Università di Catania

Dipartimento di Ingegneria Elettrica Elettronica e Informatica (DIEEI)

Corso di Laura Magistrale in Ingegneria Informatica LM32

Relazione del progetto in itinere del corso di Distributed Systems and Big Data

Fabiola Marchi' 1000042233

Matteo Terranova 1000043914

A.A. 2022/2023

Sommario

1. Abstract

L'obiettivo dell'elaborato è la realizzazione di una applicazione, costituita da diversi microservizi, che sia in grado di esporre delle metriche attraverso il *server Prometheus*.

Nel caso in esame, il *Prometheus server* è stato fornito dal Prof. Ing. Morana al seguente link http://15.160.61.227.29090.

Nello scenario implementato, *ETL_Data_Pipeline.py* permette di analizzare delle metriche, in particolare sono state individuate *cpuLoad, cpuTemp, diskUsage, availableMem* e *networkThroughput*.

I risultati dell'analisi sono stati inviati su di un *topic Kafka* –"*prometheusdata*"- per poter essere processati da *DataStorage.py* ed in seguito inviati ad un database *SQL*.

È stato creato, inoltre, un sistema di monitoraggio interno tramite *REST API.*

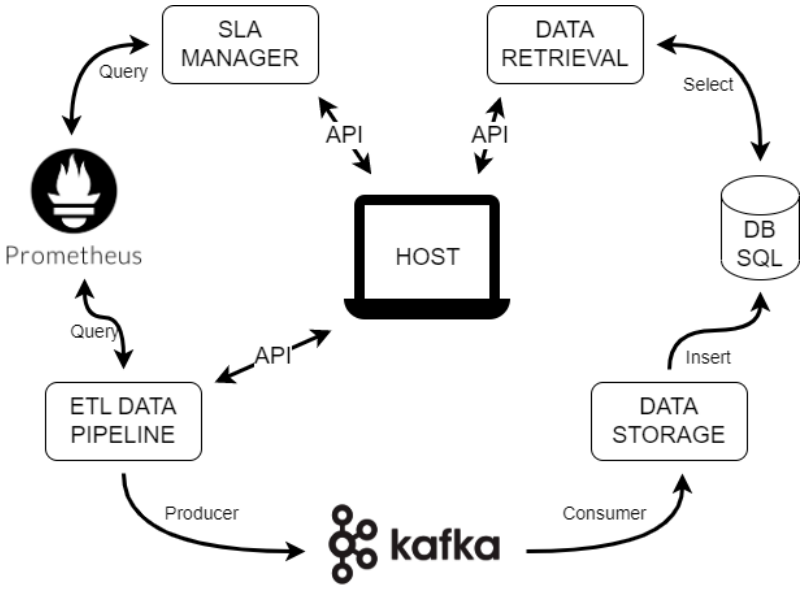

Figura 1 - Diagramma dei microservizi

Tale sistema rende disponibili i dati mediante delle *query* effettuate da *DataRetrival.py* sul database.

1.1 Comandi da eseguire

Di seguito sono elencati i comandi necessari all'esecuzione dell'elaborato:

- Eseguire *docker*

- Da *command prompt:*

  - creare la *network* "marchiterranova"
    - `docker network create marchiterranova`

  - verificare la presenza della *network* appena creata:
    - `docker network ls`

  - nella *directory ./Service* lanciare:
    - `docker-compose up – d`

- creare le tabelle del database:

  - estrapolare l'ID_SQL_CONTAINER mediante:
    - `docker ps`

  - eseguire:
    - `docker exec -it ID_SQL_CONTAINER bash`
    - `mysql -u root -p` ⟶ psw***: root

