

Bienvenidos

Clase 06.
Aplicaciones Móviles y Cloud Computing

Desarrollo FullStack

Temario

05

Validaciones PHP Archivos PHP Conexión DB

- ✓ Validaciones con PHP
- ✓ Manejo básico archivos con PHP
- ✓ Conexión a DB con PHP

06

Desarrollo FullStack

- ✓ [Conceptos de desarrollo FullStack](#)
- ✓ [Stacks de programación](#)
- ✓ [SOLID - STUPID](#)

07

Desarrollo de aplicaciones Restfull

- ✓ Arquitectura REST
- ✓ CRUD con NodeJS y Express
- ✓ Middlewares / Routers



Objetivos de la clase

- Comprender los conceptos esenciales de FullStack
- Repaso de tecnologías básicas necesarias para FullStack
- Buenas prácticas: SOLID – STUPID

Desarrollo FullStack



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

¿Qué significa FullStack?

- ✓ Un desarrollador que trabaja como FullStack, trabaja con proyectos de FrontEnd y con proyectos de Backend
- ✓ Como los desarrolladores full-stack trabajan tanto en proyectos de frontend como de backend, necesitan utilizar en su trabajo lenguajes y herramientas de programación tanto de frontend como de backend



FrontEnd – Backend:

FRONTEND

Si el desarrollo fuese construir un restaurante, los frontend serían responsables de **construir la parte que es visible, de la atención a los clientes y de coordinar con la cocina.**

Principales funciones:

1. Tomar los pedidos de los clientes
2. Pasar los pedidos a la cocina.
3. Recibir los pedidos de la cocina.
4. Servir los pedidos a los clientes.

Principales desafíos:

Adaptarse a todo los tipos clientes.
Atender rápidamente a los clientes.
Lograr que los **clientes realicen pedidos.**



Se realizan pedidos a la cocina

La cocina entrega los pedidos

BACKEND

Si el desarrollo fuese construir un restaurante, los backend serían responsables de **construir la cocina, preparar los pedidos y dejarlos listos para que el front los sirva al cliente**

Principales funciones:

1. Tomar los pedidos pasado por front.
2. Tomar los ingredientes
3. Preparar los pedidos.
4. Entregar los pedidos.

Principal desafíos:

Preparar de forma **segura.**
Prepara **varios pedidos a la vez.**
Disminuir el **tiempo de preparación.**



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

FrontEnd – Backend:

FRONTEND

Se encarga de escribir el código para **hacer visible el sitio**, que permite la **interacción con los usuarios** y de la **comunicación con el Backend**.

Principales funciones:

1. Tomar los pedidos de los usuarios.
2. Pasar los pedidos (como datos) al Backend.
3. Tomar los pedidos (datos) desde el backend.
4. Mostrar (visualizar) los datos a los clientes.

Principales desafíos:

Adaptarse a los distintos navegadores.
Desplegar rápidamente el sitio (Performance)
Lograr que los **clientes realicen pedidos** (usabilidad).



Se realizan pedidos a los servidores

Los servidores entregan los pedidos

BACKEND

Se encarga de escribir el código que **crea o utiliza los servicios que dan respuesta a lo pedidos** y de la **comunicación con el frontend**.

Principales funciones:

1. Tomar los pedidos (datos) del frontend.
2. Acceder a los datos o servicios para realizar los pedidos.
3. Realizar los pedidos.
4. Entregar las pedidos (datos) al frontend


Principal desafíos:

Garantizar la **seguridad del sitio**.
Atender y resolver **varios pedidos a la vez**.
Optimizar los **tiempo de respuesta (performance)**.





Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

¿Qué hace un desarrollador Full-Stack?

- ✓ Diseñar la arquitectura de nuevos sitios web y programas basados en la web. 
- ✓ Desarrollar interfaces de programación de aplicaciones (API). Las API son intermediarios de software que ayudan a las aplicaciones a comunicarse entre sí.
- ✓ Trabajar con los usuarios para entender sus necesidades tecnológicas al diseñar nuevos sitios web o programas.
- ✓ Desarrollar actualizaciones para mejorar la [usabilidad](#) y las capacidades del sistema.
- ✓ Mantenimiento del frontend de un sitio web para asegurarse de que funciona.



