Homework 2 - Prova in itinere DSBD 2024/25

Matteo Santanocito - 1000069999

December 20, 2024

Contents

1 Abstract

L'obiettivo di questo secondo Homework è quello di introdurre una nuova modalità di interazione con il database, adottando il **pattern CQRS** e ampliando l'architettura già esistente. Inoltre, HW2 rende disponibile una funzione di **notifica asincrona** in grado di avvisare l'utente via email quando il valore di un determinato ticker oltrepassa una soglia precedentemente impostata, sia verso l'alto (high-value) che verso il basso (low-value).

Ogni utente potrà stabilire una o entrambe le soglie, per poi modificarle in un secondo momento attraverso specifiche chiamate RPC. Le notifiche vengono gestite attraverso l'integrazione con Apache Kafka, grazie a una comunicazione costituita da un cluster di 3 broker. Per evitare lo spam eccessivo delle mail ad ogni minima variazione del ticker, il sistema memorizza lo stato precedente delle condizioni di notifica. Lo scopo di questa novità è quello di limitare la continua interazione con l'SMTP esterno migliorando l'efficienza del sistema.

2 Diagramma del sistema

Rispetto all'HW1 vengono introdotti 7 nuovi container che introducono le novità descritte nell'abstract: *kafka-admin*, *AlertSystem*, *AlertNotifier* (più sotto discuteremo nel dettaglio sulle funzionalità), ZooKeeper e gli altri per i 3 broker kafka (ogni broker kafka ha un container).

Perché dedicare un container all'admin-kafka?
Le funzionalità di amministrazione del cluster Kafka sono state isolate in un container dedicato, rispettando il principio di separazione delle responsabilità e distinguendo chiaramente la gestione del cluster dalle altre parti del sistema. La scelta di utilizzare un unico container, invece di suddividerlo ulteriormente, è motivata dalla stretta correlazione tra operazioni come la creazione dei topic e il recupero dei metadati, che condividono la stessa logica di interazione con il cluster, rendendo più semplice ed efficiente una gestione centralizzata.

Zookeper:
Zookeeper è un componente fondamentale che garantisce il coordinamento fra i broker Kafka, occupandosi della conservazione e distribuzione dei metadati di configurazione, nonché dell'individuazione del broker che funge da controller.

Pattern:
Nell'HW2 è stato introdotto il pattern "Command Query Responsibility Segregation", questa strategia ha l'obiettivo di separare i due aspetti fondamentali della gestione del sistema: tutti i command sono adesso inseriti all'interno del file "command_server.py" , mentre le query sono presenti nel file "query_server.py".

