Este examen contiene 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. Las primeras 7 cuestiones se refieren a la práctica 1, y las 14 restantes pertenecen a las prácticas 2 y 3.

Dentro de cada parte, todas las cuestiones tienen el mismo valor. Si la respuesta es errónea la contribución es negativa, equivalente a 1/5 del valor de una respuesta correcta. Por tanto, en caso de duda se recomienda dejar la cuestión en blanco.

SOLUCIONES AL EXAMEN RECUPERACIÓN PRÁCTICA 1 TSR

1. La orden "git push origin master"

A	Cuando tiene éxito, deja la rama del depósito remoto master apuntando al mismo commit que la rama master local
B	Borra el directorio de trabajo actual
C	Siempre copia los commit locales en el depósito remoto
D	Trae todos los commits del depósito remoto "origin" a nuestro depósito local
E	Todas las anteriores
F	Ninguna de las anteriores

2. Un depósito Git local…

A	Puede disponer de un depósito remoto ubicado en otro depósito local
B	No puede ser modificado
C	Debe asociarse a dos depósitos Git remotos, uno para el desarrollador y otro para el equipo
D	Solo puede asociarse a un único depósito remoto
E	Todas las anteriores
F	Ninguna de las anteriores

3. La orden "mkdir project;cd project;git init;git commit" …

A	Produce un error
B	Crea un depósito Git vacío
C	Hace un *pull* de los contenidos de su depósito remoto
D	Hace un *push* de los contenidos del directorio al remoto
E	Todas las anteriores
F	Ninguna de las anteriores

Considere este fragmento de código 3:

```
01:  var net = require('net');
02:
03:  var LOCAL_PORT  = 8000;
04:  var LOCAL_IP  = '127.0.0.1';
05:  var REMOTE_PORT = 80;
06:  var REMOTE_IP = '158.42.156.2';
07:
08:  var server = net.createServer(function (socket) {
09:      socket.on('data', function (msg) {
10:         var serviceSocket = new net.Socket();
11:         serviceSocket.connect(parseInt(REMOTE_PORT), REMOTE_IP, function (){
12:             serviceSocket.write(msg);
13:         });
14:         serviceSocket.on('data', function (data) {
15:             socket.write(data);
16:         });
17:         console.log("Client connected");
18:     });
19:  }).listen(LOCAL_PORT, LOCAL_IP);
20:  console.log("TCP server accepting connection on port: " + LOCAL_PORT);
```

Llamemos P1 al proceso que se obtiene al ejecutar este código. Se supone que los datos de los clientes que conectan al puerto 8000 llegan en bloques (*chunks*) mediante esa conexión que se ha establecido. Cada bloque provoca un evento 'data' en el socket que representa la conexión.

Sea P2 un proceso iniciado tras P1 en el mismo ordenador, de forma que P2 conecta al puerto local 8000.

Sea P3 un proceso servidor a la espera de conexiones en REMOTE_IP:REMOTE_PORT

Conteste a las 3 cuestiones siguientes.

4. Si P2 envía únicamente un byte a P1 y cierra la conexión, P1 finaliza abriendo como máximo …

A	… una conexión con P3.
B	… dos conexiones con P3.
C	… tres conexiones a localhost.
D	… 0 conexiones a P3.
E	Todas las anteriores
F	Ninguna de las anteriores

5. Imagine que P1 recibe exactamente dos bloques de datos en secuencia, D1 y a continuación D2, mediante su conexión de P2. Entonces, en ausencia de fallos, …

A	P1 abre exactamente dos conexiones con P3.
B	D2 nunca puede llegar a P3
C	D1 siempre llega a P3 antes que D2
D	D2 siempre llega a P3 antes que D1
E	Todas las anteriores
F	Ninguna de las anteriores

6. Considere un pequeño cambio en el fragmento de código, en el cual la línea 18 se coloca inmediatamente antes que la 13, y la línea 9 se coloca inmediatamente a continuación de la línea 11 (ver extracto a continuación). **Asuma que P2, usando la única conexión que establece con P1, envía 3 bytes con retardos arbitrarios entre ellos, provocando que P1 reciba uno, dos o tres bloques de datos.**

```
07:
08:  var server = net.createServer(function (socket) {
09:      var serviceSocket = new net.Socket();
10:      serviceSocket.connect(parseInt(REMOTE_PORT), REMOTE_IP, function (){
11:         socket.on('data', function (msg) {
12:            serviceSocket.write(msg);
13:         });
14:       });
15:       serviceSocket.on('data', function (data) {
16:          socket.write(data);
17:       });
18:       console.log("Client connected");
19:  }).listen(LOCAL_PORT, LOCAL_IP);
```