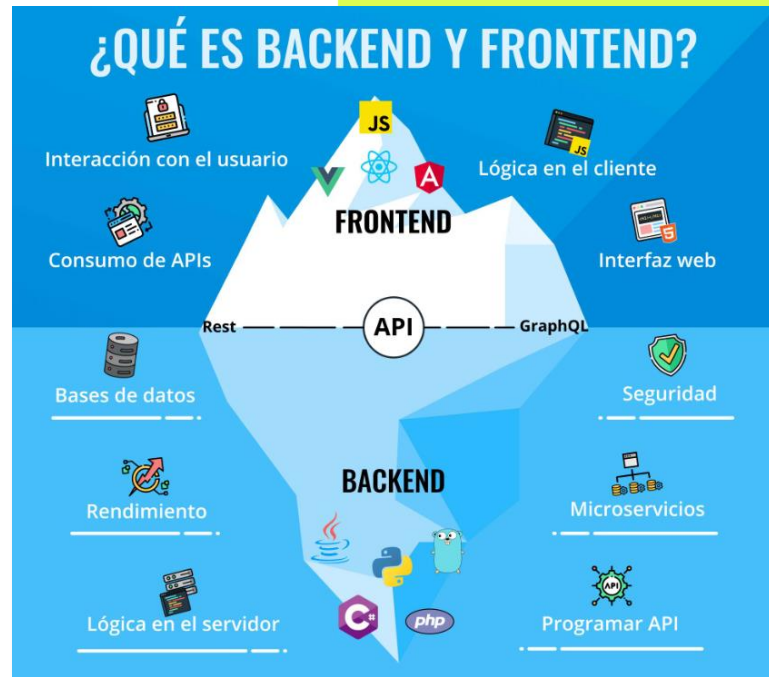
¿Qué hace un desarrollador Full-Stack?

- ✓ Optimizar los sitios web para asegurarse de que mantengan y mejoren sus prestaciones. Ej.: Artillery
- ✓ Trabajar con diseñadores gráficos para desarrollar sitios web atractivos y funcionales.
- ✓ Crear servidores y bases de datos para los sitios web.
- ✓ Supervisar un proyecto a través de cada fase del ciclo de vida de desarrollo de sistemas. 
- ✓ Trabajar con otros profesionales (como ingenieros de DevOps , expertos en ciberseguridad, diseñadores gráficos, y otros desarrolladores web).



Habilidades necesarias para ser FullStack

- ✓ Manejo de lenguajes Frontend: HTML, CSS, Javascript.
- ✓ TypeScript
- ✓ Experiencia con frameworks y librerías de Frontend: Bootstrap, React, JQuery, Angular
- ✓ Capacidad de codificar con lenguajes de backend: PHP, NodeJS, Python, Java, Ruby.
- ✓ Experiencia con frameworks de backend: Laravel, Spring, Django.
- ✓ Habilidades de base de datos: MySQL, MongoDB
- ✓ Experiencia con stacks populares
- ✓ Manejo de herramientas de desarrollo (entornos IDE, Git, etc.)
- ✓ Habilidades blandas



Lenguajes de programación Fullstack



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Categorías Lenguajes de Programación

- ✓ Los **lenguajes de programación**: pueden utilizarse en varias plataformas y suelen estar compilados en lugar de interpretados.
- ✓ Los **lenguajes de scripting**: suelen ser interpretados, lo que significa que su código se ejecuta sobre la marcha en lugar de pasar por un proceso de compilación. Los lenguajes de desarrollo web suelen ser lenguajes de scripting.
- ✓ Los **lenguajes de marcado**: no son precisamente lenguajes de programación. Son etiquetas legibles por el ser humano que se utilizan para dar formato a un documento.
- ✓ Los **lenguajes de desarrollo web** están especializados en la creación de sitios web, ya sea en el **frontend** o en el **backend**

