Exercise 3

We would like you to write a design proposal for how you might design and implement a data infrastructure for storing and representing the following data sources, given the desired use cases. You do not need to describe how the use cases are implemented - just describe how you might get the data to the consumer in a reasonable way and what the components of the proposed architecture could be (e.g. components technical capabilities, main tables, etc.).

Starting with two data sources:

1. **data source A** delivers data every day, with a 3-day delay
2. **data source B** delivers data every 14 days

Data sources properties:

- both sources deliver data via HTTP or FTP as a direct file download.
- both sources use different formats, so data model from A cannot be used to denote data from source B.

Data source C delivers either the current weather or a 7-day forecast
Data sources properties:

- this source delivers data via a JSON REST API
- the data can be retrieved using two endpoints - one for the current weather and one for the forecast
- the forecast endpoint takes two parameters: latitude and longitude
- the current endpoint takes three parameters: latitude, longitude and optionally a timestamp one market data source

Data source D delivers near real-time market data, available every 15 seconds

- this source delivers data via a JSON API for streaming (e.g. like the Twitter streaming API) each data set includes a coordinate with a latitude and longitude, where each coordinate represents a 50-mile market radius (also called "region").

Use cases:

- Using a REST API, I want to get the average weather, average market interest and accompanying data for a region
- I want to get all daily weather data and daily market prices of the past 15 years and store the data in appropriate data structures
- I want to get data for a single region of the past 15 years, run analyses on it and store the results for consumption by a REST API

Bonus challenge:
As a customer, I want to upload proprietary data Z, and then run custom analyses on data Z and the data from data sources A and B and access it through the REST API.

Soluzione

Assunzioni

- sorgenti A e B:
    - si suppone che i file sorgenti siano sempre presenti ai link invocabili tramite HTTP o FTP
    - non viene definita la tipologia di dato, ma sicuramente è stabilito in una sorta di 'contratto' il modello dati per le due sorgenti distinte (modello dati di A <> modello dati di B)

Progettazione e implementazione

Sorgente A

La sorgente A mette a disposizione i dati ogni giorno ma con 3 giorni di delay. Questo tipo di informazione deve essere considerato nella messa a punto dell'integrazione per il recupero dei dati.
In particolare dovrà essere gestito lo sfasamento rispetto all'istante attuale nel momento dell'integrazione ed elaborazione dei dati, considerando che la data di business associata al dato non è quella del giorno in cui il dato viene messo a disposizione.
Per rendere fruibile il dato, sarà opportuno mettere a punto un data model all'interno di un'istanza di database in modo che con un tool di Data Integration (e.g. Talend Data Integration, Informatica Power center, AWS Pipeline o altri tool/framework o, ancora, script custom) sia possibile estrarre i dati tramite protocollo FTP/HTTP dalla sorgente e gestirne "l'atterraggio" in un primo sistema.
L'elaborazione potrebbe avvenire online (durante l'estrazione del file) o offline (in un secondo momento).
Nel primo caso, il file verrà immagazzinato in uno storage (che può essere on-prem o cloud a seconda della data platform aziendale (e.g. Cloud: S3, on-prem: External network storage) e subito utilizzato per elaborazione tramite pipeline di etl.
Dopo la manipolazione i dati potranno essere inseriti nelle relative tabelle associate al data model della sorgente.

Nel secondo caso, invece, i file potranno essere depositati sempre in un tool di storage e successivamente può essere schedulata in differita una pipeline che in modalità batch elabori tutti i file presenti o i file 'depositati' dopo una certa data/ora.

La differenza tra i due scenari risiede nel livello di accoppiamento della soluzione: nel primo caso molto elevato, in quanto l'elaborazione avviene praticamente in contemporanea con l'estrazione del dato da sorgente; mentre nel secondo caso l'accoppiamento è molto più debole e le due operazioni potrebbero essere orchestrate.

