HTML5, CSS3 y ES6
Docker

Centro público integrado de formación profesional

Alan Turing

Índice

Bloque 1 – ¿Qué es Docker y docker-compose?

Bloque 1 – ¿Qué es Docker y docker-compose?

Antes de entrar al archivo, conviene que sepamos lo siguiente:

- **Docker** es una herramienta que nos permite **crear "contenedores"** (containers) que guardan nuestras **aplicaciones** con todas sus **dependencias**. Es como una **mini máquina virtual** ligera.
- Un **Dockerfile** es un archivo con **instrucciones** que dice cómo **construir** la **imagen** de uno de esos **contenedores** (qué sistema base usar, qué programas instalar, qué comando arrancar, etc.).
- **Docker-Compose** es una capa encima de Docker que permite definir **varios contenedores (servicios)** juntos, cómo se conectan entre sí, qué puertos exponen, qué carpetas compartidas ("volúmenes") usan, etc. Todo eso se escribe en un **archivo** llamado **docker-compose.yml**.
- Con **Docker-Compose** podemos levantar todos los **servicios** de nuestra aplicación (frontends, APIs, bases de datos) con un solo comando, en lugar de arrancarlos uno por uno manualmente.

🔍 Bloque 2 - Explicación del docker-compose.yml

Bloque 2 - Explicación del docker-compose.yml

Vayamos por partes:

version: "3.9"

Indica la **versión** del formato de **docker-compose** que estamos usando (3.9). Permite algunas características específicas.

```
services:
  angular16:
    build: ./angular16
    container_name: angular16
    working_dir: /usr/src/app
    volumes:
      - ./angular16:/usr/src/app
    ports:
      - "4216:4200"
    tty: true
    stdin_open: true
    environment:
      - CHOKIDAR_USEPOLLING=true
```

🔍 Bloque 2 - Explicación del docker-compose.yml

Este bloque define un servicio llamado angular16 con lo siguiente:

- **build: ./angular16** → Le dice que dentro de la carpeta angular16 hay un Dockerfile, úsalo para construir la imagen de este servicio.
- **container_name: angular16** → Le da un nombre fácil al contenedor resultante.
- **working_dir: /usr/src/app** → Dentro del contenedor, la carpeta /usr/src/app será el directorio de trabajo (donde estaremos trabajando).
- **volumes: - ./angular16:/usr/src/app** → Montamos la carpeta de tu proyecto en la máquina local ./angular16 dentro del contenedor en /usr/src/app. Esto hace que tus archivos de código estén sincronizados entre tu máquina y el contenedor.
- **ports: - "4216:4200"** → Mapea el puerto 4200 del contenedor al puerto 4216 de tu máquina local. De ese modo, desde el navegador local accedes al contenido del contenedor vía localhost:4216.
- **tty: true y stdin_open: true** → Permiten que podamos abrir una consola interactiva dentro del contenedor (entrar al contenedor como si fuera una terminal).
- **environment: - CHOKIDAR_USEPOLLING=true** → Define una variable de entorno interna para que el sistema de vigilancia de archivos de Angular detecte cambios correctamente dentro del contenedor (para que el “live reload” funcione bien).

Este patrón **se repite** para angular17, angular18, angular19, angular20, y para react19, con las diferencias de versión y puertos correspondientes.

Bloque 3 - Servicios de producción / estáticos con nginx

Bloque 3 - Servicios de producción / estáticos con nginx

Para JS Vanilla:

```
static-files:
  image: nginx:alpine
  container_name: static-files
  volumes:
    - ./vanilla-sites:/usr/share/nginx/html
    - ./nginx-vanilla.conf:/etc/nginx/nginx.conf:ro
  ports:
    - "4300:80"
```

- Monta la **carpeta vanilla-sites**, que contendrá múltiples proyectos JS Vanilla como subcarpetas.
- Usa configuración **nginx-vanilla.conf** que permite “autoindex” (listado de directorios).
- Expuesto al **puerto 4300** local.
- Cuando abres **http://localhost:4300/**, nginx mostrará una lista de carpetas (proyectos), y al hacer clic en una carpeta que tenga index.html, lo servirá.

Bloque 3 - Servicios de producción / estáticos con nginx

Para Angular:

```
angular-app:
  image: nginx:alpine
  container_name: angular-app
  volumes:
    - ./angular-app/dist:/usr/share/nginx/html
    - ./nginx-angular.conf:/etc/nginx/nginx.conf:ro
  ports:
    - "4200:80"
```

Este servicio no es de desarrollo, sino para servir archivos que ya están compilados (**Angular builds**).

