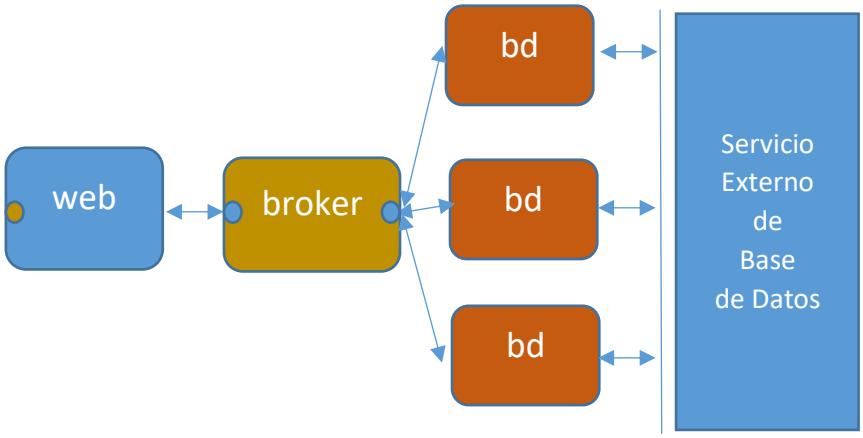
TSR – Práctica 3 (con soluciones)

Este examen consta de 6 cuestiones de opción múltiple. En cada una, solo una respuesta es correcta. Debe responderse en otra hoja. Las respuestas correctas aportan 1.67 puntos a la calificación del examen. Las erróneas descuentan 0.56 puntos.

1.- Disponemos de un servicio multicomponente y pretendemos desplegarlo utilizando las herramientas de contenerización de Docker. El servicio consta de 3 componentes:

- **web:** recibe las peticiones por parte de los usuarios, a través de su anfitrión, y las envía a **broker**.
- **broker:** gestiona las peticiones y las reparte entre los diferentes componentes **bd** reenviando las respuestas a **web**.
- **bd:** accede a la información requerida por parte de los usuarios desde un servicio externo y devuelve al resultado a **broker**.

Deseamos realizar un despliegue de este servicio como el representado en la siguiente figura:



El componente **web** escucha en el puerto **8000**, accesible a través del puerto 80 de su anfitrión, mientras que el componente **broker** escucha en el puerto **8000** con su socket *frontend*, y en el puerto **8001** con su socket *backend*.

Indica cuál de los siguientes fragmentos del **docker-compose.yml** (versión 2) de este servicio es correcto:

A

```
web:
  …
  ports:
   -"80:8000"
  …
bd:
  …
```

```
…
broker:
  …
  expose:
   -"8000"
   -"8001"
```

B

```
web:
  …
  expose:
   -"80"
bd:
  …
```

```
…
broker:
  …
  ports:
   -"8000"
   -"8001"
```

C

```
web:
  …
  expose:
   -"80:8000"
bd:
  …
```

```
…
broker:
  …
  expose:
   -"8000"
   -"8001"
```

D

El servicio no puede lanzarse porque hay un conflicto de puertos entre el componente **web** y el componente **broker**, ya que ambos utilizan el puerto **8000.**

Referencia para la solución (A): apartado 11.2 del boletín de esta práctica, páginas 40 y 41 de la Guía del Alumno del Seminario 4

TSR – Práctica 3 (con soluciones)

2.- Supongamos el ejemplo de la pregunta anterior. Seguidamente se muestran los dockerfiles de los componentes **web**, **bd** y **broker** respectivamente (ambos utilizan node y zmq, por lo que se pueden construir a partir de la imagen **fedora-node-devel** vista en la práctica 3)**:**

```
FROM fedora-node-devel
COPY ./web.js  web.js
CMD node web $URL_BRO
```

```
FROM fedora-node-devel
COPY ./bd.js  bd.js
CMD node bd $URL_BRO
```

```
FROM fedora-node-devel
COPY ./broker.js  broker.js
CMD node broker 8000 8001
```

Si pretendemos realizar una correcta inyección de dependencias entre los componentes, indica cuál de los siguientes fragmentos del **docker-compose.yml** (versión 2) asociado al servicio, es correcto:

A

```
web:
  …
  links:
   - broker
  environment:
   - URL_BRO=tcp://broker:8000
```

```
bd:
  …
  links:
   - broker
  environment:
   - URL_BRO=tcp://broker:8001
  …
broker:
  …
```

B

```
web:
  …
  links:
   - URL_BRO=tcp://broker:8000
```

```
bd:
  …
  links:
   - URL_BRO=tcp://broker:8001
  …
broker:
  …
```

