Progetto DSBD 2022-2023

a cura di Luca Cirrone e Salvatore Scandura

Abstract

Per la realizzazione del progetto è stato scelto di effettuare lo scraping di alcune delle metriche esposte dal server Prometheus all'indirizzo http://15.160.61.227:29090 (fornito dai docenti).

In particolare si è scelto un set di ben 5 metriche tra cui: availableMem, cpuLoad, diskUsage, cachedMem e bootTime. La scelta delle metriche è stata effettuata seguendo i criteri del USE method (Utilization, Saturation, Errors) di Brendan Gregg (approccio di monitoraggio con focus a livello macchina).

Sulla base di tali metriche è stato implementato un primo microservizio ETL pipeline con lo scopo di elaborare i dati e trasmetterli ad un broker di messaggi, che saranno poi consumati da un gruppo di consumatori, che si occuperanno dello storage dei dati. Ulteriori microservizi andranno a visualizzare e rendere disponibili tali dati all'utente.

Nella seguente descrizione dei vari microservizi, vengono specificate in rosso le API implementate (se presenti).

Schema architetturale

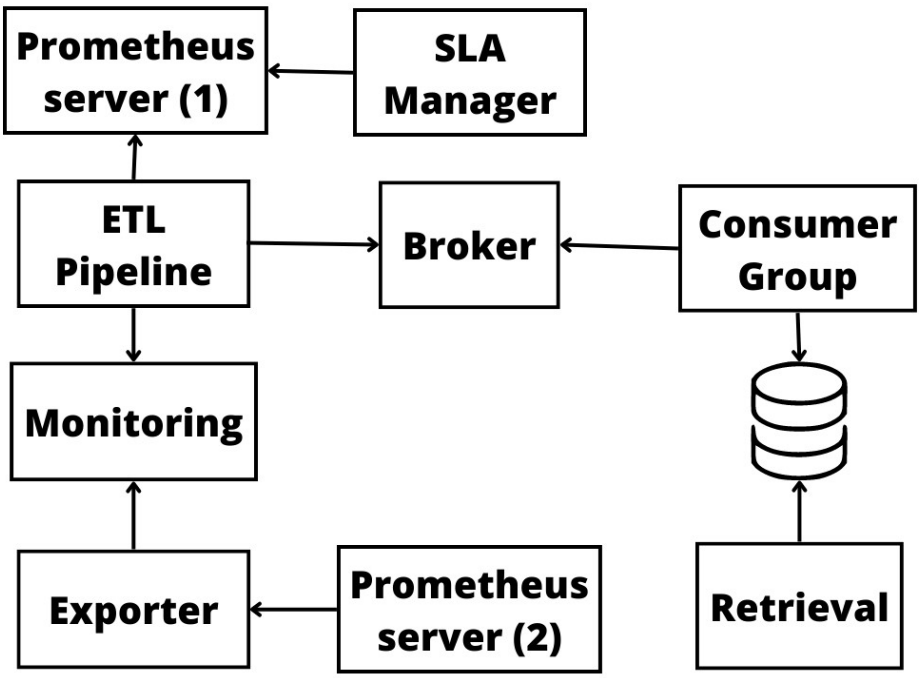
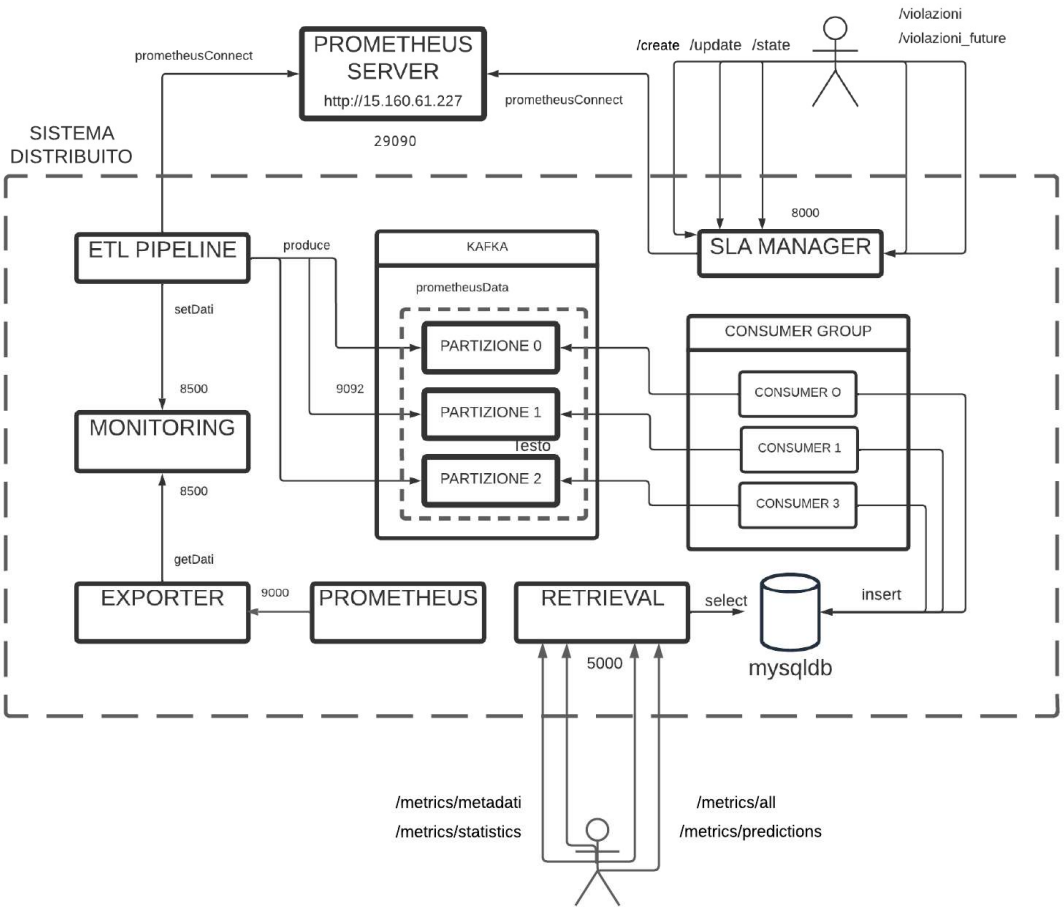


