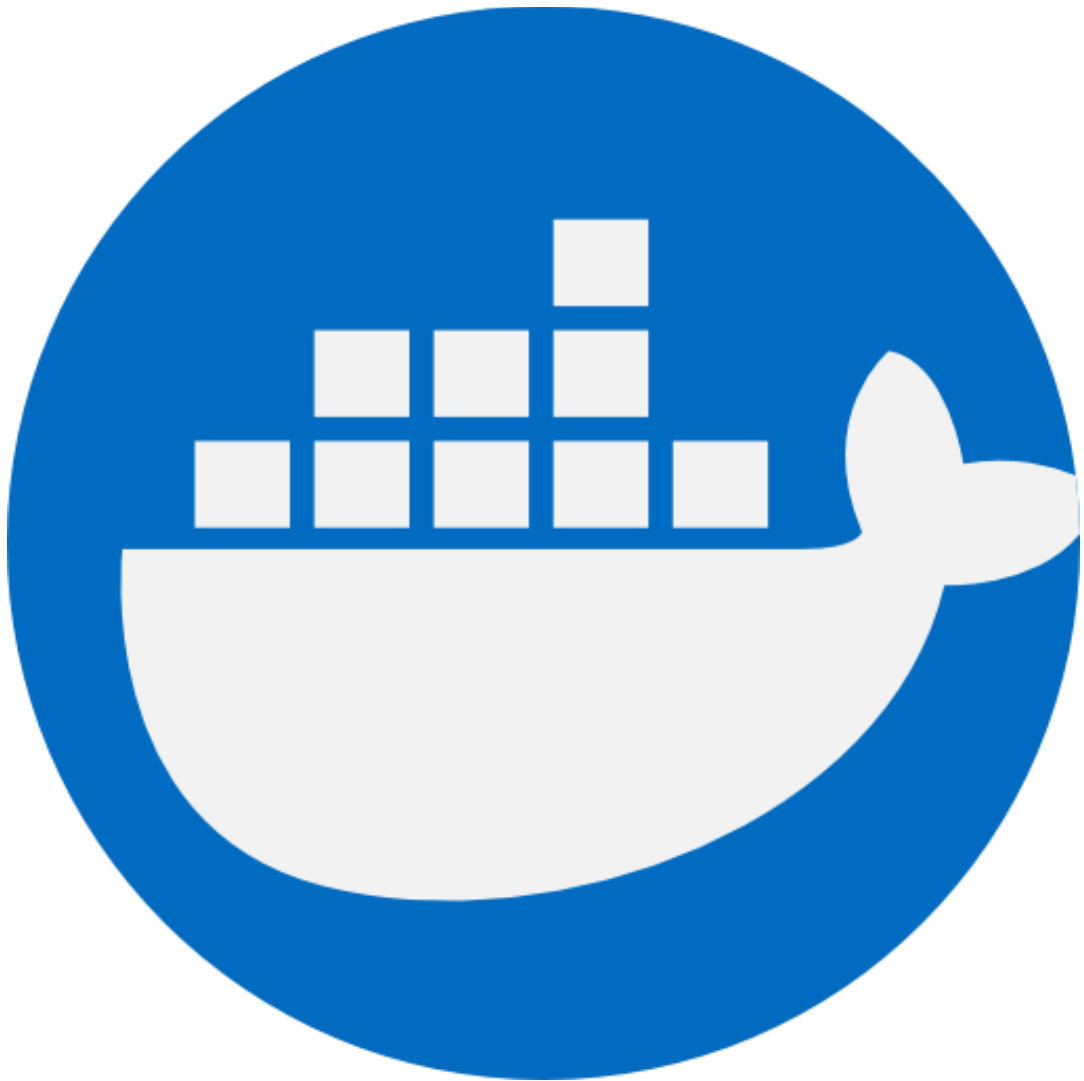


Docker

Sprint 2 Lab 1



Felipe Izquierdo Romero

Índice

<u>1</u>	<u>Descripción del problema</u>
<u>2</u>	<u>Desarrollo</u>
<u>3</u>	<u>Opcional</u>

1. Descripción del problema

Crear una red de contenedores que se llame “todo_lan” que conecte un servicio web de Python con Flask y una base de datos MySQL. Opcional añadir otro servicio o crear otra red con diferentes contenedores.

2. Desarrollo

Para realizar esta red de contenedores se ha de usar un Dockerfile para configurar la imagen de Python con Flask y un Docker-compose para dar las instrucciones de montar la imagen de MySQL y el dockerfile de Python. Además de indicar la red, variables de entorno, mapeo de puertos y volúmenes para administrar los archivos de los contenedores.

En primer lugar, se creará la siguiente estructura de directorios para que el contenedor de Python funcione correctamente y enlazar los volúmenes a sus respectivas carpetas MySQL y Python.

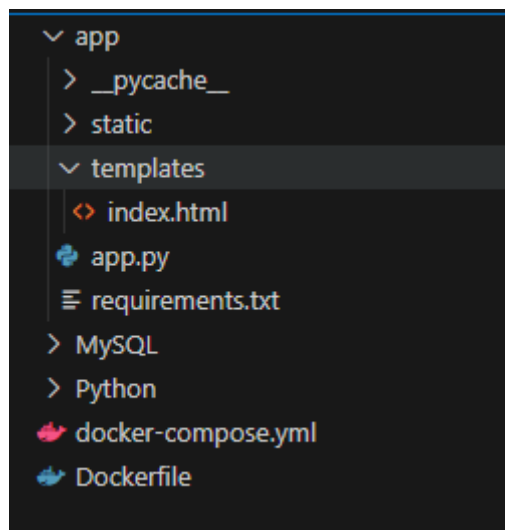


Ilustración 1- Estructura de directorios

A continuación, se creará el dockerfile de Python con sencillas instrucciones

- FROM: imagen de DockerHub de la que se construirá el contenedor.
- WORKDIR: con el directorio de trabajo del contenedor.
- COPY: copiará archivos de tu computadora al contenedor al iniciarse.
- RUN: para correr al iniciarse unos comandos en una terminal.
 - o Estos son para descargar el cliente de mysql y poder conectar flask a MySQL e instalar las librerías necesarias de Flask.
- EXPOSE: puerto donde el contenedor enlazará conexiones.
- CMD: para correr la aplicación de flask.

```

1 FROM python:3.12.3
2
3 WORKDIR /app
4
5 COPY ./app /app
6
7 RUN apt-get update
8 RUN apt-get install default-mysql-client -y
9 RUN pip install --no-cache-dir -r /app/requirements.txt
10
11 EXPOSE 5000
12
13 CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]
14 |

```

Ilustración 2- Dockerfile de Python

Posteriormente se deberá crear el Docker-compose definiendo por etiquetas diferentes configuraciones.

- Services: son los servicios que queremos ejecutar en contenedores que serán dos, Python y mysql.
- Build: para construir nuestra imagen configurada de Python.
- Ports: para mapear el puerto con el contenedor.
- Volumes: creara el volumen enlazando <ruta_local>:<ruta_contenedor>.
- Environment: variables de entorno.
- Depends_on: dependencias del contenedor.
- Networks: la red que se conectaran.

Fuera de la etiqueta services se declara la red con otra etiqueta network. En este caso no se configurarán mas valores para la red y por defecto se creará un bridge (puente) con scope en tu red local.

```

1  services:
2    python:
3      build: .
4      ports:
5        - "5000:5000"
6      volumes:
7        - ./app:/app
8      environment:
9        PYTHON_DB_HOST: mysql
10       PYTHON_DB_USER: root
11       PYTHON_DB_PASSWORD: password
12       PYTHON_DB_NAME: todo_app
13     depends_on:
14       - mysql
15   networks:
16     - todo_lan
17
18   mysql:
19     image: mysql:8.0.36
20     volumes:
21       - ./MySQL:/var/lib/mysql
22     environment:
23       MYSQL_DATABASE: todo_app
24       MYSQL_ROOT_PASSWORD: password
25     networks:
26       todo_lan:
27         aliases:
28           - mysql
29
30   networks:
31     todo_lan:
32

```

Ilustración 3- Docker-compose.yml

Antes de finalizar la configuración se escribirá en el archivo requirements.txt las librerías que Python deberá de instalar al iniciar el contenedor. Flask y flask_mysql.

Para finalizar deberemos abrir una consola de comandos y movernos al directorio raíz donde se encuentran estos dos archivos de configuración e Docker y ejecutar el comando

```
docker-compose up
```

Desde Docker desktop se observará la creación de nuestra red de contenedores y podremos acceder a ellos a través de nuestro navegador.

<http://localhost:5000>

3. Opcional

De manera opcional para este Lab se añadirán dos contenedores más para ofrecer un servicio de monitoreo de la aplicación web. Estos contenedores serán imágenes de Prometheus y Grafana.

Prometheus será el encargado de ofrecernos la funcionalidad para poder usar diferentes métricas y así obtener un análisis del funcionamiento de nuestra web. Para esta ocasión se usarán contadores simples para contabilizar el flujo de nuestra sencilla app y ver cuantas veces se ha ejecutado las funciones declaradas como endpoints.

Grafana se encargará de conectar a la API que Prometheus ofrece y otorgará una interfaz de usuario para visualizar con graficas y tablas los valores que se están midiendo en nuestra aplicación web.

La configuración de estos contenedores se añadirá al Docker-compose.yml de la siguiente manera.

```
32
33 prometheus:
34   image: prom/prometheus:v2.35.0
35   ports:
36     - "9090:9090"
37   volumes:
38     - ./prometheus:/etc/prometheus
39   command:
40     - '--config.file=/etc/prometheus/prometheus.yml'
41   networks:
42     - todo_lan
43
44 grafana:
45   image: grafana/grafana:latest
46   ports:
47     - "3000:3000"
48   volumes:
49     - ./grafana:/var/lib/grafana
50   environment:
51     - GF_SECURITY_ADMIN_PASSWORD=password
52     - GF_SECURITY_ADMIN_USER=admin
53   networks:
54     - todo_lan
55
```

Ilustración 4- Configuración Grafana y Prometheus

En el código de Python se añadirán unos cambios. Se importará la librería `prometheus_client` y de esa librería se crearan tres objetos `Counter(<nombre>,<descripción>)` que serán los contadores.

```
tasks_counter_index = Counter(
    'tasks_counter_index', 'Total de veces que se ejecuta la funcion index'
)
tasks_counter_add = Counter(
    'tasks_counter_add', 'Total de veces que se ejecuta la funcion add'
)
tasks_counter_delete = Counter(
    'tasks_counter_delete', 'Total de veces que se ejecuta la funcion delete'
)
```

Ilustración 5- Contadores

Con la función `inc()` se incrementaran en uno en cada endpoint cada vez que se ejecute esa parte del código que viene a ser cada vez que accedan a la web, creen una task o eliminen una task.

Además, se configurara un endpoint adicional que nos ofrecerá la posibilidad de acceder fuera de Grafana a las métricas de Prometheus.

```

69
70 @app.route('/metrics')
71 def metrics():
72     return Response(prometheus_client.generate_latest(), mimetype='text/plain')
73

```

Ilustración 6- Endpoint metricas prometheus

Tras realizar el paso anteriormente descrito para iniciar los contenedores podremos acceder a Grafana a través de:

<http://localhost:3000/login>

Entraremos con el usuario y la contraseña descritas en el compose “admin” y “password”.

En el menú a la izquierda de la pantalla haciendo clic en DashBoard>Create DashBoard>Add Visualization.

Aparecerá una ventana donde visualizar Datasource que hayamos creado o crearemos uno haciendo clic abajo a la derecha “Configure a new data source”.

