

La arquitectura orientada a eventos (EDA) y los sistemas de gestión de colas

Backend



La arquitectura orientada a eventos (EDA) y los sistemas de gestión de colas



Imagínate en la dirección de una gran película de Hollywood. Tienes actores (los microservicios), un guion (tu lógica de negocio) y una audiencia ansiosa por una experiencia inolvidable (tus usuarios). Pero, ¿cómo asegurarte de que cada actor esté en el lugar correcto, en el momento adecuado, con el diálogo perfecto?

A través de este fastbook, te guiaremos en esta emocionante travesía, desvelando los secretos de la arquitectura basada en eventos (EDA) y los sistemas de gestión de colas.

La EDA trabaja incansablemente detrás de escena, asegurándose de que cada parte de tu sistema esté sincronizada, como una orquesta bien afinada. Cada vez que un usuario hace clic, que un pedido se realiza o que el inventario se actualiza, la EDA está ahí, moviendo los hilos, asegurándose de que cada microservicio reciba su señal para actuar.

Piensa en los sistemas de colas como esos asistentes que corren por el set, susurrando mensajes importantes de un actor a otro. "Psst, el inventario necesita actualización", "Hey, este cliente acaba de hacer un pedido". Estos susurros no son al azar; son mensajes bien orquestados que mantienen la película (tu ecommerce) avanzando suavemente.

¡Vamos allá!

 Arquitectura basada en eventos (EDA)

 EDA en un ecommerce

 Sistemas de gestores de colas

 Kafka

 RabbitMQ

 Kafka vs. RabbitMQ

 Patrones y buenas prácticas

 Conclusiones

Arquitectura basada en eventos (EDA)



Una arquitectura orientada a eventos (EDA, por sus siglas en inglés) es un paradigma de diseño de software en el que los componentes y servicios del sistema interactúan a través de la producción, detección y reacción a eventos.

En esta arquitectura, los eventos son **objetos significativos** que se generan a partir de acciones o cambios de estado en el sistema, y su manejo es central para la lógica y la comunicación en la aplicación. Aquí, hay algunas claves para explicar qué es una EDA:

Conceptos clave de EDA

Evento

- Un evento es un cambio significativo en el estado o una actualización importante dentro de un sistema.
- Ejemplos de eventos incluyen una transacción completada, un nuevo pedido de usuario o un cambio de datos.

Productores de eventos

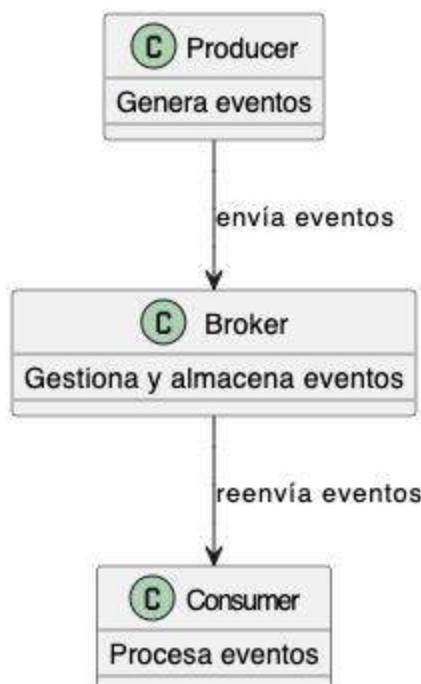
- Los productores son componentes o servicios que generan eventos.
- No necesariamente conocen o manejan cómo se procesan estos eventos.

Consumidores de eventos

- Los consumidores escuchan y reaccionan a los eventos.
- Pueden realizar acciones como actualizar una base de datos, iniciar un proceso o alterar el flujo de trabajo en respuesta a un evento.

Broker de mensajes o bus de eventos

- Un intermediario (como Kafka o RabbitMQ) que maneja la transmisión de eventos entre productores y consumidores.
- Proporciona un mecanismo para publicar eventos sin saber qué consumidores existen.



Tipos de eventos

Aunque hay multitud de tipos de eventos y numerosos motivos por los que pueden ser generados, vamos a agruparlos en **2 conjuntos**:

1

Acciones de usuarios

Estos eventos se generan como **resultado de las interacciones directas de los usuarios con el sistema**. Pueden ser acciones en una interfaz de usuario, como clics, navegación, ingreso de datos o cualquier otra forma de entrada por parte del usuario.

