TSR	Recup. Primer Parcial	Examen resuelto	2020-01-27	A

20 cuestiones, todas con el mismo valor. 4 opciones cada una (sólo una correcta). Cada error resta 1/3 de un acierto. Debe contestar en la hoja de respuestas

Desde el enunciado de algunas cuestiones se referencian los siguientes programas Node con 0MQ.

```
1  // server.js
2     const zmq = require('zeromq')
3     const rep = zmq.socket('rep')
4     const port = process.argv[2]
5     const reply = process.argv[3] || "World"
6     rep.bindSync('tcp://127.0.0.1:'+port)
7     let nMsg = 0
8     rep.on('message', msg => {
9        rep.send(reply + ' ' + msg + ' ' + ++nMsg)
10    })
```

```
1  // client.js
2     const zmq = require('zeromq')
3     const req = zmq.socket('req')
4     const nCon = process.argv[2]
5     let nMsg = 0
6     for (let i = 0; i < nCon; i++) {
7        req.connect('tcp://127.0.0.1:800' + i)
8     }
9     setInterval(() => {req.send(++nMsg)}, 100)
10    setTimeout(() => {process.exit()}, 410)
11    req.on('message', msg => {
12       process.stdout.write(nMsg +': ' + msg +' * ')
13    })
```

```
1  // publisher.js
2     const zmq = require('zeromq')
3     const pub = zmq.socket('pub')
4     const themes = ['PRG', 'TSR', 'EDA']
5     let n = 0
6     pub.bindSync('tcp://*:8001')
7     setInterval(() => {
8        for (let i = 0; i < themes.length; i++) {
9           pub.send(themes[i] + ' ' + n++)
10       }
11       if (n >= 10) process.exit()
12    }, 1000)
```

```
1  //subscriber.js
2     const zmq = require('zeromq')
3     const sub = zmq.socket('sub')
4     const themes = process.argv.slice(2)
5     sub.connect('tcp://localhost:8001')
6     for (let i = 0; i < themes.length; i++) {
7        sub.subscribe(themes[i])
8     }
9     process.stdout.write('ok ')
10    sub.on('message', msg => {
11       process.stdout.write(msg+' ')
12    })
```

1 *¿Qué ventaja aportan los servidores asincrónicos cuando se comparan con los servidores concurrentes?*

- **a** El despliegue de servidores asincrónicos puede automatizarse, el de los servidores concurrentes, no.
- **b** Los servidores asincrónicos pueden replicarse, los servidores concurrentes no pueden.
- **c** Los servidores asincrónicos no suelen introducir bloqueos, los servidores concurrentes lo hacen con frecuencia.
- **d** Todas las opciones son correctas.

2 *La Wikipedia se escogió como caso de estudio en el Tema 1 de esta asignatura porque ofrece un buen ejemplo de los siguientes aspectos:*

- **a** Es un servicio altamente escalable y está relativamente bien documentado.
- **b** Ilustra que los proxies inversos, los equilibradores de carga y la replicación no pueden utilizarse conjuntamente.
- **c** Ilustra cómo puede desplegarse un servicio muy escalable sobre una plataforma cloud (es decir, un sistema PaaS).
- **d** Todas las opciones son correctas.

3 *Indica el resultado que se escribe en pantalla al evaluar el siguiente fragmento de código*

```
const f = function () {
  for (var i=0; i<3; i++) {
    setTimeout(() => console.log(i), (3–i)*1000)
  }
}
f()
```

a Genera un error durante la ejecución

b Escribe (en líneas consecutivas) 0,1,2

c Escribe (en líneas consecutivas) 3,3,3

d Escribe (en líneas consecutivas) 2,1,0

4 *Indica el resultado al ejecutar el siguiente fragmento de código*

```
var a = [1,2,'hola']
a.push(a.shift())
console.log(a)
```

a Genera un error, porque un array no puede mezclar valores de tipos distintos

b Escribe el valor [1,2,'hola']

c Genera un error, porque el resultado de a.shift() no es un argumento válido para la función a.push

d Escribe el valor [2,'hola',1]

