Resumen de lo hecho en la practica de TSR:

1_A_MANO

Hacer que el cliente envie 10 mensajes a los workers.

Poner en el dockerfile de cliente y worker que se conecte al broker. Para ello despues de arrancar el broker se le mira la ip con **docker inspect idCont** y se modificara el fichero del worker y client con la ip del broker. 192,168,1,1

```
CMD node myclient tcp://192.168.1.1:9998
```

```
CMD node myworker tcp://192.168.1.1:9999
```

Despues de poner la ip+broker se generan las imagenes con: **docker build -t *nomImatge pathDockerfile***

Finalmente arrancar distintas imagenes en distintas pestañas de consola. **Docker run ibroker, docker run iworker, docker run iclient.**

2_CBW

Aqui se usa un docker-compose.yml para que arranque todo automaticamente utilizando dependencias entre los componentes.

En cliente y worker se usa una **variable de entorno**: $BROKER_URL

```
CMD node myclient $BROKER_URL
```

En el docker compose se pone en el apartado de *environment:*

```
services:
  cli:
    image: client
    build: ./client/
    links:
      - bro
    environment:
      - BROKER_URL=tcp://bro:9998
  wor:
    image: worker
    build: ./worker/
    links:
      - bro
    environment:
      - BROKER_URL=tcp://bro:9999
  bro:
```

Cuando esta todo listo se arranca con: **docker-compose up –scale cli=2 –scale wor=2**

3_CBW_FTC

Aqui tenemos varias clases de clientes y de workers. Del tipo A,B y C , para ello en los Dockerfile de cliente y worker se pone una variable de entorno $**CLASSID**

```
CMD node myclient $BROKER_URL $CLASSID
CMD node myworker $BROKER_URL $CLASSID
```

En el **docker-compose.yml** se declaran las ditintas clases A,B,C como SERVICIOS distintos, uno por clase. Tienen la varible de entorno del broker como apartados anteriores y a parte la classID:

```
cliA:
  image: client
  build: ./client/
  links:
    - bro
  environment:
    - BROKER_URL=tcp://bro:9998
    - CLASSID=A
cliB:
  image: client
  build: ./client/
  links:
    - bro
  environment:
    - BROKER_URL=tcp://bro:9998
    - CLASSID=B
cliC:
  image: client
  build: ./client/
  links:
    - bro
  environment:
    - BROKER_URL=tcp://bro:9998
    - CLASSID=C
```

```
worA:
  image: worker
  build: ./worker/
  links:
    - bro
  environment:
    - BROKER_URL=tcp://bro:9999
    - CLASSID=A
worB:
  image: worker
  build: ./worker/
  links:
    - bro
  environment:
    - BROKER_URL=tcp://bro:9999
    - CLASSID=B
worC:
  image: worker
  build: ./worker/
  links:
    - bro
  environment:
    - BROKER_URL=tcp://bro:9999
    - CLASSID=C
```

Arrancar el servicio con: **docker-compose up --scale cliA=3 --scale cliB=3 --scale cliC=3 --scale worA=2 --scale worB=2 --scale worC=2**

Habria que matar un worker y ver que otro worker atiende la peticion, se puede simular con un numero random según salvador. Si toca ese numero random, que mate el proceso worker como si hubiera fallado.

4_CBW_FTCL

En este apartado de la practica introducimos un nuevo componente que es el **logger**.

Aqui el broker recibe como parametro la url del logger como variable de entorno.

```
CMD node mybroker 9998 9999 $LOGGER_URL
```

El broker tiene un socket push en el que envia al logger los logs en su funcion ‘annotate’.

En el docker-compose.yml tenemos en el servicio de broker que se da valor a la variable de entorno $LOGGER_URL.

El servicio log, expone el puerto y ademas asigna la carpeta del anfitrion a una del contenedor: **carpetaAnfitrio:carpetaContenedor**

Dicha carpeta interna se declara como VOLUME en el dockerfile del logger.

```
  bro:
    image: broker
    build: ./broker/
    links:
      - log
    expose:
      - "9998"
      - "9999"
    environment:
      - LOGGER_URL=tcp://log:9995
  log:
    image: logger
    build: ./logger/
    expose:
      - "9995"
    volumes:
# /tmp/logger.log DIRECTORY must exist on host and writeable
      - /tmp/logger.log:/tmp/cbwlog
    environment:
      - LOGGER_DIR=/tmp/cbwlog
```

DockerCompose.yml

```
1   FROM tsr2021/ubuntu-zmq
2   COPY ./logger.js /mylogger.js
3   VOLUME /tmp/cbwlog
4   EXPOSE 9995
5   CMD node mylogger 9995 $LOGGER_DIR/logs
6
```

