

## 75.74 - Distribuidos I - TP1

Mermet, Ignacio Javier 98153

Abril 2022



# Índice

<b>1</b>	<b>Sobre la entrega</b>	<b>3</b>
<b>2</b>	<b>Estructura del proyecto</b>	<b>3</b>
<b>3</b>	<b>Instalación y ejecución</b>	<b>3</b>
<b>4</b>	<b>Arquitectura general</b>	<b>3</b>
4.1	Diagrama de robustez . . . . .	3
4.1.1	Server loop . . . . .	3
4.1.2	Connection dispatcher . . . . .	4
4.1.3	Metrics handlers . . . . .	4
4.1.4	Metrics writers . . . . .	4
4.1.5	Queries handlers . . . . .	4
4.1.6	Notifications workers . . . . .	4
4.1.7	Notifications messages handler . . . . .	4
4.2	Diagrama de clases . . . . .	5
4.3	Diagrama de paquetes . . . . .	5
4.4	Notas sobre escalabilidad . . . . .	5
<b>5</b>	<b>Protocolo de comunicación</b>	<b>6</b>
5.1	Envío de métrica . . . . .	6
5.2	Consulta de métrica . . . . .	7
5.3	Monitoreo de notificaciones . . . . .	8
<b>6</b>	<b>Resolución de concurrencia</b>	<b>8</b>
6.1	Concurrencia entre lectura y escritura de archivos . . . . .	8

# 1 Sobre la entrega

El código de la entrega se puede encontrar en [GitHub](#).

# 2 Estructura del proyecto

El proyecto fue desarrollado en `python`[4] y empaquetado con `poetry`[3]. Tiene dos CLIs asociadas:

- `metrics_server`: el servidor de métricas
- `metrics_client`: cliente para enviar, consultar y monitorear métricas

En la carpeta `docker` se encuentran disponibles los `Dockerfile` asociados tanto al servidor como al cliente.

# 3 Instalación y ejecución

Referirse al archivo `README.md` provisto en el repositorio.

# 4 Arquitectura general

## 4.1 Diagrama de robustez

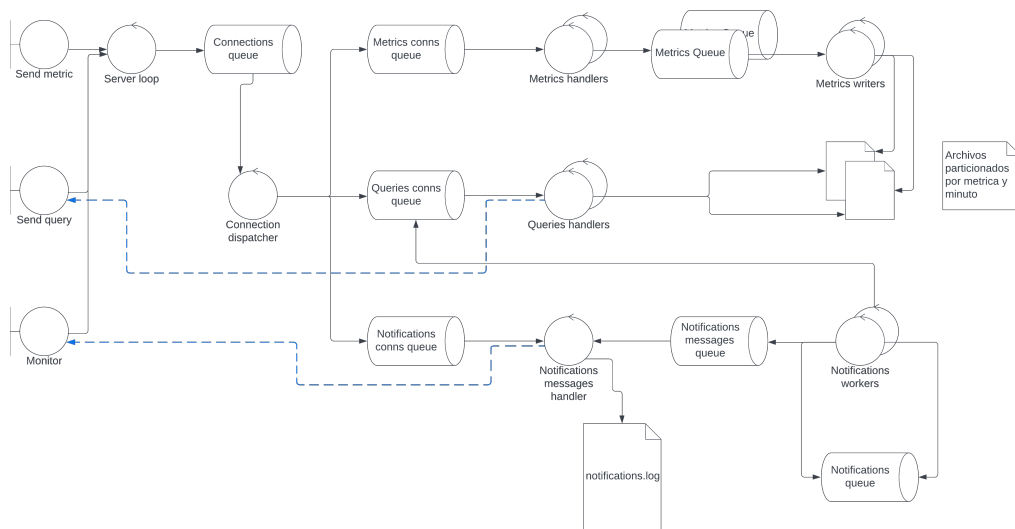


Figura 1: Diagrama de robustez

### 4.1.1 Server loop

El server loop acepta las conexiones entrantes y las mete en **Connections queue** para ser despachadas al *handler* adecuado.

