



**UNIVERSITÀ  
degli STUDI  
di CATANIA**

Dipartimento di Ingegneria Elettrica Elettronica Informatica

**CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA**

# **Progettazione e realizzazione di un software per l'analisi di un set di metriche**

Anno accademico 2022-2023

Autori: Andrea Paolo Ventimiglia (1000039024)

# 1 Sommario

<b>INTRODUZIONE .....</b>	<b>II</b>
<b>1. Struttura progetto.....</b>	<b>3</b>
<b>1.1. Componenti del progetto .....</b>	<b>3</b>
1.1.1. Prometheus .....	3
1.1.2. Kafka .....	3
1.1.3. Docker .....	4
<b>1.2. Microservizi .....</b>	<b>5</b>
<b>2. Installazione e funzionamento .....</b>	<b>8</b>
<b>2.1. Installazione .....</b>	<b>8</b>
<b>2.2. Funzionamento .....</b>	<b>8</b>
2.2.1. Docker .....	8
2.2.2. ETL Data Pipeline .....	8
2.2.3. Data Storage .....	10
2.2.4. Data Retrieval .....	10
2.2.5. SLA Manager.....	12

# INTRODUZIONE

Il progetto descritto nella seguente relazione vede la realizzazione di un software basato su quattro microservizi che comunicano tra loro. Questi dovranno prelevare di un set di metriche fornite da un exporter Prometheus, analizzarle per poter calcolare diverse statistiche ed immagazzinare il tutto all'interno di un database MySQL che conterrà al suo interno diverse tabelle.

I microservizi sono: ETL Data Pipeline, Data Storage, Data Retrieval e SLA Manager.

I microservizi comunicheranno tra loro tramite un broker Kafka che andrà a gestire lo scambio di diversi messaggi.

Ogni microservizio svolgerà un compito ben preciso:

- ETL Data Pipeline si conatterà ad un server Prometheus ed effettuerà la richiesta di alcune metriche. Ottenute le metriche il microservizio andrà ad elaborare i dati ottenuti per calcolare diversi valori utili come ad esempio Stagionalità, Stazionarietà, Autocorrelazione etc. I risultati ottenuti saranno inviati come messaggi ad un broker Kafka.
- Data Storage riceverà, tramite un broker Kafka, dei messaggi provenienti dagli altri microservizi. Il microservizio esaminerà i messaggi ed andrà a registrare i dati in opportune tabelle contenute all'interno del database MySQL.
- Data Retrieval ha il compito di fornire un'interfaccia rest api per visualizzare i risultati contenuti nel database.
- SLA Manager avrà l'obiettivo di calcolare le possibili violazioni passate e future di un set di cinque metriche e lo stato dello SLA Manager. Anche in questo caso i risultati saranno inviati tramite messaggi al broker Kafka. Sarà anche possibile visualizzare mediante un'interfaccia rest api i risultati ottenuti.

# 1. Struttura progetto

## 1.1. Componenti del progetto

Il seguente progetto utilizza i componenti descritti brevemente nei paragrafi successivi.

### 1.1.1. Prometheus

Prometheus è un'applicazione software gratuita utilizzata per il monitoraggio e la segnalazione di eventi. Registra le metriche in tempo reale in un database di serie temporali costruito utilizzando un modello pull HTTP, con query flessibili e avvisi in tempo reale. I dati di Prometheus sono archiviati sotto forma di metriche. Ogni metrica può essere analizzata in base a un numero arbitrario di coppie chiave=valore (etichette). Le etichette possono includere informazioni sull'origine dei dati e altre informazioni di ripartizione specifiche dell'applicazione come il codice di stato HTTP, il metodo di query, l'endpoint, etc. Le metriche sono esposte su Prometheus mediante un exporter.

L'exporter utilizzato nel progetto fornisce l'accesso alle metriche Ceph attraverso il seguente link:

<http://15.160.61.227:29090/>

Il software realizzato, tramite query, interroga il server per recuperare i valori delle serie temporali relative alle varie metriche. Questi saranno utili per calcolare valori come la Stagionalità, l'Autocorrelazione, Stazionarietà, valori di massimo, minimo etc.

