UNIVERSITÀ degli STUDI di CATANIA

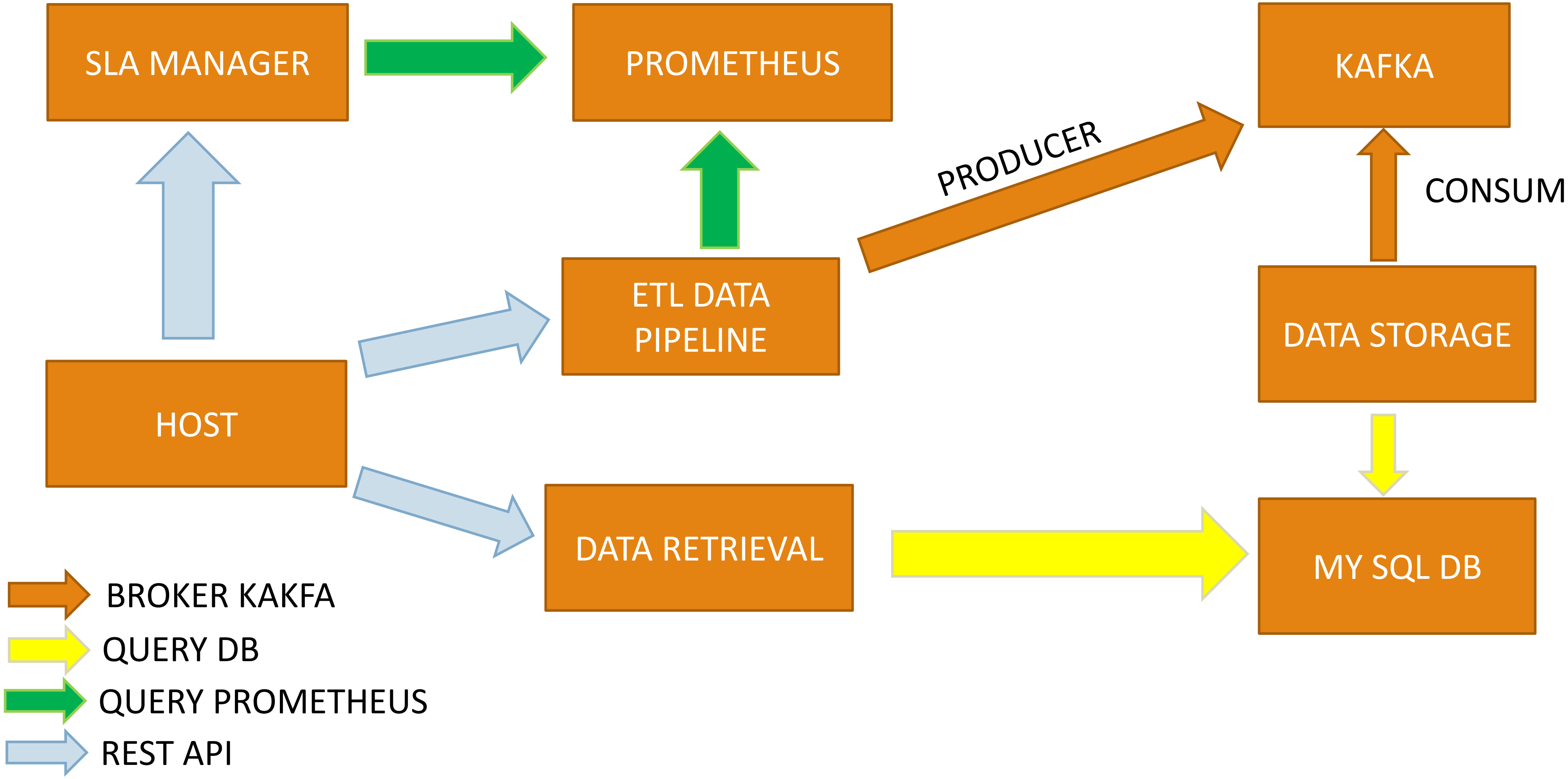
Dipartimento di Ingegneria Elettrica Elettronica e Informatica

Corso di Laurea Magistrale in Ingegneria Informatica

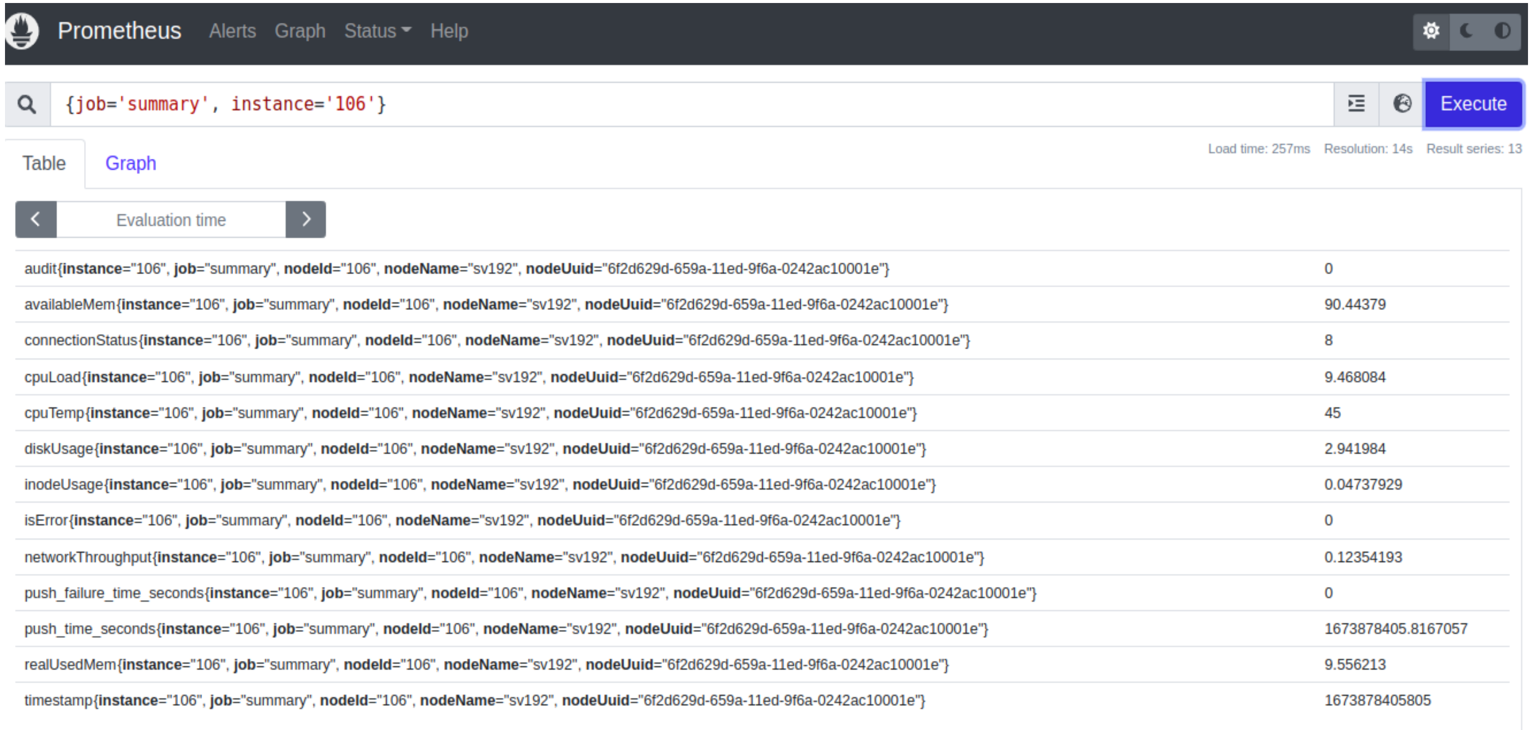
Presentazione finale progetto in itinere dell’insegnamento Distributed Systems and Big Data
Anno Accademico 2022-2023

Studenti
Luigi Fontana
Giuseppe Testa

- L'elaborato prevede la creazione di un’applicazione formata da più microservizi che permetta di monitorare le metriche esposte da un server Prometheus.
- Il server Prometheus utilizzato è stato fornito dal professore G. Morana ed è raggiungibile all’Url *http://15.160.61.227:29090*.
- Questo server fa uno scraping delle metriche esposte dell’exporter.


