# TSR - Segundo Parcial. 2025-01-13

Este examen consta de 22 cuestiones, con una puntuación total de 10 puntos. Cada cuestión tiene 4 alternativas, de las cuales debe elegirse una sola opción. La nota se calcula de la siguiente forma: tras descartar las dos peores cuestiones, cada acierto suma 0.5 puntos y cada error descuenta 1/6 puntos. Debes contestar en la hoja de respuestas.

C

1. ***¿Tiene sentido que en un Dockerfile no haya ninguna línea `CMD` ni `ENTRYPOINT`?***
   - **A.** Sí, pues la instrucción `WORKDIR` tiene un objetivo similar y puede sustituirlas.
   - **B.** Sí, pues la imagen a generar podría utilizarse como base para otros Dockerfile que sí utilicen `CMD` o `ENTRYPOINT`.
   - **C.** No, pues si esas instrucciones no están, las instrucciones `RUN` no tienen ningún efecto.
   - **D.** No, pues en ese caso la orden `docker build` produce un error y no construye ninguna imagen.

2. ***Considérese el siguiente programa JavaScript, utilizado para ejecutar un proceso servidor `S` con el que interactuará algún proceso cliente `C`:***

```
const z = require("zeromq")
const r = z.socket("router")
r.bindSync("tcp://127.0.0.1:8999")
r.on("message", (x, y, z) => {
  console.log(x+" "+y+" "+z)
  r.send([x, parseInt(y+"")*parseInt(z+"")])
})
```

   **Seleccione la afirmación correcta:**
   - **A.** El parámetro `x` definido en la cuarta línea de ese programa recibirá la identidad del socket utilizado en `C` para conectarse con `r`.
   - **B.** `C` ha podido utilizar un socket de tipo PUSH para recibir las respuestas de `S`.
   - **C.** `C` ha podido utilizar dos sockets (p.ej., de tipo PUSH y PULL) para enviar las peticiones y recibir las respuestas de `S`.
   - **D.** No puede haber más de un proceso cliente `C` conectado con `S`, pues `S` solo ha definido un listener para el evento `message`.

3. ***Considérese el siguiente programa JavaScript:***

```
const z = require("zeromq")
const r = z.socket("router")
r.bindSync("tcp://127.0.0.1:8999")
r.on("message", (x, y, z) => {
  console.log(x+" "+y+" "+z)
  r.send([x, parseInt(y+"")*parseInt(z+"")])
})
```

   **Indique qué afirmación describe el comportamiento de este programa**
   - **A.** Si la quinta línea muestra la cadena `C1 4 5` al recibir un mensaje, entonces quien emitió ese mensaje pudo haber utilizado un `s.send("C1 4 5")` para ello.
   - **B.** Ninguna de las demás afirmaciones es correcta.
   - **C.** Si la quinta línea muestra la cadena `C1 4 5` al recibir un mensaje, entonces quien emitió ese mensaje pudo haber utilizado un `s.send([4,5])` para ello.
   - **D.** Si la quinta línea muestra la cadena `C1 4 5` al recibir un mensaje, entonces quien emitió ese mensaje pudo haber utilizado un `s.send(["C1",4,5])` para ello.

4. ***Al enviar mensajes mediante sockets de tipo DEALER...:***
   - **A.** El socket siempre añade automáticamente un primer segmento adicional con la identidad del socket emisor.
   - **B.** Todas las demás afirmaciones son falsas.
   - **C.** Se sigue una política de envío de turno circular.
   - **D.** Se puede elegir, mediante el primer segmento del mensaje, a qué otro socket conectado con el DEALER se está enviando el mensaje.