- Popolare il db:

```
CREATE TABLE 1hMetrics (ID INT AUTO_INCREMENT, metric
varchar(255), max DOUBLE,min DOUBLE,mean DOUBLE,std
DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 1hAutocorrelation (ID INT AUTO_INCREMENT,
metric varchar(255),value DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 1hStationarity (ID INT AUTO_INCREMENT, metric
varchar(255),adf DOUBLE,pvalue DOUBLE,usedlag DOUBLE,nobs
DOUBLE,criticalvalues varchar(255),icbest DOUBLE,PRIMARY
KEY(ID));

CREATE TABLE 1hSeasonability (ID INT AUTO_INCREMENT, metric
varchar(255),value DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 1hPrediction (ID INT AUTO_INCREMENT, metric
varchar(255), max DOUBLE,min DOUBLE,mean DOUBLE,PRIMARY
KEY(ID));

CREATE TABLE 3hMetrics (ID INT AUTO_INCREMENT, metric
varchar(255), max DOUBLE,min DOUBLE,mean DOUBLE,std
DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 3hAutocorrelation (ID INT AUTO_INCREMENT,
metric varchar(255),value DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 3hStationarity (ID INT AUTO_INCREMENT, metric
varchar(255),adf DOUBLE,pvalue DOUBLE,usedlag DOUBLE,nobs
DOUBLE,criticalvalues varchar(255),icbest DOUBLE,PRIMARY
KEY(ID));
```

```
CREATE TABLE 3hSeasonability (ID INT AUTO_INCREMENT, metric
varchar(255),value DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 3hPrediction (ID INT AUTO_INCREMENT, metric
varchar(255), max DOUBLE,min DOUBLE,mean DOUBLE,PRIMARY
KEY(ID));

CREATE TABLE 12hMetrics (ID INT AUTO_INCREMENT, metric
varchar(255), max DOUBLE,min DOUBLE,mean DOUBLE,std
DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 12hAutocorrelation (ID INT AUTO_INCREMENT,
metric varchar(255),value DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 12hStationarity (ID INT AUTO_INCREMENT, metric
varchar(255),adf DOUBLE,pvalue DOUBLE,usedlag DOUBLE,nobs
DOUBLE,criticalvalues varchar(255),icbest DOUBLE,PRIMARY
KEY(ID));

CREATE TABLE 12hSeasonability (ID INT AUTO_INCREMENT,
metric varchar(255),value DOUBLE,PRIMARY KEY(ID));

CREATE TABLE 12hPrediction (ID INT AUTO_INCREMENT, metric
varchar(255), max DOUBLE,min DOUBLE,mean DOUBLE,PRIMARY
KEY(ID));
```

- Nella *directory DSBproject*:
    - Effettuare il *compose* globale:
        - `docker-compose up – d`

Eseguiti i comandi mostrati sopra, tutti i microservizi implementati sono in running.

L'*output* delle *query* è visionabile su qualsiasi applicazione che permetta di eseguire richieste *GET* e *POST*.

È consigliabile utilizzare l'estensione di *Google Chrome Talend API Tester* ed inserire i seguenti URL:

ETL DATA Pipeline

- GET: http://localhost:5000/all
    - restituisce tutti I dati monitorati

- GET: http://localhost:5000/performance
    - restituisce I valori di massimo, minimo, media e deviazione standard calcolate sul set di metriche; la predizione di massimo, minimo e media per i successivi dieci minuiti ed i valori di stazionarietà, stagionalità e autocorrelazione.

- POST: http://localhost:5000/SLAset
    - richiesta POST per aggiornare il set di metriche di cui effettuare la predizione.

Data Retrival

- GET: http://localhost:5050/all
    - Restituisce tutti I dati monitorati presenti nel database

- GET: http://localhost:5050/metrics
  - Resituisce tutte le metriche divise per tempo di monitoraggio (1h, 3h e 12h)

- GET: http://localhost:5050/metrics/<name>
  - Resituisce tutte le metriche divise per tempo di monitoraggio (1h, 3h e 12h) e per nome

- GET: http://localhost:5050/autocorrelation
  - Restituisce i valori di autocorrelazione divisi per tempo di monitoraggio (1h, 3h e 12h)

- GET: http://localhost:5050/autocorrelation/<name>
  - Restituisce i valori di autocorrelazione divisi per tempo di monitoraggio (1h, 3h e 12h) e per nome

- GET: http://localhost:5050/stationarity
  - Restituisce i valori di stazionarietà divisi per tempo di monitoraggio (1h, 3h e 12h)

