

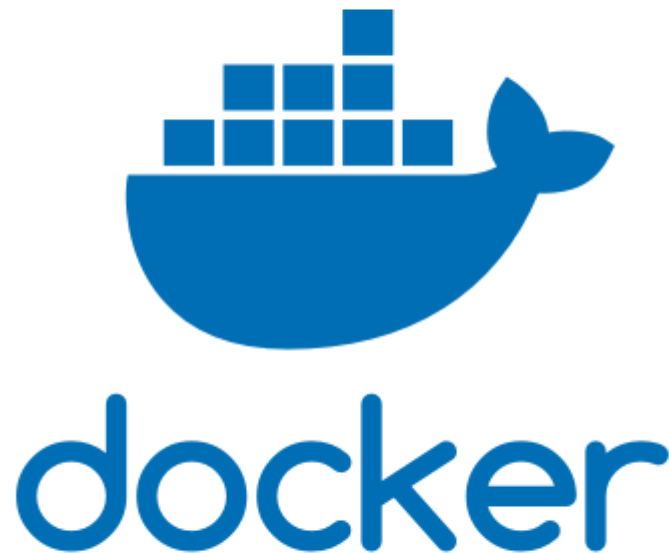
IES PÍO BAROJA  
1º DAW INTENSIVO

# **DOCKER Y DOCKER COMPOSE**

**+**

## **Entorno de desarrollo con Eclipse y MySQL**

Trabajo para el módulo de Entornos de desarrollo  
María Pilar Quintero y Vidal Torres



## ÍNDICE

<b>Conceptos Básicos de Docker</b>	<b>2</b>
<b>Ventajas de Usar Docker</b>	<b>2</b>
<b>Comparación con Máquinas Virtuales</b>	<b>3</b>
<b>Pasos para Configurar y Utilizar Docker</b>	<b>3</b>
1. Instalación de Docker	3
2. Gestión de Imágenes	3
3. Gestión de contenedores	8
<b>Los Logs en Docker</b>	<b>11</b>
Tipos de logs en Docker	11
Comandos básicos para trabajar con logs	12
<b>Uso Avanzado de Docker</b>	<b>14</b>
Redes en Docker	14
Volúmenes en Docker	14
Conexión de Contenedores con el Mundo mediante Puertos	16
Ejemplo Básico de Mapeo de Puertos	17
<b>Almacenar imágenes en ZIP</b>	<b>17</b>
<b>Tecnologías de Apoyo en el Entorno Docker</b>	<b>19</b>
Docker Compose	19
Docker Swarm	19
Kubernetes	19
Docker Machine	19
Docker Trusted Registry (DTR)	19
<b>Introducción a Docker Compose</b>	<b>19</b>
Ventajas de Docker Compose	20
Estructura de un Archivo Docker-Compose	20
<b>Limitaciones de Docker</b>	<b>25</b>
<b>Buenas Prácticas</b>	<b>25</b>
<b>Entorno de desarrollo con Eclipse y MySQL utilizando Docker Compose</b>	<b>26</b>

## Conceptos Básicos de Docker

Docker permite empaquetar aplicaciones y sus dependencias en contenedores. Esto asegura que las aplicaciones funcionen de manera uniforme en cualquier sistema que soporte Docker. Su arquitectura se basa en los siguientes componentes clave:

- **Docker Daemon:** Gestiona las imágenes, contenedores y redes de Docker.
- **Docker Client:** Proporciona una interfaz de usuario para interactuar con el Docker Daemon.
- **Docker Images:** Plantillas inmutables que contienen todo lo necesario para ejecutar una aplicación.
- **Docker Containers:** Instancias ejecutables de imágenes que operan de forma aislada.
- **Docker Hub:** Repositorio en línea para almacenar y compartir imágenes de Docker.

### *Ejemplo de Uso*

Para ejecutar un contenedor basado en una imagen de Ubuntu:

```
$ docker pull ubuntu  
$ docker run -it ubuntu /bin/bash
```

Este comando inicia un contenedor interactivo que utiliza la imagen de Ubuntu.

## Ventajas de Usar Docker

Docker ofrece múltiples beneficios que lo convierten en una herramienta esencial para el desarrollo moderno de software:

1. **Portabilidad:** Los contenedores pueden ejecutarse en cualquier máquina que tenga Docker instalado.
2. **Eficiencia de Recursos:** Comparte el kernel del sistema operativo host, utilizando menos recursos que las máquinas virtuales.
3. **Reproducibilidad:** Garantiza que las aplicaciones se comporten de manera idéntica en distintos entornos.
4. **Aislamiento:** Cada contenedor opera de forma independiente, evitando conflictos entre aplicaciones.
5. **Integración Continua:** Facilita la implementación de flujos de trabajo CI/CD.
6. **Flexibilidad:** Compatible con plataformas multi-nube como AWS, Azure y Google Cloud.

## Comparación con Máquinas Virtuales

Características	Docker	Máquina Virtual
Uso de recursos	Bajo	Alto
Tiempo de inicio	Segundos	Minutos
Portabilidad	Alta	Limitada
Aislamiento	Procesos	Sistemas operativos

## Pasos para Configurar y Utilizar Docker

### 1. Instalación de Docker

Docker está disponible para sistemas operativos como Linux, Windows y macOS. La instalación puede realizarse descargando el instalador desde [docker.com](https://docs.docker.com/get-docker/) o utilizando un gestor de paquetes.

Para confirmar que tenemos Docker correctamente instalado y que funciona sin problemas, podemos usar en git bash el comando `docker --version`

Antes de comenzar a trabajar en git bash, debemos levantar la aplicación de escritorio de Docker Desktop.