SLA MANAGER
PROMETHEUS
KAFKA
PRODUCER
CONSUMER
ETL DATA PIPELINE
HOST
DATA STORAGE
DATA RETRIEVAL
MY SQL DB
BROKER KAKFA
QUERY DB
QUERY PROMETHEUS
REST API

- Le metriche da monitorare sono state scelte in appartenenza al job:’summary’ su instance:’106’.



```
{job='summary', instance='106'}
```

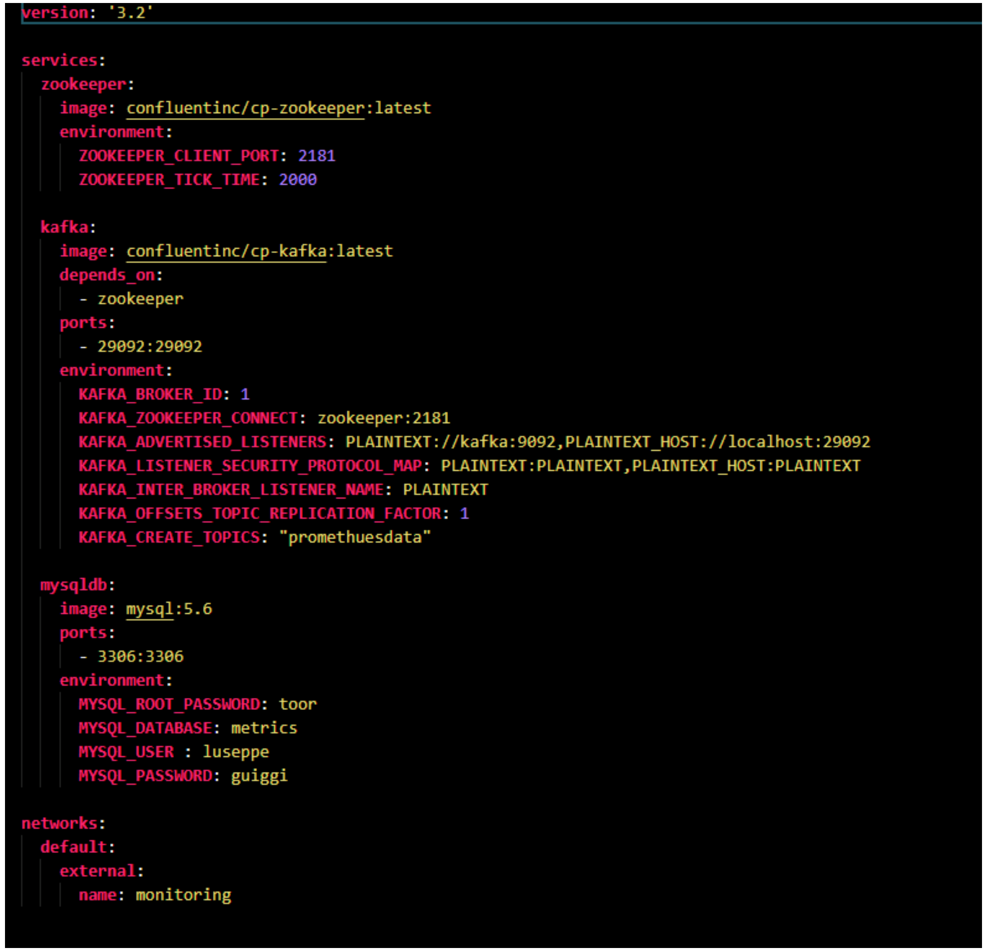
Load time: 257ms Resolution: 14s Result series: 13

```
audit{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                       0
availableMem{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                90.44379
connectionStatus{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}            8
cpuLoad{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                     9.468084
cpuTemp{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                     45
diskUsage{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                   2.941984
inodeUsage{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                  0.04737929
isError{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                     0
networkThroughput{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}           0.12354193
push_failure_time_seconds{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}   0
push_time_seconds{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}           1673878405.8167057
realUsedMem{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                 9.556213
timestamp{instance="106", job="summary", nodeId="106", nodeName="sv192", nodeUuid="6f2d629d-659a-11ed-9f6a-0242ac10001e"}                   1673878405805
```

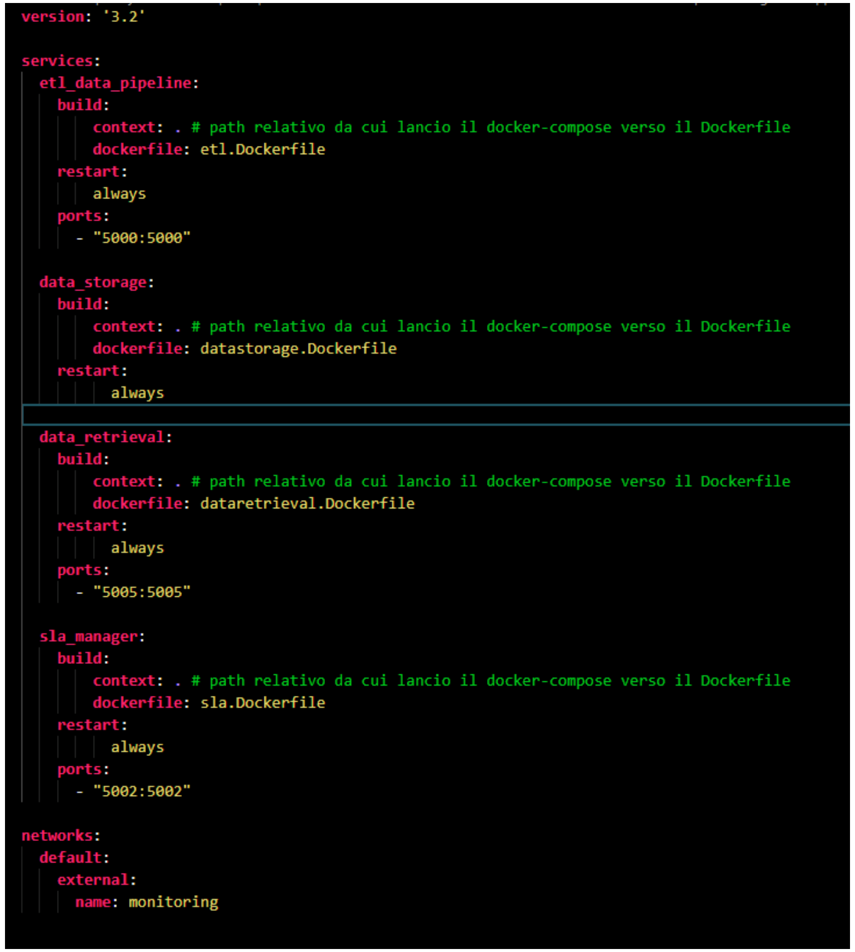
Elenco Metriche Analizzate:

- availableMem
- cpuLoad
- cpuTemp
- diskUsage
- inodeUsage
- networkThroughput
- push_time_seconds
- realUsedMem

- Per far funzionare il tutto abbiamo creato due docker compose collegati tramite una network creata da noi di nome monitoring

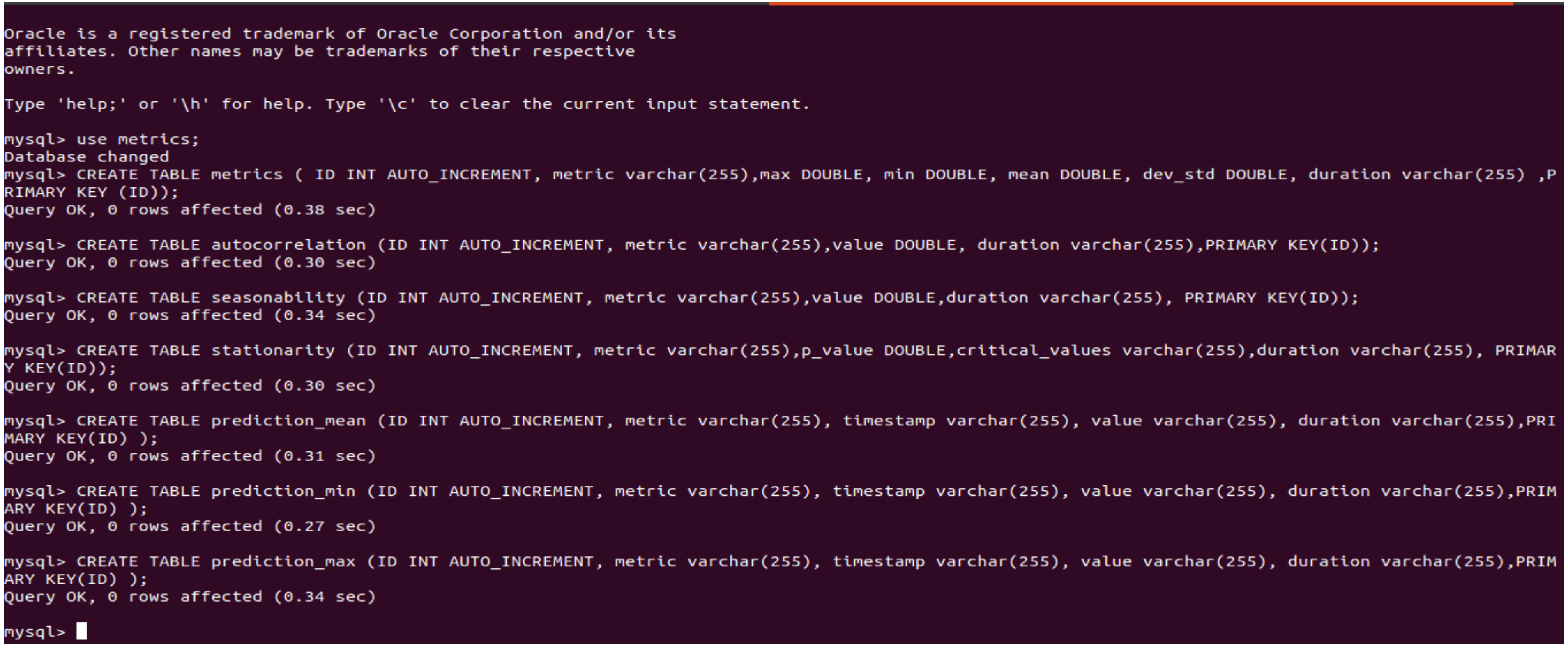


```yaml
version: '3.2'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - 29092:29092
    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
      KAFKA_CREATE_TOPICS: "promethuesdata"

  mysqldb:
    image: mysql:5.6
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: toor
      MYSQL_DATABASE: metrics
      MYSQL_USER : luseppe
      MYSQL_PASSWORD: guiggi

networks:
  default:
    external:
      name: monitoring
```

