RELAZIONE SERVIZIO OPEN SKY NETWORK

HOMEWORK 2

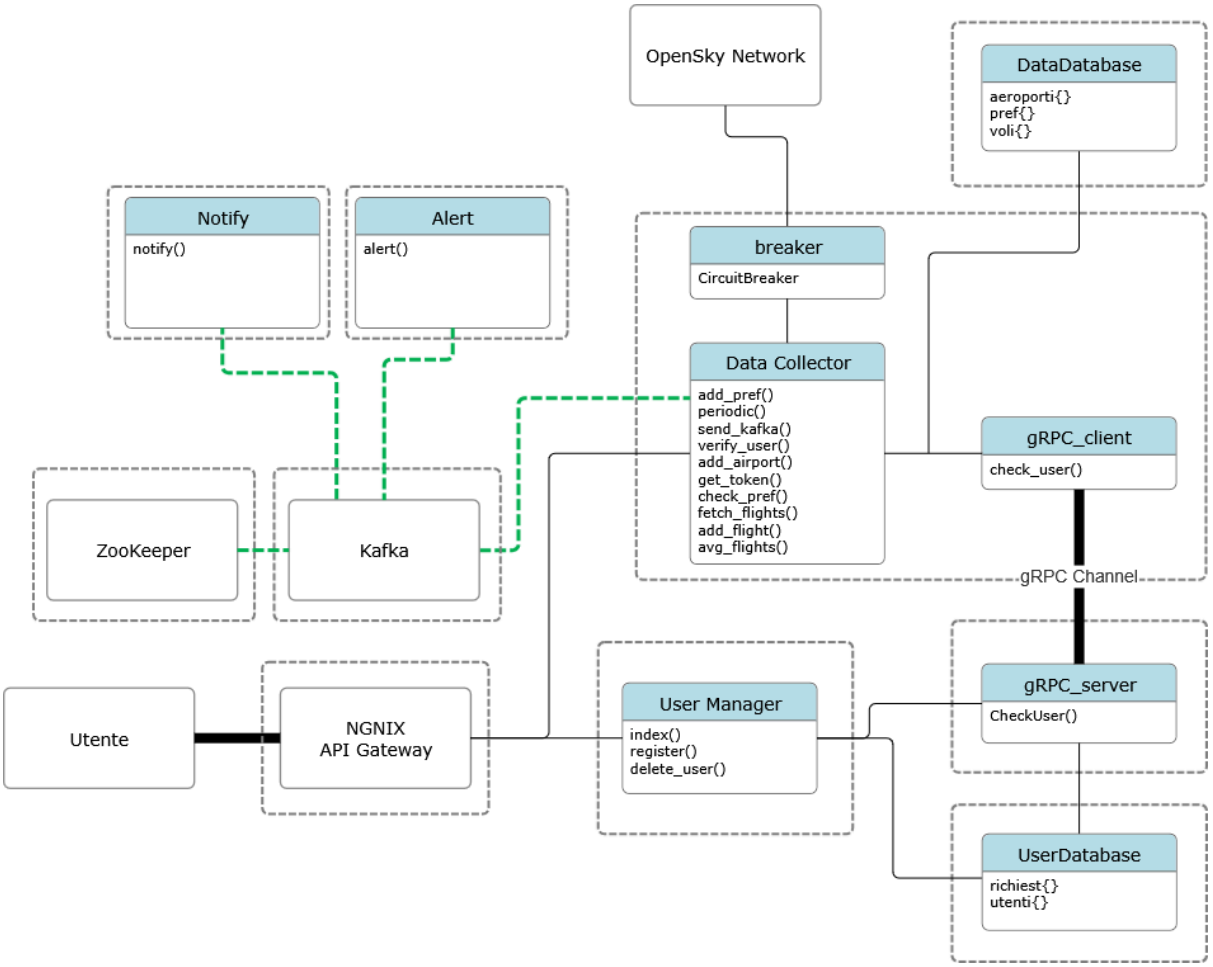
INTEGRAZIONE CON KAFKA, API GATEWAY & CIRCUIT BREAKER

Casaccio Agrippino 1000085471 **Rizzo Alessio 1000082581**

L'obiettivo è stato ampliare il sistema dockerizzato a micro-servizi precedente, estendone le funzionalità.

