Sistemi Distribuiti e Cloud Computing

Project B3: Microservice application of your choice

Luca Ardovino
Matricola: 0345008
Laurea Magistrale in Ingegneria Informatica
Università degli studi di Roma Tor Vergata
luca.ardovino@students.uniroma2.eu

***Abstract*— Questo documento ha l'obiettivo di descrivere la realizzazione di una semplice applicazione a microservizi. L'obiettivo del progetto è implementare almeno tre servizi (che non si limitino ad eseguire operazioni CRUD al database) ed almeno due pattern noti nel mondo dei microservizi. I pattern scelti sono "Database per Service" e "Circuit Breaker" e di seguito verrà presentata tutta l'architettura dell'applicazione e come tali pattern sono stati introdotti all'interno dello sviluppo della stessa. Infine, saranno discusse anche le funzionalità offerte e le possibili sfide future.**

I. Introduzione

Il progetto è stato realizzato andando a sviluppare un'architettura a microservizi basata sull'algoritmo "Top Trading Cycle". L'utente per cui è stata pensata l'applicazione, infatti, è un laureato in ingegneria informatica (sia triennale che magistrale) in un Università di Roma in cerca di lavoro in ambito IT e l'algoritmo TTC permette di eseguire un matching stabile con le aziende richiedenti di personale proprio in tale ambito. Per utilizzare l'applicazione, l'utente deve quindi registrarsi compilando l'apposito form che prevede l'inserimento di alcuni dati (tra cui il tipo di laurea ed il voto conseguito, oltre che ai classici dati anagrafici) ed a quel punto potrà accedere all'applicazione effettiva, dove avrà accesso ad alcune funzionalità. La funzionalità principale è l'esecuzione del matching tramite l'algoritmo TTC: l'utente, infatti, dovrà ordinare, in base alle proprie preferenze, in quale ambito IT vuole lavorare scegliendo tra Cybersecurity, Software e Data Science ed a quel punto potrà eseguire l'algoritmo TTC che andrà a consigliare all'utente uno dei tre ambiti, tenendo in considerazione anche in quale dei tre ambiti citati le aziende ricercano personale. Per questo, nonostante l'applicazione sia pensata per un utente in cerca di lavoro, per completezza è stata inserita la possibilità per un'azienda di effettuare una registrazione indicando l'ambito in cui cerca del personale, sempre tra Cybersecurity, Software e Data Science. Appena ottenuto il responso, l'utente riceve, quasi contemporaneamente e tramite email, l'elenco delle aziende che stanno cercando personale nell'ambito che gli è stato appena consigliato. Ottenuto questo elenco, l'utente potrà accedere alla funzionalità di raccomandazione, inserendo il nome di un'azienda tra quelle presenti nella mail per sapere quanto sia consigliata in base ai voti di altri utenti. Infatti, se un utente ha già avuto esperienze lavorative con una azienda può valutarla inserendo nel sistema un voto che va da 1 a 5, il quale contribuirà ed essere utilizzato per fornire la raccomandazione appena descritta. Ovviamente, ai fini del funzionamento dell'applicazione, un'azienda può essere votata solo se anch'essa registrata al sistema.

Di seguito nel documento, verranno presentate tutte le scelte architetturali e di implementazione per realizzare il sistema.

II. Architettura a microservizi

A. Struttura generale

L'applicazione è composta in totale da 4 servizi (+ il servizio di frontend che serve solo ad esporre nel browser le pagine html), tutti sviluppati in linguaggio Go, e sfrutta chiamate RPC per la comunicazione, effettuata sempre tramite chiamate sincrone. Ciascuno dei microservizi è inserito all'interno di un container, sfruttando Docker per la creazione dell'immagine tramite Dockerfile, mentre per la loro organizzazione è stato usato sia Docker Compose che Kubernetes. Al livello di pattern, come già anticipato, sono stati implementati "Database per Service", per cui ogni microservizio ha un proprio database ad uso esclusivo (i database usati sono stati MySql e MongoDB), e il "Circuit Breaker", che è stato inserito per le chiamate sincrone RPC al fine di evitare la formazione di errori a cascata.

I servizi sviluppati per realizzare l'applicazione sono: ApiGateway per veicolare le richieste, MatchingService che implementa l'algoritmo Top Trading Cycle, RecommendationService, che implementa un algoritmo basato su somiglianza del coseno per fornire le raccomandazioni, NotificationService, che realizza il servizio di invio della mail all'utente sfruttando un semplice client SMTP.

