Distribuidos I TP0

Mermet, Ignacio Javier `98153`

Abril 2022

1 Sobre la entrega

El código de la entrega se puede encontrar en GitHub. Cada ejercicio tiene un Tag asociado.

2 Ejercicio 1

2.1 Enunciado

Modificar la definición del docker-compose para agregar un nuevo cliente al proyecto.

2.2 Resolución propuesta

Como ver los cambios: `git diff v0..v1`.

Para este ejercicio, y en particular en vistas al ejercicio `1.1`, la forma más sencilla de resolver el enunciado es agregar una nueva sección en el archivo `docker-compose.yaml`. Esta sección es casi una copia de la sección `client1`.

3 Ejercicio 1.1

3.1 Enunciado

Definir un script (en el lenguaje deseado) que permita crear una definición de docker-compose con cantidad de clientes N.

3.2 Resolución propuesta

Como ver los cambios: `git diff v1..v1.1`.

Utilizando `docker-compose`, no es necesario agregar cada cliente como una sección al archivo `yaml` de configuración. El flag `scale` del comando `up`[2] provee la funcionalidad pedida. Utilizarlo requiere dos cambios:

1. Agregar una nueva directiva al archivo `Makefile` de modo de poder escalar la cantidad de contenedores de clientes.
2. No hardcodear la variable de entorno `CLI_ID` en cada container de cliente.

El segundo punto se resuelve modificando el entrypoint para ejecutar un script de bash que obtenga el indice de contenedor de cliente.[4] Este script obtiene la IP del contenedor y luego obtiene el indice desde el PTR entry asociado a la IP.

3.2.1 Como ejecutar la nueva directiva de make

`make docker-compose-up-scale-client` escala a un valor por defecto de 2 clientes. Si se quiere modificar esta cantidad por un valor `N`, se debe ejecutar como `make docker-compose-up-scale-client NCLIENTS=N`. [1]

4 Ejercicio 2

4.1 Enunciado

Modificar el cliente y el servidor para lograr que realizar cambios en el archivo de configuración no requiera un nuevo build de las imágenes de Docker para que los mismos sean efectivos. La configuración a través del archivo debe ser injectada al ejemplo y persistida afuera del mismo. (Hint: docker volumes)

4.2 Resolución propuesta

Como ver los cambios: `git diff v1.1..v2`.

En este punto, la resolución es sencillamente definir volumenes[10] para los archivos de configuración tanto del servidor como del cliente. En el caso del cliente, se debe eliminar el comando `COPY` que copia el archivo `config.yaml` a la imagen. En el caso del servidor, con agregar la ruta en un archivo `.dockerignore`[3], es suficiente para que lo ignore en el `COPY` de la carpeta a la imagen.

5 Ejercicio 3

5.1 Enunciado

Crear un script que permita testear el correcto funcionamiento del servidor utilizando el comando netcat. Dado que el servidor es un EchoServer, se debe enviar un mensaje el servidor y esperar recibir el mismo mensaje enviado.

Netcat no debe ser instalado en la maquina y no se puede exponer puertos del servidor para realizar la comunicación. (Hint: docker network)

5.2 Resolución propuesta

Como ver los cambios: `git diff v2..v3`.

En este caso, opté por escribir un script de bash que ejecuta `netcat` dentro de un container de la imagen `busybox`. Para ver el modo de uso, correr `sh test_server.sh help`.

6 Ejercicio 4

6.1 Enunciado

Modificar cliente o servidor (no es necesario modificar ambos) para que el programa termine de forma gracefully al recibir la signal SIGTERM. Terminar la aplicación de forma gracefully implica que todos los sockets y threads/procesos de la aplicación deben cerrarse/joinearse antes que el thread de la aplicación principal muera. Loguear mensajes en el cierre de cada recurso.(Hint: Verificar que hace el flag -t utilizado en comando docker-compose-down)

6.2 Resolución propuesta

Como ver los cambios: `git diff v3..v4`.

