PROGETTO DSBD

DARIO SAPIENZA
IVAN SCANDURA

27/01/2023

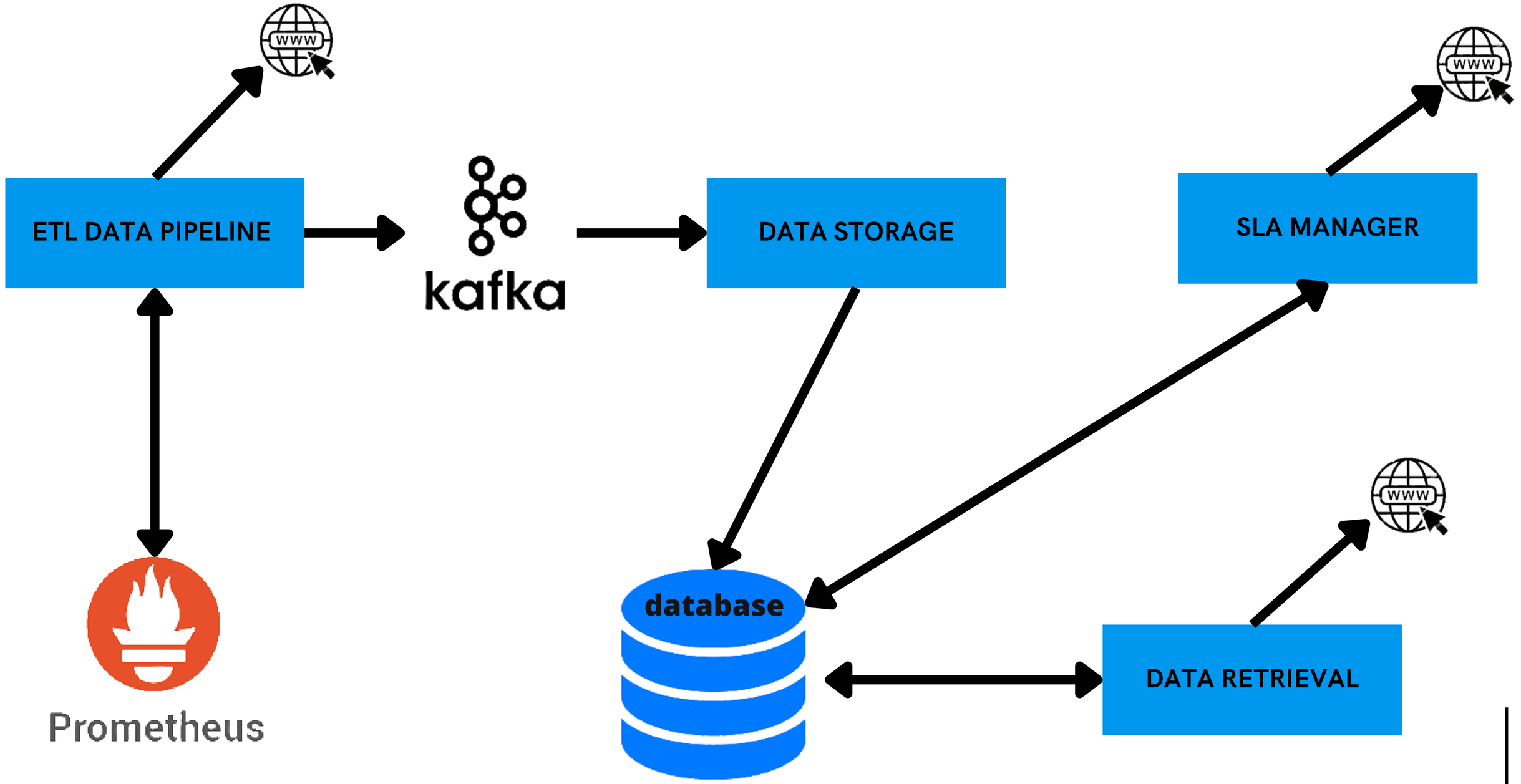
ISTRUZIONI

1 - AVVIARE IL DATABASE DA XAMPP

2 - AVVIARE DOCKER COMPOSE

ABSTRACT

Il progetto presentato si basa sull'idea di un codice che attraverso il collegamento ad un exporter Prometheus riesce ad effettuare operazioni di scraping sulle metriche esposte. Una volta estratte, sulle metriche vengono effettuate operazioni per il calcolo di massimo, minimo, media e deviazione standard in diversi intervalli temporali (1h, 3h, 12h) e operazioni per la valutazione della stazionarietà, della stagionalità e dell'autocorrelazione. Per un set ristretto di metriche vengono inoltre effettuate operazioni di predizione. I valori sopracitati vengono salvati all'interno di un database tramite una connessione mysql.
È anche presente un micro-servizio che permette per un set di 5 metriche di andare ad effettuare operazione relative alla ricerca di eventuali violazioni sia passate che eventualmente future.
Tutti i valori calcolati e le tabelle del database possono essere visualizzati su pagine Web tramite REST API GET.

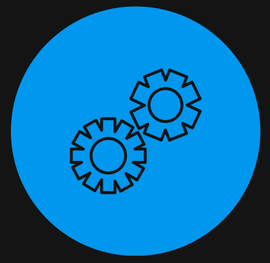

ETL DATA PIPELINE
kafka
DATA STORAGE
SLA MANAGER
database
DATA RETRIEVAL
Prometheus

MICROSERVIZI

ETL DATA PIPELINE

DATA STORAGE

DATA RETRIEVAL



SLA MANAGER

ETL DATA PIPELINE

Questo micro-servizio tramite la funzione “PrometheusConnect” si connette all’ exporter Prometheus.

```
prom = PrometheusConnect(url="http://15.160.61.227:29090/", disable_ssl=True)
```

Una volta effettuato il collegamento la prima funzione invocata è "get_metrics" che si occupa di andare a prelevare da Prometheus una serie di metriche sulla base dei filtri utilizzati. Nel nostro caso l’unico filtro è “job = ceph-metrics” per andare ad estrapolare tutte le metriche ceph. Molte di queste presentano però un valore nullo, quindi prima di andare a prelevare le metriche è stato eseguito un ulteriore controllo su questo valore in modo da analizzare solo quelle metriche con “value !=0”.

