3.2 – Contenedores y Orquestadores

Tema 2 – Docker Compose

Universidad Rey Juan Carlos

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

code URJC

Docker Compose

- Introducción
- Ejemplo Docker Compose
- Espera de servicios
- Herramientas para imágenes
- Plugin VSCode para Docker Compose
- Beneficios de Docker Compose

Docker Compose

- **Introducción**
- Ejemplo Docker Compose
- Espera de servicios
- Herramientas para imágenes
- Plugin VSCode para Docker Compose
- Beneficios de Docker Compose

Introducción

- Es una herramienta para definir aplicaciones formadas por **varios contenedores**
- Un fichero YAML define los **contenedores** (imagen, puertos, volúmenes…) y cómo se **relacionan** entre sí
- Los contenedores se comunican:
  - Protocoles de red
  - Volúmenes compartidos

https://docs.docker.com/compose/

Introducción

- El plugin de Docker Compose viene por defecto en todas las versiones de Docker Desktop
- Para la versión de Docker Engine en Linux, hay que instalar un paquete adicional
- El fichero YAML se suele llamar

`docker-compose.yml`

- En la carpeta donde está el fichero, la aplicación se ejecuta con el comando

```
$ docker compose up
```

https://docs.docker.com/compose/install/

Introducción

- **Definición de cada contenedor**
  - **Imagen**
    - Puede descargarse de DockerHub
    - Puede estar construida localmente
    - Puede construirse con un Dockerfile en el momento de iniciar la aplicación

Introducción

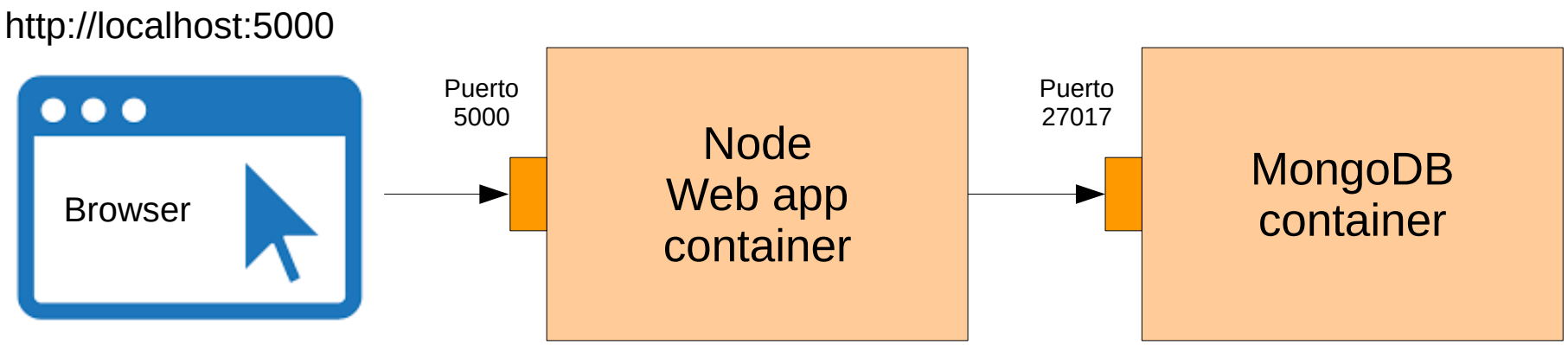
- **Definición de cada contenedor**
  - **Puertos:**
    - Mapeados en el host (para ser usados desde localhost)
    - No mapeados (sólo se pueden conectar otros contenedores)
  - **Volúmenes:**
    - Carpetas del host accesible desde el contenedor
    - Compartidos entre contenedores (en una carpeta interna de docker)