Diagramma microservizi e comunicazioni



ETL Pipeline

Tale microservizio si occupa di reperire le metriche sopra citate dal Prometheus server (1) all'indirizzo http://15.160.61.227:29090. In primo luogo viene effettuato un controllo sulla relativa esistenza del topic "prometheusdata" all'interno del broker Kafka, recuperando l'intera lista dei topic presenti nel broker, tramite un'istanza della classe AdminClient e relativo metodo list_topics(). Qualora il suddetto topic non fosse presente, esso verrà appositamente creato mediante l'oggetto di classe AdminClient e metodo create_topics(), che permette di specificare, oltre al nome del topic, il valore di replication factor e il numero delle partizioni. Avendo scelto in fase di progettazione di utilizzare un topic con 3 partizioni, non è stato possibile affidarsi alla creazione automatica del topic, poiché essa è supportata di default dal broker ma per topic con una singola partizione. Una volta instaurata la connessione con il server Prometheus (1), per ogni metrica viene

utilizzato il metodo get_current_metric_value() per il calcolo dei metadati (stazionarietà, stagionalità e autocorrelazione). Per la stazionarietà è stato utilizzato l'approccio ADFtest mediante il metodo adfuller(). In particolare questi dati verranno trasmessi al topic "prometheusdata" con riferimento alla partizione 0. Successivamente viene utilizzato il metodo get_metric_range_data() per il calcolo dei valori di max, min, avg, dev_std negli intervalli di tempo di 1h, 3h e 12h. Questi dati invece verranno poi trasmessi al topic "prometheusdata" con riferimento alla partizione 1. Infine viene utilizzato sempre il metodo get_metric_range_data() per il calcolo delle predizioni dei valori di max, min, avg nei successivi 10 minuti. Per le predizioni è stato impiegato il modello ARIMA, con calcolo degli iperparametri mediante il metodo auto_arima(). Tali dati verranno trasmessi al topic "prometheusdata" con riferimento alla partizione 2. Inoltre viene calcolato il tempo necessario per generare i dati in 1h, 3h e 12h. Tali tempi di esecuzione vengono poi inviati al microservizio Monitoring.

Broker Kafka

Il broker Kafka è stato implementato utilizzando la configurazione fornita dai docenti, mediante l'ausilio di docker compose. In particolare si fa riferimento all'immagine "confluentinc/cp-kafka:latest" per il broker Kafka e correlato servizio Zookeeper con immagine "confluentinc/cp-zookeeper:latest". Il broker mantiene un topic di nome "prometheusdata" con 3 partizioni, che viene creato dal microservizio ETL Pipeline.