```
def get_metrics():
    metric_list = []
    not_zero_metrics = []

    metric_data = prom.get_metric_range_data(
        metric_name='',
        label_config={'job': 'ceph-metrics'}
    )
    print("\nMETRICHE TOTALI: ", len(metric_data))
    for metric in metric_data:
        for p1,p2 in metric["values"]:
            if p2 != '0' and metric["metric"]["__name__"] not in not_zero_metrics:
                not_zero_metrics.append(metric["metric"]["__name__"])

    print("METRICHE CON VALUE !=0 : ",len(not_zero_metrics))
    # algoritmo scelta metriche
    for k in range(0,20):
        metric_list.append(not_zero_metrics[k])

    return metric_list
```

Dopo aver riempito un vettore con le metriche scelte (solo 20 nel nostro caso ma con la possibilità di espandere la ricerca)si prosegue andando a calcolare prima i valori autocorrelazione, stagionalità e stazionarietà tramite la funzione “calculate_values_1” e successivamente i valori di massimo, minimo, media e deviazione standard tramite la funzione “calculate_values_2” andando a considerare i diversi archi temporali (1h, 3h, 12h).

```python
def calculate_values_1(metric_df):
    list = {}
    list = {"autocorrelazione": autocorrelation(metric_df['value']),
            "stazionarietà": stationarity(metric_df['value']),
            "stagionalità": seasonal(metric_df['value'])}

    return list
```

```python
def calculate_values_2(metric_name, metric_df, start_time):
    # Calcoliamo il valore di massimo, minimo e media
    max = metric_df['value'].max()
    min = metric_df['value'].min()
    avg = metric_df['value'].mean()
    std = metric_df['value'].std()

    result_2[str(metric_name + "," + str(start_time))] = {"max": max, "min": min,
                                                          "avg": avg, "std": std}

    return result_2
```

Le funzioni "autocorrelation", "stationarirty" e "seasonal" al loro interno richiamano metodi appartenenti alla libreria <statsmodels> utili per operazioni di questo tipo.

