



UNIVERSITÀ
degli STUDI
di CATANIA

Department of Electrical, Electronic and Computer Engineering

MASTER'S DEGREE IN COMPUTER ENGINEERING

Design and implementation of a software for the analysis of a set of metrics

Academic Year 2022-2023

Authors: Andrea Paolo Ventimiglia (1000039024)

1 Summary

INTRODUCTION.....	II
1. Project structure	3
1.1. Project components	3
1.1.1. Prometheus	3
1.1.2. Kafka	3
1.1.3. Docker	4
1.2. Microservizi	5
2. Installation and operation	8
2.1. Installation	8
2.2. Operation	8
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

INTRODUCTION

The project described in the following report involves the creation of a software based on four microservices that communicate with each other. These will have to take a set of metrics provided by a Prometheus exporter, analyze them in order to calculate different statistics and store everything within a MySQL database that will contain several tables inside.

The microservices are: ETL Data Pipeline, Data Storage, Data Retrieval and SLA Manager.

The microservices will communicate with each other through a Kafka broker who will manage the exchange of different messages.

Each microservice will perform a very specific task:

- ETL Data Pipeline will connect to a Prometheus server and request some metrics. Once the metrics have been obtained, the microservice will process the data obtained to calculate various useful values such as Seasonality, Stationarity, Autocorrelation, etc. The results obtained will be sent as messages to a Kafka broker.
- Data Storage will receive, through a Kafka broker, messages from the other microservices. The microservice will examine the messages and record the data in appropriate tables contained within the MySQL database.
- Data Retrieval is responsible for providing a rest api interface to display the results contained in the database.
- SLA Manager will aim to calculate possible past and future violations of a set of five metrics and the status of the SLA Manager. Again, the results will be sent via messages to the broker Kafka. You will also be able to visualize the results obtained through a REST API interface.

1. Project structure

1.1. Project components

The following design uses the components briefly described in the following paragraphs.

1.1.1. Prometheus

Prometheus is a free software application used for monitoring and reporting events. Record real-time metrics in a time-series database built using an HTTP pull model, with flexible queries and real-time alerts. Prometheus data is stored in the form of metrics. Each metric can be analyzed based on an arbitrary number of key=value pairs (labels). Labels can include information about the source of the data and other application-specific breakdown information such as HTTP status code, query method, endpoint, etc. The metrics are displayed on Prometheus via an exporter.

The exporter used in the project provides access to Ceph metrics through the following link: <http://15.160.61.227:29090/>

The software created, through queries, queries the server to retrieve the values of the time series relating to the various metrics. These will be useful for calculating values such as Seasonality, Autocorrelation, Stationarity, maximum values, minimum etc.

1.1.2. Kafka

It is an open source software used for micro-service communication, for the exchange of data called messages, for monitoring and for log collection. Kafka is a broker used in distributed systems for two main reasons: to make communication between multiple microservices within a distributed system transparent and asynchronous. A server acts as a broker, while a client can be both a producer and a consumer. Kafka uses topics to exchange messages, which is an addressable abstraction (argument) used to show interest in a given data stream.

In the project, a single topic named "prometheusdata" is created that will be used by the producers and the consumer to exchange messages. Producers create and entrust data to the Kafka broker at the relevant topic while the consumer subscribes to the topic to receive messages in order. In addition, the consumer knows which messages he has already read and which have not yet been read, which is very important for durability.

1.1.3. Docker

Docker is a popular free software designed to run computer processes in isolated, minimal, and easily deployable environments called Linux containers (or even just containers), with the aim of simplifying the deployment processes of software applications. Using Docker, you can package an application and its dependencies into one or more containers that can run on any operating system. Each container shares the underlying operating system while maintaining strong isolation and also each container has its own set of processes, filesystems, network interfaces etc.

The following containers have been created in the project:

- Four related to implemented microservices
- One related to the database
- One related to ZooKeeper
- Uno relativo al broker Kafka

The `docker-compose.yml` file manages and defines the deployed container cluster. Under the heading "services" the various services are declared and then their configurations. The number of services is equal to the number of containers.