Entonces, en ausencia de fallos …

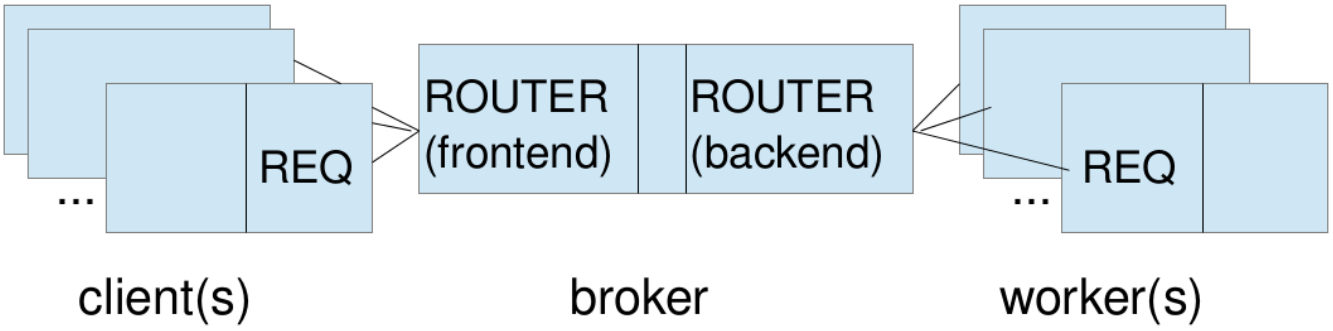
A	P1 abre un máximo de 3 conexiones con P3.
B	P1 envía exactamente tres bytes a P3.
C	P1 abre exactamente una conexión con P3.
D	P1 abre como máximo 2 conexiones con P3.
E	Todas las anteriores
F	Ninguna de las anteriores

7. En el proxy inverso que había que construir en la parte final de Lab1 (apartado 3) …

A	El proxy necesitaba disponer de un puerto “controlador” extra.
B	El controlador no necesita ejecutarse en el mismo equipo que el proxy.
C	El controlador puede configurar la IP de destino para cada puerto del proxy.
D	El controlador puede configurar el puerto del servicio remoto con un valor distinto al del puerto del proxy.
E	Todas las anteriores
F	Ninguna de las anteriores

SOLUCIONES AL EXAMEN RECUPERACIÓN PRÁCTICAS 2 Y 3 TSR

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 o varias instancias de "`client.js`", una de "`broker.js`", y una o varias de "`worker.js`" en nodos diferentes.

Responda las dos cuestiones siguientes sobre esta aplicación:

8. Suponga estos cuatro escenarios:

	client.js	worker.js	Comentarios
a)	1 instancia	1 instancia	El cliente procesa 10 peticiones secuenciales
b)	2 instancias	1 instancia	Cada cliente procesa 5 peticiones secuenciales
c)	2 instancias	2 instancias	Cada cliente procesa 5 peticiones secuenciales
d)	1 instancias	2 instancias	El cliente procesa 10 peticiones secuenciales

Si llamamos T al tiempo medio de procesamiento de una petición:

A	T en el escenario b) es aproximadamente la mitad de lo que le correspondería en el escenario a).
B	T en el escenario c) es aproximadamente la mitad de lo que le correspondería en el escenario a).
C	T en el escenario d) es aproximadamente la mitad de lo que le correspondería en el escenario a).
D	En los escenarios c) y d), T tiene aproximadamente el mismo valor.
E	Todas las anteriores
F	Ninguna de las anteriores

Considere el siguiente fragmento de código que implementa un broker:

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

9. Elija la opción correcta respecto al anterior fragmento:

A	Un cliente que se ejecute en el mismo equipo puede ejecutar este fragmento:
	`var s=zmq.socket('req');`
	`s.connect('tcp://localhost:2222');`

B	Un worker que se ejecute en el mismo equipo puede ejecutar este fragmento:
	`var s=zmq.socket('req');`
	`s.bind('tcp://localhost:2222');`

C	Un cliente que se ejecute en el mismo equipo puede ejecutar este fragmento:
	`var s=zmq.socket('sub');`
	`s.connect('tcp://localhost:1111');`

