# SEGUNDO PARCIAL PRÁCTICAS TSR

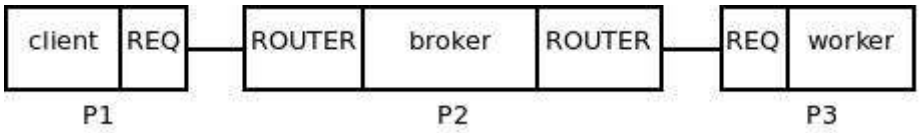
Este examen contiene 20 cuestiones multi-opción. En cada cuestión solo una de las respuestas es correcta. Las respuestas deben proporcionarse en una hoja SEPARADA que ha sido repartida con este enunciado.

Todas las cuestiones tienen el mismo valor. Si la respuesta es correcta, proporcionan 0.5 puntos. Si la respuesta es errónea la contribución es negativa, equivalente a la quinta parte del valor de una respuesta correcta; es decir, -0.1 puntos. Por tanto, en caso de duda se recomienda dejar la cuestión en blanco.

Esta parte del examen puede completarse en una hora.

---

Suponga una aplicación con tres componentes: "client.js", "broker.js" y "worker.js". Estos componentes se comunican mediante sockets ZMQ, usando una arquitectura como la descrita en Lab2:




Esta aplicación se despliega lanzando una instancia de cada "`client.js`" (proceso P1), "`broker.js`" (proceso P2) y "`worker.js`" (proceso P3) en tres máquinas distintas. Asuma que la identidad del proceso cliente es la cadena "`ADFG`" y que la identidad del proceso trabajador es la cadena "`ERTH`". Además, asuma que el broker no guarda estado para las peticiones reenviadas a los trabajadores. Responda las siguientes cuestiones sobre el comportamiento de esta aplicación:

1. **Suponga que P1 envía la cadena "**NEW REQUEST**" por su socket REQ y que recibirá la respuesta "**REQUEST PROCESSED**" a través de ese mismo socket. Seleccione la respuesta verdadera:**

| | |
|---|---|
| A | P3 recibe este mensaje en su socket REQ: ["ERTH","","ADFG","","NEW REQUEST"] |
| B | Cuando P3 responde la petición de P1, P2 recibe en su socket backend este mensaje: ["ERTH","","ADFG","","REQUEST PROCESSED"] |
| C | P2 envía a P1 esta respuesta: ["ERTH","","ADFG","","NEW REQUEST"] |
| D | Cuando P3 responde la petición de P1, P2 recibe en su socket backend este mensaje: ["ERTH","","REQUEST PROCESSED"] |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

2. **Suponga que ahora desplegamos la aplicación con una instancia "worker" adicional (proceso P4). Suponga que P1 envía 10 peticiones consecutivas por su socket REQ. Seleccione la respuesta correcta:**

| | |
|---|---|
| A | Tener dos instancias de worker.js nos permite reducir a la mitad el tiempo medio de procesamiento de las 10 peticiones realizadas por P1. |
| B | El socket ROUTER frontend de P2 permite el envío concurrente de las 10 peticiones de P1, distribuyéndolas entre los dos procesos trabajadores (P3,P4), reduciendo así el tiempo de procesamiento de las peticiones de P1. |
| C | El socket REQ de P1 permite enviar varias peticiones concurrentemente. |
| D | Tener dos instancias de worker.js no proporciona ninguna ayuda para reducir el tiempo medio de procesamiento de las 10 peticiones realizadas por P1, comparado con el caso de una sola instancia de worker.js |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

3. **Suponga que ahora tenemos una versión diferente del broker, `broker2.js`, con un socket DEALER como backend. Suponga también que tenemos otra versión del trabajador, `worker2.js`, que usa un socket REP para comunicarse con el backend del broker.**
**Se lanza la aplicación con un proceso para "`client.js`" (proceso P1), una para "`broker2.js`" (proceso P2) y otro para `worker2.js` (proceso P3). Si P1 envía la cadena "`NEW REQUEST`" por su socket REQ, seleccione la respuesta correcta:**

| | |
|---|---|
| A | Cuando la petición llega a P3 a través de su socket REP desde P2, el mensaje recibido por P3 es ["NEW REQUEST"]. |
| B | El socket REP de P3 no necesita identidad para comunicarse con P2. |
| C | El socket REQ de P1 todavía necesita identidad para comunicarse con P2. |
| D | El socket DEALER (backend) de P2 necesita enviar el siguiente mensaje: ["ADFG","","NEW REQUEST"] |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

Considere el fragmento de código siguiente:

```
01:  var zmq = require('zmq')
02:    , A = zmq.socket(X)
03:    , B = zmq.socket(Y);
04:
05:  A.bindSync('tcp://*:1111');
06:  B.bindSync('tcp://*:2222');
07:
08:  A.on('message', function() {
09:    var args = Array.apply(null, arguments);
10:    B.send(args);
11:  });
12:
13:  B.on('message', function() {
14:    var args = Array.apply(null, arguments);
15:    A.send(args);
16:  });
```

