**1 SESIÓN 1. PRIMEROS PASOS CON DOCKER**

* 1. **Despliegue de las imágenes individuales de client/broker/worker**
* El código del cliente debe emitir 10 solicitudes1 antes de finalizar.

var x = 0

...

Req.on(‘message’,…){

x++

if(x==10)process.exit(0)

...

}

…

for(let i = 0; i < 10; i++)

req.send...

* A continuación debes generar (¿ya sabes cómo?) 3 imágenes (imclient, imbroker, imworker), abrir 5 ventanas y ejecutar estas instrucciones para comprobar que todo encaja:

-Desde la carpeta client:

docker build -t tsr2021/1/imclient .

-Desde la carpeta broker:

docker build -t tsr2021/1/imbroker .

-Desde la carpeta worker:

docker build -t tsr2021/1/imworker .

* Ventana 1: Averigua y anota la IP que recibe el contenedor que ejecuta imbroker, y modifica adecuadamente los Dockerfiles de los otros.

Docker ps -> Encuentras la idContenedor

docker inspect idContenedor | grep IPAddress | cut -d '"' -f 4

docker inspect broker | grep IPAddress | cut -d '"' -f 4 -> Si no hay más contenedores broker está bien pero hay varios condicionantes

* 1. **Desplegando cbw de la práctica 2**
* Más información en el anexo 2\_CBW. ¿Puedes prever algún problema por el uso que se hace de identity?

Si la propiedad identity de los sockets se genera aleatoriamente no habrá ningún problema, sin embargo, si se tiene el caso que se generan a través de una constante el primer cliente/worker que se conecte al broker lo hará sin fallos pero el resto no se podrán conectar ya que poseen la misma identidad (podemos verlo en el caso de 2worker.js)

No se sabe que pasa cuando se conectan dos con la misma identidad pero básicamente está mal por eso.

**1.4 Nueva base para alta disponibilidad y clases de trabajo**

* Dispones del código y otras informaciones en el anexo correspondiente. Debes desplegarlo y escalar a 3 clientes y 2 trabajadores de cada clase.

sudo docker-compose up --scale cliA=3 --scale cliB=3 --scale cliC=3 --scale worA=2 --scale worB=2 --scale worC=2

* Antes de iniciar el despliegue debes observar que las imágenes a utilizar en este apartado tienen los mismos nombres que las utilizadas en el apartado anterior, pero sus programas son diferentes. Debido a esto, convendrá eliminar previamente las generadas en 2\_CBW. Revisa qué órdenes deberás utilizar para ello.

Eliminar los contenedores primero con:

docker rm $( docker ps -a -q )

Luego eliminar las imagenes

docker images

docker rmi id\_imagen

**1.4.1 Pregunta**

* Reflexiona sobre la posibilidad de idear alguna alternativa en la que no se necesite incluir los tipos en docker-compose.yml, pero permitiendo controlar el número de instancias desplegadas para cada tipo.

Una alternativa sería que se creasen workers de cada tipo y los clientes al conectarse le digan al broker su tipo, de esta manera el broker gestiona las conecciones con los workers. El inconveniente es que habria que crear workers dinstintos para cada clase.

En el Docker compose.yml debe aparecer cada tipo de componente. No hay alternativa. Lo que se ha propuesto sigue manteniendo los tipos en el Docker-compose

**2.4 Acceso a almacenamiento duradero desde logger**

* docker run –v /tmp/logger.log:/tmp/cbwlog parámetros

directorio de anfitrión : directorio de contenedor

**2.5 Despliegue conjunto del nuevo servicio CBW\_FTCL**

* Prueba a realizar: el despliegue básico requiere un archivo de anotaciones vacío, y la ejecución de una combinación compuesta por 4 clientes de tipo B, 2 trabajadores de tipo B, 1 broker y 1 logger.

sudo docker-compose up --scale cliA=0 --scale cliB=4 --scale cliC=0 --scale worA=0 --scale worB=2 --scale worC=0 --scale bro=1 --scale log=1

