Báo cáo message queue

1. Message queue

* Message queue là một hộp thư, cho phép các thành phần trong một hệ thống hoặc nhiều hệ thống gửi thông tin cho nhau
* Các thành phần cơ bản:
  + Message: Thông tin được gửi (Thường là text, json,..)
  + Message queue: Nơi chứa những message, cho phép producer và consumer có thể trao đổi với nhau
  + Producer: đưa thông tin và message queue
  + Consumer: nhận message từ message queue
* Một chương trình có thể vừa là producer vừa là consumer
* Dùng để giải quyết các vấn đề như:
  + Đảm bảo duration/ recovery: Do message được lưu trong queue, khi một service đang xử lí nhưng bị lỗi thì không lo bị mất dữ liệu, vì có thể lấy message từ trong queue ra và chạy lại. Trong 1 hệ thông có nhiều consumer, nếu một vài consumer bị lỗi cũng không gây crash hệ thống
  + Phân tách hệ thống: Giúp phân tách hệ thống thành nhiều service nhỏ hơn, mỗi service chỉ xử lí 1 chức năng nhất định
  + Hỗ trợ rate limit, batching: Trong trường hợp có nhiều yêu cầu, với message queue, có thể lấy dần dần message trong queue ra xử lí
  + Dễ scaling hệ thống: Khi có nhiều yêu cầu, có thể tăng số lượng consumer lên để xử lí được nhiều message, khi không cần có thể giảm bớt.
* Một số lưu ý:
  + Khó xử lí đồng bộ: Không phải hệ thống nào cũng cần tới message queue.
  + Làm hệ thống phức tạp hơn: cần biết rõ message nào gửi vào queue nào, ai gửi ai nhận
  + Cần đảm bảo message format: Để gửi nhận, 2 phía producer và consumer phải thống nhất format với nhau. Nếu không cản thận lỡ 1 bên thay đổi sẽ làm bên kia không đọc được dữ liệu
  + Cần monitoring queue: Cần có những biện pháp theo dõi để đảm bảo lượng message queue không quá nhiều làm đầy queue. Queue tốt nhất luôn rỗng, các message gửi vào queue đều bị consume hết

