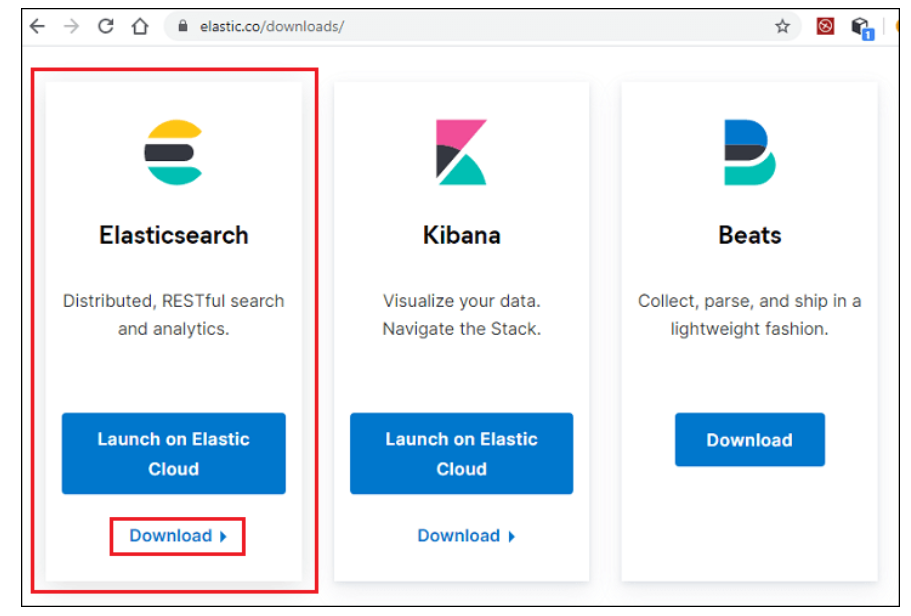
## Cài đặt elastic stack

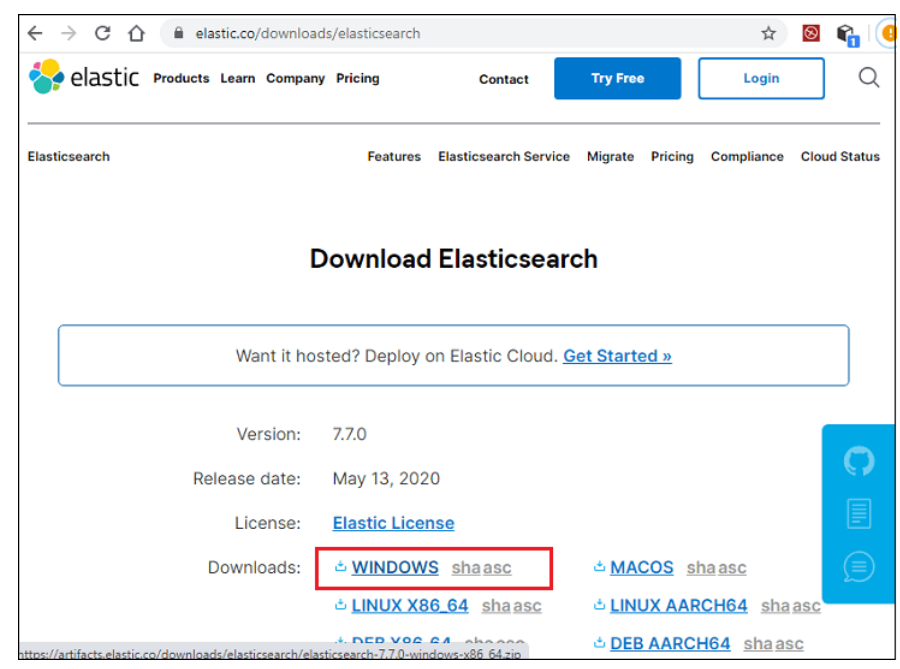
B1: Bạn cần cài java trước khi cài đặt Elastic search

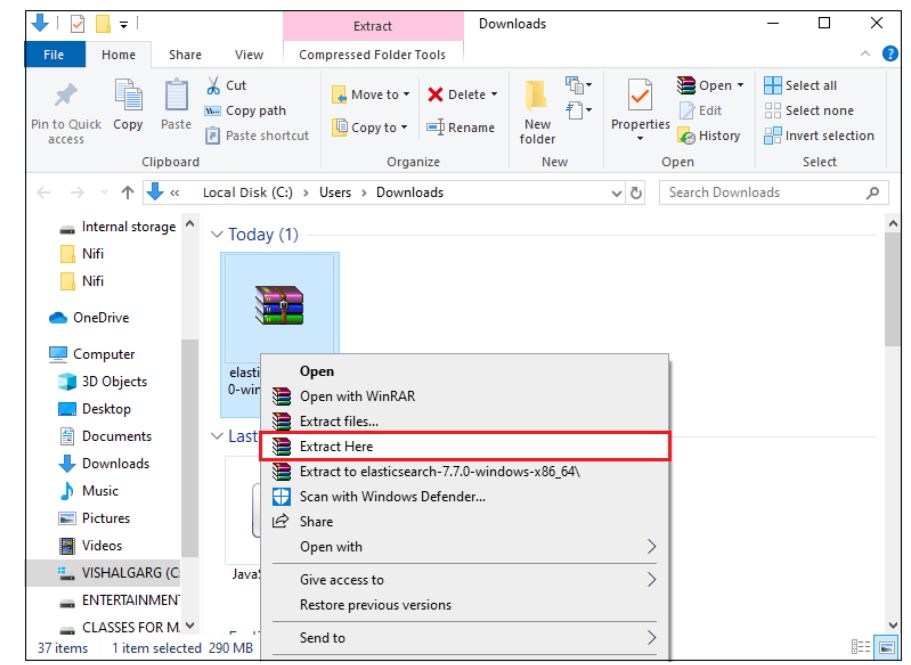
B2: Check java version của bạn phải lớn hơn hoặc bằng 11

B3: Vào trang chủ của elastic search để tải về:

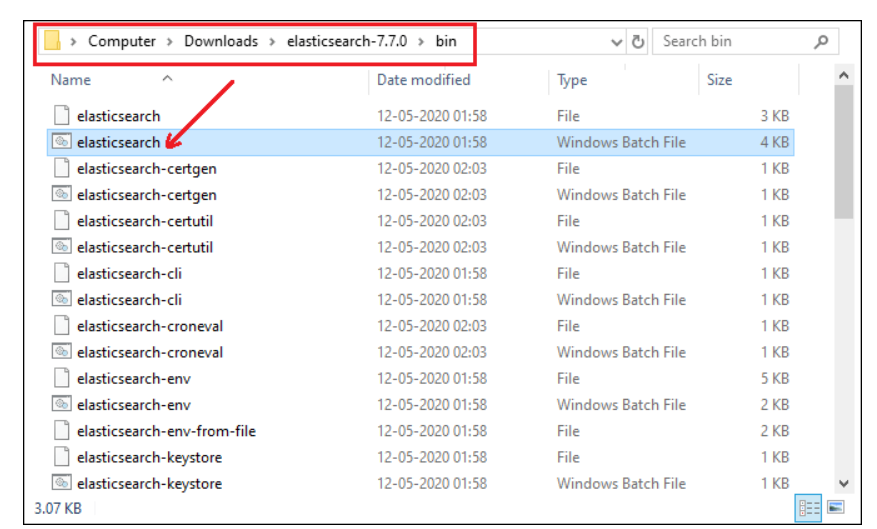
<https://www.elastic.co/downloads/>



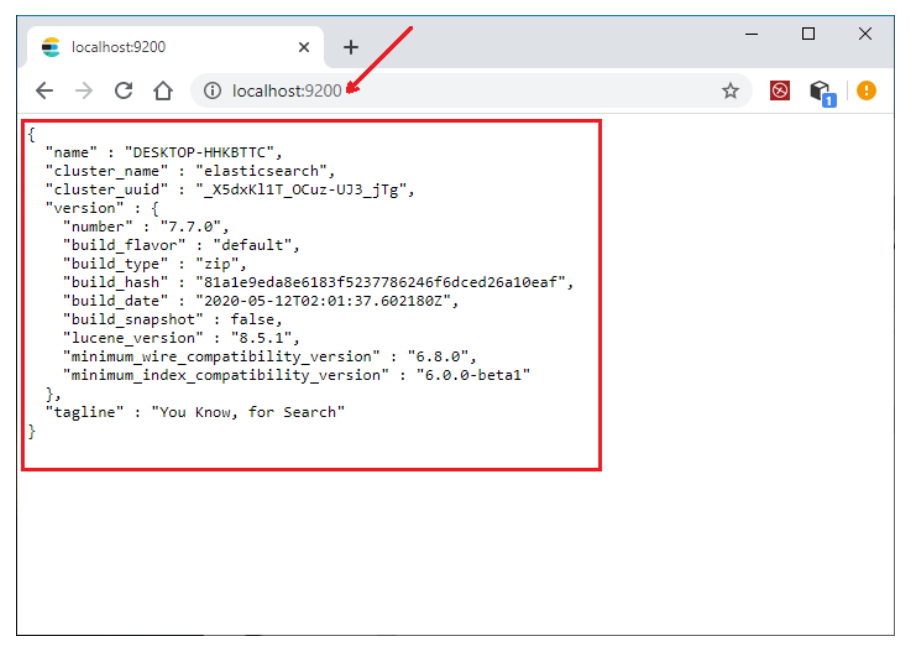




B5: Vào file bin của folder và run **elastic.bat** file



B6: Mở trình duyệt và gõ: <http://localhost:9200>



## Elasticsearch APIs

### Elasticsearch APIs

Elasticsearch APIs có 5 kiểu APIs:

