Cada parcial vale 10 puntos y consta de 17 preguntas. Cada pregunta plantea 4 alternativas y tiene sólo una respuesta correcta. Una vez descartada la peor respuesta, cada respuesta correcta aporta 10/16 puntos, y cada error deduce 10/48 puntos. Debes responder en la hoja de respuestas.

Parcial 1

1 *Consideremos el siguiente par de programas Node.js que usan ØMQ:*

```
// Program: sender.js
const zmq = require("zeromq")
const producer = zmq.socket("push")
let count = 0
producer.bind("tcp://*:8888", (err) => {
  if (err) throw err
  setInterval(function() {
    producer.send("msg# " + count++)
  }, 1000)
})
```

```
// Program: receiver.js
const zmq = require("zeromq")
const consumer = zmq.socket("pull")
consumer.connect("tcp://127.0.0.1:8888")
consumer.on("message", function(msg) {
  console.log("received: " + msg)
})
```

Todas las instancias de esos programas se ejecutan **en el mismo ordenador**, *e intentamos iniciar tantos procesos de cada tipo como sea posible. ¿Cuál es el número máximo de procesos que se pueden iniciar sin generar ningún error?*

a Muchos emisores y un solo receptor.

b No hay límite.

c Un único emisor y muchos receptores.

d Un único emisor y un único receptor.

2 *Entre otras cosas, se ha elegido JavaScript como el lenguaje de programación analizado en el Tema 2 porque…*

a Admite programación asincrónica. La programación asincrónica proporciona una buena base para diseñar y desarrollar aplicaciones distribuidas escalables.

b Admite programación multihilo. El uso de varios hilos es una característica clave para escribir programas simples y garantizar la corrección de la aplicación.

c Wikipedia se ha implementado utilizando JavaScript y muchos de los ejemplos utilizados en el Tema 2 y en los laboratorios están inspirados en fragmentos de código de Wikipedia.

d Todas las opciones son verdaderas.

3 *En el Tema 1, hemos visto que los servicios de software (por ejemplo, aquellos implementados por empresas en centros internos con clusters de ordenadores) han migrado a la nube. Una razón importante para esa migración es…*

a Los servicios en la nube siguen un modelo de aplicación cooperativo que escala de manera trivial.

b Los clusters internos son propensos a fallos.

c Los clusters internos tienen muchas más vulnerabilidades de seguridad que los centros de datos administrados por proveedores de nube.

d Los clusters internos se vuelven costosos y difíciles de administrar, mientras que los servicios en la nube siguen un modelo de pago por uso y pueden proporcionar un entorno autoadministrado.

Consideremos el siguiente programa en las siguientes dos preguntas

```
// Program: example.js
const ev = require('events')
const emitter = new ev.EventEmitter()
const e1 = "print"
const f = (x) => {
   emitter.emit(e1, x);
   return (y) => {return y+x}
}
emitter.on(e1, function(y) {
  console.log(y+"!!")
})
setTimeout(f("First"), 2000)
emitter.emit(e1,"Second")
console.log("End.")
```

4 *Si el usuario o cualquier otro proceso no lo elimina, ¿cuándo finaliza 'example.js' su ejecución?*

- **a** Nunca termina ya que genera continuamente eventos periódicos.
- **b** Tan pronto como haya mostrado un error en la pantalla.
- **c** Aproximadamente, dos segundos después de mostrar su primer mensaje vía console.log en la pantalla.
- **d** Tan pronto como muestre su último mensaje vía console.log en la pantalla.

5 *¿Qué muestra 'example.js' en la pantalla cuando se ejecuta?*

- **a** First!!
Second!!
End.
- **b** End.
Second!!
First!!
- **c** Muestra un error ya que el primer argumento de setTimeout no es una función.
- **d** End.
First!!
Second!!