Docker Compose

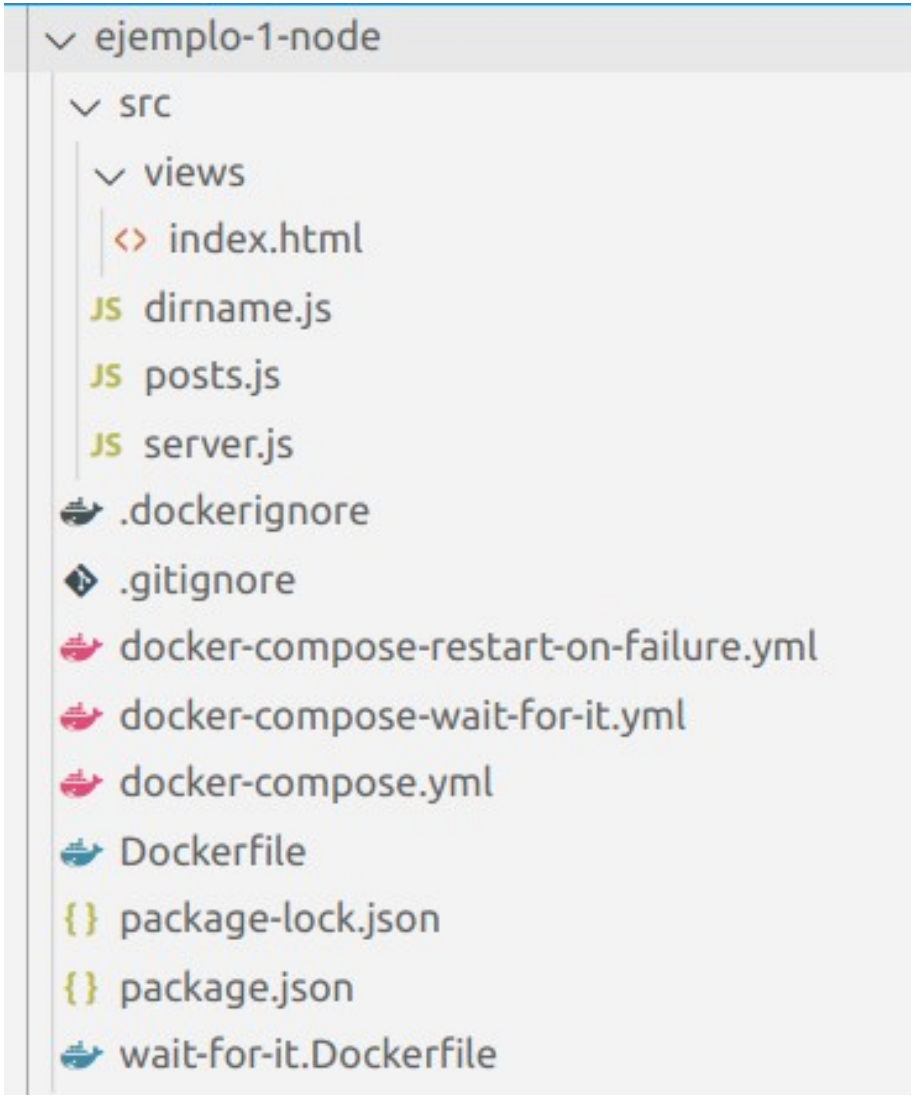
- Introducción
- **Ejemplo Docker Compose**
- Espera de servicios
- Herramientas para imágenes
- Plugin VSCode para Docker Compose
- Distribución de aplicaciones Dockerizadas

Ejemplo Docker Compose

- **Apicación web con BBDD**
  - Web con tecnología Node (express, mustache-express)
  - BBDD MongoDB
  - 2 Contenedores



Ejemplo Docker Compose



https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestadores/tree/master/docker-compose/ejemplo-1-node

Ejemplo Docker Compose

- Node Web app

server.js

```
import express from 'express';
import mustacheExpress from 'mustache-express';
import { router as postsRouter, init as postsInit } from './posts.js';
import { __dirname } from './dirname.js';

const APP_PORT = 5000;

const app = express();

// Register '.html' extension with The Mustache Express
app.set('view engine', 'html');
app.set('views', __dirname + '/views');
app.engine('html', mustacheExpress());

app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(postsRouter);

process.on('SIGINT', () => {
    process.exit(0);
});

await postsInit();

app.listen(APP_PORT, () => {
    console.log(`* Running on http://localhost:${APP_PORT}/`);
    console.log(" (Press CTRL+C to quit)");
});
```

- Node Web app

posts.js

```
import express from 'express';
import { MongoClient } from 'mongodb';

const dbHost = process.env.MONGODB_HOST || "localhost";
const dbPort = process.env.MONGODB_PORT || "27017";
const dbCollection = process.env.MONGODB_COLLECTION || "posts";
const mongoUrl = `mongodb://${dbHost}:${dbPort}/${dbCollection}`;

let posts;

async function dbConnect() {
      console.log("Database Configurations:");
      console.log(`\t- MONGODB_HOST: ${dbHost}`);
      console.log(`\t- MONGODB_PORT: ${dbPort}`);
      console.log(`\t- MONGODB_COLLECTION: ${dbCollection}`);

      const conn = await MongoClient.connect(mongoUrl, {
            useUnifiedTopology: true,
            useNewUrlParser: true
      });

      posts = conn.db().collection('posts');
}

export async function init() {
      await dbConnect();
}

export const router = express.Router();

router.get('/', async (req, res) => {
      const allPosts = await posts.find({}).toArray();
      res.render('index', { items: allPosts });
});

router.post('/new', async (req, res) => {
      let post = {
            name: req.body.name,
            description: req.body.description,
      };

      await posts.insertOne(post);
      res.redirect('/');
});
```

Ejemplo Docker Compose

- HTML Template

src/views/index.html

```
<h1>Tablón de anuncios</h1>

<form action="/new" method="POST">
    <input type="text" name="name"></input>
    <input type="text" name="description"></input>
    <input type="submit"></input>
</form>

{{ #items }}
    <h1> {{ name }} </h1>
    <p> {{ description }} </p>
{{ /items }}
```