5 *Indica el resultado al ejecutar el siguiente fragmento de código*

```
function f(n) {return n<2? 1:f(n–2)+f(n–1)}
setTimeout(()=>console.log(1),1000)
f(36) // requiere mas de 3000ms
console.log(2)
setTimeout(()=>console.log(3),100)
```

a Escribe en pantalla 1 2 3

b Escribe en pantalla 1 3 2

c Escribe en pantalla 2 3 1

d Escribe en pantalla 2 1 3

6 *Indica el resultado al ejecutar el siguiente fragmento de código*

```
let ob = {
  x: 23,
  y:{
    a: 45,
    b:[4,5]
  }
}
console.log(JSON.parse(ob))
```

a Escribe en pantalla
`{"x":23,"y":{"a":45,"b":[4,5]}}`

b Genera un error de ejecución, porque el argumento de la función JSON.parse no es válido

c Genera un error de ejecución, porque JSON.parse no puede aplicarse a objetos anidados

d Genera un error de ejecución, porque JSON.parse devuelve un objeto, y no podemos aplicar directamente un objeto como argumento a console.log

7 *JavaScript es un lenguaje de programación:*

a Compilado.

b Orientado a eventos.

c Creado por Google. Su primera edición estaba integrada en el navegador Chrome.

d Con el que se desarrolló Docker y la mayor parte de los componentes de la Wikipedia.

8 *En un programa JavaScript queremos mostrar todos los argumentos recibidos desde la línea de órdenes (sin incluir el nombre del propio programa). El código necesario para ello sería:*

a
```
for (let i=2; i<process.argv.length; i++)
    console.log(process.argv[i])
```

b
```
process.argv.forEach(
   (a,i) => {if (i>1) console.log(a)})
```

c
```
process.argv.slice(2).forEach(
    (a) => {console.log(a)})
```

d Todas las opciones son correctas.

9 *¿Qué líneas del siguiente programa utilizan clausuras?*

```
function writing(x) {
  console.log("--- Writing after " + x + 'sec.')
}
function writing2(x) {
  return function() {
   console.log("--- Writing after " + x + " seconds")
  }
}
setTimeout(function() {writing(6) }, 2000)
setTimeout(writing, 3000)
setTimeout(writing2(4) , 4000)
console.log("root(2) =", Math.sqrt(2))
```

a Todas las que incluyen algún setTimeout().

b La del primer y tercer setTimeout().

c La del primer setTimeout().

d La del tercer setTimeout().

10 *¿Cuántos callbacks se utilizan en el siguiente programa?*

```
function writing(x) {
  console.log("--- Writing after " + x + 'sec.')
}
function writing2(x) {
  return function() {
   console.log("--- Writing after " + x + " seconds")
  }
}
setTimeout(function() {writing(6) }, 2000)
setTimeout(writing, 3000)
setTimeout(writing2(4) , 4000)
console.log("root(2) =", Math.sqrt(2))
```

a Tres.

b Dos.

c Uno.

d Ninguno.

11 *Dados los programas* `client.js` *y* `server.js`, *indique la salida que se muestra en la terminal al ejecutar en dicho terminal la orden:*

```
node server 8000 & node server 8001 Moon &
node client 1 &
```

a 1: Moon 1 1 * 2: Moon 2 2 * 3: Moon 3 3 * 4: Moon 4 4 *

b 1: World 1 1 * 2: Moon 2 1 * 3: World 3 2 * 4: Moon 4 2 *

c 1: World 1 1 * 3: World 3 2 * 2: Moon 2 1 * 4: Moon 4 2 *

d 1: World 1 1 * 2: World 2 2 * 3: World 3 3 * 4: World 4 4 *

12 *Dados los programas* `client.js` *y* `server.js`, *indique la salida que se muestra en la terminal al ejecutar en dicho terminal la orden:*

```
node server 8000 & node server 8001 Bingo &
node client 3 &
```

a 1: World 1 1 * 2: Bingo 2 1 * 3: World 3 2 * 4: Bingo 4 2 *

b 1: World 1 1 * 3: World 3 2 * 2: Bingo 2 1 * 4: Bingo 4 2 *

c 1: World 1 1 * 2: Bingo 2 1 *

d 1: World 1 1 * 3: World 3 2 *

13 *Si las líneas 9 y 10 del programa client.js se sustituyen por las siguientes:*

```
setInterval(() => {req.send(++nMsg)}, 1000)
setTimeout(() => {process.exit()}, 10500)
```