Después, una vez tenemos iniciado Docker, vamos a git bash con el primer paso, que será crear nuestro directorio de trabajo:

`mkdir mi_proyecto` : para crear el directorio

`cd mi_proyecto` : para acudir al directorio creado desde el que vamos a trabajar

`pwd` : para asegurarnos de que estamos en el directorio correcto

### 2. Gestión de Imágenes

#### 2.1 Descarga de Imágenes

Las imágenes pueden descargarse desde Docker Hub utilizando el comando `docker pull`. Por ejemplo:

`$ docker pull ubuntu`

Visión de hacer un pull en la consola:

```
MINGW64:/c/Users/mari1
mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
de44b265507a: Pull complete
Digest: sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

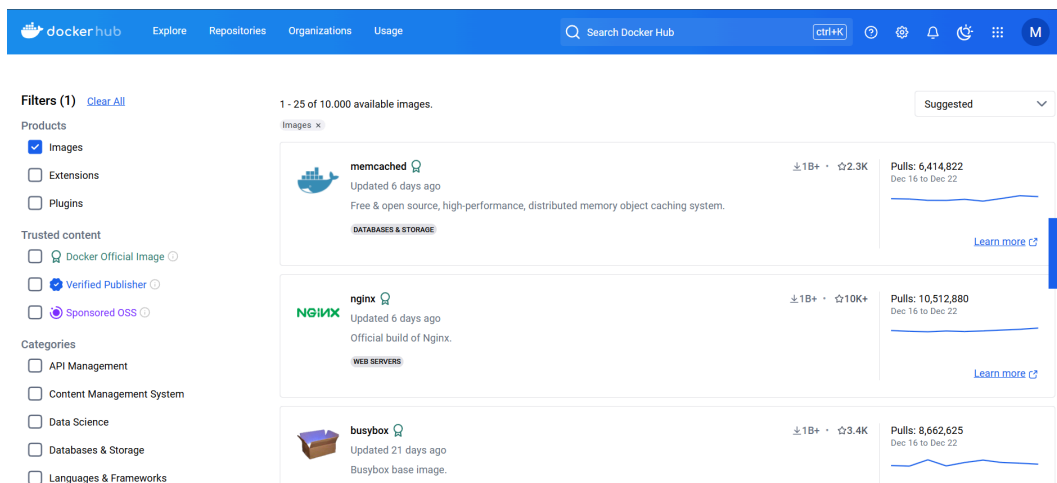
mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$
```

- **Para buscar imágenes desde la terminal:**  
\$ docker search ubuntu  
(siendo ubuntu un ejemplo de imagen) y te aparecen todas las imágenes con ese nombre, su descripción, las estrellas que tiene, si es oficial...
- **Para mostrar un numero limite de opciones:**  
\$ docker search --limit=5  
(aparecen solamente 5 opciones)
- **Para filtrar por número de estrellas:**  
\$ docker search --filter stars=500 ubuntu  
(imagenes con mas de 500 estrellas)
- **Para buscar las que son oficiales:**  
\$ docker search --filter is-official=true ubuntu

Lo más habitual es que estas imágenes se descarguen como hemos comentado antes desde Docker hub, ya que, es el sitio que mayor fiabilidad tiene.

De hecho, al hacer un docker pull como el anterior, directamente desde el bash, estamos descargando la imagen con ese nombre registrada en Docker hub de manera directa, por lo que no es necesario ir hasta la web para descargarnos dicha imagen.

A pesar de todo, esta sería la visión del apartado de la web de Docker hub donde descargar estas imágenes en caso de hacerlo desde la web:



## 2. Asignarle un nuevo nombre (etiqueta) a la imagen

Usa el comando `docker tag` para asignarle un nuevo nombre a la imagen descargada. Por ejemplo:

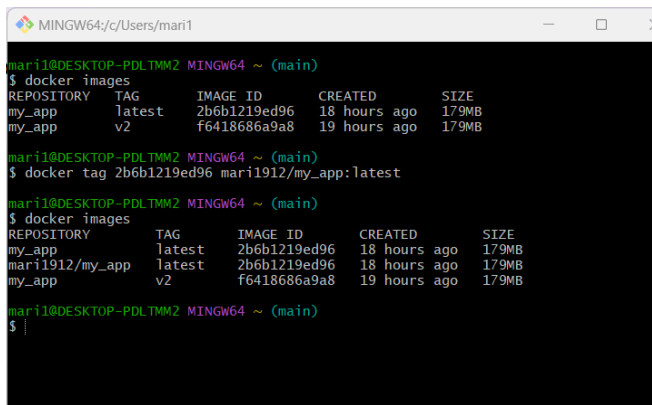
```
$ docker tag ubuntu:latest mi_nombre/mi_imagen:mi_etiqueta
```

- **ubuntu:latest:** Es la imagen descargada.
- **mi\_nombre/mi\_imagen:mi\_etiqueta:** El nuevo nombre personalizado que le quieres asignar:
  - **mi\_nombre:** Puedes usar tu usuario de Docker Hub u otro prefijo.
  - **mi\_imagen:** El nuevo nombre de la imagen.
  - **mi\_etiqueta:** Una etiqueta personalizada, por ejemplo, v1, test, etc.

Ejemplo completo:

```
$ docker pull ubuntu:latest
```

```
$ docker tag ubuntu:latest miusuario/ubuntu_personalizado:v1
```



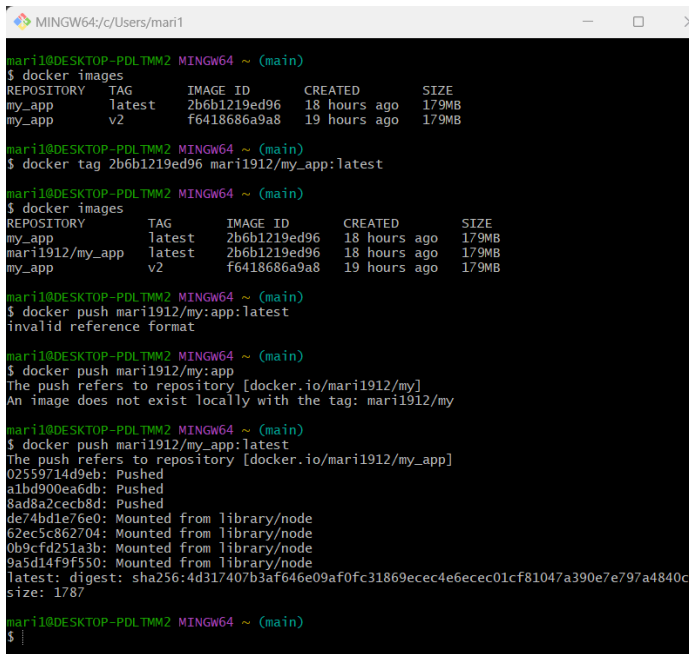
```
MINGW64/c/Users/mari1
mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
my_app        latest    2b6b1219ed96   18 hours ago   179MB
my_app        v2        f6418686a9a8   19 hours ago   179MB

mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker tag 2b6b1219ed96 mari1912/my_app:latest

mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
my_app        latest    2b6b1219ed96   18 hours ago   179MB
mari1912/my_app latest    2b6b1219ed96   18 hours ago   179MB
my_app        v2        f6418686a9a8   19 hours ago   179MB

mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ |
```