- package.js

```
"dependencies": {
    "express": "4.18.2",
    "mongodb": "5.1.0",
    "mustache-express": "1.3.2"
}
```

Ejemplo Docker Compose

- Dockerfile de la aplicación web

Dockerfile

```
# Imagen base para el contenedor de compilación
FROM node:lts-alpine as builder
WORKDIR /usr/src/app/
COPY package.json /usr/src/app/
RUN npm install --only=production

# Imagen base para el contenedor de la aplicación
FROM node:lts-alpine
ENV NODE_ENV production
WORKDIR /usr/src/app
COPY --from=builder /usr/src/app/node_modules
/usr/src/app/node_modules
COPY src /usr/src/app/src
COPY package.json /usr/src/app/
EXPOSE 5000
CMD ["node", "src/server.js"]
```

Ejemplo Docker Compose

- Fichero docker-compose.yml

Servicios

Cada contenedor es un servicio de la app

La propiedad indica el nombre del servicio

Esta app tiene dos servicios: “web” y “db”

docker-compose.yml

```
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      - db
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

https://docs.docker.com/compose/compose-file/

Ejemplo Docker Compose

- Docker Compose crea una **red nueva** cuando se ejecuta
- Todos los servicios del Docker Compose estarán **conectados** por defecto en la **nueva red**
- Todos los servicios se podrán ver entre ellos utilizando el **nombre del servicio**
- Desde el servicio **"web"** podremos usar el nombre del servicio **"db"** para conectarnos con la **base de datos**

Ejemplo Docker Compose

- Fichero docker-compose.yml

environment

Variables de entorno que cargaremos dentro del container. De esta forma el container puede hacer uso de ellas

docker-compose.yml

```yaml
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      - db
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

https://docs.docker.com/compose/compose-file/#environment

Ejemplo Docker Compose

- Fichero docker-compose.yml

docker-compose.yml

```yaml
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      - db
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

build

Se indica la ruta del Dockerfile para construir la imagen

https://docs.docker.com/compose/compose-file/#build

Ejemplo Docker Compose

- Fichero docker-compose.yml

docker-compose.yml

```yaml
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      - db
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

image

Se indica el nombre de la imagen en DockerHub o en local

https://docs.docker.com/compose/compose-file/#image

Ejemplo Docker Compose

- Fichero docker-compose.yml

docker-compose.yml

```yaml
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      - db
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

ports

Puertos "bindeados" al host. Como la opción -p al arrancar un contenedor

https://docs.docker.com/compose/compose-file/#ports

Ejemplo Docker Compose

- **Fichero docker-compose.yml**

depends_on

Indica el orden de **arranque** y **apagado**. En este caso primero arrancará el servicio 'db' y posteriormente 'web'

'web' no espera a que la base de datos esté operativa para arrancar, solo espera que el container 'db' esté levantado

Es posible indicar condiciones de espera (lo veremos más adelante)

docker-compose.yml

```
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      - db
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

https://docs.docker.com/compose/compose-file/#depends_on

Ejemplo Docker Compose

- **Fichero docker-compose.yml**

docker-compose.yml

```
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      - db
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

volumes

Carpetas del host accesibles desde el contenedor

https://docs.docker.com/compose/compose-file/#volumes

Ejemplo Docker Compose

code URJC

ejemplo-1-node

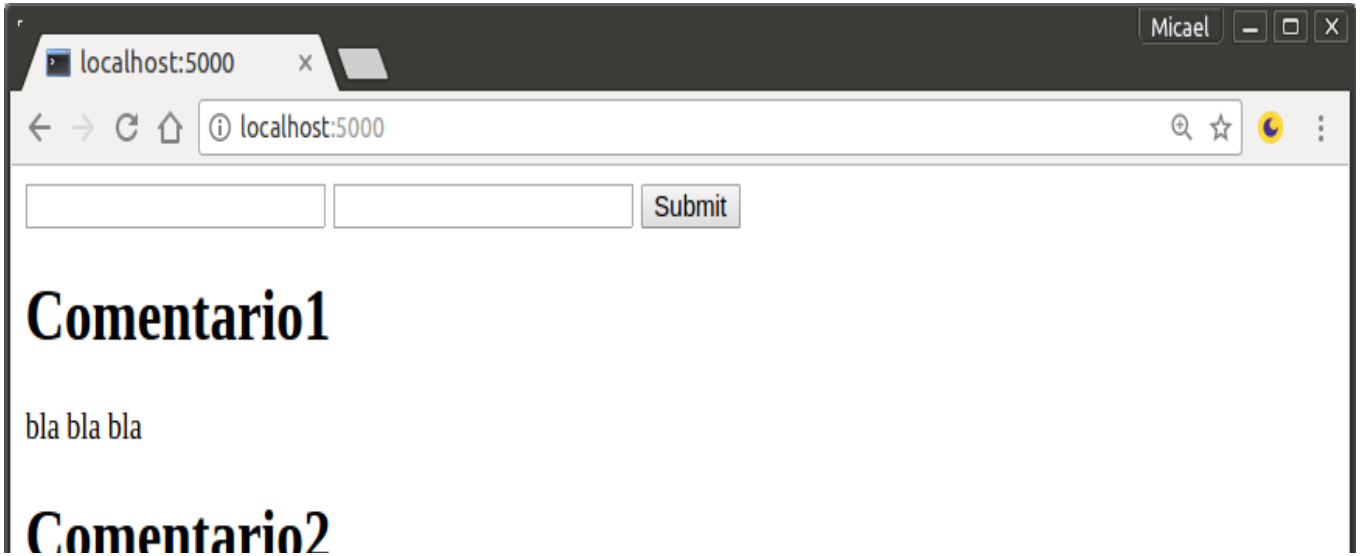
- **Arrancar la aplicación**