B. Servizi

1) ApiGateway

Il servizio *ApiGateway* serve a veicolare le richieste che arrivano dal client all'apposito microservizio. Questo significa che il client, attraverso il frontEnd che espone le pagine html, comunicherà sempre con questo servizio, il quale andrà ad

invocare tramite chiamata RPC sincrona l'apposita funzionalità richiesta. All'interno di questo *ApiGateway* avviene l'utilizzo del pattern Circuit Breaker, per cui ogni qualvolta avverrà la chiamata RPC sincrona lo si farà utilizzando tale design pattern in modo tale che, se qualche servizio non è disponibile, magari per un sovraccarico di richieste, non avvenga un sovraccarico ulteriore andando a creare dei blocchi e degli errori a cascata.

2) MatchingService

Il servizio *MatchingService* è il servizio principale dell'applicazione e va a realizzare l'algoritmo Top Trading Cycle (TTC), famoso algoritmo nell'ambito della Teoria dei Giochi. L'algoritmo, in generale, serve a risolvere il problema dell'allocazione delle case, in cui ogni giocatore possiede inizialmente una casa e ha una lista di preferenze delle altre case. Con n giocatori, l'esecuzione dell'algoritmo produce un matching stabile, ossia nessun gruppo di giocatori può formare una coalizione per scambiarsi le case in modo che ognuno ottenga una casa non peggiore. L'idea, quindi, è stata quella di sfruttare questo algoritmo per eseguire un matching stabile tra utenti che cercano lavoro e aziende che ne richiedono. Ora, poiché l'algoritmo TTC richiede che ci siano n giocatori ed n case disponibili, sono stati eseguiti dei raggruppamenti per poter eseguirlo correttamente anche in un'applicazione del genere. Per cui, quando un utente richiede di eseguire il matching succede questo: gli utenti registrati al sistema vengono raggruppati in tre categorie (laureati triennale, laureati magistrale con voto < 100, laureati magistrale con voto > 100), così come le aziende vengono raggruppate in tre categorie (aziende che cercano personale nel campo Cybersecurity, quelle che lo cercano in campo Software, quelle che lo cercano in campo Data Science). Grazie a questa semplificazione, si riesce ad eseguire semplicemente l'algoritmo con n = 3. Per eseguire correttamente l'algoritmo, l'utente deve ordinare i 3 ambiti Cybersecurity, Software e Data Science e tale scelta contribuisce a creare delle preferenze per i vari gruppi precedentemente elencati che sono fondamentali per applicare l'algoritmo nel modo giusto. Il risultato, che non è nient'altro che un ambito tra Cybersecurity, Software e Data Science, viene poi mostrato a schermo e l'ultimo responso effettuato sarà sempre visibile per l'utente.

3) NotificationService

Il servizio *NotificationService* è il servizio che va a supporto del servizio di matching principale. Infatti, nel momento in cui l'utente esegue il matching, appena il responso è pronto, il sistema invia una mail all'utente contenente l'elenco delle aziende registrate nel sistema che richiedono personale nell'ambito appena consigliato all'utente. La comunicazione trai il servizio di matching che esegue il TTC e il servizio di notifica che invia la mail è gestita tramite comunicazione asincrona direttamente tra i due servizi. Infatti, il servizio di notifica non è nient'altro che un consumatore ad una coda di cui il servizio di matching è produttore e questa cosa è gestita usata RabbitMQ. Per cui, eseguito l'algoritmo, il servizio di matching scrive nell'apposita coda le informazioni necessarie e il servizio di notifica consumatore le usa per inviare la mail corretta. Inoltre, la mail inviata viene memorizzata nel proprio database dal servizio di notifica e questo consente all'utente di utilizzare un'altra funzionalità, ossia richiedere di rispedire l'ultima mail ricevuta con l'elenco delle aziende. Tale funzionalità è stata pensata per tutelare l'utente nel caso perdesse la mail ricevuta precedentemente o per qualsiasi motivi non l'avesse ricevuta. Tale funzionalità non è accessibile all'utente se non ha ancora effettuato almeno un matching perché, ovviamente, in quel caso non c'è nessuna mail da recuperare. Infine, le mail vengono spedite utilizzando un semplice client SMTP.