![\"Close](\"img/icon_close.png\" "\\"Close\\"><img")

Lenguajes Fullstack:

- ✓ HTML y CSS
- ✓ JavaScript
- ✓ PHP
- ✓ C++ y C#
- ✓ Java
- ✓ Kotlin
- ✓ Rust
- ✓ Swift
- ✓ Python
- ✓ Scala
- ✓ Go



[Roadmap FullStack](#)

Buenas prácticas Clean Code



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región



La mayoría de los sistemas funcionan mejor si se mantienen simples en lugar de hacerlos complejos.

- ✓ No tiene sentido aumentar infinitamente el nivel de abstracción, hay que saber detenerse a tiempo
- ✓ No tiene sentido poner en el proyecto funciones redundantes «en reserva» que algún día alguien pueda necesitar.
- ✓ Para qué incluir una biblioteca enorme si solo se necesita un par de funciones de ella
- ✓ La descomposición de algo complejo en componentes simples es un enfoque arquitectónicamente correcto (aquí KISS se hace eco de DRY)
- ✓ No siempre se necesita precisión matemática absoluta o detalles extremos: Los datos pueden y deben procesarse con la precisión suficiente para una solución de alta calidad del problema, y los detalles se dan en la cantidad necesaria para el usuario, y no en el volumen máximo posible.





DRY nos recuerda que cada comportamiento repetible en el código debe estar aislado (por ejemplo, separado en una función) para su reutilización.

Si el código no está duplicado, para cambiar la lógica, basta con hacer correcciones en un solo lugar. Además, es más fácil probar una función puntual, aislada, en lugar de código suelto, perdido dentro de nuestro desarrollo. **Simplifica Testing**

Seguir este principio siempre conduce a la descomposición de algoritmos complejos en funciones simples. Y la descomposición de operaciones complejas en operaciones más simples (y reutilizables) simplifica enormemente la comprensión del código del programa. Incrementa **Mantenibilidad**, mejora la **Calidad**

El acceso a una funcionalidad específica debe estar disponible en un solo lugar, unificado y agrupado de acuerdo con algún principio, y no «disperso» dentro del sistema. Este enfoque se cruza con el principio de responsabilidad única, de los cinco principios SOLID



NO seas STUPID

Singleton

Tight Coupling

Untestability

Premature
Optimization

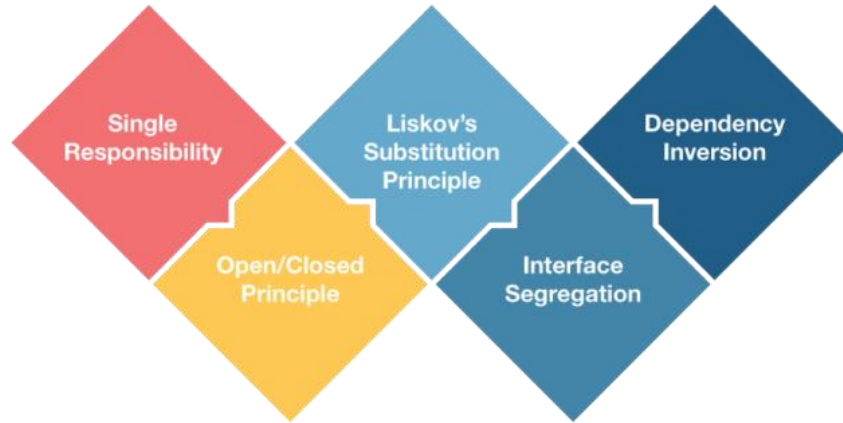
Indescriptive
naming

Duplication

- ✓ Implementación de patrón Singleton
- ✓ Código fuertemente acoplado
- ✓ Imposibilidad de realizar testing
- ✓ Optimización prematura
- ✓ Nombres poco descriptivos
- ✓ Duplicidad



S.O.L.I.D.