- **image: nginx:alpine** → Usa una imagen lista de nginx, **servidor web** liviano.
- **volumes: - ./angular-app/dist:/usr/share/nginx/html** → Monta la **carpeta dist** (tu código compilado) en la carpeta que nginx usa para servir contenido estático (/usr/share/nginx/html).
- **- ./nginx-angular.conf:/etc/nginx/nginx.conf:ro** → Monta una configuración personalizada de nginx (lectura solamente) para decirle cómo servir esos archivos (por ejemplo rutas de fallback para SPA).
- **ports: - "4200:80"** → Mapea el puerto 80 del contenedor (nginx) al **puerto 4200** de tu máquina.
- Así puedes abrir en el navegador **http://localhost:4200** y ver tu aplicación Angular compilada.

Bloque 3 - Servicios de producción / estáticos con nginx

Otros servicios: APIs, bases de datos

- **php-api**: construye tu API con PHP.
- **flask-api**: para tu API en Python / Flask.
- **nodejs-api**: para APIs Node.js.
- **mysql-db**: servicio de base de datos MySQL.
- **phpmyadmin**: interfaz web para administrar MySQL.

Cada uno define su imagen, volúmenes, puertos, variables de entorno, etc.

La sección final:

```
networks:
  default:
    name: shared_network
volumes:
  mysql_data:
```

- **networks**: define una red común llamada shared_network. Todos los servicios (si no se especifica otra) se conectarán ahí y podrán comunicarse con el nombre de servicio (por ejemplo, flask-api podrá localizar mysql-db).
- **volumes: mysql_data:** → define un volumen persistente para la base de datos, de modo que incluso si se apaga el contenedor, los datos no se pierdan.

Bloque 4 - Dockerfiles

Bloque 4 - Dockerfiles

Bloque 4 - Dockerfiles

Ejemplo de **Dockerfile** para **angular16**

```
FROM node:18-alpine
RUN apk add --no-cache bash
RUN npm install -g @angular/cli@16
RUN echo 'alias ng-serve="ng serve --host 0.0.0.0"' >> ~/.bashrc
WORKDIR /usr/src/app
CMD ["bash"]
```

- **FROM node:18-alpine**: base del contenedor: sistema liviano con Node 18.
- **RUN apk add … bash**: instala bash (shell) para tener comandos más cómodos dentro.
- **RUN npm install -g @angular/cli@16**: instala la CLI de Angular versión 16 globalmente.
- **RUN echo 'alias …' >> ~/.bashrc**: agrega un alias para que ng-serve sea más cómodo: ejecuta ng serve --host 0.0.0.0.
- **WORKDIR /usr/src/app**: define el directorio de trabajo interno.
- **CMD ["bash"]**: el comando que se ejecuta cuando se inicia el contenedor: te deja con una consola bash dentro.
- **Cada versión Angular tiene su Dockerfile** con su versión adecuada del CLI.

Bloque 4 - Dockerfiles

Dockerfile para **React**:

```
FROM node:20-alpine
RUN apk add --no-cache bash
RUN npm install -g vite@7.1.2
RUN npm install -g @vitejs/plugin-react
RUN echo 'alias react-start="npm run dev -- --host 0.0.0.0"' >> ~/.bashrc
WORKDIR /usr/src/app
CMD ["bash"]
```

Misma explicación anterior, pero esta vez para **React / Vite**.

Bloque 4 - Dockerfiles

Docker para **Python / Flask**:

```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
CMD ["flask", "run", "--host=0.0.0.0"]
```

Copia el archivo **requirements.txt** que contiene:

```
Flask==2.0.1
Werkzeug>=2.0,<2.1
flask_sqlalchemy==2.5
flask_cors==3.0.10
mysql-connector-python==8.0.23
```

Luego instala esas dependencias dentro del contenedor.

Finalmente ejecuta el comando para levantar el servidor Flask escuchando en cualquier interfaz (0.0.0.0).

Bloque 5 - Comandos básicos

Bloque 5 - Comandos básicos

Bloque 5 - Comandos básicos

Estos son los comandos que usaremos en clase:

- **docker compose up -d** → **levanta** todos los servicios definidos en docker-compose.yml en segundo plano.
- **docker compose down** → **detiene** y elimina los contenedores, redes y volúmenes creados por up.
- **docker compose up -d --build** → **reconstruye** las imágenes (útil cuando cambias un Dockerfile) y luego levanta.
- **docker compose logs <servicio>** → ver los **registros** (salida de consola) de un **servicio** específico.
- **docker compose ps** → muestra qué **contenedores** están **corriendo** y sus puertos.
- **docker exec -it <nombre_contenedor> bash** → **entrar** dentro de un **contenedor** en modo consola (shell).
- **docker compose stop <servicio> / start <servicio>** → detener / iniciar un servicio sin eliminarlo.

Ejercicio

Ejercicio. Enunciado

Implementa el código anterior.

Ejercicio. Enunciado

Resultado: