

Proyecto 1, Entrega 4 - Arquitectura, conclusiones y consideraciones

Integrantes:

- Daniel Santamaría Álvarez - 201720222
- Nestor Gonzalez – 201912670
- Álvaro Plata – 201820098
- Rafael Humberto Rodriguez Rodriguez – 202214371

Los primeros cambios que se realizaron en la aplicación fueron para cambiar el manejo de las tareas asíncronas de la aplicación de Celery con redis a un esquema de publicador suscriptor con Cloud Pub/Sub.

En el servicio web primero se importa el SDK google cloud que contiene el módulo de Pub/Sub.

```
from google.cloud import pubsub_v1
```

Ahora, para configurar el servicio se debe tener el id del proyecto y el tópico al que se publicarán los mensajes, que en este caso es "tasks". Luego se crea el cliente publicador con el destino de los mensajes.

```
topic_id="tasks"
project_id=os.environ.get("PROJECT_ID")
publisher=pubsub_v1.PublisherClient()
topic_path=publisher.topic_path(project_id,topic_id)
```

Para enviar un mensaje, por ejemplo en el endpoint de post tasks, los argumentos que se pasaban a celery se estructuran en un diccionario y se codifica su representación en string para poder ser enviado.

```
args={"task_id":task.id,"in_route":in_route,"out_route":out_route,"in_ext":in_ext,"out_ext":out_ext}
data=str(args).encode("utf-8")
```

Una vez está listo el mensaje de envío, se publica al tópico.

```
future=publisher.publish(topic_path,data)
```

Desde el worker, también se debe configurar el tópico al que se va a suscribir.

```
project_id = "swnube"
subscription_id = "tasks-sub"
```

```
subscriber = pubsub_v1.SubscriberClient()
subscription_path = subscriber.subscription_path(project_id, subscription_id)
```

Por otro lado, se definió que se usaría un esquema de pull de los mensajes con exactly once delivery, para garantizar que solo 1 worker procese una tarea y que no se repita trabajo. En el stream de pull también se define una función de callback que se ejecutará cada vez que llegue un mensaje.

```
streaming_pull_future = subscriber.subscribe(subscription_path, callback=callback)
print(f"Listening for messages on {subscription_path}...\n")
```

La función de callback procesa el mensaje traído de la suscripción, lo decodifica y se usa la función eval que convierte el string a un diccionario. Después de esto, se pasan los parámetros a la misma función definida en entregas pasadas para hacer la conversión del archivo. Finalmente, el worker le notifica al servicio de Pub/Sub que recibió el mensaje exitosamente y sigue esperando a que llegue un mensaje nuevo.