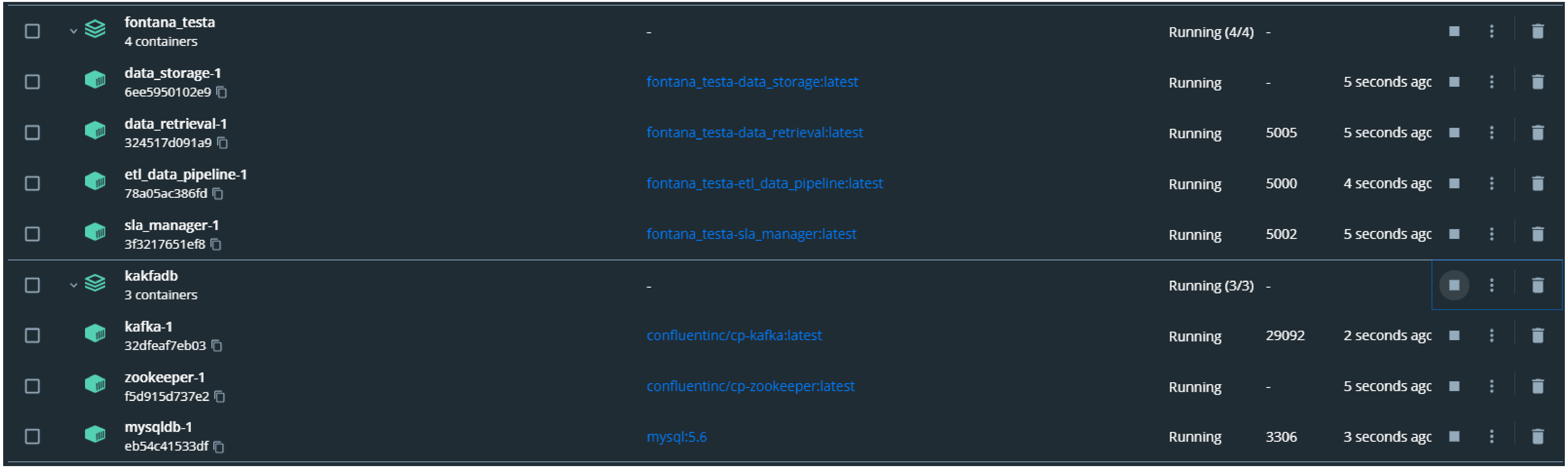


```yaml
version: '3.2'

services:
  etl_data_pipeline:
    build:
        context: . # path relativo da cui lancio il docker-compose verso il Dockerfile
        dockerfile: etl.Dockerfile
    restart:
        always
    ports:
      - "5000:5000"

  data_storage:
    build:
        context: . # path relativo da cui lancio il docker-compose verso il Dockerfile
        dockerfile: datastorage.Dockerfile
    restart:
          always

  data_retrieval:
    build:
        context: . # path relativo da cui lancio il docker-compose verso il Dockerfile
        dockerfile: dataretrieval.Dockerfile
    restart:
          always
    ports:
      - "5005:5005"

  sla_manager:
    build:
        context: . # path relativo da cui lancio il docker-compose verso il Dockerfile
        dockerfile: sla.Dockerfile
    restart:
          always
    ports:
      - "5002:5002"

networks:
  default:
    external:
      name: monitoring
```

Creazione delle tabelle sul database:



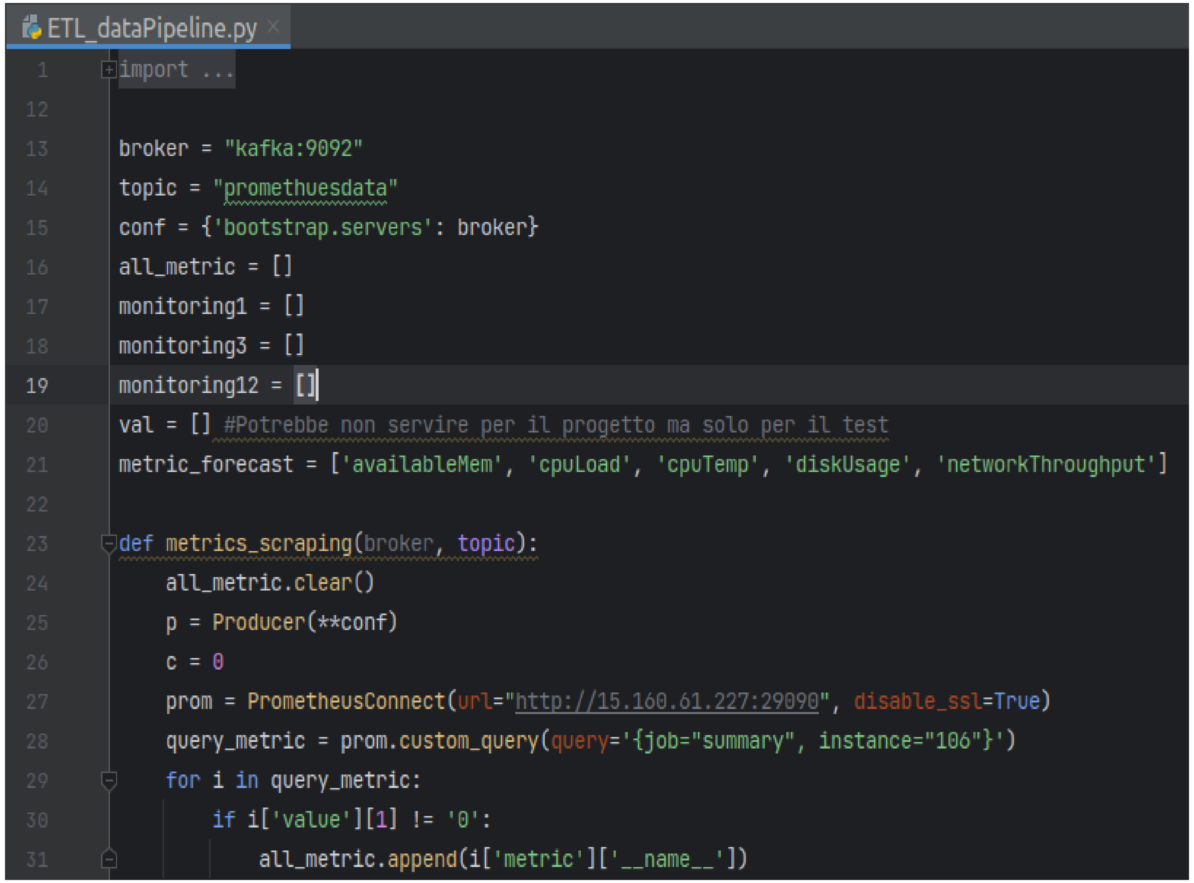
```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use metrics;
Database changed
mysql> CREATE TABLE metrics ( ID INT AUTO_INCREMENT, metric varchar(255),max DOUBLE, min DOUBLE, mean DOUBLE, dev_std DOUBLE, duration varchar(255) ,P
RIMARY KEY (ID));
Query OK, 0 rows affected (0.38 sec)

mysql> CREATE TABLE autocorrelation (ID INT AUTO_INCREMENT, metric varchar(255),value DOUBLE, duration varchar(255),PRIMARY KEY(ID));
Query OK, 0 rows affected (0.30 sec)

mysql> CREATE TABLE seasonability (ID INT AUTO_INCREMENT, metric varchar(255),value DOUBLE,duration varchar(255), PRIMARY KEY(ID));
Query OK, 0 rows affected (0.34 sec)

mysql> CREATE TABLE stationarity (ID INT AUTO_INCREMENT, metric varchar(255),p_value DOUBLE,critical_values varchar(255),duration varchar(255), PRIMAR
Y KEY(ID));
Query OK, 0 rows affected (0.30 sec)

mysql> CREATE TABLE prediction_mean (ID INT AUTO_INCREMENT, metric varchar(255), timestamp varchar(255), value varchar(255), duration varchar(255),PRI
MARY KEY(ID) );
Query OK, 0 rows affected (0.31 sec)

mysql> CREATE TABLE prediction_min (ID INT AUTO_INCREMENT, metric varchar(255), timestamp varchar(255), value varchar(255), duration varchar(255),PRIM
ARY KEY(ID) );
Query OK, 0 rows affected (0.27 sec)

mysql> CREATE TABLE prediction_max (ID INT AUTO_INCREMENT, metric varchar(255), timestamp varchar(255), value varchar(255), duration varchar(255),PRIM
ARY KEY(ID) );
Query OK, 0 rows affected (0.34 sec)

mysql> 
```