Una vez puesto el tag, hacemos un `docker push` para publicarlo en docker hub



```
MINGW64/c/Users/mari1
mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
my_app        latest    2b6b1219ed96   18 hours ago   179MB
my_app        v2        f6418686a9a8   19 hours ago   179MB

mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker tag 2b6b1219ed96 mari1912/my_app:latest

mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
my_app        latest    2b6b1219ed96   18 hours ago   179MB
mari1912/my_app latest    2b6b1219ed96   18 hours ago   179MB
my_app        v2        f6418686a9a8   19 hours ago   179MB

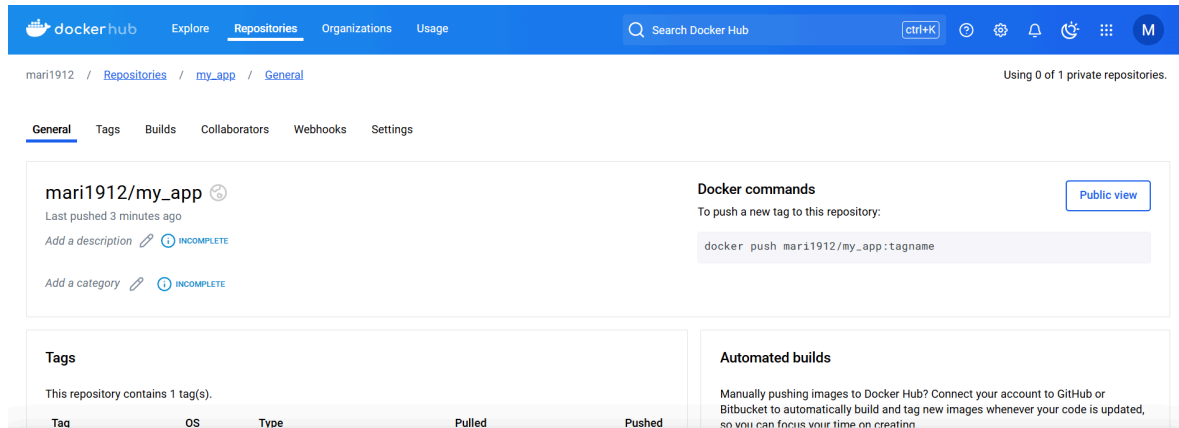
mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker push mari1912/my_app:latest
invalid reference format

mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker push mari1912/my_app
The push refers to repository [docker.io/mari1912/my]
An image does not exist locally with the tag: mari1912/my

mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker push mari1912/my_app:latest
The push refers to repository [docker.io/mari1912/my_app]
02559714d9eb: Pushed
a1bd900ea6db: Pushed
8ad8a2cecb8d: Pushed
de74bd1e76e0: Mounted from library/node
62ec5c862704: Mounted from library/node
0b9cf4251a3b: Mounted from library/node
9a5d14f9f350: Mounted from library/node
latest: digest: sha256:4d317407b3af646e09af0fc31869ecec4e6ecec01cf81047a390e7e797a4840c
size: 1787

mari1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ |
```

Una vez tenemos la imagen subida, nos debe de aparecer en nuestro perfil de Docker Desktop:



## 2.2 Construcción de Imágenes

Las imágenes personalizadas se crean mediante Dockerfiles. Un ejemplo básico de Dockerfile:

```
FROM python:3.8
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

Para construir la imagen:

```
$ docker build -t mi_imagen .
```

Con este comando estamos creando la imagen gracias al Dockerfile, con -t le asignamos el nombre a la imagen, que en este caso sería mi\_imagen, y con el "." al final le indicamos que la tiene que crear en el directorio actual, el cual debería ser el directorio nuevo que hemos creado para el trabajo.

### Detalles sobre el Dockerfile

Un Dockerfile es un archivo de texto que contiene instrucciones paso a paso para construir una imagen de Docker. Cada instrucción crea una nueva capa en la imagen, lo que hace que el proceso sea eficiente gracias al uso de caché. Algunas instrucciones comunes incluyen:

- **FROM:** Define la imagen base.
- **WORKDIR:** Establece el directorio de trabajo dentro del contenedor.
- **COPY:** Copia archivos del sistema host al contenedor.
- **RUN:** Ejecuta comandos en el contenedor durante la construcción de la imagen.
- **CMD:** Define el comando por defecto para ejecutar la aplicación.
- **EXPOSE:** Expone un puerto para permitir conexiones externas.

## Ejemplo Ampliado de Dockerfile

Este Dockerfile configura un servidor web ubuntu con un archivo HTML personalizado:

- Usar una imagen base de ubuntu  
FROM ubuntu:latest
- Copiar el archivo index.html al directorio de ubuntu  
COPY index.html /usr/share/ubuntu/html/
- Exponer el puerto 80  
EXPOSE 80

Para construir y ejecutar esta imagen:

```
$ docker build -t mi_servidor_ubuntu .
$ docker run -d -p 8080:80 mi_servidor_ubuntu
```

Esto hará que el servidor sea accesible en <http://localhost:8080>.

## 2.3 Eliminación de imágenes

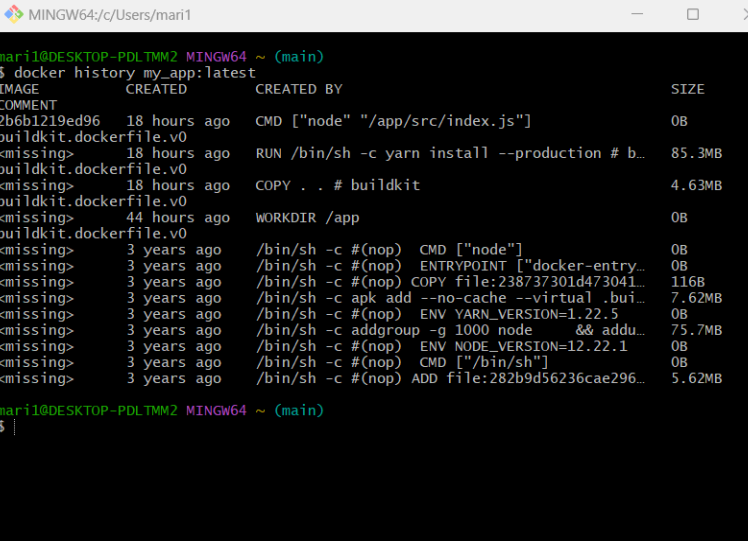
Para eliminar una imagen usamos el comando:

```
$ docker rmi imagen_id o imagen_nombre
```

Sin embargo, debemos tener en cuenta un dato muy importante, y es, que si esta imagen está siendo usada en un contenedor activo, primero tendremos que parar dicho contenedor o eliminarlo para poder eliminar esta imagen.

Las imágenes tienen capas, y cada construcción de Dockerfile da lugar a una nueva capa de imagen, cada imagen puede ejecutar un contenedor, estas capas están ligadas.

Para ver el conjunto de estas capas, se puede ejecutar mediante el comando `$ docker history my_app` (siendo esto el nombre de tu imagen)



```

MINGW64/c/Users/mari1
mar1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker history my_app:latest
IMAGE          CREATED          CREATED BY                                      SIZE
COMMENT
2b6b1219ed96   18 hours ago    CMD ["node" "/app/src/index.js"]              0B
buildkit.dockerfile.v0
<missing>      18 hours ago    RUN /bin/sh -c yarn install --production # b... 85.3MB
buildkit.dockerfile.v0
<missing>      18 hours ago    COPY . . # buildkit                            4.63MB
buildkit.dockerfile.v0
<missing>      44 hours ago    WORKDIR /app                                    0B
buildkit.dockerfile.v0
<missing>      3 years ago     /bin/sh -c #(nop) CMD ["node"]                 0B
<missing>      3 years ago     /bin/sh -c #(nop) ENTRYPOINT ["docker-entry... 0B
<missing>      3 years ago     /bin/sh -c #(nop) COPY file:238737301d473041... 116B
<missing>      3 years ago     /bin/sh -c apk add --no-cache --virtual .bui... 7.62MB
<missing>      3 years ago     /bin/sh -c #(nop) ENV YARN_VERSION=1.22.5       0B
<missing>      3 years ago     /bin/sh -c addgroup -g 1000 node && addu...    75.7MB
<missing>      3 years ago     /bin/sh -c #(nop) ENV NODE_VERSION=12.22.1     0B
<missing>      3 years ago     /bin/sh -c #(nop) CMD ["/bin/sh"]              0B
<missing>      3 years ago     /bin/sh -c #(nop) ADD file:282b9d56236cae296... 5.62MB
mar1@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$

```



Cada capa de imagen Docker es almacenada en la caché.

Cuando creas tu propia imagen, se puede mirar una imagen base, las mejores son las imágenes mínimas.

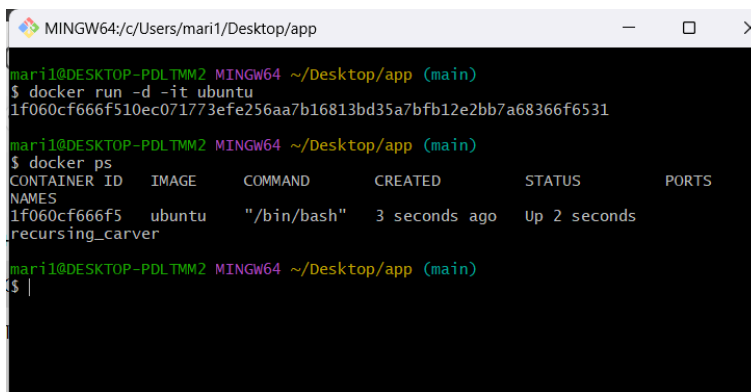
### 3. Gestión de contenedores

#### Ejecución de Contenedores

Para crear y ejecutar un contenedor:

```
$ docker pull ubuntu  
$ docker run -it ubuntu /bin/bash
```

El primer comando descarga la imagen de Ubuntu, mientras que el segundo inicia un contenedor interactivo.



```
MINGW64; c/Users/mari1/Desktop/app  
mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/app (main)  
$ docker run -d -it ubuntu  
1f060cf666f510ec071773efe256aa7b16813bd35a7bfb12e2bb7a68366f6531  
mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/app (main)  
$ docker ps  
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS  
1f060cf666f5   ubuntu   "/bin/bash"             3 seconds ago Up 2 seconds  
recurring_carver  
mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/app (main)  
$ |
```

Si no le indicamos un nombre concreto al contenedor, el sistema de Docker le asigna además de su ID, un nombre de identificación aleatorio, en este caso le ha asignado el nombre de “recurring\_carver”

Algunos comandos esenciales son:

- **Renombrar un contenedor:**  
primero hay que especificar qué container le estamos indicando con el id o el nombre y luego indicar el nuevo nombre que le damos  
\$ docker rename cool\_ride(predeterminado) mi\_redis(nuevo)
- **Listar contenedores activos:**  
\$ docker ps

Para verificar que el contenedor o contenedores que hemos creado estén activos y corriendo en este momento.

```

mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/app (main)
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
1f060cf666f5   ubuntu   "/bin/bash"             56 seconds ago  Up 55 seconds
recurring_carver

```

- **Listar todos los contenedores:**

\$ docker ps -a

con el -a le indicamos que nos muestre todos los contenedores, incluidos los contenedores que se han parado.