***(Las preguntas 5 a 11 se refieren a esta misma descripción) Se ha diseñado un sistema con tres componentes (A,B,C) cuyo código se muestra a continuación, y que se conectan según indica la figura. Para no perder en ningún caso el mensaje publicado por A, se ha introducido un retardo antes del envío: asumimos que siempre se arrancan todas las instancias dentro de ese intervalo.***

**A.js**

```
const zmq = require('zeromq')
let sin = zmq.socket('pull')
let sout = zmq.socket('pub')
sin.bind('tcp://*:1111')
sout.bind('tcp://*:2222')
sin.on('message', msg =>
  console.log('Recibido '+msg)
)
setTimeout(() => {
  console.log('empezamos')
  sout.send('hola')
}, 5000)
```

**B.js**

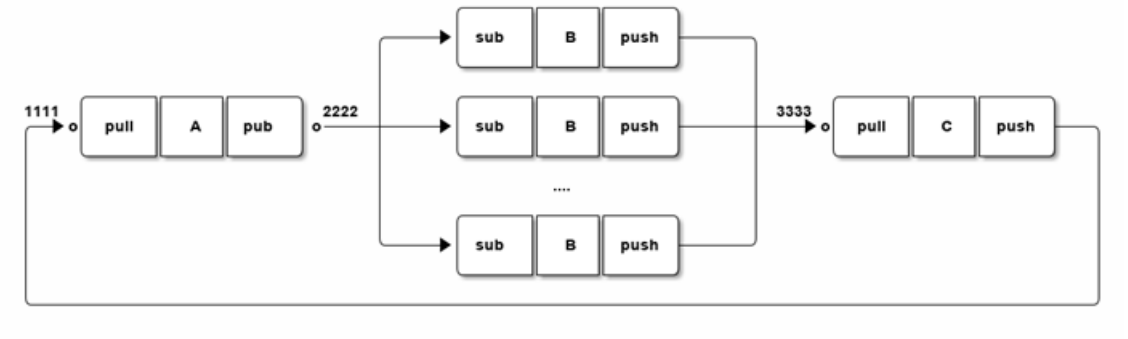
```
const zmq = require('zeromq')
let sin = zmq.socket('sub')
let sout = zmq.socket('push')
sin.connect(process.argv[2])
sout.connect(process.argv[3])
sin.subscribe('')

sin.on('message', msg => {
  console.log('recibido '+msg)
  sout.send(msg)
})
```

**C.js**

```
const zmq = require('zeromq')
let sin = zmq.socket('pull')
let sout = zmq.socket('push')
sin.bind('tcp://*:3333')
sout.connect(process.argv[2])

sin.on('message', msg => {
  console.log('recibido '+msg)
  sout.send(msg)
})
```

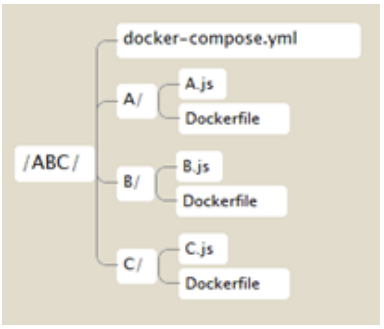



Para realizar las pruebas se lanzan varios terminales para las distintas instancias. Ejemplo:

```
node A.js
node B.js tcp://localhost:2222 tcp://localhost:3333
node B.js tcp://localhost:2222 tcp://localhost:3333
node B.js tcp://localhost:2222 tcp://localhost:3333
node C.js tcp://localhost:1111
```

Observamos que el resultado es correcto independientemente de la cantidad de instancias de B, y del orden en que se arranquen las instancias. Queremos automatizar el despliegue usando docker (todas las instancias en contenedores en el mismo anfitrión):

- Asumimos creada la imagen `tsr-zmq` (base para las imágenes de A, B y C)
- En el caso del componente C, asumimos que el puerto de su socket PULL es el 3333, y recibe por línea de órdenes la URL del socket PULL de A
- El código de B recibe por línea de órdenes las URL del socket PUB de A y del socket PULL de C




Se ha organizado el código siguiendo la estructura de directorios que se muestra a la derecha.
Responde a las cuestiones 5 a 11 relativas al despliegue usando docker

