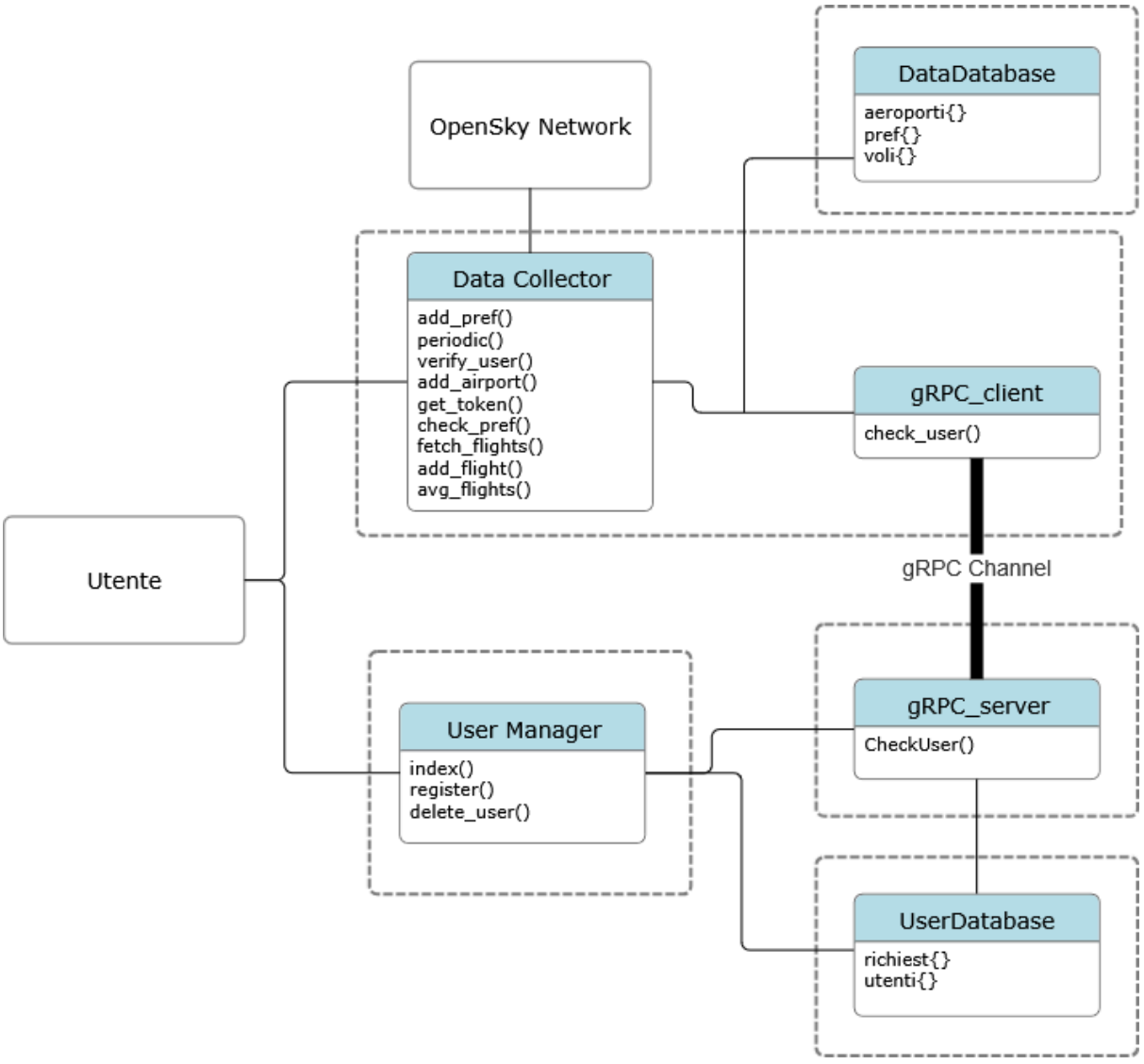
RELAZIONE SERVIZIO OPEN SKY NETWORK

CasaccioAgrippino 1000085471Rizzo Alessio 1000082581

L'obiettivo è stato progettare un sistema dockerizzato a micro-servizi per la gestione degli utenti e il recupero dati da un sito esterno: OpenSky Network.

