TSR – 19 de gener de 2017. EXERCICI 6

Donats aquests tres programes:

```
// 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 (i) {
  if (i) {
    console.error('Error "%s" binding '+
      'the broker sockets.', i);
    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 servei que compta quantes vegades una cadena curta (“pattern”) està continguda en una altra més llarga. Per a fer això, el programa “Client.js” envia ambdues cadenes a “Broker.js”. Cada “worker” processa cada petició independentment dels altres.

Respon les qüestions següents relatives al desplegament d'aquests programes. Per a aquesta fi, assumeix que la màquina amfitriona té una imatge Docker basada en “fedora:latest” amb les ordres **node** i **npm**, la biblioteca ZeroMQ i el mòdul NodeJS **zmq** instal·lats correctament. El nom de la imatge és “**zmq-devel**”:

1. Proposa una estructura de directoris **i els seus continguts** per a mantenir aquests tres fitxers i els Dockerfile necessaris per a crear les tres imatges Docker, una per programa. (1 punt)

Encara que poden admetre's múltiples alternatives, les solucions suggerides en el Seminari 4 per a aquest tipus de qüestió assumien que els fitxers fonts de cada component eren situats en una carpeta diferent. Per tant, una possible estructura seria:

- Directori antecessor comú per a tots els components.
  - Directori “Client”:
    - Fitxer “Client.js”.
    - “Dockerfile” (per al component client).
  - Directori “Broker”:
    - Fitxer “Broker.js”.
    - “Dockerfile” (per al component broker).
  - Directori “Worker”:
    - Fitxer “Worker.js”.
    - “Dockerfile” (per al component treballador).

2. Escriu el fitxer Dockerfile per a generar la imatge **broker**. (3 punts)

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

Observe's que hi haurà múltiples variants d'aquest Dockerfile que també funcionaran

correctament. Per exemple, l'ordre WORKDIR no és necessària si l'ordre CMD especifica el nom de ruta complet del programa JavaScript que anem a executar.
Els elements obligatoris seran:

- Incloure una ordre FROM en la primera línia, amb "zmq-devel" com a nom de la imatge a utilitzar.
- Un COPY o ADD del fitxer "Broker.js" des de l'amfitrió a algun directori del contenidor.
- Incloure una ordre EXPOSE per a indicar que els ports 8000 i 8001 són els que van a utilitzar-se.
- Incloure una ordre CMD o ENTRYPOINT per a executar el programa Broker.js amb l'intèrpret "node".

3. Escriu l'ordre Docker per a generar la imatge del bróker. El nom de la imatge ha de ser "**broker**". (1 punt)

Aquesta ordre ha de llançar-se en el directori "Broker":
docker build –t broker .

4. Escriu la línia d'ordres necessària per a llançar un contenidor **broker** i descriu com podràs obtenir la seua adreça IP. (2 punts)

L'ordre a utilitzar és:
docker run broker
i pot llançar-se des de qualsevol directori una vegada el "docker build" sol·licitat en la qüestió anterior haja sigut utilitzat.
Mentre el contenidor **broker** estiga funcionant, utilitzarem **docker ps** per a esbrinar el seu ID o nom. Una vegada coneguem l'identificador (assumim que el seu prefix és "b875…"), podrem esbrinar la seua adreça IP amb:
docker inspect b875 | grep IPAddress
…o simplement utilitzant **docker inspect b875** i cercant alguna línia amb una adreça IP entre l'eixida proporcionada.

5. Assumirem que l'adreça IP obtinguda en la qüestió anterior és 172.17.0.2. Amb aquesta informació, genera la imatge **worker**. Escriu el Dockerfile necessari. (2 puntus).

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

Aquesta seria una solució básica per a aquesta qüestió. Una altra solució més realista utilitzaria una variable d'entorn per a rebre l'adreça IP del broker. Per exemple, utilitzant la següent com a última línia del Dockerfile:
CMD node Worker.js $BROKER_IP
En aquest cas, el valor de la variable d'entorn hauria de passar-se en l'ordre **docker run** a utilitzar per a executar una instància del component en un contenidor. Alguna cosa així:
docker run –e BROKER_IP=172.17.0.2 worker
…assumint que ja s'ha creat una imatge **worker** com se sol·licita en la qüestió 6.

6. Escriu l'ordre Docker per a generar la imatge del treballador. El nom de la imatge ha de ser "**worker**". (1 punt)

Aquesta ordre hauria de llançar-se en la carpeta "Worker":
docker build –t worker .