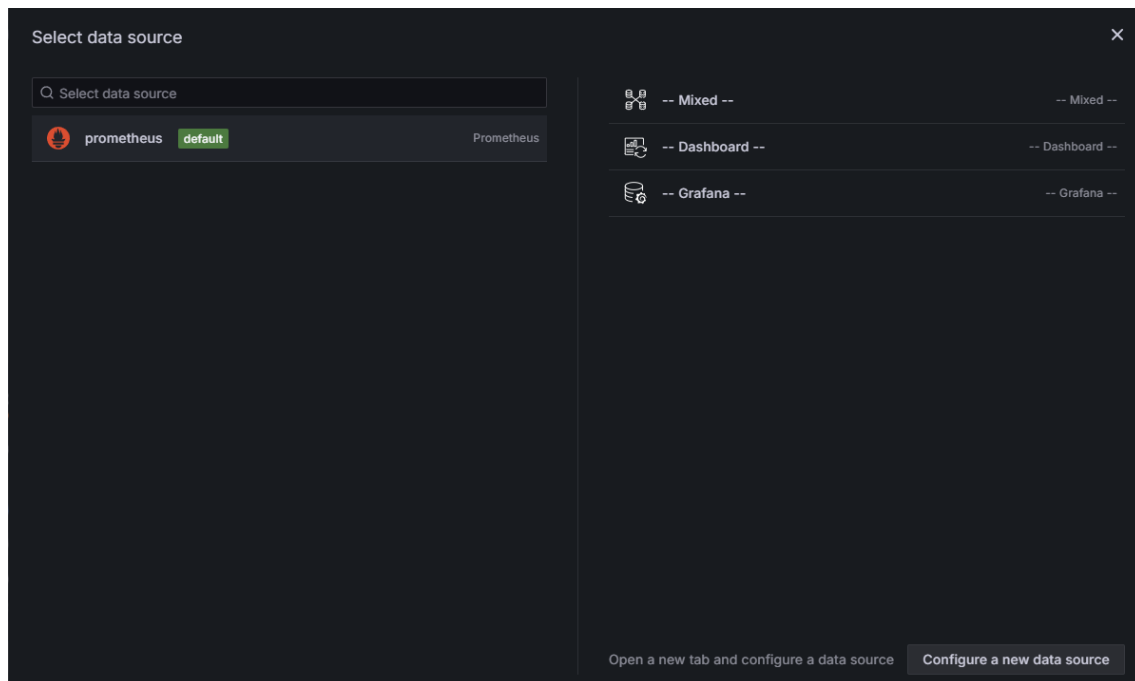


Ilustración 7- Creación DashBorad

Para crear el datasource se debe clicar en prometheus y en la siguiente pestaña de configuración escribiremos el nombre del datasource y la url del servidor de prometheus:

<http://prometheus:9090>.

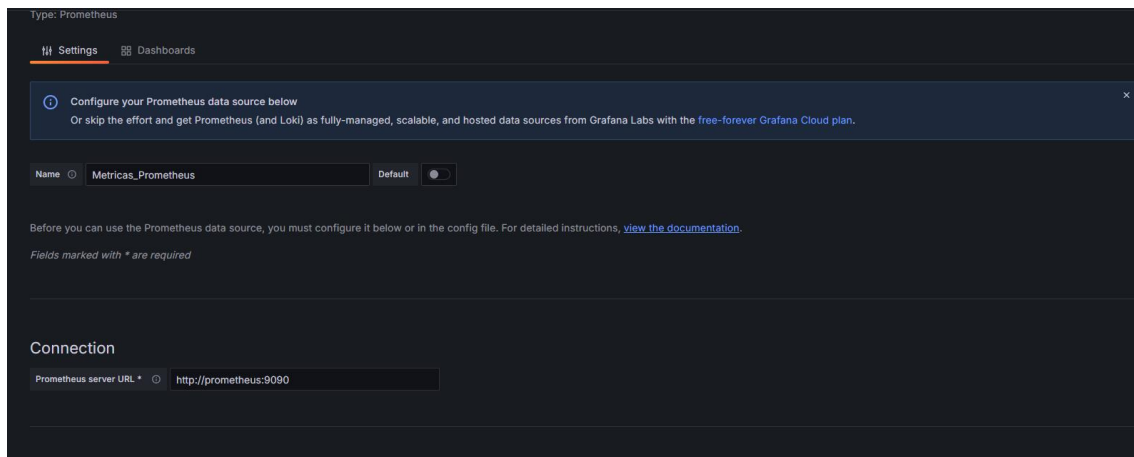


Ilustración 8- Configuración datasource

Abajo del todo clicamos en “Save & test” y ya tendremos un datasource creado ahora repetiremos los pasos hasta llegar a la “Ilustración-7”. Y clicaremos en nuestro datasource.