### 1.1.2. Kafka

È un software open source usato per la comunicazione dei micro-servizi, per lo scambio di dati chiamati messaggi, per il monitoraggio e per la raccolta dei log. Kafka è un broker utilizzato nei sistemi distribuiti per due motivi principali: rendere trasparente e asincrona la comunicazione tra più microservizi all'interno di un sistema distribuito. Un server funge da broker, mentre un client può essere sia produttore che consumatore. Kafka, per scambiare i messaggi, utilizza i topic, ovvero un'astrazione (argomento) indirizzabile usata per mostrare interesse in un dato flusso di dati.

Nel progetto viene creato un singolo topic nominato "prometheusdata" che verrà utilizzato dai producer e dal consumer per scambiare messaggi. I producer creano e affidano i dati al broker Kafka al relativo topic mentre il consumer si sottoscrive al topic per ricevere i messaggi in ordine. Inoltre, il consumer sa quali sono i messaggi che ha già letto e quelli non ancora letti, elemento molto importante per la durability.

### 1.1.3. Docker

Docker è un popolare software libero progettato per eseguire processi informatici in ambienti isolabili, minimali e facilmente distribuibili chiamati container Linux (o anche soltanto container), con l'obiettivo di semplificare i processi di deployment di applicazioni software. Tramite Docker è possibile pacchettizzare un'applicazione e le sue dipendenze in uno o più container che possono essere eseguiti su qualsiasi sistema operativo. Ogni container condivide il sistema operativo sottostante mantenendo un forte isolamento ed inoltre ciascun container ha un proprio insieme di processi, filesystem, interfacce di rete etc.

Nel progetto sono stati creati i seguenti container:

- Quattro relativi ai microservizi implementati
- Uno relativo al database
- Uno relativo a ZooKeeper
- Uno relativo al broker Kafka

Il file `docker-compose.yaml` gestisce e definisce il cluster di container deployati. Sotto la dicitura "services" vengono dichiarati i vari servizi e successivamente le relative configurazioni. Il numero di servizi è pari al numero di container.

Per ciascun microservizio è stato creato un `dockerfile` e un file `requirements.txt` contenente alcune delle dipendenze che dovranno essere installate.

## 1.2. Microservizi

Il progetto è strutturato come segue:

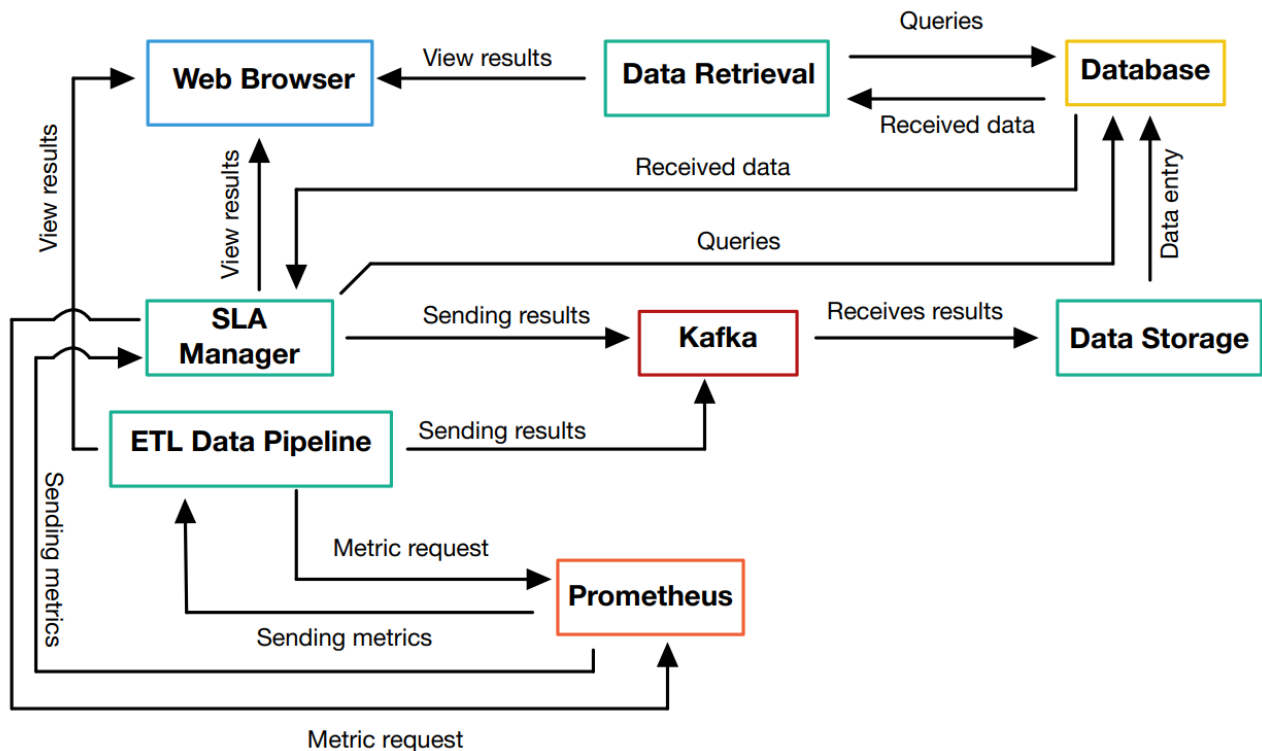


Figura 1.1 Schema architetturale

Come possiamo osservare i microservizi implementati sono quattro.

Il **primo**, l'ETL Data Pipeline si connette ad un server Prometheus per richiedere, tramite query, una lista di metriche. Con i valori delle serie temporali di queste metriche verranno fatte diverse operazioni tramite apposite funzioni. Le operazioni eseguite sono:

1. Calcolo della Stagionalità, Autocorrelazione e Stazionarietà di un set di metadati.
2. Delle stesse metriche si andranno a calcolare i valori di max, min, avg e dev\_std per 1h, 3h e 12h.
3. Utilizzando un set di 5 metriche si calcolerà la predizione dei valori di max, min e avg nei successivi 10 minuti.
4. Viene implementato un sistema di monitoraggio interno che rende visibile i tempi di esecuzione impiegati per produrre i risultati delle operazioni precedenti. Tali valori verranno visualizzati medite delle chiamate rest.

Infine, i risultati restituiti, nei punti appena descritti, verranno inoltrati mediante un messaggio ad un topic Kafka "promethuesdata".

Il **secondo** microservizio, chiamato Data Storage, avvia un consumer group del topic “promethuesdata” e, per ogni messaggio prelevato, memorizzerà i valori calcolati in un database MySQL chiamato “datastorage” contenente le seguenti tabelle:

- **Metadata:** memorizza i valori calcolati nel punto 1 dell’ETL Data Pipeline.
- **Metrics:** memorizza i valori calcolati nel punto 2 dell’ETL Data Pipeline.
- **Prediction:** memorizza i valori predetti nel punto 3 dell’ETL Data Pipeline.
- **Executiontimep1:** memorizza i tempi di esecuzione delle operazioni svolte nel punto 1 dell’ETL Data Pipeline.
- **Executiontimep2:** memorizza i tempi di esecuzione delle operazioni svolte nel punto 2 dell’ETL Data Pipeline.
- **Executiontimep3:** memorizza i tempi di esecuzione delle operazioni svolte nel punto 3 dell’ETL Data Pipeline.
- **Futurevioation:** memorizza, per ogni metrica, il numero di violazioni future con il relativo intervallo ammissibile calcolato nello SLA Manager.
- **Pastviolation:** memorizza, per ogni metrica e per ogni start\_time, il numero di violazioni passate con il relativo intervallo ammissibile calcolato nello SLA Manager.
- **Slastatus:** memorizza lo stato di ogni metrica calcolato nello SLA Manager.

Il **terzo** microservizio, chiamato Data Retrieval, fornisce un interfaccia rest che permette di estrarre le informazioni generate dalle applicazioni ETL Data Pipeline e SLA Manager contenute nel database.

Le possibile query disponibili e visibili tramite rest saranno:

- Query dei metadati
- Query dei valori di max, min, avg e std
- Query dei valori predetti
- Query dei tempi di esecuzione
- Query per la lista di tutte le metriche
- Query sulle violazioni passate e future
- Query sullo stato dello SLA Manager

Il **quarto** microservizio è lo SLA Manager, questo fornisce un elenco di metriche presenti nel database all’utente per verificare la presenza di possibili violazioni. L’utente deve scegliere cinque

metriche con i relativi range di ammissibilità, questi possono essere scritti da tastiera oppure calcolati automaticamente. Una volta selezionato il set di 5 metriche verranno calcolate tramite opportune funzioni le eventuali violazioni nelle ultime 1,3,12 ore e fornirà un'indicazione su una possibile violazione nei successivi 10 minuti.

