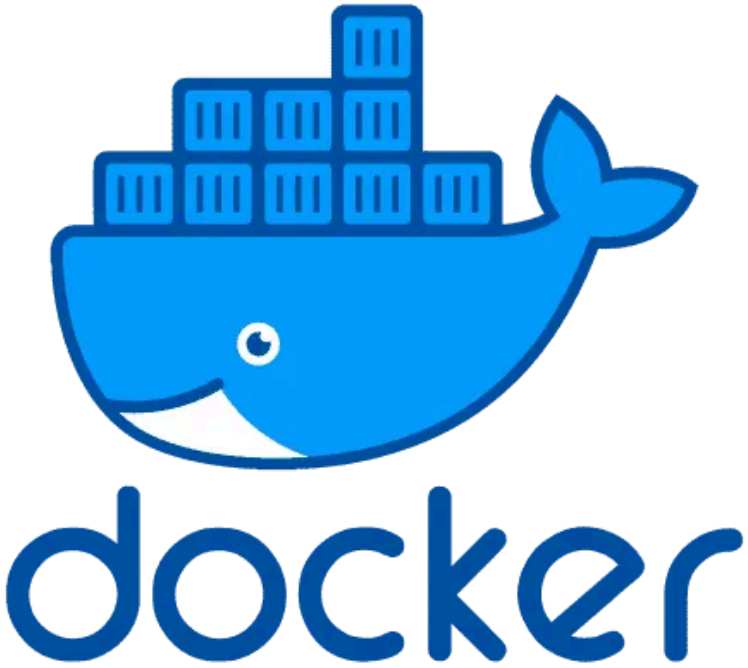


Para realizar esta práctica necesitas descargar la carpeta de la plataforma o de Júpiter. En ella encontrarás varios archivos base para llevar a cabo la práctica. La práctica consta de **tres niveles de dificultad**, el **nivel 1** abarca de los **puntos 1 a 4**, el **nivel 2** de los **puntos 5 a 6** y el **nivel 3** de los **puntos 8 a 10**. Vuestra nota será el reflejo de los puntos que hayáis logrado alcanzar y el nivel que hayáis podido completar. En este guion los puntos serán explicados individualmente, **será trabajo vuestro el unirlos todos en el mismo proyecto**, también será parte de vuestro trabajo el **añadir una web más compleja** a la vista en los ejemplos, y lo más importante tendréis que **explicar por qué añadir la funcionalidad vista en cada punto** a vuestro proyecto. Detallando los **beneficios o ventajas** que otorga, por último debéis crear un esquema de la arquitectura que estáis creando, solo será necesario entregar la versión final del esquema.

1. Despliegue de Docker (1.5 puntos)

Archivos necesarios:

Como comandos →

- **Docker permite descargar imagenes, crear contenedor con dicha imagen.**
- **Docker + Compose nos permite controlar un conjunto de contenedores cada uno con su imagen.**

[

Dokerfile: Es un archivo que sirve para configurar un servicio “en especifico”. **Le dice a DOCKER** que tiene que hacer **para mi servicio** cuando **crea el contenedor.** Sirve para quitar lineas de codigo del **docker-compose.yml.** De manera que una **buena practica** es usar un **dockerfile por servicio** para **ahorrar espacio** del **docker-compose.yml**

docker-compose.yml: Este archivo sirve para **configurar** el montaje del **contenedor** que montamos con **DOCKER**.

AMBOS ARCHIVOS TIENEN UNA SINTAXIS DIFERENTE

index.php: Archivo de **nuestra pagina en php**, similar al index.html.
init.sql: Archivo que nos sirve para **inicializar una BBDD**.