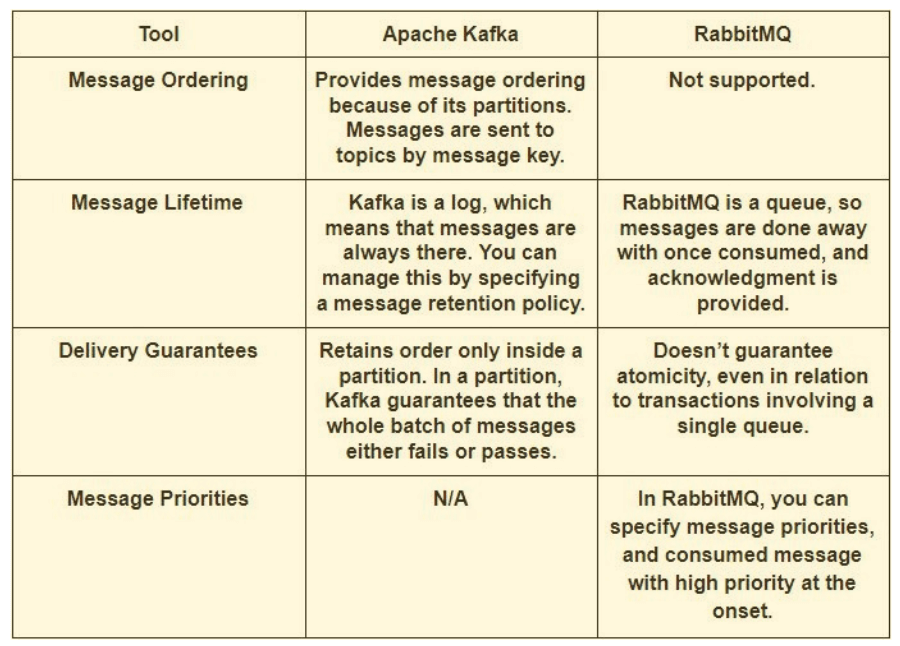
1. RabbitMQ

* Là một message broker sử dụng giao thức AMQP ( Advanced message queue protocol – Giao thức giao nhận tin nhắn sử dụng hàng đợi ). Đây là chương trình đóng vai trò trung gian lưu trữ cũng như điều phối các message giữa producer với consumer
* Cách hoạt động:
  + Producer đẩy message vào exchange. Khi tạo exchange, phải mô tả thuộc loại gì
  + Sau khi exchange nhận message, nó chịu trách nhiệm định tuyến message. Exchange chịu trách nhiệm về các thuộc tính của message
  + Việc binding phải được tạo từ exchange đến hàng đợi. Trong trường hợp này, có 2 binding đến 2 hàng đợi khac nhau từ 1 exchange. Exchange sẽ định tuyến message vào các hàng đợi dựa trên thuộc tính của từng message
  + Các message nằm ở hàng đợi đến khi chúng được xử lí bởi một consumer
  + Consumer xử lí message trong queue
* Các loại exchange
  + Direct exchange: 1 direct exchange sẽ đẩy message đến hàng đợi dựa theo khóa định truyến – routing key. Loại này hữu ích khi muốn phân biệt các thông báo được tạo cho cùng 1 trao đổi bằng cách sử dụng một mã định danh chuỗi đơn giản
  + Fanout exchange: đẩy message đến toàn bộ hàng đợi gắn với nó. Nó gửi 1 bản message đến tất cả những hàng đợi. Sử dụng khi cần một dữ liệu gửi tới nhiều thiết bị khác nhau với cùng một message nhưng xử lí ở mỗi thiết bị khác nhau
  + Topic exchange: Gắn routing key với 1 routing pattern khai báo trong binding. Consumer có thể đăng kí những topic mà nó quan tâm
  + Header exchange: Dùng các thuộc tính header của message để định tuyến. Giống với Topic exchange nhưng định tuyến dựa trên các giá trị tiêu đề
* Một số tính năng:
  + Độ tin cậy: cung cấp 1 loạt các tính năng cho phéo nâng cao hiệu suất và độ tin cậy
  + Định tuyến linh hoạt: message được định tuyến thong qua exchange trước khi vào hàng đợi. Cung cấp một số loại exchange cho logic định tuyến điển hình.
  + Clustering: 1 số máy chủ có thể nhóm với nhau tạo thành một môi giới hợp lí duy nhất
  + Liên kết: đối với các máy chủ cần được kết nối lỏng lẻo và không giảm bớt độ tin cậy, cung cấp 1 mô hình liên kết
  + Hàng đợi cao có sẵn: hàng đợi có thể mở rộng trên nhiều cụm máy chủ, đảm bảo message luôn an toàn
  + Đa giao thức: hỗ trợ nhắn tin qua nhiều giao thức nhắn tin
  + Đa ngôn ngữ: hỗ trờ hầu hết các ngôn ngữ
  + Theo dõi: cung cấp tính năng hỗ trợ theo dỗi để tìm hiểu những gì đang xảy ra