#### 4.1.2 Connection dispatcher

Es proceso se encarga tomar las conexiones entrantes en **Connections queue**, revisar el *intention* declarado por la conexión y luego despacharla a la queue correspondiente a ese *intention*.

#### 4.1.3 Metrics handlers

Estos procesos se encargan de tomar conexiones desde **Metrics conns queue** y recibir todas las métricas enviadas desde ese cliente. Cada métrica recibida se mete en una queue de entre **Metrics queues** según (1).

#### 4.1.4 Metrics writers

Cada proceso de este tipo toma metricas de una queue en particular y se encarga de escribirlas a archivos, particionando por **metric\_id** y por **timestamp**.

#### 4.1.5 Queries handlers

Estos procesos se encargan de tomar una conexión desde **Queries conns queue**, recibir la query enviada desde la conexión, leer los archivos correspondientes a la metrica y período pedidos, y sobre ellos calcular las agregaciones requeridas. Luego envían este valor por la conexión establecida.

#### 4.1.6 Notifications workers

Al levantar el servidor, se lee la configuración de notificaciones y cada elemento se carga en la queue **Notifications queue**. Estos procesos se encargan de tomar una notificacion desde la queue y revisar si es momento de ejecutarla, colocandola de nuevo en la queue si no lo fuera. Si lo fuera, se realiza la query correspondiente. Si la query arrojara un valor que supere el umbral definido, se coloca en la queue **Notifications messages queue** el mensaje correspondiente.

#### 4.1.7 Notifications messages handler

Este proceso se encarga de distribuir a donde corresponde los mensajes de notificaciones. Por un lado escribe a un archivo los mensajes.

Por otro lado, toma conexiones desde **Notifications conns queue** y las agrega a una lista interna de conexiones. Cada nuevo mensaje de notificacion es enviado a estos clientes también.

## 4.2 Diagrama de clases

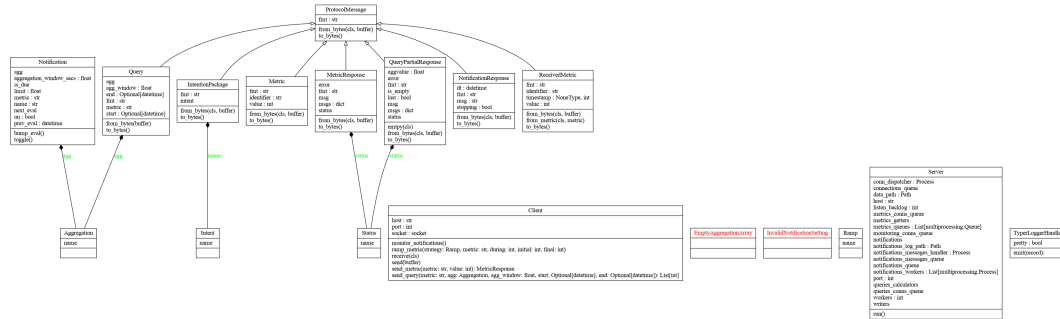


Figura 2: Diagrama de clases - generado automáticamente con `pyreverse`

### 4.3 Diagrama de paquetes

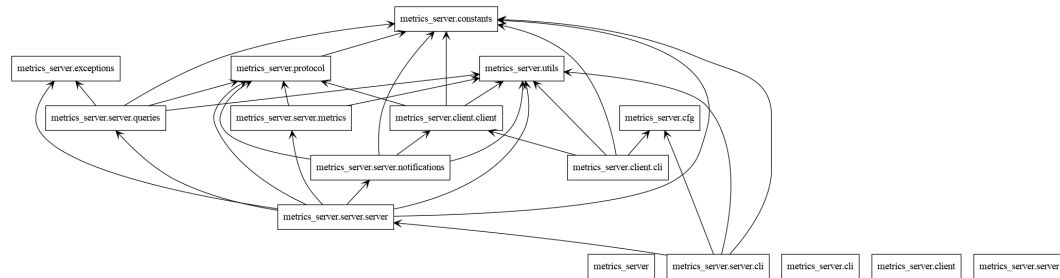


Figura 3: Diagrama de paquetes - generado automáticamente con `pyreverse`

## 4.4 Notas sobre escalabilidad

El sistema permite configurar la cantidad de procesos que procesan métricas, queries y notificaciones. Si fuéramos a recibir muchas conexiones, tener un solo despachador de conexiones podría ser un cuello de botella.

Por otro lado, se puede argumentar que tener un solo manejador de mensajes de notificaciones no sería un cuello de botella: no debieran saltar demasiadas alarmas al mismo tiempo, asumiendo que los sistemas que reportan métricas funcionan razonablemente bien y que las notificaciones están configuradas con valores sensatos.

El uso de colas para manejar la concurrencia hace que varios de los componentes del sistema escalen a un escenario multicomputing. Sin embargo, hay que tener algunas consideraciones.

Asumamos que la cantidad de queues de métricas es constante. Sea  $M_i$  el identificador de la  $i$ -ésima métrica que se ha recibido.  $M_i$  se asigna a la queue  $Q_j$  donde

$$j = \text{crc}(M_i) \quad \text{mód} \quad |\text{Queues de métricas}| \quad (1)$$

El proceso  $W_j$  toma métricas de la queue  $Q_j$  y las escribe a disco, particionadas por identificador de la métrica y por minuto del timestamp de la métrica recibida. Notar que  $M_i$  **siempre** es escrita por  $W_j$ .

Un proceso  $Y_j$  que resuelva consultas sobre una métrica  $M_i$  debe o bien ser ejecutado en la misma máquina que el proceso  $W_j$  o tener acceso al filesystem de  $W_j$ .

El segundo caso no es trivial de resolver. Tecnologías como HDFS[1] involucran una gran cantidad de ingeniería que, se entiende, supera los objetivos del presente trabajo práctico.

Para poder resolver el primer escenario se puede replicar lo que se usa para escribir las métricas: mandar cada query  $Q_x$  sobre la métrica  $M_i$  a una queue. El proceso  $Y_j$  vive en la misma máquina que  $W_j$ , toma la query de la queue y la ejecuta, teniendo acceso a los archivos de  $M_i$ .

Si la cantidad de queues de métricas/cantidad de procesos que escriben métricas a archivos cambiase, se deberían reshufflear los archivos de todas las métricas a la máquina correspondiente. Este proceso podría demorar levantar el server para poder recibir métricas y responder queries.

## 5 Protocolo de comunicación

La comunicación cliente-servidor utiliza un protocolo binario. La comunicación se inicia enviando un paquete *Intention*[2] que indica que tipo de operación se desea ejecutar:

- Enviar métrica
- Consultar métrica
- Monitorear notificaciones

Luego el protocolo de comunicación continúa de acuerdo a cada caso particular.

### 5.1 Envío de métrica

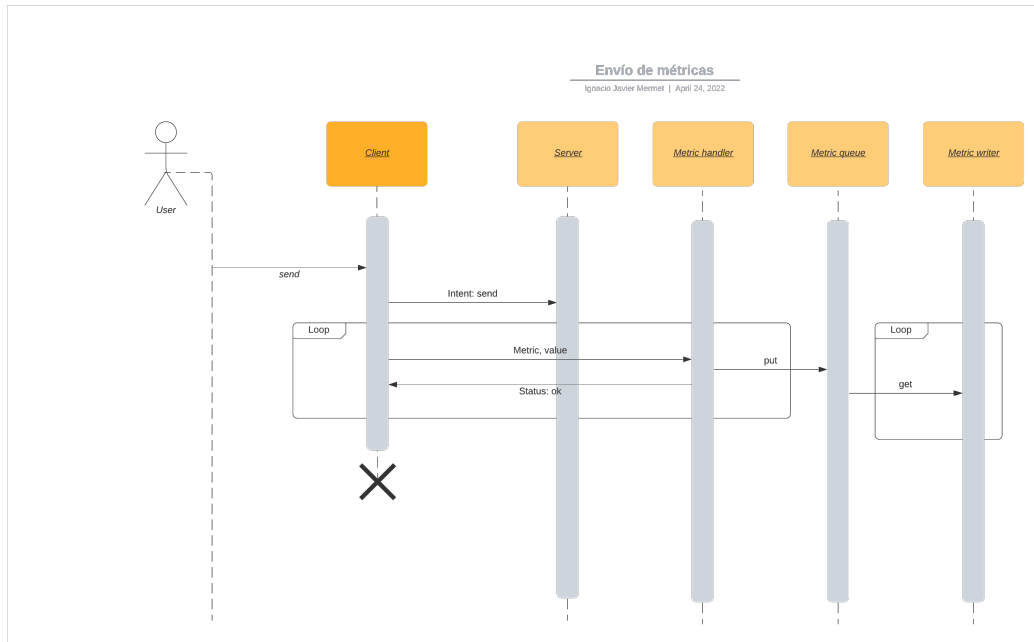


Figura 4: Diagrama de secuencia - envío de métricas

## 5.2 Consulta de métrica

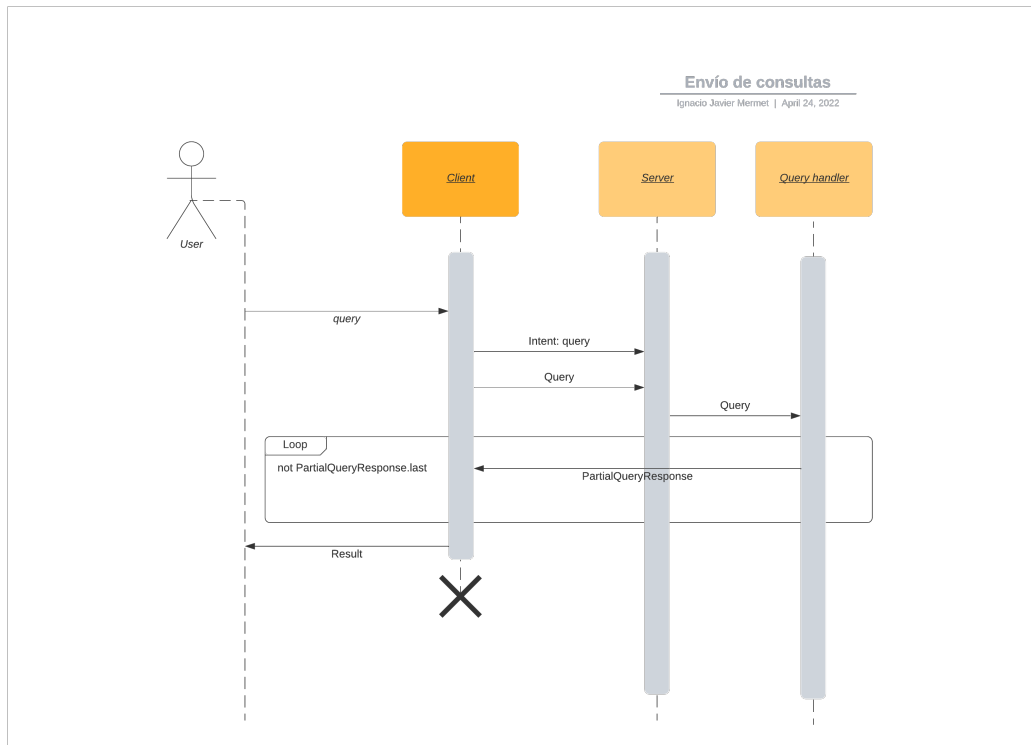


Figura 5: Diagrama de secuencia - envío de métricas

### 5.3 Monitoreo de notificaciones

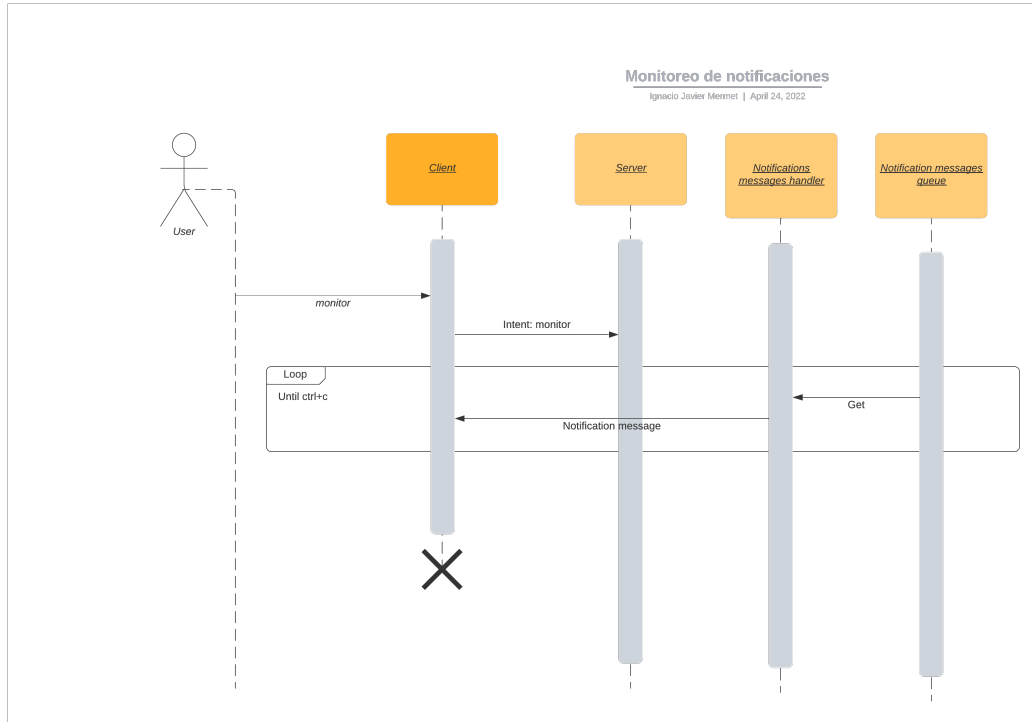


Figura 6: Diagrama de secuencia - monitoreo de notificaciones

## 6 Resolución de concurrencia

La concurrencia entre distintos procesos se resuelve, en su totalidad, con el uso de queues. De este modo podemos escalar fácilmente la cantidad de procesos de cada tipo, tomando su entrada desde la queue correspondiente y, de ser necesario, dejando su resultado en otra queue.

Para entender como se relacionan los procesos y las queues, referirse a [4.1](#).

### 6.1 Concurrencia entre lectura y escritura de archivos

Cada archivo de una métrica  $M_i$  es escrito solo por un proceso  $W_j$ , por tanto no hay concurrencia  $WW$  a resolver. La concurrencia  $RW$  puede darse en el caso que un proceso esté leyendo el archivo para resolver una query mientras otro sigue escribiendo métricas al mismo.

Sin embargo, al leer, lo que puede pasar es que haya líneas incompletas. Estas líneas se ignoran. Por otro lado, se ignoran todas las líneas de momentos anteriores al momento donde se empieza a ejecutar la query. De este modo, dos queries ejecutadas en el mismo instante, reciben la misma respuesta, sin importar el momento de completado de procesamiento de las mismas.



## Referencias

- [1] *Hadoop Distributed File System*. URL: [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html).
- [2] *Intention package*. URL: [https://github.com/CrossNox/7574-TP1/blob/master/metrics\\_server/protocol.py#L31](https://github.com/CrossNox/7574-TP1/blob/master/metrics_server/protocol.py#L31).
- [3] *poetry*. URL: <https://python-poetry.org/>.
- [4] *python*. URL: <https://www.python.org>.