Corso di Distributed Systems & Big Data

Presentazione Progetto in itinere

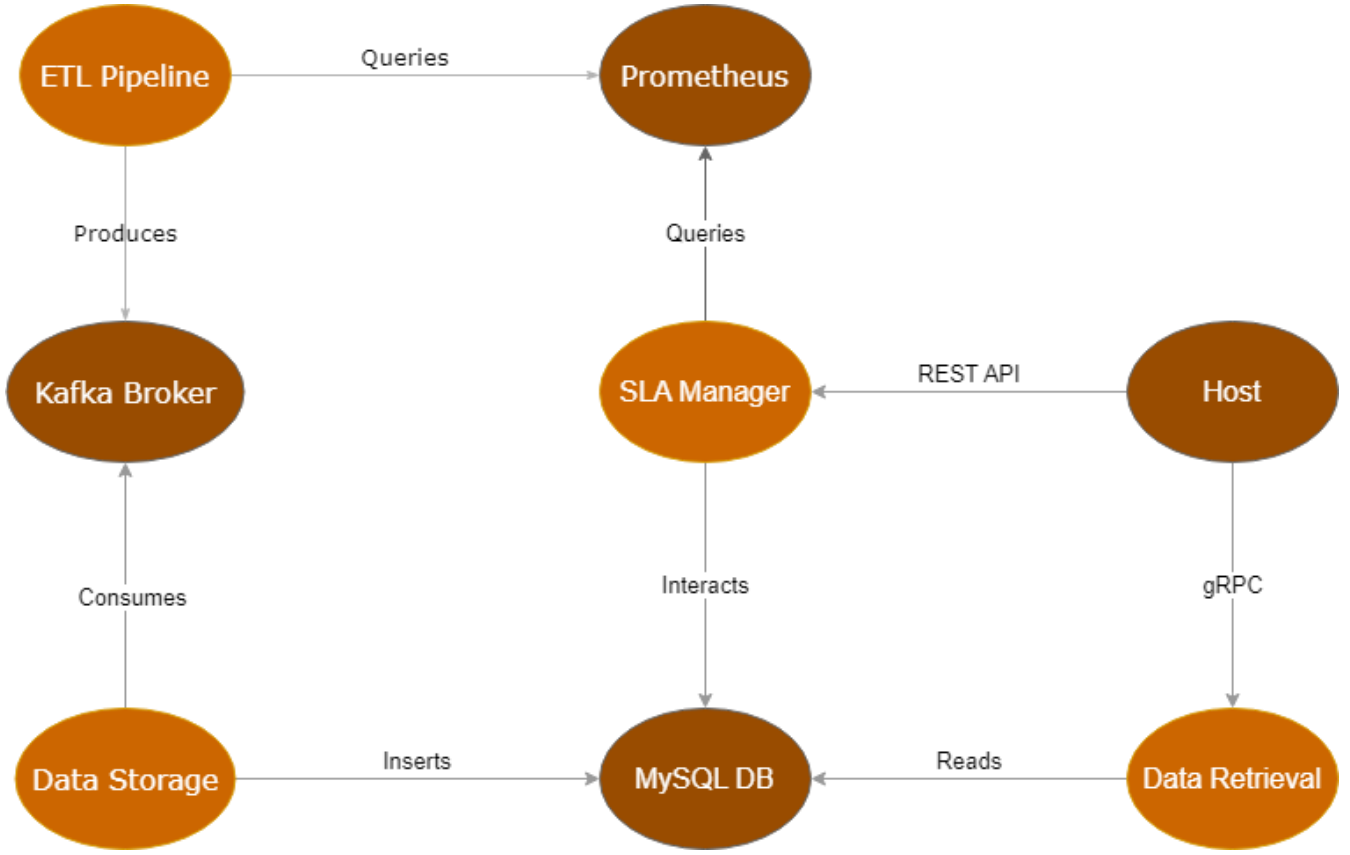
SVOLTO DA:
LEONE DAMIANO
SPADARO SAPARI LORENZO

Contenuto elaborato

L’elaborato prevede la realizzazione di un sistema formato da 4 microservizi che permetta di monitorare le metriche di funzionamento di un’applicazione distribuita.

