UNIVERSITÀ
degli STUDI
di CATANIA

RELAZIONE FINALE

Progetto prova in itinere DSBD

CORSO DI

DISTRIBUTED SYSTEMS AND BIG DATA

Samuele Baiomazzola - 1000022651

Anno Accademico 2022-2023

Abstract

L'idea sviluppata consiste nella creazione di un'architettura di microservizi per l'esposizione, il calcolo e la memorizzazione di metriche utilizzando Prometheus exporter, Apache Kafka e un database MySQL.

Utilizza tre microservizi: ETL data pipeline, Data Storage e Data Retrieval.

Il microservizio ETL data pipeline è il cuore del sistema, in quanto è responsabile del calcolo di un set di metadati con i relativi valori tra cui autocorrelazione, stazionarietà e stagionalità, nonché del calcolo di valori di max, min, avg e dev_std per diverse finestre temporali. Inoltre, utilizza un algoritmo di previsione per predire i valori di max, min e avg per un set ristretto di metriche nei prossimi 10 minuti.

Utilizza Apache Kafka e Zookeeper per far comunicare il producer, creato da ETL data pipeline, e il consumer, avviato dal Data Storage.

I dati calcolati da ETL data pipeline vengono quindi inoltrati a un topic Kafka "prometheusdata" per essere memorizzati dal microservizio Data Storage in un database MySQL tramite un servizio Adminer.

Infine, il microservizio Data Retrieval offre un'interfaccia REST API per l'estrazione di dati strutturati dal database.

Inoltre, il sistema include un sistema di monitoraggio interno che rende visibile il tempo di esecuzione delle varie funzionalità attraverso tre file di LOG.

In caso di sviluppo interno di ETL, il diagramma dei microservizi e delle loro comunicazioni potrebbe variare, ma l'obiettivo generale sarebbe lo stesso: esporre, elaborare e memorizzare dati di metriche in modo efficiente.

Sommario

Introduzione

L'applicazione descritta è un sistema di monitoraggio e analisi delle metriche esposte attraverso un Prometheus exporter. L'architettura è composta da tre microservizi: ETL data pipeline, Data Storage e Data Retrieval.

Il microservizio ETL data pipeline esegue calcoli su un set di metadati e sui valori delle metriche, come la massima, minima e media delle metriche per 1h, 3h e 12h.

Inoltra, inoltre, un messaggio contenente i valori calcolati in un topic Kafka denominato "promethuesdata".

Il microservizio Data Storage avvia un consumer group del topic "promethuesdata" e memorizza i valori calcolati in un database MySQL utilizzando il servizio Adminer.

Il microservizio Data Retrieval, infine, offre un'interfaccia REST API per estrarre le informazioni generate dall'applicazione ETL in modo strutturato tramite query.

Il sistema inoltre include un sistema di monitoraggio interno che rende visibile tramite file di LOG il tempo necessario all'esecuzione delle varie funzionalità.

Il tutto è supportato da Apache Kafka e Zookeeper per la comunicazione tra i vari microservizi.

Schema architetturale

L'architettura generale dell'applicazione sarebbe composta da diversi *container* **Docker**:

ETL data pipeline: questo microservizio esegue il calcolo dei metadati, dei valori statistici (come max, min, avg, dev_std) e predice il valore di max, min, avg per un set di 5 metriche. I dati vengono quindi inviati in un topic Kafka "prometheusdata".

Un container per **Apache Kafka** e uno per **Zookeeper**, che gestiscono la comunicazione tra il producer e il consumer.

Data Storage: questo microservizio utilizza un consumer group per leggere i messaggi dal topic "prometheusdata" e memorizza i valori calcolati in un DB MySQL e gestito attraverso Adminer.

Data Retrieval: questo microservizio esposto attraverso un'interfaccia REST API permette di estrarre le informazioni memorizzate nel database in modo strutturato attraverso delle query.

Prometheus Exporter: questo componente esposto come un endpoint http esporta le metriche dell'applicazione in un formato accessibile al server Prometheus.