Entonces, si en una terminal se ejecutara:

```
node server 8000 & node server 8001 Samba &
node client 2 &
```

El cliente recibiría:

a 10 respuestas, 5 de cada servidor

b 20 respuestas, 10 de cada servidor

c 10 respuestas del primer servidor

d 12 respuestas, 6 de cada servidor

14 *Dados los programas* `publisher.js` *y* `subscriber.js` *(con una modificación en* `publisher.js`*, indicada en las respuestas), si en una terminal se ejecutara:*

```
node publisher & node subscriber PRG TSR &
node subscriber TSR EDA &
```

La salida mostrada no variaría si la modificación en publisher.js fuera:

a Línea 9 del código. Cambiar el incremento de la variable n:

```
pub.send(themes[i] + ' ' + ++n)
```

b Línea 4 del código. Cambiar el contenido del array themes, eliminando `PRG´:

```
const themes = ['TSR', 'EDA']
```

c Línea 12 del código. Cambiar el intervalo de temporización a 5 segundos:

```
setInterval(“, 5000)
```

d Línea 11 del código. Cambiar la condición del if a valor 50:

```
if (n >= 50) process.exit()
```

15 *Tenemos dos procesos servidores que utilizan cada uno de ellos un socket REP asociado a uno de los puertos disponibles en la máquina local.*

Para interactuar con ellos se ha escrito un programa cliente que utiliza un socket REQ conectado a ambos servidores. El proceso cliente envía cada segundo un mensaje a través de su socket y gestiona las respuestas correspondientes.

Si en cierto momento el usuario mata uno de los procesos servidores, ¿qué efecto tendrá esto sobre la ejecución del proceso cliente?

a El cliente dejará de recibir las respuestas del servidor que ha fallado pero seguirá recibiendo las del otro servidor.

b En menos de dos segundos el cliente dejará de obtener respuestas.

c El cliente recibirá una excepción y abortará cuando intente enviar su siguiente petición al servidor caído.

d Ninguna de las demás opciones es correcta.

16 *El siguiente código fue utilizado en la primera parte de la práctica 1. ¿Qué salida muestra en pantalla?*

```
for(var i=0; i<10; i++)
  setTimeout(function(indice)
    {return function(){
     console.log("indice: ",indice," i: ",i)}
   }(i),i*1000);

  console.log("Terminado codigo script",
  "  valor actual de i: ",i);
```

a Una primera línea que indica que el valor actual de i es 10, seguida por diez líneas mostrando un valor de cero a nueve en cada una para índice y 10 para i.

b Una primera línea que indica que el valor actual de i es 10, seguida por diez líneas mostrando un valor de uno a diez en cada una para índice y 10 para i.

c Una primera línea que indica que el valor actual de i es 10, seguida por diez líneas mostrando en cada una el valor diez tanto para índice como para i.

d Ninguna, pues el proceso aborta en su primera llamada a setTimeout().

17 *Este es el código del proxy básico utilizado en el primer escenario de la parte 2 de la práctica 1. ¿De qué manera podremos comprobar si este proxy funciona?*

```
const net = require ('net')
const LOCAL_PORT = 8000
const LOCAL_IP = '127.0.0.1'
const REMOTE_PORT = 80
const REMOTE_IP = '158.42.4.23'
const server = net.createServer(function (socket) {
 const serviceSocket = new net.Socket()
 serviceSocket.connect (
  parseInt (REMOTE_PORT),
  REMOTE_IP,
  function () {
   socket.on('data', function (msg) {
    serviceSocket.write( msg )
   })
   serviceSocket.on('data', function( data ) {
    socket.write( data )
   })
  })
}).listen(LOCAL_PORT, LOCAL_IP)
console.log("Server accepting conn on port: " +
  LOCAL_PORT)
```

a No funciona. Había que extenderlo ampliamente para que llegara a funcionar. Ese era el objetivo de los demás escenarios de esta parte de la práctica 1.

b No podrá funcionar, pues el servidor remoto utiliza el protocolo HTTP y este proxy utiliza sockets TCP.

c Utilizando el URL http://127.0.0.1:8000 en un navegador iniciado en el mismo ordenador en el que ejecutamos este proxy.

d Ninguna de las demás opciones es correcta.