6 *Consideremos un par de programas cliente y servidor cuya comunicación se implementa utilizando el módulo 'net' de Node.js. Si varios procesos que ejecutan el programa cliente han enviado a la vez sus mensajes al servidor, ¿es posible que este último maneje esos mensajes sin usar su cola de eventos?*

- **a** No, porque la cola de eventos solo la usan las promesas y el módulo 'net' las usa.
- **b** Sí, ya que la cola de eventos no es necesaria para manejar mensajes 'net'.
- **c** No, porque ese módulo usa eventos para notificar cuándo se ha establecido una conexión o se ha entregado un mensaje.
- **d** Sí, ya que el programa servidor puede haber sido escrito usando promesas y en ese caso no se usa la cola de eventos.

7 *Wikipedia es un buen ejemplo de un servicio escalable que necesita muchos servidores. ¿Cómo se organiza ese servicio en componentes para proporcionar esa escalabilidad?*

- **a** Hay un único componente y ese componente ha sido replicado para garantizar la disponibilidad del servicio. Esa disponibilidad también garantiza la escalabilidad.
- **b** Hay un solo componente, por lo que cada solicitud es procesada por un único servidor, ya que esto reduce al mínimo el tiempo de respuesta.
- **c** Se definen varios componentes específicos que interactúan entre ellos. Esto, combinado con el uso de cachés, distribución de datos y equilibrado de carga, proporciona escalabilidad.
- **d** Está estructurado en múltiples componentes replicados. Los clientes pueden enviar sus peticiones al componente más cercano, y este último responde sin ninguna comunicación con otros componentes, minimizando así el tiempo de respuesta.

8 *El Tema 3 explica que ØMQ proporciona una comunicación débilmente persistente (es decir, el emisor mantiene los mensajes si el receptor aún no está listo). ¿Existe algún patrón de comunicación ØMQ que no pueda proporcionar esa persistencia débil de comunicación en sus canales?*

a Sí, el patrón PUB-SUB

b No. Todos los patrones son débilmente persistentes.

c Sí, el patrón REQ-REP.

d Sí, el patrón PUSH-PULL.

9 *Consideremos este programa JavaScript:*

```
function f2 () {
  for (let let_i=0; let_i<5; let_i++) {
    console.log ("let_i: " + let_i)
  }
  console.log ("end --> let_i=" + let_i)
}

f2 ("new value")
```

¿Cuál es la última línea que se muestra en la pantalla cuando ejecutamos ese programa?

a Un mensaje de error (por ejemplo, 'ReferenceError: let_i is not defined').

b new value

c end –> let_i=5

d let_i: 4

10 *Consideremos un programa cliente y otro servidor que usan un patrón de comunicación ØMQ REQ-REP. El cliente usa esta instrucción, so.send(['s1','s2']), para enviar una solicitud determinada al servidor. ¿Cuál es el contenido del mensaje realmente transmitido desde el socket REQ al socket REP?*

a Un solo segmento con el resultado de esta llamada: JSON.stringify(['s1','s2']).

b Dos segmentos: las cadenas 's1' y 's2'.

c Cuatro segmentos: la identidad del remitente, la cadena vacía, la cadena 's1' y la cadena 's2'.

d Tres segmentos: la cadena vacía, la cadena 's1' y la cadena 's2'.

11 *En la segunda sesión de la Práctica 1, implementamos un programa que generaba una serie infinita de etapas (implementadas dentro de la función 'stage') que tomaban un tiempo aleatorio (entre 2 y 5 segundos por etapa, con una duración generada por la función 'rand'). Si solo se necesitaran 6 etapas, en lugar de un número ilimitado de ellas, elija la afirmación correcta sobre la idoneidad del siguiente programa para resolver ese problema:*

```
t = 0
for (let i=0; i<N; i++) {
   t = t + rand();
   setTimeout(stage, t);
}
```

a El programa resuelve correctamente ese problema si el valor de N es 5.

b El programa no resuelve ese problema. Para solucionarlo, debe llamar a setInterval en lugar de setTimeout.

c El programa no puede manejar la tarea propuesta, ya que el segundo argumento de setTimeout debe derivarse del valor de 'i'.

d El programa resuelve correctamente ese problema si el valor de N es 6.