Infine, i risultati ottenuti, verranno inoltrati mediante un messaggio ad un topic Kafka "promethuesdata" per essere salvati nel database e verranno inoltre resi visibili mediante rest.

Le possibili query sono:

- stato dello SLA Manager
- numero di violazioni nelle ultime 1,3, 12 ore
- numero di possibili violazioni future.



## 2. Installazione e funzionamento

In questo capitolo vengono descritte le componenti essenziali da installare per l'esecuzione del progetto, il funzionamento e la descrizione dal punto di vista implementativo del software.

### 2.1. Installazione

Le componenti richieste sono le seguenti:

- Docker
- Kafka
- Flask, per le chiamate rest
- Postman (opzionale, utile per visualizzare le richieste get con possibilità di scaricarle sottoforma di file json)
- Python 3.10.9
- MySQL

### 2.2. Funzionamento

#### 2.2.1. Docker

Dopo aver installato tutte le componenti richieste bisognerà aprire Docker ed aprire il terminale nella cartella "DSBD\_Esame" contenente il file "docker-compose.yml". Fatto questo lanciare il comando:

```
docker-compose up -d&
```

Successivamente, si potranno eseguire i vari microservizi.

#### 2.2.2. ETL Data Pipeline

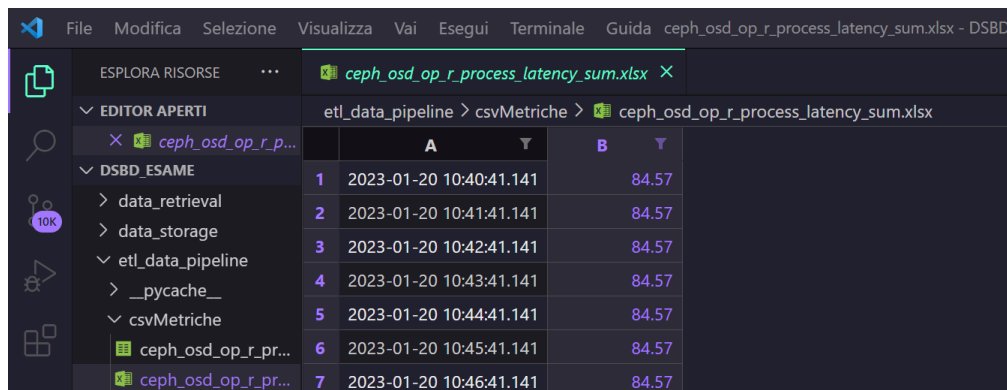
In questo microservizio viene innanzitutto configurato un sever Prometheus mediante la funzione *PrometheusConnect()* per fare lo scraping delle metriche dell'exporter Ceph esposte. Il server in questione è: <http://15.160.61.227:29090/>

Una volta connessi al server vengono filtrate le metriche da elaborare mediante la funzione *prometheus\_metrics()* che permette di selezionare le metriche aventi job uguale a "ceph-metrics". Inoltre, implementa un algoritmo per evitare di prendere metriche aventi valore pari a 0.

Una volta ricavate le metriche, tramite la funzione *setting\_parameters()* si recuperano dal server prometheus i valori delle relative serie temporali. Tali valori vengono inviati alla funzione *valuesCalculation()* e con l'utilizzo delle funzioni *autocorrelation()*, *stationarity()* e *seasonality()* si calcolano rispettivamente autocorrelazione, stazionarietà e stagionalità delle metriche. I risultati restituiti vengono salvati dentro il relativo dizionario "result\_metadata".

Successivamente, si richiama nuovamente la funzione *setting\_parameters()* impostando lo *start\_time* ai tempi stabiliti (1h, 3h e 12h) e i risultati ottenuti vengono inviati alla funzione *calculate\_values()* che calcola i valori richiesti di max, min, avg e std. Tali valori calcolati vengono salvati dentro il relativo dizionario “*result\_values*”.