Dockerfile del logger

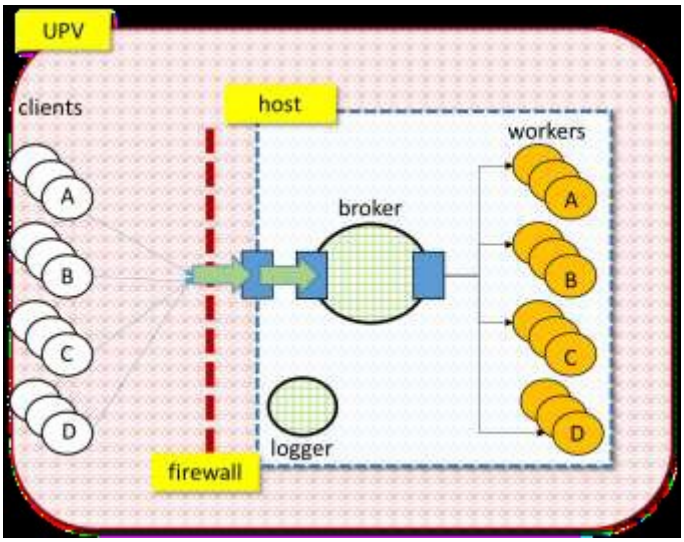
Hay que crear la carpeta: **/tmp/logger.log** antes de arrancar el servicio en la maquina anfitrion.

Arrancamos el servicio con: docker-compose up --scale cliB=4 --scale worB=2

Si hacemos un **cat /tmp/logger.log/logs** en la maquina anfitriona, veremos todos los logs del contenedor.

Con **docker-compose down** estando en la carpeta del docker-compose.yml, se para el servicio.

5_CBW_FTCL_CLEXT



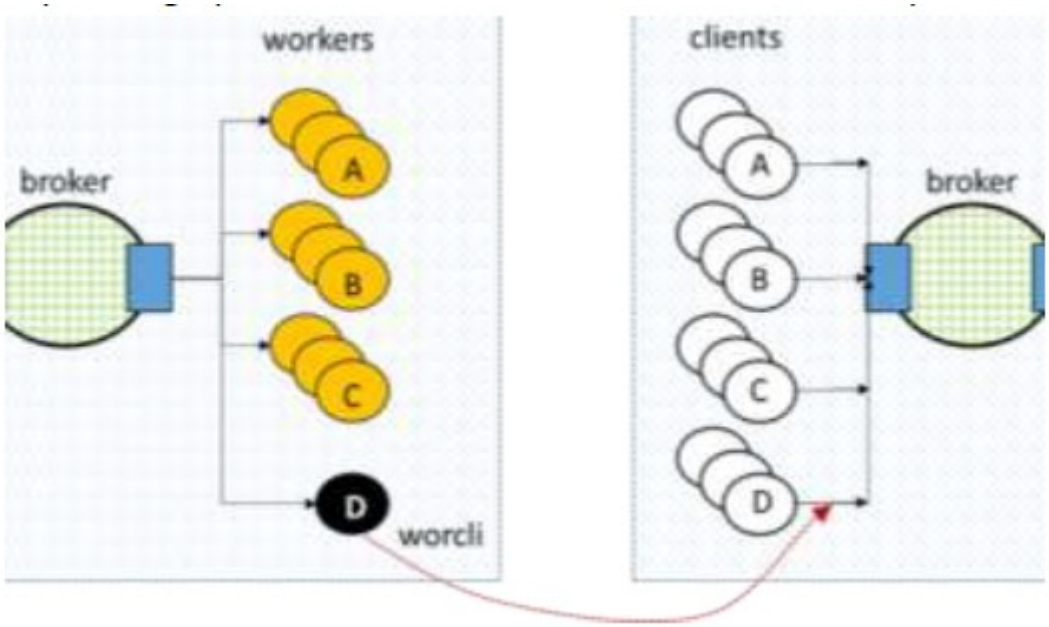
Aqui la idea es tener clientes externos a todo el docker. Para ello hay que asignar un puerto de la maquina anfitriona al puerto del broker para que haga de puente.

En el servicio de broker se asigna el puente:
`ports:`
`- "9500:9998"`

Esto asigna el puerto 9500 del anfitrion, al puerto del broker de docker 9998

Si se ejecuta **node client.js tcp://127.0.0.1:9500 A** se esta actuando como cliente externo porque nos conectamos a la ip local des de consola. Si se mira **cat /tmp/logger.log/logs** aparecera el log con la conexión.

6_CBW_FTCL_WORCLIEXT



En este caso tenemos dos brokers: Y nodo ‘worker’ de un broker1 actua de worker para ese broker1 pero a su vez es cliente de un broker2

Ese worcli, se llama con:
node worcli_2021.js urlBroker1 urlBroker2 delay tipoTrabajoprocesar
node worcli_2021.js tcp://172.23.105.111:9999 tcp://172.23.105.112:9998 200 D