Consumer Group

Tale microservizio si occupa di consumare i dati nelle varie partizioni del topic "prometheusdata" e rendere questi ultimi persistenti grazie all'inserimento in un database MySql. In primo luogo viene avviato un thread che si occuperà di stabilire la connessione con il database e di creare le rispettive tabelle per la persistenza dei dati (tabelle: metadati, stagionalita, correlazione, dataStatistics, dataPredictions). Successivamente vengono avviati 3 thread, uno per ogni consumer, tutti appartenenti al medesimo consumer group "mygroup". Ogni consumer potrà leggere da una e una sola partizione poiché appartenenti allo stesso consumer group. Le partizioni verranno assegnate a ciascun consumer dal consumer group coordinator. Ciò ha permesso di evitare di assegnare manualmente, mediante il metodo consumer.assign(), le partizioni ai vari consumer. Ciascun thread inizialmente crea una nuova connessione con il database ed effettua la

sottoscrizione, con il metodo subscribe(), al topic “prometheusdata”. Successivamente, tramite il metodo poll(), ogni consumer va a reperire dal broker kafka i messaggi relativi alla partizione a cui è stato precedentemente assegnato, per poi effettuare l’inserimento di tali dati nelle rispettive tabelle. Inizialmente viene fatto affidamento alla gestione automatica dei commit (enable.auto.commit=True, di default), ovvero commit asincroni nel ciclo di polling. Il consumatore eseguirà automaticamente il commit degli offset periodicamente all'intervallo impostato da auto.commit.interval.ms (di default, pari a 5 secondi). Tuttavia a causa della presenza di messaggi duplicati inerenti all’ultimo ciclo di esecuzione del thread, dovuti alla chiusura del consumer prima dell’esecuzione del rispettivo commit, si è pensato alle seguenti soluzioni:

- Ridurre auto.commit.interval.ms (inferiore a 5 secondi).
- Adottare una strategia ibrida, eseguendo un commit sincrono, prima di chiudere ciascun consumer (consumer.close()).

È stato scelto di adottare la seconda soluzione, in quanto essa ha fornito un maggior controllo sul commit dell’ultimo offset, risolvendo dunque il problema dei duplicati, ad un leggero discapito prestazionale.

Database MySql

Viene creato il database “metricdb”, a partire dalla definizione dell’environment direttamente all’interno del file docker-compose.yml e relativo volume “metricvol”. Come già detto precedentemente, tale database ospiterà le seguenti tabelle: metadati, stagionalita, correlazione, dataStatistics e dataPredictions. La tabella metadati sarà caratterizzata dai campi: id, metrica, p_value, time. Essa conterrà parte dei dati che verranno pubblicati nella partizione 0, inerenti alla stazionarietà della serie temporale. La tabella stagionalita sarà caratterizzata dai campi: id, metrica, seasonal, time. Essa conterrà parte dei dati che verranno pubblicati nella partizione 0, inerenti appunto alla stagionalità della serie temporale. La tabella correlazione sarà caratterizzata dai campi: id, metrica, corr, time. Essa conterrà parte dei dati che verranno pubblicati nella partizione 0, inerenti appunto all’autocorrelazione della serie temporale. La tabella dataStatistics sarà caratterizzata dai campi: id, metrica, max, min, avg, devstd, durata, time. Essa conterrà i dati che verranno pubblicati nella partizione 1, inerenti ai valori di max, min, avg, devstd delle metriche per 1h, 3h e 12h. La tabella dataPredictions conterrà

i dati che verranno pubblicati nella partizione 2, inerenti alle predizioni dei valori di max, min, avg nei successivi 10 minuti.

Retrieval

Tale microservizio si occupa di fornire un'interfaccia REST, al fine di permettere l'estrazione in modo strutturato dei dati dal database MySql, in esecuzione nel rispettivo container chiamato "mysqldb". In particolare viene implementato un server Flask che espone la porta 5000 sull'host, effettuato mediante port mapping del rispettivo container "retrieval" alla porta 5000 (5000:5000). In primo luogo il microservizio si connetterà al database MySql e tramite le varie query recupererà i dati da visualizzare.

Tali dati verranno visualizzati tramite le seguenti route:

- localhost:5000/metrics/metadati
  permette di visualizzare i dati relativi alla tabella metadati.
- localhost:5000/metrics/statistics
  permette di visualizzare i dati relativi alla tabella dataStatistics.
- localhost:5000/metrics/predictions
  permette di visualizzare i dati relativi alla tabella dataPredictions.
- localhost:5000/metrics/all
  permette di visualizzare tutte le metriche che sono state esaminate nel microservizio ETL Pipeline.