Per poterlo realizzare sono stati sviluppati ulteriori 7 servizi, sempre containerizzati e orchestrati tramite un Docker Compose. A livello implementativo sono state utilizzate le seguenti tecnologie: *confluent-kafka* per implementare Kafka a livello di codice, *nginx* come API Gateway, *smtp* per il servizio di email.

Di seguito una rappresentazione del sistema.



AGGIORNAMENTO DEI SERVIZI PRECEDENTI

1. Data Collector & Data Database

Il servizio relativo ai dati è stato modificato in modo tale che l'utente, all'atto della registrazione dell'aeroporto inserisca dei valori opzionali: low e high.
Questi numeri indicano quanti aerei al minimo e/o massimo l'utente voglia tener traccia per l'aeroporto. Viene dunque inserito un controllo per verificare che sia presente, e in caso contrario viene posto di default il valore 0, e che high sia effettivamente maggiore di low.
Dopo la verifica i dati saranno aggiunti nel database nella tabella relativa alle preferenze, dato che per uno stesso luogo possono variare per gli utenti. Ogni volta poi che verrà invocata l'API per connettersi a OpenSky Network, verranno prelevati e inviati tramite messaggio Kafka al topic *to-alert-system*, insieme alla mail dell'utente, il codice dell'aeroporto, la specifica se si tratta di voli in arrivo/partenza, i due valori e il conto dei voli trovati, al sistema di Alert che verificherà il superamento o no della soglia.

Oltre al Collector, di conseguenza è stato modificato anche il database con l'aggiunta delle informazioni descritte in precedenza.

Un altra aggiunta al Data Collector, e in particolare la comunicazione con l'API di OpenSky Network, è stata l'implementazione del pattern Circuit Breaker: per avere affidabilità, nel caso di molte richieste al servizio fallite o se vi sono ritardi, le successive vengono bloccate per sicurezza e risparmio e si aspetta un recupero del servizio.

Sono stati implementati in un file dedicato "*breaker*" i classici stati fondamentali: *CLOSED* (chiamate normali), ma a seguito di un fail_count maggiore di 3, scelto per avere alta sensibilità e individuare subito eventuali problemi, cambia; *OPEN* (chiamate bloccate) sollevata un'eccezione, terminando la richiesta e attendendo il tempo di recupero, questo è di 60s; *HALF-OPEN* (chiamate di prova).

SVILUPPO NUOVI SERVIZI

I **nuovi** micro-servizi sviluppati sono stati i seguenti:

1. NGINX

NGINX è il servizio che è stato utilizzato come API Gateway e Reverse Proxy, in questo modo è possibile centralizzare la gestione del traffico.
Sono stati definiti due blocchi per upstream, *user_service* e *data_service* che specificano le porte corrispondenti (5000 e 5001), applicano un limite di richieste (5 al secondo per evitare intasamenti) ed eseguono Health Check: se il servizio non risponde entro 30s viene escluso.
Sono state configurate tre location principali: *"/UserManager/"* e *"/DataCollector/"* che fanno riferimento ai rispettivi microservizi e la *"/"* di default e test per tutte le altre la quale ritorna semplicemente un messaggio di "Endpoint non trovato".

2. Alert

Alert fà riferimento a un servizio che funge sia da Consumer che da Producer Kafka:
Si tratta del consumer relativo al topic *to-alert-system*. Esso riceve il messaggio dal Data Collector e verifica i dati ricevuti, in particolare i valori low e high. Se essi sono presenti e risultano rispettivamente o minori o maggiori di count, ossia del numero di voli, allora essa sarà istanziata una logica di Trigger come una variabile di valore LOW o HIGH.
Sulla base di questo nuovo valore si attiverà o meno il "servizio" di Producer. Se il valore di trigger è superato, verrà prodotto, analogamente al Data Collector, un messaggio su un altro topic *to-notifier* verso il servizio omologo. Il messaggio contiene soltanto tre valori: l'email e aeroporto monitorato, ricavati dall alert ricevuto e il trigger.
In ogni caso i messaggi verranno committati sia per garantire la semantica at-least-once, sia per evitare side effect.

