Report Week4

Contents

[1. Kiến trúc Lambda Architecture và Kappa Arichitecture 1](#_Toc203158802)

[**1.1** **Giới thiệu** 1](#_Toc203158803)

[**1.2** **Lambda Architecture** 1](#_Toc203158804)

[1.2.1 **Khái niệm** 1](#_Toc203158805)

[**1.2.2** **Các thành phần chính** 2](#_Toc203158806)

[**1.2.3** **Ưu điểm và nhược điểm** 2](#_Toc203158807)

[**1.3** **Kappa Architecture** 3](#_Toc203158808)

[**1.3.1** **Khái niệm** 3](#_Toc203158809)

[**1.3.2** **Ưu điểm và nhược điểm** 3](#_Toc203158810)

[**1.4** **Kết luận** 3](#_Toc203158811)

[**2.** **Kiến trúc Microservices và Kiến trúc Event-Driven** 4](#_Toc203158812)

[**2.1** **Giới thiệu về kiến trúc Microservices** 4](#_Toc203158813)

[**2.2** **Giới thiệu về kiến trúc Event-Driven** 4](#_Toc203158814)

[**2.3** **So sánh giữa hai kiến trúc** 5](#_Toc203158815)

[**2.4** **Kết luận** 5](#_Toc203158816)

[**3.** **Ứng dụng đơn giản đếm lượng pageview theo từng url của website** 6](#_Toc203158817)

1. Kiến trúc Lambda Architecture và Kappa Arichitecture
   1. **Giới thiệu**

Trong thời đại dữ liệu lớn (Big Data), việc thiết kế kiến trúc hệ thống để xử lý dữ liệu hiệu quả và theo thời gian thực là thách thức quan trọng. Hai kiến trúc phổ biến được áp dụng là **Lambda Architecture** và **Kappa Architecture**. Mỗi kiến trúc đều có ưu nhược điểm riêng và phù hợp với từng mục tiêu sử dụng khác nhau.

* 1. **Lambda Architecture**
     1. **Khái niệm**

A diagram of a diagram

AI-generated content may be incorrect.

Kiến trúc Lambda là kiến trúc xử lý dữ liệu được thiết kế để xử lý lượng dữ liệu khồng lồ bằng cách tận dụng cả phương pháp xử lý hàng loạt (batch) và xử lý luồng (stream). Kiến trúc Lambda cân bằng độ trễ, thông lượng và khả năng chịu lỗi bằng cách áp dụng xử lý hàng loạt (batch) để cung cấp chế độ xem toàn diện và chính xác cho dữ liệu, đồng thời áp dụng xử lý luồng thời gian thực (stream) để cung cấp dữ liệu thời gian thực

* + 1. **Các thành phần chính**

Kiến trúc Lambda gồm ba lớp:

* Batch Layer: Tính toàn trước kết quả bằng cách sử dụng hệ thống xử lý phân tán có thể xử lý lượng dữ liệu rất lớn, hướng đến độ chính xác bằng cách xử lý tất cả các dữ liệu. Một số công nghệ sử dụng như: Apache Hadoop, Apache Spark, Hive..
* Speed(or Real-Time) Layer: Xử lý các luồng dữ liệu trong thời gian thực và không có yêu cầu sửa chữa hoặc hoàn thiện. Lớp này hi sinh thông lượng vì nó nhằm mục đích giảm thiệu độ trễ bằng cách xử lý dữ liệu trong thời gian thực. Một số công nghệ sử dụng như: Apache Storm, Spark Streaming, Apache Flink..
* Serving Layer: Dữ liệu được xử lý từ các lớp batch và speed được lưu trữ trong lớp verving, đáp ứng các truy vấn bằng cách trả dữ liệu được tính toán trước hoặc dữ liệu từ kho dữ liệu (data warehouse). Một số công nghệ sử dụng như Apache Cassandra, Elasticsearch, Druid..
  + 1. **Ưu điểm và nhược điểm**
  + Ưu điểm:
    - * Độ chính xác cao: Kết hợp độ chính xác của cả hai lớp (batch) và (speed)
      * Khả năng chịu lỗi: Lớp Batch cung cấp các phương pháp dự phòng đang tin cậy trong trường hợp xử lý thời gian thực bị lỗi
      * Hệ sinh thái trưởng thành: nhiều quy mô lớn đã triển khai Lamdba thành công như Netflix, Uber…
  + Nhược điểm:
* Độ phức tạp cao: Việc duy trì hai quy trình riêng biệt đồng nghĩa với việc cần thêm mã, thao tác và khả năng xảy ra sai sót
* Độ trễ có thể cao hơn: Các quy trình xử lý hàng loạt có thể gây ra chậm trễ ảnh hưởng đến tính kịp thời của hệ thống
* Tốn nhiều tài nguyên: Việc vận hành cả về cơ sở hạ tầng xử lý hàng loạt và thời gian thực có thể dẫn đến chi phí vận hành cao hơn
  1. **Kappa Architecture**
     1. **Khái niệm**

