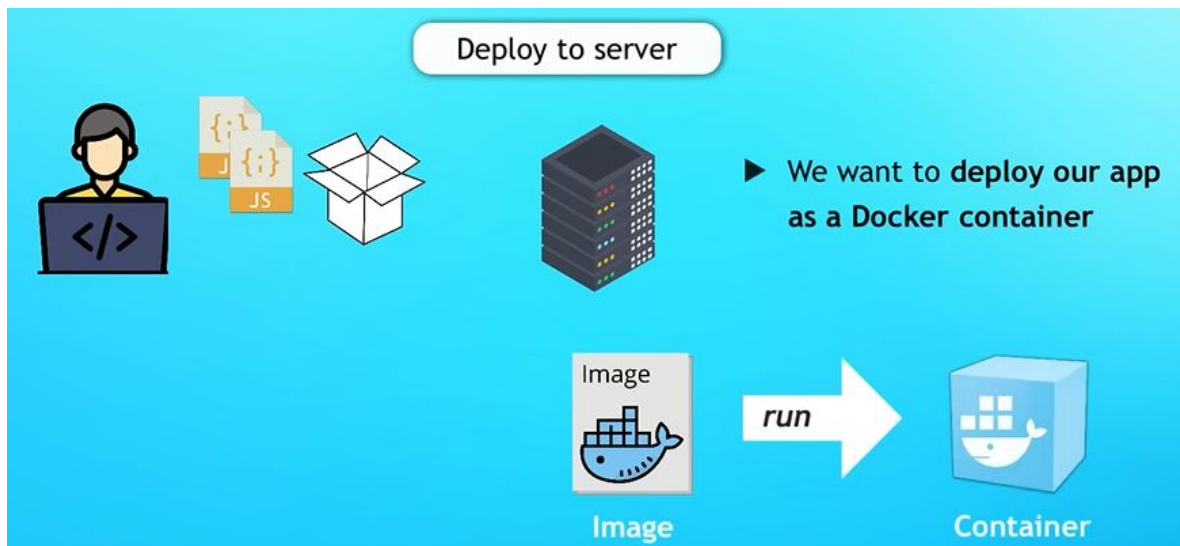


# Docker - Build

Dockerize

# Desplegar una aplicación sobre un servidor

Con Docker, una vez desarrollada una aplicación, se debe **desplegar en el servidor dentro de un contenedor**. Para esto, el programa y todos los recursos necesarios para que funcione se deben empaquetar en una imagen. En el servidor se pueden **desplegar más contenedores necesarios para el funcionamiento de la aplicación** (pe. Base de datos mysql).



# Crear una imagen de aplicación

Se debe definir cómo se ha de construir la imagen: para ello se genera un fichero **“Dockerfile”** que contiene las instrucciones de montaje de la imagen.




# Ejemplo aplicación node.js

## tutorial/src/server.js:

```
const express = require('express');  
const app = express();  
app.get('/', (req,res)=>{ res.send("¡Hola Mundo!"); });  
app.listen(3000, function () { console.log( "escuchando puerto 3000" ); } );
```

## tutorial/package.json:

```
{  
  "name": "mi-app",  
  "version": "1.0",  
  "dependencies": {  
    "express": "4.18.2"  
  }  
}
```



# Ejemplo dockerfile para aplicación node.js

## **tutorial/Dockerfile:**

FROM node:19-alpine

COPY package.json /app/

COPY src /app/

WORKDIR /app

RUN npm install

CMD ["node", "server.js"]



# Estructura de Dockerfile - FROM

Los Dockerfiles comienzan haciendo uso de una “**imagen base**”. Esta imagen contiene, como mínimo, una **versión de linux ligera** sobre la que todo se va a ejecutar.

**Este SO es el que tendrá instaladas las herramientas que se vaya a emplear.**

La elección de esta imagen base dependerá de dichas herramientas.

Por tanto, los **Dockerfiles deben comenzar con una instrucción FROM.**



# Estructura de Dockerfile - FROM

El concepto de apilar imagen sobre imagen significa que se irán formando **múltiples capas**. Este aporta el beneficio de poder tener imágenes en caché.



En el ejemplo:  
*FROM node:19-alpine*

# Estructura de Dockerfile - RUN

Mediante RUN en Dockerfile se indican comandos a ejecutar al construir el contenedor.

La directiva FROM declarada en el ejemplo anterior incluye npm, pero se requiere instalar en el SO, por tanto, se debe indicar la instrucción:

*RUN npm install*





# Estructura de Dockerfile - COPY

**Copia directorios** o ficheros y los añade al sistema de ficheros del contenedor, dentro de la ruta “**dest**”.

A diferencia de RUN que ejecuta comandos dentro del contenedor, **COPY se ejecuta en la máquina HOST**.

Se puede indicar una ruta específica donde copiar. Si esta ruta termina en /, Docker interpreta que debe crear las carpetas en el contenedor si no existen.

*COPY package.json /app/*

*COPY src /app/*



# Estructura de Dockerfile - WORKDIR

**WORKDIR** sirve para indicar el directorio de trabajo. Especifica donde se van a ejecutar los comandos que se declaren a continuación.

Equivale a cambiar de directorio ("cd").

*WORKDIR /app*




# Estructura de Dockerfile - CMD

La instrucción CMD indica el comando que se debe ejecutar cuando se arranque el contenedor.

**Solo puede haber una instrucción CMD en un Dockerfile**, y, por tanto, normalmente será siempre la última a declarar.

Se le debe pasar, en forma de parámetros de un array, el comando y su valor.

*CMD ["node", "server.js"]*



# Construir la imagen - Build

Una vez creado el fichero Dockerfile, se puede construir la imagen mediante el comando **build** de Docker, **indicando la ruta** donde se encuentra el fichero:

```
docker build [Ruta_de_Dockerfile]
```

El flag **-t** o **--tag** sirve para darle un **nombre** a la imagen **y** (opcionalmente) una **etiqueta de versión**, empleando el formato “nombre:tag”.

Para generar la imagen de “mi-app” se puede emplear:

```
docker build -t mi-app:1.0 .
```

El punto (.) indica que se utilice la ruta desde donde se ejecuta el comando.




# Estructura de Dockerfile y Construir la imagen

La ejecución de Dockerfile es *secuencial*; **cada instrucción crea una capa o estado dentro de la imagen.**

Esto se puede ver claramente en los logs de la construcción de la imagen, donde cada instrucción declarada es un paso con sus propios registros.

*Recordatorio:* usar “docker images” para ver las imágenes descargadas o creadas.



# Run de mi-app

Ahora que se ha generado la imagen, es posible crear el contenedor y ejecutar el programa mediante el comando RUN:

***docker run -d -p 3000:3000 mi-app:1.0***

Ahora **localhost:3000** debería mostrar “¡Hola Mundo!” vía navegador.

Ver también los **docker logs** para este contenedor; debería mostrar “escuchando puerto 3000”.



# Ciclo de vida