3. Notify

Notify fà riferimento a un servizio, stavolta solo come Consumer Kafka:
Si tratta del consumer relativo al topic *to-notify*. Esso riceve il messaggio dall'Alert e in base a i dati ricevuti si occupa di avvisare l'utente del superamento della soglia.
Ricevuti i dati di email, codice aeroporto e tipo di soglia, crea una email avente come destinatario l'email dell'utente e come corpo della stessa una frase del tipo:
"Alert per l'aeroporto XXX. La soglia YYY è stata superata."
Per poter effettivamente inviarla, avente come mittente "*notifyvoli@example.com*", è stato sfruttato il modulo di Python smtplib, che sfrutta il protocollo SMTP ed è molto semplice da implementare.

4. KAFKA

Kafka è il message-broker ed è usato dai microservizi per scambiarsi i dati in modo affidabile e offre disaccoppiamento. In questo caso sono stati configurati due topic, creati automaticamente quando un Producer tenta di scriverci per la prima volta: *to-alert-system* per connettere DataCollector e Alert, rispettivamente come Consumer e Producer; *to-notify* invece per connettere Alert e Notify. La replicazione dei messaggi è stata impostata a 1 dato che si utilizza un singolo broker per semplicità dato che il sistema non è di dimensioni elevate.
Inoltre è presente un healthcheck per verificare che Kafka sia avviato, operativo e in grado di comunicare con ZooKeeper. In tal modo sia Producer sia Consumer si possono connettere al broker in piena funzionalità.

5. ZooKeeper

ZooKeeper è un servizio di coordinamento utilizzato da Kafka. Esso funge e viene utilizzato per gestire il broker Kafka, monitorare lo stato di salute e fornire i metadata sui topic. Anch'esso presenta un healthcheck di verifica. Se il servizio è funzionante, allora Kafka, che ne è dipendente, si avvierà e proverà a connettersi. In tal modo è garantito che ZooKeeper sia attivo e non vi siano errori.
Il servizio nelle versioni più aggiornate di Kafka non è più necessario, dato che introduce dipendenze, ma per non avere problemi con il server Kafka, è stato scelto di usarlo per avere garanzia di funzionamento.

6. Smtp-Relay

Per poter inviare le email al superamento di una soglia si è scelto di utilizzare smtp (Simple Mail Transfer Protocol) per la sua semplicità di implementazione, oltre a non richiedere credenziali per poterlo utilizzare. Come microservizio è presente un smtp-relay, un server intermedio utilizzato per inoltrare le email create dal Notify, reindirizzando tra i vari smtp server fino al destinatario. Questo server è come standard esposto sulla porta 2525.

ULTERIORI CONSIDERAZIONI

I servizi principali user_manag e data_collector vengono avviati dopo i rispettivi database (il data_collector anche al server_gRPC), dato che ne dipendono, allo stesso modo sia il data_collector che alert e notify attendono che il servizio kafka sia attivo e funzionante. Ognuno dei 3 microservizi infatti prevede nel file docker-compose una specifica nelle dipendenze “condition: service_healty” così

Per le reti è stata aggiunta una denominata app_kafka_network per connettere il broker Kafka e i tre servizi usufruenti. In aggiunta è stato aggiunto per buona parte degli script la libreria logging: si tratta di uno strumento usato in fase di test per verificare effettivamente che tutto vada per il meglio, inserendo dei messaggi di log visualizzabili su un file.

Per poterlo eseguire è necessario inserire le credenziali di OpenSky Network sottoforma di file denominato configuration.json nella directory DataCollector. L’esecuzione è avvenuta mediante riga di comando “docker compose up” nella root principale e testata utilizzando il terminale stesso e l’applicazione Postman.