18 *En el último escenario de la práctica 1 había que desarrollar otro cliente capaz de reprogramar el proxy. Ese programador recibe en línea de órdenes la dirección IP del proxy, y los nuevos valores de IP y puerto correspondientes al servidor. El programa codifica esos valores y los remite como mensaje al puerto 8001 del proxy, tras lo cual termina. El código del proxy se muestra en el enunciado de otra de las cuestiones de este examen. Revíselo y seleccione cuál de las siguientes afirmaciones sobre cuántos sockets deberá emplear el proxy programable es cierta.*

a El proxy no necesita generar nuevos sockets. Basta con hacer un server.listen(8001) para recibir los mensajes de reconfiguración.

b El proxy necesitará crear otro socket mediante net.createServer() y utilizar listen() con él para que escuche en el puerto 8001.

c El proxy no necesita generar nuevos sockets. Basta con hacer un serviceSocket.listen(8001) para que se reciban y procesen los mensajes de reconfiguración.

d Deben generarse al menos dos nuevos sockets y uno de ellos escuchará en el puerto 8001.

19 *En el módulo* `fiSys` *de la práctica de laboratorio 1 se define la siguiente función:*

```
const fs=require("fs");
function readFile(fichero,callbackError,callbackLectura){
  fs.readFile(fichero,"utf8",function(error,datos){
    if(error) callbackError(fichero);
    else callbackLectura(datos);
  });
}
exports.readFile=readFile;
```

El siguiente ejemplo muestra cómo utilizar dicha función:

```
const fiSys=require("./fiSys")
console.log("Invocación lectura asíncrona")
fiSys.readFile("/proc/loadavg",cbError,formato)
```

Determinar la afirmación CIERTA sobre el uso de la función readFile del módulo fiSys:

a Su uso es similar a readFile del módulo fs, pero se utilizan 2 funciones callback

b Se redefine la función readFile para obtener una versión síncrona

c Es incorrecto porque la función cbError y la función formato deberían aceptar un parámetro

d Es incorrecto porque `readFile` ya está definido en el módulo `fs` y no puede ser definido dos veces

20 *En la práctica de laboratorio 1, se usa una función getLoad:*

```
function getLoad(){
  data=fs.readFileSync("/proc/loadavg")
  var tokens = data.toString().split(' ')
  var min1 = parseFloat(tokens[0])+0.01
  var min5 = parseFloat(tokens[1])+0.01
  var min15 = parseFloat(tokens[2])+0.01
  return min1*10+min5*2+min15
}
```

Determinar la afirmación CIERTA sobre dónde se usa esta función.

a En el cliente para pasar la carga del cliente al servidor

b En el cliente para determinar si se puede realizar la petición asíncrona al servidor en función de la carga

c En el servidor para determinar el número de peticiones recibidas

d En el servidor para determinar la carga del mismo expresada como un número

10 cuestiones, todas con el mismo valor. 4 opciones cada una (sólo una correcta). Cada error resta 1/3 de un acierto. Debe contestar en la hoja de respuestas

1 *Para resolver el primer apartado de la práctica 2 debe desarrollar un programa publicador a iniciar desde la línea de órdenes así:*

```
node publicador port numMensajes tema1 tema2 …
```

y que publicaría mensajes de los temas facilitados, circularmente, con una frecuencia de un mensaje por segundo.