Le metriche sono state recuperate da un server Prometheus fornito dal prof. Morana.

Diagramma del Sistema



Metriche selezionate

Tra le molteplici metriche esposte dal server Prometheus fornito, le 5 metriche scelte per la formazione di un Service Level Agreement sono:

- node_memory_memFree_bytes: Byte di Memoria Liberi (non include cache e buffer)
- node_memory_memAvailable_bytes: Memoria disponibile (include memoria cache e buffers)
- node_filesystem_free_bytes: Spazio libero su disco (include lo spazio riservato per root)
- node_filesystem_avail_bytes: Spazio libero su disco (spazio disponibile come utente user)
- node_filefd_allocated: Numero totale filedescriptors allocati

Queste metriche fanno parte del job ‘host’ (Node Exporter).

Data Pipeline

```python
def elaboraMetrica(nome_metrica,metrica,partizione,producer):
    isStationary=False
    critValues=[]
    metrics_data={}
    cyclical_component={}
    seasonal_component={}
    trend_component={}

    #Augmented dickey fuller test per valutare la staticità
    result_adf = adfuller(metrica,maxlag=3)
    out = pd.Series(result_adf[0:4],index=['ADF test statistic','p-value','# lags used','# observations'])
    if result_adf[1] <0.05:
        isStationary=True

    for key,val in result_adf[4].items():
        critValues.append(val)

    #filtro hodrick-prescott per prendere informazioni sulla ciclicità
    cyclical, trend = hpfilter(metrica, lamb=1600)
    cyclical.keys=cyclical.keys().strftime('%Y-%m-%d %H:%M:%S')

    #Autocorrelazione
    result_acf= acf(metrica)
    np.nan_to_num(result_acf,copy=False)
```

```python
#valutazione metriche nelle ultime 1,3,12h
hourly_data = metrica.resample('1H').last().agg(['max', 'min', 'mean','std']).to_dict()
three_hourly_data = metrica.resample('3H').last().agg(['max', 'min', 'mean','std']).to_dict()
twelve_hourly_data = metrica.resample('12H').last().agg(['max', 'min', 'mean','std']).to_dict()

tsr = trend.resample(rule='1T').mean()

#predizione delle metriche per i prox 10 min
tsmodel = ExponentialSmoothing(tsr, trend='mul', seasonal='add',seasonal_periods=550).fit()
prediction = tsmodel.forecast(10)
predicted_metrics=prediction.agg(['max','min','mean']).to_dict()

#invio dati al broker kafka
total_data={'metric_name':nome_metrica,partizione:{'values':metrics_data,'adfuller_statistic':result_adf[0],'adfuller_p_val':result_adf[1],
            'adfuller_stationary':isStationary,'adfuller_critical':critValues,'acf':result_acf.tolist(),
            'decompose_season':seasonal_component,'decompose_trend':trend_component,'cyclical_component':cyclical_component,
            'hourly_data':hourly_data,'three_hourly_data':three_hourly_data,'twelve_hourly_data':twelve_hourly_data,
            '10m_prediction':predicted_metrics}}

producer.send("prometheusdata",total_data)
producer.flush()
```

Data Storage

```
cnx = mysql.connector.connect(host = "db",database= "test_dsbd",user = "user",password = "password")
cursor = cnx.cursor()

consumer=KafkaConsumer(bootstrap_servers='kafka:9092',api_version=(0,10,2),auto_offset_reset='earliest',enable_auto_commit=False)
consumer.subscribe(['prometheusdata'])

# Read data from kafka
for message in consumer:
     data=json.loads(message[6])
     metric_name=data['metric_name']
     for disk, values in data.items():
          if disk == 'metric_name':
               continue

          sampled_values=values['values']

          adfuller_statistic = values['adfuller_statistic']
          adfuller_p_value = values['adfuller_p_val']
          adfuller_stationary = values['adfuller_stationary']
          adfuller_critical= values['adfuller_critical']

          acf= values['acf']
```

```
          hourly_data = values['hourly_data']
          hourly_max=str(hourly_data['value']['max'])
          hourly_min= str(hourly_data['value']['min'])
          hourly_mean=str(hourly_data['value']['mean'])
          hourly_std= str(hourly_data['value']['std'])

          three_hourly_data = values['three_hourly_data']
          three_hourly_max= str(three_hourly_data['value']['max'])
          three_hourly_min= str(three_hourly_data['value']['min'])
          three_hourly_mean=str(three_hourly_data['value']['mean'])
          three_hourly_std= str(three_hourly_data['value']['std'])

          twelve_hourly_data = values['twelve_hourly_data']
          twelve_hourly_max= str(twelve_hourly_data['value']['max'])
          twelve_hourly_min= str(twelve_hourly_data['value']['min'])
          twelve_hourly_mean=str(twelve_hourly_data['value']['mean'])
          twelve_hourly_std= str(twelve_hourly_data['value']['std'])

          prediction = values['10m_prediction']
          prediction_max= str(prediction['max'])
          prediction_min= str(prediction['min'])
```