La funzione *setting\_parameters()* viene invocata una terza volta per calcolare i valori di un set di cinque metriche prestabilite. I risultati sono inviati a due funzioni: *creating\_file\_csv()* che crea un file csv con la coppia tempo-valore utile per poter analizzare i valori della serie temporale. La seconda è la funzione *predictFiveMetrics()* che predice i valori di max, min, avg nei successivi 10 minuti. In tale funzione si trovano i parametri “*tred*”, “*seasonal*” e “*seasonal\_periods*” che vengono settati a seconda delle metriche scelte poiché ogni metrica ha il suo modello di predizione. I risultati delle predizioni vengono anch’essi salvati nel relativo dizionario “*result\_predict*”.



	A	T	B	T
1	2023-01-20 10:40:41.141		84.57	
2	2023-01-20 10:41:41.141		84.57	
3	2023-01-20 10:42:41.141		84.57	
4	2023-01-20 10:43:41.141		84.57	
5	2023-01-20 10:44:41.141		84.57	
6	2023-01-20 10:45:41.141		84.57	
7	2023-01-20 10:46:41.141		84.57	

Figura 2.1 File csv

I risultati ottenuti (*result\_metadata*, *result\_values*, *result\_predict*), insieme ai tempi di esecuzione impiegati per produrli, sono inviati alla funzione *kafkaResultProducer()* che configura un producer Kafka che ha il compito di inoltrare i dati al topic di nome “*prometheusdata*” creato precedentemente.

Infine, mediante Flask è possibile visualizzare tramite rest i valori dei tempi di esecuzioni calcolati al seguente indirizzo: <http://127.0.0.1:5000/>

Al suo interno vi sono le seguenti chiamate:

- [http://127.0.0.1:5000/execution\\_time\\_p1](http://127.0.0.1:5000/execution_time_p1)
- [http://127.0.0.1:5000/execution\\_time\\_p2](http://127.0.0.1:5000/execution_time_p2)
- [http://127.0.0.1:5000/execution\\_time\\_p3](http://127.0.0.1:5000/execution_time_p3)

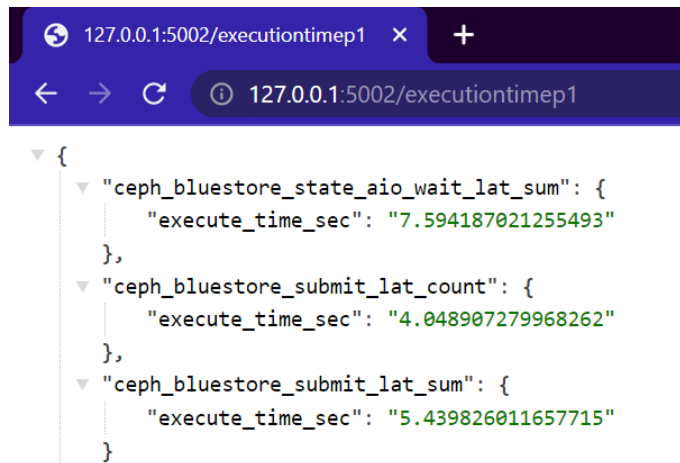


Figura 2.2 Get dei tempi di esecuzione del punto 1

### 2.2.3. Data Storage

In questo microservizio viene inizialmente configurato il database MySQL “datastorage” e un consumer Kafka che si sottoscriverà al topic “prometheusdata”. Man mano che i microservizi “ETL Data Pipeline” e “SLA Manager” inoltreranno i messaggi al topic il Data Storage, a seconda della richiesta, distingue il tipo di messaggio, converte questo da stringa a file json e inserisce i valori nelle relative tabelle.

