75.74 - Distribuidos I - TP1

Facultad de Ingeniería
Universidad de Buenos Aires

Mermet, Ignacio Javier
98153

Abril 2022

Índice

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 para ver las instrucciones de instalación.

3.0.1 Ejecución del server

A continuación se muestra la ayuda de la CLI asociada al servidor.

```
$ poetry run metrics_server --help
Usage: metrics_server [OPTIONS]

Options:
  --host TEXT                     Host to bind server to  [default: 0.0.0.0]
  --port INTEGER                  Port to bind the server to  [default: 5678]
  --workers INTEGER               Amount of processes handling metrics
                                  [default: 16]
  --backlog INTEGER               How many unaccepted connections the system
                                  allows before refusing new ones  [default:
                                  8]
  --writers INTEGER               Amount of processes writing metrics to disk
                                  [default: 16]
  --queriers INTEGER              Amount of processes handling queries
                                  [default: 4]
  --notifiers INTEGER             Amount of processes reviewing active
                                  notifications  [default: 4]
  --data-path PATH                Path to save data to  [default: /tmp]
  --notifications-log-path PATH   Path to write notification messages to
                                  [default:
                                  /tmp/metrics_server_notifications.log]
  --notifications-cfg TEXT        Path to notifications configuration ini file
                                  [default: /home/nox/repos/fiuba/7574-Distrib
                                  uidosI/7574-TP1/notifications.ini]
  -v, --verbose                   Level of verbosity. Can be passed more than
                                  once for more levels of logging.  [default:
                                  0]
  --pretty                        Whether to pretty print the logs with colors
```

Figura 1: Mensaje de ayuda de la CLI del servidor.

La configuración de la ejecución se hace a través de un archivo `.ini` en el root del proyecto o bien por variables de entorno. La sección 3.2 provee mayor detalle.

3.1 Cliente de ejemplo

A continuación se muestra la ayuda de la CLI asociada al cliente provisto.

```
$ poetry run metrics_client --help
Usage: metrics_client [OPTIONS] COMMAND [ARGS]...

Options:
  --host TEXT                     Host address of the server  [default:
                                  0.0.0.0]
  --port INTEGER                  Port of the server  [default: 5678]
  --retries INTEGER               Retries to connect to the server  [default:
                                  5]
  -v, --verbose                   Level of verbosity. Can be passed more than
                                  once for more levels of logging.  [default:
                                  0]
  --pretty                        Whether to pretty print the logs with colors

Commands:
  monitor  Monitor triggered notifications.
  query    Send a query to the server and print the result.
  ramp     Send a burst of metrics to a server.
  send     Send a single metric to a server.
```

Figura 2: Mensaje de ayuda de la CLI del cliente.

Vemos que hay cuatro subcomandos disponibles: dos modos de envío de métricas, consultas y monitoreo de notificaciones.

Para ver el mensaje de ayuda de alguno de los subcomandos, se puede hacer con `poetry run metrics_client <subcomando> --help`.

3.2 Configuración

En el root del proyecto se encuentra un archivo `sample_settings.ini`, donde se ven los posible valores a configurar:

```
[server]
host=0.0.0.0
port=5678
workers=16
backlog=8
writers=16
queriers=4
notifiers=4
data_path=/tmp
notifications_log_path=/tmp/metrics_server_notifications.log
notifications_cfg=/var/notifications.ini
```

Figura 3: Archivo de configuración de ejemplo.

El proyecto, sin embargo, espera un archivo llamado `settings.ini`. Por motivos obvios de seguridad, este archivo es ignorado en el sistema de versionado.

Puede copiar el archivo de prueba provisto, renombrarlo y modificar los valores según necesidad.

Cada posible configuración se puede sobreescribir con variables de entorno con la nomenclatura `<Sección>_<Clave>`. Por ejemplo `SERVER_HOST`.