Seleccione qué opción permite completar esta solución parcial adecuadamente.

```
const zmq = require('zeromq')
let pub = zmq.socket('pub')
let port = process.argv[2] || 56621
let nm = parseInt(process.argv[3]) || 5
let topics = process.argv.slice(4)
if (!topics[0]) topics = ['t1', 't2', 't3']
let counter = 0
function emit() {
  let m=topics[0]
  pub.send(++counter + ': ' + m)
  topics.shift(); topics.push(m)
}
setInterval(emit,1000)
```

a Añadir `pub.bindSync('tcp://*:'+port)` tras haber asignado valor a `port` y añadir `if (counter==nm) return` como última línea de la función `emit`.

b Añadir `pub.connect('tcp://localhost:'+port)` tras haber asignado valor a `port` y añadir `if (counter>nm) return` como última línea del programa.

c Añadir `pub.bindSync('tcp://*:'+port)` tras haber asignado valor a `port` y añadir `if (counter>nm) return` como última línea de la función `emit`.

d Ninguna de las demás opciones es correcta.

2 *Para resolver el primer apartado de la práctica 2 se necesitaba desarrollar un programa publicador que sería lanzado desde la línea de órdenes de la siguiente manera:*

```
node publicador port numMensajes tema1 tema2 …
```

y que publicaría mensajes de los temas facilitados, siguiendo un turno circular, con una frecuencia de un mensaje por segundo. Además, también debía modificarse el código del correspondiente suscriptor, para que recibiera TODOS LOS MENSAJES difundidos por el publicador.

Suponga que hemos desarrollado el siguiente programa suscriptor y que el DNI del usuario es 23456789. Entonces, ¿qué línea de órdenes deberíamos utilizar para iniciar este suscriptor?

```
// subscriber.js
const zmq = require('zeromq')
let sub = zmq.socket('sub')
sub.connect('tcp://127.0.0.1:'+process.argv[2])
sub.subscribe(process.argv[3])
sub.on('message', (m) => {console.log(m+")})
```

a `node subscriber`

b `node subscriber 56789 all`

c `node subscriber 57890 ″`

d Ninguna de las demás opciones es correcta.

3 *El tercer apartado de la práctica 2 solicitaba la parametrización de los componentes client, broker y worker mencionados en el boletín. Asuma que hemos lanzado ya el broker con esta línea:* `node broker 51230 51231` *en el ordenador donde posteriormente iniciaremos el worker. En ella el primer argumento es el puerto frontend y el segundo el puerto backend. Seleccione qué línea de órdenes se necesitará para lanzar el worker, si queremos que las respuestas del worker sean la cadena* `worker3` *y su identificador sea* `W3`*. Para ello, suponga que el código del worker es el siguiente:*

```
const zmq = require('zeromq')
let req = zmq.socket('req')
req.identity=process.argv[3]
req.connect(process.argv[2])
req.on('message', (c,sep,msg)=> {
 setTimeout(()=> {
  req.send([c,'',process.argv[4]])
 }, 1000)
})
req.send(['','',''])
```

a node worker 51231 worker3 W3

b node worker 51231 W3 worker3

c node worker localhost:51231 W3 worker3

d Ninguna de las demás opciones es correcta.

4 *El cuarto apartado de la práctica 2 solicitaba una ampliación en el broker, de manera que este mostrara cada cinco segundos el número total de peticiones atendidas hasta el momento. Indique qué ampliación de entre las propuestas para la siguiente solución parcial resuelve ese aspecto:*

```
// broker.js
const zmq = require('zeromq')
let cli=[], req=[], workers=[]
let sc = zmq.socket('router') // frontend
let sw = zmq.socket('router') // backend
let counter = 0
sc.bind('tcp://*:9998'); sw.bind('tcp://*:9999')
function stats() {
 console.log('Total requests: ', counter)
}
sc.on('message',(c,sep,m)=> {
 if (workers.length==0) {
  cli.push(c); req.push(m)
 } else {
  sw.send([workers.shift(),'',c,'',m])
 }
})
sw.on('message',(w,sep,c,sep2,r)=> {
 if (cli.length>0) {
  sw.send([w,'', cli.shift(),'',req.shift()])
 } else {
  workers.push(w)
 }
 if (c!='') {
  sc.send([c,'',r])
 }
})
```

a Añadir `counter++` como primera línea del callback para `sw.on()` y añadir `setInterval(stats(),5000)` como última línea del programa.

b Añadir `counter++` tras la línea `sc.send([c,'',r])` y añadir `setInterval(stats,5000)` como última línea del programa.

c Añadir `counter++` como primera línea del callback para `sw.on()` y añadir `setInterval(stats,5000)` como última línea del programa.

d Ninguna de las demás opciones es correcta.