```
$ docker compose up
```

  - Construye la imagen si no está construida ya.
    - Si la imagen está disponible, no se reconstruye aunque cambie el código o el Dockerfile, es necesario ejecutar **docker compose up --build**
  - Se descarga la imagen de MongoDB si no está en local
  - Inicia los dos contenedores

Ejemplo Docker Compose

- **Arrancar la aplicación**
  - La app está disponible en http://locahost:5000/



Ejemplo Docker Compose

- **¿Cómo funciona?**
  - Se muestran los logs de todos los contendores
  - Para parar la app, **Ctrl+C** en la consola (también **docker compose stop**)
  - Si se para y arranca de nuevo el servicio sin cambiar la configuración de un contenedor, **se vuelve a iniciar el mismo** (no se crea uno nuevo)
  - Los datos de la base de datos no se pierden ya que los estamos guardando en una carpeta del Host

Docker Compose

- Introducción
- Ejemplo Docker Compose
- **Espera de servicios**
- Herramientas para imágenes
- Plugin VSCode para Docker Compose
- Beneficios de Docker Compose

Espera de servicios

- **¿Qué pasa si nuestra aplicación necesita esperar a un servicio?**
  - Tendremos que utilizar una de las siguientes estrategias:
    - 1 - Reiniciar el contenedor
    - 2 - Utilizar una utilidad para esperar el servicio
    - 3 - Healthcheck

https://docs.docker.com/compose/startup-order/

Reiniciar contenedor

- Por defecto los servicios de Docker Compose no se reinician bajo ninguna circunstancia
- Este comportamiento se puede modificar haciendo uso de la etiqueta **restart**
- Si permitimos que una aplicación que depende de otro servicio pueda reiniciarse ante un fallo **(Error de conexión)**, conseguiremos que tarde o temprano conecte con el servicio del cual depende

https://docs.docker.com/compose/compose-file/#restart

Reiniciar contenedor

ejemplo-1-node

restart

Especifica cuándo se debe reiniciar el servicio. Por defecto nunca se reinicia un servicio

Si le definimos el valor **“on-failure”** el servicio se reiniciará siempre que la aplicación finalice con un error

docker-compose-restart-on-failure.yml

```yaml
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
        - MONGODB_HOST=db
        - MONGODB_PORT=27017
        - MONGODB_COLLECTION=posts
    depends_on:
      - db
    restart: on-failure
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

https://docs.docker.com/compose/compose-file/#restart

Utilidad de espera

- Existen herramientas como **"wait-for-it.sh"** que comprueban si existe conectividad en una **Ip** y **Puerto**
- **"wait-for-it.sh"** estará constantemente intentando detectar conectividad (el tiempo entre intentos se puede configurar)
- Cuando **"wait-for-it.sh"** detecta conectividad ejecutará el comando que le indiquemos, como por ejemplo levantar una aplicación de (Node/Java/Python)

https://github.com/vishnubob/wait-for-it

Utilidad de espera

- ¿Cuales son los pasos para utilizar la herramienta **"wait-for-it"**?
  - 1 - Añadir la herramienta dentro de la imagen de nuestra aplicación
    - La imagen que utilicemos deberá disponer de Bash instalado
  - 2 - Sobreescribir el comando de ejecución en el Docker Compose

https://github.com/vishnubob/wait-for-it

Añadir “wait-for-it” dentro de la imagen

ejemplo-1-node

wait-for-it.Dockerfile

```
# Imagen base para el contenedor de compilación
FROM node:lts-alpine as builder
WORKDIR /usr/src/app/
COPY package.json /usr/src/app/
RUN npm install --only=production

# Imagen base para el contenedor de la aplicación
FROM node:lts
ENV NODE_ENV production
WORKDIR /usr/src/app
RUN curl -LJO
https://raw.githubusercontent.com/vishnubob/wait-for-it/master/wait-for-it.sh \
    && chmod +x /usr/src/app/wait-for-it.sh
COPY --from=builder /usr/src/app/node_modules /usr/src/app/node_modules
COPY src /usr/src/app/src
EXPOSE 5000
CMD [ "node", "src/server.js" ]
```

https://github.com/vishnubob/wait-for-it

command

Sobreescribimos el comando por defecto de la imagen y utilizamos el script **“wait-for-it”** indicando que debe esperar al servicio de la base de datos **“db”** en el puerto **27017** y después ejecutar el comando **node** para levantar la aplicación

docker-compose-wait-for-it.yml

```yaml
services:
  web:
    build:
      context: .
      dockerfile: wait-for-it.Dockerfile
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      - db
    command: [ "./wait-for-it.sh", "db:27017", "--", "node", "src/server.js" ]
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
```