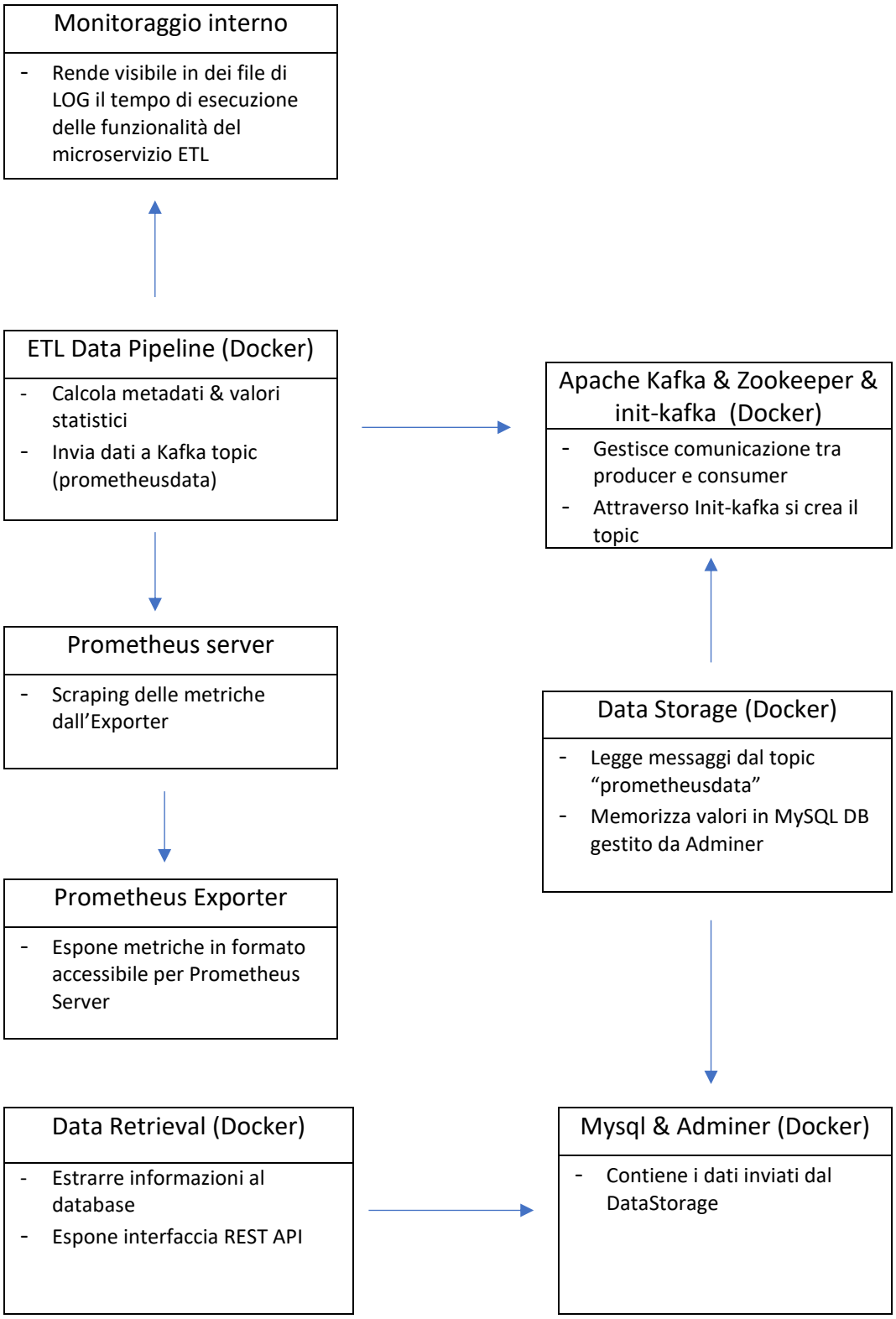
Prometheus Server: il server Prometheus configurato per fare scraping delle metriche esposte dall'exporter.