12 *Consideremos este programa JavaScript:*

```
function f1 (a,b,c) {
  console.log ("Receiving " + arguments.length + " arguments")
  return a+b+c;
}
let vector = [1,2,3,4]

console.log ("resultv1: " + f1 (...vector))
```

¿Qué se muestra en la pantalla cuando ejecutamos ese programa?

a Un mensaje de error.

b Receiving 4 arguments
resultv1: 6

c Receiving 1 arguments
resultv1: [1,2,3,4]undefinedundefined

d Receiving 3 arguments
resultv1: 6

13 *Hay dos procesos A y B que se ejecutan en la misma computadora y usan sockets ØMQ. La dirección IP de la computadora es 158.42.60.120. A ha utilizado con éxito esta declaración para iniciar una conexión con B: so.connect('tcp://158.42.60.120:8888'). ¿Qué instrucción debería usar B para establecer esa conexión AB?*

a so.connectSync('ipc://158.42.60.120:8888')

b Ninguna de las demás opciones es válida.

c so.connect('tcp://158.42.60.120:8888')

d so.connect('tcp://*:8888')

14 *En un servicio de chat necesitamos combinar dos patrones de comunicación unidireccionales simples. Llamémoslos A y B. A envía los mensajes del cliente al servidor, mientras que B envía mensajes del servidor a los clientes. Si se necesita minimizar el número de operaciones send() a utilizar, ¿qué patrón (o patrones) de comunicación ØMQ es el mejor para implementar A y B?*

a A: PUSH-PULL, B: PUB-SUB.

b Con un solo REQ-REP podemos implementar A y B simultáneamente.

c A: PUB-SUB, B: PUSH-PULL.

d Con un solo PUB-SUB podemos implementar A y B simultáneamente.

15 *¿Qué se muestra en la pantalla cuando ejecutamos este programa?*

```
function f(x) {
  return function (y) {
    x = x + y
    return x
  }
}
g = f(1)
process.stdout.write(g(2)+"")
process.stdout.write(g(3)+"")
```

a 23

b 36

c 34

d 12

16 *¿Cuál es la opción que describe el comportamiento del siguiente programa?*

```
var i = 1
do {
  let k = i
  setTimeout(()=>{console.log(k)},
      k*1000)
  i++
} while (i<11)
console.log("End of program. i: " + i)
```

a Ninguna de las demás opciones es válida.

b Escribe primero 'End of program. i: 11' y luego escribe los valores del 1 al 10, uno por línea y uno por segundo.

c Escribe primero 'End of program. i: 11' y luego escribe 11 diez veces, una por línea y una por segundo.

d Escribe los valores del 1 al 10, uno por línea y uno por segundo. Una vez terminado esto escribe inmediatamente 'End of program. i: 11'.

17 *¿Qué se muestra en la pantalla cuando ejecutamos este programa?*

```
function f(x) {
  return function (y) {
    x = x + y
    return x
  }
}
g = f(1)
h = f(1)
process.stdout.write(g(2)+"")
process.stdout.write(h(3)+"")
```

a 36

b 23

c 45

d 34

Parcial 2

18 *Esta es una característica que generalmente mejora la escalabilidad de un servicio distribuido:*

a Necesidad frecuente de coordinación entre procesos.

b Modelo de consistencia fuerte.

c Todas las opciones son verdaderas.

d Uso de proxies inversos.

19 *Este es el archivo docker-compose.yml que se utiliza en la primera sesión de la práctica 3 para desplegar un sistema CBW:*

```
version: '2'
services:
  cli:
    image: client
    build: ./client/
    links:
     - bro
    environment:
     - BROKER_HOST=bro
     - BROKER_PORT=9998
  wor:
    image: worker
    build: ./worker/
    links:
     - bro
    environment:
     - BROKER_HOST=bro
     - BROKER_PORT=9999
  bro:
    image: broker
    build: ./broker/
    expose:
     - "9998"
     - "9999"
```

¿Cuáles son los otros archivos donde también se usa la **variable de entorno** *BROKER_HOST?*

a Los Dockerfiles para client y worker.

b Los programas JavaScript client.js y worker.js.

c El programa JavaScript broker.js.

d Todas las opciones son correctas.