Per poterlo realizzare sono stati sviluppati 5 servizi, containerizzati e orchestrati tramite un Docker Compose. A livello implementativo sono state utilizzate le seguenti tecnologie: Python (in particolare Flask) per la scrittura dei file; MySQL come database e gRPC per la comunicazione tra i due microservizi principali.

Per ogni operazione, specialmente quelle di comunicazione con il database, sono stati implementati degli avvisi che saranno ritornati agli utenti nell'eventualità che vi sia un'eccezione.



.

I micro-servizi sviluppati sono stati i seguenti:

1. User Manager

Lo User Manager è il micro-servizio che si occupa di ricevere le richieste di registrazione e cancellazione dall'utente, interagendo con un database MySQL. Il servizio è stato sviluppato utilizzando il framework Flask e presenta 3 metodi implementati.

Test:
Mediante la route "/" si accede ad un semplice endpoint il quale restituisce un JSON informativo. Serve solamente a verificare che il servizio sia operativo.

Registrazione:
L'utente è in grado di fare la registrazione all'applicazione inviando i propri dati, in particolare l'email, il nome, il cognome e il codice fiscale.
Tale processo avviene mediante metodo POST ed è implementata con una politica "at-most-once". Per implementare la logica di idempotenza si esegue la verifica su un parametro, nominato "Request-Key", inviato con POST per ogni operazione di registrazione.

- Il Client invia la richiesta avendo in header la Request-Key ed in body un JSON contenente i parametri dell'utente.
- Lo User Manager interroga il database e verifica se la richiesta è già stata processata e presente in una tabella.
    - Se esiste la richiesta viene bloccata.
- Se il controllo è andato a buon fine si tenta di inserire i dati utente in un'altra tabella e si aggiorna il parametro status.
    - Se l'inserimento ha successo si impone status = 200.
    - Se l'inserimento fallisce si impone un codice di errore.
- Si memorizza nel database la Request-Key e lo status.

Eliminazione:
L'operazione prevede che l'utente venga verificato, analogamente alla registrazione e il record corrispondente nel database viene infine eliminato.

2. User Database

Il micro-servizio del database è utilizzato per memorizzare i dati utenti. Per realizzarlo, data la sua facilità di implementazione e interazione, è stato adoperato MySQL ed integrato nell'applicazione mediante mysql-connector-python: un driver di Python specializzato proprio nella comunicazione con la base dati.
Nell'ambito dell'User Database vengono create all'esecuzione due tabelle: *richiest* ove sono memorizzate le richieste, adoperato per garantire l'idempotenza; e *utenti* per memorizzare i dati, quali email, nome, cognome e codice fiscale di colui che intende usare il servizio.

3. Data Manager

Il Data Manager è il micro-servizio che si occupa di ricevere le richieste degli utenti, verificando che siano già stati registrati dall'User Manager, memorizzare gli aeroporti, monitorare indipendentemente e/o sotto richiesta dell'utente i voli in arrivo o in partenza da un aeroporto e memorizzare tali informazioni su un database MySQL. Il servizio è stato sviluppato utilizzando Flask e presenta diversi metodi implementati.

Verifica Utente:
Per poter accedere al servizio l'utente invia in una richiesta POST la propria email. Questa verrà poi indirizzata ad una funzione check_user del client-gRPC per separazione delle responsabilità. Questa funzione verifica se l'utente con quella email è stato precedentemente registrato dall'User Manager.
In caso affermativo l'utente risulta aver effettuato un login ed ha a disposizione le altre route del servizio, altrimenti indisponibili. Per semplicità di implementazione è stata utilizzata una variabile globale per tenere traccia dell'utente, pur non trattandosi di una scelta sicura.

Aggiungi Aeroporto:
L'utente, se risulta registrato, potrà inviare mediante JSON i dettagli relativi all'aeroporto di suo interesse; tali dati verranno poi aggiunti nel database: nella tabella aeroporti verranno posti tutti i dati, mentre nella tabella pref la correlazione tra l'email, la variabile globale memorizzata in precedenza, e l'identificativo icao dell'aeroporto.