4 Arquitectura general

4.1 Diagrama de robustez

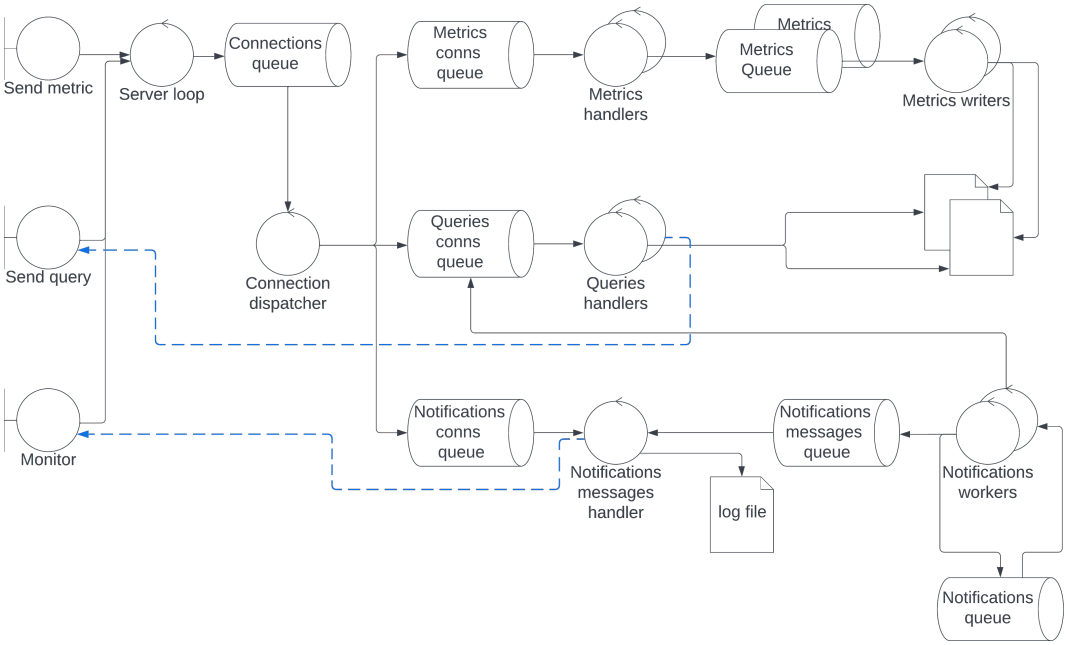

Figura 4: Diagrama de robustez

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 métricas 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 métrica 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 notificación desde la queue y revisar si es momento de ejecutarla, colocándola 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 notificación es enviado a estos clientes también.