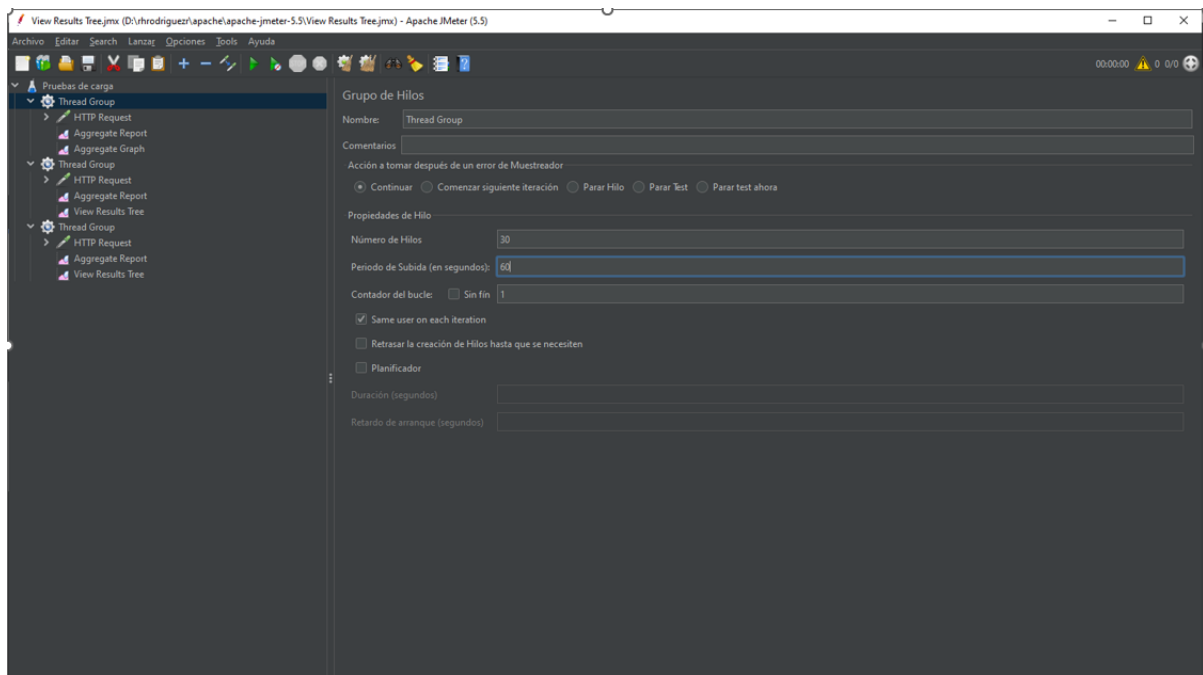
```
def callback(message: pubsub_v1.subscriber.message.Message) -> None:
    print(f"Received {message}.")
    message_dict = eval(message.data.decode("utf-8"))
    print(message_dict)
    print(type(message_dict))
    start_conversion(message_dict["task_id"], message_dict["in_route"], message_dict["out_route"], message_dict["in_ext"], message_dict["out_ext"])

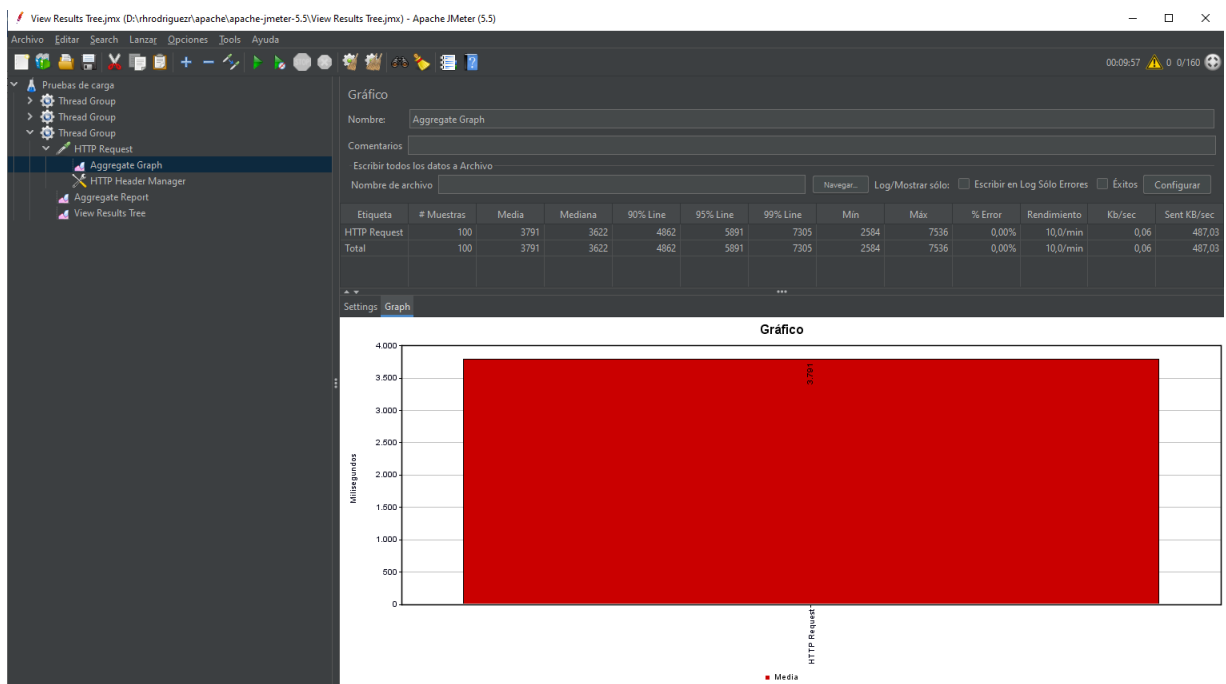
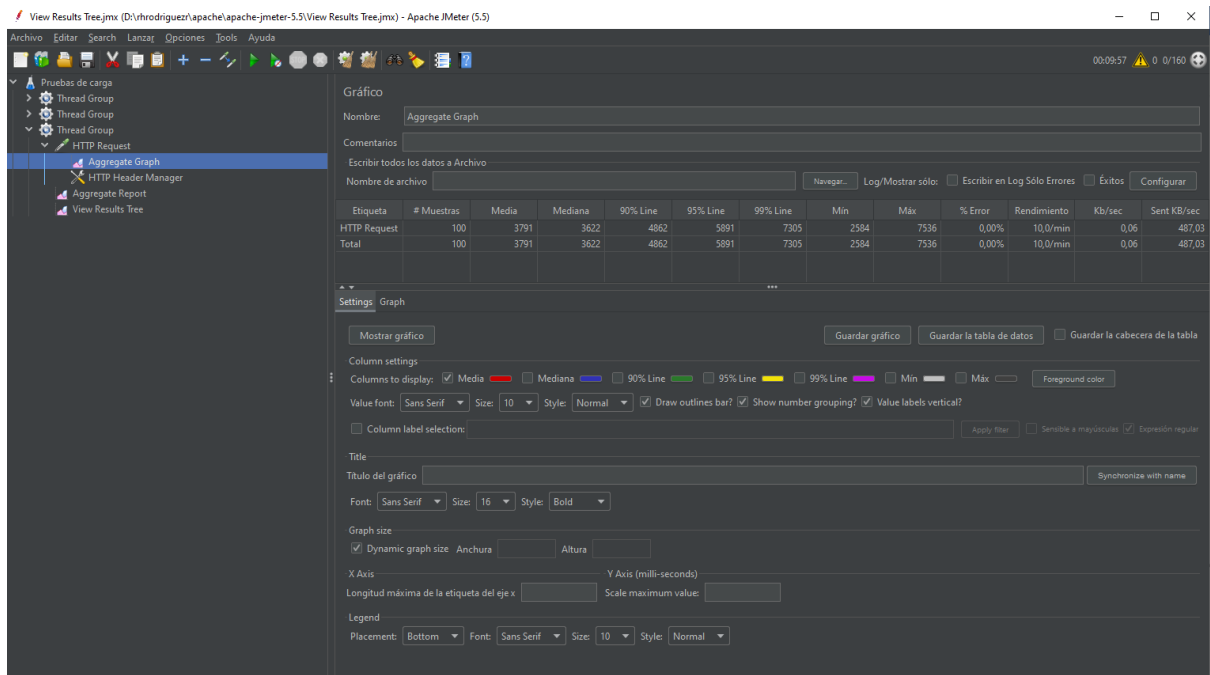
    # Use `ack_with_response()` instead of `ack()` to get a future that tracks
    # the result of the acknowledge call. When exactly-once delivery is enabled
    # on the subscription, the message is guaranteed to not be delivered again
    # if the ack future succeeds.
    ack_future = message.ack_with_response()
```

Análisis de Capacidad.

Escenario 1

Conversión de archivos





Respecto a la anterior entrega el porcentaje bajo considerablemente, el promedio de ejecución subió y se mantiene el rendimiento de conversión de 10 archivos por minuto.