TSR – 19 de enero de 2017. EJERCICIO 6

Dados estos tres programas:

```
// Client.js
const zmq = require('zmq');
const req = zmq.socket('req');

var args = process.argv.slice(2);

if (args.length < 3) {
  console.error('Three arguments are needed:');
  console.error('  - IP address of the broker.');
  console.error('  - Pattern to be looked for.');
  console.error('  - String to be processed.');
  process.exit(1);
}

req.connect('tcp://'+args[0]+':8000');

req.send([args[1],args[2]]);

req.on('message', function(m) {
  console.log('The string "%s" has been '+
    'found %d times.', args[1], m+");
  process.exit(0);
});
```

```
// Broker.js
const zmq = require('zmq');
const rou = zmq.socket('router');
const dea = zmq.socket('dealer');

try {
  rou.bindSync('tcp://*:8000');
  dea.bindSync('tcp://*:8001');
} catch (e) {
  if (e) {
    console.error('Error "%s" binding '+
      'the broker sockets.', e);
    console.error('Exiting...');
    process.exit(1);
  }
}

rou.on('message', function() {
  var segs = Array.apply(null, arguments);
  dea.send(segs);
});

dea.on('message', function() {
  var segs = Array.apply(null, arguments);
  rou.send(segs);
});
```

```
// Worker.js
const zmq = require('zmq');
const rep = zmq.socket('rep');

var args = process.argv.slice(2);

if (args.length < 1) {
  console.error('One argument is needed:');
  console.error('  - IP address of the broker.');
  process.exit(1);
}

rep.connect('tcp://'+args[0]+':8001');

rep.on('message', function(pattern,str) {
  var str2 = str+";
  var count = 0;
  for (var i=0; i<str2.length; i++) {
    if (pattern ==
    str2.substr(i,pattern.length))
      count++;
  }
  rep.send(count);
});
```

El programa "Worker.js" implanta un servicio que cuenta cuántas veces una cadena corta ("pattern") está contenida en otra más larga. Para ello, el programa "Client.js" envía ambas cadenas a "Broker.js". Cada "worker" procesa cada petición independientemente de los demás.
Responde las cuestiones siguientes relativas al despliegue de estos programas. Para este fin, asume que la máquina anfitriona tiene una imagen Docker basada en "fedora:latest" con las órdenes **node** y **npm**, la biblioteca ZeroMQ y el módulo NodeJS **zmq** instalados correctamente. El nombre de esa imagen es "**zmq-devel**":

1. Propón una estructura de directorios **y sus contenidos** para mantener estos tres ficheros y los Dockerfile necesarios para crear las tres imágenes Docker, una por programa. (1 punto)

Aunque pueden admitirse múltiples alternativas, las soluciones sugeridas en el Seminario 4 para este tipo de cuestión asumían que los ficheros fuentes de cada componente eran ubicados en una carpeta diferente. Por tanto, una posible estructura sería:

- Directorio antecesor común para todos los componentes.
  - Directorio "Cliente":
    - Fichero "Client.js".
    - "Dockerfile" (para el componente cliente).
  - Directorio "Broker":
    - Fichero "Broker.js".
    - "Dockerfile" (para el componente broker).
  - Directorio "Worker":
    - Fichero "Worker.js".
    - "Dockerfile" (para el componente trabajador).

2. Escribe el fichero Dockerfile para generar la imagen **broker**. (3 puntos)

```
FROM zmq-devel
RUN mkdir /d1
COPY ./Broker.js /d1/Broker.js
EXPOSE 8000 8001
WORKDIR /d1
CMD node Broker.js
```

Obsérvese que habrá múltiples variantes de este Dockerfile que también funcionarán correctamente. Por ejemplo, la orden WORKDIR no es necesaria si la orden CMD especifica el nombre de ruta completo del programa JavaScript que vayamos a ejecutar.

Los elementos obligatorios serán:

- Incluir una orden FROM en la primera línea, con “zmq-devel” como nombre de la imagen a utilizar.
- Un COPY o ADD del fichero “Broker.js” desde el anfitrión a algún directorio del contenedor.
- Incluir una orden EXPOSE para indicar que los puertos 8000 y 8001 son los que van a utilizarse.
- Incluir una orden CMD o ENTRYPOINT para ejecutar el programa Broker.js con el intérprete “node”.

3. Escribe la orden Docker para generar la imagen del bróker. El nombre de esa imagen debe ser “**broker**”. (1 punto)

Esta orden debe lanzarse en el directorio “Broker”:
docker build –t broker .

4. Escribe la línea de órdenes necesaria para lanzar un contenedor **broker** y describe cómo podrás obtener su dirección IP. (2 puntos)

La orden a utilizar es:
docker run broker
y puede lanzarse desde cualquier directorio una vez el “docker build” solicitado en la cuestión anterior haya sido utilizado.
Mientras el contenedor **broker** esté funcionando, utilizaremos **docker ps** para averiguar su ID o nombre. Una vez conozcamos ese identificador (asumamos que su prefijo es “b875…”), podremos averiguar su dirección IP con:
docker inspect b875 | grep IPAddress
…o simplemente utilizando **docker inspect b875** y buscando alguna línea con una dirección IP entre la salida proporcionada.

5. Asumamos que la dirección IP obtenida en la cuestión anterior es 172.17.0.2. Con esta información, genera la imagen **worker**. Escribe el Dockerfile necesario. (2 puntos).

```
FROM zmq-devel
RUN mkdir /d1
COPY ./Worker.js /d1/Worker.js
WORKDIR /d1
CMD node Worker.js 172.17.0.2
```

Esa sería una solución básica para esta cuestión. Otra solución más realista utilizaría una variable de entorno para recibir la dirección IP del broker. Por ejemplo, utilizando la siguiente como última línea del Dockerfile:
CMD node Worker.js $BROKER_IP
En ese caso, el valor de esa variable de entorno debería pasarse en la orden **docker run** a utilizar para ejecutar una instancia de ese componente en un contenedor. Algo así:
docker run –e BROKER_IP=172.17.0.2 worker
…asumiendo que ya se ha creado una imagen **worker** como se solicita en la cuestión 6.

6. Escribe la orden Docker para generar la imagen del trabajador. El nombre de esa imagen debe ser “**worker**”. (1 punto)

Esta orden debería lanzarse en la carpeta “Worker”:
docker build –t worker .