4.2 Diagrama de clases

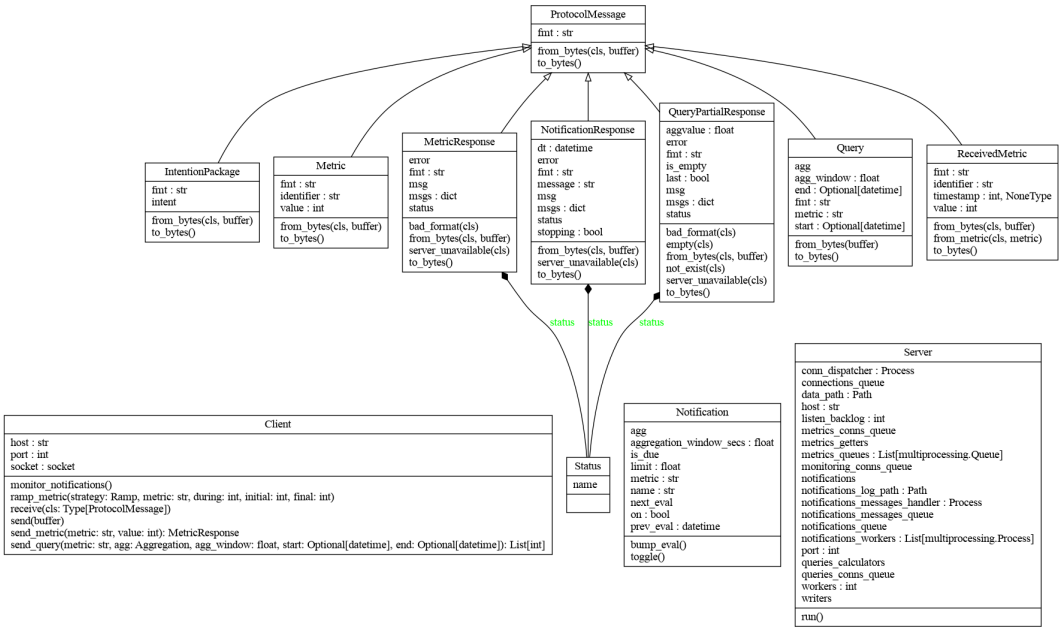

Figura 5: Diagrama de clases - generado automáticamente con `pyreverse`

4.3 Diagrama de paquetes

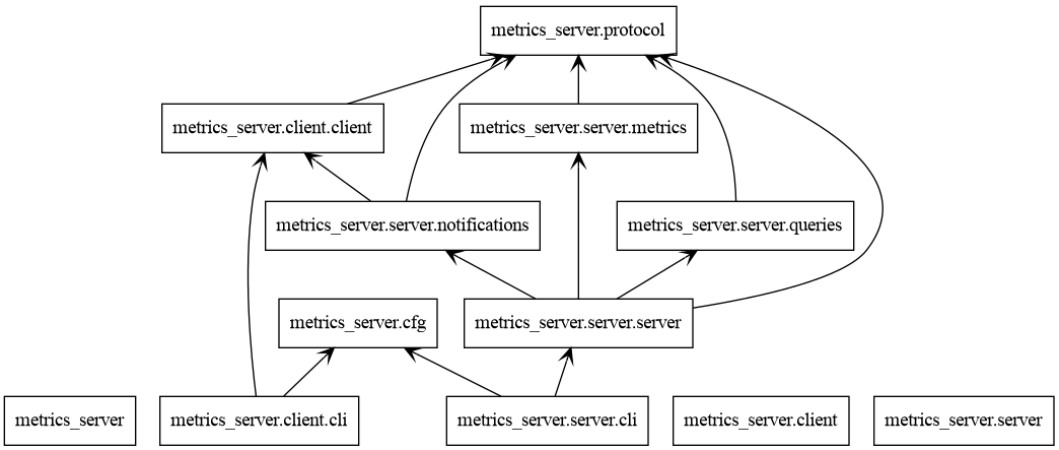

Figura 6: Diagrama de paquetes - generado automáticamente con `pyreverse`

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

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

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. Los tamaños de los paquetes corresponden al struct de `C` subyacente:

Paquete	*String de formato*	*Tamaño del paquete*
IntentionPackage	!i	4
Metric	!28pL	32
Query	!28p12pddd	64
QueryPartialResponse	!Hf?	7
ReceivedMetric	!28pLf	36
NotificationResponse	!d128p?H	139
MetricResponse	!H	2

Notar que se espera tamaño *standard* sin alineación, con bytes en orden de red (big-endian).

Para poder desarrollar un ecosistema de aplicaciones que puedan utilizar este protocolo, se debería desarrollar una guía que explique en profundidad que representa cada campo del string de formato de cada paquete.