5 *El cuarto apartado de la práctica 2 solicitaba una ampliación en el broker, de manera que este mostrara cada cinco segundos el número de peticiones atendidas por cada worker hasta el momento. Indique qué ampliación de entre las propuestas para la siguiente solución parcial resuelve ese aspecto:*

```
// broker.js
const zmq = require('zeromq')
let cli=[], req=[], workers=[]
let sc = zmq.socket('router') // frontend
let sw = zmq.socket('router') // backend
let wcount = []
sc.bind('tcp://*:9998'); sw.bind('tcp://*:9999')
sc.on('message',(c,sep,m)=> {
  if (workers.length==0) {
   cli.push(c); req.push(m)
  } else {
   sw.send([workers.shift(),'',c,'',m])
  }
})
sw.on('message',(w,sep,c,sep2,r)=> {
  if (cli.length>0) {
   sw.send([w,'', cli.shift(),'',req.shift()])
  } else {
   workers.push(w)
  }
  if (c!='') {
   sc.send([c,'',r])
  } else {
  }
})
setInterval(() => {for (let i in wcount)
  console.log( 'Worker ',i,': ', wcount[i])},5000)
```

a Añadir `wcount[w]++` como primera línea del callback para `sw.on()`.

b Añadir `wcount[r]++` como primera línea del callback para `sw.on()` y añadir `wcount[r]=0` en el bloque `else` vacío.

c Añadir `wcount[w]++` tras la línea `sc.send([c,'',r])` y añadir `wcount[w]=0` en el bloque `else` vacío.

d Ninguna de las demás opciones es correcta.

6 *El quinto apartado de la práctica 2 solicitaba la división del componente broker en dos componentes: un broker para clientes y otro broker para trabajadores. ¿Qué broker se debía utilizar como base para efectuar esa división?*

a Cualquiera de los explicados en el documento RefZMQ.pdf.

b El broker tolerante a fallos.

c El broker tolerante a fallos que puede interactuar con un logger.

d El broker ROUTER-ROUTER básico.

7 *El quinto apartado de la práctica 2 solicitaba la división del componente broker en dos componentes: un broker para clientes y otro broker para trabajadores. ¿Qué estructuras de datos debían mantenerse en el broker para clientes?*

a Ninguna. Todas las estructuras se podían mantener en el broker para trabajadores.

b Los vectores `cli[]` y `req[]`.

c El vector `workers[]`.

d Todas. El broker para trabajadores no debía gestionar ningún vector.

8 *El quinto apartado de la práctica 2 solicitaba la división del componente broker en dos componentes: un broker para clientes y otro broker para trabajadores. ¿Qué patrón de comunicación debía establecerse entre esos dos brokers?*

a REQ/REP.

b PUSH/PULL.

c ROUTER/DEALER.

d DEALER/DEALER.

9 *¿Qué prueba se sugiere en el boletín para evaluar el funcionamiento del broker tolerante a fallos?*

- **a** Una ejecución en la que se replique ese broker y se elimine con `kill` alguna de esas réplicas.
- **b** Una ejecución en la que se utilice ese broker y múltiples trabajadores donde se elimine algún trabajador.
- **c** Una ejecución en la que se utilice ese broker y múltiples clientes donde se elimine algún cliente.
- **d** Comparar la ejecución de un broker básico ROUTER-ROUTER en la que se elimine un trabajador con la ejecución de un broker tolerante a fallos en la que suceda lo mismo.

10 *¿Qué mecanismo utiliza el broker tolerante a fallos para detectar esos fallos?*

- **a** Mensajes de latido periódicos.
- **b** Mensajes de latido, pero solo en caso de que los clientes o trabajadores lleven tiempo sin interactuar con el broker.
- **c** Establecer un plazo para la recepción de una respuesta.
- **d** Ninguna de las demás opciones es correcta.