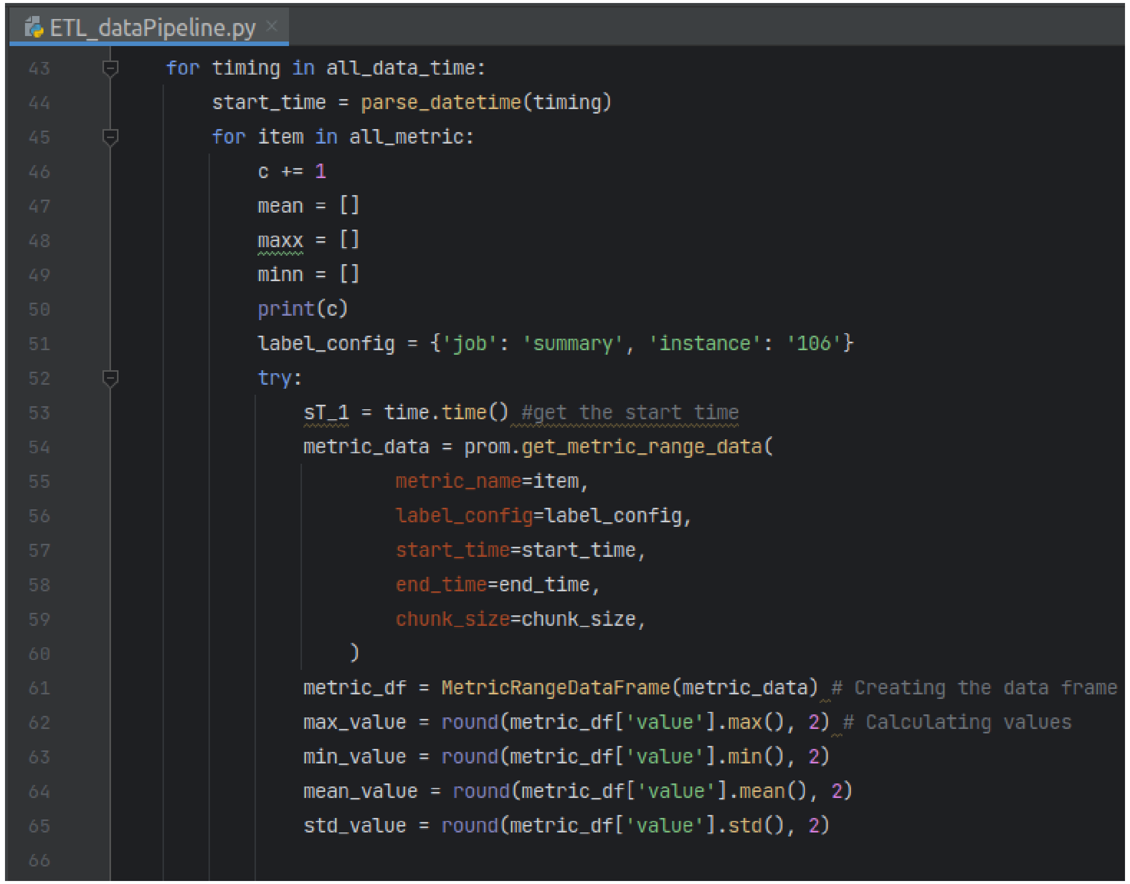
□	fontana_testa 4 containers	-	Running (4/4)	-	
□	data_storage-1 6ee5950102e9	fontana_testa-data_storage:latest	Running	-	5 seconds agc
□	data_retrieval-1 324517d091a9	fontana_testa-data_retrieval:latest	Running	5005	5 seconds agc
□	etl_data_pipeline-1 78a05ac386fd	fontana_testa-etl_data_pipeline:latest	Running	5000	4 seconds agc
□	sla_manager-1 3f3217651ef8	fontana_testa-sla_manager:latest	Running	5002	5 seconds agc
□	kakfadb 3 containers	-	Running (3/3)	-	
□	kafka-1 32dfeaf7eb03	confluentinc/cp-kafka:latest	Running	29092	2 seconds agc
□	zookeeper-1 f5d915d737e2	confluentinc/cp-zookeeper:latest	Running	-	5 seconds agc
□	mysqldb-1 eb54c41533df	mysql:5.6	Running	3306	3 seconds agc

- Prima far partire il compose col database e kafka
- Successivamente far partire il compose con I microservizi rimanenti.

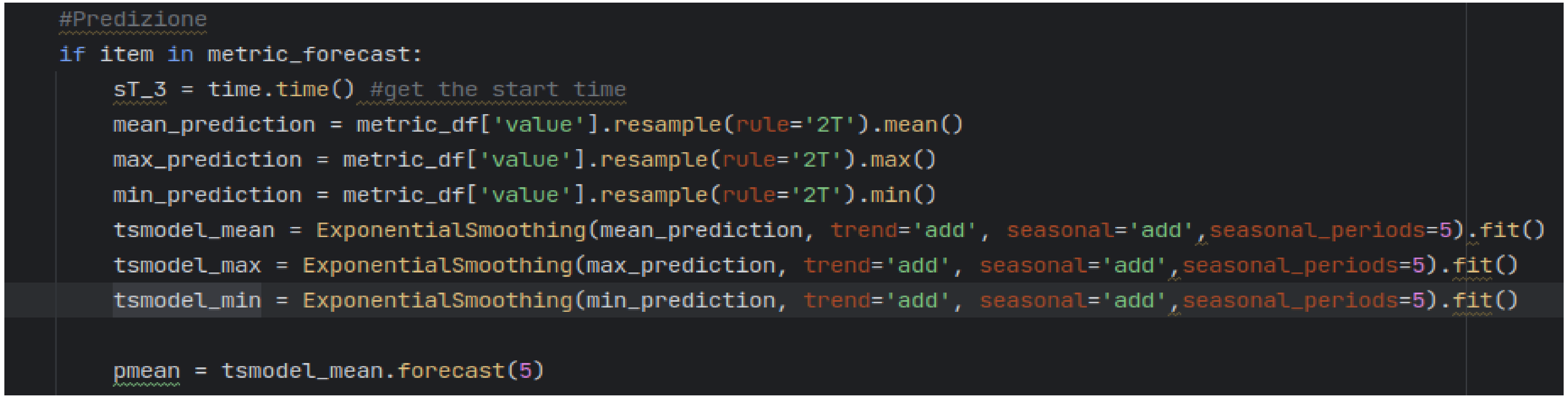
ETL_dataPipeline.py



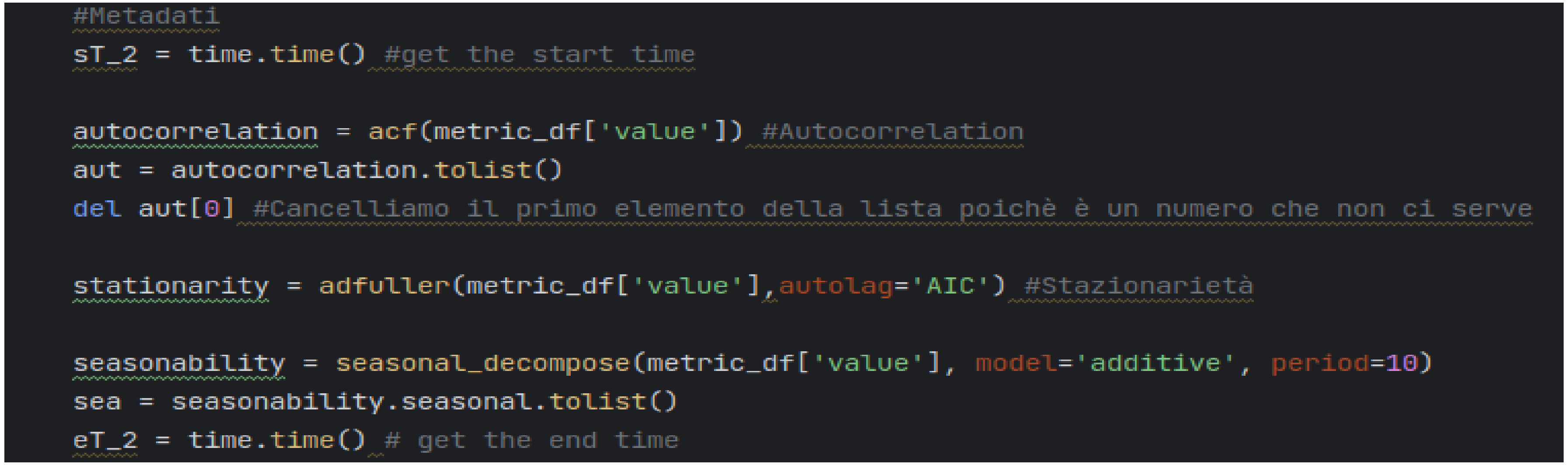
```python
import ...

broker = "kafka:9092"
topic = "promethuesdata"
conf = {'bootstrap.servers': broker}
all_metric = []
monitoring1 = []
monitoring3 = []
monitoring12 = []
val = [] #Potrebbe non servire per il progetto ma solo per il test
metric_forecast = ['availableMem', 'cpuLoad', 'cpuTemp', 'diskUsage', 'networkThroughput']

def metrics_scraping(broker, topic):
    all_metric.clear()
    p = Producer(**conf)
    c = 0
    prom = PrometheusConnect(url="http://15.160.61.227:29090", disable_ssl=True)
    query_metric = prom.custom_query(query='{job="summary", instance="106"}')
    for i in query_metric:
        if i['value'][1] != '0':
            all_metric.append(i['metric']['__name__'])
```



```python
    for timing in all_data_time:
        start_time = parse_datetime(timing)
        for item in all_metric:
            c += 1
            mean = []
            maxx = []
            minn = []
            print(c)
            label_config = {'job': 'summary', 'instance': '106'}
            try:
                sT_1 = time.time() #get the start time
                metric_data = prom.get_metric_range_data(
                        metric_name=item,
                        label_config=label_config,
                        start_time=start_time,
                        end_time=end_time,
                        chunk_size=chunk_size,
                    )
                metric_df = MetricRangeDataFrame(metric_data) # Creating the data frame
                max_value = round(metric_df['value'].max(), 2) # Calculating values
                min_value = round(metric_df['value'].min(), 2)
                mean_value = round(metric_df['value'].mean(), 2)
                std_value = round(metric_df['value'].std(), 2)