Sia l'estrazione del dato che l'elaborazione andranno schedulati in ogni caso con cadenza giornaliera con tool di coordinamento/amministrazione integrati a quelli di integrazione (e.g. Talend Administration Center) oppure di terze parti (e.g. Apache Airflow).

Sorgente B

Le considerazioni sono analoghe a quelle della sorgente A.
Le modifiche riguardano chiaramente la destinazione (il data model non è lo stesso di A), la pipeline implementata (necessario un job differente per gestire la diversa sorgente e le diverse trasformazioni) e la schedulazione (una volta ogni 14 giorni invece che una volta al giorno).

Per il resto tutte le considerazioni fatte sulla sorgente A rimangono valide.

Sorgente A + B

Per entrambe le sorgenti, una volta inseriti i dati in un database/dwh (può essere anche lo stesso, gestendo i dati tra schemi diversi o comunque riferiti a sezioni di data model differenti) i dati possono essere utilizzati direttamente per interrogazione via SQL e/o reportistica, oppure, in caso di dati grezzi, consolidati in un layer superiore all'interno del database e resi fruibili ad eventuali utenti che li necessitano.

La seconda soluzione è preferibile, in quanto permette di mantenere il dato come da sorgente, andando ad elaborarlo poi successivamente: si è soliti avere un layer che fornisca da replica della sorgente da cui i dati vengono estratti.

Questo tipo di soluzione descritta - da servizio sorgente a database con job etl -, tuttavia, genera un'altra accoppiabilità tra sorgente e destinazione ed è poco robusto nel caso si presentasse un secondo utilizzatore dei dati provenienti dalla sorgente A o dalla sorgente B.
In quel caso avrebbe senso disaccoppiare sorgente/destinazione inserendo nel mezzo un data bus, per far sì che non si debbano replicare inutilmente dati in diversi punti dell'architettura e che gli stessi dati possano essere fruibili da più consumatori.

Sorgente C

In questo caso può essere fatta una scelta di integrazione a seconda dello use case necessario.

Se lo scopo di utilizzo dei dati è la Business Intelligence, allora è utile adottare un'integrazione simile alle prime due sorgenti, cambiando ovviamente il modello dati, definendo quale endpoint utilizzare (se per il meteo corrente o per le previsioni su 7 giorni), integrando ed elaborando le informazioni per storicizzarle.
Ad esempio può essere definito un data model per le previsioni del meteo correnti per una certa coppia lat/long in cui l'informazione sulle previsioni è il fatto principale di un eventuale dwh e assieme ad esso vengono definite una serie di dimensioni popolate al contorno relative alla geografia e ai possibili stati metereologici, etc.

Un'altra possibilità invece è la messa a disposizione dei dati a terzi.
In questo caso, può essere definito un layer di disaccoppiamento con una tecnologia che funga da gateway (e.g. AWS API Gateway, Akana, Kong, etc.) e gestisca le invocazioni provenienti da terzi per estrarre i dati dalla sorgente e renderli disponibili.

In questo caso possono essere definiti ad esempio due path da invocare a livello di gateway, uno per le previsioni correnti e l'altro per le previsioni sui 7 giorni in base ai parametri forniti in ingresso.
Una volta giunta l'invocazione, l'API gateway può fare semplicemente da proxy inoltrando la richiesta alla sorgente oppure gestire la scalabilità della soluzione attraverso altri servizi cloud come AWS Lambda, che diventeranno quindi responsabili dell'invocazione della sorgente 'reale' e restituiranno il dato corretto.
L'API gateway diventa anche il responsabile di autenticazione e autorizzazione (quest'ultima può essere demandata alla Lambda invocata).

Sorgente D

Possono essere riprese le considerazioni fatte per la sorgente C a seconda dello use case.