- GET: http://localhost:5050/stationarity/<name>
  - Restituisce i valori di stazionarietà divisi per tempo di monitoraggio (1h, 3h e 12h) e per nome

- GET: http://localhost:5050/seasonability
  - Restituisce i valori di stagionalità divisi per tempo di monitoraggio (1h, 3h e 12h)

- GET: http://localhost:5050/seasonability/<name>
  - Restituisce i valori di stazionarietà divisi per tempo di monitoraggio (1h, 3h e 12h) e per nome

- GET: http://localhost:5050/prediction
  - Restituisce i valori della predizione di massimo, minimo e media divisi per tempo di monitoraggio (1h, 3h e 12h)

- GET: http://localhost:5050/prediction/<name>
  - Restituisce i valori della predizione di massimo, minimo e media divisi per tempo di monitoraggio (1h, 3h e 12h) e per nome

SLA Manager

- POST: http://192.168.0.12:5100/SLA_Manager
  - Inserendo il seguente *JSON*
    - {
      - "availableMem": [0, 94.33874],
      - "cpuLoad": [0, 1.5],
      - "cpuTemp": [0, 38],
      - "diskUsage": [0, 21.7735],
      - "networkThroughput": [0, 0.01]

      }

- GET: http://192.168.0.12:5100/show_Violation
  - Restituisce la lista delle violazioni

- GET: http://192.168.0.12:5100/Violations_Number
  - Restituisce il numero di violazioni verificatesi

- GET: http://192.168.0.12:5100/SLA_status
  - Restituisce il nome della metrica e il numero di violazioni avvenite divise per tempo (1h, 3h e 12h)

- GET: http://192.168.0.12:5100/predict_Violations
  - Restituisce la lista delle violazioni future nei successivi dieci minuti

- GET: http://192.168.0.12:5100/predict_Violations&Number
  - Restituisce il numero di violazioni future nei successivi dieci minuti

1. Soluzioni implementative

In questo paragrafo vengono esposte le soluzioni utilizzate per soddisfare le richieste dell'elaborato commissionato.

1.1 ETL Data Pipeline

È stata richiesta la creazione di un microservizio, che per ogni metrica esposta, calcoli un *set* di *metadati* con i relativi valori di autocorrelazione, stazionarietà e stagionalità.

Il microservizio è, inoltre, in grado di predire i valori di massimo, minimo, media e deviazione standard per una, tre e dodici ore del set di metriche selezionato.

I valori sopraesposti, mediante un *topic Kafka*, definito "*Prometheusdata*", sono inviati ad un *consumer*, il *datastorage*, e ad un *database* SQL, *test*_DSB.

Un sistema di monitoraggio interno, tramite *REST API*, rende possibile visionare i risultati delle *query* ed il tempo di computazione delle funzioni.

1.1.2 Implementazione

Il file *ETL_DataPipeline.py* funge da *producer* connettendosi al *broker* di *Kafka*.
Viene effettuata una *query* al *server Prometheus* per estrapolare le metriche da monitorare.
Le metriche selezionate sono cinque, in particolare *cpuLoad, cpuTemp, diskUsage, availableMem* e *networkThroughput*, del job '*summary'*, nodo 'sv122'.

L'*output* della *query* è impacchettato in un *dataframe*, sul quale vengono effettuate i calcoli richiesti.

I metodi della libreria di *Pandas* sono stati usati per calcolare massimo, minimo, media e deviazione standard; la funzione *acf* della libreria *Statsmodels* per calcolare l'autocorrelazione; la funzione *adfuller* della libreria *Dickey-fuller* per la stazionarietà; *seasonal*_decompose per la stagionalità.

Viene calcolata una predizione relativa al massimo, al minimo e alla media per i dieci minuti successivi alla chiamata, attraverso la funzione *ExponentialSmoothing* della libreria *Statsmodels*.
Le metriche di cui effettuare la predizione vengono definite in una lista, è possibile modificarle mediante una funzione *REST* o modificato lo *SLA set* nello *SLA manager*.