Data Storage cont.

```
query6="INSERT INTO aggregates(nome,partizione,intervallo,tipologia,value) VALUES(%s,%s,%s,%s,%s)"
tuples=[]
tuples.append([metric_name,disk,'1h','max',hourly_max])
tuples.append([metric_name,disk,'1h','min',hourly_min])
tuples.append([metric_name,disk,'1h','mean',hourly_mean])
tuples.append([metric_name,disk,'1h','std_dev',hourly_std])
tuples.append([metric_name,disk,'3h','max', three_hourly_max])
tuples.append([metric_name,disk,'3h','min',three_hourly_min])
tuples.append([metric_name,disk,'3h','mean',three_hourly_mean])
tuples.append([metric_name,disk,'3h','std_dev',three_hourly_std])
tuples.append([metric_name,disk,'12h','max',twelve_hourly_max])
tuples.append([metric_name,disk,'12h','min',twelve_hourly_min])
tuples.append([metric_name,disk,'12h','mean',twelve_hourly_mean])
tuples.append([metric_name,disk,'12h','std_dev',twelve_hourly_std])

for tup in tuples:
     cursor.execute(query6, tup)

query7="INSERT INTO aggregates(nome,partizione,intervallo,tipologia,valore_predetto) VALUES(%s,%s,%s,%s,%s)"
value7=(metric_name,disk,'10m','max',prediction_max)
cursor.execute(query7, value7)
value8=(metric_name,disk,'10m','min',prediction_min)
cursor.execute(query7, value8)
value9=(metric_name,disk,'10m','mean',prediction_mean)
cursor.execute(query7, value9)
cnx.commit()
```

Data Retrieval – .proto file

```
syntax = "proto3";

package dataretrieval;


//Run in the working directory: python -m grpc_tools.protoc -I ./ --python_out=. --pyi_out=. --grpc_python_out=. ./retrieval.proto

service RetrievalService {

  //unary rpc
  rpc GetDickeyFuller(RetrievalRequest) returns (RetrievalDickeyFuller) {}

  //client-unary, server-streaming rpc
  rpc ListDecomposition(RetrievalRequest) returns (stream RetrievalDecomposition) {}

  rpc ListHodrickPrescott(RetrievalRequest) returns (stream RetrievalHodrickPrescott) {}

  rpc ListAutocorrelation(RetrievalRequest) returns (stream RetrievalAutocorrelation) {}

  rpc ListAggregates(RetrievalRequest) returns (stream RetrievalAggregates) {}
}

// Il messaggio di richiesta conterrà la query.
message RetrievalRequest {
  string query = 1;
}

// Il messaggio di risposta conterrà i dati in modo strutturato in base alla tabella interrogata.
message RetrievalDickeyFuller {
  string name = 1;
  string partition = 2;
  float test_statistic = 3;
  float p_value = 4;
  bool is_stationary = 5;
  float crit_value1 = 6;
  float crit_value5 = 7;
  float crit_value10 = 8;
}
```

```
message RetrievalDecomposition {
  string name = 1;
  string partition = 2;
  string timestamp = 3;
  string typology = 4;
  float value = 5;
}

message RetrievalHodrickPrescott {
  string name = 1;
  string partition = 2;
  string timestamp = 3;
  float value = 4;
}

message RetrievalAutocorrelation {
  string name = 1;
  string partition = 2;
  float value = 3;
}

message RetrievalAggregates {
  string name = 1;
  string partition = 2;
  string range = 3;
  string typology = 4;
  float value = 5;
  float predicted_value = 6;
}
```