Características:

INTERACTIVIDAD	INMEDIATEZ	PERSONALIZACIÓN
Están estrechamente ligados a la experiencia del usuario, ya que representan sus acciones y decisiones.		

INTERACTIVIDAD	INMEDIATEZ	PERSONALIZACIÓN
Suelen requerir una respuesta rápida del sistema para mantener una experiencia de usuario fluida y eficiente.		

INTERACTIVIDAD	INMEDIATEZ	PERSONALIZACIÓN
Pueden ser utilizados para entender y adaptar el comportamiento del sistema a las necesidades y preferencias individuales del usuario.		

Ejemplos:

- Un usuario que añade un producto al carrito de compras en un ecommerce.
- La selección de una fecha en un calendario para ver eventos o disponibilidades.

2

Cambios de estado

Representan alteraciones en el estado interno o externo del sistema o de sus componentes. Estos eventos pueden ser desencadenados por acciones de usuarios, pero también pueden ocurrir independientemente de la interacción humana, como resultado de **procesos automatizados, cambios en datos externos o reglas de negocio**.

Características:

Automatización

Muchas veces son el resultado de procesos automatizados o reglas de negocio.

Integración

Pueden involucrar la interacción con otros sistemas o servicios, reflejando cambios en el estado o en los datos que provienen de fuentes externas.

Gestión del sistema

Son clave para el monitoreo, control y adaptación del estado del sistema a lo largo del tiempo.

Ejemplos:

- Actualización del inventario en un sistema de ecommerce después de una compra.
- Cambio en el estado de un pedido, como pasar de 'pendiente' a 'enviado'.

Ventajas de EDA

- Desacoplamiento:** en EDA, los productores y consumidores están desacoplados, lo que significa que pueden operar independientemente unos de otros. Este desacoplamiento permite una mayor flexibilidad y escalabilidad del sistema.
- Escalabilidad:** permite escalar componentes individualmente y manejar grandes volúmenes de eventos.
- Flexibilidad y agilidad:** facilita la incorporación de nuevos componentes y servicios en el sistema.
- Resiliencia:** al estar desacoplados, los fallos en un componente no paralizan todo el sistema.
- Tiempo real y reactividad:** permite reaccionar a eventos en tiempo real, lo que es crucial en muchos sistemas modernos.

Aplicaciones de EDA

EDA es especialmente útil en **entornos donde la reactividad, la escalabilidad y la flexibilidad son críticas**. Este enfoque fomenta un diseño de sistema robusto y adaptable, capaz de manejar dinámicamente la evolución de los requisitos y las condiciones del entorno.

- Sistemas de comercio electrónico:** para manejar pedidos, inventario y transacciones de usuarios.
- Aplicaciones financieras:** como en el procesamiento de transacciones y análisis de mercado en tiempo real.
- Internet de las cosas (IoT):** en la gestión de datos de sensores y actuadores distribuidos.
- Microservicios:** para comunicar servicios en arquitecturas complejas y distribuidas.

EDA en un ecommerce



Nuestro ecommerce

Pensemos en un ecommerce que está diseñado utilizando una arquitectura de microservicios. Esta arquitectura divide la funcionalidad del sistema en servicios más pequeños, autónomos y desacoplados, cada uno de ellos responsable de una parte específica de la funcionalidad del negocio. **Los microservicios interactúan entre sí a través de una combinación de llamadas a API y un modelo de eventos para comunicarse de forma asincrónica.**

Microservicios clave

- 1 **Servicio de gestión de productos:** responsable del catálogo de productos, incluyendo detalles, precios y disponibilidad.
- 2 **Servicio de gestión de pedidos:** maneja la creación, actualización y seguimiento de pedidos.
- 3 **Servicio de gestión de clientes:** administra cuentas de usuario, preferencias y datos de contacto.
- 4 **Servicio de gestión de inventario:** controla el stock de productos, actualizaciones de inventario y alertas de bajo stock.
- 5 **Servicio de pago:** gestiona las transacciones, incluyendo autorización, procesamiento de pagos y seguridad.
- 6 **Servicio de logística y envío:** coordina la logística de entrega y gestiona los detalles de envío.

