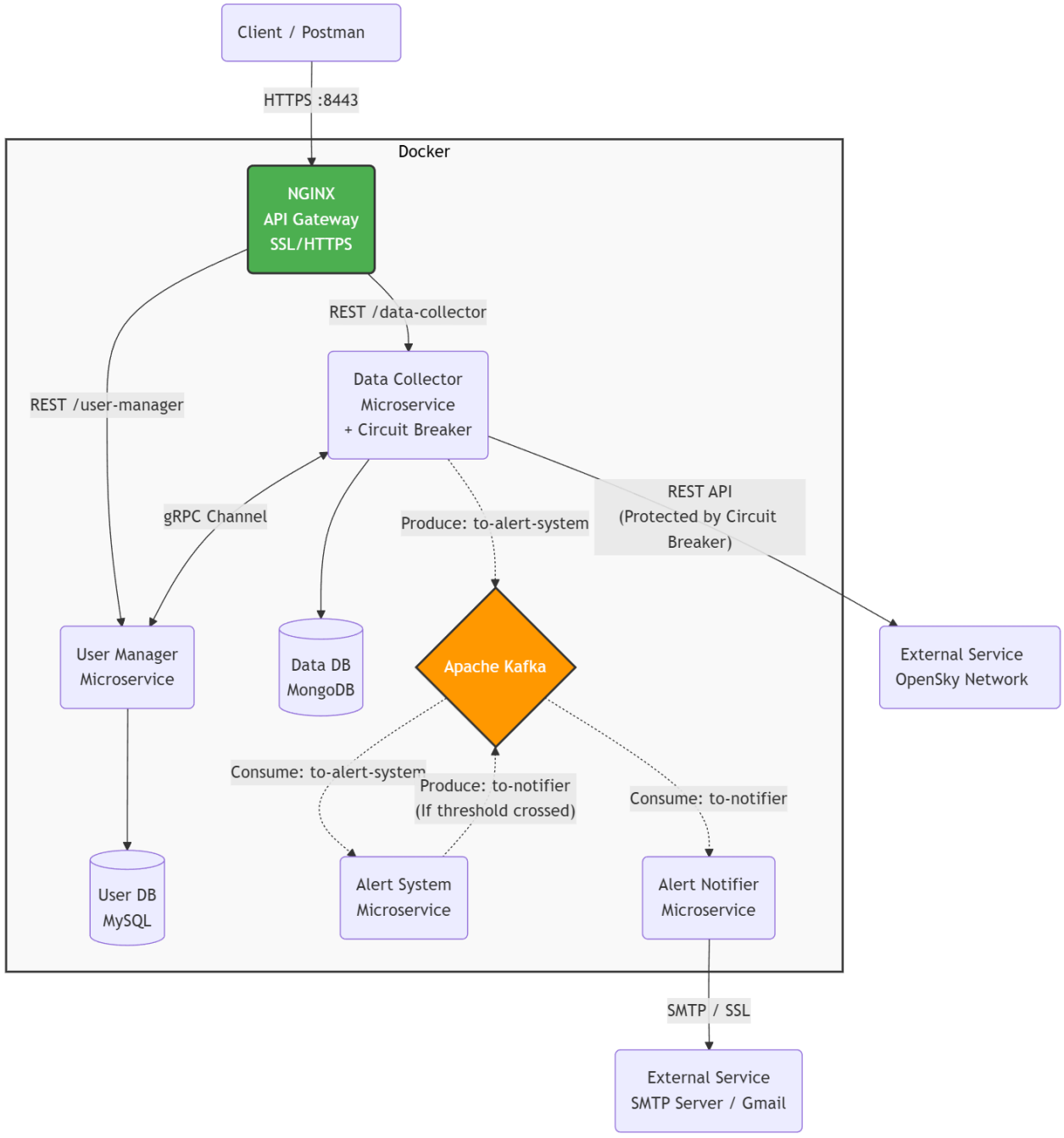
# Relazione Homework 2

## Florio Gabriele – Di Rosolini Enricomaria

## Descrizione

L'applicazione rappresenta lo sviluppo di un sistema distribuito composto da due microservizi, uno che gestisce gli utenti e l'altro che gestisce le informazioni riguardanti gli aeroporti/voli integrando un servizio esterno denominato OpenSky Network per recuperare i dati in tempo reale.