TSR	Recup. Segundo Parcial	Examen resuelto	2020-01-27	A

20 cuestiones, todas con el mismo valor. 4 opciones cada una (sólo una correcta). Cada error resta 1/3 de un acierto. Debe contestar en la hoja de respuestas

1 *La propiedad* `identity` *en un objeto socket ZeroMQ es relevante cuando se trata de un socket de este tipo:*

- **a** ROUTER.
- **b** DEALER.
- **c** Los que interactúen con un socket ROUTER, sean del tipo que sean.
- **d** Los que interactúen con un socket DEALER, sean del tipo que sean.

2 *Un proceso que ejecuta un programa JavaScript pretende enviar este vector* `['xFz34aA78',"','myAnswer']` *a través de un socket ZeroMQ a otro proceso que también utiliza un socket ZeroMQ para efectuar la recepción. Si sabemos que el proceso receptor únicamente recibirá la cadena* `'myAnswer'` *como contenido de ese mensaje, ¿qué tipos de sockets están utilizando ambos procesos?*

- **a** PUSH en el emisor y PULL en el receptor.
- **b** DEALER en el emisor y ROUTER en el receptor.
- **c** ROUTER en el emisor y DEALER en el receptor.
- **d** ROUTER en el emisor y REQ en el receptor.

3 *Algunas tareas contenidas en el despliegue son:*

- **a** Desarrollo de software.
- **b** Diseño de software.
- **c** Análisis de software.
- **d** Actualización de software.

4 *En el Tema 4, un servicio es:*

- **a** Una aplicación distribuida que ha sido desplegada y permanece activa.
- **b** Un conjunto de scripts independientes con un plan de despliegue.
- **c** Una futura aplicación distribuida que todavía está en sus etapas de análisis o diseño.
- **d** Un programa Node.js que es ejecutado por un único usuario.

5 *Una de las tareas del despliegue es la adaptación. De ella forma parte esta actividad:*

- **a** Recuperación de un agente en caso de fallo.
- **b** Modificar la configuración de los agentes.
- **c** Escalado en función de los cambios en la carga.
- **d** Todas las opciones son correctas.

6 *¿Cuál de los siguientes elementos no resulta necesario para automatizar el despliegue de una aplicación?*

- **a** Descriptor de despliegue.
- **b** Mecanismo de resolución de dependencias.
- **c** Un logger como el utilizado en la práctica 3 para identificar rápidamente los problemas que haya.
- **d** Herramienta de gestión del despliegue.

7 *Una de las siguientes operaciones no la lleva a cabo la orden* `docker run`*. ¿Cuál es?*

a Procesar un Dockerfile para construir la imagen Docker correspondiente.

b Poner en ejecución un contenedor a partir de una imagen Docker existente en el depósito local.

c Descargar una imagen del Docker Hub.

d Dar valor a las variables de entorno que necesita un contenedor.

8 *Según el teorema CAP en un sistema distribuido:*

a Nunca pueden conseguirse simultáneamente disponibilidad y elasticidad.

b Nunca pueden conseguirse altas disponibilidades, elevados throughputs y bajas latencias simultáneamente.

c Nunca pueden conseguirse simultáneamente tolerancia al particionado y robustez.

d Nunca pueden conseguirse simultáneamente disponibilidad, tolerancia al particionado de la red y consistencia fuerte.

9 *¿Qué modelo de replicación puede necesitar una etapa de elección cuando falle alguna de sus réplicas?*

a Activo.

b Pasivo.

c Multi-máster.

d Todos.

10 *¿Qué modelo de consistencia es más fuerte que la consistencia eventual?*

a FIFO.

b Causal.

c Secuencial.

d Ninguno, pues la consistencia eventual no puede compararse con ningún otro modelo.

11 *Suponga un servicio replicado, inicialmente con cuatro réplicas. En cierto momento una de esas réplicas deja de funcionar, pero el servicio se sigue ofreciendo sin problemas. Esa situación es un ejemplo de:*

a Defecto.

b Error.

c Fallo.

d Ninguna de las demás opciones es correcta.

12 *En los almacenes NoSQL:*

a No se cumplen las garantías ACID.

b No se simplifica el esquema.

c No se eliminan las transacciones.

d No se potencia la escalabilidad.