Eventos

Implementar una arquitectura EDA en un ecommerce requiere una cuidadosa planificación de los eventos que serán **generados y consumidos** por cada servicio.

Veamos algunos de los eventos que podríamos tener en nuestro ecommerce en cada microservicio, considerando su producción y consumo:

Servicio de gestión de productos

- Eventos generados:
 - ProductoCreado.
 - ProductoActualizado.
 - ProductoEliminado.
- Eventos consumidos:
 - InventarioActualizado (para reflejar cambios en la disponibilidad del producto).

Servicio de gestión de pedidos

- Eventos generados:
 - PedidoCreado
 - PedidoActualizado (cambios en el estado del pedido, como 'procesando', 'enviado' o 'entregado').
 - PedidoCancelado
- Eventos consumidos:
 - ProductoActualizado (para actualizar precios o disponibilidad en pedidos activos).
 - ClienteActualizado (para cambios en la información del cliente).

Servicio de gestión de clientes

- Eventos generados:
 - ClienteCreado.
 - ClienteActualizado.
 - ClienteEliminado.
- Eventos consumidos:
 - PedidoCreado (para rastrear historial de compras).

Servicio de gestión de inventario

- Eventos generados:
 - InventarioActualizado (cambios en stock, alertas de bajo inventario).
- Eventos consumidos:
 - PedidoCreado (para reducir stock).
 - PedidoCancelado (para reponer stock).

Servicio de pago

- Eventos generados:
 - PagoProcesado.
 - PagoFallido.
- Eventos consumidos:
 - PedidoCreado (para iniciar el proceso de pago).

Servicio de logística y envío

- Eventos generados:
 - EnvioIniciado.
 - EnvioActualizado (cambios en el estado del envío, como 'en tránsito' o 'entregado').
 - EnvioCompletado.
- Eventos consumidos:
 - PedidoActualizado (cuando un pedido está listo para ser enviado).

Esta estructura de eventos permite que cada servicio de nuestro ecommerce funcione de manera independiente, pero coordinada, a través de una comunicación basada en eventos. Los eventos **facilitan el desacoplamiento de los servicios**, mejoran la escalabilidad y la capacidad de respuesta del sistema, y permiten una mayor flexibilidad para futuras expansiones o modificaciones en la arquitectura.

Veamos un ejemplo **de documentación de evento**:

Nombre del evento

- **Descripción breve:** *ProductoAñadidoAlCarrito* (se genera cuando un usuario añade un producto a su carrito de compras).

Detalles del evento

- 1 **ID del evento:** evt-producto-añadido-101.
- 2 **Tipo de evento:** evento de dominio.
- 3 **Origen del evento:** servicio de carrito de compras.
- 4 **Consumidores del evento:** servicio de recomendaciones, servicio de análisis de usuario.
- 5 **Datos del evento:** incluye ID del usuario, ID del producto, cantidad añadida y *timestamp*.

Especificaciones técnicas

Formato

JSON.

Esquema del mensaje

- **usuarioid:** string.
- **productoid:** string.
- **cantidad:** integer.
- **timestamp:** datetime.

Validaciones

Verificar que el *productoid* y *usuarioid* sean válidos y que la cantidad sea un número positivo.

Versionado

v1.0.

Reglas de negocio asociadas

- El evento no se genera si el producto está fuera de stock.
- Se actualiza la cantidad del producto si ya está en el carrito.

Casos de uso

Producción del evento

Cada vez que un usuario añade un producto a su carrito, ya sea desde la página del producto o a través de una recomendación.

Consumo del evento

El Servicio de Recomendaciones utiliza esta información para ajustar futuras sugerencias; el Servicio de Análisis de Usuario lo usa para comprender mejor el comportamiento de compra.

Seguridad y privacidad

Consideraciones de seguridad

Los ID de usuario y producto se manejan de manera anónima para evitar la identificación directa del usuario.

Consideraciones de privacidad

Cumple con la GDPR y otras normativas de privacidad en la gestión de datos del usuario.

Historial de cambios

- **Registro de versiones.** v1.0 (2023-12-07): creación del evento.