In aggiunta, essendo un dato fornito in near real time può essere gestita un'integrazione che preveda una parte di estrazione del dato ad alta frequenza e il conseguente popolamento di un topic o di una coda (a seconda dei servizi e dei tool disponibili[1]) in modo tale da gestire in autonomia il connettore alla sorgente vera e propria.
In questo caso, tuttavia, è necessario tener conto di un dimensionamento della piattaforma di integrazione, poichè se lo use case richiedesse un'ampia customizzazione (ad esempio elevata cardinalità di coppia lat/long richieste) implementare una piattaforma in grado di gestirle tutte diventerebbe complesso e costoso.

Dovrà essere poi gestita, come descritto per la sorgente C, la fruizione del dato a terza parti.

Use cases

1. Using a REST API, I want to get the average weather, average market interest and accompanying data for a region

[1] ogni tool ha i suoi pro e i suoi contro: una coda semplice non consente di memorizzare il dato, mentre un topic di una tecnologia come Kafka, ad esempio consente di gestirne la storicizzazione e rende il dato disponibile nel tempo

In questo caso si ipotizza di utilizzare un'integrazione diretta con il servizio a cui viene anteposto un tool tipo API Gateway.

Il backend (e.g. Lambda function o qualsiasi tool ETL) recupera le informazioni dalle entità definite in cui sono stati immagazzinati i dati elaborandoli in real-time (l'elaborazione può anche essere realizzata in precedenza in caso lo use caso sia richiesto da diversi consumatori).

Altrimenti, senza strutture dati memorizzate, l'elaborazione può essere fatta esplicitando un certo periodo temporale e quindi calcolando tutto on-line (in corso di esecuzione).
Questa pratica ha il drawback di utilizzare un elevato costo computazionale e potrebbe non fornire risposte sufficientemente celeri agli utilizzatori finali

2. I want to get all daily weather data and daily market prices of the past 15 years and store the data in appropriate data structures

Può essere ripreso il concetto dello use case precedente considerando i dati come immagazzinati in strutture stabili (dwh/database).
Quindi una volta interrogati i servizi delle sorgenti originali i dati vengono elaborati e immagazzinati attraverso un processo di ETL/ELT all'interno di uno storage primario.
Qui viene tenuta la storicizzazione e i dati possono essere elaborati a seconda di necessità degli utenti finali (e.g. calcolando kpi/metriche per la reportistica, aggregando in base a requisiti specifici, etc.).

3. I want to get data for a single region of the past 15 years, run analyses on it and store the results for consumption by a REST API

Viene creato un servizio, che verrà utilizzato per accedere ai dati elaborati in precedenza.
Il servizio viene esposto via API gateway o viene costruito un front-end anteposto ad esso (abbastanza inutile in questo caso, in quanto il servizio è singolo).
Quando il servizio viene invocato, si attiva un processo ETL che è incaricato di recuperare i dati elaborati in precedenza ma solo per una specifica regione.

Ovviamente le strutture in cui i dati vengono immagazzinati in precedenza possono essere ottimizzate a livello fisico (manipolazione delle proiezioni) e logico (partizioni, ordinamento, segmentazione, a seconda dell'engine utilizzato).

Bonus challenge

Viene reso disponibile un servizio o una webapp dove il customer può caricare dei dati (in un formato condiviso) proprietari perchè vengano inseriti in uno storage e vengano poi processati insieme ai dati delle sorgenti A e B che devono a loro volta essere opportunamente immagazzinati all'interno di un database/dwh.

A partire dall'invocazione del servizio può essere usata poi un job di ETL per l'elaborazione e inserimento dei dati dalla sorgente Z al database e successivamente un secondo job di ETL (coordinato o orchestrato) in modo tale che esegua l'analisi aggregando i dati delle varie sorgenti.

Il risultato può essere poi messo a disposizione con un secondo servizio che nel backend si occupi di interrogare i dati elaborati.

Un esempio di servizio in grado di coordinare tutto il processo sono le step function di aws che invocando opportune Lambda, sono in grado di orchestrare la trasformazione e inserimento dei dati della sorgente Z e la successiva elaborazione dei dati inseriti.