1. Kafka

* Là nền tảng streaming phân tán, có thể mở rộng và là mã nguồn mở.
* Được viết bằng ngôn ngữ Scala và Java.
* Mục đích cung cấp 1 nền tảng có độ trễ thấp và băng thông cao cho việc xử lí các nguồn cấp dữ liệu theo thời gian thực
* Xây dựng trên mô hình publish/ subcribe, tương tự bất kì hệ thống message khác. Các producer gửi message tới broker và nói rằng những message được xử lí bởi coonsumer. Các messages được sửi tới kafka được lưu trữ trong topic và sau đó consumer có thể subcribe tới topic đó và lắng nghe những message này.
* Topic có thể có kích thước rất lớn, như vậy không nên lưu trữ tất cả dữ liệu của 1 topic trên 1 node, dữ liệu cần được phân chi thành nhiều partition để bảo toàn dữ liệu cũng như xử lí dễ dàng hơn. Partitions cho phép thực hiện subcribe song song tới 1 topic cụ thể bằng cách phân chia dữ liệu trong 1 topic cụ thể bằng cách phân chia dữ liệu trong 1 topic cụ thể ra cho nhiều broker khác nhau, mỗi partition có thể được đặt trên một máy riêng biệt – cho phép nhiều consumer đọc dữ liệu từ một topic diễn ra một cách song song.
* Cấu trúc dữ liệu log trong kafka: Chìa khóa chính dẫn tới khả năng mở rộng và hiệu suất của kafka chính là log. Log là một cấu trúc dữ liệu có thứ tự nhất quán mà chỉ hỗ trợ dạng nối thêm (append), không thể chỉnh sửa hay xóa các records từ nó. Nó được đọc từ trái sang phải và được đảm bảo thứ tự các item. Một Producer sẽ ghi message vào log và một hoặc nhiều consumer khác sẽ đọc message từ log tại thời điểm họ lựa chọn. Mỗi entry trong log được định danh bởi một con số gọi là offset, hay nói một cách dễ hiểu hơn, offset giống như chỉ số tuần tự trong một array vậy. Vì chuỗi/offset chỉ có thể được duy trì trên từng node/broker cụ thể và không thể được duy trì đối với toàn bộ cluster, do đó Kafka chỉ đảm bảo sắp xếp thứ tự dữ liệu cho mỗi partition.
* Parsistence data trong Kafa: Kafka lưu trữ tất cả message vào disk (không hề lưu trên RAM) và được sắp xếp có thứ tự trong cấu trúc log cho phép kafka tận dụng tối đa khả năng đọc và ghi lên disk một cách tuần tự. Nó là một cách lựa chọn khá phổ biến để lưu trữ dữ liệu trên disk mà vẫn có thể sử dụng tối đa hóa hiệu năng, có một số lý do chính dưới đây:
  + Kafka có một giao thức mà nhóm các message lại với nhau. Điều này cho phép các request network nhóm các message lại với nhau, giúp giảm thiểu chi phí sử dụng tài nguyên mạng, server, gom các message lại thành một cục và consumer sẽ tìm nạp một khối message cùng một lúc – do đó sẽ giảm tải disk cho hệ điều hành.
  + Kafka phụ thuộc khá nhiều vào pagecache của hệ điều hành cho việc lưu trữ dữ liệu, sử dụng RAM trên máy một cách hiệu quả.
  + Kafka lưu trữ các messages dưới định dạng nhị phân xuyên suốt quá trình (producer > broker > consumer), làm cho nó có thể tận dụng tối ưu hóa khả năng zero-copy. Nghĩa là khi hệ điều hành copy dữ liệu từ pagecache trực tiếp sang socket, hoàn toàn bỏ qua ứng dụng trung gian là kafka.
  + Đọc/ghi dữ liệu tuyến tính trên disk nhanh. Vấn đề làm cho disk chậm hiện nay thường là do quá trình tìm kiếm trên disk nhiều lần. Kafka đọc và ghi trên disk tuyến tính, do đó nó có thể tận dụng tối đa hóa hiệu suất trên disk.
* Consumer đọc các messages từ bất kỳ partition nào, cho phép bạn mở rộng lượng message được sử dụng tương tự như cách các producer cung cấp message. Consumer cũng được tổ chức thành các consumer groups cho một topic cụ thể – mỗi consumer bên trong group đọc message từ một partition duy nhất, để tránh việc có 2 consumer cùng xử lý đọc cùng một message 2 lần và toàn bộ group xử lý tất cả các message từ toàn bộ topic.
* Zookeeper đóng vai trò là nơi lưu trữ dữ liệu phân tán dạng key-value. Nó được tối ưu hóa cho tác vụ đọc nhanh nhưng ghi chậm. Kafka sử dụng Zookeeper để thực hiện việc bầu chọn leader của Kafka broker và topic partition. Zookeeper cũng được thiết kế cho khả năng chịu lỗi cao, do đó Kafka phụ thuộc khá nhiều vào Zookeeper. Producer và consumer không tương tác trực tiếp với Zookeeper để biết leader của partition hay những metadata khác, thay vào đó chúng sẽ truy vấn metadata tới Kafka broker – sau đó Kafka tương tác với Zookeeper và gửi phản hồi metadata về lại cho chúng. Sử dụng để lưu trữ tất cả metadata như:
  + Offset cho mỗi partition của consumer group
  + ACL (Access control list) – được sử dụng cho việc giới hạn truy cập/ủy quyền
  + Quota của consumer/producer – số lượng message tối đa mỗi giây
  + Partition Leader và trạng thái của chúng