 Puedes [descargar la plantilla Word](#) desde el repositorio GitHub del lab. Esta plantilla puede ser adaptada o ampliada según las necesidades específicas de tu proyecto o empresa. Es importante mantener esta documentación actualizada y accesible para todos los equipos involucrados en el desarrollo y mantenimiento del sistema EDA.

Sistemas de gestores de colas



Los sistemas de gestión de colas son una parte fundamental de la arquitectura de software en muchas aplicaciones modernas en entornos de alto rendimiento y escalabilidad.

En su forma más básica, un sistema de colas proporciona un mecanismo para gestionar el flujo de información o tareas entre diferentes partes de un sistema, de manera que estas tareas puedan ser procesadas de forma asíncrona. Esto significa que el envío y la recepción de información no necesitan ocurrir al mismo tiempo, permitiendo así una mayor flexibilidad y eficiencia en el manejo de cargas de trabajo.

Aplicación en los sistemas EDA

Los sistemas de colas son **un componente vital en la implementación de una arquitectura orientada a eventos**, proporcionando un medio robusto y eficiente para manejar eventos, mejorar la escalabilidad, y asegurar la resiliencia del sistema.

Con ellos conseguiremos o mejoraremos lo siguiente:

Desacoplamiento de componentes

En EDA, los sistemas de colas permiten desacoplar los productores de eventos de los consumidores de eventos. Esto significa que el componente que emite un evento no necesita saber qué otros componentes actuarán en respuesta a ese evento.

Esta separación mejora la modularidad y la flexibilidad del sistema, permitiendo que las partes del sistema se desarrollen, escalen y mantengan de manera independiente.

Manejo de cargas de trabajo

- **Escalabilidad:** los sistemas de colas pueden manejar fluctuaciones en el volumen de eventos, lo que es crucial en aplicaciones de ecommerce donde los picos de tráfico son comunes.
- **Rendimiento:** permiten que los eventos se procesen de manera asincrónica, lo que significa que las tareas intensivas en recursos no bloquean el flujo principal de ejecución del programa.

Resiliencia y fiabilidad

- **Tolerancia a fallos:** los sistemas de colas pueden ofrecer mecanismos para reintentar automáticamente el procesamiento de eventos en caso de fallos, mejorando así la resiliencia del sistema.
- **Persistencia:** algunos sistemas de colas garantizan que los eventos no se pierdan, incluso en caso de fallos del sistema, almacenando los eventos de manera persistente hasta que se procesan con éxito.

Monitorización y análisis

- **Seguimiento de eventos:** los sistemas de colas facilitan el seguimiento y monitoreo de eventos, lo que es esencial para la depuración, el análisis de rendimiento y la auditoría de seguridad.
- **Datos en tiempo real:** pueden utilizarse para alimentar sistemas de análisis en tiempo real, proporcionando información valiosa sobre el comportamiento del usuario y el rendimiento del sistema.

Sistema gestor de cola en un ecommerce

En un ecommerce, **un sistema gestor de colas** actúa como un intermediario para gestionar las solicitudes y tareas que no necesitan ser procesadas inmediatamente. Estas tareas se encolan y se procesan de manera asíncrona, lo que ayuda a mejorar la eficiencia del sistema, balancear la carga y garantizar que las operaciones intensivas en recursos no bloqueen el procesamiento principal del servidor.

Escenario: un usuario realiza un pedido en un sitio de ecommerce.

Creación del pedido (productor)

- El usuario completa su compra y realiza el pedido.
- El sistema de frontend envía un mensaje al sistema gestor de colas con los detalles del pedido. Este mensaje entra en la cola de 'Pedidos Pendientes'.

Procesamiento asincrónico (cola):

- El pedido se almacena en la cola hasta que un servicio de backend (consumidor) esté disponible para procesarlo.
- La cola asegura que los pedidos se procesen en el orden en que se reciben (o según cualquier otra lógica de priorización).

Gestión del inventario y confirmación (consumidor)

- Un servicio de backend toma el pedido de la cola.
- Realiza varias operaciones, como actualizar el inventario, confirmar el pago y preparar la orden para envío.
- Una vez completadas estas tareas, el servicio puede enviar una confirmación al usuario, ya sea directamente o colocando un mensaje en otra cola dedicada a notificaciones.