+ Document APIs

+ Search APIs

+ Aggregation

+ Index APIs

+ Cluster APIs

### ES Document APIs

ES Document APIs lưu dữ liệu theo document form

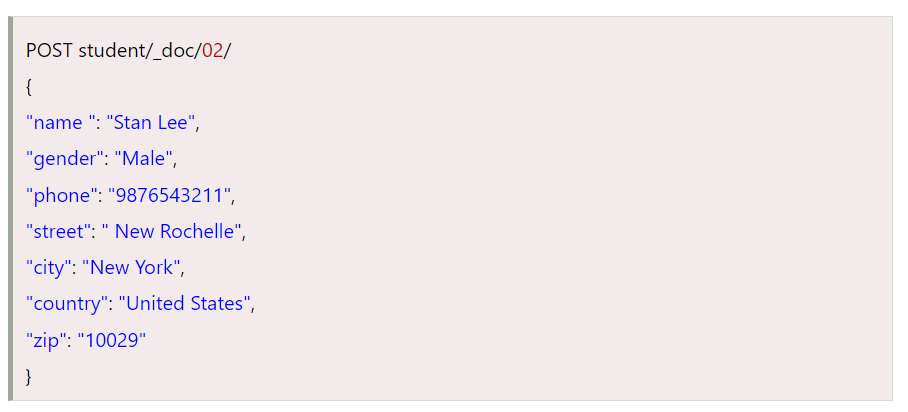
Có 2 loại ES Document APIs

* + Single Document API
  + Multi-Document API

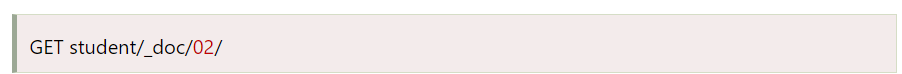
1. Single Documant API

Các API này rất hữu ích nếu bạn muốn thực hiện các hoạt động trên một tài liệu thay vì một nhóm tài liệu. Vì vậy bạn có thể sử dụng Single Document API này khi bạn muốn thực hiện các hoạt động trên một tài liệu . Được chia là 4 cái như sau:

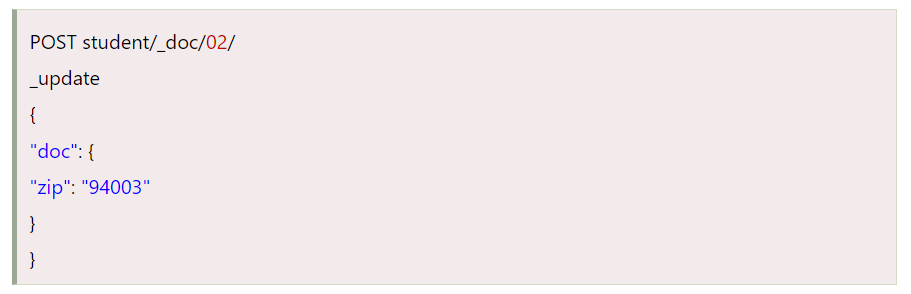
* + Index API
    - This API giúp add một document vào một index cụ thể khi request được tạo
    - Nó cho phép bạn update tài liệu nếu tài liệu đã tồn tại
    - Nó sử dụng thay thế toàn bộ tài liệu đã tồn tại
    - Sử dụng Post method để thêm dữ liệu và chúng ta phải nếu rõ id của document



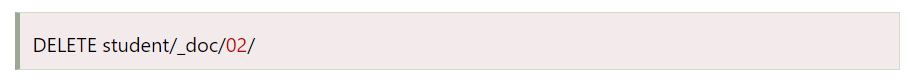
* + Get API
    - Get API giúp để nhận hoặc đọc dữ liệu được lưu trong Index
    - API này có nhiệm vụ nhận tài liệu JSON từ index



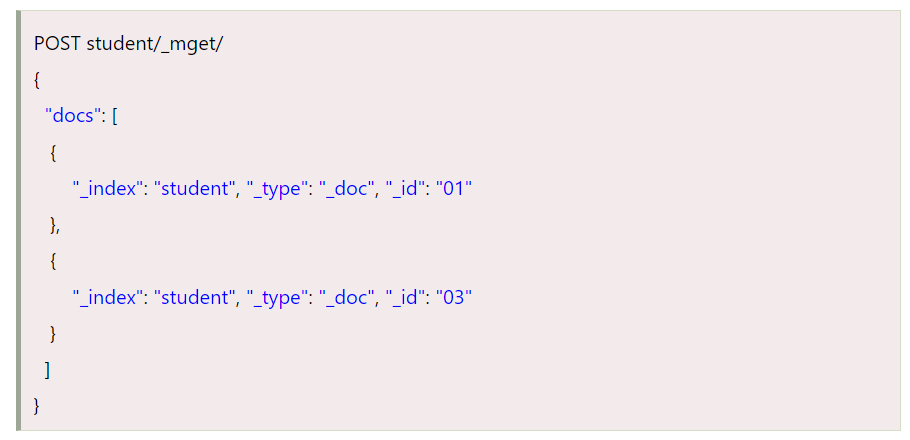
* + Update API



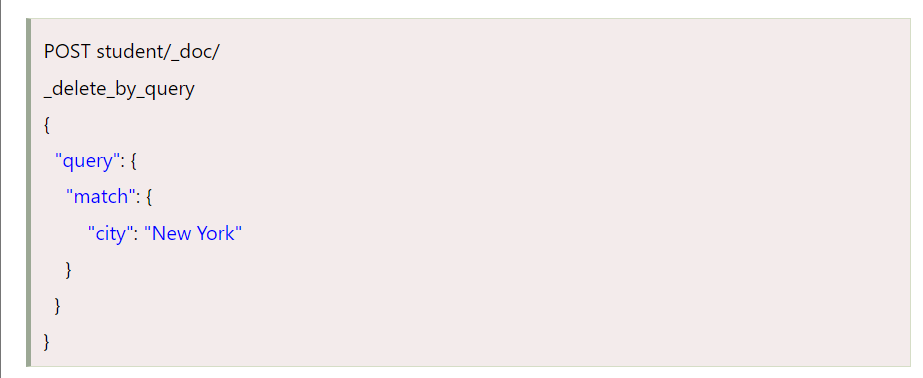
* + DeleeteAPI



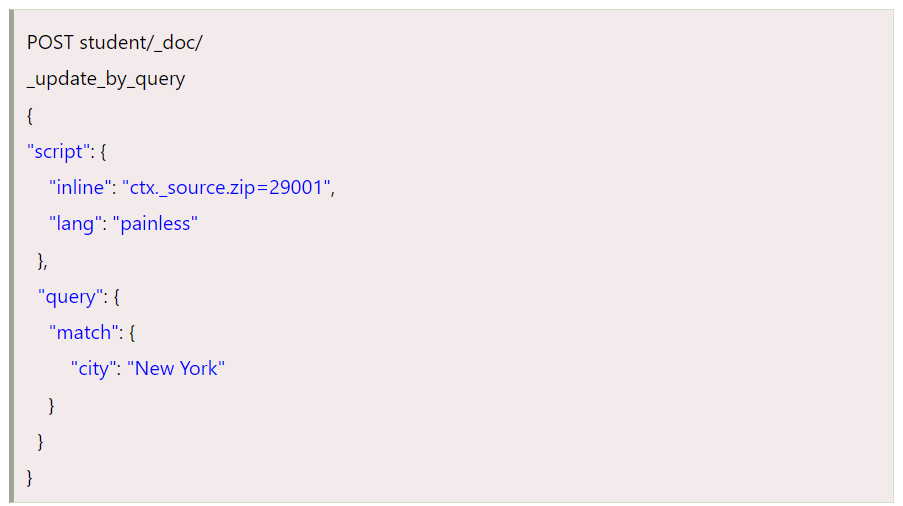
1. Multi-Document API
   * Không giống Single Document API bạn có thể sử dụng Multi-Document cho query giữa nhiều documents
   * Có 5 loại API:
     + Multi Get API