Data Retrieval - asynch_client.py

```python
async def run() -> None:
    async with grpc.aio.insecure_channel('server_data_retrieval:50051') as channel:
        stub = retrieval_pb2_grpc.RetrievalServiceStub(channel)

        while(True):
            user_input = int(input("Inserisci un numero intero corrispondente a una metrica:\n\n1-node_memory_MemFree_bytes\r

            match user_input:

                case 1:
                    await node_memory_MemFree_bytes(stub)

                case 2:
                    await node_memory_MemAvailable_bytes(stub)

                case 3:
                    await node_filesystem_free_bytes_dev_sda2(stub)
                    await node_filesystem_free_bytes_tmpfs(stub)

                case 4:
                    await node_filesystem_avail_bytes_dev_sda2(stub)
                    await node_filesystem_avail_bytes_tmpfs(stub)

                case 5:
                    await node_filefd_allocated(stub)

                case 6:
                    await node_memory_MemFree_bytes(stub)
                    await node_memory_MemAvailable_bytes(stub)
                    await node_filesystem_free_bytes_dev_sda2(stub)
                    await node_filesystem_free_bytes_tmpfs(stub)
                    await node_filesystem_avail_bytes_dev_sda2(stub)
                    await node_filesystem_avail_bytes_tmpfs(stub)
                    await node_filefd_allocated(stub)

                case 7:
                    sys.exit(0)

                case _:
                    print( "Errore! Inserire un numero intero compreso tra 1 e 7.\n")
```

```
Inserisci un numero intero corrispondente a una metrica:

1-node_memory_MemFree_bytes
2-node_memory_memAvail_bytes
3-node_filesystem_free_bytes
4-node_filesystem_avail_bytes
5-node_filefd_allocated
6-Richiedi i dati relativi a tutte le metriche
7-Exit
>>
```

asynch_client.py cont. - esempio metrica 1

```python
async def node_memory_MemFree_bytes(stub: retrieval_pb2_grpc.RetrievalServiceStub) -> None:
    print( "Metrica 1: node_memory_MemFree_bytes\n")

    request = retrieval_pb2.RetrievalRequest(query='select * from dickey_fuller where nome="memory_MemFree_bytes"')
    print("-------------- GetDickeyFuller --------------\n")
    await get_dickey_fuller(stub, request)

    request = retrieval_pb2.RetrievalRequest(query='select * from decomposition where nome="memory_MemFree_bytes"')
    print("-------------- ListDecomposition --------------\n")
    await list_decomposition(stub, request)

    request = retrieval_pb2.RetrievalRequest(query='select * from hodrick_prescott where nome="memory_MemFree_bytes"')
    print("-------------- ListHodrickPrescott --------------\n")
    await list_hodrick_prescott(stub, request)

    request = retrieval_pb2.RetrievalRequest(query='select * from autocorrelation where nome="memory_MemFree_bytes"')
    print("-------------- ListAutocorrelation --------------\n")
    await list_autocorrelation(stub, request)

    request = retrieval_pb2.RetrievalRequest(query='select * from aggregates where nome="memory_MemFree_bytes"')
    print("-------------- ListAggregates --------------\n")
    await list_aggregates(stub, request)

    print( "Fine stampa Metrica 1: node_memory_MemFree_bytes\n")

```

```python
async def list_aggregates(stub: retrieval_pb2_grpc.RetrievalServiceStub,
                          request: retrieval_pb2.RetrievalRequest) -> None:
    results = stub.ListAggregates(request)
    async for result in results:
        if result.name:
            print(f"| nome metrica: {result.name} | partizione: {result.partition} | range: {result.range} | tipologia: {result.typology} | value: {round(result.value, 1)} | valore predetto: {round(result.predicted_value, 1)} |")
    print("\n")