## Architettura



L’architettura adottata segue il pattern a microservizi dockerizzati, dove ognuno di essi ha una responsabilità ben definita.

Il sistema è suddiviso nei seguenti blocchi:

- **User manager microservice:** espone API REST per la gestione degli utenti, come la registrazione, la cancellazione o l’ottenimento delle relative informazioni.
- **Data collector microservice:** espone API REST per la gestione delle informazioni sui voli degli aeroporti di interesse degli utenti. Inoltre, esegue dei task periodici per scaricare dati da OpenSky Network.
- **Database:** sono stati utilizzati due database separati, uno per la gestione delle informazioni degli utenti, uno per la gestione dei dati dei voli/aeroporti, in modo tale da garantire il disaccoppiamento dei dati tra i servizi.
- **API Gateway (NGINX):** Gestisce il routing delle richieste verso *User Manager* e *Data Collector*, offrendo un layer di sicurezza tramite terminazione SSL (HTTPS) e nascondendo la complessità interna della rete Docker.
- **Message Broker (Apache Kafka):** Gestisce la comunicazione asincrona per il sistema di allerta.
- **Alert System:** Microservizio *Consumer* che analizza i dati ricevuti da Kafka e verifica se le condizioni impostate dall'utente (valori High/Low) sono soddisfatte.
- **Alert Notifier:** Microservizio consumatore dedicato all'invio delle notifiche (Email).
- **Circuit Breaker:** Modulo software integrato nel Data Collector per proteggere le chiamate verso OpenSky Network.

## Scelte progettuali

Sono state adottate le seguenti scelte progettuali a tal fine da rendere l’applicazione più scalabile e sicura.

- **API Gateway (NGINX):** è stato introdotto un server NGINX configurato come Reverse Proxy, per poter disaccoppiare il client dalla topologia interna dei microservizi. Tutte le chiamate API passano dalla porta 443 o 80, e NGINX si occupa di instradare la richiesta verso user-manager o data-collector in base al path (/user-manager/… o /data-collector/…). Inoltre, è stato generato un certificato SSL autofirmato per abilitare HTTPS, garantendo crittografia tra client e l’infrastruttura.
- **Comunicazione Asincrona (Apache Kafka):** per la gestione delle notifiche di soglia è stato abbandonato il paradigma richiesta/risposta sincrono adottato per la comunicazione tra i due microservizi (user manager e data collector) in favore di un approccio Event-Driven. Il flusso è diviso su due topic distinti:
    - **Topic to-alert-system:** Il Data Collector agisce da *Producer*. Dopo aver aggiornato i dati dei voli, pubblica un messaggio contenente i nuovi dati e la lista degli utenti interessati. Il Data Collector invia nel messaggio Kafka sia i dati dei voli che i profili utente interessati per evitare che l’Alert System debba interrogare il database appartenente al microservizio data-collector.

    - **Topic to-notifier**: L'Alert System, agisce da *Consumer* e *Producer*, processa i dati e se rileva il superamento di una soglia, pubblica un evento di "notifica" su questo topic.
- **Circuit Breaker:** L'implementazione del Circuit Breaker non si limita a conteggiare gli errori generici, ma adotta una logica raffinata per distinguere tra malfunzionamenti transitori e risposte valide. Il componente gestisce tre stati distinti:
    - **Closed:** Stato normale, le richieste passano OPEN**:** Dopo il superamento della soglia di fallimenti (failure_threshold=5), le richieste vengono bloccate preventivamente sollevando un'eccezione CircuitBreakerOpenException, riducendo il carico sul servizio esterno in difficoltà.
    - **Half-open:** Trascorso il recovery_timeout (30s), il sistema lascia passare una singola richiesta "sonda". Se questa ha successo, il circuito si chiude e il conteggio dei fallimenti viene resettato; altrimenti torna aperto.