, docker-compose.yml, index.php, init.sql]

Dockerfile*

```
FROM php:8.2-apache

# mysqli
RUN docker-php-ext-install -j"$(nproc)" mysqli
WORKDIR /var/www/html
COPY src/ .
```

Docker-compose.yaml

```
services:
  web:
    build:
      context: ./services/web
      dockerfile: Dockerfile
    ports:
      - "8080:80"
    depends_on:



     - db
```

```
  db:
    image: mariadb:10.5
    environment:
      MYSQL_USER: user
      MYSQL_ROOT_USER: root
      MYSQL_PASSWORD: pass
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: testdb

    volumes:
      - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
```

index.php

```
<?php
$conn = new mysqli("db", "root", "pass", "testdb");
if ($conn->msqli_connect_error) {
    die("Error de conexión: " . $conn->msqli_connect_error);
}
$result = $conn->query("SELECT mensaje FROM mensajes");
$row = $result->fetch_assoc();
echo "Mensaje desde la BD: " . $row["mensaje"];
?>
```

init.sql

```
CREATE DATABASE IF NOT EXISTS testdb;
USE testdb;
CREATE TABLE mensajes (id INT AUTO_INCREMENT PRIMARY KEY, mensaje
VARCHAR(100));
INSERT INTO mensajes (mensaje) VALUES ('¡Hola desde MariaDB en
Docker!');
```

2. Añadimos interfaz a la bd (0.5 puntos)

Descripción: Crear una UI para nuestra base de datos. Añadimos un **nuevo servicio** llamado **phpmyadmin**. Con la **imagen phpmyadmin:5.** Declarando variables de entorno para acceder una vez creado el contenedor **(SON LAS MISMAS QUE LAS DE NUESTRA BBBDD. PMA_HOST == el nombre de nuestro servicio de bbdd).** Lo vamos a mostrar por el **puerto 8081.** Y este servicio **depende de BBDD,** es necesario que la **BBDD este levantada antes de que se levante este servicio**.

Docker-compose.yaml

```
phpmyadmin:
   image: phpmyadmin:5
   ports:
     - "8081:80"
   environment:
     PMA_HOST: db
     PMA_USER: user
     PMA_PASSWORD: pass
   depends_on:
     - db
```

3. Añadimos variables de entorno en un .env (1 puntos)

Descripción: Crear un nuevo archivo en la raíz de nuestro proyecto, llamado **“.env”(es la propia extensión).** Que contiene claves o datos que ya estábamos usando.

.env

```
MYSQL_USER: user
MYSQL_ROOT_USER: root
MYSQL_PASSWORD: pass
MYSQL_ROOT_PASSWORD: rootpass
MYSQL_DATABASE: testdb
WEB_PORT=8080
PMA_PORT=8081
```

Y modificamos el **docker-compose.yml**

`${NOMBRE VARIABLE DE ENTORNO}`

4. Añadimos persistencia al contenedor (1 punto)

Descripción: Añadimos un nuevo volumen al servicio **db,** indicamos el **nombre del volumen: Ruta donde se va a guardar en el contenedor esos datos. Al final del docker-compose es necesario declarar todos los volúmenes creados**

Docker-compose.yaml

```
  db:
      volumes:
        - dbdata:/var/lib/mysql

volumes:
  dbdata:
```

5. Distinguimos entre servicios (1 punto)

Descripción: Mediante la etiqueta networks vamos a crear dos redes, **frontend y backend. A continuación indica a qué red pertenece cada servicio. También es necesario indicar las redes al final del docker-compose. Es una forma de clasificar los servicios.**

Docker-compose.yaml

```
services:
  web:
    networks: [frontend, backend]
  db:
    networks: [backend]

networks:
  frontend:
  backend:
```

6. Check de la base de datos (0.5 puntos)

Descripción: Añadimos un comprobante de que se ha levantado el servicio **db. Para comprobarlo realizar el comando "docker compose ps". Siempre que los contenedores estén levantados.**

Docker-compose.yaml

```
 db:
    healthcheck:
      test: ["CMD-SHELL", "mysqladmin ping -h 127.0.0.1 -uroot
-p\"$${MYSQL_ROOT_PASSWORD}\" --silent"]
      interval: 5s
      timeout: 3s
      retries: 20
      start_period: 20s

  web:
    depends_on:
      db:
        condition: service_healthy
```

7. Certificado autofirmado (TLS) (1.5 puntos)

Generar el certificado **Necesitas descargar openSSL para utilizar el comando para generar el certificado. O ejecuta el comando a traves de Git Bash (https://slproweb.com/products/Win32OpenSSL.html?utm_source)**