- ✓ Responsabilidad única
- ✓ Principio Abierto Cerrado
- ✓ Principio de sustitución de Liskov
- ✓ Principio de segregación de interfaces
- ✓ Principio de inversión de dependencias

¿Preguntas?



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Muchas gracias.



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Ciclo de vida de un desarrollo de software



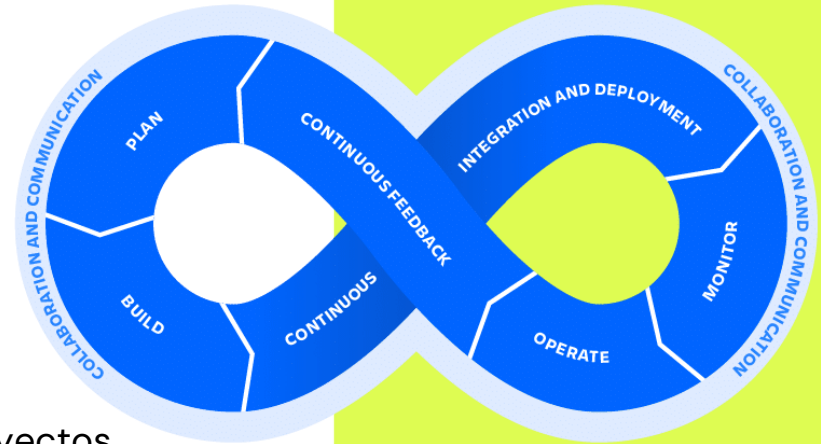
¿Qué es DevOps?

Un ingeniero de DevOps es un profesional de TI que trabaja con los desarrolladores de software, para optimizar el despliegue de nuevas actualizaciones y programas.

Los ingenieros de DevOps aplican una versión del SDLC conocida como ciclo de vida de DevOps, que hace un bucle y se repite continuamente a través de estas etapas:

- ✓ Planificación
- ✓ Retroalimentación continua
- ✓ Funcionamiento
- ✓ Integración continua y despliegue continuo
- ✓ Construcción

El enfoque DevOps guía a los desarrolladores en la construcción y lanzamiento de actualizaciones y proyectos de forma incremental («Integración Continua/Entrega Continua» o CI/CD)



¿Qué responsabilidades tiene el personal de DevOps?

- ✓ Gestión de proyectos
- ✓ Gestión de la seguridad del sistema
- ✓ Mejoras en la infraestructura de IT
- ✓ Automatización de tareas repetitivas
- ✓ Evaluación comparativa y pruebas de rendimiento
- ✓ Optimización de ciclos de lanzamiento
- ✓ Supervisión y notificación de errores.



¿Qué habilidades requiere?

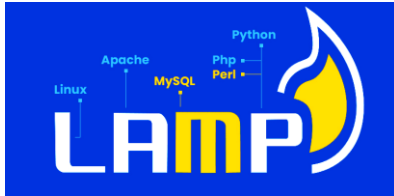
- ✓ Experiencia con herramientas de automatización ([Jenkins](#))
- ✓ Manejo de sistemas de control de versiones
- ✓ Experiencia con sistemas de alojamiento de repositorios
- ✓ Manejo de herramientas de gestión de la configuración ([Puppet](#) o [Chef](#))
- ✓ Experiencia con software de monitoreo ([Nagios](#))
- ✓ Habilidades de codificación: tanto de scripting, como de programación
- ✓ Experiencia en herramientas de contenerización ([Docker](#), [Kubernetes](#))
- ✓ Experiencia con herramientas de Gestión de Proyectos ([Jira](#), [Trello](#))
- ✓ Experiencia con servicios Cloud: Azure, Google Cloud, y AWS
- ✓ Habilidades blandas: interacción con clientes, manejo de equipos de trabajo



¿Qué es un “stack”?

Son ciertas tecnologías que, en conjunto, nos brindarán la posibilidad de desarrollar sistemas completos, debido a su máxima compatibilidad.

Podríamos decir que es la forma en la que el FrontEnd y el backEnd hacen las pases, ya que trabajan en conjunto.



STACK DE DESARROLLO

IDE's



VS Code



Atom



Sublime
Text



PHPStorm



WebStorm



Eclipse



Android
Studio



XCode



Vim

Control de versiones



GitLab



Github



Bitbucket

Analizadores de sintaxis



PHP
CodeSniffer



PHPmd



ESLint



TSLint

Testing

PHP



PHPUnit

Javascript



Jasmine



Mocha



QUnit



Karma

Pruebas end2end



Selenium



Protractor



Nightwatchjs

Testeo de API's
y documentación



Postman



Apidocs

DevOps

Gestión de contenedores



Docker

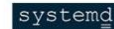


Docker-compose

Integración



Jenkins



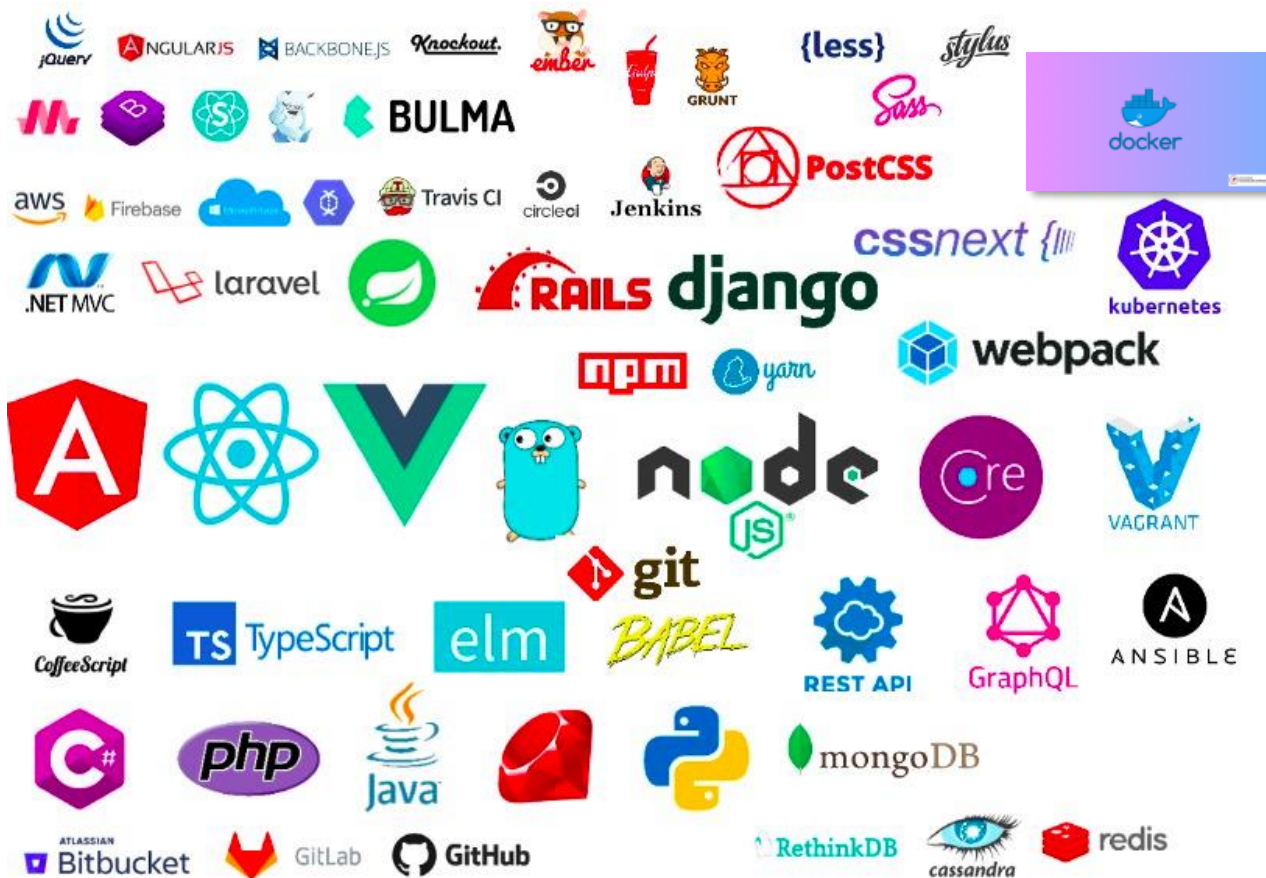
Systemd



Foreman



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

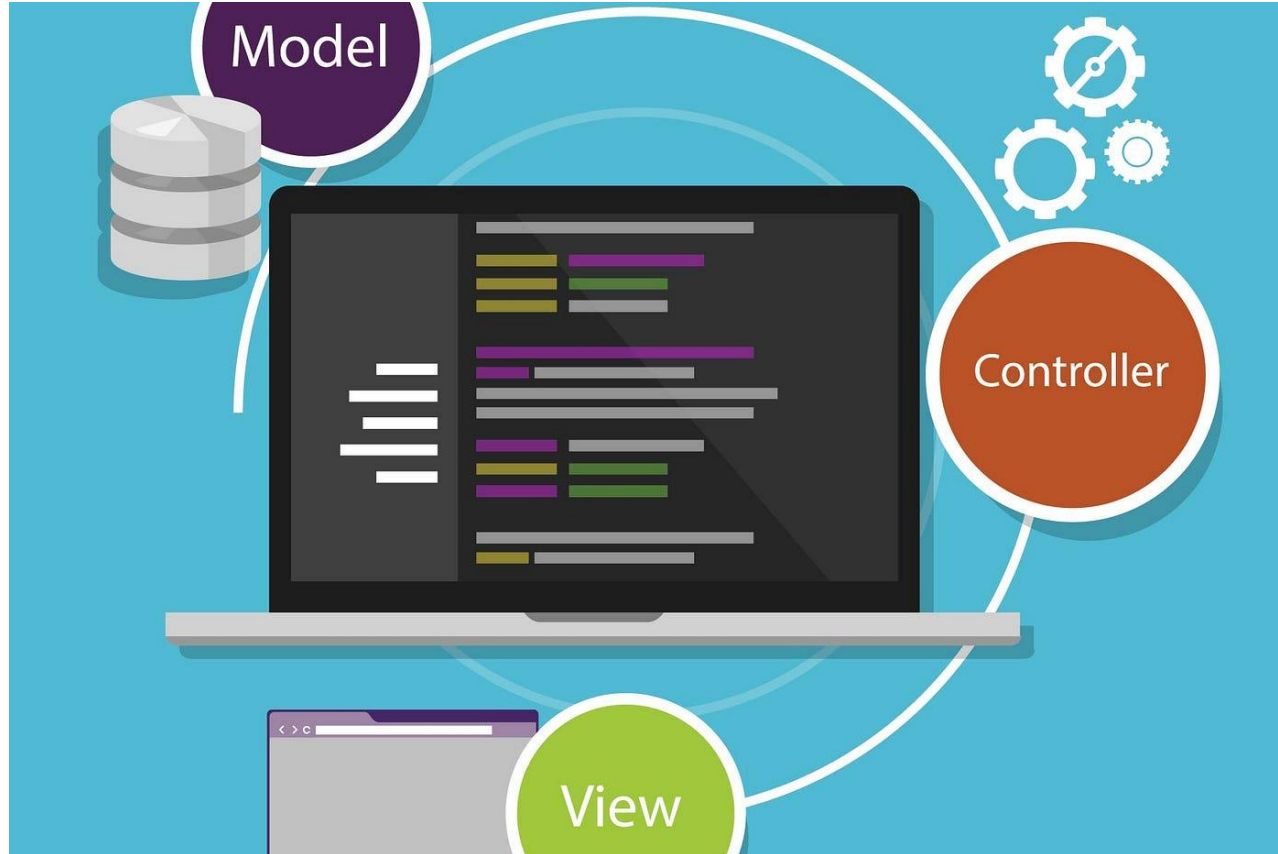


Herramientas de Desarrollo



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Modelo Vista Controlador





docker



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Docker

Docker es una plataforma que permitirá crear, probar e implementar aplicativos en unidades de software estandarizadas llamadas **contenedores**.