```

asynch_client.py cont. – esempio output:1

```
-------------- ListAggregates --------------

| nome metrica: memory_memFree_bytes | partizione: default | range: 1h | tipologia: max | value: 12012399616.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 1h | tipologia: min | value: 8310600192.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 1h | tipologia: mean | value: 10646500352.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 1h | tipologia: std_dev | value: 889110976.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 3h | tipologia: max | value: 11879600128.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 3h | tipologia: min | value: 8948230144.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 3h | tipologia: mean | value: 10516400128.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 3h | tipologia: std_dev | value: 894534976.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 12h | tipologia: max | value: 11392600064.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 12h | tipologia: min | value: 9631039488.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 12h | tipologia: mean | value: 10618000384.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 12h | tipologia: std_dev | value: 899774016.0 | valore predetto: 0.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 10m | tipologia: max | value: 0.0 | valore predetto: 10221400064.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 10m | tipologia: min | value: 0.0 | valore predetto: 10138699776.0 |
| nome metrica: memory_memFree_bytes | partizione: default | range: 10m | tipologia: mean | value: 0.0 | valore predetto: 10164599808.0 |
```

Data Retrieval - async_server.py

```python
async def serve() -> None:
    server = grpc.aio.server()
    retrieval_pb2_grpc.add_RetrievalServiceServicer_to_server(RetrievalService(), server)
    listen_addr = '[::]:50051'
    server.add_insecure_port(listen_addr)
    logging.info("Starting server on %s", listen_addr)
    await server.start()
    await server.wait_for_termination()


if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)
    asyncio.run(serve())
```

```python
class RetrievalService(retrieval_pb2_grpc.RetrievalServiceServicer):
    def __init__(self) -> None:
        super().__init__()

        # Connect to mysql
        try:
            self.cnx = sql.connect(
                user="user",
                password="password",
                host="db",
                database="test_dsbd")
            self.cursor = self.cnx.cursor()
        except sql.Error as e:
            print(f"Error connecting to Mysql DB: {e}")
            sys.exit(1)


    def __del__(self) -> None:
        self.cnx.close()
```

```python
    async def ListAggregates(
            self, request:retrieval_pb2.RetrievalRequest,
            context: grpc.aio.ServicerContext) -> AsyncIterable[retrieval_pb2.RetrievalAggregates]:
        self.cursor.execute(request.query)
        for result in self.cursor.fetchall():
            yield retrieval_pb2.RetrievalAggregates(
                    name=result[1], partition=result[2], range=result[3],
                    typology=result[4], value=result[5], predicted_value=result[6])
        yield retrieval_pb2.RetrievalAggregates(
                name='', partition='', range='',
                typology='', value=0, predicted_value=0)
```

SLA MANAGER REST API

GET -> http://localhost:80/read.php

POST -> http://localhost:80/create.php

POST -> http://localhost:80/update.php

POST -> http://localhost:80/delete.php

POST -> http://localhost:80/read_status.php

POST -> http://localhost:80/read_violation_1h.php

POST -> http://localhost:80/read_violation_3h.php

POST -> http://localhost:80/read_violation_12h.php

POST -> http://localhost:80/read_execution_time_12h.php

POST -> http://localhost:80/read_future_violation_10m.php

SLA MANAGER - read_violation_1h.php

```php
<?php
//headers
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");

include_once '../config/database.php';
include_once '../models/SLA.php';

$database = new Database();
$db = $database->getConnection();
$sla = new SLA($db);

$data = json_decode(file_get_contents("php://input"));

$sla->Nome = $data->Nome;

// query products
$stmt = $sla->read_sli_status();
$num = $stmt->rowCount();

// se viene trovato l'SLI con quel nome nel database
if($num>0){
    $arr = array();
    $arr["SLI"] = array();
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    extract($row);
    $item = array(
            "ID" => $id,
            "Nome metrica" => $nome,
            "Max value" => $max_value,
            "Min value" => $min_value,
        );
    array_push($arr["SLI"], $item);
    $tempo = "1h";
    $campioni = 60;
    $command = 'python query_violation.py node_'.$sla->Nome." ".$max_value." ".$min_value. " ".$tempo. " ".$campioni;
    $json = exec($command, $out, $status);
    $array = json_decode($json, true);
    $arr2 = array();
    $arr2["VIOLAZIONI"] = array();
    if(count($array)>0) {
        $item2 = array(
            "Numero di violazioni" => count($array)
            );
    array_push($arr2["VIOLAZIONI"], $item2);
    foreach($array as $item)
    {
    $item3 = array(
        "timestamp" => $item['timestamp'],
        "value" => $item['value']
    );
    array_push($arr2["VIOLAZIONI"], $item3);
    }
    echo json_encode($arr);
    echo json_encode($arr2);
    }
    else {
        echo json_encode($arr);
        echo json_encode(
        array("message" => "Nessuna violazione")
        );
    }
}else{
    echo json_encode($arr);
    echo json_encode(
        array("message" => "Nessun SLI Trovato avente tale nome")
    );
}

?>
```