For each microservice, you've created a `dockerfile` and a `requirements.txt` file containing some of the dependencies that you'll need to install.

1.2. Microservizi

The project is structured as follows:

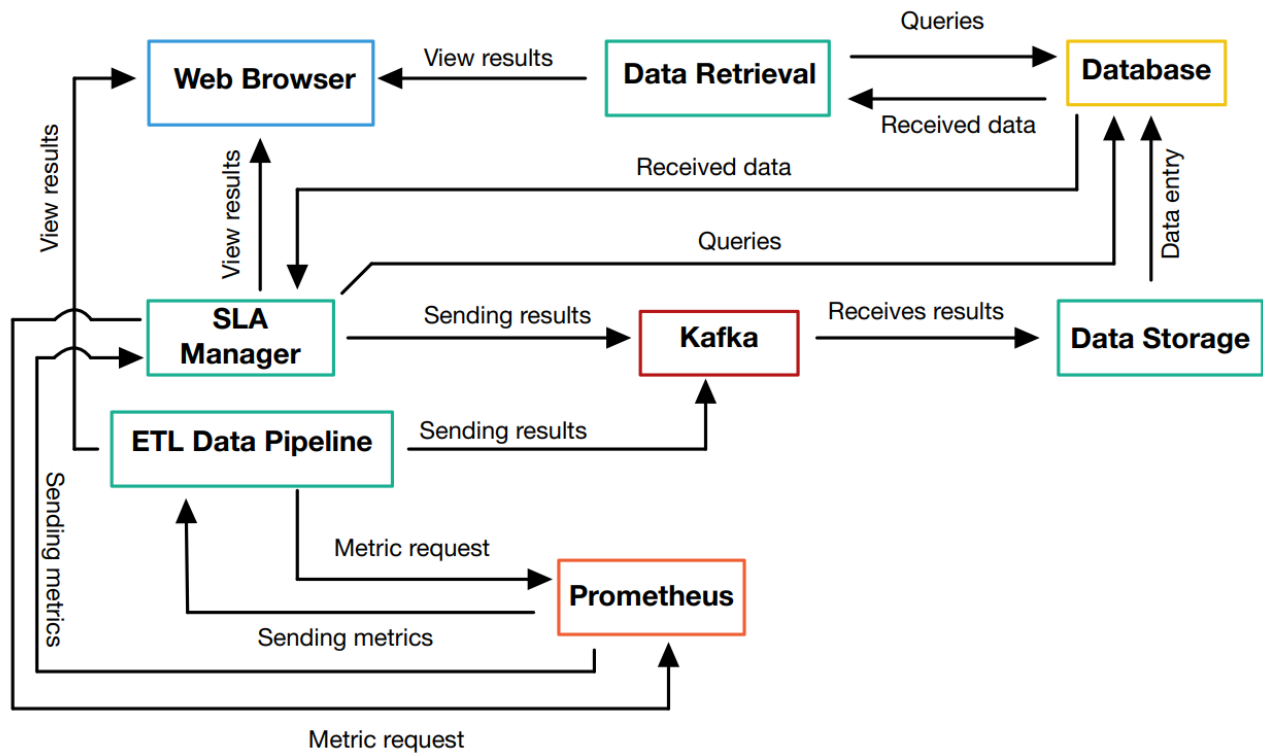


Figure 1.1 Architectural scheme

As we can see, there are four microservices implemented.

The **first**, the ETL Data Pipeline connects to a Prometheus server to request, via query, a list of metrics. With the values of the time series of these metrics, various operations will be carried out through special functions. The operations performed are:

1. Calculation of the Seasonality, Autocorrelation and Stationarity of a set of metadata.
2. The same metrics will be calculated for the values of max, min, avg and dev_std for 1h, 3h and 12h.
3. Using a set of 5 metrics we will calculate the prediction of the max, min and avg values over the next 10 minutes.
4. An internal monitoring system is implemented that makes visible the execution times taken to produce the results of previous operations. These values will be displayed in the middle of the rest calls.

Finally, the results returned, in the points just described, will be forwarded by means of a message to a Kafka topic "promethuesdata".