Monitoraggio interno: questo sistema rende visibile tramite file di LOG il tempo necessario all'esecuzione delle varie funzionalità ed è implementato all'interno di ETL.

Tutti i componenti dell'applicazione sarebbero contenuti in contenitori Docker separati e comunicherebbero tra loro attraverso network Docker.

Inoltre, l'applicazione utilizza un sistema di orchestrazione dei container, come Docker Compose, per gestire la creazione e la distribuzione dei container su un cluster di nodi.

Diagramma dell'applicazione



Descrizione del diagramma

In generale, osservando il diagramma si nota che:

Il microservizio ETL data pipeline calcola i metadati e i valori di metriche e li invia al topic Kafka "promethuesdata".

Il microservizio Data Storage riceve i messaggi dal topic Kafka "promethuesdata" e li memorizza nel DB MySql.

Il microservizio Data Retrieval espone un'interfaccia REST API per estrarre le informazioni dal DB.

Il sistema interno di monitoraggio raccoglie i tempi di esecuzione dei metadati dei valori di metriche e delle predizioni dall'ETL data pipeline e li espone attraverso file di LOG.

Tutti i microservizi sono esposti tramite un Prometheus Exporter, che raccoglie le metriche e le esporta in un endpoint specifico per l'analisi e la visualizzazione tramite Prometheus.

Istruzioni per BUILD & DEPLOYMENT

Il codice sorgente dell'applicazione è stato scritto utilizzando il linguaggio di programmazione Python e attraverso l'IDE PyCharm si è creato un package contenente il codice sorgente, le risorse e le dipendenze necessarie per l'applicazione.

BUILD

La fase di build consiste nel creare immagini Docker per ciascuno dei microservizi che compongono l'applicazione. Ciò può essere fatto utilizzando il file Dockerfile per ciascun microservizio, che specifica come creare l'immagine del contenitore. In questa fase, i sorgenti del codice, le dipendenze, le configurazioni e le librerie necessarie per ciascun microservizio vengono raccolti e inseriti all'interno dell'immagine del contenitore. Ciò include anche la configurazione delle dipendenze necessarie per l'esecuzione dei microservizi, come **Apache Kafka**, **Zookeeper**, **MySQL** e **Prometheus exporter**. Le dipendenze e le librerie necessarie per ciascun microservizio si trovano all'interno di un file chiamato " *requirements.txt* " e si trova all'interno di ogni package di ogni microservizio.

DOCKER

Se si necessita di provare il progetto con client esterni ai container docker, bisogna andare a modificare il main del microservizio di ETL data pipeline e il main del microservizio di DataStorage.

Nello specifico:

bisogna andare a cercare il file " *main_etl.py* " che si torva all'interno della directory *etl_data_pipeline* e accedervi. Successivamente, cercare la funzione " *kakfaJsonProducer* " e modificare la variabile broker con "*localhost:9092*".

Analogo passaggio da andare a fare per il microservizio datastorage. Quindi, si va a cercare il file *"main_data_storage.py"* che si torva all'interno della directory *data_storage* e accedervi. Successivamente, cercare la configurazione del consumer (definito con la variabile ***c***) e cercare al suo interno *'bootstrap.servers'* e cambiare il suo valore, anche qui, con "*localhost:9092*".

Bisogna andare a cambiare anche il nome dell'host del db associato nei microservizi *data-storage* e *data-retrieval* immettendo l'host esterno che si desidera far comunicare. Ad esempio, se il client esterno è la propria macchina potrebbe essere inserito come host "*localhost*".

DEPLOY

La fase di deploy consiste nell'utilizzo di ***Docker Compose*** per gestire la distribuzione e la configurazione dei contenitori su un singolo host o su più host. In questa fase, viene utilizzato un file *"docker-compose.yml"* (che si trova all'interno del package *dsbd_project* già configurato) per specificare come i vari microservizi devono essere eseguiti e configurati, e quindi si va ad eseguire il comando "*docker-compose up*" per avviare i contenitori.