* + - Bulk API
      * Add \_bulk keyword to call API
      * Nó được sử dụng để delete/update/index trong bulk
    - Delete by query API
      * Dùng để delete tất cả các match document



* + - Update By Query API

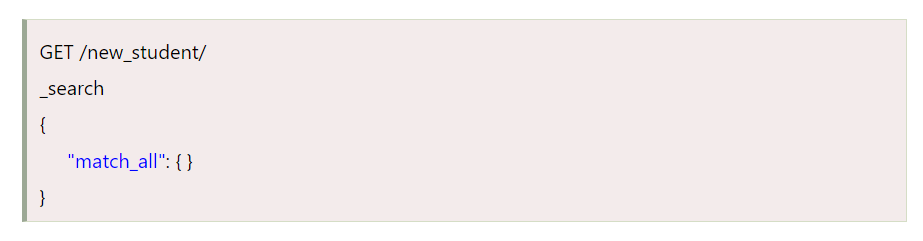


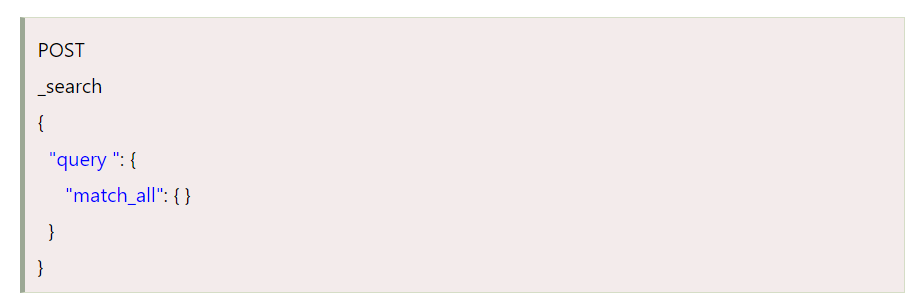
* + - Reindex API



### ES Search APIs

* + Được sử dụng để search dữ liệu trên index và tất cả các types





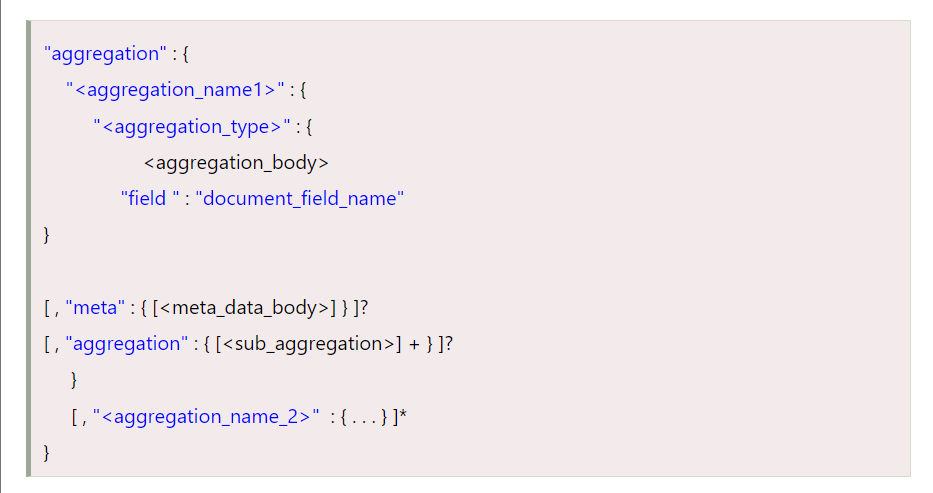
### ES Aggregation APIs

Elasticsearch cung cập API tổng hợp, được sử dụng để tổng hợp dữ liệu. Khung tổng hợp cung cấp dữ liệu tổng hợp dữ trên truy vấn tìm kiếm. Nói một cách đơn giản, Khung tổng hợp thu thập tất cả các dữ liệu được truy vấn tìm kiếm và cung cấp cho người dùng. Nó chứa một số khối xây dựng giúp xây dựng một bản tóm tắt dữ liệu phức tạp. Tập hợp tạo ra thông tin phân tích có sẵn trong Elastic search

Một số điểm quan trọng của Aggregation APIs:

* + - Aggregation APIs có thể được cấu hình thành cùng nhau để xây dựng các bản tóm tắt dữ liệu phức tạp.
    - Nó có thể được coi là một unit-of-work. Tạo ra thông tin phân tích trên một bộ tài liệu có sẵn trong Elasticsearch
    - Về cơ bản nó dựa trên các khối xây dựng

Syntax:

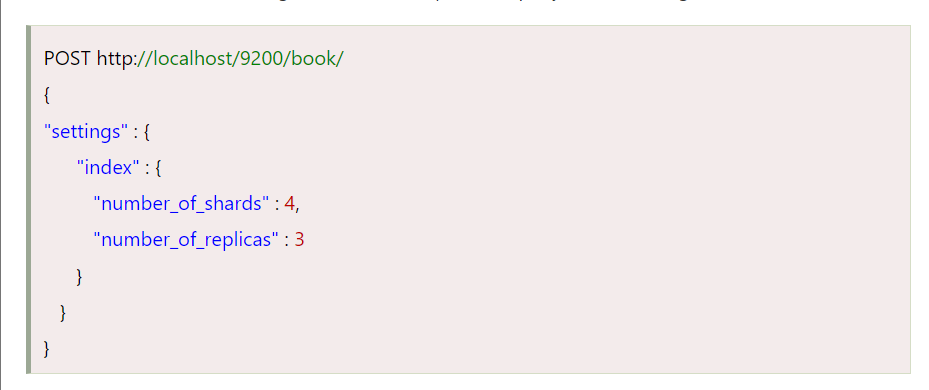


Chi tiết có thể lên trang: h<ttps://www.javatpoint.com/elasticsearch-a>ggregation-apis

### ES Index APIs

ES Index APIs để quản lí tất cả các khía cạnh của một index ví dụ như index template, mapping, aliases, settings…

Một số ví dụ:



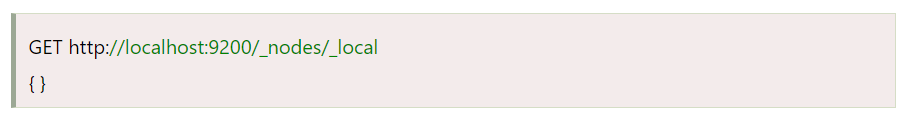


Chi tiết lên trang: https://www.javatpoint.com/elasticsearch-aggregation-apis

### ES Cluster APIs

Trong Elasticsearch, Cluster API lấy thông tin về một cụm và nút của nó. Nó cũng thực hiện những thay đổi hơn nữa trong chúng. Với sự trợ giúp của Cluster API , ta có thể thực hiện 21 hoạt động ở cập Cluster. Chúng ta có thể sử dụng Cluster API để quản lý các cluster của chúng tôi.