The **second** microservice, called Data Storage, starts a consumer group of the topic "promethuesdata" and, for each message retrieved, will store the calculated values in a MySQL database called "datastorage" containing the following tables:

- **Metadata:** stores the values calculated in point 1 of the ETL Data Pipeline.
- **Metrics:** stores the values calculated in step 2 of the ETL Data Pipeline.
- **Prediction:** Stores the values predicted in step 3 of the ETL Data Pipeline.
- **Executiontimep1:** stores the execution times of the operations carried out in point 1 of the ETL Data Pipeline.
- **Executiontimep2:** stores the execution times of the operations carried out in point 2 of the ETL Data Pipeline.
- **Executiontimep3:** stores the execution times of the operations carried out in point 3 of the ETL Data Pipeline.
- **Futurevioation:** Stores, for each metric, the number of future violations with their allowable range calculated in the SLA Manager.
- **Pastviolation:** Stores, for each metric and for each start_time, the number of past violations with the relative allowable range calculated in the SLA Manager.
- **Slastatus:** Stores the status of each metric calculated in the SLA Manager.

The **third** microservice, called Data Retrieval, provides a rest interface that allows you to extract the information generated by the ETL Data Pipeline and SLA Manager applications contained in the database.

The possible queries available and visible via rest will be:

- Metadata Queries
- Querying the values of max, min, avg, and std
- Querying predicted values
- Execution time queries
- Query for the list of all metrics
- Querying past and future violations
- SLA Manager Status Query

The **fourth** microservice is the SLA Manager, which provides a list of metrics in the database to the user to check for possible violations. The user must choose five metrics with the relevant eligibility

ranges, these can be typed on the keyboard or calculated automatically. Once the set of 5 metrics has been selected, any violations in the last 1, 3, 12 hours will be calculated through appropriate functions and will provide an indication of a possible violation in the next 10 minutes.

Finally, the results obtained will be forwarded via a message to a Kafka topic "promethuesdata" to be saved in the database and will also be made visible via rest.

Possible queries are:

- SLA Manager status
- Number of violations in the last 1, 3, 12 hours
- number of possible future violations.

2. Installation and operation

This chapter describes the essential components to be installed for the execution of the project, the operation and the description from the implementation point of view of the software.

2.1. Installation

The required components are as follows:

- Docker
- Kafka
- Flask, for rest calls
- Postman (optional, useful for viewing get requests with the possibility of downloading them in the form of json files)
- Python 3.10.9
- MySQL

2.2. Operation

2.2.1. Docker

After installing all the required components, you will need to open Docker and open the terminal in the "DSBD_Esame" folder containing the "docker-compose.yml" file. Once this is done, run the command:

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

Next, you can run the various microservices.

2.2.2. ETL Data Pipeline

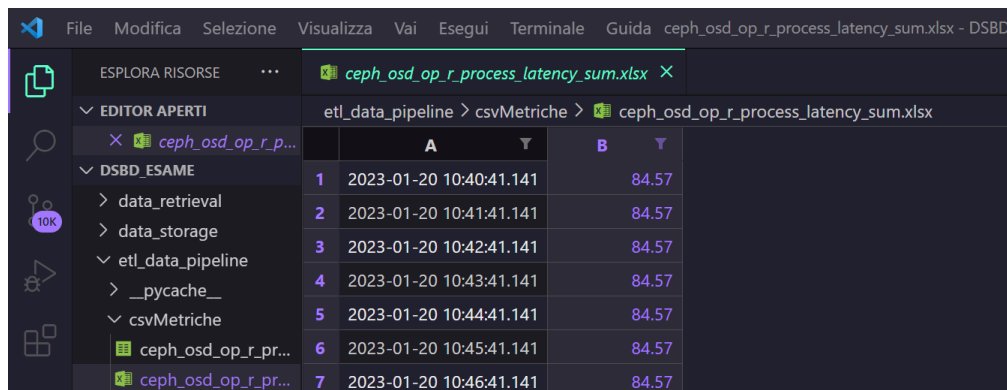
In this microservice, you first configure a Prometheus server using the *PrometheusConnect()* function to scrape the exposed Ceph exporter metrics. The server in question is: <http://15.160.61.227:29090/>

Once connected to the server, the metrics to be processed are filtered using the *prometheus_metrics()* function which allows you to select the metrics with a job equal to "ceph-metrics". It also implements an algorithm to avoid taking metrics with a value of 0.