Utilidad de espera

- ¿Podemos utilizar **"wait-for-it.sh"** con **JIB**?
  - **wait-for-it.sh** debe estar descargado en el proyecto
  - Hay que modificar el fichero **pom.xml** para indicar a JIB que copie el Script dentro de la imagen
  - Para las versiones antiguas de JIB es necesario cambiar la imagen **distroless** por una imagen con Shell

https://github.com/vishnubob/wait-for-it

Utilidad de espera en imágenes con JIB

- **Modificamos el POM**: Copiando la **herramienta** y añadiendo **permisos** de ejecución

```
<plugin>
     <groupId>com.google.cloud.tools</groupId>
     <artifactId>jib-maven-plugin</artifactId>
     <version>3.3.1</version>
     <configuration>
          <extraDirectories>
               <paths>
                    <path>src/main/resources/scripts</path>
               </paths>
               <permissions>
                    <permission>
                         <file>/wait-for-it.sh</file>
                         <mode>755</mode>
                    </permission>
               </permissions>
          </extraDirectories>
     </configuration>
</plugin>
```

https://github.com/GoogleContainerTools/jib/tree/master/jib-maven-plugin#adding-arbitrary-files-to-the-image

Utilidad de espera en imágenes con JIB

- En Quarkus no es necesario modificar el fichero **pom.xml**
- Crearemos la carpeta **"src/main/jib"**
- Todos los ficheros que añadamos a la carpeta serán copiados en la **raíz "/"** de la imagen resultante
- Los ficheros se copiarán con los **mismos permisos** que tengan en la carpeta **"src/main/jib"**

Desde **Windows** no podemos marcar como **ejecutable** un **Script**, por lo tanto se **copiará** dentro de la imagen **sin estos permisos** y no se podrá ejecutar. En **Linux** no existe esta limitación.

https://quarkus.io/guides/container-image#jib

Healthcheck

- En docker compose podremos utilizar la propiedad **healthcheck** con la cual definiremos si un servicio está **"sano" o no**
- La propiedad **healthcheck** también se puede definir en el fichero **Dockerfile**
- Si utilizamos la propiedad en el fichero **docker compose** y el **healthcheck** está especificado en la **imagen** este **será sobrescrito**

https://docs.docker.com/compose/compose-file/#healthcheck

Healthcheck

- Para utilizar la propiedad **healthcheck** debemos especificar un **comando** que chequee si el servicio está **"sano"**
- También podremos especificar el intervalo de chequeo, el timeout, el número de intentos...

https://docs.docker.com/compose/compose-file/#healthcheck

Healthcheck

code URJC

ejemplo-1-node

- **test:** comando que chequea la **"salubridad"** del servicio
- **interval:** tiempo entre chequeos
- **timeout:** tiempo de espera para la respuesta del chequeo
- **start_period:** tiempo de inicio de los chequeos
- **retries:** número de intentos cuando el chequeo falla

docker-compose-healthcheck.yml

```yaml
db:
  image: mongo:5.0-focal
  volumes:
    - ./mongo:/data/db
  healthcheck:
    test: [ "CMD", "mongo", "--eval", "db.adminCommand('ping')" ]
    interval: 5s
    timeout: 5s
    start_period: 10s
    retries: 5
```

https://docs.docker.com/compose/compose-file/#healthcheck

Healthcheck

- En docker compose podemos indicar el orden de inicio de los servicio utilizando la propiedad **depends_on**
- Desde la ultima especificación de Docker Compose es posible indicar una **condición** de espera en la propiedad **depends_on**
- Combinando la propiedad **healtcheck** junto a la propiedad **depends_on** podremos esperar a que un servicio esté correctamente iniciado para iniciar otro

https://docs.docker.com/compose/startup-order/

Healthcheck

- En la propiedad **depends_on** podremos indicar varias condiciones de espera para cada servicio:
  - **service_started:** Esta es la propiedad por defecto, inicia una vez que el servicio del cual depende es creado
  - **service_healthy:** Inicia una vez que el servicio del cual depende este en un estado de **"sano"**
  - **service_completed_successfully:** Inicia una vez que el servicio del cual depende a terminado correctamente

https://docs.docker.com/compose/compose-file/#depends_on

Healthcheck

code URJC

ejemplo-1-node

docker-compose-healthcheck.yml

```yaml
services:
  web:
    build: .
    ports:
      - 5000:5000
    environment:
      - MONGODB_HOST=db
      - MONGODB_PORT=27017
      - MONGODB_COLLECTION=posts
    depends_on:
      db:
        condition: service_healthy
  db:
    image: mongo:5.0-focal
    volumes:
      - ./mongo:/data/db
    healthcheck:
      test: [ "CMD", "mongo", "--eval", "db.adminCommand('ping')" ]
      interval: 5s
      timeout: 5s
      start_period: 10s
      retries: 5
```

https://docs.docker.com/compose/compose-file/#depends_on

Docker Compose

- Introducción
- Ejemplo Docker Compose
- Espera de servicios
- **Herramientas para imágenes**
- Plugin VSCode para Docker Compose
- Distribución de aplicaciones Dockerizadas