No lo ha hecho nunca pero en principio debería funcionar. Su alternativa es evitar todos los tipos excepto el B en el .yml

* Cuestión: reflexiona, sin necesidad de ejecutar, qué ocurriría si intentáramos desplegarlo en los siguientes escenarios:

• 2 clientes B, 1 trabajador B, 2 brokers, 1 logger

No es del todo correcto, ya que el código proporcionado solo es capaz de funcionar con un broker, ya que los clientes y workers solo realizan un connet -> Correcto

Si se conectase a los dos brokers, harían un Round Robin. Pero es “kafkiano” XD

Nada una fumada…

• 2 clientes B, 1 trabajador B, 1 broker, 2 loggers

No es del todo correcto, ya que el código proporcionado solo es capaz de funcionar con un loggerr, ya que el broker solo realiza un connet al logger y no hay ningún otro componente que requiera otro logger

**4 SESIÓN 3. ENCADENANDO VARIOS SISTEMAS CBW: EL WORCLI**

* Para su ejecución requerirá 4 parámetros: el URL de bk1, el URL de bk2, el retardo de red que añade (argumento delay) y el tipo de trabajo que debe procesar.

Para ello añadir en el docker-compose.yml de A lo siguiente:

-ports:

"9999:9999"

y en el de B lo siguiente:

-ports:

"9998:9998"

Correcto 😊 pero hay que poner donde

En la maquina B el port estará asociado al frontend del bróker desplegado en B.

En la maquina A el port estará asociado al backend del bróker (+ cómodo). O q conteste el bróker dos al worcli, lo asociaríamos a worcli.

* Su código aparece en el anexo correspondiente, y solo requiere destacar que requiere dos sistemas cbw\_fctl (posiblemente en dos hosts diferentes) para funcionar.

node worcli\_2021 tcp://localhost:9999 tcp://localhost:9998 0 D

* + 1. **Pregunta 1**
* Si el *worcli* se ejecuta en un anfitrión, ¿puede formar parte de un despliegue del servicio (mediante un docker-compose.yml)?

Te está pidiendo que en caso de que si ambos estuviesen desplegados en la misma maquina cada uno con su Docker-compose.yml se pudiese realizar el despliegue automáticamente.

Podría formar parte del despliegue de servicio siempre que se conozca la dirección ip y puerto del broker del otro anfitrión. Pero no lo puedes conocer sin ponerlo en marcha…

La respuesta es que no ya que no hay forma de que el resultado de desplegar el primero sea conocido por el segundo automáticamente.

* Como alternativa, ¿podrías ponerlo en funcionamiento “a mano”? Argumenta la respuesta a ambas preguntas. Si la segunda es afirmativa, explica cómo lo harías.

Si, conociendo las ips y puertos de los anfitriones en los que se conectan los brokers, se puede lanzar por separado con nodejs y las urls (debe conectarse via los anfitriones y no los brokers).

Haciendo el despliegue del B, mediante ps e inspect averiguamos la IP del A, editamos el dockerfile de worcli con ese valor y hacemos el depliegue de A

**4.1.2 Pregunta 2**

* Argumenta por qué hay grandes diferencias entre las 3 invocaciones siguientes, y qué repercusiones tendrá cuando el broker alojado en el primer anfitrión (192.168.105.111) reciba alguna petición de un cliente de tipo D

Puede ser un problema, ya que si el retardo se aumenta, puede que el worcli sea considerado como que ha fallado por el broker alojado en el afitrión 192.168.105.111:

- 1200 puede ocurrir que el tiempo que tardan en llegar los mensajes de un anfitrión a otro sea alto por lo que será considerado como un fallo.

- 2200 si o si será considerado como fallo.

Nota: Cuando vence un Timeout se ejecuta la acción asociada a el, que es pasarle el trabajo a otro trabajador y le pasará lo mismo. Por lo tanto acabaremos dando de baja a todos los trabajadores de ese tipo y pasará a la cola de mensajes pendientes. De esta forma el cliente nunca recibirá respuesta. A

OJO a lo de arriba que si que estaba mal |