Chúng ta cần add thêm \_local keword vào trong GET method

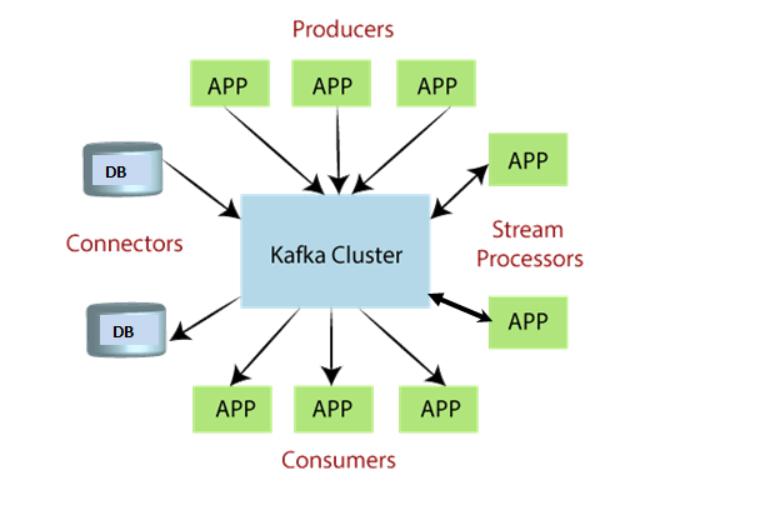


Chi tiết Tham khảo ở: https://www.javatpoint.com/elasticsearch-cluster-apis

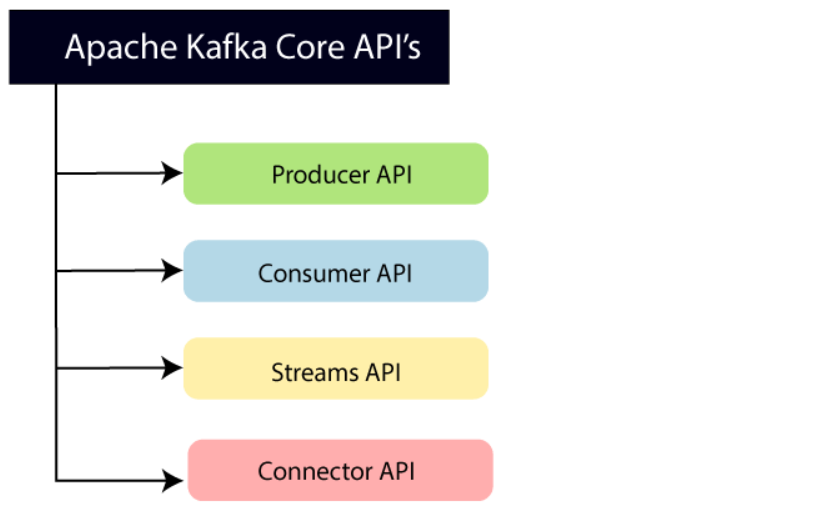
# Kafka

## Kafka là gì

* **Apache Kafka** là một nền tảng phần mềm xử lý luồng nguần mở được sử dụng để xử lý lưu trữ dữ liệu thời gian thực. Nó hoạt động như một nhà môi giới giữa hai bên, tức là, người gửi và người nhận. Nó có thể thử xý về hàng nghìn tỷ sự kiện dữ liệu trong một ngày.
* Apche Kafka ban đầu được phát triển bởi Linkedin và sau đó được ủng hộ lại cho Apache Software Foundation.
* **Hệ thống messaging** là gì: là một cuộc trao đổi đơn giản các tin nhắn giữa hai hoặc nhiều người, thiệt bị… Một hệ thống messaging được xuất bản subcribe cho phép người gửi gửi/ghi tin nhắn và người nhận nhận để đọc tin nhắn đó. In Apache Kafka, người gửi là Producer còn người nhận là Consumer.
* **Streaming process** là gì: là việc xử lý dữ liệu trong các hệ thống kết nối Parallelly. Quá trình này cho phép các ứng dụng khác nhau giới hạn việc thực thi dữ liệu song song, trong đó một bản ghi thực thi mà không chờ đợi đầu ra của bản ghi trước đó. Do đó, một nền tảng phát trực tuyến phân tán cho phép người dùng đơn giản hóa nhiệm vụ của quá trình phát trực tuyến và thực hiện song song. Do đó, một nền tảng phát trực tuyến trong Kafka có các khả năng chính sau:
  + Ngay khi các luồng record xảy ra, nó xử lý nó.
  + Nó hoạt động tương tự như một hệ thống nhắn tin doanh nghiệp nơi nó xuất bản và đăng ký các luồng của record.
  + Nó lưu trữ các luồng record một cách bền bỉ có lỗi.



* Bốn core concept của Apache Kafka:
  + **Producer API**: API này cho phép ứng dụng đẩy các bản ghi lên một hoặc nhiều topics
  + **Consumer API**: API này cho phép ứng dụng đăng ký lên một hoặc nhiều topics và xử lý nhận các record từ các topics đó
  + **Streams API**: API này cho phép một ứng dụng chuyển đổi hiệu quả các luồng đầu vào thành các luồng đầu ra. Nó cho phép một ứng dụng hoạt động như một bộ xử lý luồng tiêu thụ một luồng đầu vào từ một hoặc nhiều chủ đề và tạo một luồng đầu ra cho một hoặc nhiều chủ đề đầu ra.
  + **Connector API**: API này thực thi producer và consumer API có thể tái sử dụng với các hệ thống hoặc ứng dụng dữ liệu hiện có.

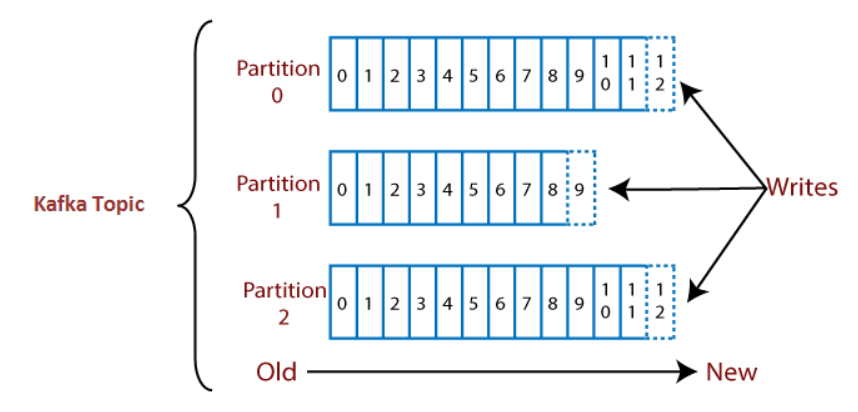


## Kafka core concept

### Kafka topics

* Topics của Kafka được dùng để lưu trữ và xuất bản một luồng dữ liệu cụ thể nào đó
* Chúng ta có thể tưởng tưởng Topics của Kafka như tables trong database nhưng nó không chứa các rằng buộc.
* Bạn có thể tạo nhiều topic được xác định theo tên. Và Producer sẽ đẩy dữ liệu lên trên Topic và Consumer sẽ đọc dữ liệu từ Topic mà nó đăng ký

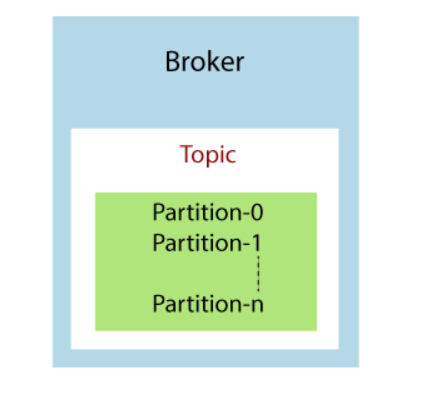
1. **Partitions**
   * Topic có thể được chia thành nhiều phần các phần đó được biết đến là partition của topic
   * Các partition này được phân tách theo thứ tự
   * Nội dung của dữ liệu được lưu trữ trong Partition của topic
   * Khi tạo một topic chúng ta cần cho biết là sẽ có bao nhiều partition được tạo ra.
   * Mỗi thông báo được lưu trữ vào các phân vùng với ID gia tăng được gọi là offset value của nó

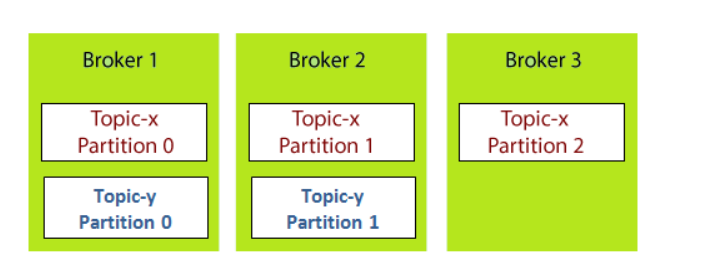


NOTE: Dữ liệu một khi được ghi vào một Partition không bao giờ có thể được thay đổi. Nó là bất biến. Giá trị offset luôn ở trạng thái gia tăng, nó không bao giờ quay trở lại một không gian trống. Ngoài ra, dữ liệu được giữ trong một partition chỉ trong một thời gian giới hạn.

1. **Brokers**

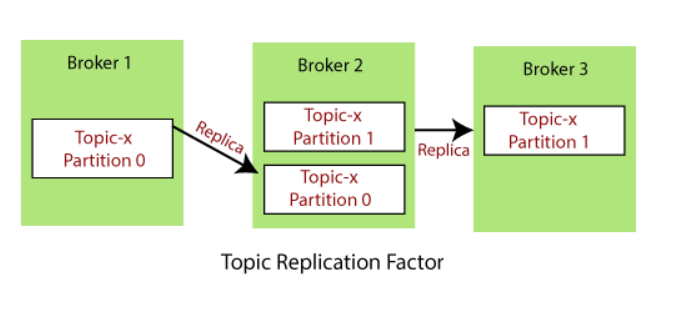
* Một Kafka cluster được kết hợp từ một hoặc nhiều server cái mà được biết đến như một **brokers** hoặc **Kafka broker**.
* Một broker chứa một vài topic
* Một broker trong cluster được xác định bưởi số nguyên ID.
* Kafka brokers cũng được biết đến như **Bootstrap brokers** bởi vì khi kết nối với một broker nghĩa là kết nối với toàn bộ cluster. Mặc dù một broker không chứa toàn bộ dữ liệu nhưng mỗi broker trong cluster biết đến tất cả các brokers, partitions, topics khác

