Laboratorio 8. Aplicaciones Multi-Container a través de Docker

Parte A. (se expone a nivel general para entender el propósito de aplicaciones multi-container)

Introducción:

¿Qué es docker compose?

Docker compose es una herramienta que simplifica la gestión de aplicaciones multi-contenedor en Docker. Permite definir la configuración de servicios, redes y volúmenes en un archivo YAML, facilitando la orquestación y despliegue de aplicaciones complejas. Para configurar variables de entorno y contraseñas, puedes utilizar la sección `environment` en el archivo YAML.
Por ejemplo, al definir un servicio de base de datos, especifica variables como `MYSQL_ROOT_PASSWORD` para establecer la contraseña del usuario root.
A continuación, se puede ver un resumen práctico:
1. Instalación de Docker Compose:
Asegúrate de tener Docker Compose instalado en tu sistema.
2. Creación de un archivo docker-compose.yml: (recuerda las prácticas del principio del curso, sobre la estructura y sintaxis del lenguaje yaml).
Puedes solo ver o crear un archivo `docker-compose.yml` para tu aplicación (No hace falta realizarlo, solo se expone a nivel de ayudar a comprender los conceptos). Define servicios, como una base de datos y una aplicación web, y configura las variables de entorno según sea necesario. Atención, es muy importante en la creación del archivo respetar la jerarquía.
```terminal
version: '3'

services:
  web:
    image: nginx:latest
    ports:
      - "80:80"

  database:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: example_password
```

```

3. Ejecución de la aplicación:
   Ejecuta `*docker-compose up*` en el directorio que contiene el archivo YAML. Docker Compose descargará las imágenes, creará los contenedores y aplicará la configuración.

4. Acceso a la aplicación:
   Accede a tu aplicación web en `http://localhost`. La base de datos estará disponible con la contraseña especificada en el archivo YAML.

Este ejemplo permite comprender cómo Docker Compose simplifica la gestión de servicios y cómo configurar variables de entorno, en este caso, para establecer contraseñas de bases de datos.

**Node.js**:

Node.js es un entorno de ejecución para JavaScript basado en el motor V8 de Google Chrome. Permite ejecutar código JavaScript en el lado del servidor, lo que facilita la creación de aplicaciones web escalables y de alto rendimiento. Node.js es especialmente conocido por su capacidad para manejar operaciones de entrada/salida de manera eficiente y por su enfoque asincrónico, lo que lo hace adecuado para aplicaciones en tiempo real y API (Interfaz de Programación de Aplicaciones) basadas en eventos.

**REST API**:

**REST** (Representational State Transfer) es un estilo de arquitectura de software que utiliza estándares y restricciones específicas para construir servicios web. Una API REST (Interfaz de Programación de Aplicaciones basada en Transferencia de Estado Representacional) se basa en los principios REST y utiliza operaciones HTTP estándar (GET, POST, PUT, DELETE) para realizar operaciones en recursos, los cuales son identificados por URLs. Las API REST son utilizadas para permitir la comunicación y la transferencia de datos entre sistemas de manera simple y eficiente.
```

En el contexto de Node.js, puedes construir una REST API utilizando frameworks como Express.js. Esto implica definir rutas, manejar solicitudes HTTP y realizar operaciones en la base de datos, proporcionando así un conjunto de endpoints que otros sistemas pueden utilizar para interactuar con tu aplicación.

React SPA:

React es una biblioteca de JavaScript desarrollada por Facebook para construir interfaces de usuario interactivas y reutilizables. Una SPA (Single Page Application) es un tipo de aplicación web que carga una sola página HTML y actualiza dinámicamente el contenido a medida que el usuario interactúa con la aplicación, sin necesidad de cargar páginas adicionales desde el servidor. React es comúnmente utilizado para construir SPAs debido a su enfoque en componentes reutilizables y su capacidad para manejar eficientemente el estado de la interfaz de usuario.