```

mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/app (main)
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
1f060cf666f5   ubuntu   "/bin/bash"             58 seconds ago  Up 58 seconds
recurring_carver
9ae159c4254f   ubuntu   "/bin/bash"             4 minutes ago   Exited (0)
mi_contenedor
bceea91c8a6c   adminer   "entrypoint.sh php -..." 20 hours ago   Exited (255)
adminer-container
40dd2b1e7a9f   mysql:8.0 "docker-entrypoint.s..." 20 hours ago   Exited (255)
mysql-container

```

- **Eliminar un contenedor:**

\$ docker rm contenedor\_id o contenedor\_nombre

Si el contenedor está activo en el momento de lanzar este comando para eliminarlo, va a darnos un fallo, primero tenemos que pararlo con:

\$ docker stop contenedor\_id o contenedor\_nombre

o, en su defecto, si queremos eliminarlo directamente sin perder tiempo en pararlo, podemos usar el primer comando añadiendo -f de la siguiente manera:

\$ docker rm -f contenedor\_id o contenedor\_nombre

```

MINGW64:/c:/Users/mari1/Desktop/app
mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/app (main)
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
1f060cf666f5   ubuntu   "/bin/bash"             About a minute ago  Up About a minute
recurring_carver

mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/app (main)
$ docker rm -f recurring_carver
recurring_carver

mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/app (main)
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS          NAMES

```

- **Para reiniciar el contenedor** sin tener que pararlo y luego volver a activarlo se hace un restart

\$ docker restart dfa4d97be605

- **Para eliminar un gran número de contenedores a la vez:**

primero podemos filtrar por los que queramos eliminar, por ejemplo si solamente queremos eliminar los que estén en estatus parado:

```
$ docker ps --filter status=exited
```

Una vez los tenemos controlados, podemos poner el filtro anterior dentro de la condición del comando de eliminación de la siguiente manera:

```
$ docker rm $(docker ps --filter status=exited -q)
```

- **Para paralizar todos los contenedores a la vez:**

```
$ docker stop $(docker ps -q)
```

una vez paralizados todos, se podría usar el comando anterior para eliminar todos los comandos paralizados

- **Podemos ver las estadísticas de los contenedores**

```
$ docker stats
```

para ello tienes que tener contenedores activos

muestra container id, el nombre, el porcentaje de cpu que usa, cuanta memoria ocupa, info de la red etc.

**Para ver toda la información del sistema de Docker:**

```
$ docker info
```

Dentro de este comando, hay dos partes, la parte del cliente y la del server.

*En la parte del server podemos encontrar:*

- Los contenedores que tenemos y cuales tenemos corriendo, cuales pausados y cuales paralizados...
- Las imágenes que tenemos.
- La versión del server.
- El driver de logging que es con ficheros j-son.
- Mucha más información referente a docker, como el sistema operativo, la memoria que utiliza etc.

```
MINGW64/c/Users/mari1
$ docker info
Client:
  Version: 27.3.1
  Context: desktop-linux
  Debug Mode: false
  Plugins:
    ai: Ask Gordon - Docker Agent (Docker Inc.)
       Version: v0.1.0
       Path: C:\Program Files\Docker\cli-plugins\docker-ai.exe
    buildx: Docker Buildx (Docker Inc.)
       Version: v0.18.0-desktop.2
       Path: C:\Program Files\Docker\cli-plugins\docker-buildx.exe
    compose: Docker Compose (Docker Inc.)
       Version: v2.30.3-desktop.1
       Path: C:\Program Files\Docker\cli-plugins\docker-compose.exe
    debug: Get a shell into any image or container (Docker Inc.)
       Version: 0.0.37
       Path: C:\Program Files\Docker\cli-plugins\docker-debug.exe
    desktop: Docker Desktop commands (Alpha) (Docker Inc.)
       Version: v0.0.15
       Path: C:\Program Files\Docker\cli-plugins\docker-desktop.exe
    dev: Docker Dev Environments (Docker Inc.)
       Version: v0.1.2
       Path: C:\Program Files\Docker\cli-plugins\docker-dev.exe
    extension: Manages Docker extensions (Docker Inc.)
       Version: v0.2.27
       Path: C:\Program Files\Docker\cli-plugins\docker-extension.exe
    feedback: Provide feedback, right in your terminal! (Docker Inc.)
       Version: v1.0.5
       Path: C:\Program Files\Docker\cli-plugins\docker-feedback.exe
    init: Creates Docker-related starter files for your project (Docker Inc.)
       Version: v1.4.0
       Path: C:\Program Files\Docker\cli-plugins\docker-init.exe
    sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc.)
       Version: 0.6.0
       Path: C:\Program Files\Docker\cli-plugins\docker-sbom.exe
    scout: Docker Scout (Docker Inc.)
       Version: v1.15.0
       Path: C:\Program Files\Docker\cli-plugins\docker-scout.exe
Server:
  Containers: 2
   Running: 0
   Paused: 0
   Stopped: 2
  Images: 0
  Server Version: 27.3.1
  Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Using metacopy: false
  Native Overlay Diff: true
  userxattr: false
```

### Los Logs en Docker

Los *logs en Docker* son los registros generados por los contenedores y el propio sistema Docker. Estos logs contienen información sobre la actividad y los eventos relacionados con un contenedor en ejecución. Por ejemplo, pueden incluir mensajes del sistema operativo, mensajes de error, eventos de depuración, salidas de aplicaciones en contenedores, entre otros.

En resumen, *los logs son una forma de rastrear lo que ocurre dentro de un contenedor* y son útiles para monitorear, depurar y diagnosticar problemas en aplicaciones y servicios ejecutados en Docker.

### Tipos de logs en Docker

#### 1. Logs de contenedores:

- Incluyen la salida estándar (stdout) y la salida de error (stderr) de las aplicaciones dentro del contenedor.
- Útiles para ver lo que está haciendo la aplicación (mensajes de depuración, errores, eventos).

#### 2. Logs del demonio de Docker:

- Generados por el propio Docker Engine (demonio).
- Contienen información sobre la actividad de Docker, como la creación de contenedores, fallos de imágenes o configuraciones.

## Comandos básicos para trabajar con logs

### 1. Ver los logs de un contenedor

Usa el comando `docker logs` para inspeccionar los logs de un contenedor específico.

