Kafka: streaming data

Motivazioni

Negli ultimi anni l'avvento delle architetture a microservizi ha portato la necessità di studiare nuove soluzioni al problema della gestione di molteplici fonti di dati.

In sistemi complessi formati da più microservizi tanti componenti interdipendenti comunicano tra loro scambiandosi dati e attingendo da numerose fonti di dati comuni come database, data warehouses oppure servizi esterni.

La necessità di filtrare, standardizzare e gestire molte fonti di dati aveva portato alla nascita del processo di **Extract, Transform, Load** (ETL) per l'estrazione, trasformazione e caricamento di dati in sistemi di sintesi come data warehouse o data mart, questo processo si sta però rivelando complicato ed impegnativo in un mondo dove la mole di dati prodotta dal logging di eventi critici ad un qualsiasi business è in continua crescita: semplici esempi sono la gestione degli eventi in un sistema **Internet of things** (IoT) oppure lo studio delle abitudini dei propri clienti per un servizio di e-commerce.

Lo stream processing tra microservizi propone un nuovo approccio per la gestione di questi problemi, fornendo una soluzione adatta alla gestione di dati in real-time altamente scalabile e ad alto throughput.

Apache Kafka è una piattaforma nata in un contesto aziendale importante che mira a rivoluzionare il modo con cui i microservizi di un business comunicano tra loro, favorendo un approccio improntato sulla gestione di eventi legati al comportamento dei dati, più che i dati in se.
Kafka nasce per sfruttare a pieno lo stream processing e favorire una gestione intelligente di grosse moli di dati, abbandonando il classico processo "batch" ETL per una soluzione, appunto, basata sullo streaming dei dati tra microservizi.

1. Introduzione: [Il ruolo dello streaming nelle architetture moderne] / [Storia] / [Confronto ETL-ES+stream]

Prima di poter discutere della architettura fornita da Apache Kafka, è necessario comprendere le differenze tra ETL e stream processing.
Per poter utilizzare pienamente stream processing, è inoltre necessario comprendere come la gestione di dati basata su inserimento, cancellazione e modifica da database, può essere modellata come una serie di eventi e quali sono i vantaggi di un approccio alla gestione di questi dati basato su **event sourcing** (ES).

1.1 ETL

Cos'è un processo di etl

Un processo di Extract, Transform, Load (ETL) è un processo mirato alla trasformazione di dati contenuti su più database per ottenere un nuovo insieme di dati, filtrato e traformato secondo una particolare logica, destinato ad essere salvato in una data warehouse.
Verso la fine degli anni '70 molte aziende iniziarono ad utilizzare molteplici database per salvare e gestire informazioni, è proprio in questo contesto che nascono i processi di ETL: con l'avanzare del tempo è stato necessario studiare un metodo per l'aggregazione e gestione delle varie fonti di dati.

Un qualsiasi processo di ETL si compone di tre parti:

Extract

E' la prima parte del processo di ETL ed involve l'estrazione dei dati da più data sources come database relazionali o non-relazionali, file JSON od XML ma anche risorse "adhoc" come, ad esempio, dei dati generati da programmi di web analytics.

L'obiettivo di questa fase è estrarre tutti i dati necessari dalle possibili sorgenti e prepararli alla fase di Transform.

Un importante problema legato a questa fase è il processo di **validazione delle sorgenti dati**: con più sorgenti dati spesso ci si ritrova a dover gestire più *formati* non necessariamente compatibili tra loro.
Per poter garantire alla fase di Transform dei dati comprensibili, durante la fase di Extract vengono definite delle *regole di validazione* per filtrare i dati provenienti dalle varie sorgenti, un esempio di regola di validazione è il controllo dei tipi di dati presenti nella fonte.

Transform

Nella fase di Transform una serie di regole e funzioni vengono applicate ai dati generati dalla fase di Extract per prepararli alla fase di Load nella data warehouse.

Il primo compito della fase di Transform è la **pulizia dei dati**: spesso le varie fonti di dati, nonostante siano state validate, possono presentare incongruenze tra loro come caratteri speciali legati all'encoding della propria sorgente oppure formati dei dati diversi ma compatibili (un esempio può essere la differneza di formattazione tra date americane ed europee).
Per garantire un corretto funzionamento delle operazioni di trasformazione è quindi necessario pulire i dati ed adattarli ad un formato comune.

Il secondo compito della fase di Transform è la **trasformazione dei dati** in nuovi dati richiesti dal business, esempi di trasformazioni sono: * Joining di tabelle da più sorgenti * Mapping e trasformazione di dati (esempio: "Maschio" in "M") * Aggregazione di dati * Generazione/calcolo di nuovi dati * Selezione di insiemi di dati * Validazione del nuovo formato di dati prodotto

Load

Nella fase di Load l'insieme di dati generati dalla fase di Transform vengono inseriti in un target, il quale potrebbe essere una data warehouse ma anche più semplicemente un file in un formato utile.
Business diversi hanno necessità diverse, per questo l'implementazione della fase di load può avere più modalità implementative, il punto focale di questa fase è proprio stabilire la frequenza e le modalità di aggiornamento dei dati presenti nel target.
Decidere la frequenza (giornaliera, mensile, ecc.) e le modalità (sovrascrizione dei vecchi dati o meno) del target possono portare ad un processo di ETL più o meno utile ad una azienda.

Per generare un buon target è buona norma definire uno schema *preciso e chiaro* della tipologia di dati a cui il target deve aderire.
Come detto in precedenza un processo di ETL è utilizzato per aggregare più fonti di informazioni comuni ad un processo aziendale, questo suppone che le informazioni presenti nella data warehouse potrebbero venire usate da più parti di una azienda, le quali potrebbero essere abituate a particolari formati dei dati. Senza definire uno schema dei dati chiaro e preciso, si correrebbe il rischio di generare un insieme di dati inutilizzabile da determinati reparti in quanto non conforme al formato di dati da loro conosciuto.