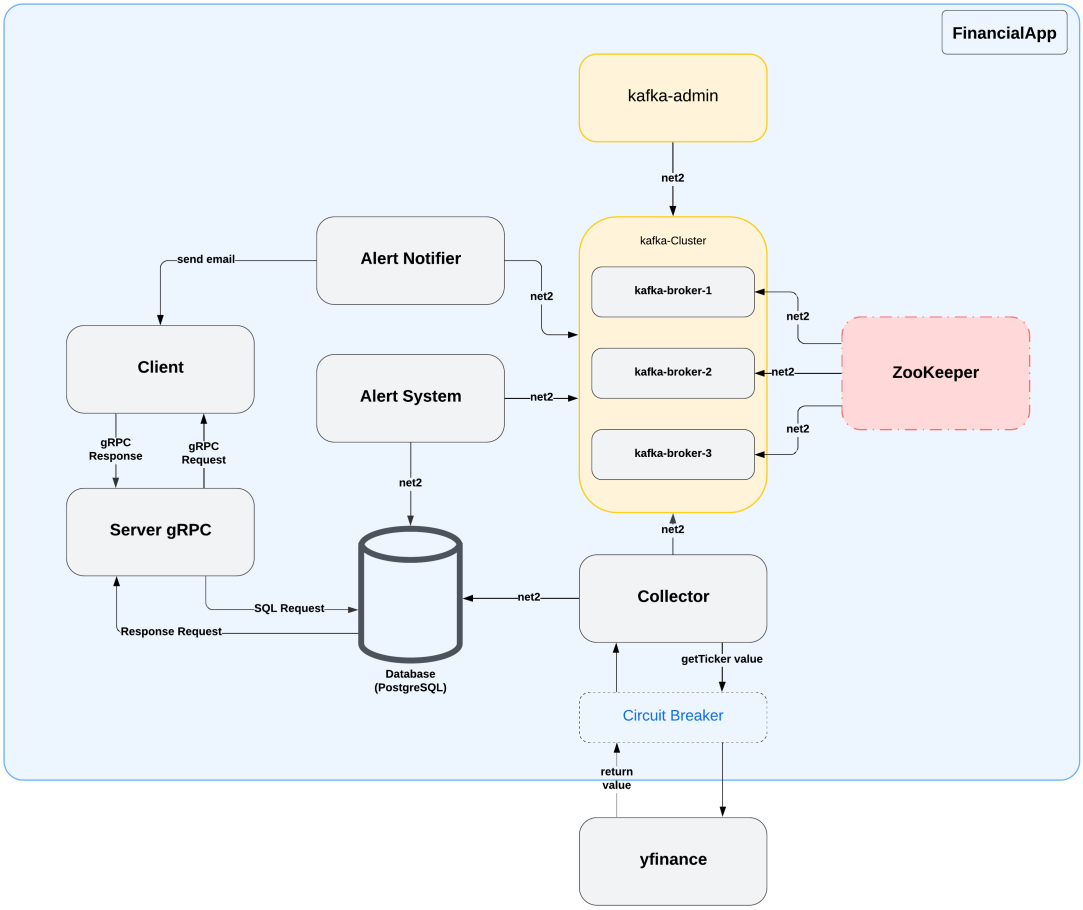

Figure 1: Diagramma del sistema

3 Dettagli delle novità introdotte nell'HW2

3.1 CQRS

Iniziamo trattando nel dettaglio le novità che porta l'introduzione del pattern CQRS. È un modello architetturale che separa le operazioni di scrittura (comandi) dalle operazioni di lettura (query) all'interno di un sistema. Questa separazione permette di ottimizzare e scalare indipendentemente le due tipologie di operazioni, migliorando la manutenibilità e le prestazioni complessive del sistema.

Il principio chiave di CQRS è che le operazioni di modifica dello stato del sistema (Comandi) e le operazioni di recupero dei dati (Query) devono essere gestite da modelli separati. Questo significa che la logica e le strutture dati utilizzate per gestire i comandi sono diverse da quelle utilizzate per le query. Infatti adesso sono presenti due classi "UserCommandService" e "UserQueryService":

- "*UserCommandService*": si occupa delle operazioni di scrittura e gestisce i controlli di validazione (come il controllo sulla sintassi dell'email in fase di registrazione). Il command rappresenta un'azione che deve essere eseguita, come aggiornare un profilo