Prima di chiamare i metodi mostrati in figura sono state invocate le funzioni “setting_parameters” per l’impostazione dei parametri sulle metriche e le funzioni “MetricList” e “MetricRangeDataFrame” appartenenti alla libreria <prometheus-api-client> per l’analisi delle metriche.

Per il calcolo della predizione è stata invocata la funzione “predict” che tramite una lista di metriche in ingresso ne va ad eseguire la predizione. Nel nostro caso è stato passato un vettore creato ad hoc per effettuare la predizione su delle metriche presentanti andamenti non banali).

```
def predict(metric_name, metric_df):

    dictPred = {}

    data = metric_df.resample(rule='10s').mean(numeric_only='True')
    tsmodel = ExponentialSmoothing(data.dropna(), trend='add', seasonal='add', seasonal_periods=5).fit()
    prediction = tsmodel.forecast(6) # numero di valori per 10min

    dictPred = {"max": prediction.max(), "min": prediction.min(), "avg": prediction.mean()}

    return dictPred
```

Infine, viene invocata la funzione “kafka_producer” grazie alla quale, passando i file contenenti i valori precedentemente calcolati e la rispettiva chiave permette di comunicare tramite topic Kafka con il micro-servizio “DATA STORAGE” (libreria <confluent-kafka>).

```python
def kafka_producer(file, key):
    broker = "localhost:29092"
    topic = "promethuesdata"
    conf = {'bootstrap.servers': broker}

    #---- Crea l'istanza Produttore ----#
    p = Producer(**conf)

    def delivery_callback(err, msg):
        if err:
            sys.stderr.write('%% Message failed delivery: %s\n' % err)
        else:
            sys.stderr.write('%% Message delivered to %s [%d] @ %d\n' %
                             (msg.topic(), msg.partition(), msg.offset()))

    #---- Legge le linee dalla stdin, produce ogni linea a Kafka ----#
    try:
        record_key = key
        record_value = json.dumps(file)
        print("Producing record: {}\t{}".format(record_key, record_value))
        p.produce(topic, key=record_key, value=record_value, callback=delivery_callback)

    except BufferError:
        sys.stderr.write('%% Local producer queue is full (%d messages awaiting delivery): try again\n' %
                         len(p))
    p.poll(0)

    # Aspetta finche tutti i messaggi sono stati consegnati
    sys.stderr.write('%% Waiting for %d deliveries\n,' % len(p))
    p.flush()
```

Tale micro-servizio, inoltre, prevede un sistema di monitoraggio che tiene traccia dei tempi che sono stati necessari per lo svolgimento delle operazioni sopracitate. Tramite REST API GET prevede la visualizzazione su pagina web dei valori monitorati.

REST API GET:

- get_tt, tempo di esecuzione totale
- get_pt, tempo di esecuzione parziale

```python
@app.get('/')
def Hello():
    return 'HELLO FROM ETL DATA PIPELINE'

@app.get('/t_time')
def get_TT():
    return 'TOTAL TIME:' + str(total_time)

@app.get('/p_time')
def get_PT():
    return 'PARTIAL TIME:' + str(partial_time)
```

DATA STORAGE

Questo micro-servizio si occupa di andare a salvare all'interno di un database MySQL quanto calcolato nel micro-servizio "ETL DATA PIPELINE". Per prima cosa si effettua la connessione al database 'datastorage' tramite la funzione "mysql.connector.connect".

```
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    database="datastorage"
)
```