```



```
#Predizione
if item in metric_forecast:
    sT_3 = time.time() #get the start time
    mean_prediction = metric_df['value'].resample(rule='2T').mean()
    max_prediction = metric_df['value'].resample(rule='2T').max()
    min_prediction = metric_df['value'].resample(rule='2T').min()
    tsmodel_mean = ExponentialSmoothing(mean_prediction, trend='add', seasonal='add',seasonal_periods=5).fit()
    tsmodel_max = ExponentialSmoothing(max_prediction, trend='add', seasonal='add',seasonal_periods=5).fit()
    tsmodel_min = ExponentialSmoothing(min_prediction, trend='add', seasonal='add',seasonal_periods=5).fit()

    pmean = tsmodel_mean.forecast(5)
```



```
#Metadati
sT_2 = time.time() #get the start time

autocorrelation = acf(metric_df['value']) #Autocorrelation
aut = autocorrelation.tolist()
del aut[0] #Cancelliamo il primo elemento della lista poichè è un numero che non ci serve

stationarity = adfuller(metric_df['value'],autolag='AIC') #Stazionarietà

seasonability = seasonal_decompose(metric_df['value'], model='additive', period=10)
sea = seasonability.seasonal.tolist()
eT_2 = time.time() # get the end time
```

ETL DATA PIPELINE REST API:

GET -> http://localhost:5000/metrics/1h

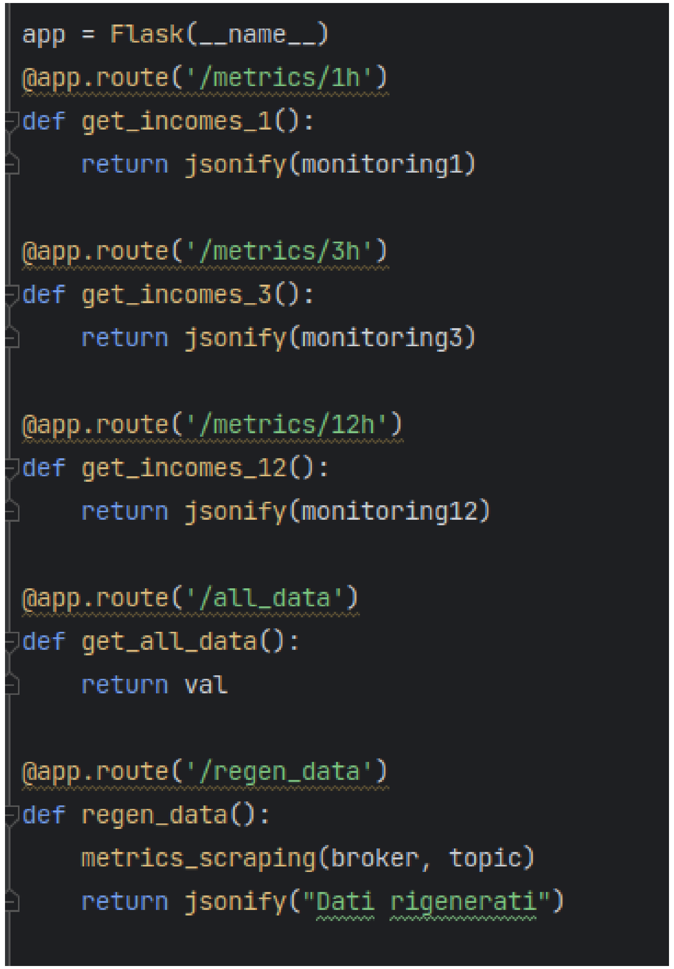
GET -> http://localhost:5000/metrics/3h

GET -> http://localhost:5000/metrics/12h

GET -> http://localhost:5000/regen_data

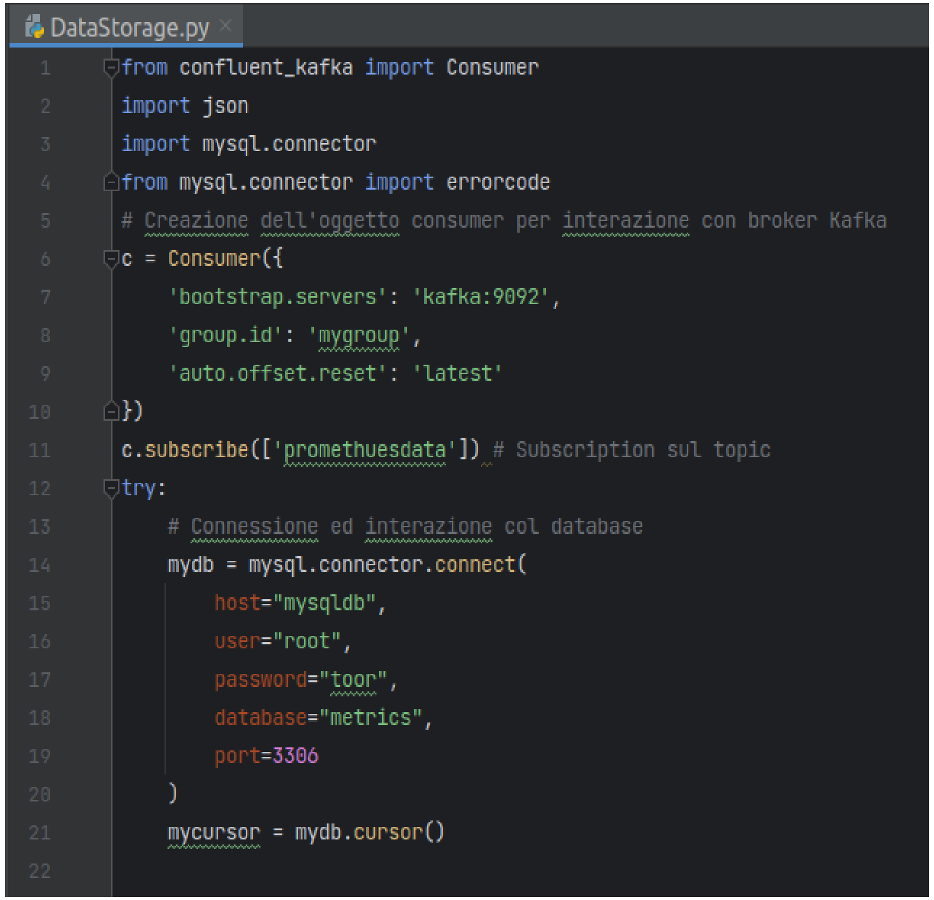
POST -> http://localhost:5000/forecasting

GET -> http://localhost:5000/all_data

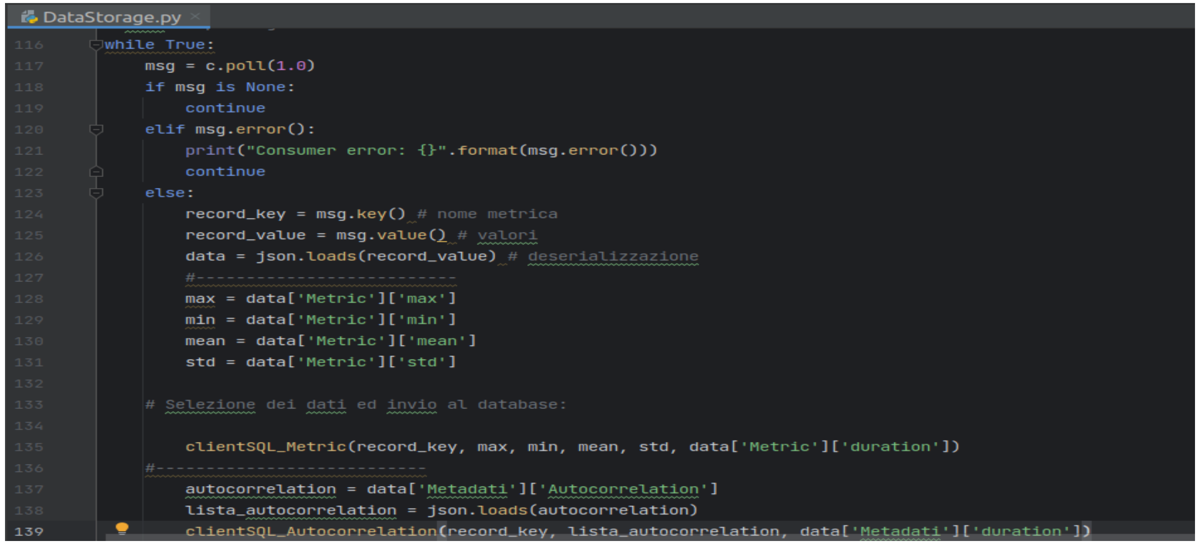


```
app = Flask(__name__)
@app.route('/metrics/1h')
def get_incomes_1():
    return jsonify(monitoring1)

@app.route('/metrics/3h')
def get_incomes_3():
    return jsonify(monitoring3)

@app.route('/metrics/12h')
def get_incomes_12():
    return jsonify(monitoring12)

@app.route('/all_data')
def get_all_data():
    return val

@app.route('/regen_data')
def regen_data():
    metrics_scraping(broker, topic)
    return jsonify("Dati rigenerati")
```