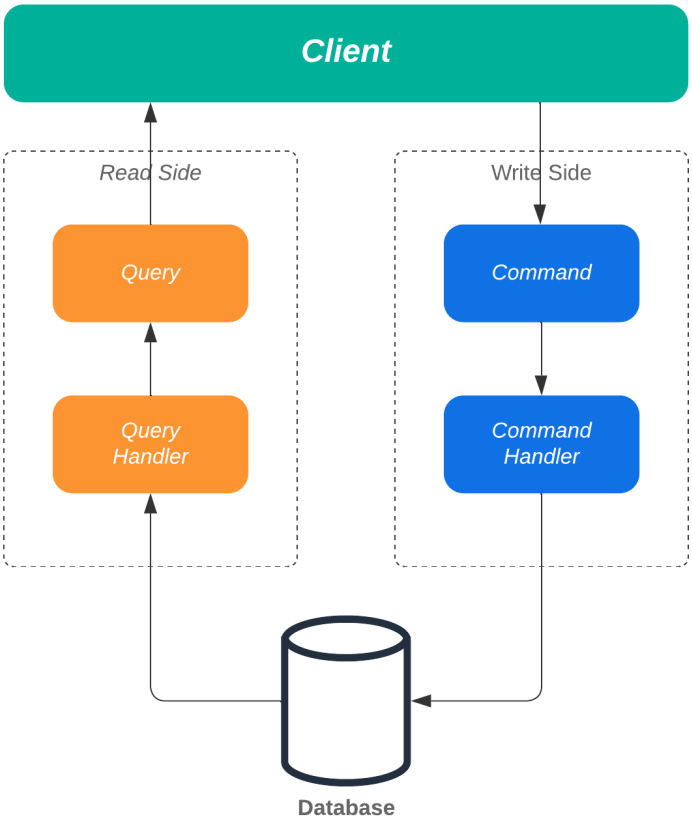

Figure 2: Diagramma del pattern CQRS

utente o eliminare un record.

- "*UserQueryService*": si occupa delle operazioni di lettura. La query è invece una richiesta che intende recuperare dati **senza modificarne lo stato**.

3.2 Implementazione Apache Kafka

3.2.1 KafkaAdmin

:
Il Kafka Admin ha il compito di gestire e monitorare il cluster Kafka. Di seguito spiego le sue funzionalità implementate in "cluster_metadata.py" e "topic_creation.py".

1. **Creazione di topic nel cluster Kafka topic_creation.py**
   Si occupa di controllare se il topic è già presente nel cluster utilizzando *KafkaConsumer.topics()*: se il topic non esiste, crea un nuovo Topic con il numero di partizioni e il fattore di replica specificati. Dopo aver fatto il controllo invoca *admin_client.create_topics()* per aggiungere il topic al cluster.

2. **Elenco e dettagli dei topic disponibili - topic_creation.py**
   Va a recuperare l'elenco dei topic disponibili grazie a *KafkaConsumer.topics()* e per ognuno dei topic, ottiene le partizioni associate grazie a *consumer.partitions_for_topic()*.

3. **Recupero dei metadati del cluster Kafka - cluster_metadata.py**
   Grazie alla chiamata alla funzione admin_client.list_topics() ottiene i metadati e per ogni topic registra i log con i dettagli delle partizioni.

4. **Monitoraggio periodico del cluster - cluster_metadata.py**
   Fa un controllo sui topic, verificando che siano presenti chiamando create_topic_if_not_exists() e list_topics_and_details(). Successivamente, in un ciclo infinito, chiama get_metadata() ogni 120 secondi per monitorare lo stato del cluster.

- **data_collector e alert_system:** Questi servizi utilizzano i topic creati e configurati dal Kafka Admin (to-alert-system, to-notifier).

- **alert_notifier:** Consuma messaggi dal topic to-notifier e invia notifiche, utilizzando il Kafka Admin per assicurarsi che il topic esista e sia configurato correttamente.

3.2.2 Data Collector

Il Data Collector è responsabile di inviare segnali al sistema per "svegliare" il consumer collegato al topic to-alert-system. Dopo aver completato il ciclo di raccolta, il Data Collector invia un messaggio al topic Kafka to-alert-system. Questo messaggio funge da segnale per "svegliare" il consumer sottoscritto a questo topic, tipicamente l'Alert System, per avviare ulteriori elaborazioni.

3.2.3 AlertSystem

L'Alert System è responsabile di monitorare i dati finanziari aggiornati per identificare condizioni che richiedono notifiche agli utenti. Questo componente, grazie all'integrazione con Kafka e il database, garantisce un'elaborazione efficiente e accurata delle soglie di allerta definite dagli utenti. Una volta elaborato il messaggio e completate tutte le verifiche, l'Alert System effettua un commit manuale dell'offset, garantendo che il messaggio non venga rielaborato.

- **Consumo del messaggio Kafka**: Si iscrive al topic *to-alert-system* utilizzando un consumer Kafka. Il messaggio ricevuto, pur non contenendo dati significativi, serve come segnale per avviare l'elaborazione. Una volta ricevuto il messaggio, controlla che la chiave del messaggio sia update per confermare che si tratta di un segnale valido.