20 *Consideremos que este Dockerfile ha sido escrito para crear una imagen llamada 'broker2':*

```
FROM tsr-zmq
COPY ./tsr.js tsr.js
RUN mkdir broker
WORKDIR broker
COPY ./broker.js mybroker.js
EXPOSE 9998 9999
ENTRYPOINT ["/usr/bin/node","mybroker","9998","9999"]
```

¿Podemos crear 'broker2' usando alguna orden?

a Sí, con esta orden:

```
docker commit broker2
```

b Sí, ya que no hay ningún error en su contenido si los archivos mencionados existen y están en las ubicaciones previstas.

c No, porque un Dockerfile no puede usar ENTRYPOINT. En su lugar, debería usar CMD.

d Sí, con esta orden:

```
docker run broker2
```

21 *La primera sesión de la Práctica 3 comienza con* **un despliegue manual** *de un sistema broker-worker-client. En ese sistema, el broker se inicia primero y los otros dos componentes necesitan saber* **la dirección IP del contenedor del broker**. *En ese despliegue manual, ¿dónde se usa esa dirección IP?*

a Ninguna otra opción es correcta.

b En algunas de las instrucciones del archivo docker-compose.yml.

c En la instrucción CMD o ENTRYPOINT del Dockerfile de los componentes correspondientes.

d En algunos argumentos de la orden docker-compose up.

Consideremos estos programas JavaScript:

```
// Program: sender.js
const zmq = require( 'zeromq' )
const rq = zmq.socket( 'push' )
rq.connect( 'tcp://127.0.0.1:8888' )
rq.send( 'Hello' )
```

```
// Program: receiver.js
const zmq = require( 'zeromq' )
const rp = zmq.socket( 'pull' )
rp.bindSync( 'tcp://127.0.0.1:8888' )
rp.on('message', function( msg ) {
  console.log( 'Received: ' + msg )
})
```

Estos programas utilizan el patrón de comunicación PUSH-PULL. Planeamos cambiar ambos programas para que utilicen sockets DEALER y ROUTER, proporcionando la misma funcionalidad.

22 *Los cambios a aplicar en 'sender.js' son:*

a

```
const rq = zmq.socket( 'dealer' ) // Línea 3
rq.send( [ rq.identity, 'Hello' ] ) // Línea 5
```

b

```
const rq = zmq.socket( 'router' ) // Línea 3
```

c El cambio propuesto no tiene sentido. El patrón DEALER-ROUTER no puede gestionar ese tipo de comunicación.

d

```
const rq = zmq.socket( 'dealer' ) // Línea 3
```

23 *Los cambios a aplicar en 'receiver.js' son:*

a

```
const rp = zmq.socket( 'router' ) // Línea 3
```

b

```
const rp = zmq.socket( 'dealer' ) // Línea 3
```

c

```
const rp = zmq.socket( 'router' ) // Línea 3
rp.on('message', function( who, msg ) { // Línea 5
```

d El cambio propuesto no tiene sentido. El patrón DEALER-ROUTER no puede gestionar ese tipo de comunicación.

24 *Consideremos que este docker-compose.yml está en el directorio /home/user/docker:*

```
version: '2'
services:
  svca:
    image: imga
    links:
      - svcb
    environment:
      - B_HOST=svcb
  svcb:
    image: imgb
    links:
      - svcc
    environment:
      - C_HOST=svcc
    expose:
      - "9999"
  svcc:
    image: imgc
    expose:
      - "9999"
```

Por favor, elija la oración VERDADERA sobre el servicio que se implementará utilizando ese archivo.