SLA MANAGER - query_violation.py

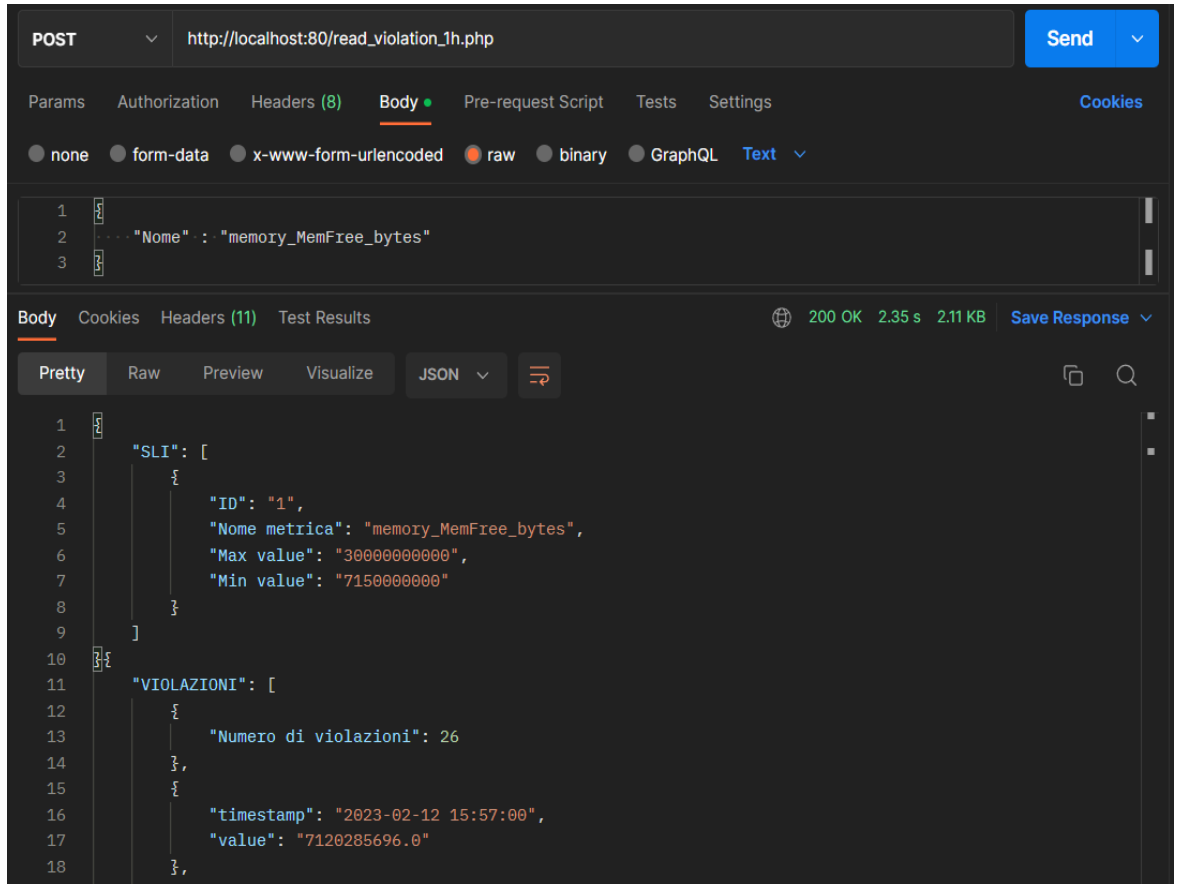
```python
import sys
from prometheus_api_client import PrometheusConnect,MetricRangeDataFrame
from prometheus_api_client.utils import parse_datetime
import json


client = PrometheusConnect(url='http://15.160.61.227:29090',disable_ssl=True)
label_config = {'job': 'host'}
start_time = parse_datetime(sys.argv[4])
#start_time = parse_datetime("1h")
end_time = parse_datetime("now")
metric_data = client.get_metric_range_data(
    metric_name=sys.argv[1],
    #metric_name="node_memory_MemAvailable_bytes",
    label_config=label_config,
    start_time=start_time,
    end_time=end_time,
)
metric_df = MetricRangeDataFrame(metric_data)
if sys.argv[1] == 'node_filesystem_avail_bytes' or sys.argv[1] == 'node_filesystem_free_bytes':
    data = metric_df.loc[metric_df['device']=='/dev/sda2', ['value']]
else:
    data = metric_df.loc[:, ['value']]

sampled_metrica = data.resample('T').mean()
sampled_metrica = sampled_metrica.reset_index()
sampled_metrica = sampled_metrica.applymap(str)
sampled_metrica_dict = sampled_metrica.to_dict(orient='index')
max_value=float(sys.argv[2])
min_value=float(sys.argv[3])
interval=int(sys.argv[5])
#max_value=32000000000
#min_value=21840000000
for i in range (interval):
#for i in range (60):
    current_value = float(sampled_metrica_dict[i]['value'])
    if current_value < max_value and current_value > min_value:
        del sampled_metrica_dict[i]

json_object = json.dumps(sampled_metrica_dict)
print(json_object)
```