D	Un worker que se ejecute en el mismo equipo puede ejecutar este fragmento:
	`var s=zmq.socket('req');`
	`s.connect('tcp://localhost:1111');`

E	Todas las anteriores
F	Ninguna de las anteriores

10. Suponga una única instancia del cliente (con identidad "C") y del worker (con identidad "W"), pero dos escenarios diferentes:

	backend del broker	socket del worker
a)	ROUTER	REQ
b)	DEALER	REP

Si un proceso cliente envía la cadena "NEW REQUEST" por medio de su socket REQ, elija cuál es la respuesta correcta:

A	Cuando los mensajes llegan al socket del worker, ese mensaje es el mismo en ambos escenarios.
B	En ambos escenarios, el socket del worker necesita una identidad para poder comunicarse con el backend del broker.
C	En ambos escenarios, el socket del cliente necesita una identidad para poder comunicarse con el frontend del broker.
D	En ambos escenarios, el broker procesa llamadas al método send() del socket backend con este mensaje: ["W", "","C","","NEW REQUEST"]
E	Todas las anteriores
F	Ninguna de las anteriores

En la parte **`zmqavanzado`** (Lab2), aparecen mensajes compuestos por 3, 4 y 5 segmentos:

11. Desde el punto de vista del broker, los mensajes que envía o recibe tienen 4 segmentos cuando:

A	El worker informa sobre su disponibilidad con un mensaje al socket backend.
B	El worker comunica un error enviando un mensaje al socket backend.
C	El worker contesta a una petición (enviando la respuesta al broker), enviando un mensaje al socket backend.
D	El cliente envía una petición al socket frontend.
E	Todas las anteriores
F	Ninguna de las anteriores

12. Desde el punto de vista del broker, los mensajes que envía o recibe por su socket `frontend`

A	Siempre tienen una longitud de 3 segmentos.
B	Pueden tener 5 segmentos cuando devuelva un error.
C	Pueden tener 4 segmentos cuando devuelva un error procedente del worker.
D	Su tamaño nunca será de 3 segmentos.
E	Todas las anteriores
F	Ninguna de las anteriores

13. En la parte zmqexperto de Lab2:

A	El broker puede configurarse para que use múltiples estrategias de distribución de las peticiones entrantes.entre los workers.
B	Existe una función auxiliar que calcula la carga actual de un worker
C	El broker es informado periódicamente de la carga de trabajo de cada worker.
D	No debería admitirse que dos clientes empleen el mismo identificador
E	Todas las anteriores
F	Ninguna de las anteriores

14. En zmqexperto …

A	Los clientes mantienen información sobre los workers mediante una estructura de datos.
B	Cuando todos los workers se encuentran ocupados, el broker rechaza nuevas peticiones.
C	El broker debe recibir información sobre el alta/baja de workers
D	El broker difunde a todos los clientes el estado de cada worker.
E	Todas las anteriores
F	Ninguna de las anteriores

15. En Lab3, los Dockerfile definen las imágenes a construir. Al construir una imagen, Docker inicia un contenedor en el que se pueden ejecutar órdenes que modifiquen los contenidos de ese mismo contenedor. En dichos Dockerfile, la directiva …:

A	`RUN <command>` ejecuta `command` inmediatamente dentro del contenedor
B	`ADD . /app` copia recursivamente todos los archivos y directorios desde el directorio de creación del contenedor (*build directory*) al directorio `/app` dentro del contenedor.
C	`ONBUILD <dockercommand>` consigue que `<dockercommand>` se ejecute la próxima vez que se construya una imagen basada en ésta que ahora se está creando.
D	`ENTRYPOINT <command>` provoca que cualquier contenedor iniciado a partir de esta imagen ejecute `<command>` como proceso inicial.
E	Todas las anteriores
F	Ninguna de las anteriores

16. Un componente en Lab3 …:

A	Se implementa dentro de un directorio que contenga todo su código junto con un `Dockerfile`
B	Debe empaquetarse mediante una imagen `docker`, construida considerando el `Dockerfile` que se encuentre en el directorio que defina al componente
C	Suponiendo que no puede realizarse una carga dinámica de código en un contenedor que ya esté ejecutándose, TODO el código que emplee debe encontrarse empaquetado dentro de la imagen del componente para que sus instancias se puedan desplegar correctamente.
D	Por defecto, su programa principal se encuentra en el archivo `server.js`
E	Todas las anteriores
F	Ninguna de las anteriores