13 *Con relación al módulo cluster de node.js es FALSO que:*

a Permita implementar escalabilidad horizontal.

b El máster puede conocer las propiedades intrínsecas de sus workers mediante el objeto cluster.workers.

c Cada worker puede conocer sus propiedades intrínsecas mediante el objeto cluster.worker.

d Dentro del clúster, de modo intrínseco, se encuentra implementado un medio mediante el cual puede comunicarse bidireccionalmente el máster con cada worker a través de mensajes con transporte tcp.

14 *La replicación multi-máster es generalmente más escalable que la replicación activa y pasiva porque:*

a No utiliza mecanismos de sincronización entre las réplicas.

b No soporta consistencias fuertes

c No necesita reconfigurarse cuando alguna réplica falla.

d Todas las opciones son correctas.

15 *Cuando se realiza un despliegue complejo de MongoDB, ¿qué función tienen los procesos* `mongos`*?*

a Interacción con la aplicación cliente.

b Equilibrado de carga si la base de datos está particionada.

c Encaminan las peticiones a la instancia `mongod` adecuada.

d Todas las opciones son correctas.

16 *El despliegue de un cliente o de un worker en nuestro sistema CBW necesita que ciertas dependencias sean resueltas. Para hacerlo automáticamente, si en el archivo docker-compose.yml aparece un fragmento como:*

```
bro:
    expose:
      - "9998"
      - "9999"
```

a El fragmento relativo al worker deberá incluir

```
wor:
   links:
      - bro
   environment:
      - BROKER_URL=tcp://bro:9999
```

… y el Dockerfile del worker deberá incluir `CMD node myworker $BROKER_URL`

b El fragmento relativo al cliente deberá incluir

```
cli:
   links:
      - wor
   environment:
      - WORKER_URL=tcp://wor:9998
```

… y el Dockerfile del cliente deberá incluir `CMD node myclient $WORKER_URL`

c El fragmento relativo al broker debe incluir además una sección `image: broker`

d El fragmento relativo al broker no puede incluir una sección `image: broker`

17 *En el archivo docker-compose.yml para el despliegue del servicio CBW_FTCL, en la sección referente al logger aparece un fragmento…*

```
volumes:
    - /tmp/logger.log:/tmp/cbwlog
```

a Especifica dos ficheros o directorios del anfitrión que el logger puede compartir con otros componentes del despliegue

b Especifica un conjunto de rutas de ficheros del logger que el anfitrión debe guardar en almacenamiento persistente

c Especifica una ruta (la primera) en el sistema de ficheros del logger a la que otros componentes del despliegue consiguen acceder indicando otra ruta (la segunda) propia

d Se asemeja a la orden ports porque el primer argumento se refiere al anfitrión y el segundo al contenedor

18 *Al final de la práctica 3 se dejan abiertas varias preguntas relacionadas con worcli. En una de ellas se menciona un parámetro de la invocación que se refiere a un retardo añadido. Selecciona la afirmación verdadera en este contexto, y su relación con el latido:*

a Ese retardo representa el tiempo invertido por el worker para procesar la petición del cliente

b Ese retardo establece una nueva duración para el latido por tratarse de una solicitud diferente

c El tiempo de servicio máximo visto por el bróker se debe calcular para superar la suma de ese parámetro y el tiempo de servicio en el worker final

d Ese retardo establece el tiempo tras el cual el cliente recibirá automáticamente (en caso de fallo) una notificación

19 *¿Cuántos Dockerfile debían modificarse para realizar el despliegue en el ejemplo* `1_A_MANO` *de la práctica 3?*

a Ninguno.

b Uno.

c Dos.

d Todos.

20 *¿Qué cambios introduce el ejemplo* `2_CBW` *sobre el ejemplo inicial* `1_A_MANO` *en la práctica 3?*

a Se añade un fichero `docker-compose.yml`.

b Se resuelven automáticamente las dependencias del despliegue.

c Se puede especificar cuántas instancias de los componentes `worker` y `client` iniciarán su ejecución.

d Todas las opciones son correctas.