### 2.2.4. Data Retrieval

L’obiettivo di questo microservizio è prelevare i dati dal database “datastorage” e visualizzarli tramite rest. Inizialmente viene configurato il database e poi vengono implementate diverse query. Le query sono disponibili al seguente link: <http://127.0.0.1:5002/>

Al suo interno vi sono le seguenti chiamate:

- <http://127.0.0.1:5002/metadata>
- <http://127.0.0.1:5002/predict>
- [http://127.0.0.1:5002/all\\_metrics](http://127.0.0.1:5002/all_metrics)
- <http://127.0.0.1:5002/pastviolation>
- <http://127.0.0.1:5002/futureviolation>
- [http://127.0.0.1:5002/sla\\_status](http://127.0.0.1:5002/sla_status)
- <http://127.0.0.1:5002/executiontimep1>
- <http://127.0.0.1:5002/executiontimep2>
- <http://127.0.0.1:5002/executiontimep3>

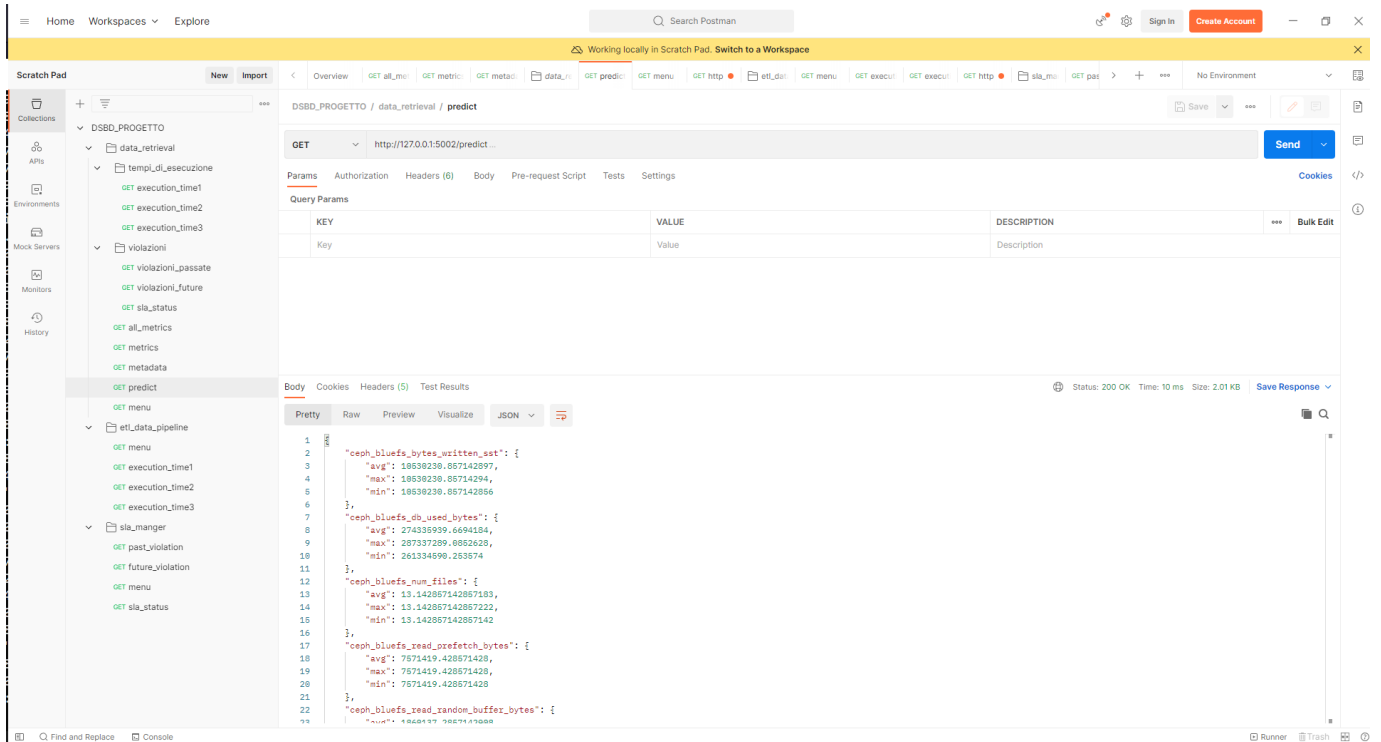


Figura 2.3 Pagina Postman delle varie richieste get

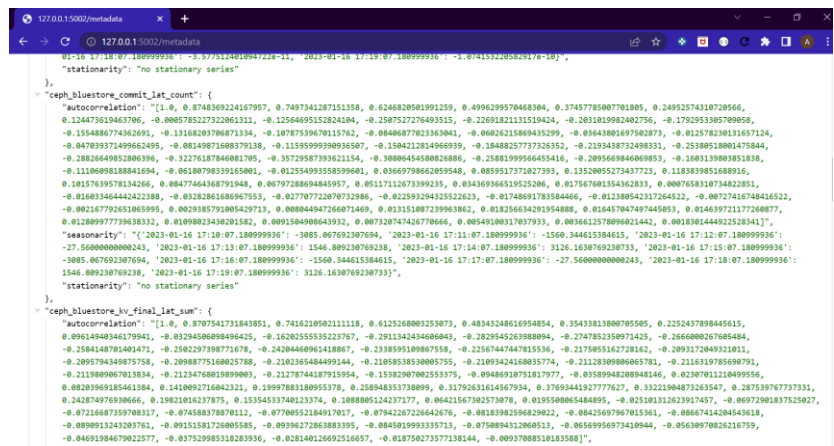
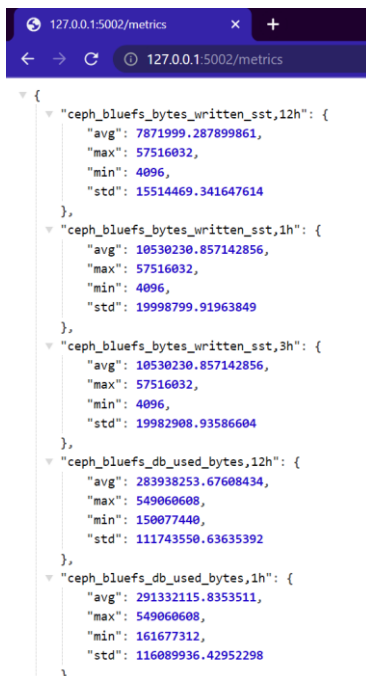


Figura 2.4 Stagionalità, l'autocorrelazione e la stazionarietà delle metriche

Figura 2.5 Max, min, avg e std

### 2.2.5. SLA Manager

Questo microservizio si occupa di verificare se le metriche che vengono analizzate subiscono delle violazioni. Inizialmente, viene invocata la funzione *enterMetricCodes()* che stampa a video la lista delle metriche precedentemente memorizzate nel database, come mostrato nella figura sotto.

```
0 - ceph_bluefs_bytes_written_sst
1 - ceph_bluefs_db_used_bytes
2 - ceph_bluefs_num_files
3 - ceph_bluefs_read_random_buffer_bytes
4 - ceph_bluestore_commit_lat_count
5 - ceph_bluestore_kv_final_lat_sum
6 - ceph_bluestore_kv_sync_lat_count