1. So sáng kafka và RabbitMQ
   1. Kiến trúc

* RabbitMQ:
  + Thông điệp message chung: sử dụng các biến thể của request/reply, point to point, and pub-sub
  + Thông minh môi giới/Dumb người tiêu dùng mô hình-nhất quán cung cấp các thông điệp đến với tiêu dùng, ở xung quanh cùng một tốc độ như các nhà môi giới theo dõi tình trạng người tiêu dùng.
  + Nền tảng— hỗ trợ tốt, có sẵn cho Java, thư viện khách hàng, .net, Ruby, node. js. Cung cấp hàng chục plugins.
  + Giao tiếp— có thể đồng bộ hoặc không đồng bộ.
  + Tình huống triển khai— cung cấp các tình huống triển khai phân tán.
  + Cụm nhiều nút cụm liên kết-không dựa trên các dịch vụ bên ngoài, Tuy nhiên, các plugin hình thành cụm cụ thể có thể sử dụng DNS, API, lãnh sự, v.v.
* Kafka:
  + đăng ký tin nhắn và nền tảng stream-bền, nhanh chóng và khả năng mở rộng.
  + Lưu trữ thư bền— như log, chạy trong cụm máy chủ, Giữ luồng record trong các topic (danh mục).
  + Tin nhắn— tạo thành một giá trị, một key và một dấu thời gian.
  + Dumb môi giới/mô hình người tiêu dùng thông minh-không cố gắng theo dõi những tin nhắn được đọc bởi coonsumer và chỉ giữ tin nhắn chưa đọc. Kafka giữ tất cả các tin nhắn trong một khoảng thời gian.
  + Yêu cầu các dịch vụ bên ngoài chạy— trong một số trường hợp Apache Zookeeper.
  1. Phương pháp tiếp cận
* RabbitMQ: RabbitMQ sử dụng một mô hình Push và ngừng áp đảo consumer thông qua một giới hạn Prefetch xác định trên consumer. Điều này có thể được sử dụng để nhắn tin độ trễ thấp. Mục đích của mô hình push là để phân phối các tin nhắn riêng lẻ và nhanh chóng, để đảm bảo rằng công việc được song hành đồng đều và các tin nhắn được xử lý xấp xỉ theo thứ tự mà họ đến trong hàng đợi.
* Kafka: Kafka sử dụng mô hình pull. Consumers yêu cầu nhiều mesage từ một cụ thể bù. Kafka cho phép tổng hợp dài, ngăn ngừa vòng chặt chẽ khi không có thông điệp nào qua bù đắp. Một mô hình pull là hợp lý cho Kafka vì phân vùng của nó. Kafka cung cấp thông điệp trong một phân vùng mà không có Consumers contending. Điều này cho phép người dùng tận dụng việc phân phối tin nhắn để gửi tin nhắn hiệu quả và thông lượng cao hơn.
  1. Xử lí message
  2. Hiện năng
* RabbitMQ: RabbitMQ cũng có thể xử lý một triệu message mỗi giây nhưng đòi hỏi nhiều tài nguyên hơn (khoảng 30 nút). Có thể sử dụng RabbitMQ cho nhiều trường hợp sử dụng như Kafka, nhưng sẽ cần phải kết hợp nó với các công cụ khác như Apache Cassandra.
* Kafka: Kafka cung cấp hiệu suất cao hơn nhiều so với các brokers tin nhắn như RabbitMQ. Nó sử dụng tuần tự disk I/O để tăng hiệu suất, làm cho nó một lựa chọn phù hợp để thực hiện hàng đợi. Nó có thể đạt được công suất cao (hàng triệu tin nhắn mỗi giây) với nguồn lực hạn chế, một điều cần thiết cho các trường hợp sử dụng dữ liệu lớn.

4.5 Khi nào cần sử dụng

- RabbitMQ: RabbitMQ có thể được sử dụng khi máy chủ web cần phải nhanh chóng đáp ứng các yêu cầu. Điều này giúp loại bỏ sự cần phải thực hiện các hoạt động tài nguyên trong khi người dùng chờ kết quả. RabbitMQ cũng được sử dụng để chuyển tải một tin nhắn đến người nhận khác nhau để tiêu thụ hoặc chia sẻ tải giữa các thành phần dưới tải cao (20K + tin nhắn/giây).

+ Các ứng dụng cần hỗ trợ các giao thức cũ, chẳng hạn như STOMP, MQTT, AMQP, 0-9-1.

+ Kiểm soát chi tiết hơn nhất quán/bộ bảo lãnh trên cơ sở mỗi thư

+ Định tuyến phức tạp đến người tiêu dùng

+ Các ứng dụng cần một loạt các publish/subscribe, point-to-point request/reply

* Kafka: Kafka cung cấp broker chính nó và được thiết kế theo hướng kịch bản. Gần đây, nó đã được thêm Kafka Streams, một thư viện khách hàng để xây dựng các ứng dụng và microservice. Apache Kafka hỗ trợ các trường hợp sử dụng như số liệu, theo dõi hoạt động, kết tập log, xử lý trực tiếp, record thực hiện và nguồn cung cấp sự kiện.
  + Stream với định tuyến phức tạp, thông qua các sự kiện 100K/SEC hoặc nhiều hơn, với "ít nhất một lần" đặt hàng phân hoạch
  + Các ứng dụng yêu cầu một lịch sử stream, phân phối trong "ít nhất một lần" đặt hàng phân hoạch. Khách hàng có thể xem "phát lại" của luồng sự kiện.
  + Sự kiện tìm nguồn cung ứng, mô hình hóa thay đổi cho một hệ thống như một chuỗi các sự kiện.
  + Dòng dữ liệu xử lý trong đường ống nhiều giai đoạn. Các đường ống dẫn tạo đồ thị của dòng dữ liệu