A diagram of a company

AI-generated content may be incorrect.

Được đề xuất bởi Jay Kreps đồng sáng lập của Apache Kafka kiến trúc Kappa ra đời nhằm để khắc phục hạn chế vốn có của kiến trúc lambda. Mục đích chính của Kappa là đơn giản hóa thiết kế tổng thể bằng cách xử lý tất cả dữ liệu dưới dạng một luồng, do đó loại bỏ nhu cầu về một lớp xử lý hàng loạt riêng biệt

Các công nghệ hỗ trợ kiến trúc Kappa tương đối ít, nổi tiếng nhất là Kafka Stream của chính tác giả kiến trúc Kappa. Kafka stream hỗ trợ xử lý luồng mạnh mẽ và lập trình viên có thể dùng SQL trực tiếp trên luồng một cách dễ dàng mà không phải lập trình phức tạp. KSQL cũng cung cấp đầy đủ các câu lệnh từ việc tạo các luồng xử lý, xử lý theo khung thời gian, tạo Materialize View trên luồng cũng như ghi các kết quả xử lý xuống một cơ sở dữ liệu khác.

* + 1. **Ưu điểm và nhược điểm**
  + Ưu điểm:
* Kiến trúc đơn giản: Một quy trình xử lý duy nhất giúp giảm độ phức tạp của mã và dễ dàng bảo trì
* Dễ triển khai, dễ bảo trì: các ứng dụng được thiết kế với độ trễ thấp
  + Nhược điểm:
* Phụ thuộc vào hệ thống lưu trữ Stream: hệ thống phải mạnh mẽ và khả năng chịu lỗi ở quy mô lớn
* Không phù hợp nếu cần xử lý batch phức tạp
  1. **Kết luận**

Sự lựa chọn giữa Lambda và Kappa phụ thuộc vào yêu cầu kinh doanh cụ thể và hạn chế kỹ thuật của bạn

* Chọn lambda khi:
* Bạn cần xử lý dữ liệu có độ chính xác cao, khả năng chịu lỗi và thoải mái quản lý hai luồng riêng biệt
* Hệ thống của bạn phụ thuộc rất nhiều vào việc tính toán dữ liệu lịch sử để báo cáo hoặc tuân thủ quy định
* Bạn có thể xử lý độ phức tạp vận hành và độ trễ tăng lên từ việc xử lý hàng loạt
* Chọn kappa khi:
* Trọng tâm của dự án là xử lý thời gian thực, độ trễ thấp và bạn thích phương pháp tiếp cận đơn giản, hợp lý hóa trên một cơ sở mã duy nhất
* Ứng dụng có tính hướng sự kiện cao: như IoT, game online hoặc gian lận thi cử ….
* Xây dựng một nền tảng dữ liệu hiện đại từ đầu và muốn tận dụng tính linh hoạt của xử lý luồng

1. **Kiến trúc Microservices và Kiến trúc Event-Driven**
   1. **Giới thiệu về kiến trúc Microservices**

Kiến trúc Microservices là một phong cách kiến trúc xây dựng ứng dụng như một tập hợp các dịch vụ nhỏ, độc lập, mỗi dịch vụ chịu trách nhiệm cho một chức năng nghiệp vụ cụ thể. Các dịch vụ này giao tiếp vói nhau thông qua các API được xác định rõ ràng và có thể được triển khai độc lập, cho phép các nhóm phát triển, triển khai và mở rộng ứng dụng hiệu quả hơn

Các tính năng chính của kiến trúc Microservices:

* Độc lập dịch vụ: Mỗi microservices hoạt động độc lập, cho phép phát triển và triển khai riêng biệt
* Không phụ thuộc công nghệ: Các nhóm có thể lựa chọn công nghệ phù hợp cho Microservice
* Khả năng mở rộng: Các dịch vụ có thể được mở rộng riêng lẻ dựa trên nhu cầu, tối ưu hóa các tài nguyên
  1. **Giới thiệu về kiến trúc Event-Driven**

Kiến trúc Event-Driven (EDA) là mô hình thiết kế phần mềm tập trung vào việc sản xuất phát hiện, sử dụng và phản ứng với các sự kiện. Trong kiến trúc này, các ứng dụng được thiết kế để phản hồi các sự kiện, có thể được kích hoạt bởi hành động của người dùng, thay đổi các hệ thống hoặc các dịch vụ bên ngoài. EDA thúc đẩy việc tách rời các thành phần, cho phép các hệ thống linh hoạt và khả năng mở rộng hơn.

Các tính năng chính của EDA:

* Giao tiếp bất đồng bộ: Các thành phần giao tiếp thông qua sự kiện thay vì các lệnh gọi trực tiếp, giảm sự phụ thuộc
* Liên kết lỏng lẽo: Các thành phần có thể phát triển độc lập, cải thiện khả năng bảo trì
* Xử lý thời gian thực: EDA cho phép các ứng dụng phản hồi ngay lập tức với các sự kiện, lý tưởng cho các tình huống đòi hỏi hành động nhanh chóng
  1. **So sánh giữa hai kiến trúc**

|  |  |  |
| --- | --- | --- |
| **Tính năng/khía cạnh** | **Event-Driven Architecture** | **Microservices Architecture** |
| **Mẫu giao tiếp** | Bất đồng bộ thông qua các sự kiện | Chủ yếu là đồng bộ thông qua API, nhưng cũng có thể bao gồm sự kiện |
| **Kết nối giữa các thành phần** | Kết nối lỏng lẻo giữa các thành phần | Các dịch vụ độc lập, nhưng có thể có sự phụ thuộc lân nhau |
| **Quản lý dữ liệu** | Thường sử dụng lưu vết theo sự kiện, quản lý dữ liệu phức tạp | Mỗi dịch vụ có cơ sở dữ liệu riêng, thúc đẩy tính tự chủ |
| **Khả năng mở rộng** | Mở rộng tốt với luồng sự kiện, xử lý hiệu quả các đột biến tải | Các dịch vụ có thể mở rộng độc lập dựa theo yêu cầu |
| **Tính linh hoạt trong phát triển** | Các thành phần có thể phát triển độc lập mà không cần thay đổi lớn | Các nhóm có thể sử dụng công nghệ khác nhau cho từng dịch vụ |
| **Xử lý thời gian thực** | Tuyệt vời cho xử lý dữ liệu theo thời gian thực | Có thể hỗ trợ các công cụ xử lý theo thời gian thực nhung không phải trọng tâm chính |
| **Độ phức tạp** | Có thể làm tăng độ phức tạp trong quản lý sự kiện | Phức tạp do quản lý nhiều dịch vụ và sự phụ thuộc giữa chúng |
| **Tình huống sử dụng** | Lý tưởng cho phân tích thời gian thực. IoT và các ứng dụng theo sự kiện | Tốt cho các ứng dụng lớn như thương mại điển tự, ngân hàng, CMS |
| **Quản lý lỗi** | Lỗi ỏ một số thành phần không ảnh hưởng trực tiếp đến các thành phần khác | Lỗi độc lập cần được quản lý cẩn thận để tránh lan truyền lỗi hàng loạt |

* 1. **Kết luận**

Cả Microservices Architecture và Event-Driven Architecture đều là xu hướng thiết kế hiện đại, hướng đến khả năng mở rộng, dễ bảo trì và linh hoạt. Tùy vào đặc điểm hệ thống, đội ngũ phát triển có thể lựa chọn:

* Microservices để chia nhỏ hệ thống, quản lý hiệu quả.
* Event-Driven khi cần xử lý bất đồng bộ, nhiều event và phản ứng nhanh.

Trong nhiều trường hợp thực tế, kết hợp cả hai kiến trúc sẽ đem lại hiệu quả tốt nhất.

1. **Ứng dụng đơn giản đếm lượng pageview theo từng url của website**

Em xây dựng 1 trang web đơn giản đếm số lần vào trang web

(kể cả f5) sử dụng

* Backend: python(flask)
* Cơ sở dữ liệu: Sqlite
* Frontend: Html