Successivamente si procede con l'iscrizione al topic Kafka per effettuare la comunicazione.

```
c = Consumer({
    'bootstrap.servers': 'localhost:29092',
    'group.id': 'mygroup',
    'auto.offset.reset': 'latest'
})

c.subscribe(['promethuesdata'])
```

Una volta effettuata l'iscrizione, andando a verificare la chiave del messaggio si chiama la query SQL e si procede con la scrittura dei dati sul database.

```
while True:
    msg = c.poll(1.0)

    if msg is None:
        continue

    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue

    if str(msg.key()) == "b'etl_data_pipeline#2'":
        key_vector.append(msg.key())

        file_json_2 = json.loads(msg.value())
        #print(file_json_2)
        print("FILE JSON 2 STAMPATO\n")

        sql = """INSERT INTO metrics (metric_info, max, min, average, std) VALUES (%s,%s,%s,%s,%s);"""

        for key in file_json_2:
            param1 = key
            param2 = file_json_2[key]["max"]
            param3 = file_json_2[key]["min"]
            param4 = file_json_2[key]["avg"]
            param5 = file_json_2[key]["std"]
            val = (param1, param2, param3, param4, param5)

            print(sql, val)
            mycursor.execute(sql, val)
            mydb.commit()
            print(mycursor.rowcount, "was inserted.")
            mycursor.execute("SELECT * FROM metrics;")

            for x in mycursor:
                print(x)
```

DATA RETRIEVAL

Questo micro-servizio si occupa di andare a visualizzare tramite REST API GET le tabelle del database caricate dal micro-servizio DATA STORAGE.
Per prima cosa si effettua il collegamento al database tramite la funzione “mysql.conncetor.connect”.
Successivamente si vanno a chiamare diverse funzioni che tramite la gestione di query SQL permettono di estrapolare i valori presenti nelle tabelle del database per poi visualizzarli tramite get.

REST API GET:

- get_all_metrics, tutte le metriche
- get_values_predict, max, min, avg, std
- get_values_metadata, autocorrelazione, stagionalità e stazionarietà
- get_values_metrics, predizione

```python
@app.get('/all_metrics')
def get_all_metrics():
    sql = """SELECT * from metadata;"""
    # print(sql)
    mycursor.execute(sql)
    all_metrics = {}
    metric_list = []
    y = 0
    for x in mycursor:
        # print(x)
        metric_list.append(x)
        all_metrics[y] = metric_list[y][0]
        y = y + 1

    return all_metrics
```

SLA MANAGER

Questo micro-servizio definisce uno SLA composto da un set di 5 metriche a scelta tra quelle analizzate e si occupa di andare ad effettuare un'analisi sulle metriche in relazione alle violazioni.

In particolare, abbiamo due funzioni differenti:

- la prima, *"past_violations"* realizzata per la ricerca di eventuali violazioni passate (sempre in relazione agli intervalli temporali definiti nel micro-servizio ETL DATA PIPELINE)
- la seconda, *"fut_violations()"* realizzata per la ricerca di possibili violazioni nell' immediato futuro (10 minuti).

```
def past_violations():
    for key in range(0, len(metrics_sla)):
        num_violations = 0
        for h in range(0, len(start_time)):
            metric_data, t = ETL_D_P_main.setting_parameters(DR_main.get_all_metrics().get(key),
                                                             label_config, parse_datetime(start_time[h]),
                                                             end_time, chunk_size)

            metric_df = MetricRangeDataFrame(metric_data)
            for k in metric_df['value']:
                if not int(SLA_result[DR_main.get_all_metrics().get(key)]['min']) < k < int(
                        SLA_result[DR_main.get_all_metrics().get(key)]['max']):
                    num_violations = num_violations + 1

            print("\n\nViolazioni per metrica: " + DR_main.get_all_metrics().get(key) + " per: " + str(
                start_time[h]) + " -> " + str(num_violations))

            if num_violations > 0:
                state[DR_main.get_all_metrics().get(key)] = "E' STATA VIOLATA"
            else:
                state[DR_main.get_all_metrics().get(key)] = "NON E' STATA VIOLATA"

            violations[DR_main.get_all_metrics().get(key)+","+ start_time[h]] = num_violations

    return violations
```

```
def fut_violations():
    for key in range(0, len(metrics_sla)):
        num_violations = 0

        metric_data, t = ETL_D_P_main.setting_parameters(DR_main.get_all_metrics().get(key),
                                                         label_config, parse_datetime('1h'),
                                                         end_time, chunk_size)
        metric_df = MetricRangeDataFrame(metric_data)
        num_samples = 5
        data = metric_df.resample(rule='10s').mean(numeric_only='True')
        tsmodel = ExponentialSmoothing(data.dropna(), trend='add', seasonal='add', seasonal_periods=5).fit()
        prediction = tsmodel.forecast(steps=round((10 * 60) / num_samples))
        for k in prediction.values:
            if not SLA_result[DR_main.get_all_metrics().get(key)]['min'] < k \
                    < SLA_result[DR_main.get_all_metrics().get(key)]['max']: num_violations = num_violations + 1

        print("\n\nPossibili violazioni per metrica: " + DR_main.get_all_metrics().get(key) + " -> " + str(num_violations))

        future_violations[DR_main.get_all_metrics().get(key)] = {num_violations}

    return future_violations
```

Prima di effettuare la ricerca sulle violazioni viene chiesto di inserire i codici delle metriche da analizzare e gli eventuali range di massimo e minimo da prendere in considerazione. Se si lasciano vuoti tali campi (impostando n quando richiesto) verranno automaticamente presi i valori di massimo e minimo calcolati in precedenza.

```
print("\n- - - Benvenuto nello Sla Manager - - -\n")
print("Inserisci il codice delle 5 metriche che vuoi utillare")

for key in DR_main.get_all_metrics():
    print(str(key) + " - " + DR_main.get_all_metrics().get(key) + "\n")


# Controllo sull'esistenza del codice della metrica
for key in range(0, 5):
    metric = input('\nSeleziona metriche da analizzare: ')
    if metric not in metrics_sla:
        metrics_sla.append(metric)

print(metrics_sla)
for key in range(0, len(metrics_sla)):
    print(DR_main.get_all_metrics().get(key))
    x = input('vuoi inserire tu massimo e minimo? s/n \n')
    if x == 's':
        min = input("inserisci minimo: ")
        max = input("inserisci massimo: ")
        SLA_result[DR_main.get_all_metrics().get(key)] = {'min': min, 'max': max}
    else:
        metric_data, t = ETL_D_P_main.setting_parameters(DR_main.get_all_metrics().get(key),
                                                         label_config, start_time_range,
                                                         end_time, chunk_size)
        metric_df = MetricRangeDataFrame(metric_data)
        SLA_result[DR_main.get_all_metrics().get(key)] = {'min': metric_df['value'].min(),
                                                          'max': metric_df['value'].max()}

    print(SLA_result)
```

Come si può notare dalle figure mostrate in questo micro-servizio le funzioni di impostazione dei parametri e di estrapolazione dal database vengono richiamate dai rispettivi “main” dei micro-servizi ETL DATA PIPELINE e DATA STORAGE.

Anche per questo micro-servizio è prevista una visualizzazione dei valori ottenuti in una pagina web grazie all’utilizzo di REST API GET.

REST API GET:

- get_state, stato delle metriche
- get_violations, numero di violazioni
- get_future_violations, numero di violazioni future

NOTA

È stato creato un file “Docker-compose” al cui interno sono definiti tutti i micro-servizi da creare. Per tutti i micro-servizi sono staTi inoltre creati i rispettivi file “Dockerfile” e “requirements” necessari per l’avvio del compose.