**4. El código anterior <u>no puede</u> ser el de un broker si...**

| | |
|---|---|
| A | Los clientes conectan con el broker vía un socket REQ. |
| B | X es 'router' y los clientes se conectan al puerto 1111 del nodo del broker. |
| C | X es 'dealer' y los trabajadores se conectan al puerto 2222 del nodo del broker. |
| D | Los trabajadores se conectan con un socket REP. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

En la parte **`zmqavanzado`** de la práctica 2, hay mensajes compuestos por 3, 4 y 5 segmentos:

**5. Los mensajes enviados o recibidos por P2 utilizando su socket backend <u>no tienen</u> 5 segmentos cuando...**

| | |
|---|---|
| A | El worker avisa de su disponibilidad. |
| B | El worker avisa de un error. |
| C | El worker termina de procesar una petición. |
| D | El mensaje no ha sido enviado por el worker. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**6. Los mensajes enviados o recibidos por P2 utilizando su socket frontend <u>no tienen</u> 3 segmentos cuando...**

| | |
|---|---|
| A | El cliente realiza una petición. |
| B | El broker retorna un resultado. |
| C | El broker retorna un error. |
| D | El cliente conecta por primera vez. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

La sección **zmqexperto** de la práctica 2 sugiere el uso de promesas...

**7. Las promesas son transmitidas desde el broker al cliente:**

| | |
|---|---|
| A | Directamente, como un segmento adicional del mensaje. |
| B | Codificadas en JSON, como un segmento adicional. |
| C | Modificando uno de los segmentos existentes. |
| D | De ninguna manera, pues las promesas son variables globales y no necesitan ser transmitidas. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**8. En 0MQ …**

| | |
|---|---|
| A | Los mensajes son cadenas de caracteres. |
| B | Los mensajes son vectores de valores. |
| C | Los mensajes son diccionarios, compuestos por pares clave/valor. |
| D | Los mensajes contienen varios segmentos y se entregan atómicamente. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**9. En la práctica 2…**

| | |
|---|---|
| A | El conjunto de trabajadores es estático y su tamaño se le da al broker como un argumento. |
| B | Pueden añadirse nuevos trabajadores, enviando su URL a los procesos clientes. |
| C | Pueden añadirse nuevos trabajadores, enviando su URL al broker. |
| D | El conjunto de trabajadores es estático y su tamaño se pasa a los clientes como un argumento. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**10. En zmqexperto, la configuración dinámica del broker se realiza mediante...**

| | |
|---|---|
| A | …el socket frontend del broker. |
| B | …el socket backend del broker. |
| C | …un socket REQ adicional del broker. |
| D | …un socket PUB adicional del broker. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**11.** **En la práctica 3, la orden "`docker run -i -t tsir/baselab3`"**

| | |
|---|---|
| A | Crea un contenedor que ejecutará el shell en la imagen **`tsir/baselab3`**. |
| B | Inicia la ejecución de una máquina virtual, basada en la imagen **`tsir/baselab3`**. |
| C | Inicia un proceso sh en una máquina virtual preexistente ejecutando la imagen **`tsir/baselab3`**. |
| D | Inicia el programa sh en la imagen **`tsir/baselab3`** ejecutándose en el entorno actual. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

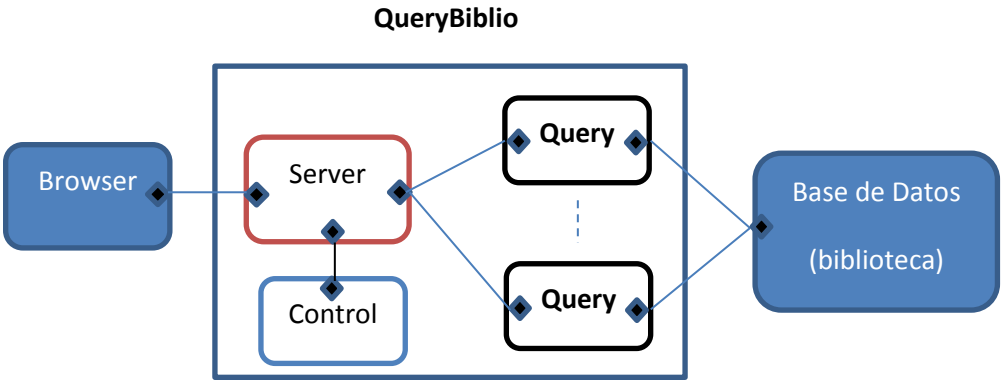
**12.** **En la práctica 3 …**

| | |
|---|---|
| A | Un componente se define mediante un directorio que contiene un **`Dockerfile`** concreto. |
| B | El fichero ejecutado por omisión cuando la imagen de un componente es ejecutada no puede ser cambiado por el desarrollador del componente. |
| C | El fichero ejecutado por omisión cuando la imagen de un componente es ejecutada es **`index.js`**, en el directorio raíz del componente. |
| D | El componente no necesita especificar un fichero **`package.json`**. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**13.** **Suponga que hemos desplegado un servicio utilizando contenedores docker con los identificadores "client", "server", "server_1", "server_2"**
**Podemos parar el servicio completo con la orden…**

| | |
|---|---|
| A | docker rm –f client server server_1 server_2 |
| B | docker rm –f client server |
| C | docker rmi –f client server |
| D | docker rmc client server server_1 server_2 |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

Considere un servicio web llamado QueryLibrary, que contiene información sobre la base de datos de una biblioteca. El servicio tiene tres componentes: ***Server*** (que gestiona la interacción con los navegadores), ***Query*** (para gestionar los accesos a la base de datos) y ***Control*** (que a través de un puerto de control puede modificar dinámicamente la apariencia y comportamiento de la interfaz web).