1.2 L'importanza dei dati e degli eventi {piccola introduzione per ES}

Lo status quo delle moderne applicazioni web è basato sul utilizzo di database per rappresentare le specifiche di dominio, spesso espresse da un cliente e/o da un esperto del dominio esterno all'ambiente di sviluppo.

Durante la fase di analisi dei requisiti (supponendo un modello di sviluppo del software agile) cliente e team di sviluppo si confrontano, cercando di trovare un linguaggio comune per definire la logica e l'utilizzo del software richiesto; Una volta stabiliti i requisti, il team di sviluppo generalmente inizia uno studio interno atto a produrre un **modello dei dati** che verrà usato come base per definire lo schema dei database utilizzati dal sistema.
Un cliente comune molto spesso non ha padronanza del concetto di 'stato di una applicazione', ma piuttosto si limita ad esporre i propri requisiti descrivendo i possibili **eventi** che, traslati sul modello di sviluppo incentrato su i database, portano il team di sviluppo a ragionare sui possibili stati di un database in risposta a questi eventi.

Lo stato di un database di una applicazione è strettamente legato all'insieme degli eventi del dominio applicativo; L'unico modo per modificare o interagire con questo database è tramite i comandi di inserimento, cancellazione o lettura, tutti comandi che vengono eseguiti solamente all'avvenire di un particolare evento.

Un database mantiene solo lo stato corrente di una applicazione; Non esiste il concetto di cronologia del database a meno dell'utilizzo dei pattern di **Change**

Data Capture (CDC), che generalmente sono utilizzati per generare un transactional log contenete tutte le operazioni eseguite sul suddetto database.
In questo modello database-driven, un evento genera un cambiamento su una base di dati; Gli eventi e lo stato di un database sono però concetti diversi e slegati tra loro, l'esecuzione di un evento a volte può portare ad una asincronia tra l'esecuzione di un evento e lo stato di un database, tanto più se questo database è utilizzato da tutti i microservizi di una applicazione.

Una soluzione al problema di più microservizi che utilizzano lo stesso database è di utilizzare delle views del database locali ad ogni microservizio: ogni servizio lavorerà su una copia locale del database ed un job esterno si occuperà di compattare le views e mantenere il database aggiornato rispetto a tutti i cambiamenti.
Questa soluzione ha un enorme problema: supponiamo di notare un errore sul database e di doverlo correggere, come possiamo decidere quale delle views ha "più ragione" delle altre? Per aiutarci nella ricerca dell'errore potremmo utilizzare il transactional log di ogni views, ma su database di grandezze importanti potrebbe essere un problema non indifferente.

Event sourcing propone di risolvere questo genere di problemi allontanandosi da una progettazione database-driven e basata sul pattern di richiesta/risposta elevando gli eventi a elementi chiavi del modello dei dati di una applicazione.

1.3. Event sourcing

> Event Sourcing ensures that all changes to application state are stored as a sequence of events. Not just can we query these events, we can also use the event log to reconstruct past states, and as a foundation to automatically adjust the state to cope with retroactive changes.

Event sourcing è un pattern basato su due fondamenti:
* Ogni cambiamento di stato del mondo^ è da vedersi come un evento; Ogni evento deve essere salvato
* Tutti gli eventi devono essere salvati in sequenza, seguendo l'ordine in cui sono avvenuti

> Esempio di utilizzo di event sourcing => come faccio? è giusto copiare l'esempio che c'è già su un sito in modo da sfruttarne le figure?
>
> Problemi risolti da event sourcing => ho un log di cambiamenti, semplicità nella gestione dei cambiamenti
>
> Problemi creati da event sourcing
> => può essere complicato eseguire query del tipo "voglio tutti gli ordini con valore >50€" in quanto richiede la generazione dello stato

corrente del mondo, ovvero la lettura ed esecuzione di tutto il log
-> risolvibile creando views parallele al log

1.4 Stream Processing

Descrivere le caratteristiche principali di un sistema di streaming : distribuito, append only, etc.

Differenza tra Message Brokers e transactional append only logs: Rabbit MQ vs Kafka

1.5. Svantaggi di un processo ETL: perchè favorire stream processing ed event sourcing

Introdurre il concetto di schema

Perchè la gestione dei dati è importante nelle architetture a microservizi

2. Apache Kafka e l'ecosistema

2.1 Cos'è - come funziona: Log

Struttura di base: log => log compaction

Struttura architetturale: * brokers * clusters * topic

Come collegare event sourcing e kafka => perchè kafka è una buona piattaforma per event sourcing

2.2 Kafka Connect

Schema

Source connectors

Sink connectors

Community involment

2.3 Kafka Streams

cos'è streams

KSQL/LSQL

3. Esempi di utilizzo di Kafka

4. Bibliografia

https://martinfowler.com/eaaDev/EventSourcing.html
https://www.confluent.io/blog/data-dichotomy-rethinking-the-way-we-treat-data-and-services/
https://www.confluent.io/blog/build-services-backbone-events/
https://www.confluent.io/blog/apache-kafka-for-service-architectures/
https://www.confluent.io/blog/messaging-single-source-truth/
https://www.confluent.io/blog/building-a-microservices-ecosystem-with-kafka-streams-and-ksql/
https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka
https://qconsf.com/sf2016/system/files/keynotes-slides/etl_is_dead_long-live_streams.pdf <= https://www.youtube.com/watch?v=I32hmY4diFY