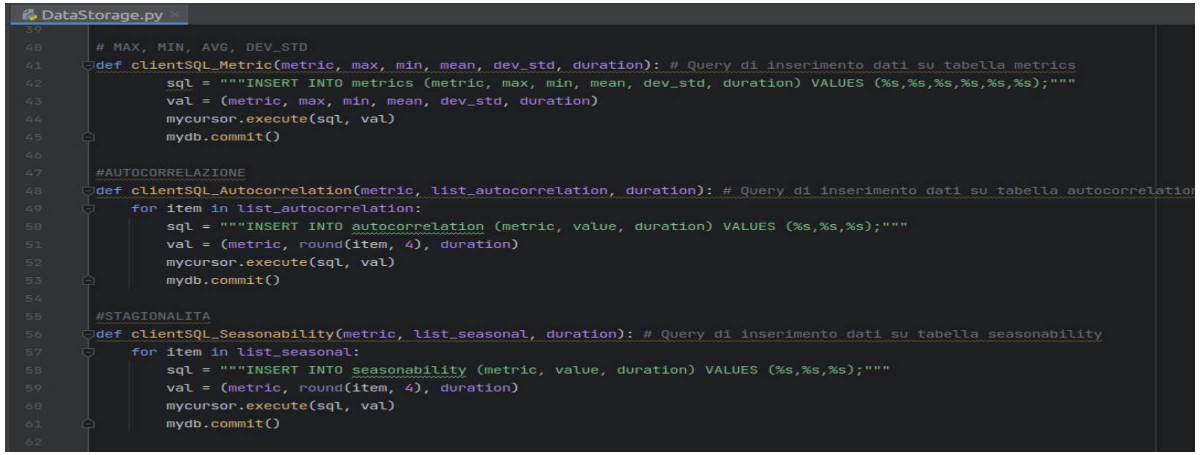
DataStorage.py



```python
from confluent_kafka import Consumer
import json
import mysql.connector
from mysql.connector import errorcode
# Creazione dell'oggetto consumer per interazione con broker Kafka
c = Consumer({
    'bootstrap.servers': 'kafka:9092',
    'group.id': 'mygroup',
    'auto.offset.reset': 'latest'
})
c.subscribe(['promethuesdata']) # Subscription sul topic
try:
    # Connessione ed interazione col database
    mydb = mysql.connector.connect(
        host="mysqldb",
        user="root",
        password="toor",
        database="metrics",
        port=3306
    )
    mycursor = mydb.cursor()

```

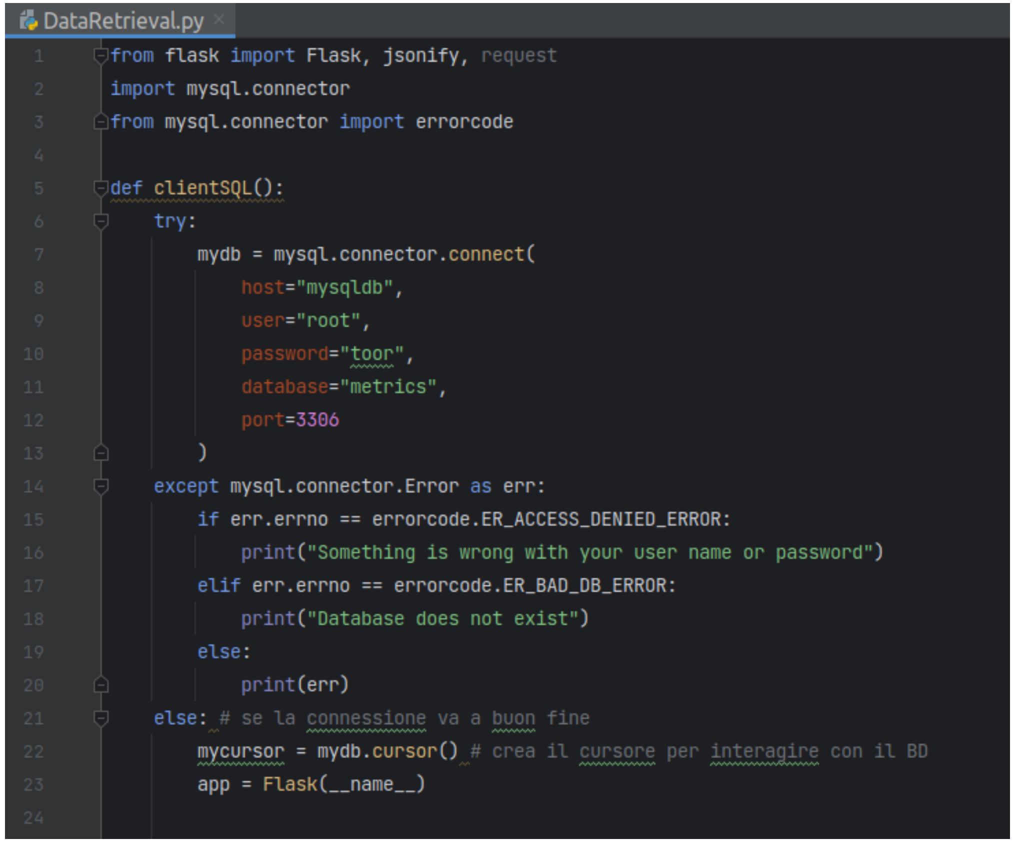


```python
while True:
    msg = c.poll(1.0)
    if msg is None:
        continue
    elif msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue
    else:
        record_key = msg.key() # nome metrica
        record_value = msg.value() # valori
        data = json.loads(record_value) # deserializzazione
        #-------------------------
        max = data['Metric']['max']
        min = data['Metric']['min']
        mean = data['Metric']['mean']
        std = data['Metric']['std']

    # Selezione dei dati ed invio al database:

        clientSQL_Metric(record_key, max, min, mean, std, data['Metric']['duration'])
    #--------------------------
        autocorrelation = data['Metadati']['Autocorrelation']
        lista_autocorrelation = json.loads(autocorrelation)
        clientSQL_Autocorrelation(record_key, lista_autocorrelation, data['Metadati']['duration'])
```



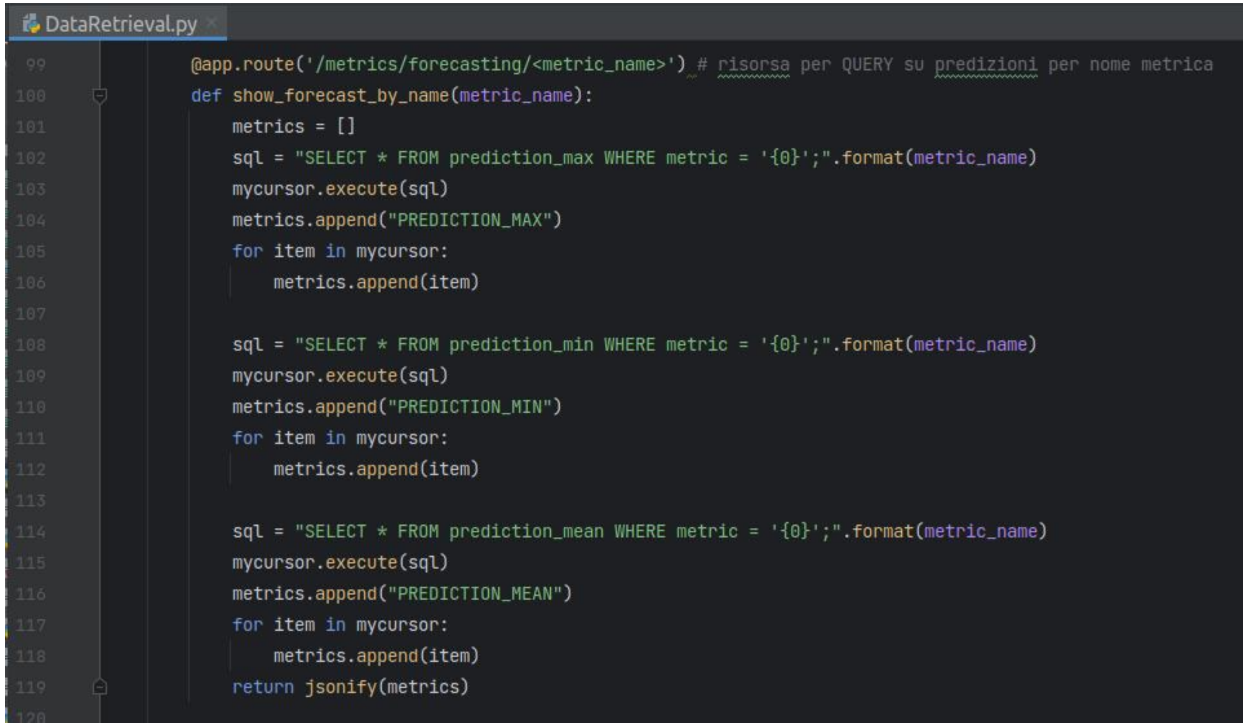
```python

# MAX, MIN, AVG, DEV_STD
def clientSQL_Metric(metric, max, min, mean, dev_std, duration): # Query di inserimento dati su tabella metrics
        sql = """INSERT INTO metrics (metric, max, min, mean, dev_std, duration) VALUES (%s,%s,%s,%s,%s,%s);"""
        val = (metric, max, min, mean, dev_std, duration)
        mycursor.execute(sql, val)
        mydb.commit()

#AUTOCORRELAZIONE
def clientSQL_Autocorrelation(metric, list_autocorrelation, duration): # Query di inserimento dati su tabella autocorrelatio
    for item in list_autocorrelation:
        sql = """INSERT INTO autocorrelation (metric, value, duration) VALUES (%s,%s,%s);"""
        val = (metric, round(item, 4), duration)
        mycursor.execute(sql, val)
        mydb.commit()

#STAGIONALITA
def clientSQL_Seasonability(metric, list_seasonal, duration): # Query di inserimento dati su tabella seasonability
    for item in list_seasonal:
        sql = """INSERT INTO seasonability (metric, value, duration) VALUES (%s,%s,%s);"""
        val = (metric, round(item, 4), duration)
        mycursor.execute(sql, val)
        mydb.commit()