****

****

**Note:** Một partition có thể thuộc ở nhiều broker

### Topic replication



### Kafka producer

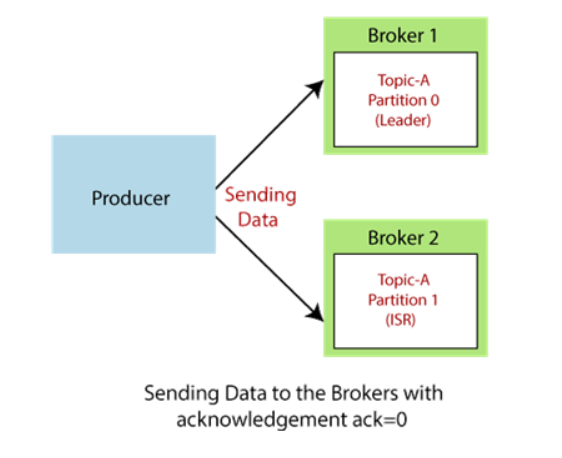
* Được dùng để xuất bản và ghi dữ liệu vào topic trong các partition khác nhau. Producer tự động biết dữ liệu nên ở partition và broker nào nếu người dùng không chỉ định partition hoặc broker.
* Kafka producer hoạt động và ghi data theo kiểu:
  + Message keys
  + Acknowledgment

1. **Message keys**

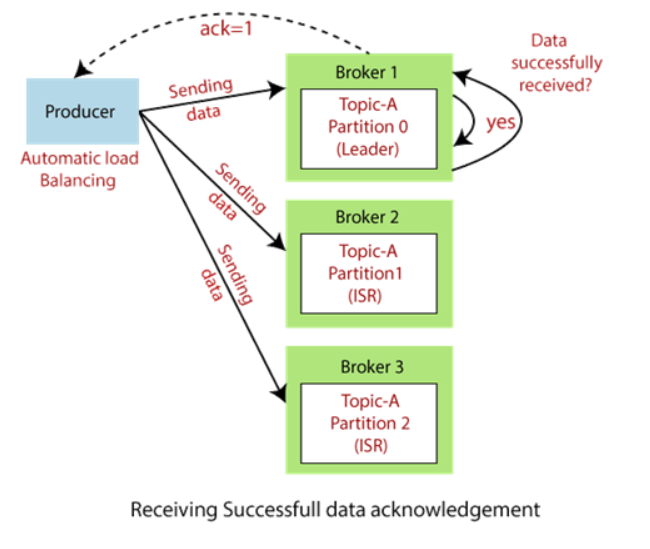
* Kafka Producer có thể gửi dữ liệu có key hoặc không có key
  + Key=null: Không có key, Nó sẽ được phân phối trong round-robin manner (distributed cho mỗi parition)
  + Key!=null: Có key, Tất cả messages sẽ luôn được phân phối đến cùng partition

1. **Acknowledgment**

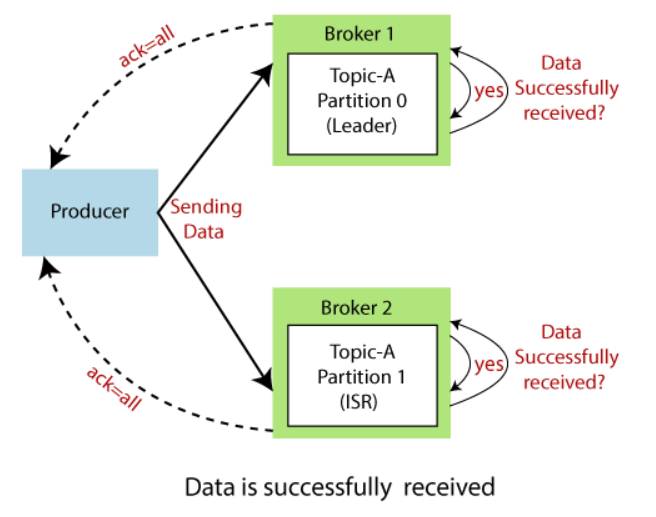
* Để viết dữ liệu vào Kafka Cluster, Producer có một lựa chọn khác đó là acknowledgment. Điều đó có nghia là Producer có thể nhận được xác nhận dữ liệu của mình ghi bằng cách chọn các acknowledgment:
  + **Acks = 0:** Điều này có nghĩ là khi Producer gửi dữ liệu lên broker sẽ không đợi acknowledgment. Điều này có thể dẫn đến mất mát dữ liệu vì nó k xác nhận rằng dữ liệu có được gửi lên broker thành công hay không

****

* + **Acks = 1:** Producer có đợi xác nhận Acks từ broker cho biết dữ liệu đã được gửi lên broker thành không hay không. Giới hạn xảy ra mất mát dữ liệu

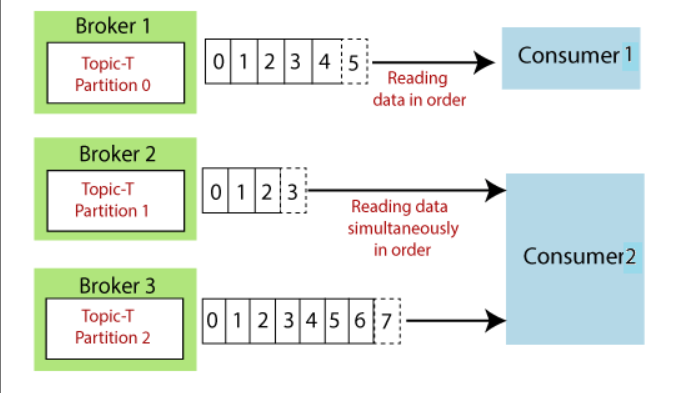
****

* + **Acks = all:** Acknowledgment thành công bới leader và người theo dõi. Dữ liệu được nhận thành công. Không xảy ra mất mát dữ liệu

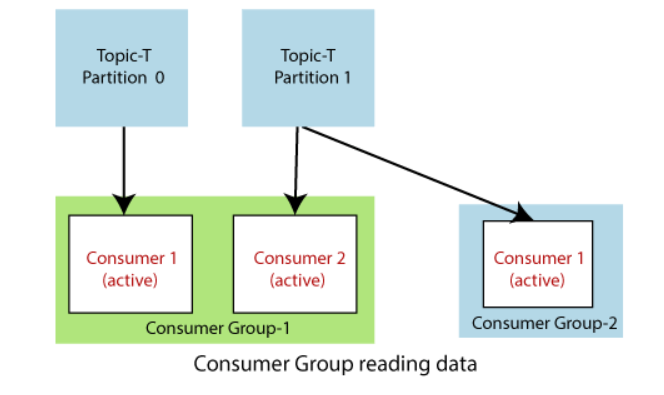
****

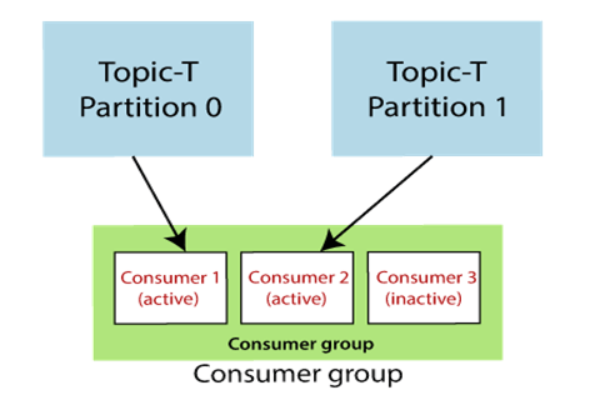
### Consumer & Comsumer Groups

* Consumer sẽ tiêu thị và đọc dữ liệu từ cluster thông qua topic
* Consumer đọc dữ liệu trong mỗi partition một cách có trật tự



* Một consumer group làm một nhóm nhiều consumer. Mỗi mội consumer đại diện để đọc một nhóm dữ liệu trên partition nào đó.





