[MDAS] - Principios y herramientas de desarrollo
Entregable 1 - Docker (v1.0, 07/12/25)
Rylan James Graham

1. [3 ptos] Indica para cada una de las siguientes afirmaciones si es verdadera o falsa.

A. [0,3 ptos] Podemos usar sin riesgo ADD para construir imágenes por su mayor versatilidad.
Falso

B. [0,3 ptos] Es importante organizar las capas de nuestras imágenes para reducir re-construcciones innecesarias.
Verdadero

C. [0,3 ptos] Para arrancar un contenedor debemos indicar siempre el `tag` que queremos usar.
Falso

D. [0,3 ptos] Al construir una imagen OCI obtenemos un único fichero que contiene todo lo que necesitamos.
Falso

E. [0,3 ptos] Para comunicar 2 contenedores sin filtrar tráfico, deberemos configurar una red de tipo `bridge`.
Verdadero

F. [0,3 ptos] Los procesos en los contenedores son visibles desde cualquier sistema operativo.
Falso

G. [0,3 ptos] Una de las ventajas de Docker es que nos aísla de la máquina física mediante una capa de virtualización.
Falso

H. [0,3 ptos] Una de las ventajas de usar `bind mounts` es poder hacer shadowing the carpetas.
Verdadero

I. [0,3 ptos] En caso de problemas con un contenedor tendremos que reiniciarlo manualmente.
Falso

J. [0,3 ptos] Un contenedor puede contener cualquier tipo de aplicación, incluyendo también servicios de corta duración.
Verdadero

2. [3 ptos] El equipo de Ops nos pide que preparemos la imagen para nuestra aplicación. Nuestro código está organizado de la siguiente forma:

- **`/src` contiene el código fuente.**
- **`/test` contiene tests.**
- **`runtime.config` en la raíz, es un fichero de configuración necesario sólo para ejecutar la aplicación. Nos indican desde Ops que este fichero cambia con mucha frecuencia.**
- **`run.sh` es un script que se encarga de arrancar la aplicación asumiendo que el binario y la configuration se encuentran en la misma ruta.**
- **Para la construcción usamos la herramienta `its_magick` que ofrece los siguientes comandos:**
  - **`its_magick build` en la raíz del proyecto compila las fuentes y genera un ejecutable Linux llamado `run` en la ruta `/target`.**
  - **`its_magick test` en la raíz del proyecto lanza los tests y genera un report bajo `/target/tests`.**
- **Existe una imágen pública en Docker Hub para usar `its_magick` llamada "magick", la última versión es la 0.3.0**

A. [0,5 ptos] Escribe un Dockerfile para construir la imágen teniendo en cuenta los puntos del enunciado (rutas, comandos, etc.).

```
#Build Image
FROM magick:0.3.0 AS builder

WORKDIR /src
COPY src ./src
COPY test ./test

RUN its_magick test
RUN its_magick build

#Runtime Image
FROM alpine:3.22 AS runtime

WORKDIR /app
COPY --from=builder /target/run ./run
COPY run.sh ./run.sh
COPY runtime.config ./runtime.config

ENTRYPOINT ["./run.sh"]

```

B. [0,5 ptos] ¿Que imagen o imágenes base has usado como base? Explica brevemente los motivos (límite 50 palabras).

Usé magick:0.3.0 para compilar y ejecutar los tests porque incluye la herramienta its_magick. Para producción usé alpine:3.22 por ser ligera, segura y estable, evitando latest para garantizar reproducibilidad.

C. [0,4 ptos] ¿En qué momento has ejecutado los tests y qué has hecho con el report?

Ejecuté los tests en la fase builder. El report se generó en /target/tests pero no lo copié a la imagen final, porque solo sirve en CI/CD y no en runtime.

D. [0,4 ptos] ¿En que directorio has guardado los ficheros para ejecutar la app en la imagen? Emplica brevemente los motivos (límite 30 palabras).

Los guardé en /app porque es un directorio limpio y dedicado a la aplicación, evitando mezclar archivos con el sistema base y facilitando la organización y el mantenimiento.

E. [0,4 ptos] ¿Cuantas capas tiene la imágen construida? (límite 30 palabras).

La imagen final tiene 4 capas: la base de Alpine y las tres instrucciones COPY. WORKDIR y ENTRYPOINT no generan capas.

F. [0,4 ptos] Nos facilitan una nueva versión de `run.sh` con mejoras ¿Qué capas será necesario reconstruir y por qué? (límite 30 palabras).

Solo se reconstruye la capa donde se copia run.sh y las posteriores. No afecta a la fase de build porque run.sh en runtime está en una etapa distinta.

G. [0,4 ptos] Entre las mejoras de `run.sh` está la de poder sobrescribir parte de la configuración de `runtime.config` mediante variables de entorno. En concreto, podremos indicar el usuario y password de una conexión externa con *DB_USER* y *DB_PASSWORD* ¿Cuál es el comando que el equipo de Ops deberá usar para ejecutar la imagen?

docker run -e DB_USER=usuario -e DB_PASSWORD=secreto

3. [4 ptos] Parece que uno de nuestros servicios Tomcat no responde, en concreto el contenedor con id `2ae389ba767d`. Indica los comandos usados para responder a las preguntas.

A. [0,8 ptos] ¿Cómo podemos ver si está en ejecución actualmente? Indica exactamente que nos indica el estado en el que está.

El comando docker ps -a muestra el contenedor y su columna STATUS, que indica si está "Up" (en ejecución), "Exited" (parado) o "Restarting".

B. [0,8 ptos] Vemos que efectivamente no está en marcha ¿Cómo podemos obtener el

standard output hasta ahora?

docker logs 2ae389ba767d

C. [0,8 ptos] Intentamos arrancarlo de nuevo y vemos que no hace nada, simplemente termina. ¿Cómo podemos saber que "command" usa? (límite 30 palabras)

Con docker inspect 2ae389ba767d, revisando las claves Config.Cmd y Config.Entrypoint, vemos qué comando ejecuta el contenedor al arrancar.

D. [0,8 ptos] Finalmente, tras varios re-intentos con "docker run", parece que el servicio está en marcha. ¿Como podemos acceder directamente al contenedor para validar desde dentro que todo está correcto?

docker exec -it 2ae389ba767d bash

E. [0,8 ptos] ¿Debemos realizar alguna limpieza? En caso afirmativo, explica los comandos y por qué es necesario (límite 50 palabras).

Sí. Podemos limpiar contenedores parados con docker rm 2ae389ba767d si ya no es necesario. Para liberar espacio, docker image prune elimina imágenes sin uso y docker container prune borra contenedores detenidos.

Fe de erratas

Versión	*Fecha*	*Cambio(s)*