Manejo de errores

Si ocurre un error en el procesamiento (por ejemplo, un producto no está en stock), el sistema puede reintentar automáticamente el proceso o enviar una notificación de error.

La implementación de un sistema gestor de colas en un entorno de ecommerce ayuda a manejar eficientemente las operaciones en segundo plano, mejora el rendimiento del sitio web y asegura una experiencia de usuario fluida y profesional. Además, proporciona una arquitectura más resiliente y escalable, elementos clave para el éxito en el dinámico mundo del ecommerce.

Dos de los principales gestores de colas de eventos del mercado son: **Kafka** y **RabbitMQ**.

Kafka



Apache Kafka es un **sistema de procesamiento de flujos de datos y gestión de colas de mensajes distribuido**, originalmente desarrollado por LinkedIn y luego, donado a la Apache Software Foundation, donde se ha convertido en uno de los sistemas de mensajería y streaming de datos más populares y utilizados en la industria.

Características principales de Kafka

1

Distribuido y escalable: Kafka está diseñado para ser distribuido desde su base, lo que facilita su escalabilidad horizontal. Puede manejar grandes volúmenes de datos y tráfico elevado de mensajes.

2

Alto rendimiento: su arquitectura permite altas tasas de *throughput* tanto para la lectura como para la escritura de mensajes, lo que lo hace adecuado para escenarios de big data y procesamiento en tiempo real.

3

Almacenamiento de mensajes duradero: Kafka almacena los mensajes en discos, lo que garantiza la durabilidad y permite que los consumidores puedan leer mensajes antiguos.

4

Modelo de publicación-subscripción: funciona en un modelo de productor-consumidor, donde los productores publican mensajes en *topics* y los consumidores se suscriben a esos *topics* para recibir los mensajes.

5

Particionamiento de datos: los *topics* en Kafka se pueden dividir en particiones, lo que permite un paralelismo en el procesamiento y una alta disponibilidad al distribuir los datos a través de varios nodos.

6

Tolerancia a fallos: Kafka es resistente a fallos de nodos y soporta replicación de datos para asegurar que la información no se pierda en caso de un fallo en un servidor.

Estructura básica

Topics

En Kafka, los mensajes se organizan en *topics*. Un *topic* es una categoría o un nombre de alimentación a la que los registros (mensajes) se publican.

Productores (producers)

Los productores son procesos que publican datos (envían mensajes) a los *topics* de Kafka. En el contexto de una aplicación, un productor podría ser cualquier servicio o componente que genere eventos o datos.

Consumidores (consumers)

Los consumidores leen los mensajes de los *topics*. Pueden suscribirse a uno o varios *topics* y procesar los datos recibidos.

Brokers

Un servidor Kafka se llama broker. Los brokers manejan el almacenamiento de mensajes en disco y el procesamiento de lecturas y escrituras de los consumidores y productores. En un entorno de producción, Kafka se ejecuta como un clúster de uno o más brokers para mejorar la tolerancia a fallos y el rendimiento.

ZooKeeper

Kafka usa ZooKeeper para gestionar y coordinar los *brokers*. ZooKeeper sirve para mantener la sincronización entre los distintos nodos del clúster de Kafka.

Particiones y réplicas

Cada *topic* en Kafka se divide en particiones para permitir el escalado horizontal; los datos se distribuyen y se balancean entre estas particiones. Las réplicas de particiones se utilizan para garantizar la tolerancia a fallos.

Uso de Kafka

Kafka se utiliza en **una amplia gama de aplicaciones**, que incluyen:

Streaming de datos en tiempo real

Para procesar y analizar flujos de datos en tiempo real, como datos de sensores, registros de servidores o acciones de usuarios en una aplicación web.

Sistemas de colas de mensajes

Como un sistema de mensajería robusto para desacoplar sistemas y procesos dentro de una arquitectura de microservicios.

Integración de datos y ETL

Para recopilar y mover grandes volúmenes de datos entre diferentes sistemas y aplicaciones.

Monitoreo y logística

En sistemas de monitoreo, para recopilar y procesar métricas y logs de diversos sistemas y aplicaciones.

En resumen, Apache Kafka es una solución poderosa y versátil para el manejo de grandes volúmenes de datos en tiempo real, la integración de sistemas y el procesamiento de flujos de datos, siendo una pieza clave en muchas arquitecturas de sistemas modernas.