Herramientas para imágenes

- **dockerize**
  - Librería para imágenes Docker
    - Creación de configuraciones en tiempo de ejecución usando plantillas
    - Control de la salida de error en múltiples ficheros
    - Espera de servicios

```
RUN apt-get update && apt-get install -y wget

ENV DOCKERIZE_VERSION v0.6.1
RUN wget https://github.com/jwilder/dockerize/releases/download/$DOCKERIZE_VERSION/dockerize-linux-amd64-
$DOCKERIZE_VERSION.tar.gz \
    && tar -C /usr/local/bin -xzvf dockerize-linux-amd64-$DOCKERIZE_VERSION.tar.gz \
    && rm dockerize-linux-amd64-$DOCKERIZE_VERSION.tar.gz
```

https://github.com/jwilder/dockerize

Docker Compose

- Introducción
- Ejemplo Docker Compose
- Espera de servicios
- Herramientas para imágenes
- **Plugin VSCode para Docker Compose**
- Distribución de aplicaciones Dockerizadas

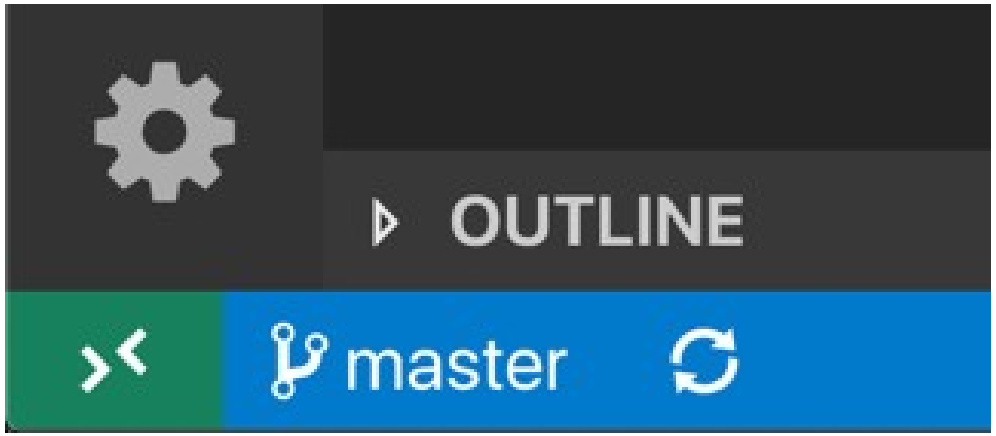
VSCode Docker Compose

- **Desarrollar usando un docker-compose**
  - 1 - Click en el icono de desarrollo remoto
  - 2 - Click en "Remote-Containers: Reopen in Container"
  - 3 - Click en From "docker-compose.yml"
  - 4 - Seleccionar el servicio