Vi può essere anche la remota possibilità che se venissero eseguiti contemporaneamente tutti i microservizi, il microservizio *data-storage* e *data-retrieval* potrebbero dare dei problemi di connessione al *dabase*. Questo accade perchè, probabilmente, i due microservizi vengono creati ed eseguiti prima che venga configurato il container *Mysql*. In questo caso, per il data storage attendere che etl mandi i valori al topic e subito dopo dovrebbe eseguirsi il data storage senza problemi. Per, invece, il data retrieval attendere che finisca di inserire i valori il data storage nel db e subito dopo potranno essere disponibili i vari url. In caso se non dovessero ripartire basta eseguire di nuovo il comando *"docker-compose up"* ma stavolta singolarmente per i due

microservizi. Quindi: eseguire i comandi *"docker-compose up data_storage"* e *"docker-compose up data_retrieval"*. In questo modo, non verrà più generato l'errore di connessione al DB.

Inoltre, per quanto riguarda il data retrieval in cui vengono mostrati i vari url associati ad una determinata richiesta api, potrebbe accadere che per la prima esecuzione dell'applicazione accedendo a tali url diano un codice di errore. In questo caso, se dovesse accadere, basta eseguire il run del main del microsevizio data retrieval che si chiama "main_data_retrieval" e cliccare su uno dei url che vengono mostrati e in quel caso dovrebbe ritornare non più errore ma il risultato della della richiesta eseguita. Adesso andando su docker si potrà eseguire il data retrieval e non darà più quel codice di errore.

Per configurare il *server Prometheus* per fare scraping delle metriche esposte dall'exporter si è cercato di configurare Prometheus per connettersi all'endpoint esposto dall'exporter e configurare le regole di scraping delle metriche desiderate.

Per rendere visibile il tempo necessario all'esecuzione delle varie funzionalità è stato configurato un sistema di *log* per il microservizio *ETL data pipeline* in modo che registrino le informazioni sulle prestazioni necessarie. Il risultato dei vari log è scritto in tre file chiamati rispettivamente: "*time_metadata.log*", "*time_predict.log*" e "*time_value.log*".

I file si possono trovare all'interno del package del microservizio *ETL data pipeline.*

Lista delle API implementate

Server Name: *'127.0.0.1:5002'*

Schema URL preferito: *'http'*

- ***Nome funzione:***

  Hello():

  - *Endpoint dell'API:* ***/***
  - Metodo HTTP: GET
  - Descrizione: Home del microservizio
  - URL completo: *http://127.0.0.1:5002/*

- ***Nome funzione:***

  All_metrics():

  - *Endpoint dell'API:* ***/all_metrics***
  - *Method HTTP: GET*
  - *Descrizione: ottengo la QUERY di tutte le metriche disponibili in Prometheus*
  - *URL completo: http://127.0.0.1:5002/all_metrics*

- ***Nome funzione:***

  Autocorrelation_stationarity_seasonality_metadata():

  - *Endpoint dell'API:* ***/autocorrelation_stationarity_seasonality_metadata***
  - *Metodo HTTP: GET*
  - *Descrizione: ottengo la QUERY dei metadati e i relativi valori*
  - *URL completo:* *http://127.0.0.1:5002/autocorrelation_stationarity_seasonality_metadata*

- ***Nome funzione:***

  Max_min_avg_std_metric():

  - *Endpoint dell'API:* ***/max_min_avg_std_metric***
  - *Metodo HTTP: GET*
  - *Descrizione: ottengo la QUERY dei valori max, min, avg,dev_std per le ultime 1,3,12 ore.*
  - *URL completo:* *http://127.0.0.1:5002/max_min_avg_std_metric*

- ***Nome funzione:***

  prediction_of_metric_values():

  - *Endpoint dell'API:* ***/prediction_of_metric_values***
  - *Metodo HTTP: GET*
  - *Descrizione: ottengo la QUERY dei valori predetti*
  - *URL completo:* *http://127.0.0.1:5002/prediction_of_metric_values*