Con docker, podremos “virtualizar” el sistema operativo de un servidor con el fin de realizar ejecución de aplicaciones con la máxima compatibilidad.

Gracias a tener nuestro aplicativo en un contenedor que corra un software con exactamente las especificaciones que necesita esta app, evitamos el típico problema del desarrollador **“en mi computadora sí funcionaba”**

- En mi computadora si funciona
- Si pero no le vamos a dar tu computadora al cliente



[Instalar Docker Desktop](#)

[¿Qué es WSL? - Proceso de instalación](#)

Docker run

Docker run levanta imágenes (system images) en contenedores aislados. Su forma más genérica es la siguiente:

```
docker run [OPTIONS] IMAGE
```

[Documentación Oficial](#)

Opciones útiles:

- **--name:** permite especificar el nombre del contenedor a generar
- **-e:** permite definir variables de entorno; se utiliza el formato `-e VARIABLE=Valor_de_la_variable`
- **-p:** permite definir el puerto; recordar que la sintaxis es `-p n:m`, donde `n` es el puerto que se expone al exterior del contenedor, y `m` un puerto determinado característico de la imagen
- **-d:** quiere decir "detached". Ejecuta el contenedor en segundo plano. Si no se coloca este flag, al ejecutar Docker run, la terminal queda "tomada" por el contenedor.

Ejemplo:

```
docker run --name server-mysql -e MYSQL_ROOT_PASSWORD=123 -p 3306:3306 -d mysql
```



Docker exec

Este comando se utiliza para ejecutar un comando dentro de un contenedor en ejecución. Acompañado por el flag “**-it**”, permite interactuar con el comando ejecutado dentro del contenedor.

- **-i** mantiene abierta la entrada estándar (stdin) del contenedor,
- **-t** asigna una pseudo terminal (tty) para la interacción.

Por ejemplo, podemos ejecutar mysql en el contenedor recientemente creado de esta forma:

```
docker exec -it server-mysql mysql -uroot -p
```

Donde “**mysql -uroot -p**” es el comando que quiero ejecutar. Con mysql corro precisamente el gestor de base de datos. Con la opción **-uroot** indico que quiero ingresar como root. Con **-p** el sistema me va a solicitar la contraseña.

Ingresando 123 (la contraseña seteada en la variable de entorno al ejecutar Docker run), accedo al CLI de mysql, dentro del contenedor. Y puedo desde allí crear DB's, crear tablas, etc.



Para generar imágenes: Crear un archivo Dockerfile

FROM: define la imagen de la cual vamos a partir

WORKDIR: define el directorio de trabajo, dentro de la imagen

COPY: copia archivos desde el directorio actual, a la carpeta de la imagen

RUN nos permitirá ejecutar comandos. Al usar la imagen base **node**, significa que el entorno podrá correr comandos de node y npm sin problema.

EXPOSE: expone un puerto al exterior del contenedor, para que este pueda ser "tomado" por nuestra computadora

CMD al final es la ejecución del comando final que se utilizará al momento de echar a andar el servidor cuando hagamos **docker run**

```
Dockerfile x package.json
Dockerfile > ...
1 #Primero definimos una imagen base: node
2 FROM node
3
4 #Después creamos una carpeta interna donde vamos a guardar nuestro proyecto (usualmente es app)
5 WORKDIR /app
6
7 #Con esto, copiamos el package.json de nuestra carpeta actual, a la carpeta dockeroperations
8 COPY package*.json ./
9
10 #Una vez copiado el package.json, procedemos a ejecutar un npm install interno en esa carpeta.
11 RUN npm install
12
13 #Después de la instalación, procedemos a tomar todo el código del aplicativo
14 COPY . .
15
16 #Exponemos un puerto para que éste escuche a partir de un puerto de nuestra computadora.
17 EXPOSE 8080
18
19 #Una vez realizado, se deberá ejecutar "npm start" para iniciar la aplicación (ten listo el comando en tu package.json)
20 CMD ["npm","start"]
```

Ejecutamos el comando build

Una vez configurado nuestro respectivo dockerfile, podemos poner a prueba éste ejecutando el comando build.

```
docker build -t dockeroperations .
```

El comando build leerá el archivo y comenzará con la construcción de la imagen para nuestro aplicativo.