```
$ docker logs <container_id_or_name>
```

### 2. Ver logs en tiempo real

Para seguir viendo los logs a medida que se generan, usa la opción `-f` (similar al comando `tail -f`):

```
$ docker logs -f <container_id_or_name>
```

Esto es útil para monitorear el comportamiento en tiempo real de una aplicación.

### 3. Limitar los logs mostrados

Puedes limitar la cantidad de logs que ves con las opciones:

- **--tail <n>**: Muestra solo las últimas <n> líneas de logs.

```
$ docker logs --tail 100 mysql_dev
```

- **--since y --until**: Filtra los logs generados en un rango de tiempo específico.

```
$ docker logs --since "2023-01-01T00:00:00" mysql_dev
```

```
docker logs -f - --details numid (para ver más detalles)
```

```
docker logs -f - --details -t numid (sale el timestamp en la parte izquierda)
```

Para filtrar el número de líneas que quieres que te aparezcan –tail

```
$ docker-compose logs -f
```

## Uso Avanzado de Docker

Docker ofrece funcionalidades avanzadas como redes, volúmenes y orquestación. Estas características permiten implementar arquitecturas complejas de manera eficiente.

### Redes en Docker

Docker soporta diferentes tipos de redes:

- **Bridge:** Red predeterminada para contenedores en un solo host.
- **Overlay:** Permite la comunicación entre contenedores en múltiples hosts.
- **Host:** Usa la pila de red del host directamente.

### Volúmenes en Docker

Los volúmenes son esenciales para persistir datos más allá del ciclo de vida de los contenedores. Permiten compartir datos entre el contenedor y el host, o entre múltiples contenedores.

Tipos de Volúmenes

1. **Volúmenes Anónimos:** Se crean automáticamente y se eliminan con el contenedor.
2. **Volúmenes Nombrados:** Son gestionados por Docker y pueden ser reutilizados entre contenedores.
3. **Bind Mounts:** Montan un directorio específico del host dentro del contenedor.

### *Creación y Uso de Volúmenes*

Ejemplo básico:

- Crear un volumen

```
$ docker volume create mi_volumen
```

- Usar el volumen en un contenedor

```
$ docker run -d -v mi_volumen:/datos ubuntu
```

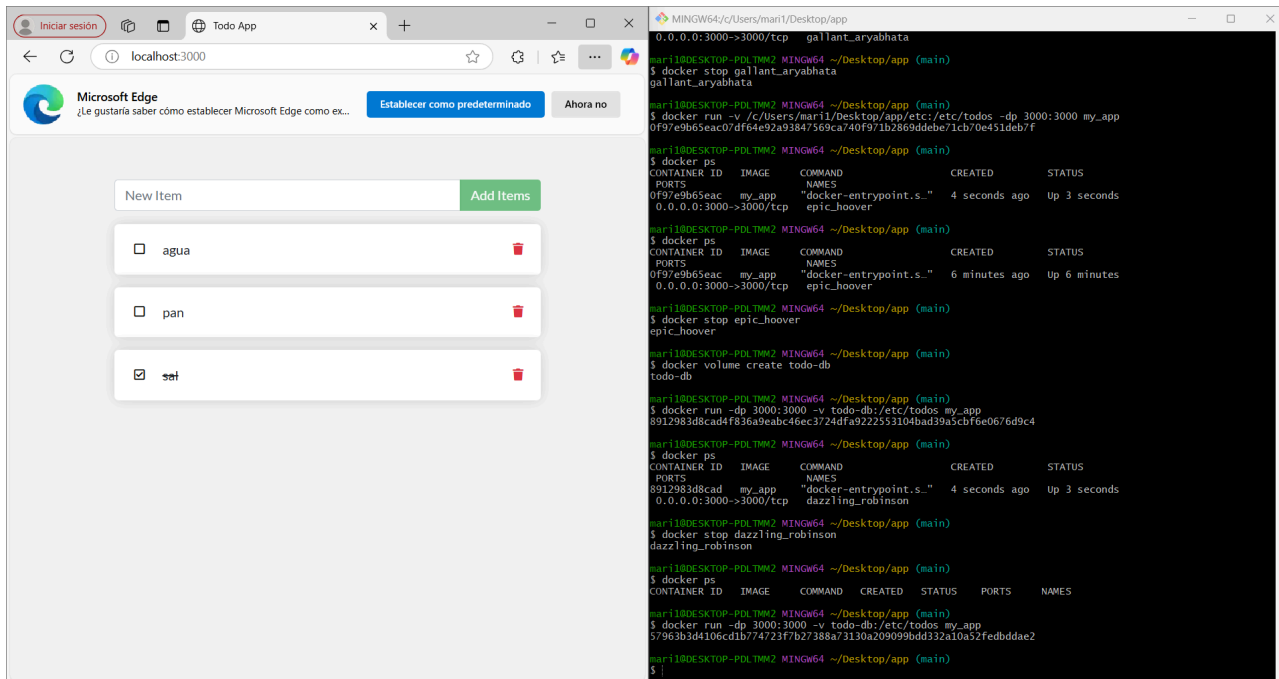
En este ejemplo, los datos almacenados en /datos dentro del contenedor persistirán incluso si el contenedor es eliminado.

Hemos realizado este ejemplo de creación de volumen según el curso que hemos estado siguiendo:

```
$ docker volume create todo-db (primero creamos la base de datos)
```

```
$ docker run -dp 3000:3000 -v todo-db:/etc/todos my_app (después corremos un contenedor indicando que use esa base de datos creada anteriormente)
```

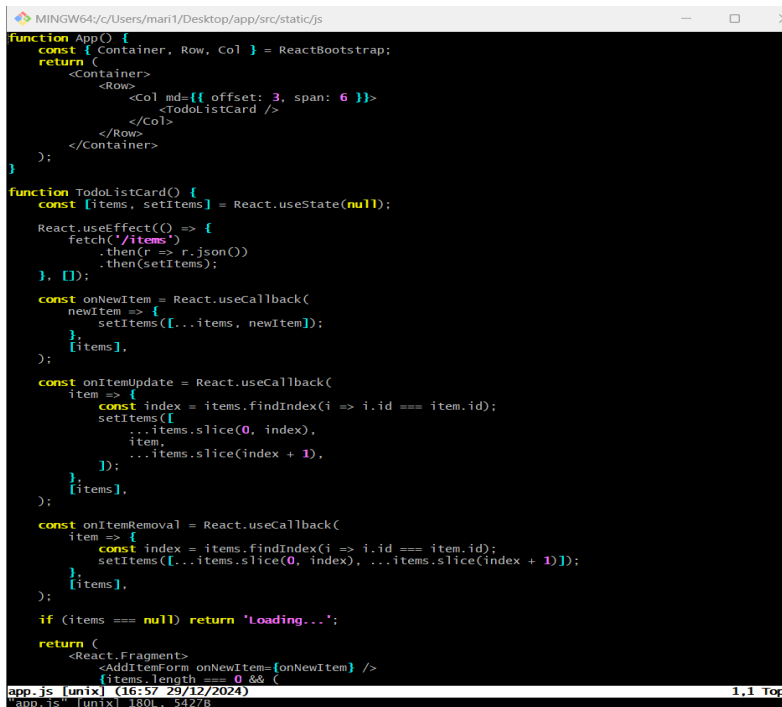
## Entornos de desarrollo - Docker y Docker Compose



\$ docker run -dp 3000:3000 -v todo-db:/etc/todos -v src-db:/src my\_app

\$ vim app.js que está dentro de src

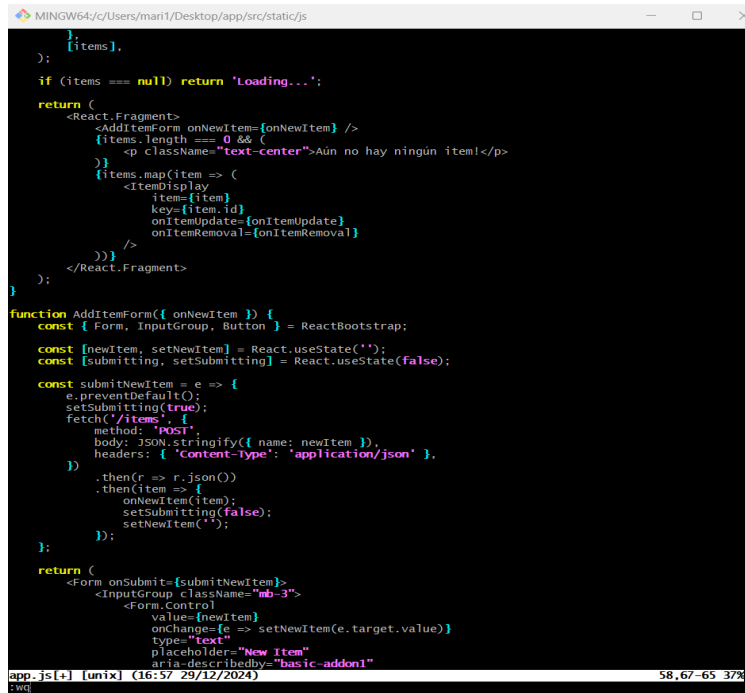
vamos a modificar el código abierto de la app



Entramos para modificar con el carácter “i” y tabulando.



Una vez modificado pulsamos escape :wq para guardar lo modificado



```
};
  items],
);
if (items === null) return 'Loading...';
return (
  <React.Fragment>
    <AddItemForm onNewItem={onNewItem} />
    {items.length === 0 && (
      <p className="text-center">Aún no hay ningún item!</p>
    )}
    {items.map(item => (
      <ItemDisplay
        item={item}
        key={item.id}
        onItemUpdate={onItemUpdate}
        onItemRemoval={onItemRemoval}
      />
    ))}
  </React.Fragment>
);
}

function AddItemForm({ onNewItem }) {
  const { Form, InputGroup, Button } = ReactBootstrap;
  const [newItem, setNewItem] = React.useState('');
  const [submitting, setSubmitting] = React.useState(false);

  const submitNewItem = e => {
    e.preventDefault();
    setSubmitting(true);
    fetch('/items', {
      method: 'POST',
      body: JSON.stringify({ name: newItem }),
      headers: { 'Content-type': 'application/json' },
    })
      .then(r => r.json())
      .then(item => {
        onNewItem(item);
        setSubmitting(false);
        setNewItem('');
      });
  };

  return (
    <Form onSubmit={submitNewItem}>
      <InputGroup className="mb-3">
        <Form.Control
          value={newItem}
          onChange={e => setNewItem(e.target.value)}
          type="text"
          placeholder="New Item"
          aria-describedby="basic-addon1"
        />
      </InputGroup>
    </Form>
  );
}
```

### Compartir Volúmenes entre Contenedores

1. Iniciar un contenedor que comparta un volumen

```
$ docker run -d --name contenedor1 -v mi_volumen:/app/data ubuntu
```

2. Iniciar otro contenedor que use el mismo volumen

```
$ docker run -d --name contenedor2 --volumes-from contenedor1 busybox
```

Este enfoque es útil para compartir datos entre servicios relacionados.

### Inspección de Volúmenes

Para verificar la información de un volumen:

```
$ docker volume inspect mi_volumen
```

Esto proporciona detalles sobre la ubicación y el uso del volumen.

### Conexión de Contenedores con el Mundo mediante Puertos

Una de las características más importantes de Docker es su capacidad para conectar contenedores con el mundo exterior a través del mapeo de puertos. Esto permite que los servicios que se ejecutan dentro de un contenedor sean accesibles desde la red del host o incluso desde Internet.

## ¿Cómo funciona el Mapeo de Puertos?

El mapeo de puertos asocia un puerto interno del contenedor con un puerto externo del sistema host. Esto se configura al iniciar un contenedor con las opciones `-p` o `--publish`.

### Ejemplo Básico de Mapeo de Puertos