RabbitMQ



Qalentum Lab

RabbitMQ es un sistema de mensajería de código abierto que actúa como un intermediario para la gestión de mensajes entre aplicaciones.

Es conocido por **su robustez, escalabilidad y facilidad de uso** y se utiliza ampliamente en la industria para la implementación de comunicaciones entre sistemas y componentes de software.

Características principales de RabbitMQ

1

Basado en el protocolo AMQP: RabbitMQ implementa el protocolo de mensajería avanzado (AMQP), que es un protocolo estándar para sistemas de mensajería.

2

Modelo de mensajería flexible: ofrece un modelo de mensajería altamente flexible que soporta patrones como publicación-suscripción, enrutamiento de mensajes, y trabajo en colas.

3

Durabilidad y entrega garantizada: permite configuraciones para garantizar que los mensajes no se pierdan, a través de la persistencia de mensajes y confirmaciones de entrega.

4

Escalabilidad y alta disponibilidad: RabbitMQ puede ser configurado para funcionar en clústeres, proporcionando alta disponibilidad y la capacidad de escalar horizontalmente.

5

Gestión de colas y exchanges: los mensajes se envían a través de exchanges y colas. Los exchanges toman los mensajes y los enrutan a las colas apropiadas basándose en reglas de enrutamiento.

6

Rendimiento y eficiencia: aunque está diseñado para la fiabilidad, RabbitMQ también ofrece un buen rendimiento y es capaz de manejar un gran número de mensajes simultáneamente.

7

Soporte para múltiples protocolos de mensajería: además de AMQP, RabbitMQ soporta MQTT, STOMP y otros protocolos, lo que lo hace versátil para diferentes necesidades de mensajería.

Estructura básica

Exchange

En RabbitMQ, un exchange recibe mensajes de los productores y los reenvía a las colas basándose en ciertas reglas. Los tipos de exchanges incluyen *direct*, *topic*, *headers* y *fanout*.

Colas (queues)

Las colas almacenan los mensajes hasta que son consumidos. Los mensajes en RabbitMQ se enrutan desde un exchange a una cola.

Productores (producers)

Los productores son aplicaciones o procesos que envían mensajes a un exchange.

Consumidores (consumers)

Los consumidores son aplicaciones o procesos que extraen mensajes de las colas.

Binding

Un binding es una configuración que define cómo los mensajes son enrutados desde el exchange a las colas.

Broker

Un RabbitMQ, el broker es el servidor que gestiona los exchanges, colas, bindings, productores y consumidores.

Uso de RabbitMQ

RabbitMQ se utiliza en **una variedad de escenarios**, que incluyen:

Desacoplamiento de componentes de aplicaciones

Como un intermediario en arquitecturas de microservicios, ayudando a desacoplar los servicios y gestionar la comunicación entre ellos.

Colas de trabajo y distribución de carga

En sistemas que requieren procesamiento de tareas en segundo plano o distribución de carga entre varios trabajadores.

Sistemas de notificaciones y alertas

Para enviar notificaciones o alertas en tiempo real a diferentes usuarios o servicios.

Integraciones entre aplicaciones

Como un bus de mensajería para integrar diferentes aplicaciones, sistemas y bases de datos.

Balanceo de carga y tolerancia a fallos

En sistemas donde es crucial balancear la carga entre múltiples nodos y asegurar la continuidad del servicio.

En conclusión, RabbitMQ es una herramienta poderosa y confiable para la gestión de mensajes en arquitecturas de software complejas.

Su flexibilidad, capacidad de escalabilidad y soporte para múltiples protocolos la hacen ideal para una amplia gama de aplicaciones, desde sistemas simples de notificaciones hasta arquitecturas de microservicios avanzadas.

Kafka vs. RabbitMQ



Veamos, a continuación, **una tabla comparativa entre Kafka y RabbitMQ** para entender sus **diferencias y similitudes**, así como para identificar en qué situaciones es mejor usar uno u otro.