- **Verifica delle soglie**: Se il valore del ticker supera la soglia massima (high_value) o scende al di sotto della soglia minima (low_value), il sistema genera un messaggio di notifica. I messaggi di notifica includono informazioni dettagliate, come: email dell'utente, ticker interessato, valore corrente del ticker, condizione di superamento della soglia (HIGH o LOW), valore della soglia corrispondente.

- **Produzione del messaggio Kafka**: Le notifiche vengono inviate al topic to-notifier utilizzando un producer Kafka. Questi messaggi saranno successivamente consumati dall'Alert Notifier, che si occupa di inviare le email agli utenti.

3.2.4 Parametri di configurazione

Il Data Collector e l'AlertSystem utilizzano un producer Kafka configurato con parametri ottimizzati per garantire affidabilità e coerenza temporale:

- **bootstrap.servers:** Specifica gli indirizzi dei broker Kafka. I broker sono configurati come: kafka-broker-1:9092, kafka-broker-2:9092, kafka-broker-3:9092. Questo consente al producer di connettersi al cluster Kafka distribuito.
- **acks:** Configurato su all, garantisce che il messaggio venga considerato inviato solo dopo che tutte le repliche sincronizzate (ISR) lo hanno ricevuto.
- **batch.size:** configurato a 500 byte, specifica la dimensione massima dei messaggi che possono essere accumulati in un batch prima di essere inviati.
- **compression.type**: Configurato su gzip, riduce la dimensione dei messaggi inviati, ottimizzando la larghezza di banda.
- **max.in.flight.requests.per.connection:** Limitato a 1, garantisce che i messaggi vengano inviati in ordine corretto.
- **retries**: Configurato su 3, consente al producer di tentare nuovamente l'invio dei messaggi in caso di errore, migliorando la resilienza.

3.2.5 AlertNotifier

Alert Notifier è un componente fondamentale del sistema per gestire le notifiche agli utenti in base ai messaggi ricevuti dal topic Kafka "to-notifier". Questo modulo garantisce che le notifiche via email siano inviate solo quando necessario, evitando ridondanze e ottimizzando l'efficienza attraverso l'uso di una cache.

L'Alert Notifier opera come **consumer** Kafka sul topic to-notifier. La sua funzione principale è ricevere messaggi che contengono dettagli relativi alle soglie superate (valore massimo o minimo di un ticker) e inviare notifiche via email agli utenti interessati.

Utilizza una cache per memorizzare l'ultima notifica inviata a ciascun utente. Infatti prima di inviare una nuova email, verifica se i dati attuali (valore del ticker, tipo di soglia e ticker) differiscono da quelli precedentemente inviati. Se non ci sono variazioni significative, l'email non viene inviata.

Se i dati differiscono, utilizza il modulo send_email_template per inviare una notifica via email. La notifica userà dei campi come come il ticker, il valore attuale, la soglia superata e il tipo di superamento (HIGH o LOW).

Dopo aver elaborato il messaggio e inviato l'email (se necessario), l'offset Kafka viene committato manualmente. Questo garantisce che i messaggi vengano elaborati esattamente una volta, evitando duplicazioni.

3.2.6 Parametri di configurazione

Il consumer Kafka è configurato per garantire affidabilità e controllo manuale sull'elaborazione dei messaggi:

- **bootstrap.servers:** Specifica gli indirizzi dei broker Kafka. I broker sono configurati come: kafka-broker-1:9092, kafka-broker-2:9092, kafka-broker-3:9092. Questo consente al producer di connettersi al cluster Kafka distribuito.
- **group.id**: impostato su alert_notifier_group, questo permette al consumer di lavorare in un gruppo dedicato per elaborare i messaggi.
- **auto.offset.reset**: impostato su earliest, consente di iniziare l'elaborazione dall'inizio del topic se non ci sono offset salvati.
- **enable.auto.commit**: Disabilitato (False) per gestire manualmente il commit degli offset dopo l'elaborazione del messaggio.