C

```
web:
  …
  links:
   - broker
  environment:
   - URL_BRO=tcp://bro:8000
```

```
bd:
  …
  links:
   - broker
  environment:
   - URL_BRO=tcp://bro:8001
  …
broker:
  …
```

D

```
web:
  …
  environment:
   - URL_BRO=tcp://broker:8000
bd:
  …
  environment:
   - URL_BRO=tcp://broker:8001
  …
```

```
broker:
  …
  links:
   - web
   - bd
```

Referencia para la solución (A): docker-compose.yml del apartado 2 del boletín de esta práctica, ejercicio resuelto del apartado 11.2 de la Guía del Alumno del Seminario 4

TSR – Práctica 3 (con soluciones)

3.- Continuamos con el mismo ejemplo de las preguntas anteriores, donde cada directorio de componente incluye su código y su Dockerfile. Supongamos que…

- el directorio del servicio (donde se haya ubicado el **docker-compose.yml**) es **"/home/service_web"**
- el directorio del componente **web** es **"/home/service_web/web"**
- el directorio del componente **bd** es **"/home/service_web/bd"**
- y el directorio del componente **broker** es **"/home/service_web/broker"**.

Supongamos que ya está creada la imagen **fedora-node-devel**. Indica cuál de los siguientes fragmentos del **docker-compose.yml** (junto con las anotaciones que los acompañan) es correcto si pretendemos lanzar el servicio con el comando **"docker-compose up":**

A	`web:`
	`  build: ./web`
	`  …`
	`bd:`
	`  image: bd`
	`  …`
	`broker:`
	`  build: ./broker`
	`  …`
		Antes de ejecutar el comando "**docker-compose up"** hemos de ejecutar el comando **"docker build –t bd ."** en el directorio donde está el fichero **docker-compose.yml.**

B	`web:`
	`  image: ./web`
	`  …`
	`bd:`
	`  image: ./bd`
	`  …`
	`broker:`
	`  image: ./broker`
	`  …`	

C	`web:`
	`  image: web`
	`  …`
	`bd:`
	`  build: ./bd`
	`  …`
	`broker:`
	`  build: ./broker`
	`  …`
		Antes de ejecutar el comando "**docker-compose up"** hemos de ejecutar el comando **"docker build –t web ./web"** en el directorio donde está el fichero **docker-compose.yml.**

D	`web:`
	`  build: ./web`
	`  …`
	`bd:`
	`  build: ./bd`
	`  …`
	`broker:`
	`  image: ./broker`
	`  …`
		Antes de ejecutar el comando "**docker-compose up"** hemos de ejecutar el comando **"docker build –t broker ./broker"** en el directorio donde está el fichero **docker-compose.yml.**

Referencia para la solución (C): apartados 10.2 y 10.3 de la Guía del Alumno del Seminario 4 (directivas image y build)

TSR – Práctica 3 (con soluciones)

4.- La inyección de dependencias en esta práctica:

A	Se realiza a través de ficheros de configuración copiados en las imágenes generadas.
B	Se realiza mediante variables de entorno.
C	Se realiza copiando ficheros JavaScript en cada componente.
D	En los ejemplos vistos no hay inyección de dependencias.

Referencia para la solución (B): segunda página del apartado 2 del boletín de esta práctica

5.- Dentro de las tareas del despliegue visto en las prácticas, señala la afirmación CIERTA:

A	Es conveniente que cada componente contenga los elementos imprescindibles para su funcionamiento, pudiendo desechar aquellos elementos únicamente necesarios para el desarrollo.
B	Es necesario que cada componente contenga todos los elementos utilizados para el desarrollo del componente.
C	Uno de los pasos necesarios al desplegar componentes es copiar todos los elementos de desarrollo desde una imagen usada para el desarrollo a la imagen que se utilizará como base para el despliegue.
D	Es necesario eliminar todos los elementos usados en el desarrollo de los componentes usando para ello la utilidad docker-compose.

Referencia para la solución (A): apartado 2.1 del boletín de esta práctica

6.- Cuando en el bloque 3 de la práctica, se solicita reestructurar el código del servidor (miniWebServer) en dos componentes (señalar la afirmación CIERTA):

A	No se puede usar ZeroMQ para comunicar dichos componentes debido a que ningún patrón se adapta a las necesidades concretas del ejemplo.
B	Es necesario que en la parte donde se reciben las peticiones se mantenga el contexto de cada petición HTTP hasta obtener la respuesta por parte del generador de imágenes.
C	Ante cada petición, la petición al generador de imágenes se hace de forma sincrónica.
D	El contexto de la petición HTTP debe ser pasado al generador de imágenes para poder generar su respuesta.

Referencia para la solución (B): segunda página del apartado 3.4 del boletín de esta práctica