Característica	Apache Kafka	RabbitMQ
Modelo de mensajería	Basado en logs, orientado a flujos de datos y eventos.	Modelo de mensajería tradicional con enfoque en la entrega de mensajes individuales.
Protocolo	Propio, basado en TCP.	AMQP, con soporte para MQTT, STOMP y otros.
Rendimiento en alto volumen	Muy alto rendimiento con grandes volúmenes de datos	Buen rendimiento, aunque puede ser menos óptimo para volúmenes muy grandes.
Escalabilidad	Alta escalabilidad horizontal.	Escalable, pero con una complejidad mayor en la gestión de clústeres.
Durabilidad y retención de mensajes	Alta durabilidad, retiene grandes volúmenes de datos por largos períodos.	Durabilidad configurable, generalmente orientada a mensajes transitorios.

Gestión de estado	No mantiene estado de los consumidores.	Mantiene y gestiona el estado de los consumidores.
Casos de uso ideales	Streaming de datos en tiempo real, procesamiento de eventos, sistemas de log.	Mensajería entre microservicios, tareas en segundo plano o notificaciones.
Complejidad	Mayor complejidad en configuración y manejo.	Más sencillo y directo de configurar y usar.
Persistencia de mensajes	Sí, basado en disco.	Sí, con varias opciones de configuración.
Tolerancia a fallos	Alta, con replicación de datos.	Alta, con soporte para clústeres y espejado de colas.

Cuándo usar Apache Kafka

1. **Procesamiento de eventos en tiempo real:** ideal para aplicaciones que requieren el procesamiento de grandes flujos de datos en tiempo real, como el monitoreo de transacciones financieras.
1. **Sistemas de log y monitoreo:** para sistemas que capturan y procesan grandes volúmenes de logs.
1. **Plataformas de datos y big data:** en escenarios donde se necesitan integrar y procesar grandes cantidades de datos de diversas fuentes.

Cuándo usar RabbitMQ

1. **Comunicación entre microservicios:** cuando se requiere un sistema confiable y fácil de usar para la comunicación entre diferentes servicios en una arquitectura de microservicios.
1. **Colas de trabajo y tareas en segundo plano:** en aplicaciones que necesitan gestionar tareas en segundo plano, como el envío de correos electrónicos o la generación de reportes.
1. **Sistemas de notificaciones:** para el envío de notificaciones en tiempo real, donde se requiere asegurar la entrega del mensaje, pero no necesariamente manejar grandes volúmenes de datos.

En algunos casos, tanto Kafka como RabbitMQ podrían ser adecuados, especialmente, en sistemas donde la prioridad es la mensajería confiable y no necesariamente el volumen de datos o el procesamiento en tiempo real. En estas situaciones, **la elección puede depender más de otros factores** como:

- La familiaridad del equipo con la tecnología.
- La infraestructura existente.
- Requisitos específicos de integración.

Patrones y buenas prácticas



Qualentum Lab

Patrón *Event Sourcing*

Event Sourcing es **un patrón de diseño de software** en el que todos los cambios en el estado de una aplicación se almacenan como una secuencia de eventos. En lugar de almacenar simplemente el estado actual de los datos en una base de datos, *Event Sourcing* almacena cada cambio como un evento individual que puede ser reproducido para reconstruir el estado del sistema en cualquier momento.

Características clave:

Inmutabilidad de los eventos

Los eventos una vez registrados no se modifican ni eliminan.

Registro completo

Cada cambio en el estado del sistema se registra como un evento.

Reconstrucción del estado

El estado actual se reconstruye aplicando secuencialmente los eventos registrados.

Facilita el deshacer/rehacer

Permite fácilmente deshacer acciones o reconstruir estados históricos.

Auditoría y rastreo

Proporciona un registro detallado de las acciones, lo cual es útil para auditorías y diagnósticos.

Consideraciones:

COMPLEJIDAD

RENDIMIENTO

PROCESAMIENTO DE EVENTOS

La implementación de *Event Sourcing* puede añadir complejidad al diseño del sistema.

COMPLEJIDAD

RENDIMIENTO

PROCESAMIENTO DE EVENTOS

Es importante gestionar eficientemente el almacenamiento y la reproducción de eventos para mantener un alto rendimiento.

COMPLEJIDAD	RENDIMIENTO	PROCESAMIENTO DE EVENTOS
Puede requerirse un sistema robusto para el procesamiento de eventos como Apache Kafka.		