Posteriormente en metrics explorer podremos encontrar nuestras métricas definidas en nuestro código de Python y crear graficas según nuestras necesidades de monitoreo.

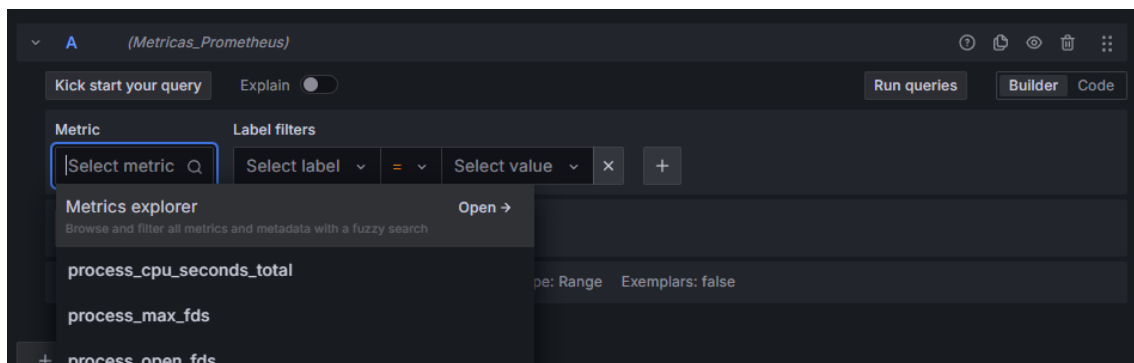


Ilustración 9- Creación de métricas

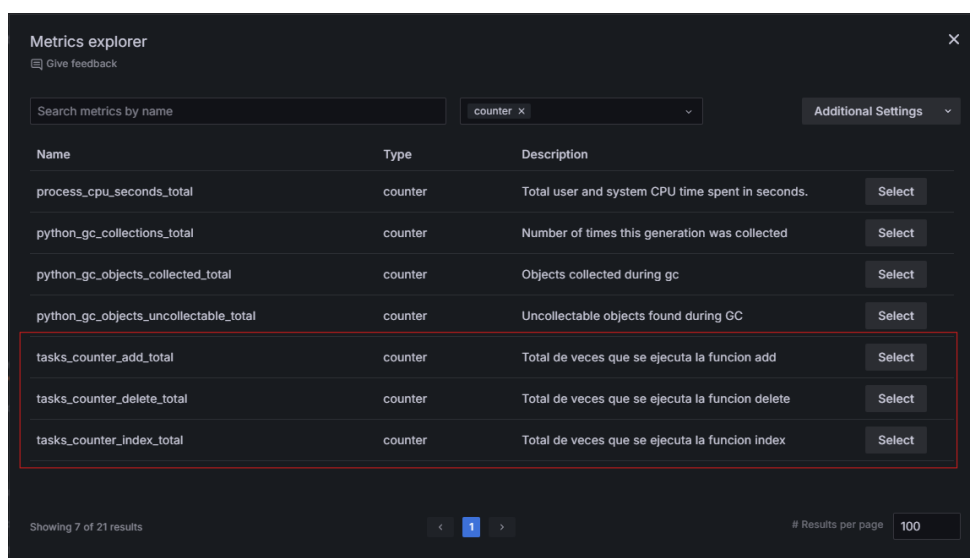
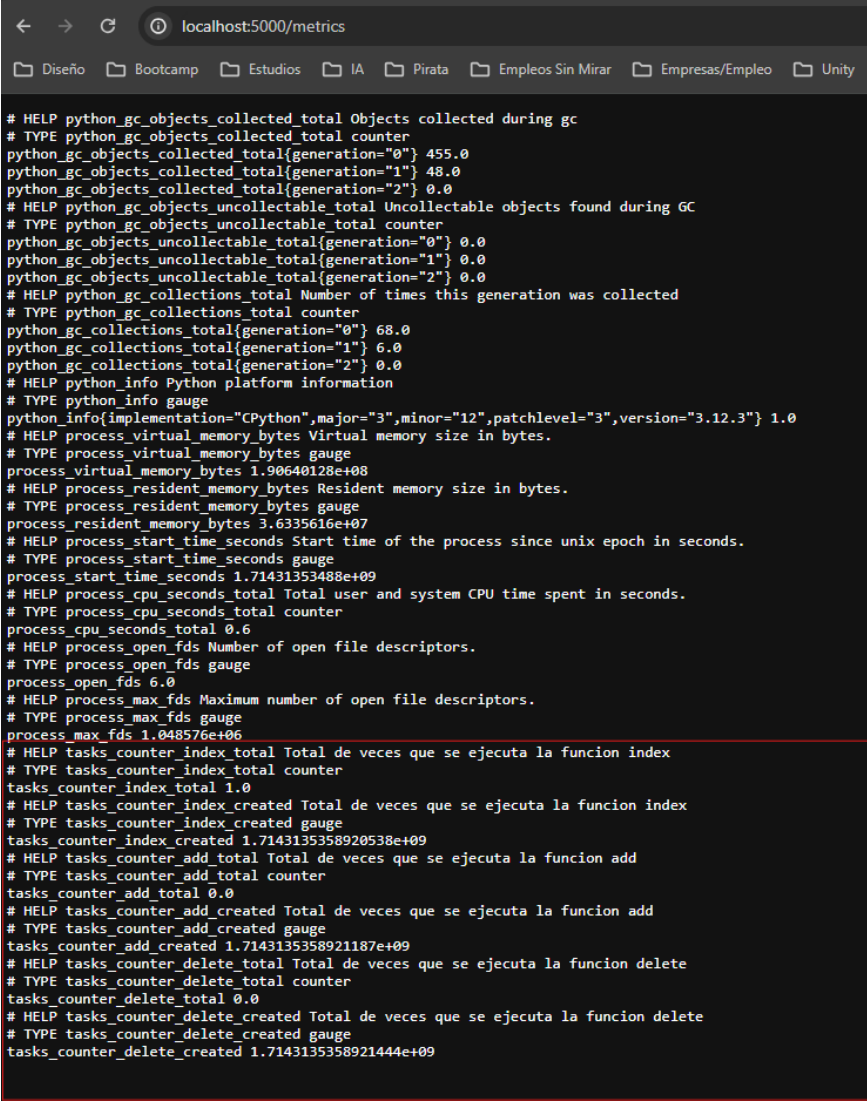


Ilustración 10- Nuestras métricas definidas

Además como se ha mencionado a través de <http://localhost:5000/metrics> podremos ver un JSON con las métricas de Prometheus y abajo del todo las métricas que hemos definido.



```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 455.0
python_gc_objects_collected_total{generation="1"} 48.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 68.0
python_gc_collections_total{generation="1"} 6.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="12",patchlevel="3",version="3.12.3"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.90640128e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 3.6335616e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.71431353488e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.6
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 6.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP tasks_counter_index_total Total de veces que se ejecuta la funcion index
# TYPE tasks_counter_index_total counter
tasks_counter_index_total 1.0
# HELP tasks_counter_index_created Total de veces que se ejecuta la funcion index
# TYPE tasks_counter_index_created gauge
tasks_counter_index_created 1.7143135358920538e+09
# HELP tasks_counter_add_total Total de veces que se ejecuta la funcion add
# TYPE tasks_counter_add_total counter
tasks_counter_add_total 0.0
# HELP tasks_counter_add_created Total de veces que se ejecuta la funcion add
# TYPE tasks_counter_add_created gauge
tasks_counter_add_created 1.7143135358921187e+09
# HELP tasks_counter_delete_total Total de veces que se ejecuta la funcion delete
# TYPE tasks_counter_delete_total counter
tasks_counter_delete_total 0.0
# HELP tasks_counter_delete_created Total de veces que se ejecuta la funcion delete
# TYPE tasks_counter_delete_created gauge
tasks_counter_delete_created 1.7143135358921444e+09
```

Ilustración 11- Metricas Prometheus