Monitoring

Tale microservizio si occupa di fornire un'interfaccia REST, per visualizzare i tempi di esecuzione delle varie funzionalità, ovvero il tempo impiegato per generare i dati di 1h, 3h e 12h. In particolare viene implementato un server Flask che espone la porta 8500 sull'host, effettuato mediante port mapping del rispettivo container "monitoring" alla porta 8500 (8500:8500).

Sono state implementate le seguenti route:

- localhost:8500/setDati
  questa viene richiamata dal microservizio ETL Pipeline per l'invio dei tempi calcolati, mediante metodo POST.
- localhost:8500/getDati
  permette di visualizzare i dati ricevuti dal microservizio ETL Pipeline.

Exporter

Questo microservizio implementa le funzionalità di un Promethues Exporter. In particolare vengono recuperati i dati dal microservizio Monitoring, mediante route http://monitoring:8500/getDati, trasformandoli in delle metriche di tipo Gauge, utilizzabili dal server Prometheus (2), in esecuzione su un differente container chiamato "prometheus". Mediante l'utilizzo di start_http_server(9000), tali metriche saranno esposte su un server http, che sarà consultato dal server Prometheus (2). Viene inoltre effettuato un port mapping per l'esposizione della porta 9000 del container "exporter" (9000:9000).

Prometheus Server (2)

Tale microservizio è containerizzato ed espone la porta 9090; è stata utilizzata l'immagine "bitnami/prometheus". In particolare viene effettuato un bind mounts (./prometheus-conf/:/etc/prometheus/), al fine di specificare la configurazione del server Prometheus. Mediante command viene eseguito il seguente comando '--config.file=/etc/prometheus/prometheus.yml', che permette dunque di caricare il suddetto file di configurazione. All'interno di tale file prometheus.yml, tra i vari settaggi, viene indicato il target da cui effettuare lo scraping; in questo caso "exporter:9000".

SLA Manager

Questo microservizio si occupa di fornire un'interfaccia REST per l'esecuzione delle varie query. Viene implementato un server Flask, che espone la porta 8000 mediante port mapping (8000:8000). Viene effettuata una connessione al Prometheus server (1), all'indirizzo http://15.160.61.227:29090.

Sono state implementate le seguenti route:

- localhost:8000/create
  permette di definire un set di metriche e, per ogni metrica, il range di valori ammissibili. I precedenti dati verranno trasmessi mediante metodo POST, utilizzando il formato JSON. In particolare le richieste al server sono state testate mediante l'ausilio del tool Postman, facilitando la definizione del body della richiesta http. Per un esempio di body di tale richiesta si faccia riferimento al file body_request.txt.
- localhost:8000/update

permette di aggiornare il set di metriche precedentemente creato. Qualora vi fossero metriche non precedentemente create o denominate in maniera non corretta queste non verranno elaborate. Anche in questo caso è stato utilizzato il tool Postman, in particolare per la definizione del body della richiesta http con metodo POST.

- localhost:8000/state
  ritorna e visualizza il set di metriche che è stato precedentemente creato.
- localhost:8000/violazioni
  viene calcolato il numero delle violazioni occorse relative al set di metriche, in base ai valori ammissibili precedentemente impostati. Ciò in riferimento ai seguenti archi temporali: 1h, 3h e 12h.
- localhost:8000/violazioni_future
  viene calcolato il numero delle violazioni future relative al set di metriche, in base ai valori ammissibili precedentemente impostati. Per le predizioni è stato utilizzato il modulo arimaPredictor.py, che fornisce le predizioni per i successivi 10 minuti.

Build & Deploy

Per ogni microservizio è stata costruita appositamente la relativa immagine mediante Dockerfile. Tutte le immagini sviluppate, utilizzate all'interno del file docker-compose.yml, sono reperibili nel repository Docker hub "lucacirrone", al seguente link: https://hub.docker.com/u/lucacirrone. Altre immagini sono state utilizzate direttamente dal Docker hub.

Per quanto riguarda il deploy è stato scelto di utilizzare docker compose. All'interno del file docker-compose.yml vengono definiti tutti i servizi e le relative configurazioni. Inoltre è stata definita una rete custom di nome "ragnatela" per connettere i vari container.

Il comando per l'esecuzione è il seguente: docker compose up -d --build

Si noti che il pull delle varie immagini da Docker hub potrebbe fallire in funzione del numero massimo di pull legate al corrente account nell'arco di 6 ore.