https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers

VSCode Docker Compose

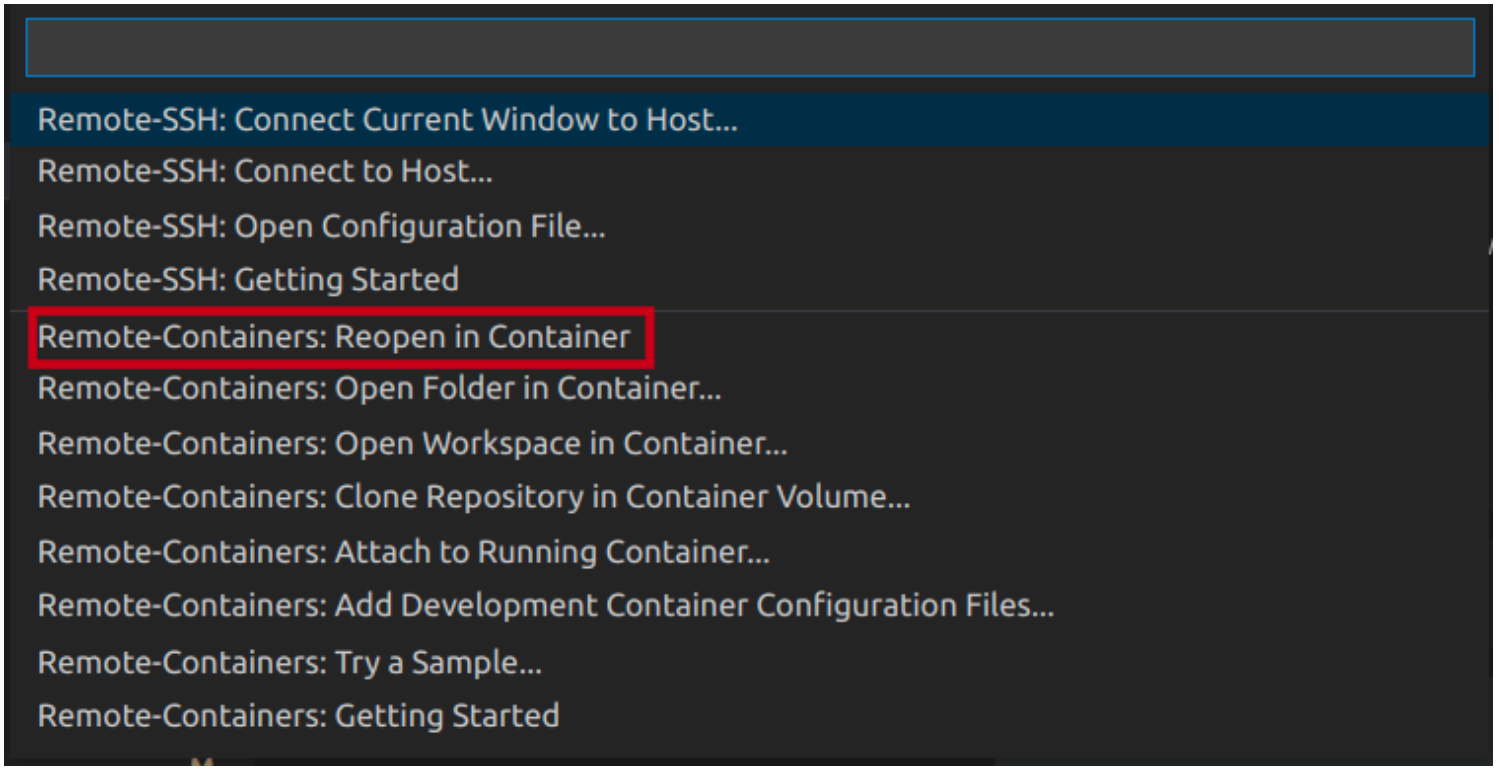
1 – Click en el icono de desarrollo



https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers

VSCode Docker Compose

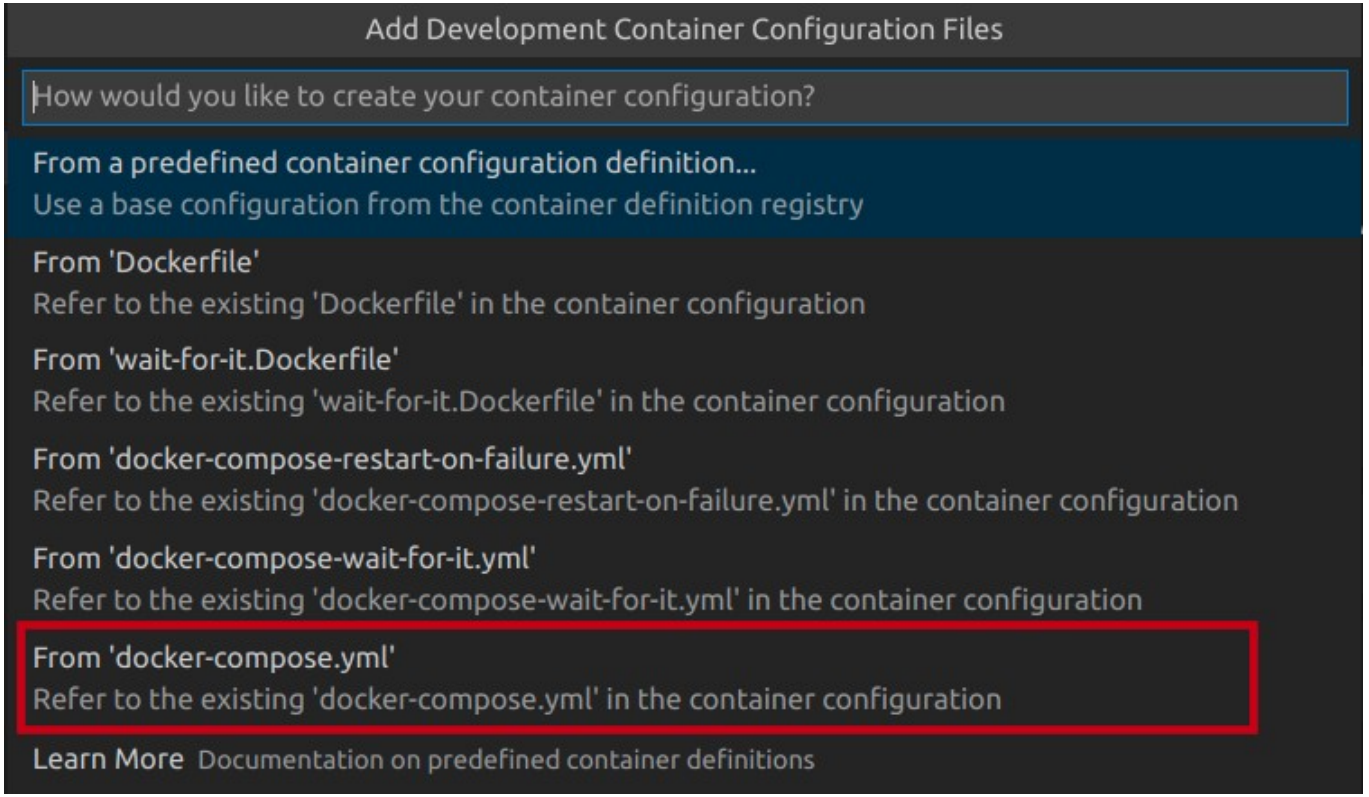
2 – Click en "Remote-Containers: Reopen in container"



