Distributed Systems and Big Data

Relazione Finale

Matteo Pidone - 1000042321
Tomas Prifti - 1000040388

1 - Introduzione

L'obiettivo del progetto è la realizzazione di un sistema che permetta di recuperare delle metriche da un *server Prometheus*. Il server espone delle metriche attraverso diversi *exporter*, tra cui **node_exporter**. Dopo aver estratto le metriche, queste vengono analizzate ed elaborate, generando alcune statistiche di monitoraggio. I dati prodotti vengono poi esposti all'utente attraverso interfacce grafiche accessibili direttamente dal *browser*. L'architettura del sistema è a **microservizi** gestito tramite l'ausilio di ***Docker***. Ciascun microservizio è in grado di comunicare con gli altri per lo scambio delle informazioni e la gestione di esse. I principali microservizi coinvolti sono i seguenti:

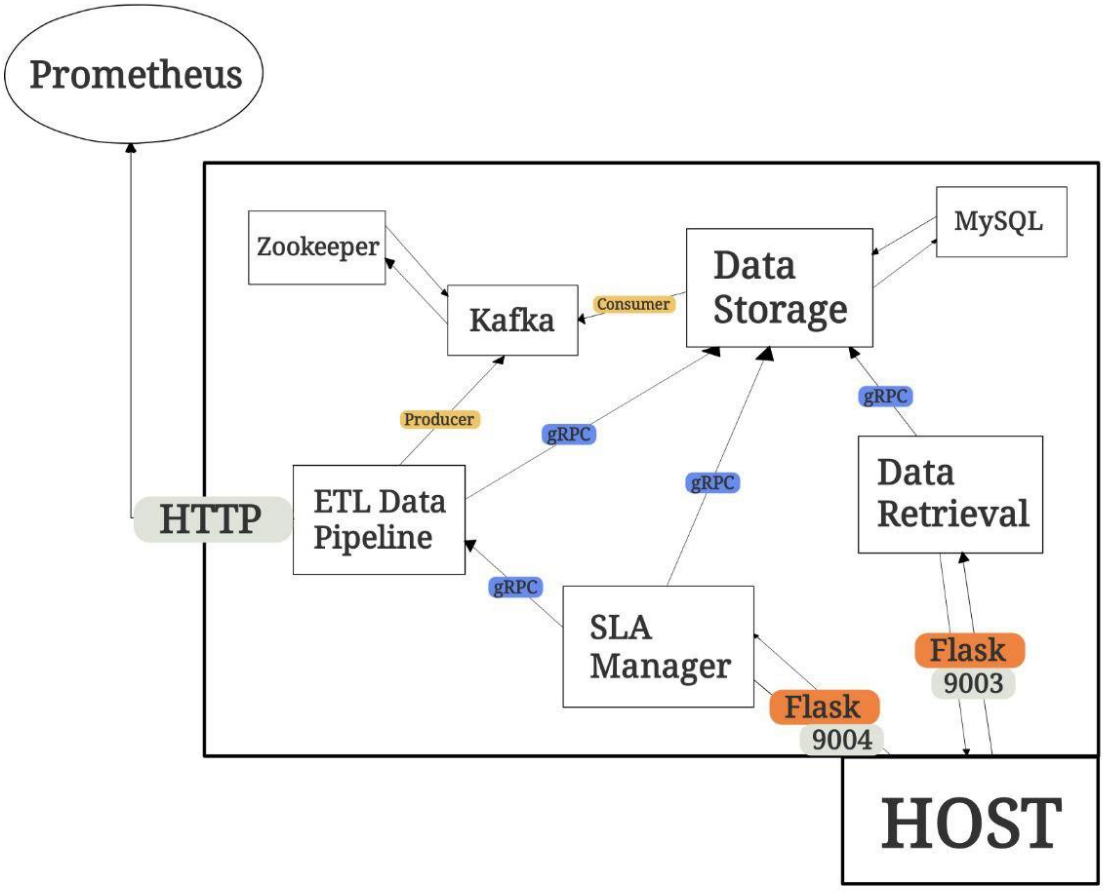
- ***ETL data pipeline***, un microservizio che, ogni 10 minuti, calcola metadati, statistiche (max, min, avg e dev_std) per 1h, 3h, 12h e predice i valori di max, min e avg per i successivi 10 minuti. I risultati vengono poi inviati ad un **topic Kafka** *"prometheusdata"*. È presente un sistema di monitoraggio interno per visualizzare il tempo di esecuzione delle diverse funzionalità.
- ***Data Storage***, un microservizio che si occupa di leggere i dati inviati su *Kafka* e di memorizzarli all'interno di un database. Inoltre espone ad altri microservizi i risultati prodotti.
- ***Data Retrieval***, un microservizio che offre un'interfaccia *REST* per la visualizzazione delle informazioni ricavate dai precedenti microservizi.
- ***SLA Manager***, un microservizio che permette di definire un SLA composto da un set ristretto di metriche con range di valori ammissibili (max e min). Inoltre, fornisce informazioni sulle violazioni delle metriche nelle ultime 1, 3, 12 ore e sulle possibili violazioni future. Queste informazioni vengono poi mostrate attraverso un'interfaccia REST all'utente.