### Advantage & Disadvantage

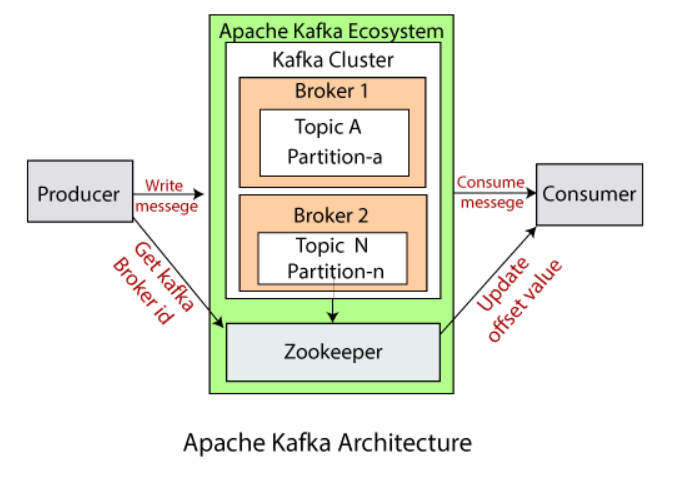
1. Advantage

* **Low latency**: Apache Kafka cung cấp giá trị độ trễ thấp, tức là, tối đa 10 mili giây. Đó là bởi vì nó phân tách thông điệp cho phép người tiêu dùng tiêu thụ tin nhắn đó bất cứ lúc nào.
* **High throughput**: Do độ trễ thấp, Kafka có thể xử lý nhiều thông điệp có khối lượng lớn và vận tốc cao. Kafka có thể hỗ trợ hàng ngàn tin nhắn trong một giây. Nhiều công ty như Uber sử dụng Kafka để tải khối lượng dữ liệu lớn.
* **Fault tolerance**: Kafka có một tính năng thiết yếu để cung cấp khả năng chống lại lỗi nút/máy trong cụm.
* **Durability**: Kafka cung cấp tính năng sao chép, tạo ra dữ liệu hoặc tin nhắn để tồn tại nhiều hơn trên cụm qua đĩa. Điều này làm cho nó bền.
* **Reduces the need for multiple integrations**: Tất cả các dữ liệu mà một nhà sản xuất viết đều đi qua Kafka. Do đó, chúng ta chỉ cần tạo một tích hợp với Kafka, tự động tích hợp chúng ta với mỗi hệ thống sản xuất và tiêu thụ.
* **Easily accessible**: Khi tất cả dữ liệu của chúng tôi được lưu trữ trong Kafka, nó sẽ dễ dàng truy cập cho bất kỳ ai.
* **Distributed system**: Apache Kafka chứa một kiến ​​trúc phân tán làm cho nó có thể mở rộng. Phân vùng và sao chép là hai khả năng trong hệ thống phân tán.
* **Real** **time handling:** Apache Kafka có thể xử lý đường ống dữ liệu thời gian thực. Xây dựng một đường ống dữ liệu thời gian thực bao gồm bộ xử lý, phân tích, lưu trữ, v.v.
* **Batch approach:** Kafka sử dụng các trường hợp sử dụng giống như hàng loạt. Nó cũng có thể hoạt động như một công cụ ETL vì khả năng tồn tại dữ liệu của nó.
* **Scalability:** Chất lượng của Kafka để xử lý một lượng lớn tin nhắn đồng thời làm cho nó trở thành một sản phẩm phần mềm có thể mở rộng.

1. Disadvantage

* Không có bộ công cụ giám sát hoàn chỉnh
* Không hỗ trợ lựa chọn wildcard topic
* Giảm performance
* Thiếu một số mô hình tin nhắn

### Kafka Architecture

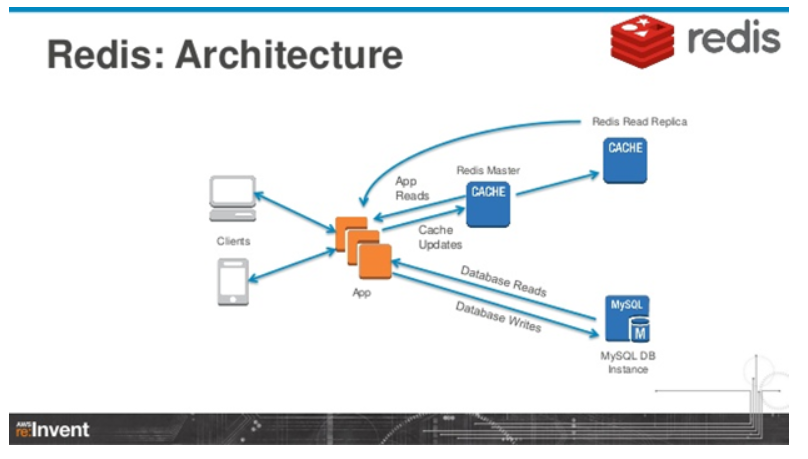


# Redis Cache

## Giới thiệu về Redis

* Redis là một cơ sở dữ liệu No SQL, nó hoạt động, lưu trữ dữ liệu theo cơ chế key – value pair.
* Redis là một mã nguần mở, linh hoạt (được cấp phép BSD), lưu trữ cấu trúc dữ liệu trong bộ nhớ, được sử dụng làm cơ sở dữ liệu, bộ đệm và nhà môi giới tin nhắn. Redis là cơ sở dữ liệu NoSQL, vì vậy nó tạo điều kiện cho người dùng lưu trữ lượng dữ liệu lớn khổng lồ mà không có giới hạn của cơ sở dữ liệu quan hệ.
* Redis hỗ trợ đa dạng các cấu trúc dữ liệu khác nhau như Strings, Hashes, Lists, Sets, Sorted Sets, Bitmaps….
* Redis Architecture:
  + Redis Client
  + Redis Server

Note: Redis Client và Redis Server có thể ở cùng trên một máy tính hoặc là trên hai máy tính khác nhau



Redis server được sử dụng để lưu trữ dữ liệu trong bộ nhớ. Nó kiểm soát tất cả các loại quản lý và tạo thành phần chính của kiến trúc. Bạn có thể tạo Redis Client hoặc Redis Console khi bạn cài đặt ứng dụng với Redis Speed: Redis lưu trữ toàn bộ dữ liệu trong bộ nhớ chính nên nó hoạt động rất nhanh. Redis còn hỗ trọ pipeline và khả năng bạn có thể sữ dụng nhiều giá trị trên một command để tăng tốc độ giao tiếp với Client

Một số chức năng chính của Redis

* + Persistence: Mặc dù tất cả các dữ liệu được lưu trong bộ nhớ, các thay đổi được lưu không đồng bộ trên đĩa bằng chính sách linh hoạt dựa trên thời gian trôi qua hoặc số lượng truy cấp kể từ lần lưu cuối.
  + Data structures: Redis hỗ trợ đa dạng nhiều kiểu dữ liệu khác nhau.
  + Supported Languages: Redis cũng hỗ trợ trên nhiều ngôn ngữ khác nhau như: Java, C, C++, C#....
  + Sharding: Redis còn hỗ trợ Sharding. Từ đó nó giúp dễ dàng để phân chia dữ liệu trên nhiều Redis Intances.

## Redis Commands

1. Redis keys

Syntax:



Các loại Redis keys

|  |  |  |
| --- | --- | --- |
| **Index** | **Command** | **Description** |
| 1 | DEL key | This command is used to delete the key, if it exists. |
| 2 | DUMP key | This command is used to return a serialized version of the value stored at the specified key. |
| 3 | EXISTS key | This command is used to check whether the key exists or not. |
| 4 | EXPIRE key | This command is used to set the expiry of the key after the specified time in seconds. |
| 5 | EXPIREAT key | This command is used to set the expiry of the key after the specified time. Here time is in UNIX timestamp format. |
| 6 | PEXPIRE key | This command is used to set the expiry of key in milliseconds. |
| 7 | PEXPIREAT key | This command is used to set the expiry of the key in UNIX timestamp specified as milliseconds. |
| 8 | KEYS pattern | This command is used to find all keys matching the specified pattern. |
| 9 | MOVE key | It is used to move a key to another database. |
| 10 | PERSIST key | It is used to remove the expiration from the key. |
| 11 | PTTL key | It is used to retrieve the remaining time in keys expiry in milliseconds. |
| 12 | TTL key | It is used to retrieve the remaining time in keys expiry. |
| 13 | randomkey | It is used to get a random key from Redis. |
| 14 | RENAME key newkey | It is used to change the key name. |
| 15 | RENAMENX key newkey | It is used to rename the key, if a new key doesn't exist. |
| 16 | TYPE key | It is used to fetch the data type of the value stored in the key. |

1. Redis strings

* Dùng để quản lý giá trị là string
* Các commands line thông dụng

|  |  |  |
| --- | --- | --- |
| **Index** | **Command** | **Description** |
| 1 | SET key value | This command is used to set the value at the specified key. |
| 2 | GET key | This command is used to retrieve the value of a key. |
| 3 | GETRANGE key start end | This command is used to get a substring of the string stored at a key. |
| 4 | GETSET key value | This command is used to set the string value of a key and return its old value. |
| 5 | GETBIT key offset | This command is used to return the bit value at the offset in the string value stored at the key. |
| 6 | MGET key1 [key2..] | This command is used to get the values of all the given keys |
| 7 | SETBIT key offset value | This command is used to set or clear the bit at the offset in the string value stored at the key |
| 8 | SETEX key seconds value | This command is used to set the value with the expiry of a key |
| 9 | SETNX key value | This command is used to set the value of a key, only if the key does not exist |
| 10 | SETRANGE key offset value | This command is used to overwrite the part of a string at the key starting at the specified offset |
| 11 | STRLEN key | This command is used to retrieve the length of the value stored in a key |
| 12 | MSET key value [key value ...] | This command is used to set multiple keys to multiple values |
| 13 | MSETNX key value [key value ...] | This command is used to set multiple keys to multiple values, only if none of the keys exist |
| 14 | PSETEX key milliseconds value | This command is used to set the value and expiration in milliseconds of a key |
| 15 | INCR key | This command is used to increment the integer value of a key by one |
| 16 | INCRBY key increment | This command is used to increment the integer value of a key by the given amount |
| 17 | INCRBYFLOAT key increment | This command is used to increment the float value of a key by the given amount |
| 18 | DECR key | This command is used to decrement the integer value of a key by one |
| 19 | DECRBY key decrement | This command is used to decrement the integer value of a key by the given number |
| 20 | APPEND key value | This command is used to append a value to a key |