Once the metrics have been obtained, the *setting_parameters()* function retrieves the values of the relative time series from the prometheus server. These values are sent to the *valuesCalculation()* function, and using the *autocorrelation()*, *stationarity()*, and *seasonality()* functions, the autocorrelation, stationarity, and seasonality of the metrics are calculated, respectively. The results returned are saved in the relative "result_metadata" dictionary.

Subsequently, the `setting_parameters()` function is called again, setting the `start_time` to the set times (1h, 3h and 12h) and the results obtained are sent to the `calculate_values()` function which calculates the required values of max, min, avg and std. These calculated values are saved in the relevant "result_values" dictionary.

The `setting_parameters()` function is invoked a third time to calculate the values of a set of five predetermined metrics. The results are sent to two functions: `creating_file_csv()` which creates a csv file with the time-value pair useful for being able to analyze the values of the time series. The second is the `predictFiveMetrics()` function that predicts the values of max, min, avg in the next 10 minutes. In this function there are the parameters "trend", "seasonal" and "seasonal_periods" which are set according to the chosen metrics since each metric has its own prediction model. The prediction results are also saved in the prediction dictionary "result_predict".



	A	B
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

Figure 2.1 File csv

The results obtained (`result_metadata`, `result_values`, `result_predict`), together with the execution times taken to produce them, are sent to the `kafkaResultProducer()` function which configures a Kafka producer that has the task of forwarding the data to the topic named "prometheusdata" created earlier.

Finally, using Flask you can display the calculated execution time values via rest at the following address: <http://127.0.0.1:5000/>

Within it there are the following calls:

- 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_p3

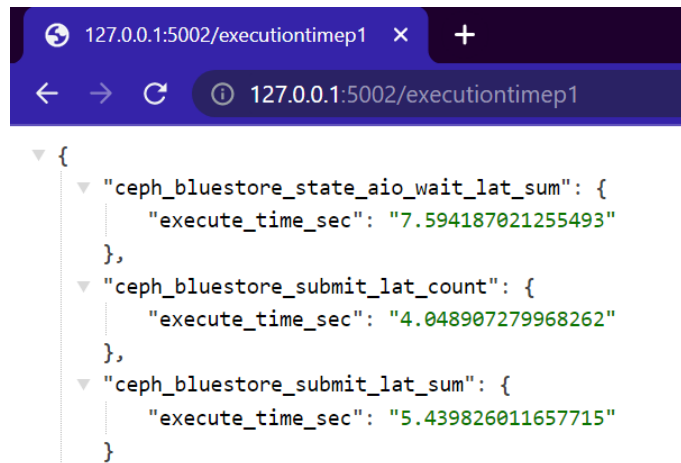


Figure 2.2 Get step 1 execution times

2.2.3. Data Storage

In this microservice, the MySQL database "datastorage" is initially configured and a Kafka consumer that will subscribe to the topic "prometheusdata". As the microservices "ETL Data Pipeline" and "SLA Manager" forward messages to the topic, Data Storage, depending on the request, distinguishes the type of message, converts it from string to json file and inserts the values in the relative tables.

2.2.4. Data Retrieval

The goal of this microservice is to take data from the "datastorage" database and visualize it via rest. Initially, the database is configured and then several queries are implemented. The queries are available at the following link: <http://127.0.0.1:5002/>

Within it there are the following calls:

- <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/pastviolation>
- <http://127.0.0.1:5002/futureviolation>
- 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>