a Al usar ese docker-compose.yml, no podemos usar ningún Dockerfile para crear las imágenes que faltan si alguna de ellas no existe cuando ejecutamos

```
docker-compose up
```

b Los componentes de ese servicio se iniciarán en este orden: svca, svcb, svcc.

c Ese servicio se puede desplegar, pero sus componentes svcc y svcb escucharán el mismo puerto en el anfitrión (9999) al mismo tiempo, generando un conflicto.

d Podemos desplegar dos instancias de cada componente usando esta orden en /home/user/docker:

```
docker-compose up -d --scale svc=2
```

Supongamos que la imagen 'tsr-zmq' existe y tiene el contenido y la funcionalidad explicados en el Tema 4 y la Práctica 3. Supongamos también que este Dockerfile (al que nos referiremos como 'Dockerfile A', aunque su nombre real es 'Dockerfile') se ha guardado en el directorio /home/user/docker/config:

```
FROM tsr-zmq
COPY ./tsr.js tsr.js
RUN mkdir broker
WORKDIR broker
COPY ./broker.js mybroker.js
EXPOSE 9998 9999
CMD node mybroker 9998 9999
```

25 *Consideremos Dockerfile A. ¿Por qué utiliza la instrucción 'EXPOSE 9998 9999'?*

- **a** Ninguna otra opción es correcta.
- **b** Para asignar el puerto 9999 en el contenedor al puerto 9998 en el anfitrión.
- **c** Para asignar el puerto 9999 en el anfitrión al puerto 9998 en el contenedor.
- **d** Para advertir que el proceso a ejecutar en el contenedor atenderá conexiones en sus puertos 9998 y 9999.

26 *Consideremos Dockerfile A. ¿Por qué utiliza sus instrucciones RUN y WORKDIR?*

- **a** Ninguna otra opción es correcta.
- **b** Para garantizar que el programa **mybroker** se ejecute en el directorio raíz de los contenedores generados.
- **c** Para colocar tsr.js en el directorio padre de **mybroker.js** ya que esa es la ubicación asumida en el código de este último programa.
- **d** Para garantizar que **broker.js** (en el anfitrión) y **mybroker.js** (en la imagen) estén ubicados en el mismo lugar en sus respectivas computadoras.

27 *Consideremos Dockerfile A. ¿Qué archivos se necesita para crear una imagen usando ese Dockerfile?*

- **a** tsr.js y mybroker.js, ambos ubicados en el mismo directorio que ese Dockerfile.
- **b** tsr.js y broker.js, ambos ubicados en el mismo directorio que ese Dockerfile.
- **c** tsr-zmq
- **d** tsr.js, client.js, broker.js y worker.js

28 *¿Cuál de estas alternativas es una diferencia correcta entre los modelos de replicación multi-master y activo?*

- **a** Ninguna de las demás opciones es correcta.
- **b** El modelo multi-master puede usar una réplica de procesamiento diferente (es decir, el master) por cliente, mientras que el modelo activo siempre usa todas las réplicas.
- **c** En el modelo activo, cada solicitud solo se envía a la réplica principal, mientras que en el modelo multi-master, el cliente transmite cada solicitud a cada réplica.
- **d** El modelo multi-master puede gestionar el modelo de fallos arbitrarios, mientras que el modelo activo no puede.

29 *Si comparamos la replicación activa y pasiva, el modelo activo es el modelo de replicación preferido cuando las operaciones modifican gran parte del estado del servicio porque…:*

- **a** Ninguna otra opción es verdadera.
- **b** El modelo pasivo debe enviar esas modificaciones a las réplicas secundarias y estas últimas deben aplicarlas, mientras que en el modelo activo no se necesita transferir las modificaciones.
- **c** Cuando esas modificaciones son pequeñas, el modelo de replicación activo intercambia esas modificaciones entre todas las réplicas.
- **d** Cuando las modificaciones son grandes, el modelo pasivo debe generar muchos hilos de ejecución para aplicarlas en la réplica primaria, y esto bloquea la ejecución en la réplica principal.