Una vez que tenga la imagen del aplicativo, necesita colocarle un nombre. El flag **-t** significa "tag" y es para nombrar la imagen.

El punto **.** sirve para indicarle que el **dockerfile** que necesitamos que lea está en la misma ubicación donde estamos corriendo el comando

```
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 32B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:latest
=> [1/5] FROM docker.io/library/node@sha256:743707dbaca64ff4ec8673a6e0c4d03d048e32b4e8ff3e89e
=> [internal] load build context
=> => transferring context: 27.69kB
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY package*.json ./
=> CACHED [4/5] RUN npm install
=> CACHED [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:a13f7eb192efbdf0d43b55a2ab79d0032a5d10c6e8ec1d76ac6fba834e6ce32c
=> => naming to docker.io/library/dockeroperations
```

Volumenes en Docker

Docker permite definir estructuras denominadas Volúmenes, que se asocian por un lado a un path de nuestro sistema de archivos, y por otro a determinado contenedor. De esta forma, podemos mantener información del contenedor, en nuestros directorios. Accesibles desde cmd, powershell, explorador de windows, etc.

Nos sirve entre otras cosas para persistir de manera simple los datos sensibles que tengamos en el contenedor. Además, permite interactuar fácilmente con el mismo, incluyendo archivos desde nuestro file-system. Archivos que luego podremos tomar sin inconvenientes desde el contenedor.

¿Cómo generamos un volumen nuevo en Docker? Así:

```
docker volume create Volumen01 --opt type=none --opt o=bind --opt device="C:\data\Volumen01"
```

El flag **--opt type=none**, especifica el tipo de volumen como "none". Significa que no se utilizará un controlador específico de Docker para el volumen. En otras palabras, el volumen no será gestionado internamente por Docker.

La opción **--opt o=bind** permite que los archivos y directorios dentro del volumen sean almacenados y accesibles en una ubicación específica del host



Docker run & Volúmenes

Para levantar un contenedor, indicando un volumen, y garantizar de esta forma la persistencia de ciertos datos (comúnmente los sensibles dentro de la imagen con la que estamos trabajando), se utiliza Docker run, con el flag **-v**. La sintaxis del mismo es "**-v NOMBRE_VOLUMEN/PATH_DENTRO_DEL_CONTENEDOR**". Completamos el ejemplo propuesto un par de pantallas atrás:

```
docker run --name server-mysql -e MYSQL_ROOT_PASSWORD=123 -p 3306:3306 -d -v Volumen01:/var/lib/mysql mysql
```

En este caso, la imagen con la que estamos trabajando es la de mysql (es el último argumento del comando, la palabra final). Y la carpeta donde almacena los archivos de base de datos es precisamente la que se indica en el comando: `/var/lib/mysql`



Docker run & MongoDB

De la misma forma en que configuramos con los comandos que se muestran en las pantallas anteriores un contenedor en donde corre un servidor de MySQL, podemos configurar un contenedor donde trabaje un server de MongoDB. Sería de esta forma:

```
docker volume create Volumen02 --opt type=None --opt o=bind --opt device="C:\data\Volumen02"
```

para crear un volumen. Y luego:

```
docker run --name server-mongo -p 27017:27017 -d -v Volumen02:/data/db mongo
```

Esto último para levantar un contenedor con nombre server-mongo, que exponga al exterior el puerto 27017 (el que utiliza MongoDB), de manera "detached", y asociando el volumen creado en el paso anterior. Parte de la imagen "mongo" (última palabra del comando).