4) RecommendationService

Il servizio *RecommendationService* è l'altra funzionalità importante che l'applicazione offre. Infatti, una volta che l'utente ha ricevuto il responso del matching con tanto di mail contenente l'elenco delle aziende di quell'ambito, ha a disposizione la possibilità di richiedere un consiglio su una delle aziende per capire quale sia quella che fa al caso suo. Tale funzionalità, però, come nel caso della richiesta di ricevere nuovamente la mail, non è accessibile per l'utente se prima non ha eseguito almeno una volta il matching. Anche qui, l'idea è che, se l'utente non ha un elenco di aziende ricevute. non avrebbe senso cercare un'azienda. A questo punto, il funzionamento del servizio è semplice: eseguito il matching, l'utente può accedere all'apposita pagina in cui può inserire il nome di un'azienda e richiedere il consiglio per tale azienda. Ovviamente, se l'azienda non è presente nel sistema la procedura non verrà effettuata, mentre se l'utente digita un'azienda che non ha avuto nessun voto da altri utenti verrà segnalato all'utente. Infatti, questo servizio di raccomandazione si basa su un sistema di voti eseguito da altri andando a funzionare come una sorta di "collaborative filtering". Per cui, l'utente, oltre al matching, ha disposizione fin da subito la possibilità di esprimere un voto all'esperienza di lavoro vissuta con un'azienda, potendo inserire una valutazione con un numero da 1 a 5. Questo voto verrà utilizzato proprio nel servizio di raccomandazione. In particolare, il servizio implementa un algoritmo basato sulla similarità del coseno che funziona nel seguente modo: in input l'algoritmo avrà l'utente, il tipo di laurea dell'utente (triennale, magistrale con voto > 100, magistrale con voto < 100), voto di laurea dell'utente ed azienda di cui si vuole sapere se consigliata o no e l'obiettivo è restituire una lista di aziende consigliate basandosi sui voti di altri utenti simili all'utente richiedente. Per raggiungere tale obiettivo, l'algoritmo recupera, dal proprio database del servizio, gli utenti simili in tipo di laurea e voto all'utente target, costruisce un profilo medio usando tali utenti simili e per ogni utente simile calcola la similarità con tale utente medio utilizzando proprio la similarità del coseno, per ogni azienda si sommano le valutazioni date dagli utenti simili pesate per la loro similarità al profilo medio ed infine si ottiene una lista ordinate delle aziende raccomandate in base al punteggio calcolato. Ora, se l'azienda richiesta dall'utente target, è presente nella lista delle aziende raccomandate, il sistema risponde con la posizione in classifica di tale azienda in modo che l'utente possa capire quanto essa sia raccomandata, altrimenti la risposta è

semplicemente che l'azienda non è raccomandata perché nemmeno presente nell'elenco.

C. Pattern

1) Database per Service

Ciascun servizio presentato (ad eccezione, ovviamente, dell'ApiGateway) possiede un proprio database per la gestione dei dati. Il MatchingService possiede un database MySQL con 2 tabelle: una tabella aziende in cui vengono memorizzate le aziende registrate al sistema con il relativo ambito in cui cercano personale, ed una tabella datiutente in cui vengono memorizzate le credenziali degli utenti ed i responsi del matching. Già da qui emerge come questo database venga riempito già nelle fasi di registrazione al sistema sia per gli utenti che per le aziende e poi nuovamente acceduto durante l'effettiva esecuzione dell'algoritmo di matching. Il RecommendationService possiede anch'esso un proprio database MySQL con una sola tabella: la tabella votazione memorizza i dati degli utenti che effettuano una votazione per le aziende in cui hanno lavorato. Quindi, tale database viene utilizzato sia quando l'utente vota un'azienda e sia quando viene eseguito l'algoritmo di raccomandazione precedentemente descritto. Il NotificationService posssiede un proprio database, stavolta noSQL ed in particolare un database MongoDb. Questo perché tale servizio si limita a memorizzare il contenuto della mail ed il destinatario e quindi non ha bisogno delle *"*rigidità*"* presenti in un database MySQL. Tale database, quindi, viene utilizzato quando il sistema invia le mail all'utente dopo l'esecuzione del matching e, successivamente, se l'utente richiede di ricevere nuovamente la mail.