Recupero Voli (automatico e manuale):
Il recupero dei voli avviene attraverso due meccanismi:

Periodic: viene eseguito ogni 12 ore, grazie al meccanismo di scheduling, un polling in cui si interroga l'API di OpenSky relativa ai voli in partenza da tutti i differenti aeroporti che i vari utenti hanno segnalato come preferiti. Una volta recuperati i dati vengono aggiornati gli stessi sulla tabella voli, eliminando prima eventuali inutili duplicati (stesso aeroporto e stessa data). Il meccanismo utilizzato si avvale della libreria APScheduler per eseguire attività in background.

Airport/type: manualmente l'utente inserisce l'id dell'aeroporto, controllato per verificare sia effettivamente indicato come preferito dagli utenti, e tramite il campo type viene richiamata la funzione di recupero voli in partenza (usando "departure") o arrivo (usando "arrival"). I dati recuperati vengono poi inseriti nella tabella relativa ai voli. In caso l'utente abbia già inserito l'aeroporto verrà inviato un errore, e se invece fosse già stato inserito da un altro sarà registrata soltanto la preferenza.

La funzione che si occupa di prelevare i voli richiama l'API del servizio OpenSky Network, il quale richiede un access token di autenticazione. A questo scopo la funzione get_token() contatta auth.open sky-network mediante protocollo OAuth inviando dati quali client_Id e secret_Id del progettista in questo caso, per verificare che sia effettivamente registrato al servizio esterno. Se l'operazione va a buon fine si riceve il token ed è possibile fare la richiesta sui voli.

Media voli in una settimana:
Viene calcolata la media dei voli per un dato aeroporto negli ultimi 7 giorni. Una volta calcolato il range di date si prelevano dal database, per ciascun giorno, il numero di voli filtrati, ordinati e in seguito elencati in un dizionario; alla fine si calcola la media e i dati analizzati vengono mostrati all'utente.

4. Data Database

Si tratta del micro-servizio database utilizzato per memorizzare i dati degli aeroporti, dei voli e degli aeroporti interessati ad una scansione. Per realizzarlo è stato adoperato MySQL ed integrato, come il suo omologo precedente, nell'applicazione mediante il driver mysql-connector-python.
Nell'ambito del Data Database vengono create all'esecuzione tre tabelle: *aeroporti* ove sono memorizzati gli aeroporti con il loro identificativo detto icao, il nome e il posto in cui sono situati (città e nazione); *voli* ove sono memorizzati i voli da/per gli aeroporti precedentemente salvati; sono memorizzati gli identificativi icao24, gli aeroporti di partenza e arrivo e i rispettivi orari; *pref* per tener traccia dell'associazione tra utente e aeroporto di cui è interessato conoscere i voli.

5. Server gRPC

Per poter verificare che gli utenti siano registrati, Data Collector e User Manager comunicano tra di loro sfruttando il framework gRPC.
Viene usata una funzione del client_gRPC (facente parte del container del Data Collector) detta check_user() per verificare la email. Esso si occupa di creare un canale di comunicazione verso il server corrispettivo tramite uno stub e inviare la richiesta.
Il micro-servizio server_gRPC si occupa invece di registrare la procedura chiamata AuthService, definita in un file .proto, e mettersi in ascolto sulla porta concordata 5004. In caso di comunicazione ricevere la richiesta, elaborare e spedire indietro il risultato.

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, e si collegano ad essi tramite parametri quali il nome, i database utilizzati e le password di accesso.

Il servizio relativo agli utenti è esposto sulla porta 5000, quello dei dati sulla 5001. Utilizzano inoltre due reti: app_network, che consente al manager di accedere al proprio database; app_data_network, che consente al collector sia di accedere al database che al servizio utenti, dato che devono comunicare per la verifica dell'email.
Per quanto riguarda i dati, vengono usati i file tabu.sql e tabd.sql al primo avvio per creare i due database con le rispettive tabelle, oltre ai volumi mysqldata e mysqldata2 per la memoria persistente dei dati da MySQL. Dato che il server gRPC è strettamente legato all' user_manag si è deciso, per facilità di sviluppo, di usare la stessa immagine, eseguendo di volta in volta il servizio richiesto.

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.