17. Docker resuelve los siguientes aspectos del despliegue:

A	Prepara toda la pila software necesaria para la instanciación de un componente, de manera adecuada y fiable.
B	Gestiona los objetos de configuración de los componentes.
C	Configura los parámetros necesarios para los algoritmos internos implementados en cada componente.
D	Configura las dependencias respecto a otros componentes.
E	Todas las anteriores
F	Ninguna de las anteriores

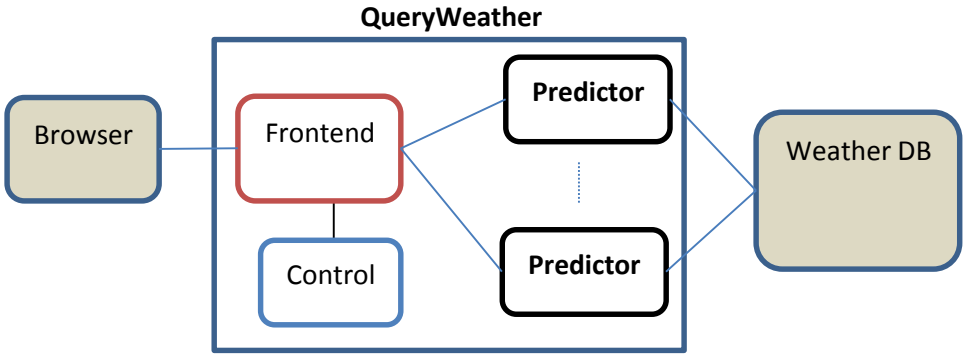
18. En el servicio webLabel, se solicitó modificar el componente Frontend para obtener un nuevo componente FrontendP que pueda maximizar la concurrencia de peticiones que circulan por el socket con el que conecta al broker. El tipo de este socket es …

A	REQ
B	PULL
C	DEALER
D	SUB
E	Todas las anteriores
F	Ninguna de las anteriores

19. En el servicio webLabel, se solicitó implementar el componente LabelBuilder de manera que pueda conectar con el broker usado en Lab2. Para que esto funcione, LabelBuilder debe usar un socket de tipo …

A	PULL
B	SUB
C	REQ
D	PUB
E	Todas las anteriores
F	Ninguna de las anteriores

Suponga un servicio web denominado QueryWeather, que obtiene información y predicciones sobre meteorología. El servicio consiste en 3 componentes: ***Frontend*** se encarga de la interacción con los navegadores; ***Predictor*** accede a la base de datos meteorológica y calcula las predicciones correspondientes a las consultas de los usuarios. Por último, ***Control*** puede modificar dinámicamente el aspecto de la interfaz web mediante un puerto de control del ***Frontend***.



Deseamos desplegar esta aplicación usando las tecnologías de resolución de dependencias aplicadas en Lab3. Disponemos de tres directorios: Components/Frontend, Components/Predictor y Components/Control, donde cada uno contiene la implementación y definición del componente correspondiente. Suponga que los ficheros tienen los siguientes contenidos:

"Components/Frontend/config/default.js"

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

"Components/Predictor/config/default.js"

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

"Components/Control/config/default.js"

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

"Components/Control/server.js"

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

"Components/utils.js"

```
…
…
```

20. En el servicio QueryWeather …

A	Predictor DEBE usar sockets ZMQ para comunicarse con la base de datos meteorológica
B	Todos los despliegues deben tener únicamente dos instancias del componente Predictor
C	Todos los despliegues deben tener únicamente una instancia del componente Frontend.
D	Ningún despliegue puede declarar más de una instancia del componente Control.
E	Todas las anteriores
F	Ninguna de las anteriores

21. Al desplegar un servicio, se inicia un contenedor por cada instancia desplegada de cada componente. Si por cualquier razón el programa de un componente no pudiera iniciarse correctamente dentro de su contenedor, ese contenedor se detendría nada más iniciarse. Con los ficheros anteriores, suponga que el descriptor de despliegue se define como: 1 instancia Frontend, 5 instancias Control, y 10 instancias Predictor. Tras concluir el despliegue del servicio QueryWeather …

A	Quedan 2 instancias del componente Control en ejecución
B	El componente Control encontrará dentro de su contenedor al fichero util.js
C	El componente Control no podrá usarse para cambiar el aspecto presentado por el Frontend una hora después de la finalización del despliegue
D	El componente Frontend siempre fallará
E	Todas las anteriores
F	Ninguna de las anteriores