```

DataRetrieval.py



DataRetrieval.py

```python
from flask import Flask, jsonify, request
import mysql.connector
from mysql.connector import errorcode

def clientSQL():
    try:
        mydb = mysql.connector.connect(
            host="mysqldb",
            user="root",
            password="toor",
            database="metrics",
            port=3306
        )
    except mysql.connector.Error as err:
        if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
            print("Something is wrong with your user name or password")
        elif err.errno == errorcode.ER_BAD_DB_ERROR:
            print("Database does not exist")
        else:
            print(err)
    else: # se la connessione va a buon fine
        mycursor = mydb.cursor() # crea il cursore per interagire con il BD
        app = Flask(__name__)

```



DataRetrieval.py

```python
        @app.route('/metrics/forecasting/<metric_name>') # risorsa per QUERY su predizioni per nome metrica
        def show_forecast_by_name(metric_name):
            metrics = []
            sql = "SELECT * FROM prediction_max WHERE metric = '{0}';".format(metric_name)
            mycursor.execute(sql)
            metrics.append("PREDICTION_MAX")
            for item in mycursor:
                metrics.append(item)

            sql = "SELECT * FROM prediction_min WHERE metric = '{0}';".format(metric_name)
            mycursor.execute(sql)
            metrics.append("PREDICTION_MIN")
            for item in mycursor:
                metrics.append(item)

            sql = "SELECT * FROM prediction_mean WHERE metric = '{0}';".format(metric_name)
            mycursor.execute(sql)
            metrics.append("PREDICTION_MEAN")
            for item in mycursor:
                metrics.append(item)
            return jsonify(metrics)
```

DATA RETRIEVAL REST API:

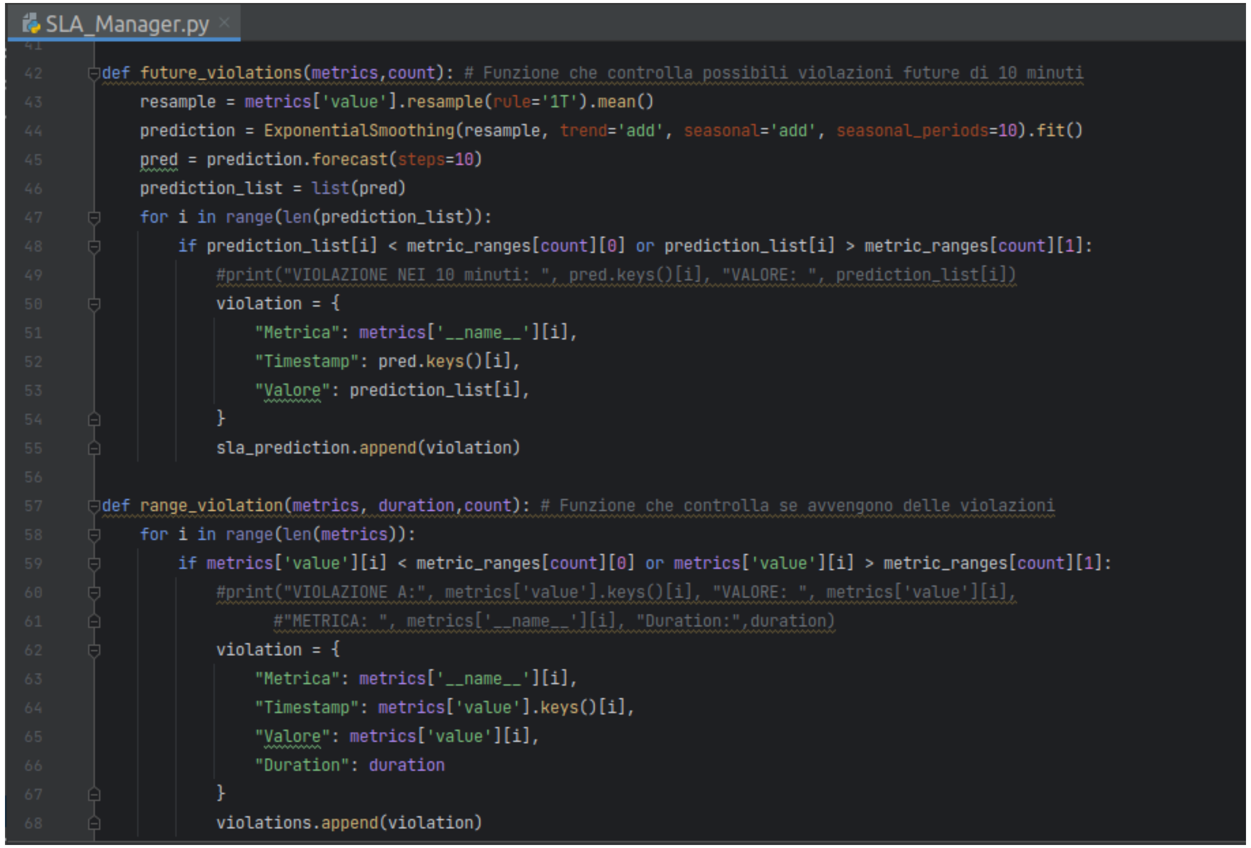
GET: http://localhost:5005/metrics/metric/<nome_della_metrica>

GET: http://localhost:5005/metrics/metadati/<nome_della_metrica>

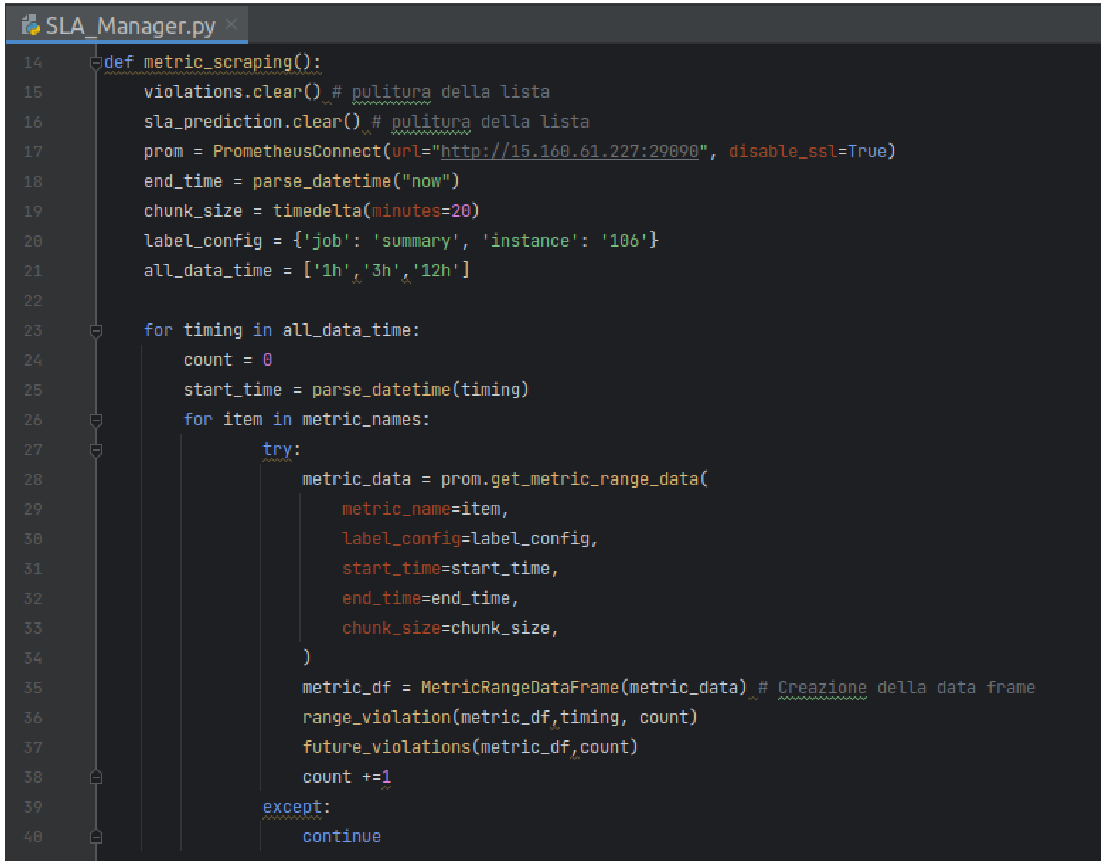
GET: http://localhost:5005/metrics/forecasting/<nome_della_metrica>