Descripción: Generamos los certificados necesarios para establecer nuestra web como segura. Para ello vamos a generar los certificados con el comando:

Comando

```
mkdir certs openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
certs/selfsigned.key -out certs/selfsigned.crt -subj "/CN=localhost"
-addext "subjectAltName=DNS:localhost,IP:127.0.0.1"
```

… Pero es necesario un servicio para **buenas prácticas**, vamos a levantar un **servicio proxy inverso, para ello necesitamos en nginx.conf que define la configuración del proxy.**

nginx.conf

```
server {
  listen 80;
  server_name localhost;
  return 301 https://$host:8443$request_uri;
}

server {
  listen 443 ssl;
  server_name localhost;

  ssl_certificate     /etc/nginx/certs/selfsigned.crt;
  ssl_certificate_key /etc/nginx/certs/selfsigned.key;

  set $app web;

  location / {
    proxy_pass http://$app:80;
    proxy_set_header Host              $host;
    proxy_set_header X-Real-IP         $remote_addr;
    proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
}
}
```

…El nuevo servicio se llamara **proxy,** indicamos la imagen en este caso nginx:alpine, e indicamos los puertos, además **copiamos nuestro fichero de conf** en el **contenedor. Tambien necesita que copiemos los certificados en el contenedor.** El proxy depende de que la web este lavanda. Es necesario **cambiar los puertos a "expose"** borrando la sección **"ports",** ya que es el proxy el que nos va a redirigir al servicio web en el puerto 8443

Docker-compose.yaml

```
proxy:
    image: nginx:alpine
    ports:
```

```
      - "80:80"
      - "443:443"
    volumes:
      - ./services/nginx/conf.d:/etc/nginx/conf.d:ro
      - ./certs:/etc/nginx/certs:ro
    depends_on:
      web:
        condition: service_started
web:
    build:
      context: ./services/web
      dockerfile: Dockerfile
    expose: ["80"]
```

8. Balanceo de carga (1 punto)

Descripción: vamos a **ejecutar dos versiones de nuestro servicio web**, para balancear la carga del servidor. Para ello es necesario **añadir la linea de codigo en php**, para mostrar en que versión del servicio nos encontramos.

index.php

```
echo "Servidor: " . gethostname();
```

Comando

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

9. Backups (1 punto)

Descripción: Tenemos que **crear un archivo backup.sh**, que guarda el codigo de un **script de bash** para crear **copias de seguridad de la db cada cierto tiempo(60S)**.

Por otra parte, **añadimos un nuevo servicio**, para ejecutar el **script de backup.sh**. Además de la imagen, añadimos la variable de la **contraseña de root de db. También copiamos el script para que el contenedor lo pueda ejecutar. Y le pasamos el comando a ejecutar para lanzar el script.**

backup.sh

```
#!/bin/sh
set -e

: "${SLEEP_SECONDS:=60}"
: "${MYSQL_HOST:=dockerdb}"
: "${MYSQL_USER:=root}"
: "${MYSQL_PASSWORD:=pass}"
: "${MYSQL_DATABASE:=testdb}"

echo "[backup] Iniciando ciclo (cada ${SLEEP_SECONDS}s) contra
${MYSQL_HOST}/${MYSQL_DATABASE}"
while true; do
  TS=$(date +%F-%H%M%S)
  FILE="/backups/backup-$TS.sql"
  echo "[backup] Generando $FILE ..."
  mariadb-dump -h "$MYSQL_HOST" -u "$MYSQL_USER" -p"$MYSQL_PASSWORD"
--databases "$MYSQL_DATABASE" --skip-ssl > "$FILE"
  echo "[backup] OK -> $FILE"
  sleep "$SLEEP_SECONDS"
done
```

Docker-compose.yaml

```
 backup:
    image: alpine:3
    environment:
      MYSQL_ROOT_PASSWORD: "${MYSQL_ROOT_PASSWORD}"
    volumes:
      - ./backup.sh:/backup.sh:ro
    command: sh -c "apk add --no-cache mariadb-client && while true; do
sh /backup.sh; sleep 60; done"
    depends_on:
      - db
```