2) Circuit Breaker

Il pattern Circuit Breaker è implementato all'interno dell'ApiGateway. Infatti, è responsabilità dell'ApiGateway invocare i vari servizi all'interno dell'applicazione e questa cosa viene effettuata quasi sempre tramite chiamate RPC sincrone. Di conseguenza, l'implementazione di questo pattern riguarda i metodi che invocano effettivamente tali chiamate RPC sincrone, in modo che il problema ad un servizio non conduca ad un ulteriore sovraccarico del sistema o all'accumularsi degli errori all'interno dell'applicazione. Per questa ragione, ogni esecuzione di una chiamata RPC è gestita dal Circuit Breaker, il quale interviene aprendo il circuito se un determinato servizio non dovesse essere raggiungibile, per poi richiuderlo nel caso in cui il servizio dovesse tornare ad essere disponibile. Entrando nello specifico, dopo aver effettuato un numero di tentativi falliti pari a 3 ad un certo servizio, viene aperto il circuito (in particolare il circuito passa allo stato "open"). In questa fase, le richieste vengono direttamente bloccate, ma dopo 10 secondi il circuito passa dallo stato "open" allo stato "half-open" e qui viene ammessa una richiesta al servizio per capire se è tornato disponibile: se il servizio è effettivamente disponibile, il circuito passa allo stato "closed", altrimenti rimane nello stato "open" e riparte il timeout, e così via. Questo modus operandi è stato utilizzato per tutti i Circuit Breaker nell'ApiGateway, ma anche per il Circuit Breaker del MatchingService, il quale lo utilizza per connettersi al servizio RabbitMQ in qualità di produttore.

D. Tecnologie utilizzate

1) Librerie

Le librerie utilizzate per realizzare l'applicazione sono state le seguenti: - *github.com/go-sql-driver/mysql*: libreria che fornisce un driver per connettere applicazioni Go ad un database MySQL. Nel codice, l'import è preceduto da un "_", che indica che la libreria viene importata solo per eseguire il suo *init()*, ovvero per registrare il driver MySQL nel pacchetto *database/sql*. Tale libreria è stata usata nei servizi MatchingService e RecommendationService che, appunto, usano un database MySQL

- *github.com/rabbitmq/amqp091-go*: libreria che implementa il protocollo AMQP 0.9.1, utilizzato per comunicare con un broker RabbitMQ. Serve per la gestione di code di messaggi, publish/subscribe, scambio di messaggio tra servizi, etc etc. Nel progetto, la libreria è usata dal matchingService per comunicare con RabbitMQ come produttore, e dal notificationService per comunicare con RabbitMQ come consumatore
- *github.com/sony/gobreaker*: libreria che implementa il pattern Circuit Breaker in Go, utile per aumentare l'affidabilità delle comunicazioni tra servizi, prevenendo chiamate ripetute verso componenti in errore e favorendo un recupero controllato. Nel progetto è presente nell'apiGateway per le chiamate RPC ai vari servizi e nel matchingService per l'utilizzo del pattern nella comunicazione con RabbitMQ
- *go.mongodb.org/mongo-driver/mongo*: è il pacchetto principale del driver MongoDB per Go. Fornisce le funzionalità per connettersi a un cluster MongoDB, eseguire query, inserimenti, aggiornamenti, cancellazioni, gestione delle collezioni e dei database. Nel progetto è presente (sia questa che le prossime due librerie) nel notificationService, unico servizio che usa MongoDB
- *go.mongodb.org/mongo-driver/mongo/options*: pacchetto che fornisce strutture di configurazione per controllare il comportamento delle operazioni MongoDB
- *go.mongodb.org/mongo-driver/bson*: pacchetto che serve per serializzare e deserializzare dati in formato BSON (Binary JSON), che è il formato nativo usato da MongoDB. È fondamentale per mappare le strutture dati Go nei documenti MongoDB e viceversa

2) Software

Gli strumenti software utilizzati nella realizzazione dell'applicazione sono stati i seguenti: - *Docker*: piattaforma di containerizzazione che consente di impacchettare applicazioni e le loro dipendenze in ambienti isolati e portabili, detti container, semplificando la distribuzione e la scalabilità. Nel

progetto, è stato usato per scrivere i Dockerfile di ciascun servizio.

- *Kubernetes*: sistema open-source per l'orchestrazione dei container, utilizzato per automatizzare il deployment, la scalabilità e la gestione di applicazioni containerizzate in ambienti distribuiti. In particolare, nel progetto sono stati scritti file *deployment.yaml* e *service.yaml* per tutti i servizi, con la aggiunta di file *pvc.yaml* per gestire i volumi nel caso dei database. Inoltre, per l'esecuzione del cluster Kubernetes è stata usata l'estensione di Docker Desktop per Kubernetes e l'interazione è stata gestita tramite *kubectl*.
- *RabbitMQ*: message broker che implementa il protocollo AMQP (Advanced Message Queuing Protocol), utilizzato per lo scambio asincrono di messaggi tra componenti di un sistema, facilitano la comunicazione tra servizi. Nel progetto è usato per gestire la comunicazione tra matchingService (produttore) e notificationService (consumatore)
- *Database MySQL*: sistema di gestione di basi di dati relazionali e che supporta il linguaggio SQL per la gestione e l'interrogazione dei dati strutturati. Nel progetto, matchingService e recommendationService usano un proprio database MySQL
- *Database MongoDB*: database noSQL orientato ai documenti che utilizza BSON per la memorizzazione dei dati e permette una gestione flessibile e scalabile di strutture dati semi-strutturate. Nel progetto, il servizio NotificationService utilizza un proprio database MongoDB
- *Client SMTP*: componente utilizzato per inviare mail tramite il protocollo Simple Mail Transfer Protocol. Nel progetto, tale client è usato dal NotificationService per l'invio della mail all'utente
- *Locust*: strumento open-source per il load testing, scritto in Python, che consente di simulare utenti concorrenti in modo programmabile per testare le prestazioni e la scalabilità di sistemi e API. Tale strumento è stato usato per eseguire test di carico sull'applicazione
- *Docker Compose*: strumento di Docker che consente di definire e gestire configurazioni multicontainer tramite un file YML, facilitando l'orchestrazione e l'avvio simultaneo di più servizi in ambienti di sviluppo o test. Il progetto supporta principalmente questa modalità per l'orchestrazione dei container, per questo è stato scritto un file *docker-compose.yml*