1. Redis hashes

* Redis hashes là kiểu dữ liệu hoàn hảo để biểu diễn Objects. Chúng sử dụng map giữa string fields và string values.
* Redis hash Commands

|  |  |  |
| --- | --- | --- |
| **Index** | **Command** | **Description** |
| 1 | HDEL key field2 [field2] | It is used to delete one or more hash fields. |
| 2 | HEXISTS key field | It is used to determine whether a hash field exists or not. |
| 3 | HGET key field | It is used to get the value of a hash field stored at the specified key. |
| 4 | HGETALL key | It is used to get all the fields and values stored in a hash at the specified key |
| 5 | HINCRBY key field increment | It is used to increment the integer value of a hash field by the given number |
| 6 | HINCRBYFLOAT key field increment | It is used to increment the float value of a hash field by the given amount |
| 7 | HKEYS key | It is used to get all the fields in a hash |
| 8 | HLEN key | It is used to get the number of fields in a hash |
| 9 | HMGET key field1 [field2] | It is used to get the values of all the given hash fields |
| 10 | HMSET key field1 value1 [field2 value2 ] | It is used to set multiple hash fields to multiple values |
| 11 | HSET key field value | It is used to set the string value of a hash field |
| 12 | HSETNX key field value | It is used to set the value of a hash field, only if the field does not exist |
| 13 | HVALS key | It is used to get all the values in a hash |

1. Redis lists

* Redis lists có để dùng để lưu danh sách strings được sắp xếp theo thứ tự thêm vào. Bạn có thể add thêm phần tử vào đầu hoặc cuối của Redis
* Redis lists commands:

|  |  |  |
| --- | --- | --- |
| **Index** | **Command** | **Description** |
| 1 | BLPOP key1 [key2 ] timeout | It is used to remove and get the first element in a list, or blocks until one is available. |
| 2 | BRPOP key1 [key2 ] timeout | It is used to remove and get the last element in a list, or blocks until one is available. |
| 3 | BRPOPLPUSH source destination timeout | It is used to pop a value from a list, pushes it to another list and returns it; or blocks until one is available. |
| 4 | LINDEX key index | It is used to get an element from a list by its index. |
| 5 | LINSERT key before|after pivot value | It is used to insert an element before or after another element in a list. |
| 6 | LLEN key | It is used to get the length of a list. |
| 7 | LPOP key | It is used to remove and get the first element in a list. |
| 8 | LPUSH key value1 [value2] | It is used to prepend one or multiple values to a list. |
| 9 | LPUSHX key value | It is used to prepend a value to a list, only if the list exists. |
| 10 | LRANGE key start stop | It is used to get a range of elements from a list. |
| 11 | LREM key count value | It is used to remove elements from a list. |
| 12 | LSET key index value | It is used to set the value of an element in a list by its index. |
| 13 | LTRIM key start stop | It is used to trim a list to the specified range. |
| 14 | RPOP key | It is used to remove and get the last element in a list. |
| 15 | RPOPLPUSH source destination | It is used to remove the last element in a list, append it to another list and returns it. |
| 16 | RPUSH key value1 [value2] | It is used to append one or multiple values to a list. |
| 17 | RPUSHX key value | It is used to append a value to a list, only if the list exists. |

1. Redis sets

* Redis sets là một tập các giá trị đơn nhất strings được lưu trữ không theo một thứ tự nhất định nào. Đơn nhất nghía là các giá trị trong redis sets không có 2 phần tử nào là giống nhau
* Redis sets commands:

|  |  |  |
| --- | --- | --- |
| **Index** | **Command** | **Description** |
| 1 | SADD key member1 [member2] | It is used to add one or more members to a set. |
| 2 | SCARD key | It is used to getsthe number of members in a set. |
| 3 | SDIFF key1 [key2] | It is used to subtract multiple sets. |
| 4 | SDIFFstore destination key1 [key2] | It is used to subtract multiple sets and stores the resulting set in a key. |
| 5 | SINTER key1 [key2] | It is used to intersect multiple sets. |
| 6 | SINTERSTORE destination key1 [key2] | It is used to intersect multiple sets and stores the resulting set in a key. |
| 7 | SISMEMBER key member | It is used to determine if a given value is a member of a set. |
| 8 | SMOVE source destination member | It is used to move a member from one set to another. |
| 9 | SPOP key | It is used to remove and returns a random member from a set. |
| 10 | SRANDMEMBER key [count] | It is used to get one or multiple random members from a set. |
| 11 | SREM key member1 [member2] | It is used to remove one or more members from a set. |
| 12 | SUNION key1 [key2] | It is used to add multiple sets. |
| 13 | SUNIONSTORE destination key1 [key2] | It is used to add multiple sets and stores the resulting set in a key. |
| 14 | SSCAN key cursor [match pattern] [count count] | It is used to incrementally iterates set elements. |

1. Redis sorted sets

* Redis sorted sets cũng giống như redis sets nhưng chỉ khác là các giá trị trong redis sorted sets được sắp xếp theo một thứ tự nhất định nào đó
* Redis sorted sets commands:

|  |  |  |
| --- | --- | --- |
| **Index** | **Commands** | **Description** |
| 1 | ZADD key score1 member1 [score2 member2] | It is used to add one or more members to a sorted set, or updates its score, if it already exists |
| 2 | ZCARD key | It is used to get the number of members in a sorted set |
| 3 | ZCOUNT key min max | It is used to count the members in a sorted set with scores within the given values |
| 4 | ZINCRBY key increment member | It is used to increment the score of a member in a sorted set |
| 5 | ZINTERSTORE destination numkeys key [key ...] | It is used to intersect multiple sorted sets and stores the resulting sorted set in a new key |
| 6 | ZLEXCOUNT key min max | It is used to count the number of members in a sorted set between a given lexicographical range |
| 7 | ZRANGE key start stop [WITHSCORES] | It is used to return a range of members in a sorted set, by index |
| 8 | ZRANGEBYLEX key min max [LIMIT offset count] | It is used to return a range of members in a sorted set, by lexicographical range |
| 9 | ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT] | It is used to return a range of members in a sorted set, by score |
| 10 | ZRANK key member | It is used to determine the index of a member in a sorted set |
| 11 | ZREM key member [member ...] | It is used to remove one or more members from a sorted set |
| 12 | ZREMRANGEBYLEX key min max | It is used to remove all members in a sorted set between the given lexicographical range |
| 13 | ZREMRANGEBYRANK key start stop | It is used to remove all members in a sorted set within the given indexes |
| 14 | ZREMRANGEBYSCORE key min max | It is used to remove all members in a sorted set within the given scores |
| 15 | ZREVRANGE key start stop [WITHSCORES] | It is used to return a range of members in a sorted set, by index, with scores ordered from high to low |
| 16 | ZREVRANGEBYSCORE key max min [WITHSCORES] | It is used to return a range of members in a sorted set, by score, with scores ordered from high to low |
| 17 | ZREVRANK key member | It is used to determine the index of a member in a sorted set, with scores ordered from high to low |
| 18 | ZSCORE key member | It is used to retrieve the score associated with the given member in a sorted set |
| 19 | ZUNIONSTORE destination numkeys key [key ...] | It is used to add multiple sorted sets and stores the resulting sorted set in a new key |
| 20 | ZSCAN key cursor [MATCH pattern] [COUNT count] | It is used to incrementally iterates sorted sets elements and associated scores |

1. Redis transaction

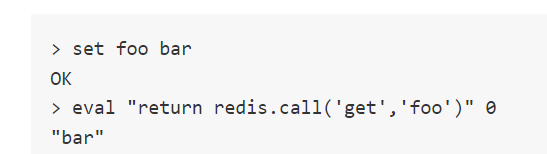
* Redis transaction được sử dụng để tạo điều kiện cho người dùng thực thi nhóm lệnh trong một bước duy nhất.
* Có hai thuộc tính thực thi như:
  + Tất cả các lệnh trong một transaction được thực thi tuần tự dưới dạng một hoạt động cô lập. Bạn không thể đưa ra yêu cầu một client khác được phục vụ giữ việc thực hiện redis transaction
  + Giao dịch Redis cũng là nguyên tử. Nguyên tử có nghĩa là tất cả các lệnh hoặc không có lệnh nào được xử lý.
* Redis transaction commands

|  |  |  |
| --- | --- | --- |
| **Index** | **Command** | **Description** |
| 1 | DISCARD | It is used to discard all commands issued after MULTI |
| 2 | EXEC | It is used to execute all commands issued after MULTI |
| 3 | MULTI | It is used to mark the start of a transaction block |
| 4 | UNWATCH | It is used to forget about all watched keys |
| 5 | WATCH key [key ...] | It is used to watche the given keys to determine the execution of the MULTI/EXEC block |