2 - Schema Architetturale

Il sistema si basa sul funzionamento di 4 microservizi principali :

- **ETL Data Pipeline**, denominato anche *microservice_1*
- **Data Storage**, denominato anche *microservice_2*
- **Data Retrieval**, denominato anche *microservice_3*
- **SLA Manager**, denominato anche *microservice_4*

Nella seguente immagine viene riportato il sistema a microservizi realizzato :



Sopra vengono rappresentati gli elementi che costituiscono il sistema.

Le frecce (direzionali) rappresentano lo schema di comunicazione tra le parti.

Nella strutturazione del progetto e durante le varie iterazioni, si è cercato di **isolare** le funzionalità dei microservizi, definendo il loro campo di lavoro. Il sistema è stato strutturato in modo tale che esso non dipenda fortemente da un microservizio in particolare, ma dipenda dalla **cooperazione** tra tutti i microservizi. Per far ciò abbiamo cercato di garantire l'isolamento tra i microservizi separando le **responsabilità**.
Sono comunque presenti relazioni di dipendenza tra loro. Questo è dovuto al fatto che, per fare in modo che il sistema funzioni correttamente, ciascun microservizio deve aspettare che gli altri abbiano raggiunto una condizione tale da garantire il corretto funzionamento. Ovviamente un'analisi approfondita sarebbe ideale per la costruzione di un software ancora più **solido** e **resiliente** ai guasti.
La comunicazione interna tra i container avviene principalmente attraverso l'uso di **gRPC**, in modo **sincrono**, all'interno di una rete **bridge** privata al sistema. Inoltre, è stato utilizzato il **Framework *Flask*** per l'esposizione e la visualizzazione dei dati verso l'***host***. Abbiamo previsto una comunicazione **asincrona** e **indiretta** tramite l'utilizzo di ***producer*** e ***consumer*** che scambiano messaggi attraverso un ***broker kafka.***

3 - Configurazione e avvio

È stata dedicata molta attenzione al setup del progetto, configurando in modo dettagliato il file ***docker-compose.yml***. È stato reso tutto il più automatizzato possibile, in questo modo la fase di ***deploy*** diventa più semplice.

Costanti

Di seguito elenchiamo le costanti di configurazione utilizzate all'interno del file :

- **ETL Data Pipeline**
  - **PROMETHEUS_SERVER**: http://15.160.61.227:29090
  - **INTERVAL_TIME_SECONDS**: 600
  - **PATH_LOG_MONITOR**: /usr/src/application/log/
  - **KAFKA_BROKER**: kafka:9092
  - **KAFKA_TOPIC**: prometheusdata
- **Data Storage**
  - **KAFKA_BROKER**: kafka:9092
  - **KAFKA_TOPIC**: prometheusdata
  - **MYSQL_HOST**: mysql
- **Data Retrieval**
  - **DATA_STORAGE_GRPC_SERVER**: microservice_2:50051
- **SLA Manager**
  - **DATA_STORAGE_GRPC_SERVER**: microservice_2:50051
  - **ETL_DATA_PIPELINE_GRPC_SERVER**: microservice_1:50051

Networks

È stata impostata una ***network*** di **default**, in modo da far comunicare tutti i microservizi attraverso una **rete privata** visibile solo dal sistema. Questo è stato possibile attraverso la configurazione ***networks***, definendo la tipologia di **driver** come **bridge**, in modo da far comunicare i microservizi attraverso un intermediario comune a tutti.

Volumes

È stato utilizzato un ***bind-mount*** per la gestione della **persistenza** dei file. I file in questione sono dei **log** generati dall' **ETL Data Pipeline** che, in questo modo, offre un sistema di monitoraggio interno al microservizio.

Healthcheck

A causa della dipendenza dei microservizi da altri servizi, quali ***mysql*** e ***kafka***, sono state introdotte delle dipendenze di avvio. In questo modo è stato specificato quale condizione risulta valida per l'avvio dei microservizi, attraverso la definizione di ***healthcheck***. Sia nel servizio ***mysql***, che in ***kafka***, è stato introdotto un controllo sullo stato attuale del servizio offerto. Questo è stato realizzato attraverso una richiesta **HTTP**, con i comandi ***curl*** e ***nc*** rispettivamente, all'***url*** su cui il servizio, una volta eseguito, è in ascolto.

Configurazione JSON

Sono presenti due file di configurazione in formato ***json***. Il primo (***config.json***) si trova all'interno dell' **ETL Data Pipeline** ed il secondo (***database_schema.json***) all'interno del **Data Storage.**

- ***config.json*** - Permette di inserire le metriche e le statistiche che verranno analizzate all'interno del sistema, in modo strutturato ed intuitivo. È possibile, infatti, modificare questo file cambiando le metriche desiderate.
- ***database_schema.json*** - Permette di definire la struttura relazionale all'interno del database ***MySQL***.

4 - I microservizi

In questo capitolo vengono descritte le principali funzionalità esposte da ogni microservizio. Alla fine vengono poi raccolte delle considerazioni sulle scelte adottate, adottabili e sui possibili punti di forza o debolezza.

4.1 - ETL Data Pipeline

Il primo microservizio recupera le metriche da un *server Prometheus* all'indirizzo http://15.160.61.227:29090 .

All'interno di esso, sono **due** i processi che vengono avviati:

- Il primo processo viene lanciato dal *Dockerfile*, si occupa di fare *scraping* delle metriche. Esso affida il calcolo di metadati, statistiche e predizioni a più *Thread* e continua la sua esecuzione. Durante il calcolo dei vari dati, viene monitorato il tempo di esecuzione necessario per eseguire ciascun calcolo. È presente un **sistema di monitoraggio** che raccoglie i tempi di esecuzione così generati e scrive le informazioni in file di **log** generati **giornalmente**. Essi sono anche **persistenti** nel sistema attraverso l'utilizzo di **bind-mount.**
- Il secondo viene lanciato a *run-time* dal primo ed avvia un *server gRPC*. Esso espone tramite *Remote Procedure Call* alcune funzioni che permettono di:
    - dato un **SLA Set**, calcola il numero di violazioni.
    - dato un **SLA Set**, predice il numero di violazioni.