  **Gestione Eccezioni (404 vs 500):** È stata introdotta una distinzione fondamentale nella gestione degli errori HTTPS. Mentre gli errori server (5xx) o i timeout incrementano il contatore dei fallimenti, gli errori client come il 404 Not Found vengono ignorati dal Circuit Breaker. Questa scelta è dovuta da OpenSky Network che non trova voli in partenza o in arrivo per determinate fasce orarie restituendo un errore 404 Not Found, non strettamente legato alla mancanza di voli che effettivamente partono da un determinato aeroporto, ma bensì da problemi di OpenSky Network.

## Microservizi

### AlertSystem

Il microservizio AlertSystem è un servizio completamente indipendente e leggero che agisce sia da consumer che da producer, prendendo i dati dal topic “to-alert-system” e verificando se la somma del numero di voli in partenza e arrivo da un determinato aeroporto violino le soglie impostate dall’utente, in caso affermativo produce un messaggio contenente i parametri utente, aeroporto e condizione di superamento soglia, pubblicandolo sul topic “to-notifier”

### AlertNotifierSystem

Il microservizio AlertNotifierSystem è un servizio completamente indipendente che agisce da consumer, prendendo i dati dal topic “to-notifier” e notificando l’utente del superamento della soglia da lui imposta tramite server SMTP (Gmail).

Sia il microservizio AlertSystem che AlertNotifierSystem, implementano una funzione chiamata wait_for_kafka. In un ambiente distribuito containerizzato, l'ordine di avvio definito da depends_on in Docker Compose non garantisce che l'applicazione all'interno del container sia pronta a ricevere connessioni. Per risolvere questo problema di "Race Condition" all'avvio, è stata implementata una logica all'interno dei microservizi consumer che tenta ciclicamente di ottenere i metadata dal broker prima di avviare il loop di consumo

principale. Questo approccio impedisce crash immediati dei microservizi in fase di deploy, rendendo l'intera infrastruttura più stabile e capace di “auto-guarigione” in caso di riavvii del broker.

### DataCollector

L'API /add_interest del Data Collector è stata aggiornata per accettare due nuovi parametri opzionali:

- highValue (soglia superiore)
- lowValue (soglia inferiore)

Il Data Collector esegue una validazione (assicurandosi che lowValue < highValue) e persiste questi dati nel documento MongoDB relativo all'interesse dell'utente.

## Lista API

### User manager

Tutte le API rivolte al microservizio User Manager dovranno adottare questo tipo di url:

- **https://localhost:8443/user-manager/ENDPOINT**

| Metodo | Endpoint | Descrizione | Body (JSON) |
|---|---|---|---|
| POST | /add_user | Registrazione nuovo utente | { "request_id": ...,<br>"email" : "...",<br>"name" : "...",<br>"surname": "...",<br>"age" : ...,<br>"CF" : "...",<br>"phone": "..." } |
| POST | /rmv_user | Rimuovi utente | { "email" : "..." } |
| POST | /get_user | Ottieni informazioni utente | { "email" : "..." } |

### Data-collector

Tutte le API rivolte al microservizio User Manager dovranno adottare questo tipo di url:

- **https://localhost:8443/data-collector/ENDPOINT**

| Metodo | Endpoint | Descrizione | Body (JSON) |
|---|---|---|---|
| POST | /add_interest | Aggiunge un nuovo interesse | {"email" : "...",<br>"airport_code": "..."<br>"lowValue" : ...,<br>"highValue": ...} |
| POST | /rmv_interest | Rimuove un interesse | {"email" : "...",<br>"airport_code": "..." } |

| | | | |
|---|---|---|---|
| POST | /list_interest | Mostra la lista degli interessi dell'ut. | {"email" : "..." } |
| POST | /get_flight | Ottiene i voli in arrivo e in partenza dall'aeroporto | {"email" : "...",<br>"airport_code": "..." } |
| POST | /force_update | Forza l'aggiornamento del db | // |
| POST | /get_last_flight | Ottiene l'ultimo volo in arrivo e in partenza dall'aeroporto | {"email" : "...",<br>"airport_code": "..." } |
| POST | /average | Calcola la media voli degli ultimi x giorni | {"email" : "...",<br>"airport_code": "..."<br>"days": ...} |
| POST | /get_arrivals | Ottiene i voli in arrivo che partono da un determinato aeroporto | {"email" : "...",<br>"airport_code_arrivals":"...",<br>"airport_code_departures":"..."<br>} |

## Diagramma delle interazioni