GET: http://localhost:5005/metrics

SLA_Manager.py

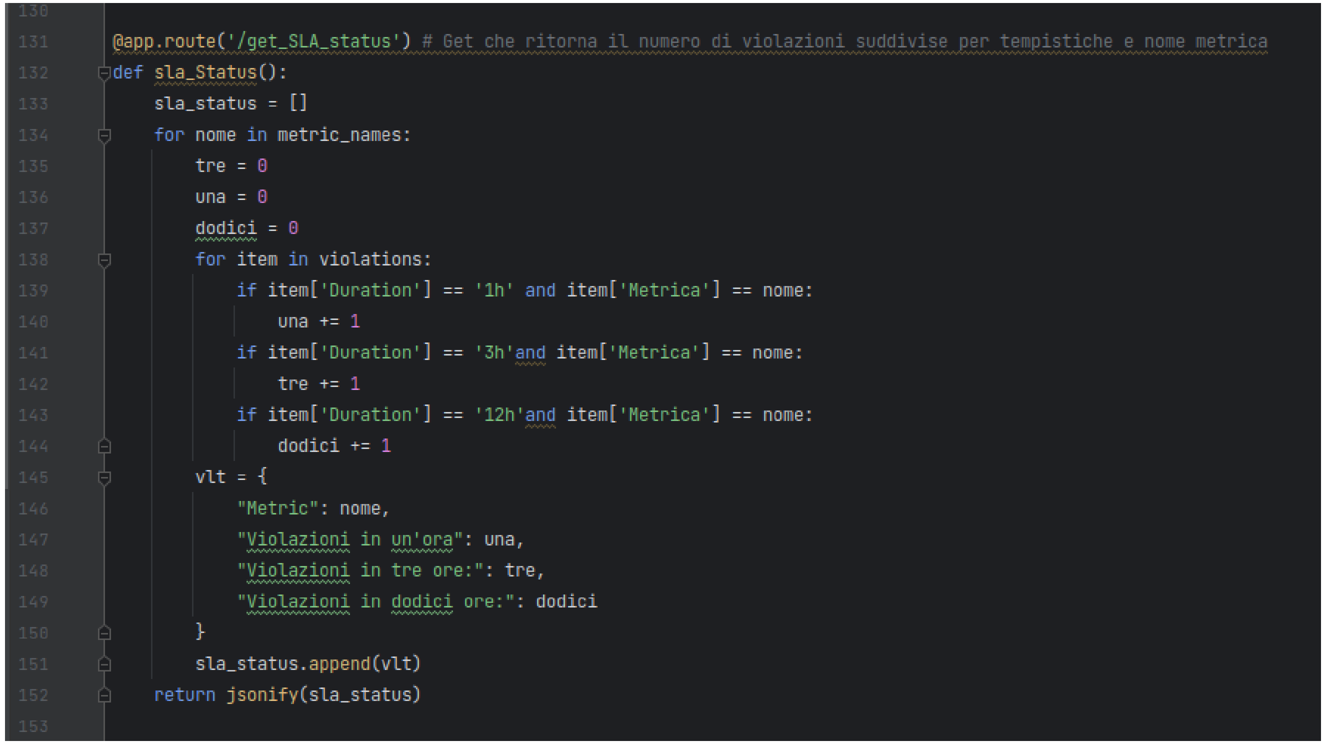


```python
def future_violations(metrics,count): # Funzione che controlla possibili violazioni future di 10 minuti
    resample = metrics['value'].resample(rule='1T').mean()
    prediction = ExponentialSmoothing(resample, trend='add', seasonal='add', seasonal_periods=10).fit()
    pred = prediction.forecast(steps=10)
    prediction_list = list(pred)
    for i in range(len(prediction_list)):
        if prediction_list[i] < metric_ranges[count][0] or prediction_list[i] > metric_ranges[count][1]:
            #print("VIOLAZIONE NEI 10 minuti: ", pred.keys()[i], "VALORE: ", prediction_list[i])
            violation = {
                "Metrica": metrics['__name__'][i],
                "Timestamp": pred.keys()[i],
                "Valore": prediction_list[i],
            }
            sla_prediction.append(violation)

def range_violation(metrics, duration,count): # Funzione che controlla se avvengono delle violazioni
    for i in range(len(metrics)):
        if metrics['value'][i] < metric_ranges[count][0] or metrics['value'][i] > metric_ranges[count][1]:
            #print("VIOLAZIONE A:", metrics['value'].keys()[i], "VALORE: ", metrics['value'][i],
                  #"METRICA: ", metrics['__name__'][i], "Duration:",duration)
            violation = {
                "Metrica": metrics['__name__'][i],
                "Timestamp": metrics['value'].keys()[i],
                "Valore": metrics['value'][i],
                "Duration": duration
            }
            violations.append(violation)
```

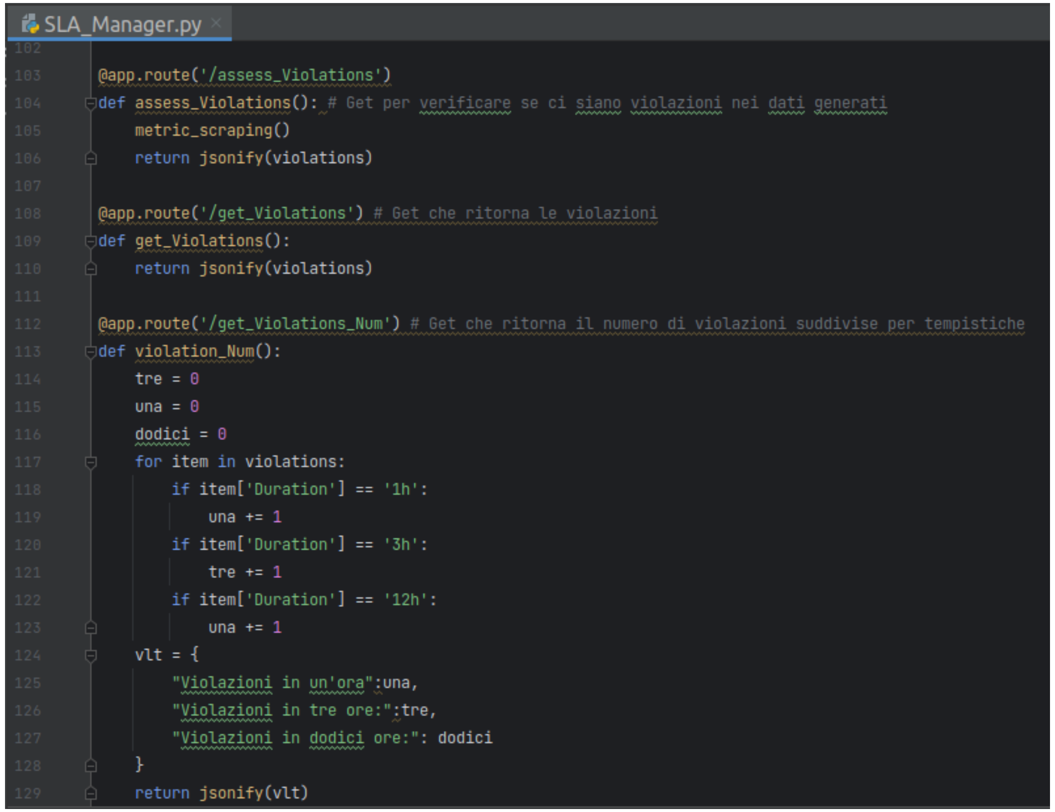


```python
def metric_scraping():
    violations.clear() # pulitura della lista
    sla_prediction.clear() # pulitura della lista
    prom = PrometheusConnect(url="http://15.160.61.227:29090", disable_ssl=True)
    end_time = parse_datetime("now")
    chunk_size = timedelta(minutes=20)
    label_config = {'job': 'summary', 'instance': '106'}
    all_data_time = ['1h','3h','12h']

    for timing in all_data_time:
        count = 0
        start_time = parse_datetime(timing)
        for item in metric_names:
                try:
                    metric_data = prom.get_metric_range_data(
                        metric_name=item,
                        label_config=label_config,
                        start_time=start_time,
                        end_time=end_time,
                        chunk_size=chunk_size,
                    )
                    metric_df = MetricRangeDataFrame(metric_data) # Creazione della data frame
                    range_violation(metric_df,timing, count)
                    future_violations(metric_df,count)
                    count +=1
                except:
                    continue
```

SLA MANAGER REST API:



```python
@app.route('/get_SLA_status') # Get che ritorna il numero di violazioni suddivise per tempistiche e nome metrica
def sla_Status():
    sla_status = []
    for nome in metric_names:
        tre = 0
        una = 0
        dodici = 0
        for item in violations:
            if item['Duration'] == '1h' and item['Metrica'] == nome:
                una += 1
            if item['Duration'] == '3h'and item['Metrica'] == nome:
                tre += 1
            if item['Duration'] == '12h'and item['Metrica'] == nome:
                dodici += 1
        vlt = {
            "Metric": nome,
            "Violazioni in un'ora": una,
            "Violazioni in tre ore:": tre,
            "Violazioni in dodici ore:": dodici
        }
        sla_status.append(vlt)
    return jsonify(sla_status)
```



```python
@app.route('/assess_Violations')
def assess_Violations(): # Get per verificare se ci siano violazioni nei dati generati
    metric_scraping()
    return jsonify(violations)

@app.route('/get_Violations') # Get che ritorna le violazioni
def get_Violations():
    return jsonify(violations)

@app.route('/get_Violations_Num') # Get che ritorna il numero di violazioni suddivise per tempistiche
def violation_Num():
    tre = 0
    una = 0
    dodici = 0
    for item in violations:
        if item['Duration'] == '1h':
            una += 1
        if item['Duration'] == '3h':
            tre += 1
        if item['Duration'] == '12h':
            una += 1
    vlt = {
        "Violazioni in un'ora":una,
        "Violazioni in tre ore:":tre,
        "Violazioni in dodici ore:": dodici
    }
    return jsonify(vlt)
```

SLA MANAGER REST API :

POST: http://localhost:5002/SLA

GET: http://localhost:5002/assess_Violations

GET: http://localhost:5002/get_Violations

GET: http://localhost:5002/get_Violations_Num

GET: http://localhost:5002/get_SLA_status

GET: http://localhost:5002/get_SLA_pred

GET: http://localhost:5002/get_SLA_pred_status

GRAZIE PER L’ATTENZIONE