Para este ejercicio, opté por modificar el servidor. Quiero hacer foco en dos partes distintas del enunciado para explicar la resolución.

6.2.1 Terminación por SIGTERM

Para este punto, en el archivo principal del servidor configuré como handler la señal `SIGTERM` a un handler del servidor instanciado[7]. Este handler configura un flag del servidor que lo hace salir de su loop de ejecución.

6.2.2 Liberación de recursos

Al salir del programa, finaliza el programa. En este momento, la referencia del servidor instanciado se pierde y se llama al *dunder* `__del__`[6], donde se cierra el socket del servidor. La guía de programación con sockets de Python[8] indica que si bien cuando el recolector de basura recolecta un socket, los cierra automaticamente si fuera necesario. Sin embargo, indica, es un muy mal hábito. Por tanto, también se cierran los sockets creados al aceptar las conexiones, dentro del `handler` que maneja ese cliente.[1]

7 Ejercicio 5

7.1 Enunciado

Modificar el servidor actual para que el mismo permita procesar mensajes y aceptar nuevas conexiones en paralelo.

El alumno puede elegir el lenguaje en el cual desarrollar el nuevo código del servidor. Si el alumno desea realizar las modificaciones en Python, tener en cuenta las limitaciones del lenguaje.

7.2 Resolución propuesta

Como ver los cambios: `git diff v4..v5`. Como preámbulo a la resolución propuesta, quiero hacer foco en la última oración del enunciado. La *rule of thumb* en *Python* es que se utiliza `multiprocessing` cuando se trata de procesamiento CPU-bounded y `threading` cuando se trata de procesamiento IO-bounded. En el escenario del presente trabajo práctico, no nos encontramos con un caso del llamado *Convoy Effect*[9], por tanto utilizar `threading.Thread` es totalmente aceptable[5], dado que el GIL no es un problema.

Dicho eso, la implementación es relativamente sencilla. Si el cliente mandara más de un mensaje, deberíamos considerar la posibilidad de tener que frenar forzadamente los `handlers` de las conexiones en un escenario como el del punto 4. Por tanto, creé una clase `StoppableThread` que hereda de `threading.Thread` que puede ser frenada forzadamente.[2]. La lógica para manejar las nuevas conexiones queda encapsulada en una clase `ConnectionHandler` que hereda de la anterior.[3].

[1] No consideré que esto no es necesario considerando que este socket vive dentro del scope de un context manager.

[2] Esto me ayudó a validar algunos puntos sobre lo dicho anteriormente sobre el GIL.

[3] Noté luego de haber creado el tag que hay un `close` del socket en el método `__del__` que es redundante.

Referencias

[1] *9.5 Overriding Variables*. URL: https://www.gnu.org/software/make/manual/make.html#Overriding.

[2] *docker-compose-up*. URL: https://docs.docker.com/compose/reference/up/.

[3] *Dockerfile reference*. URL: https://docs.docker.com/engine/reference/builder/#dockerignore-file.

[4] *How to simply scale a docker-compose service and pass the index and count to each?* URL: https://stackoverflow.com/a/64799824.

[5] *Python behind the scenes #13: the GIL and its effects on Python multithreading*. URL: https://tenthousandmeters.com/blog/python-behind-the-scenes-13-the-gil-and-its-effects-on-python-multithreading/.

[6] *Python Data model*. URL: https://docs.python.org/3/reference/datamodel.html#object.__del__.

[7] *signal — Set handlers for asynchronous events*. URL: https://docs.python.org/3/library/signal.html#signal.signal.

[8] *Socket Programming HOWTO*. URL: https://docs.python.org/3/howto/sockets.html#disconnecting.

[9] *Understanding the Python GIL*. URL: https://youtu.be/Obt-vMVdM8s?t=1933.

[10] *Use volumes*. URL: https://docs.docker.com/storage/volumes/.