En una SPA con React, la interfaz de usuario se actualiza de manera dinámica sin necesidad de recargar la página completa. React gestiona el estado de la aplicación y renderiza componentes según sea necesario. La comunicación con el servidor, como la obtención de datos de una REST API construida con Node.js, se realiza mediante solicitudes AJAX o fetch para actualizar dinámicamente la interfaz de usuario sin recargar la página completa.
AJAX (Asynchronous JavaScript and XML):

AJAX es una técnica de desarrollo web que permite realizar solicitudes asíncronas al servidor desde una página web sin tener que recargar la página completa. Utiliza JavaScript para enviar y recibir datos en segundo plano, permitiendo actualizaciones dinámicas de la interfaz de usuario sin interrupciones perceptibles por parte del usuario.

En lugar de esperar a que se cargue una página completa, AJAX permite que ciertas partes de la página se actualicen de forma

independiente. Esto mejora la velocidad y la eficiencia de las aplicaciones web al minimizar el tiempo de espera del usuario.

Fetch:

`fetch` es una API moderna de JavaScript que proporciona una interfaz para realizar solicitudes HTTP. Se utiliza para recuperar recursos desde el servidor de manera asíncrona. `fetch` es más potente y flexible que las técnicas tradicionales de AJAX y se ha convertido en una forma común de realizar solicitudes de red en aplicaciones web modernas.

Un ejemplo básico de `fetch` se vería así:

```terminal
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

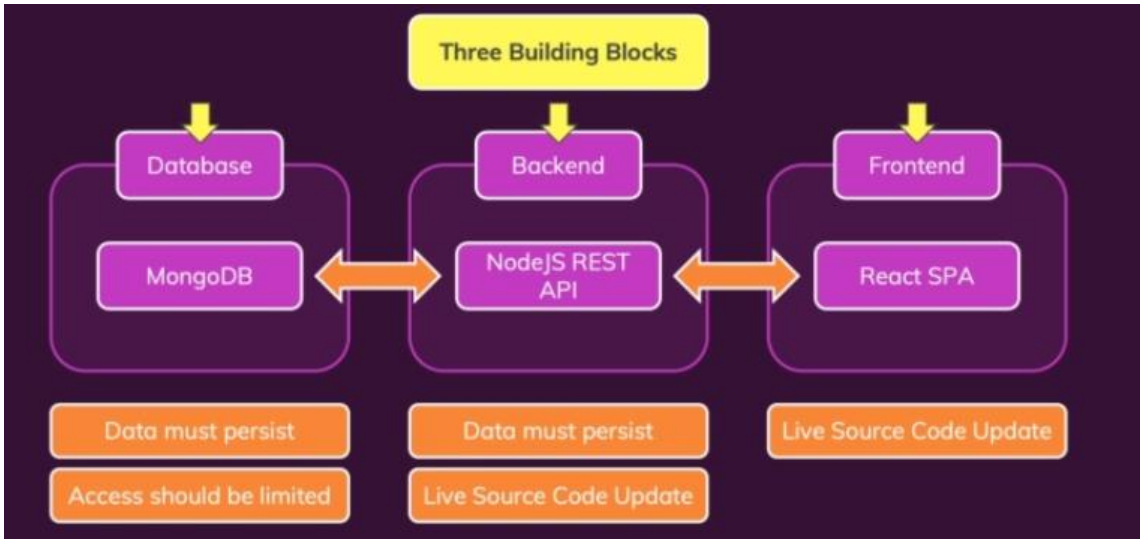
En este ejemplo, `fetch` realiza una solicitud GET a la URL proporcionada. La promesa resultante se maneja con `.then()` para procesar la respuesta en formato JSON. La cadena de `.catch()` maneja cualquier error que pueda ocurrir durante la solicitud.

Ambas, AJAX y `fetch`, son herramientas esenciales para realizar comunicación asíncrona en aplicaciones web y permiten la actualización dinámica de contenido sin recargar la página completa.

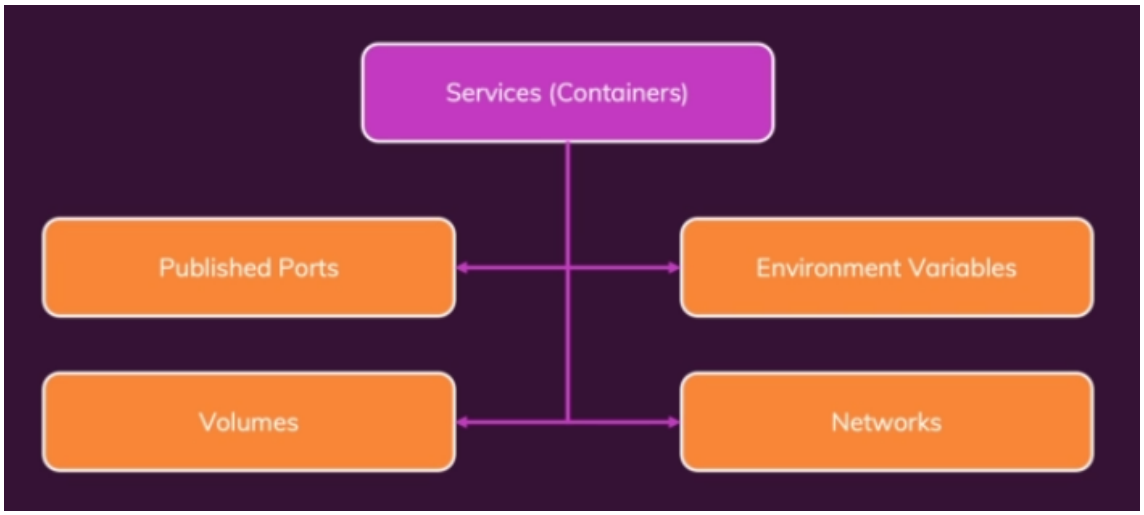
Parte B.

El objetivo de la práctica es la construcción de aplicaciones Multi-Container a través de Docker. En concreto, se hará uso de *docker compose*.

A veces, tenemos aplicaciones que constan de varios servicios diferentes que trabajan juntos. Imaginad, una base de datos, un servidor, etc. En resumen, de múltiples contenedores que funcionan juntos.
Esta práctica trata de combinar y practicar lo que anteriormente se ha aprendido. Una aplicación de múltiples servicios y contenedores.
Tenemos una base de datos MongoDB y previamente debemos realizar una forma de agregar un usuario y una contraseña a esta base. Igual que para la base de datos, para nuestra API de backend de Node, los datos también deben persistir.



Contenido de los servicios



Primero creamos una carpeta en nuestro directorio para el proyectó docker-compose , vamos a descomprimir la carpeta adjunta ***compose-01-starting-setup.***