Prometheus espone metriche provenienti da diverse sorgenti, tra cui un ***node_exporter***. Una volta recuperate, vengono elaborate delle statistiche, tra cui: **MAX**, **MIN**, **AVG**, **DEV_STD** e **predizioni future**. Tutti questi dati vengono poi inviati ad un ***Producer*** e, tramite ***broker Kafka***, vengono scritti su un *topic* ***'prometheusdata'.***

Per quanto riguarda il secondo processo, ecco un elenco delle *Remote Procedure Call:*

```
rpc getNumberOfViolationsPast (listMetricsParam)
                                returns(resultValue) {}
```

Funzione che, dato un SLA set di metriche, calcola il numero di violazioni nel passato a 1h, 3h e 12h

```
rpc getNumberOfViolationsFuture (listMetricsParam) returns (resultValue) {}
```

Funzione che, dato un SLA set di metriche, calcola il numero di violazioni nel futuro a partire dalle predizioni calcolate

In particolare, questo processo attende i dati provenienti dal primo, e se essi sono pronti e se l' **SLA Manager** richiede il numero di violazioni, esse vengono calcolate e inoltrate. Se queste non sono pronte, semplicemente non verranno inviate.

Considerazioni Importanti

Le considerazioni durante la progettazione di questo microservizio sono state molte durante le iterazioni. La scelta di affidare la computazione a più *Thread* è stata fatta per avere un guadagno in termini di ***performance*** e quindi un calcolo dei dati più veloce. L'implementazione di più processi invece ha richiesto notevole tempo per capire come gestire la comunicazione tra di essi (si è optato infine per due **code** di tipo ***`SimpleQueue`***) e come queste dovessero interfacciarsi tramite le funzioni ***gRPC***. Per quanto riguarda ***Kafka***, sarebbe stato opportuno creare più **partizioni** per lo stesso topic, con un numero di partizioni pari al numero di metriche, e distribuire in ciascuna di esse le informazioni relative a tali metriche. Questa implementazione è stata **iniziata** ma non è stata conclusa, è possibile vedere questa implementazione a questo link, dove vengono mostrate le differenze tra il branch main e quello di questa feature:
https://github.com/matteopidone/DSBD-Project/compare/matteo/feat/admin-client?expand=1

4.2 - Data Storage

Al *Data Storage* viene affidato l'incarico di gestire *Consumer* che si sottoscrivono al *topic*. Una volta raccolti i dati dal *broker*, vengono eseguite delle procedure di scrittura sul *database* ***MySQL***. Inoltre il microservizio fa anche da server **gRPC sincrono** in modo da rendere accessibili agli altri microservizi i dati all'interno di esso.

Il Database ***Mysql*** contiene le informazioni associate alle metriche, calcolate dall' *ETL Data Pipeline.* Le tabelle principali sono:

- `METRICHE (ID, NOME, METADATA)`
- `STATISTICHE (ID, NOME)`
- `STATISTICHE_METRICHE (ID_METRICA, ID_STATISTICA, 1H, 3H, 12H)`
- `PREDIZIONI_METRICHE (ID_METRICA, ID_STATISTICA, VALORI)`

Queste tabelle vengono aggiornate una volta arrivati i valori aggiornati, si è optato per scartare i valori vecchi, ritenuti appunto **obsoleti**.
Il microservizio, tramite **server gRPC sincrono,** espone le seguenti *Remote Procedure Call*:

`rpc getAllMetrics (emptyParam) returns (resultValue) {}`

Funzione che permette di ricevere tutte le metriche presenti.

`rpc getAllStatistics(emptyParam) returns (resultValue) {}`

Funzione che permette di ricevere tutte le statistiche presenti.

`rpc getMetadataForMetrics (idMetricParam) returns(resultValue) {}`

Dato l'ID di una metrica, restituisce i metadati (autocorrelazione, stazionarietà, stagionalità) associati alla metrica.

`rpc getHistoryForMetrics (idMetricParam) returns (resultValue) {}`

Dato l'ID di una metrica, restituisce massimo, minimo, media e dev_std.

`rpc sendStats (statsNameParam) returns (resultValue) {}`

Funzione che permette di inserire statistiche all'interno del servizio.

`rpc sendMetrics (statsNameParam) returns (resultValue) {}`

Funzione che permette di inserire metriche all'interno del servizio.