5. ***Deseamos lanzar una ejecución con 1 instancia de A, 5 instancias de B, y una instancia de C. Para ello debemos ejecutar la orden:***
   - **A.** `docker compose up --scale B=5`
   - **B.** `docker compose up`
   - **C.** `docker run --scale B=5`
   - **D.** `docker build B 5`

6. ***El fichero `/ABC/B/Dockerfile` debe contener la siguiente línea:***
   - **A.** `CMD node $URL_A $URL_B`
   - **B.** Ninguna de las restantes respuestas es correcta
   - **C.** `CMD node B.js`
   - **D.** `CMD node B.js tcp://localhost:1111 tcp://localhost:3333`

7. ***Si estamos situados en el directorio `/ABC/`, indica cuál de las siguientes órdenes no se ejecuta correctamente***
   - **A.** `docker images`
   - **B.** `docker ps`
   - **C.** `docker build -t a .`
   - **D.** `docker compose up`

8. ***En relación con el contenido del fichero `docker-compose.yml`***
   - **A.** En la sección correspondiente al componente A debe aparecer una sección `links`
   - **B.** En la sección correspondiente al componente C debe aparecer una sección `links`
   - **C.** En la sección correspondiente al componente A debe aparecer una sección `environment`
   - **D.** En la sección correspondiente al componente B debe aparecer una sección `expose`

9. ***El orden de despliegue de estos contenedores deberá ser:***
   - **A.** Primero C, luego A y después B
   - **B.** Primero A, el resto en cualquier orden
   - **C.** Primero C, el resto en cualquier orden
   - **D.** Primero A, luego C y después B

10. ***Suponiendo que hemos lanzado un contenedor que ejecuta una instancia de A, si queremos averiguar su IP ...***
    - **A.** Al ejecutar el contenedor mediante `docker run A` obtenemos como resultado la IP asociada al mismo
    - **B.** Debemos ejecutar `docker images` para averiguar la identidad de la imagen de A, seguido de `docker rmi identidad`
    - **C.** Es una información interna a la que no podemos acceder
    - **D.** Debemos ejecutar `docker ps` para averiguar la identidad de dicho contenedor seguido de `docker inspect identidad`

11. ***Sobre los Dockerfile respectivos...***
    - **A.** En el de B debe aparecer `EXPOSE 1111 2222`
    - **B.** En el de C debe aparecer `EXPOSE 1111 2222`
    - **C.** Todas las opciones son correctas
    - **D.** En el de A debe aparecer `EXPOSE 1111 2222`

12. ***Supongamos que añadimos un socket ROUTER asociado al puerto 3333 del componente A del esquema anterior, con los cambios adecuados en el código de A.js, y deseamos que sea accesible desde el exterior mediante el puerto 99 del anfitrión.***
    **Tras generar la nueva imagen, deberemos...**
    - **A.** No podemos desplegar conjuntamente el nuevo sistema porque necesitamos iniciar manualmente A con `docker run -p 99:3333 ...`
    - **B.** Modificar `docker-compose.yml` para colocar una entrada `ports` adecuada en el servicio A
    - **C.** Generar las imágenes del resto de componentes B y C
    - **D.** No funcionará porque el uso de un único puerto es incompatible con un socket de múltiples colas

13. ***Al considerar el Teorema CAP para un servicio escalable se suele recomendar la renuncia a una consistencia fuerte. En base a eso, seleccione qué modelo de consistencia no podría soportarse cuando haya problemas de conectividad que generen una partición en la red:***
    - **A.** FIFO
    - **B.** Causal
    - **C.** Secuencial
    - **D.** De manera general, cualquier modelo "rápido"

14. ***La orden `docker commit` puede utilizarse para:***
    - **A.** Iniciar la ejecución de un contenedor utilizando la imagen indicada en sus argumentos
    - **B.** Todas las demás opciones son incorrectas
    - **C.** Construir una imagen docker a partir de un fichero Dockerfile
    - **D.** Generar una imagen docker a partir del estado actual de un contenedor determinado