7 - ceph_bluestore_pricache:data_pri11_bytes
8 - ceph_bluestore_pricache:kv_committed_bytes
9 - ceph_bluestore_pricache:kv_onode_pri1_bytes
10 - ceph_bluestore_pricache:kv_pri1_bytes
11 - ceph_bluestore_pricache:meta_pri11_bytes
12 - ceph_bluestore_pricache_cache_bytes
13 - ceph_bluestore_pricache_target_bytes
14 - ceph_bluestore_read_lat_sum
15 - ceph_bluestore_state_aio_wait_lat_sum
16 - ceph_bluestore_submit_lat_count
17 - ceph_bluestore_submit_lat_sum
```

Figura 2.6 Elenco delle metriche disponibili

Tale funzione permette all'utente di inserire il codice di cinque metriche a sua scelta tra quelle elencate. In seguito, vengono applicati dei controlli per verificare se il codice relativo alla metrica inserito è corretto.

L'elenco delle metriche scelte viene passato alla funzione *enterAllowedValues()* che dà all'utente la possibilità d'inserire manualmente i valori di max e min o far calcolare in automatico il range di valori ammissibili. Nel primo caso, l'utente inserirà il valore di massimo e di minimo da tastiera (verranno fatti pure dei controlli per verificare se i valori inseriti sono accettabili oppure no). Nel secondo caso, verrà invocata la funzione *setting\_parameters()* che prenderà i valori prodotti dalla metrica per un certo start\_time e di questi calcola il massimo e il minimo.

Una volta ottenuti i valori di range ammissibili si passano alle funzioni *pastViolations()* e *futureViolations()* che hanno il compito di calcolare il numero di violazioni delle metriche analizzate verificando che il valore assunto dalle metriche si trova all'interno dell'intervallo selezionato oppure no.