I valori calcolati vengono convertiti in formato *JSON* per essere inviati nel *prometheusdata*.

Il sistema di monitoraggio interno creato sfrutta un *framenetowrk* di *python Flask*.
Le REST API sono disponibili ai *link* sopra riportati.

1.2 Data Storage

1.2.1 Implementazione

Il file *DataStorage.py* funge da consumer connettendosi al database *mySQL test_DSB* e a *Kafka*. Vengono prelevati soltanto i dati prodotti successivamente all'avvio del consumer.

Mediante una funzione di *polling* vengono letti i dati inseriti nel *topic*, tali dati sono inseriti nel *database*.

Le tabelle presenti nel *database* sono:

- *1hMetrics*: che contiene i valori di massimo, minimo, media e deviazione standard calcolati per il set di metriche in un'ora

- *1hAutocorrelation*: che contiene il valore dell'autocorrelazione calcolata nel set di metriche in un'ora
- *1hStationarity*: che contiene il valore della stazionarietà calcolata nel set di metriche in un'ora
- *1hSeasonability*: che contiene il valore della stagionalità calcolata nel set di metriche in un'ora
- *1hPrediction*: che contiene il valore delle predizioni dii massimo, minimo e media calcolati nel set di metriche in un'ora
- *3hMetrics*: che contiene i valori di massimo, minimo, media e deviazione standard calcolati per il set di metriche in tre ore
- *3hAutocorrelation*: che contiene il valore dell'autocorrelazione calcolata nel set di metriche in tre ore
- *3hStationarity*: che contiene il valore della stazionarietà calcolata nel set di metriche in tre ore
- *3hSeasonability*: che contiene il valore della stagionalità calcolata nel set di metriche in tre ore
- *3hPrediction*: che contiene il valore delle predizioni dii massimo, minimo e media calcolati nel set di metriche in tre ore
- *12hMetrics*: che contiene i valori di massimo, minimo, media e deviazione standard calcolati per il set di metriche in dodici ore
- *12hAutocorrelation*: che contiene il valore dell'autocorrelazione calcolata nel set di metriche in dodici ore

- *12hStationarity*: che contiene il valore della stazionarietà calcolata nel set di metriche in dodici ore
- *12hSeasonability*: che contiene il valore della stagionalità calcolata nel set di metriche in dodici ore
- *12hPrediction*: che contiene il valore delle predizioni dii massimo, minimo e media calcolati nel set di metriche in dodici ore

1.3 Data Retrival

Il microservizio *DataRetrival.py* permette di effettuare delle query al *database* mediante il *framework Flask* di *python* per estrapolare i dati generati da *ETL DataPipeline.*

1.3.1 Implementazione

Le *REST API* sono disponibili ai *link* sopra riportati.

1.4 SLA Manager

Il microservizio *SLA_Manager.py* è in grado di gestire un *set* di cinque metriche con i relativi *range* di valori ammessi e restituisce, tramite *REST API*, l'eventuale numero di violazioni di tali valori nelle ultime una, tre e dodici ore.

Permette, inoltre, di predire le possibili violazioni nei successivi dieci minuti.

1.4.1 Implementazione

Le metriche analizzate, SLA set, sono inviate mediante una funzione *POST*, utilizzando il *framework Flask* di *python*.

Lo SLA set aggiorna il set di metriche da predire nell'ETL DataPipeline.

Le *REST API* sono disponibili ai *link* sopra riportati.

2. Docker Compose

È bene sottolineare che tutti i microservizi implementati sono stati "*containerizzati*" mediante il *software Docker*.

I *docker compose* creati sono due; il primo per contenere tutti i servizi necessari quali *Kafka*, *Zookyper* e *mySQL*; il secondo per contenere i restanti microservizi che eseguiranno i *Dockerfile* per lanciare il codice.

I *compose* forniscono i file di configurazione necessari per la corretta esecuzione dell'elaborato.

I container scambiano informazioni mediante una *network*, "*marchiterranova*", dedicata.

Tale soluzione implementativa è stata scelta affinché si abbia una duplice modalità di esecuzione del codice, mediante *docker* o da terminale.