`rpc getPredictionForMetrics(idMetricParam) returns (resultValue) {}`

Dato l'ID di una metrica, restituisce la predizione dei valori per i succesivi 10 minuti.

Considerazioni Importanti

Su questo microservizio sono state fatte molte considerazioni. In particolare un'ampia discussione è nata relativamente alla gestione del **database**. In una prima iterazione, era stato previsto di utilizzare l'immagine di ***`mysql`*** come immagine base del microservizio. È possibile vedere questa implementazione nella seguente Pull Request: https://github.com/matteopidone/DSBD-Project/pull/5/files#diff-ceb87235745b0ff8943741a2dbc7d27a3ad65ff162898bab59f37f152e72007d

Successivamente è stato scelto di separare in un microservizio differente il database stesso ed affidare al **Data Storage** la gestione di esso. Durante le ultime iterazioni però ci sono stati ripensamenti riguardo questa scelta: da un lato abbiamo la separazione fisica tra la gestione e l'accesso ai dati, dall'altro lato però questi dati hanno senso solo per il **Data Storage** e solo lui ha accesso ad essi.

4.3 - Data Retrieval

Il terzo microservizio si occupa di mostrare tutti i dati presenti nel *database* all'utente. Con l'utilizzo del **Framework *Flask*** viene esposta un'interfaccia grafica (accessibile dal *browser* alla porta ***9003***) per la visualizzazione delle informazioni recuperate. Il microservizio instaura una connessione **sincrona** verso il server ***gRPC*** presente nel **Data Storage**, per il recupero dei dati. Il **Framework** espone numerose route tra cui :

- **"/"** - *Route* di default in cui vengono visualizzate tutte le metriche disponibili e dei link verso le altre route;
- **"/<id_metric>/metadata"** - *Route* che mostra i metadati relativi ad una specifica metrica (autocorrelazione, stazionarietà e stagionalità);
- **"/<id_metric>/history"** - *Route* che mostra tutte le statistiche relative ad una specifica metrica nelle ultime *1h*, *3h* e *12h*;
- **"/<id_metric>/prediction"** - *Route* che mostra le predizioni relative ad una specifica metrica nei successivi *10 min*;

4.4 - SLA Manager

Il quarto microservizio permette di definire un set di metriche, con i relativi range di valori associati, all'interno di un **Service Level Agreement** (**SLA Set)**. Anche in questo microservizio è stato utilizzato il **Framework Flask** per offrire all'utente un'interfaccia grafica per la visualizzazione delle informazioni (accessibile dal *browser* alla porta ***9004***).
È possibile ottenere il numero di violazioni dell' **SLA** che si sono verificate in passato nelle ultime *1h*, *3h* e *12h* e anche il numero di possibili violazioni nel futuro, sulla base delle predizioni calcolate dall' **ETL Data Pipeline**.
L' **SLA Manager** comunica con il **server gRPC** presente nel **Data Storage**, per ottenere le informazioni riguardanti le metriche e le statistiche disponibili. Esso comunica anche con il **server gRPC** presente nell' **ETL Data Pipeline** inviando l' **SLA** su cui verranno calcolate le violazioni. Il **Framework** espone numerose route tra cui :

- **"/pastViolation"** - *Route* che mostra tutte le metriche disponibili e le statistiche su cui poter definire l' **SLA** e verificare le violazioni nel passato;
- **"/submitPastViolations" - POST** - *Route* (accessibile **solo** tramite metodo **POST**) che mostra le violazioni relative alle metriche specificate nell' **SLA** nelle ultime *1h*, *3h* e *12h*;
- **"/futureViolation**" - *Route* che mostra tutte le metriche disponibili e le statistiche su cui poter definire l' **SLA** e verificare le possibili violazioni nel futuro;
- **"/submitFutureViolations" - POST** - *Route* (accessibile **solo** tramite metodo **POST**) che mostra le possibili violazioni future relative alle metriche specificate nell' **SLA**;