15. ***Para desplegar el sistema CBWL se utiliza un fichero `docker-compose.yml` en el que, dentro de la sección correspondiente al componente `bro`, aparecen estas líneas:***

```
environment:
  - LOGGER_HOST=log
  - LOGGER_PORT=9995
```

    **Seleccione la opción correcta sobre las implicaciones que tiene esa subsección `environment`:**
    - **A.** El componente `log` debe conectarse a un puerto, que no puede ser el `9995`, del componente `bro`
    - **B.** Los componentes `wor` y `cli` deben ser iniciados antes que `bro`, pues no se les llega a mencionar en esa subsección
    - **C.** En el Dockerfile de la imagen usada para generar el componente `bro`, se utilizan las variables de entorno `LOGGER_HOST` y `LOGGER_PORT`
    - **D.** El componente `bro` debe ser iniciado durante el despliegue antes que el componente `log`

16. ***Seleccione qué característica se respeta en la replicación multi-máster:***
   **A.** Todas las réplicas ejecutan la secuencia de instrucciones de cada operación solicitada por los procesos clientes
   **B.** Puede tolerar fallos arbitrarios
   **C.** Suele ser más eficiente que los modelos de replicación activo y pasivo
   **D.** Generalmente ofrece consistencia secuencial

17. ***Selecciona la afirmación correcta sobre la replicación pasiva:***
   **A.** Generalmente soporta fallos arbitrarios.
   **B.** Exige que cada solicitud de los clientes llegue a todas las réplicas servidoras antes de que alguna de ellas inicie el procesamiento de esa petición.
   **C.** Permite que las réplicas servidoras ejecuten localmente cada solicitud de los clientes sin necesidad de enviar las modificaciones generadas a las demás réplicas.
   **D.** Permite que una sola réplica ejecute la operación solicitada y difunda las modificaciones a las demás.

18. ***Suponga un sistema formado por tres procesos P1, P2 y P3, donde se ha dado la siguiente ejecución:*** `W1(x)2, W2(x)1, R1(x)1, R2(x)2, R3(x)2, R3(x)1`***. Esa ejecución respeta, entre otras, las consistencias:***
   **A.** FIFO y caché
   **B.** Solo la consistencia caché
   **C.** Estricta y secuencial
   **D.** FIFO y causal

19. ***El despliegue realizado en la primera sesión de la práctica 3 fue...***
   **A.** Un sistema de chat con patrones PUB/SUB y PUSH/PULL
   **B.** Un sistema client-broker-worker con doble broker
   **C.** Ninguna de las demás opciones es correcta
   **D.** Un sistema client-broker-worker con tolerancia a fallos del broker

20. ***Selecciona la afirmación correcta sobre el componente*** `logger` ***que añadimos al patrón CBW para generar el patrón CBWL:***
   **A.** Vuelca los mensajes que recibe a través del socket PULL a un fichero interno, accesible únicamente desde el propio contenedor que ejecuta la instancia del `logger`
   **B.** Difunde los mensajes que le llegan del broker al resto de componentes
   **C.** Añade cada mensaje en un fichero que forma parte de un volumen accesible desde la máquina anfitriona
   **D.** Mantiene en fichero únicamente el último mensaje recibido (cada mensaje reemplaza el contenido anterior)

21. ***Si una ejecución respeta la consistencia causal, entonces también respetará la consistencia...***
   **A.** Secuencial
   **B.** Estricta
   **C.** FIFO
   **D.** Caché

22. ***La imagen*** `tsr-zmq`***, mencionada a lo largo de la práctica 3, se debe construir mediante el Dockerfile proporcionado en el propio enunciado de la práctica. Indica cuál de las siguientes afirmaciones sobre ese Dockerfile es cierta.***
   **A.** Añade soporte para NodeJS
   **B.** Todas las respuestas son correctas
   **C.** No incluye la biblioteca `tsr.js` en la imagen producida, pero sí que incorpora la de ZeroMQ
   **D.** Se basa en una imagen Ubuntu