**Task 1: Understanding the Basics**

Service Level Objectives (SLOs) and Service Level Indicators (SLIs) play crucial roles in Site Reliability Engineering (SRE) by ensuring the reliability and availability of software services. These concepts are essential in maintaining the dependability and accessibility of services.

**SLO (Service Level Objective):**

An SLO sets a standard for the reliability and effectiveness of a service, representing a commitment to users. It usually specifies uptime percentage over a specific period, like achieving 99.9% availability in a month. SLOs are vital for aligning technical and business goals by defining expected service reliability.

**SLI (Service Level Indicator):**

An SLI is a measurable indicator that reflects how well a service is performing. It is used to monitor the actual performance of a service and is selected to accurately reflect the user experience. Examples of SLIs include response time, error rates, throughput, and availability. SLIs are important because they offer specific data to evaluate if SLOs are being achieved and to identify issues when they are not.

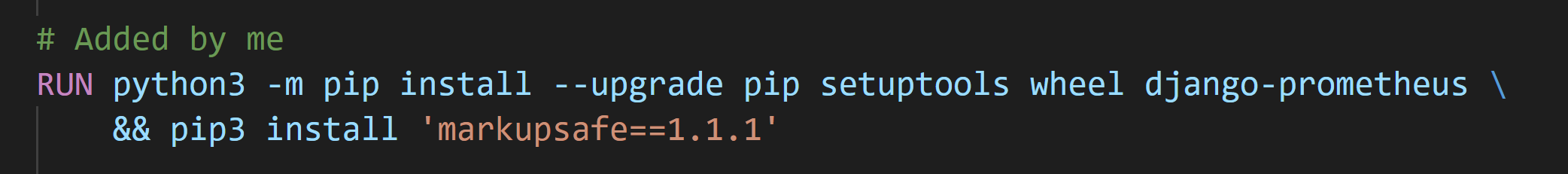
**Significance in SRE:**

SRE focuses on building scalable and highly reliable software systems, with SLOs and SLIs serving as essential frameworks for assessing and managing reliability. SLOs define reliability objectives aligned with user expectations, while SLIs quantify the user experience and identify performance issues. The interaction between SLOs and SLIs guides data-driven decisions on resource allocation and improvement efforts. Achieving SLOs consistently indicates reliable system performance, while unmet SLOs prompt reviews and enhancements to address underlying issues. Overall, SLOs, SLIs, and SRE principles are interconnected and vital for ensuring software service reliability and continual improvement.

**Task 2.**

**Getting started with Shuup:**

Git clone this repository:<https://github.com/shuup/shuup>



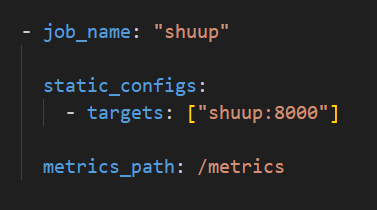
I added these lines of code, because I got the error and after putting this lines of code, the errors was disappear.

Create Prometheus config by PATH “prometheus/prometheus.yml” file in the project,



Add this properties to the Prometheus.yml:

**`metrics\_path`** is a configuration option that defines the URL path used by Prometheus to collect metrics.



After that we should go to **shuup\_workbench\urls.py** andwrite the following code:

from django\_prometheus import exports as django\_prometheus\_exports

add the line

    url(r"^", include('django\_prometheus.urls')),

into “urlpatterns” array.

**`r"^"`** defines the URL pattern to which the view will respond. It uses a regular expression `^`, matching any URL path. Thus, the view will handle requests to any URL starting with a forward slash, representing the root path of the website.

**`django\_prometheus.urls`** refers to the view responsible for exporting metrics in the Prometheus format. It's used by the Django Prometheus package to expose metrics, allowing Prometheus to collect them.

The next part is the `docker-compose.yml` file, where we define configurations to create three indexes:

Volumes for saving data from “Prometheus” and “Grafana”:

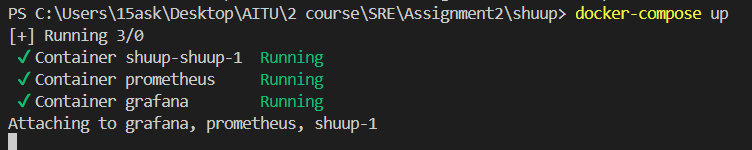
volumes:

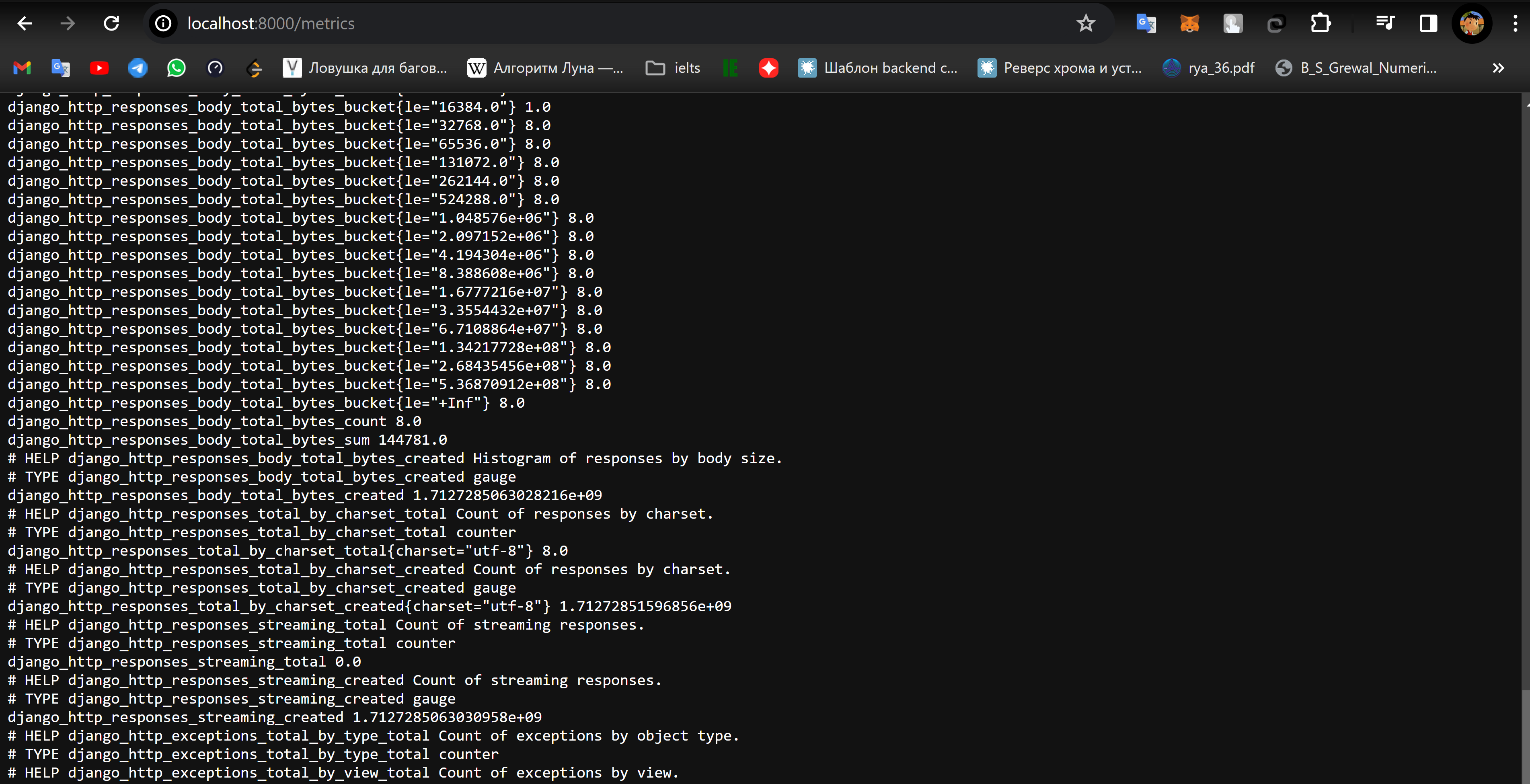
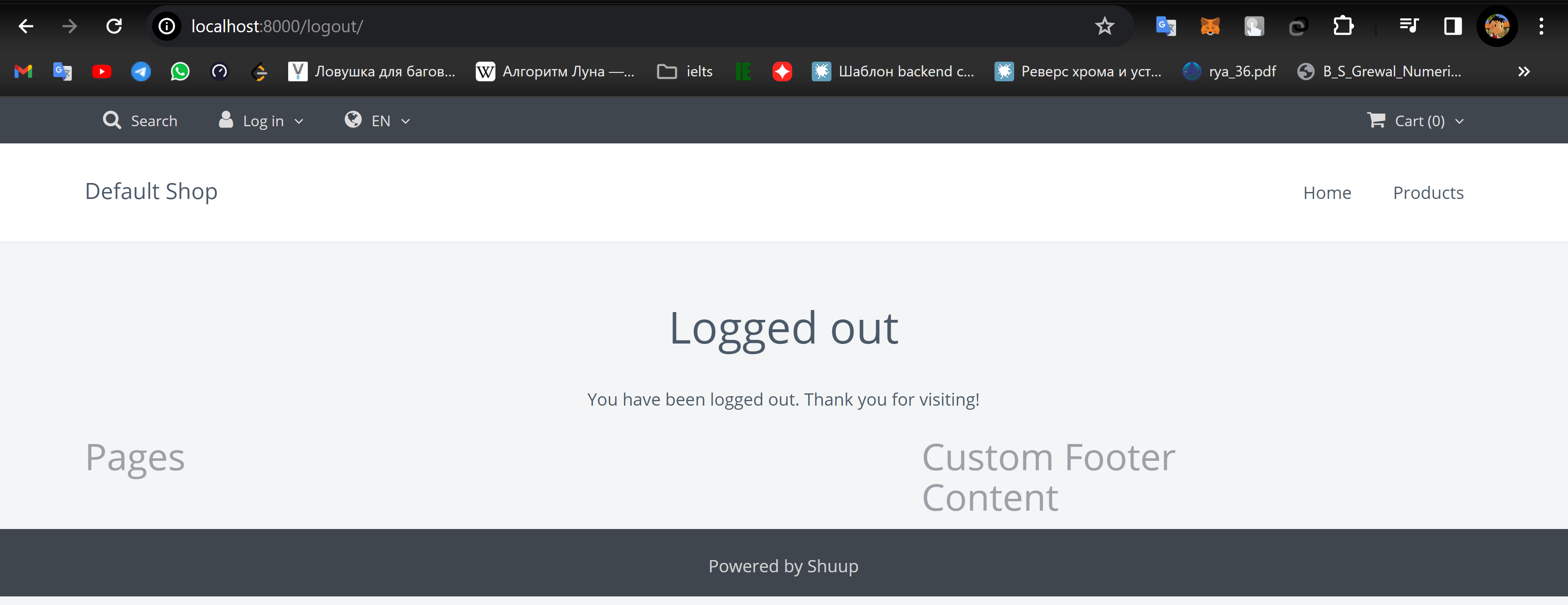