Patrón CQRS (*Command Query Responsibility Segregation*)

CQRS (*Command Query Responsibility Segregation*) es **un patrón de diseño en arquitectura de software** que separa claramente las operaciones de lectura (*queries*) de las operaciones de escritura (*commands*) en la aplicación. Este patrón se basa en el principio de que los modelos para actualizar la información no necesariamente tienen que ser los mismos que los modelos utilizados para leer la información.

Características clave:

- 1 **Separación de responsabilidades:** divide el modelo de dominio y la lógica de negocio en dos partes distintas: una para comandos (escritura) y otra para consultas (lectura).
- 2 **Flexibilidad en la escalabilidad:** permite optimizar de manera independiente las operaciones de lectura y escritura según las necesidades.
- 3 **Mejora del rendimiento:** la lectura y la escritura pueden escalar y optimizarse de forma independiente.
- 4 **Complejidad aumentada:** aumenta la complejidad de la arquitectura y el diseño del sistema.

Consideraciones:

CONSISTENCIA DE DATOS	COMPLEJIDAD DE LA ARQUITECTURA	CASO DE USO ADECUADO
Se debe manejar adecuadamente la consistencia eventual entre los sistemas de lectura y escritura.		

CONSISTENCIA DE DATOS	COMPLEJIDAD DE LA ARQUITECTURA	CASO DE USO ADECUADO
CQRS puede aumentar la complejidad del diseño y mantenimiento del sistema.		

CONSISTENCIA DE DATOS	COMPLEJIDAD DE LA ARQUITECTURA	CASO DE USO ADECUADO
Es más beneficioso en sistemas con clara disparidad entre la cantidad de operaciones de lectura y escritura y donde la escalabilidad y el rendimiento son críticos.		

Patrón Pub/Sub (publicador/suscriptor)

Pub/Sub (publicador/suscriptor) es un patrón de diseño de mensajería ampliamente utilizado en sistemas distribuidos para comunicación asíncrona entre diferentes partes de una app.

En este patrón, los emisores de mensajes (publicadores) no envían mensajes directamente a los receptores (suscriptores), sino que publican los mensajes en un canal o *topic*, y los suscriptores reciben estos mensajes de forma asíncrona al estar suscritos a esos canales o *topics*.

Características clave:

- 1 **Desacoplamiento:** los publicadores y suscriptores no necesitan conocerse entre sí, lo que facilita el desacoplamiento de los componentes del sistema.
- 2 **Escalabilidad:** permite escalar publicadores y suscriptores de manera independiente.
- 3 **Comunicación asíncrona:** mejora la eficiencia y el rendimiento al permitir que los procesos continúen su ejecución sin esperar respuestas.
- 4 **Flexibilidad en la distribución de mensajes:** los mensajes pueden ser distribuidos a múltiples suscriptores, lo que facilita la implementación de lógicas de difusión.

Consideraciones:

GESTIÓN DE SUSCRIPCIONES	RESILIENCIA Y TOLERANCIA A FALLOS	SEGURIDAD
Es importante gestionar adecuadamente las suscripciones para evitar la sobrecarga de mensajes y asegurar que los suscriptores reciban solo la información relevante.		

GESTIÓN DE SUSCRIPCIONES	RESILIENCIA Y TOLERANCIA A FALLOS	SEGURIDAD
Implementar mecanismos para manejar la desconexión temporal de suscriptores o la pérdida de mensajes.		

GESTIÓN DE SUSCRIPCIONES	RESILIENCIA Y TOLERANCIA A FALLOS	SEGURIDAD
Asegurar que los mensajes publicados no expongan datos sensibles y que los canales de comunicación estén seguros.		

Conclusiones



En este fastbook, hemos hecho un repaso a la arquitectura orientada a eventos (EDA), así como a los mecanismos para la gestión de dichos eventos con dos de los sistemas gestores de colas más usados en el mercado: Kafka y RabbitMQ.

Y, además, hemos analizado algunos patrones para aplicar buenas prácticas en tu día a día, que te servirán de gran utilidad.

Pues bien, ahora que ya conoces lo importante que es la arquitectura basada en eventos dentro del proceso de desarrollo software, es el momento de ponerse manos a la obra y poner en práctica todos estos conocimientos. ¡A por ello!

¡Enhорabuena! Fastbook superado



Qualentum.com