10. Servidor de aplicaciones (1 punto)

Es necesario descargar el archivo war de prueba,
https://tomcat.apache.org/tomcat-9.0-doc/appdev/sample/sample.war

Descripción: Creamos un nuevo servicio para alojar aplicaciones JAVA, usando tomcat, un servidor de aplicaciones. Además de la imagen de tomcat, **exposeamos el puerto 8080, para usar el proxy que teníamos creado.** No olvides copiar el fichero de ejemplo(enlace) de Tomcat en el contenedor.

Por último para que funcione el proxy, debemos añadir ese trozo de código a su configuración.

Docker-compose.yaml

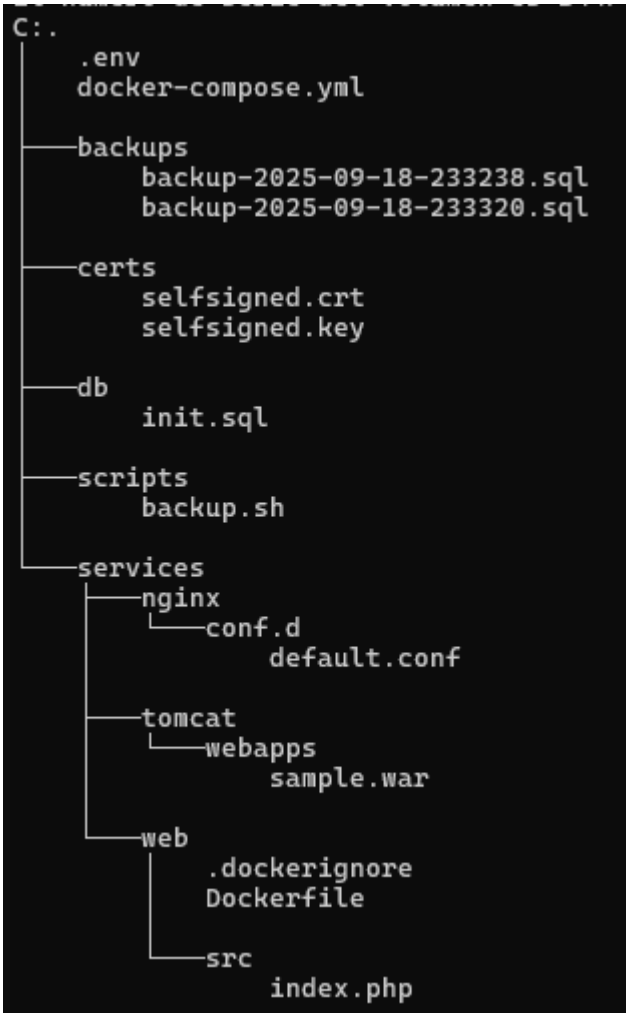
```
tomcat:
    image: tomcat:9.0
    expose: ["8080"]
    volumes:
      -
./services/tomcat/webapps/sample.war:/usr/local/tomcat/webapps/sample.war
    networks: [backend]
```

Nginx.conf

```
server {
  listen 8080;
  server_name localhost;

  location /sample/ {
  proxy_pass http://tomcat:8080/sample/;
  proxy_set_header Host $host;
  }
}
```

Estructura de carpetas de ejemplo:



Entrega:

Es necesario entregar el proyecto con **todos los archivos** en un **.RAR** con el nombre, **Apellido1_Nombre_PracticaT1.rar**, añadir el **enlace al documento o** el propio **documento** en el que se muestran las distintas **capturas de cada punto realizado** y la descripción de los beneficios que otorga cada punto al proyecto. Incluye en el documento los **problemas que has tenido** al realizar los puntos, si los has tenido y **cómo los has resuelto**.

(SI NO SE CUMPLEN LAS REGLAS DE ENTREGA, SE CONSIDERARÁ QUE LA PRÁCTICA NO SE HA ENTREGADO)