Distributed Systems and Big Data

Relazione Finale

Matteo Pidone - 1000042321
Tomas Prifti - 1000040388

Introduzione

Introduzione

L’obiettivo del progetto è la realizzazione di un sistema che monitori alcune metriche estratte da un ***server Prometheus***. 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*.

Microservizi

- ***ETL data pipeline***, un microservizio che calcola metadati, statistiche e predizioni. I risultati vengono poi inviati ad un **topic Kafka**. 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.

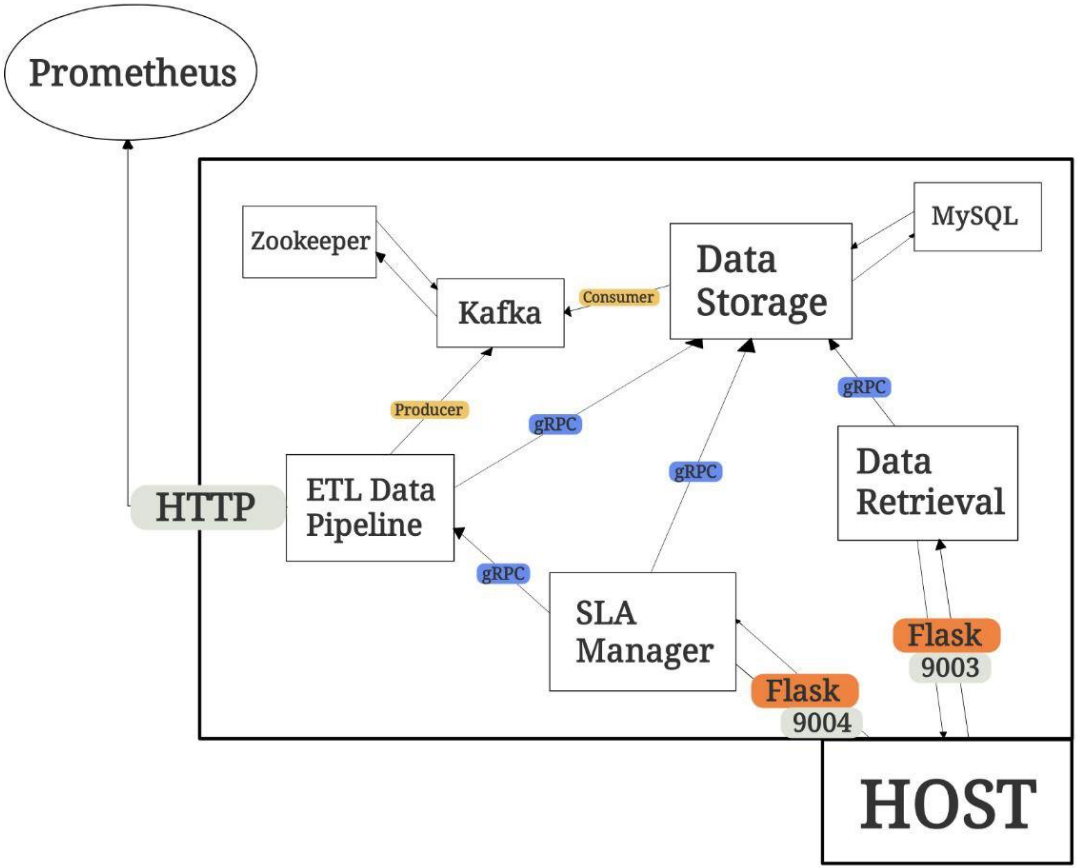
- ***Data Retrieval***, un microservizio che offre un'interfaccia *REST* per la visualizzazione delle informazioni ricavate dai precedenti microservizi.

- ***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 sul numero di violazioni delle metriche e sulle possibili violazioni future. Queste informazioni vengono poi mostrate attraverso un'interfaccia REST all'utente.

Schema Architetturale

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.



ETL Data Pipeline

ETL Data Pipeline

Il primo microservizio recupera le metriche da un *server Prometheus*

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.

ETL Data Pipeline

Computazione del primo processo:

Statistiche

- Media, massimo, minimo, deviazione standard

Metadati

- Stazionarietà, calcolata tramite test ***Augmented Dickey–Fuller***
- Stagionalità, calcolata tramite ***Seasonal Decompose,*** modello additivo
- Autocorrelazione, calcolata tramite ***Autocorrelation Function***

Predizioni

- Calcolate tramite funzione ***Exponential Smoothing***

ETL Data Pipeline

Remote Procedure Call del secondo processo*:*

```
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

Il primo processo comunica con il secondo tramite code di messaggi. In particolare, quando vengono recuperate le metriche, vengono inviate a quest'ultimo.
Quando l' **SLA Manager** richiede il numero di violazioni, esse vengono calcolate dal secondo processo ed inoltrate.

Data Storage

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)`

Data Storage

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.

Data Storage

Il microservizio, tramite **server gRPC sincrono,** espone le seguenti *Remote Procedure Call*:

```
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.

Data Retrieval

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*;

SLA Manager

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**;