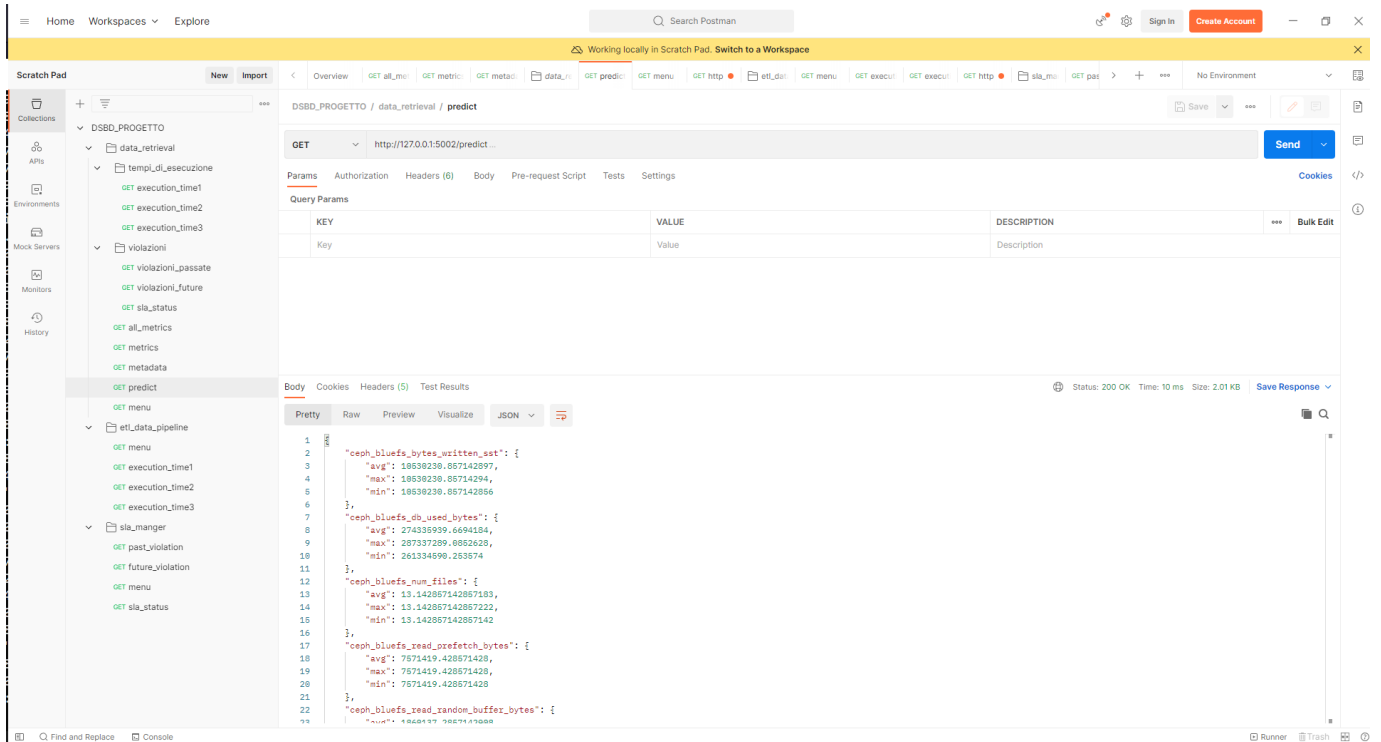


Figure 2.3 Postman page of the various get requests

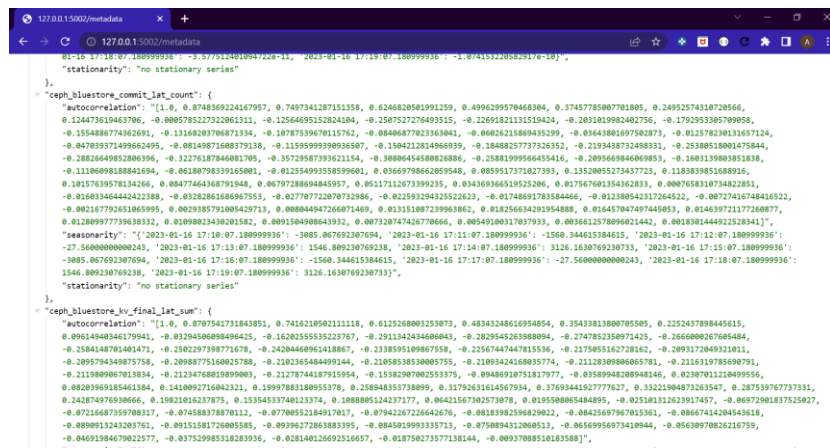


Figure 2.4 Seasonality, autocorrelation, and stationarity of metrics

Figure 2.5 Max, min, avg e std

2.2.5. SLA Manager

This microservice is responsible for verifying whether the metrics that are analyzed are violated. Initially, the *enterMetricCodes()* function is invoked which prints on the screen the list of metrics previously stored in the database, as shown in the figure below.

```
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_procache:data_pri11_bytes
8 - ceph_bluestore_procache:kv_committed_bytes
9 - ceph_bluestore_procache:kv_onode_pri1_bytes
10 - ceph_bluestore_procache:kv_pri1_bytes
11 - ceph_bluestore_procache:meta_pri11_bytes
12 - ceph_bluestore_procache_cache_bytes
13 - ceph_bluestore_procache_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
```

Figure 2.6 List of available metrics

This function allows the user to enter the code of five metrics of his choice from those listed. Next, checks are applied to verify that the metric code entered is correct.

The list of chosen metrics is passed to the *enterAllowedValues()* function which gives the user the possibility to manually enter the max and min values or have the range of allowable values automatically calculated. In the first case, the user will enter the maximum and minimum value from the keyboard (checks will also be made to check if the values entered are acceptable or not). In the second case, the *setting_parameters()* function will be invoked, which takes the values produced by the metric for a certain start_time and calculates the maximum and minimum of these.