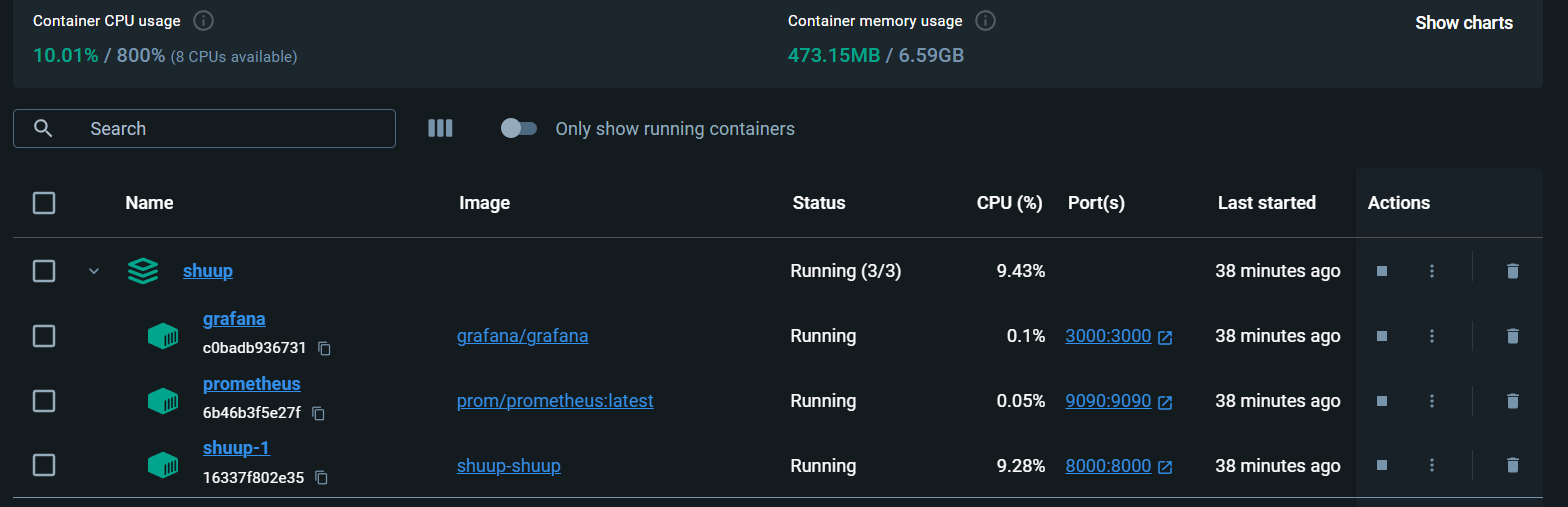
    prometheus\_data:

    grafana\_data:

* shuup
* shuup:
* build:
* context: .
* ports:
* - "8000:8000"
* tty: true
* restart: unless-stopped
* grafana
* grafana:
* container\_name: grafana
* image: grafana/grafana
* depends\_on:
* - prometheus
* ports:
* - 3000:3000
* volumes:
* - grafana\_data:/var/lib/grafana
* - ./grafana/provisioning/:/etc/grafana/provisioning/
* restart: always
* Prometheus
* prometheus:
* container\_name: prometheus
* image: prom/prometheus:latest
* volumes:
* - ./prometheus/:/etc/prometheus/
* - prometheus\_data:/prometheus
* command:
* - '--config.file=/etc/prometheus/prometheus.yml'
* - '--storage.tsdb.path=/prometheus'
* - '--web.console.libraries=/usr/share/prometheus/console\_libraries'
* - '--web.console.templates=/usr/share/prometheus/consoles'
* ports:
* - 9090:9090
* links:
* - shuup:shuup
* depends\_on:
* - shuup
* restart: always

Container started:



Containers in “Docker Desktop”:

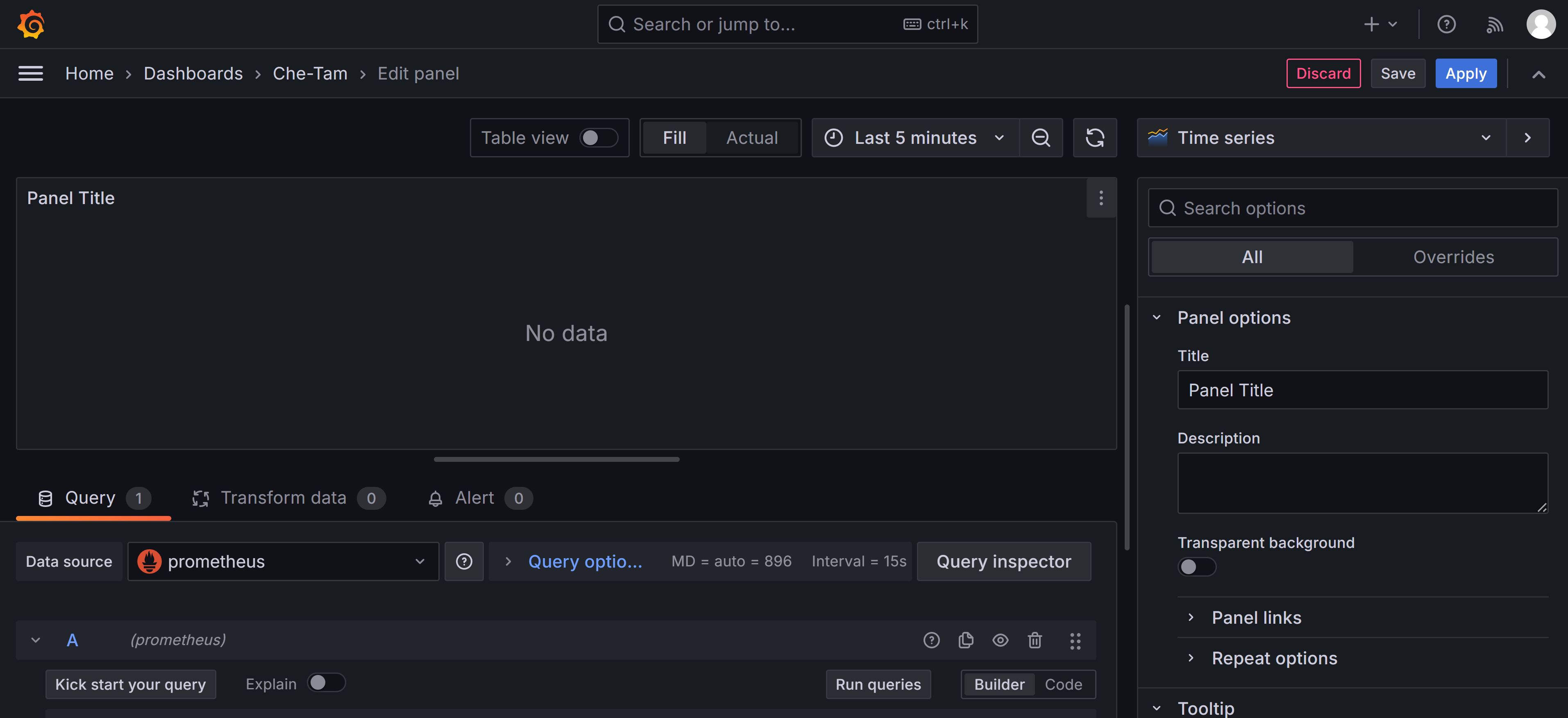
Add connection to Promotheus in Grafana.