30 *Este es el archivo docker-compose.yml utilizado en la última sesión de la Práctica 3:*

```
version: '2'
services:
  mariadb:
    image: docker.io/bitnami/mariadb:11.1
    volumes:
      - 'mariadb_data:/bitnami/mariadb'
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
      - MARIADB_USER=bn_wordpress
      - MARIADB_DATABASE=bitnami_wordpress
  wordpress:
    image: docker.io/bitnami/wordpress:6
    ports:
      - '80:8080'
      - '443:8443'
    volumes:
      - 'wordpress_data:/bitnami/wordpress'
    depends_on:
      - mariadb
    environment:
      - ALLOW_EMPTY_PASSWORD=yes
      - WORDPRESS_DATABASE_HOST=mariadb
      - WORDPRESS_DATABASE_PORT_NUMBER=3306
      - WORDPRESS_DATABASE_USER=bn_wordpress
     - WORDPRESS_DATABASE_NAME=bitnami_wordpress
volumes:
  mariadb_data:
    driver: local
  wordpress_data:
    driver: local
```

¿Necesitamos utilizar algún Dockerfile para desplegar este servicio?

a No, las imágenes necesarias se descargan automáticamente desde el repositorio remoto correcto cuando ejecutamos la orden **docker-compose up**.

b Sí, necesitamos descargar el Dockerfile de MariaDB y WordPress del sitio Bitnami para crear manualmente sus imágenes Docker correspondientes.

c Sí, necesitamos descargar desde hub.docker.com el Dockerfile de MariaDB y WordPress para crear manualmente sus imágenes Docker correspondientes.

d No, ya que las imágenes 'mariadb' y 'wordpress' están en el repositorio local de nuestra computadora de forma predeterminada cuando instalamos Docker.

31 *Si comparamos los modelos de consistencia causal y secuencial, ¿cuál de ellos es más relajado que el otro?*

a Su grado de relajación no se puede comparar.

b Tienen grados de relajación equivalentes.

c Secuencial

d Causal

32 *Entre otras cosas, para ejecutar un contenedor Docker en un ordenador anfitrión determinado, necesitamos…*

a Que el SO anfitrión no sea Windows 11.

b Un sistema operativo (SO) anfitrión como el que se supone en la imagen a ejecutar.

c La orden **docker commit** para iniciar el contenedor.

d Una ejecución previa del comando **docker images** para crear la imagen deseada.

33 *Si consideramos el teorema CAP, ¿cuál de estos modelos de consistencia puede respetarse cuando se necesita disponibilidad en un sistema particionado?*

a Cualquier modelo lento

b Estricto

c Ninguna de las demás opciones es correcta

d Secuencial

34 *La segunda sesión de la Práctica 3 propone el despliegue de otro tipo de cliente (un cliente externo) que se ejecutará en otro ordenador, diferente al anfitrión donde docker y docker-compose administran los contenedores CBW. ¿A qué dirección IP se conecta este cliente externo?*

a La del anfitrión donde se ejecuta el contenedor del cliente.

b La del contenedor del broker.

c La del contenedor del worker.

d La del anfitrión donde se ejecuta el contenedor del broker.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

D

DNI NIE PASAPORTE

0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9

ETSInf - TSR

Recuperación - 08/02/2024

Apellidos .....

Nombre .....

Marque así Así NO marque

NO BORRAR, corregir con corrector

Primer Parcial

1 a b c d
2 a b c d
3 a b c d
4 a b c d
5 a b c d
6 a b c d
7 a b c d
8 a b c d
9 a b c d
10 a b c d
11 a b c d
12 a b c d
13 a b c d
14 a b c d
15 a b c d
16 a b c d
17 a b c d

Segundo Parcial

1 a b c d
2 a b c d
3 a b c d
4 a b c d
5 a b c d
6 a b c d
7 a b c d
8 a b c d
9 a b c d
10 a b c d
11 a b c d
12 a b c d
13 a b c d
14 a b c d
15 a b c d
16 a b c d
17 a b c d