Queremos desplegar esta aplicación utilizando las tecnologías de resolución de dependencias presentadas en la práctica 3.

Tenemos tres directorios: Components/Server, Components/Query y Components/Control, conteniendo cada uno la definición de componentes. Suponga los siguientes contenidos para los ficheros:

**"Components/Server/config/default.js"**

```
module.exports = {
  provides : {
     controlPort : 8001,
     queryPort : 8002
     },
  external : {
   webPort : 8000 }
}
```

**"Components/Query/config/default.js"**

```
module.exports = {
  requires : {
     serverUrl: tcp://localhost:8002
     },
  parameter : {
   BDServer : 192.168.1.1:8003 }
}
```

**"Components/Control/config/default.js"**

```
module.exports = {
  requires : {
     serverUrl: tcp://localhost:8001
     }
}
```

**"Components/Control/server.js"**

```
var utils = require('../utils.js')
……
```

**"Components/Query/server.js"**

```
var utils = require('../utils.js')
……
```

**"Components/utils.js"**

```
…
…
…
…
..
```

**14. Selecciona el valor correcto para el atributo "`links`" en el descriptor de servicio de QueryLibrary …**

| | |
|---|---|
| A | `links: {`<br>`  Query : {  serverUrl : [ "Server", "queryPort"] },`<br>`  Control : { serverUrl : [ "Server", "controlPort"] }`<br>`}` |
| B | `links: {`<br>`  Query : {  QueryUrl : [ "Server", "queryPort"] },`<br>`  Control : { ControlUrl : [ "Server", "controlPort"] }`<br>`}` |
| C | `links: {`<br>`  Query : [ "Server", "queryPort"] ,`<br>`  Control : [ "Server", "controlPort"]`<br>`}` |
| D | `links: {`<br>`  Query : {  queryPort : [ "Server", "serverUrl"] },`<br>`  Control : { controlPort : [ "Server", "serverUrl"] }`<br>`}` |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**15. En el servicio QueryLibrary…**

| | |
|---|---|
| A | Los valores de las referencias externas (en Server) pueden ser modificados en el descriptor de servicio. |
| B | Los valores de los puertos especificados en los atributos "provides" permanecerán invariables en cada despliegue. |
| C | Los valores de las referencias externas (en Server) pueden ser modificados en el descriptor de despliegue. |
| D | El valor de la URL especificada en el atributo "requires" de "Components/Control/config/default.js" no puede cambiar en el despliegue. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**16. Tras terminar el despliegue del servicio QueryLibrary…**

| | |
|---|---|
| A | Las únicas instancias en ejecución pertenecen al componente Server. |
| B | Habrá como máximo dos instancias en ejecución del componente Query. |
| C | Habrá como máximo cinco instancias en ejecución del componente Control. |
| D | Habrá el mismo número de instancias en ejecución de los componentes Control y Query. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**17.** **El `noLocationDeployer` mencionado en la práctica 3 se construye modificando…**

| | |
|---|---|
| A | El código principal de `deployer.js` |
| B | El método `deploy` de `Deployer` |
| C | El método `buildImage` de `Deployer` |
| D | La función `configureComponent()` en `basicdeployer.js` |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**18.** **El `noImageDeployer` mencionado en la práctica 3 se construye modificando:**

| | |
|---|---|
| A | El código principal de `deployer.js` |
| B | El método `deploy` de `Deployer` |
| C | El método `buildImage` de `Deployer` |
| D | La función `configureComponent()` en `basicdeployer.js` |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**19.** **La orden '`docker run –t -r –entrypoint=/bin/cat tsir/balancer /app/config/default.js`'**
**NOTA:** La opción –entrypoint es equivalente a la palabra reservada ENTRYPOINT de los Dockerfile.

| | |
|---|---|
| A | Muestra en la consola el contenido del fichero de configuración por omisión del componente `balancer` |
| B | Ejecuta el componente `balancer` normalmente. |
| C | Termina con un error. |
| D | Lanza una instancia del componente `balancer` |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |

**20.** **La orden "`docker build –t tsir/balancer Components/balancer`"…**

| | |
|---|---|
| A | Construye la imagen tsir/balancer para el componente `balancer` a partir del contenido del directorio `Components/balancer`. |
| B | Falla porque no se ha lanzado desde el directorio `Components/balancer`. |
| C | Lanza el componente `balancer`. |
| D | Elimina una instancia del componente `balancer`. |
| E | Todas las anteriores. |
| F | Ninguna de las anteriores. |