La funzione *pastViolations()* calcola per ogni metrica il numero di violazioni nelle ultime 1h, 3h e 12h. Mentre la funzione *futureViolations()* calcola la predizione di ogni metrica e verifica la presenza di violazioni nei successivi 10 minuti.

I risultati delle funzioni vengono inseriti dentro i rispettivi dizionari “past\_violation\_list” e “future\_violation\_list”. Per entrambe le funzioni viene implementato un dizionario chiamato “sla\_status” che memorizza lo stato delle metriche.

I risultati ottenuti (past\_violation\_list, future\_violation\_list, sla\_status) sono inviati alla funzione *kakfaResultProducer()* che li inoltra al topic “prometheusdata” che a sua volta li invia al Data Storage per inserirli nel database.

Infine, mediante Flask è possibile visualizzare tramite rest i valori ottenuti al seguente indirizzo:

<http://127.0.0.1:5003/>

Al suo interno vi sono le seguenti chiamate:

- [http://127.0.0.1:5003/past\\_violation](http://127.0.0.1:5003/past_violation)
- [http://127.0.0.1:5003/future\\_violation](http://127.0.0.1:5003/future_violation)
- [http://127.0.0.1:5003/sla\\_status](http://127.0.0.1:5003/sla_status)

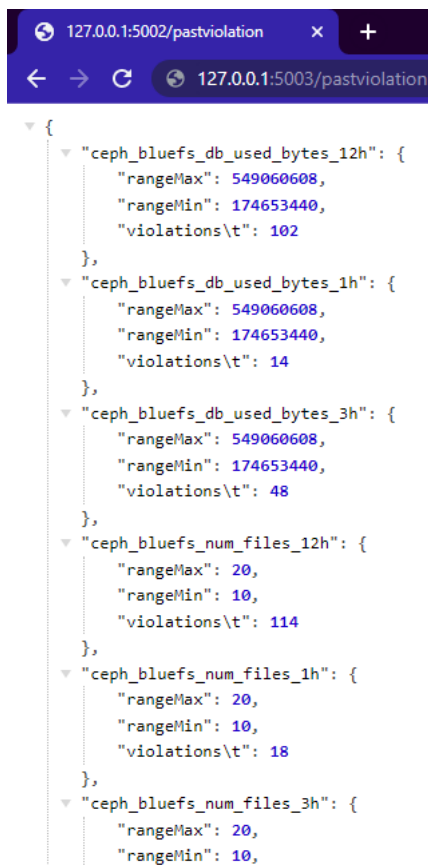


Figura 2.8 Violazioni passate

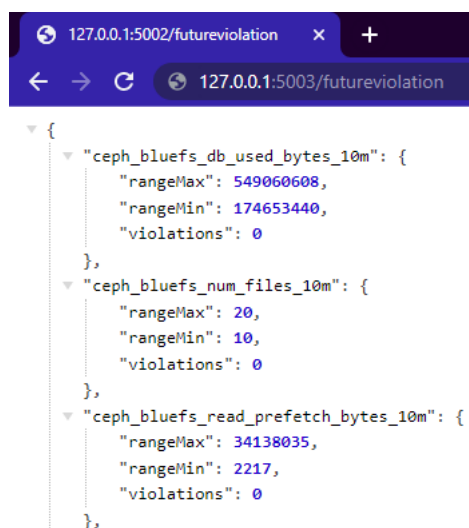


Figura 2.9 Violazioni future

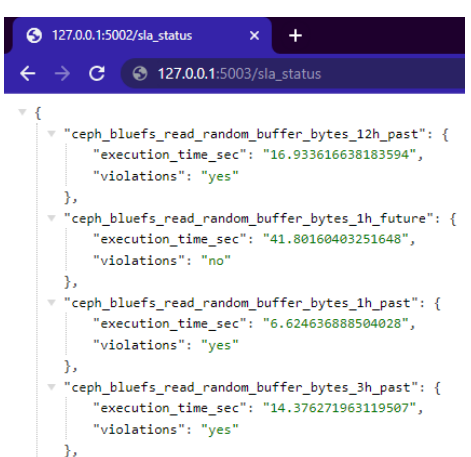


Figura 2.7 Stato dello SLA Manager