SLA MANAGER - POSTMAN



Docker Compose

Ciascuno di questi servizi viene eseguito all'interno del proprio container. Per gestire la creazione e l'esecuzione di tutti i container è stato creato un unico file docker-compose.yml

```yaml
version: '3'
services:
  prometheus:
    image: prom/prometheus
    ports:
      - "9090:9090"

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - 9092:9092
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

Docker Compose cont.

```yaml
db:
  image: mysql:5.7
  environment:
    MYSQL_ROOT_PASSWORD: password
    MYSQL_DATABASE: test_dsbd
    MYSQL_USER: user
    MYSQL_PASSWORD: password
  volumes:
  - ./Database.sql:/docker-entrypoint-initdb.d/Database.sql
  restart: always

etl_data_pipeline:
  build:
    context: ./etl_data_pipeline
    dockerfile: Dockerfile
  depends_on:
    - prometheus
    - kafka
  environment:
    - PROMETHEUS_URL=http://15.160.61.227:29090
    - KAFKA_BOOTSTRAP_SERVERS=kafka:9092
    - KAFKA_TOPIC= 'prometheusdata'
```

```yaml
data_storage:
  build:
    context: ./data_storage
    dockerfile: Dockerfile
  depends_on:
    - kafka
    - db
  environment:
    - KAFKA_BOOTSTRAP_SERVERS=kafka:9092
    - MYSQL_HOST=db
    - MYSQL_USER=user
    - MYSQL_PASSWORD=password
    - MYSQL_DB=test_dsbd

client_data_retrieval:
 build:
  context: ./data_retrieval/client
  dockerfile: Dockerfile
 stdin_open: true
 tty: true
 depends_on:
  - server_data_retrieval
```

```yaml
server_data_retrieval:
 build:
  context: ./data_retrieval/server
  dockerfile: Dockerfile
 command: python3 /usr/app/async_server.py
 volumes:
  - ./data_retrieval/server:/usr/app/
 depends_on:
   - db
 environment:
    - MYSQL_HOST=db
    - MYSQL_USER=user
    - MYSQL_PASSWORD=password
    - MYSQL_DB=test_dsbd

sla_manager:
  build:
    context: ./sla_manager
    dockerfile: Dockerfile
  depends_on:
    - prometheus
    - db
  ports:
    - 80:80
  environment:
   - PROMETHEUS_URL=http://15.160.61.227:29090
   - MYSQL_HOST=db
```

Grazie per l’attenzione