https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers

VSCode Docker Compose

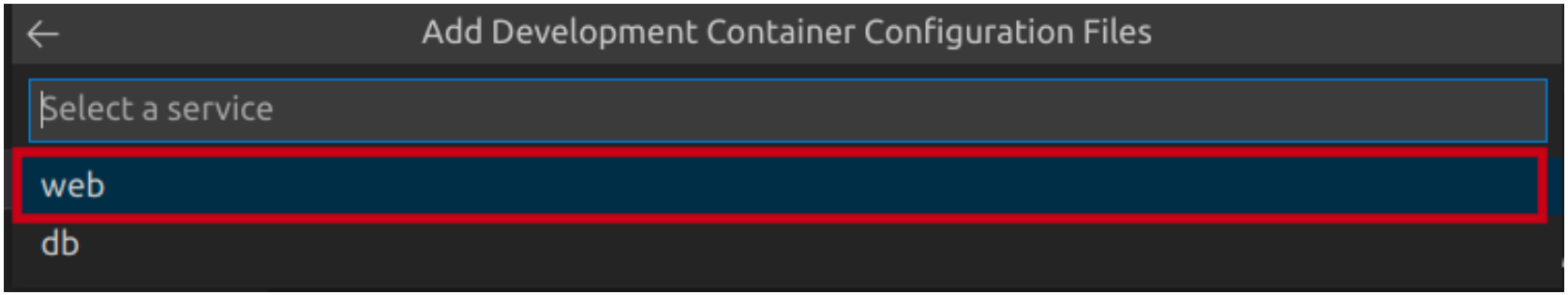
3 – Click en From “docker-compose.yml”



https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers

VSCode Docker Compose

4 – Seleccionar el servicio



https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers

VSCode Docker Compose

- Todos los servicios definidos en el **docker-compose** menos el seleccionado arrancarán
- Las variables de entorno se **inyectan**
- Los puertos definidos se **bindean**
- Podremos programar en el contenedor y conectarnos al resto de **servicios** del Docker Compose

https://code.visualstudio.com/docs/remote/containers#_attaching-to-running-containers

Docker Compose

- Introducción
- Ejemplo Docker Compose
- Espera de servicios
- Herramientas para imágenes
- Plugin VSCode para Docker Compose
- **Beneficios de Docker Compose**

Beneficios de Docker Compose

- **Ideal para desarrollo**
  - Podemos definir una app con múltiples contenedores en un fichero de texto (y subirlo a un repositorio)
  - Cualquier desarrollador puede arrancar la app sin tener nada instalado en local (sólo docker y docker-compose)
  - Es muy cómodo iniciar y parar todos los servicios a la vez y sólo cuando realmente se necesitan (no tienen que estar iniciados al arrancar la app)
  - Todos los logs centralizados

Beneficios de Docker Compose

- **Distribución de apps dockerizadas**
  - Si todos los contenedores del compose están en DockerHub, para distribuir una app multicontenedor dockerizada basta con descargar el docker-compose.yml y arrancarlo.
  - Con wget y el docker-compose.yml en github:

```
$ wget https://git.io/JIkMh -O docker-compose.yml
$ docker compose up -d
```

  - Con curl disponible y el docker-compose.yml en github:

```
$ curl -L https://git.io/JIkMh | docker compose -f - up -d
```

Ejercicio 1

- **Dockerizar una aplicación SpringBoot**
  - Se usará docker-compose
  - La aplicación necesita una BBDD MySQL
    - Password de root: pass
  - Utiliza una de las estrategias de espera vistas
  - En una aplicación SpringBoot se puede configurar la ruta de la BBDD y el esquema con la variable de entorno:
    - SPRING_DATASOURCE_URL=jdbc:mysql://<host>/<database>

https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestadores/tree/master/docker-compose/ejercicio-1-enunciado