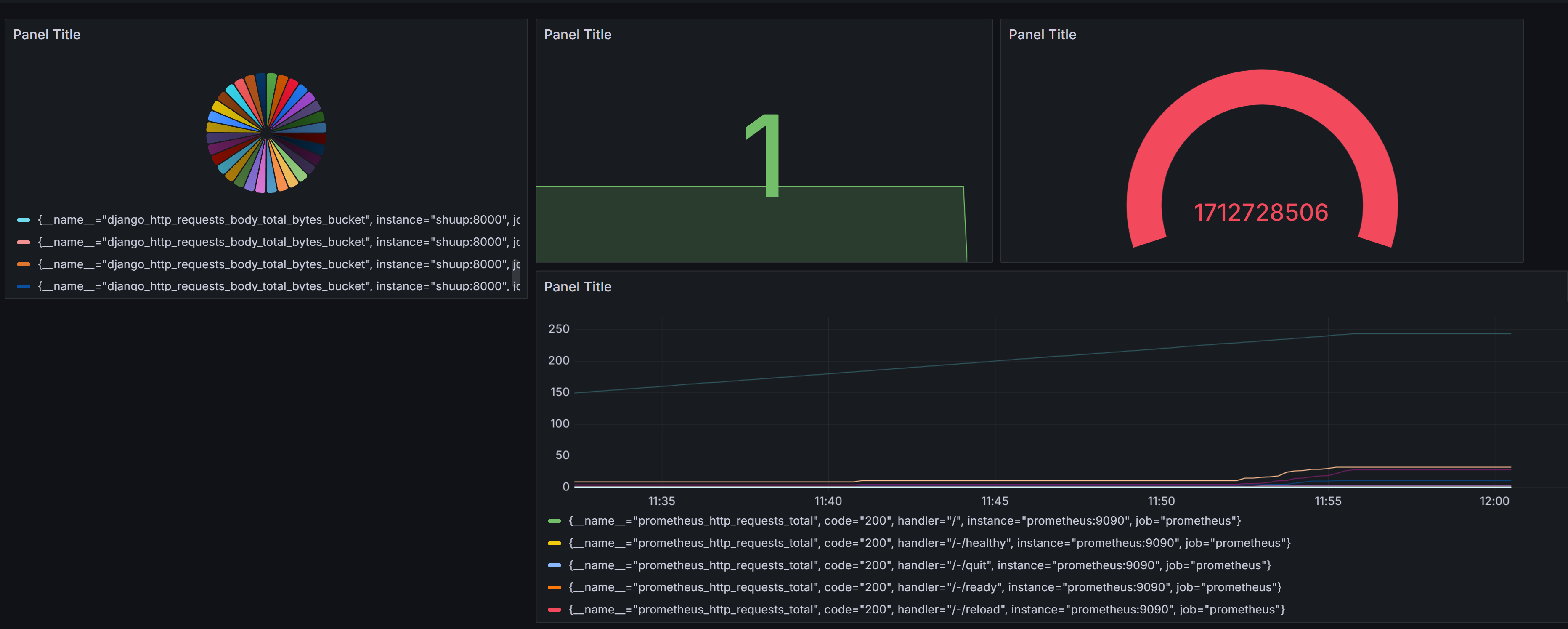
1. Home 🡪 Connections 🡪 Add new connection
2. Select prometheus.
3. Click Button “Add new data source”
4. Input <http://prometheus:9090> into Connection input.
5. Click “Save & Test”

Adding “Dashboard” for visualization:

1. Click Dashboard 🡪 New 🡪 New Dashboard

After this steps we get Dashboard, where we can execute queries.





Afterwords we create visualizations. Something like this:

A screenshot of a computer

Description automatically generated

Average Request Latency (p50)

Calculates the 50th percentile (median) of response latency for HTTP requests in a Django application.



Success Rate