Once the admissible range values have been obtained, we move on to the *pastViolations()* and *futureViolations()* functions which have the task of calculating the number of violations of the analyzed metrics by verifying that the value assumed by the metrics is within the selected range or not.

The *pastViolations()* function calculates for each metric the number of violations in the last 1h, 3h, and 12h. While the *futureViolations()* function calculates the prediction of each metric and checks for violations in the next 10 minutes.

The results of the functions are entered into the respective "past_violation_list" and "future_violation_list" dictionaries. For both functions, a dictionary called "sla_status" is implemented that stores the state of the metrics.

The results obtained (past_violation_list, future_violation_list, sla_status) are sent to the *kakfaResultProducer()* function which forwards them to the topic "prometheusdata" which in turn sends them to the Data Storage to be inserted into the database.

Finally, using Flask you can display the values obtained via rest at the following address:

<http://127.0.0.1:5003/>

Within it there are the following calls:

- http://127.0.0.1:5003/past_violation
- http://127.0.0.1:5003/future_violation
- http://127.0.0.1:5003/sla_status



Figure 2.8 Past violations

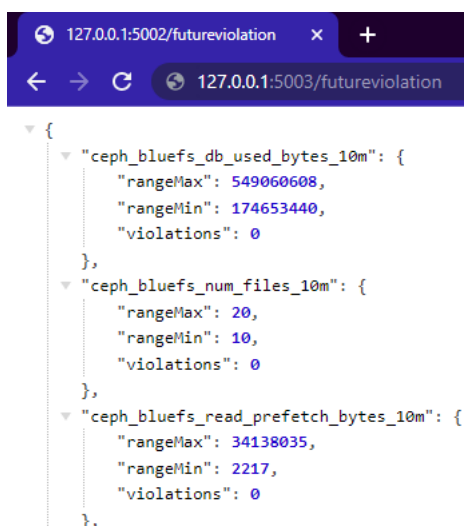


Figure 2.9 Future violations

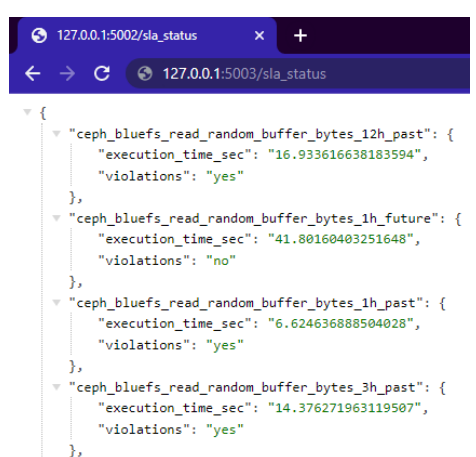


Figure 2.7 SLA Manager Status