```
version: '3.8'

services:
  server:
    build:
      context: .
      dockerfile: dockerfiles/nginx.dockerfile
    ports:
      - '8000:80'
    volumes:
      - ./src:/var/www/html
      -
./nginx/nginx.conf:/etc/nginx/conf.d/default.conf:ro
    depends_on:
      - php
      - mysql

  #php:
  #  build:
  #    context: .
  #    dockerfile: dockerfiles/php.dockerfile
  #  volumes:
  #    - ./src:/var/www/html:delegated

  php:
```

```
    image: laravel:7.4 # Puedes cambiar la versión según
tus necesidades
    build:
       context: .
       dockerfile: dockerfiles/php.dockerfile
    volumes:
       - ./src:/var/www/html:delegated


  mysql:
    image: mysql:5.7
    env_file:
      - ./env/mysql.env

  composer:
    image: composer:latest
    volumes:
      - ./src:/var/www/html
    entrypoint: ["composer", "--ignore-platform-reqs"]

  artisan:
    image: php:7.4-cli
    volumes:
      - ./src:/var/www/html
    entrypoint: ['php', '/var/www/html/artisan']
    depends_on:
      - php
      - mysql

  npm:
    image: node:14
    working_dir: /var/www/html
    entrypoint: ["npm"]
    volumes:
      - ./src:/var/www/html
```