5.1 Envío de métrica

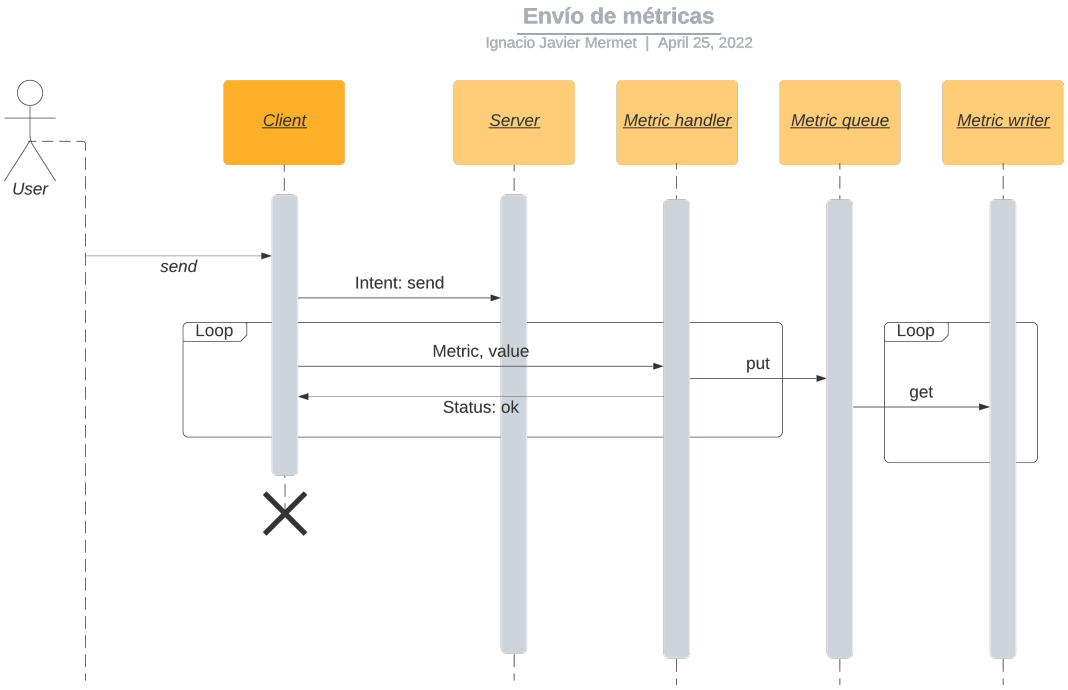

Figura 7: Diagrama de secuencia - envío de métricas

5.2 Consulta de métrica

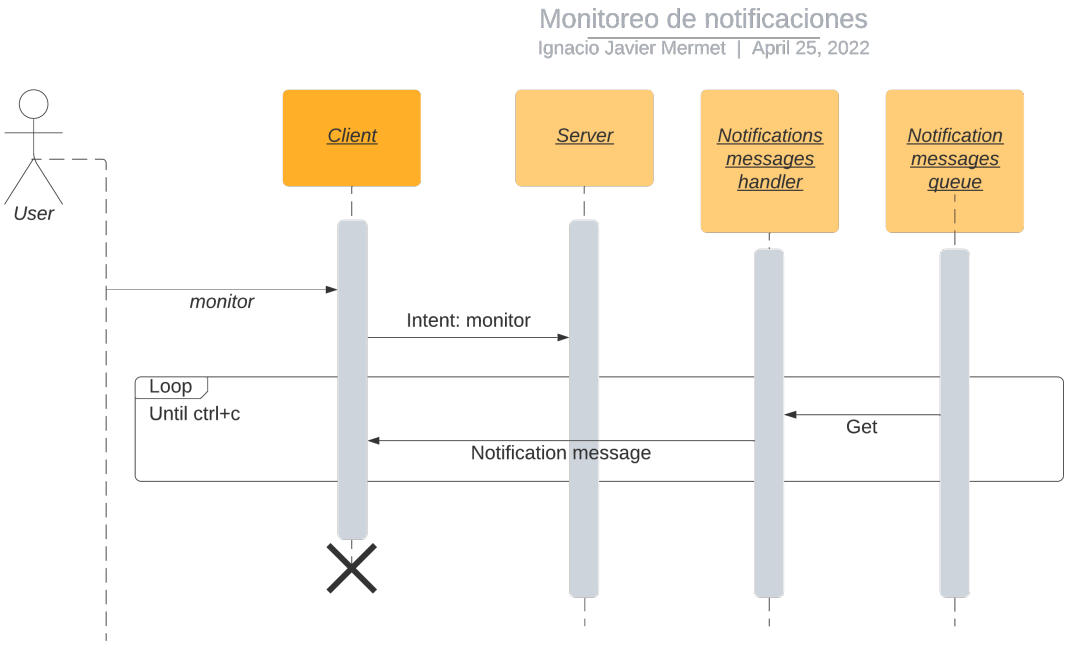

Figura 8: Diagrama de secuencia - envío de métricas

5.3 Monitoreo de notificaciones

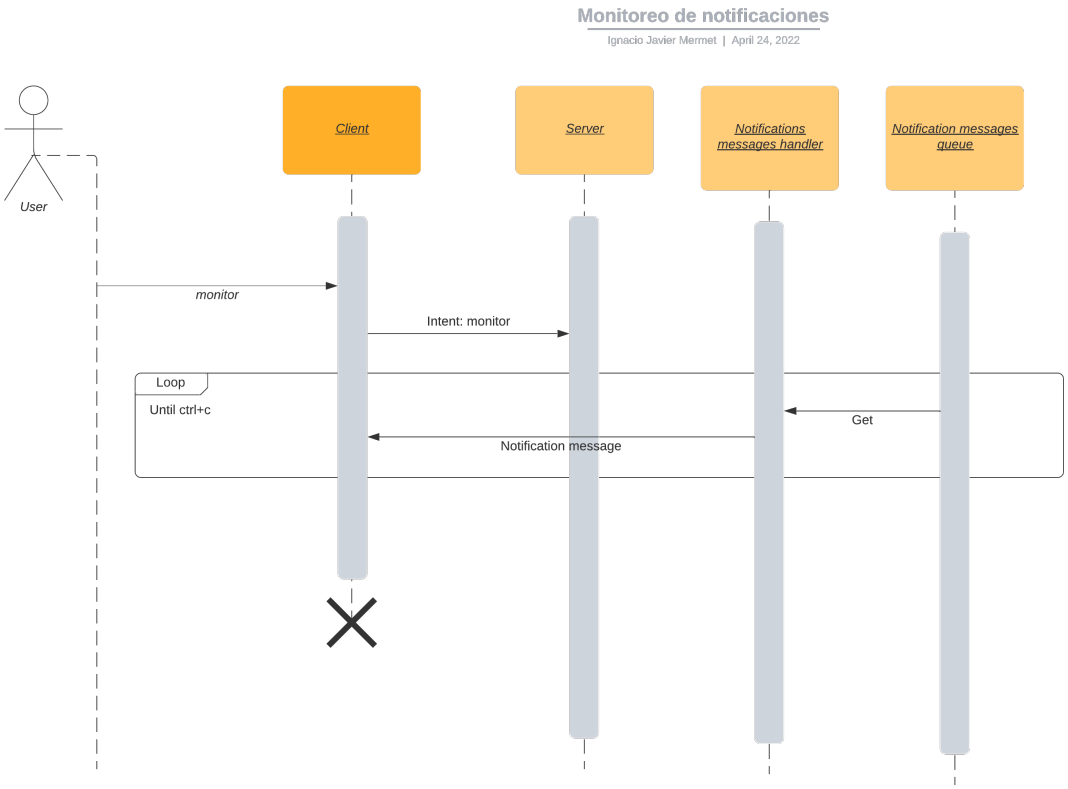

Figura 9: Diagrama de secuencia - monitoreo de notificaciones

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. Originalmente el CSV se escribía con los campos `timestamp`, `metric_id`, `value`. Podemos pensar varios escenarios:

- El caso feliz: Tenemos todos los campos y el `n` que termina la línea

- Un caso poco feliz, pero poco problemático: tenemos una coma, pero no el valor que sigue
- Un caso sutilmente problemático: que el campo `value` esté truncado!

El segundo caso se puede resolver sencillamente ignorando las lineas que no tengan todos los campos. El tercer caso no podemos resolverlo con el esquema de datos planteado. Por tanto se agrega una cuarta columna `check`, que es una función de `timestamp`. Para poder escribir rápidamente las líneas, necesitamos que sea una función rápida, un hash quizás sea demasiado lento. Por tanto se prefirió usar la función mód 10. Luego al momento de leer, descartamos las lineas donde timestamp mód $10 \neq$ check.

Por otro lado, se ignoran todas las lineas 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.

[2] *Intention package*. URL: https://github.com/CrossNox/7574-TP1/blob/master/metrics_server/protocol.py#L35.

[3] *poetry*. URL: https://python-poetry.org/.

[4] *python*. URL: https://www.python.org.