1. Redis scription

* Redis Scripting được sử dụng để đánh giá các tập lệnh bằng trình thông dịch LUA. Nó được tích hợp vào redis bắt đầu từ phiên bản 2.6.0. Lệnh được sử dụng để viết kịch bản là lệnh eval.
* Eg:





1. Redis connection

* Redis connection được sử dụng để điểu kiển và quản lý các kết nối từ client vơi Redis Server
* Redis connection commands:

|  |  |  |
| --- | --- | --- |
| **Index** | **Commands** | **Description** |
| 1 | AUTH password | It is used to authenticate to the server with the given password. |
| 2 | ECHO message | It is used to print the given string. |
| 3 | PING | It is used to check whether the server is running or not. |
| 4 | QUIT | It is used to close the current connection. |
| 5 | SELECT index | It is used to change the selected database for the current connection |

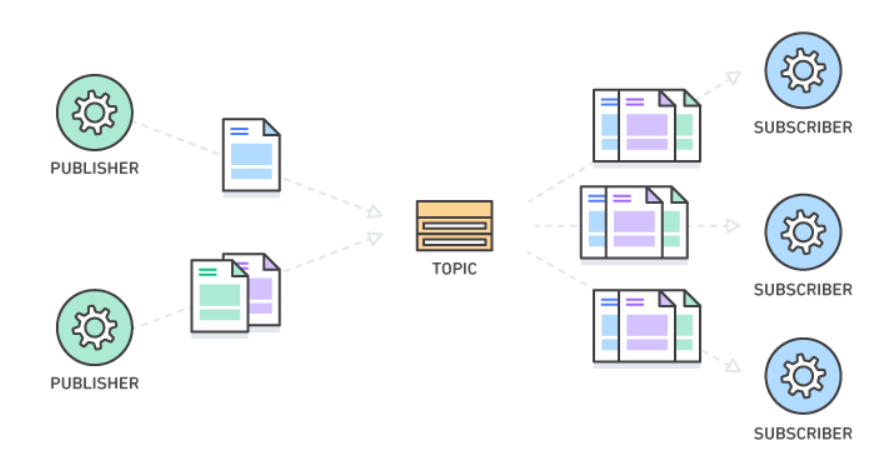
1. Redis Server

* Dùng để quản lý server
* Có nhiều lệnh dùng để lấy thông tin của server, thống kê …
* Redis server commands:

|  |  |  |
| --- | --- | --- |
| **Index** | **Command** | **Description** |
| 1 | BGREWRITEAOF | This command is used to asynchronously rewrite the append-only file. |
| 2 | BGSAVE | This command is used to asynchronously save the dataset to the disk. |
| 3 | CLIENT KILL [ip:port] [ID client-id] | This command is used to kill the connection of a client. |
| 4 | CLIENT LIST | This command is used to get the list of client connections to the server. |
| 5 | CLIENT GETNAME | This command is used to get the name of the current connection. |
| 6 | CLIENT PAUSE timeout | This command is used to stop processing commands from the clients for a specified time. |
| 7 | CLIENT SETNAME connection-name | This command is used to set the current connection name. |
| 8 | CLUSTER SLOTS | This command is used to get an array of Cluster slot to node mappings. |
| 9 | COMMAND | This command is used to get an array of Redis command details. |
| 10 | COMMAND COUNT | This command is used to get total number of Redis commands. |
| 11 | COMMAND GETKEYS | This command is used to extract the keys given a full Redis command. |
| 12 | BGSAVE | This command is used to asynchronously save the dataset to the disk. |
| 13 | COMMAND INFO command-name [command-name ...] | This command is used to get an array of specific Redis command details. |
| 14 | CONFIG GET parameter | This command is used to get the value of a configuration parameter. |
| 15 | CONFIG REWRITE | This command is used to rewrite the configuration file with the in-memory configuration. |
| 16 | CONFIG SET parameter value | This command is used to get a configuration parameter to the given value. |
| 17 | CONFIG RESETSTAT | This command is used to reset the stats returned by INFO. |
| 18 | DBSIZE | This command is used to return the number of keys in the selected database. |
| 19 | DEBUG OBJECT key | This command is used to get debugging information about a key. |
| 20 | DEBUG SEGFAULT | This command is used to make the server crash. |
| 21 | FLUSHALL | This command is used to remove all the keys from all databases. |
| 22 | FLUSHDB | This command is used to remove all the keys from the current database. |
| 23 | INFO [section] | This command is used to get information and statistics about the server. |
| 24 | LASTSAVE | This command is used to retrieve the UNIX time stamp of the last successful save to the disk. |
| 25 | MONITOR | This command is used to listen for all the requests received by the server in real time. |
| 26 | ROLE | This command is used to return the role of the instance in the context of replication. |
| 27 | SAVE | This command is used to synchronously saves the dataset to the disk. |
| 28 | SHUTDOWN [NOSAVE] [SAVE] | This command is used to synchronously saves the dataset to the disk and then shuts down the server . |
| 29 | SLAVEOF host port | This command is used to make the server a slave of another instance, or promotes it as a master . |
| 30 | SLOWLOG subcommand [argument] | This command is used to manage the Redis slow queries log. |
| 31 | SYNC | This command is used for replication. |
| 32 | TIME | This command is used to return the current server time. |

## Publisher và Subcriber với Redis

* Redis Pub/Sub là một kiểu messaging pattern lâu đời nhất được Redis hỗ trợ và sử dụng loại dữ liệu gọi là "channel". Redis Pub/Sub có cơ chế giống với pub/sub messaging, là 1 trong 2 [message broker](https://www.ibm.com/cloud/learn/message-brokers) phổ biến.
* Publish/subscribe messaging, hay pub/sub messaging, là một hình thức giao tiếp service-to-service bất đồng bộ được sử dụng trong các kiến trúc phân tán. Pub/sub messaging có cơ chế hoạt động như sau:
  + Pub - Publish: Publisher sẽ đẩy message(data) vào một hoặc nhiều topic(channel).
  + Sub - Subscribe: Subscriber sẽ đăng ký nhận message(data) từ một hoặc nhiều topic(channel)



* Một channel có thể có 0 hoặc nhiều subscribers, và message sẽ được gửi tới tất cả các subscribers đã được kết nối. Redis Pub/Sub hỗ trợ linh hoạt nhiều cấu trúc liên kết bao gồm **fan-in**(multiple publishers, single subscriber), **fan-out**(single publisher, multiple subscribers), and **1-1**(one, one).
* Những điểm cần lưu ý:
  + Khi message được gửi đi tới tất các các subscriber thì nó sẽ bị xóa khỏi channel
  + Nếu một subscriber hủy đăng ký và đăng ký lại vào channel thì:
    - Nó sẽ không nhận được bất kỳ tin nhắn nào mà nó đã bỏ lỡ khi hủy đăng ký.
    - Nó sẽ không biết đã đã bị lỡ những tin nhắn nào.
  + Nếu hiện tại không có subscriber nào subscribe vào kênh, tin nhắn sẽ bị loại bỏ và không được gửi cho bất kỳ subscriber nào.
  + Vì message phải đến được tất cả các subscriber hiện hành trước khi bị xóa. Điều này sẽ tốn thời gian nhiều hơn khi có nhiều subscriber.

# Nguồn tham khảo

Tài liệu hướng dẫn elasticsearch ở trang chủ chính:

<https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

Cấu trúc và cách tổ chức dữ liệu của elasticsearch

<https://viblo.asia/p/elasticsearch-cach-to-chuc-du-lieu-OeVKB3oyZkW#_mo-hinh-luu-tru-4>

Giao tiếp với elasticsearch qua power-shell

<https://www.thecodebuzz.com/invoke-webrequest-vs-invoke-restmethod-differences/>

Configuring SAML single-sign-on on the Elastic Stack

<https://www.elastic.co/guide/en/elasticsearch/reference/current/saml-guide-stack.html>

<https://www.elastic.co/guide/en/elasticsearch/reference/current/security-api-saml-logout.html>

too many indexes is not good for ES and that it is better to have as least indexes as possible

Object field type

<https://www.elastic.co/guide/en/elasticsearch/reference/current/object.html>

if the elastic user does not already have a password, it uses a default bootstrap password

<https://discuss.elastic.co/t/detecting-bottleneck-in-elasticsearch-indexing/242060/3>

<https://lukemerrett.com/getting-started-with-elasticsearch/>

Kafka:

<https://www.javatpoint.com/apache-kafka>

Redis:

<https://www.javatpoint.com/redis-tutorial>

<https://laptrinh.vn/books/redis/page/redis-publish-subscribe>

<https://helpex.vn/question/su-dung-redis-lam-pubsub-tren-socket.io-60be6570d24b80926dc00b6d>