E. *Test e risultati*

L'applicazione è stata testata usando l'orchestrazione di Docker Compose attraverso Locust. La GUI di Locust è stata eseguita tramite *localhost* e si collegava all'applicazione in esecuzione sulla macchina EC2 ed i risultati sono stati raccolti tramite la gui fornita da Locust stesso. Nella configurazione di Locust sono stati usati 30 utenti (15 utenti laureati e 15 utenti che inseriscono solo una nuova azienda) che accedono contemporaneamente al sistema, e ciascun utente ha utilizzato con varie frequenze tutte le chiamate presenti nell'applicazione. Per le funzionalità principali è possibile dire che il tempo massimo di risposta è stato rispettivamente il seguente:

- Funzionalità di matching: 432 ms
- Funzionalità di raccomandazione: 608 ms
- Funzionalità di inserimento voto: 470 ms

L'applicazione non ha presentato fallimenti per le feunzionalità principali, garantendo quindi una buona stabilità. Nella sezione *IV. Figure e Tabelle* è presente una tabella con i risultati ottenuti per singolo metodo.

III. Conclusioni e sviluppi futuri

L'applicazione rappresenta un'interessante (e molto semplice) implementazione dell'algoritmo Top Trading Cycle applicato in un ambito come quello del lavoro in cui è sempre alla ricerca di nuove idee per migliorare la situazione di domanda-risposta tra lavoratori ed aziende. Ovviamente, l'implementazione dell'algoritmo può essere resa ottimale per casi più complessi (banalmente, si possono aggiungere più categorie o anche, in generale, far diventare l'applicazione aperta al mondo del lavoro in generale e non solo in ambito IT). Attualmente, l'applicazione applica alcune delle tecniche e dei pattern più importanti dei microservizi ed i risultati confermano buone prestazioni.

IV. Figure e tabelle

Di seguito è possibile trovare la tabella con i tempi di risposta risultanti dal test Locust ed il diagramma di flusso dell'applicazione.

TABLE I. Tempi di risposta

Tipo	**Risultati**		
	Nome	***# richieste***	***50%ile (ms)***
POST	/service1/match/matching	116	140
POST	/service2/notify	68	1200
POST	/service3/recommendation/insert	253	130
POST	/service3/recommendation/recommend	264	130

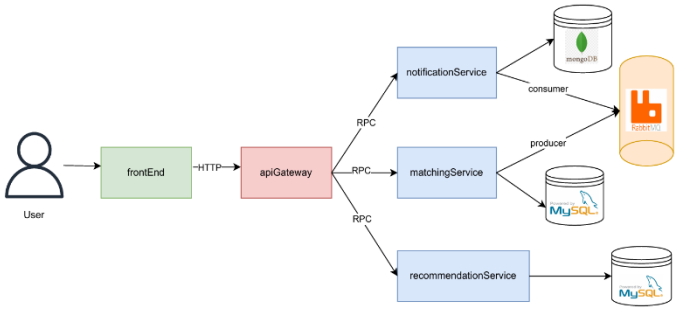

Fig. 1. Diagramma di flusso dell'applicazione

Referenze

[1] C. Richardson, "API Gateway pattern," Microservices.io, 2025. [Online]. Available: https://microservices.io/patterns/apigateway.html

[2] M.Clinton, "Microservice Circuit Breaker Pattern with RabbitMQ," *Level Up Coding* on Medium, Mar. 1, 2023. [Online]. Available: https://levelup.gitconnected.com/microservice-circuit-breaker-pattern-with-rabbitmq-a8cc1ebd3c7c