```
$ docker run -d -p 8080:80 nginx
```

En este caso:

- El puerto interno 80 del contenedor (donde escucha Nginx por defecto) está vinculado al puerto 8080 del sistema host.
- El servidor Nginx es accesible desde `http://localhost:8080`.

### Publicación Automática de Puertos

Usar `-P` en lugar de `-p` asigna automáticamente los puertos del host disponibles a los puertos del contenedor:

```
$ docker run -d -P nginx
```

Para conocer los puertos asignados, utiliza:

```
$ docker port <contenedor_id>
```

### Configuración Avanzada de Puertos

Docker también soporta especificar protocolos (TCP o UDP):

```
$ docker run -d -p 9090:80/tcp -p 9091:80/udp nginx
```

Esto configura el contenedor para aceptar tráfico TCP en el puerto 9090 y UDP en el puerto 9091.

### Consideraciones de Seguridad

Cuando se mapean puertos para acceso público, es importante considerar:

1. **Firewalls:** Configurar reglas para proteger puertos expuestos.
2. **Certificados SSL:** Garantizar la seguridad de las conexiones con HTTPS.
3. **Restricciones de IP:** Permitir el acceso solo a IPs específicas mediante reglas de red.

### Almacenar imágenes en ZIP

```
$ docker save redis:latest | gzip > myredis.tar.gz
```

para desplegar imágenes de docker a la nube es muy útil para poder enviarlas como un archivo comprimido.

1. Primero, podemos crear la carpeta manualmente, o escribir en la máquina de comandos:

```
$ mkdir -p /c/Users/mari1/Desktop/my_image/
```

Para verificar que lo hemos hecho bien usamos:

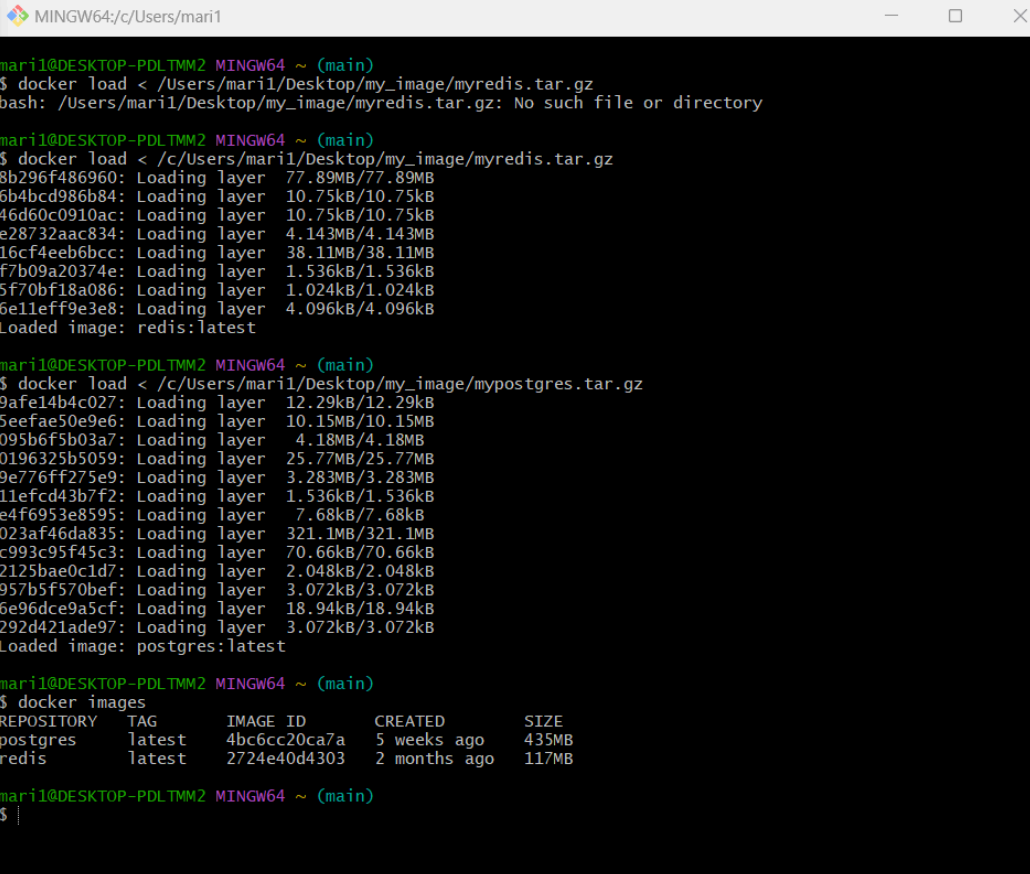
```
$ ls /c/Users/mari1/Desktop/my_image/
```

2. Una vez vemos que la carpeta está correctamente creada, para guardar ahí el archivo zip de la imagen que queramos dentro de esa carpeta, usamos la siguiente:

```
$ docker save redis:latest | gzip > /c/Users/mari1/Desktop/my_image/myredis.tar.gz
```

De la misma manera que se pueden comprimir imágenes, también las imágenes se pueden importar desde archivos comprimidos a docker.

`$ docker load < /c/Users/mari1/Desktop/my_image/myredis.tar.gz` (ruta donde tienes el zip de la imagen que quieres importar)



```
MINGW64:/c/Users/mari1
maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker load < /c/Users/mari1/Desktop/my_image/myredis.tar.gz
bash: /c/Users/mari1/Desktop/my_image/myredis.tar.gz: No such file or directory

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker load < /c/Users/mari1/Desktop/my_image/myredis.tar.gz
8b296f486960: Loading layer 77.89MB/77.89MB
6b4bcd986b84: Loading layer 10.75kB/10.75kB
46d60c0910ac: Loading layer 10.75kB/10.75kB
e28732aac834: Loading layer 4.143MB/4.143MB
16cf4eeb6bcc: Loading layer 38.11MB/38.11MB
f7b09a20374e: Loading layer 1.536kB/1.536kB
5f70bf18a086: Loading layer 1.024kB/1.024kB
6e11eff9e3e8: Loading layer 4.096kB/4.096kB
Loaded image: redis:latest

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker load < /c/Users/mari1/Desktop/my_image/mypostgres.tar.gz
9afe14b4c027: Loading layer 12.29kB/12.29kB
5eefae50e9e6: Loading layer 10.15MB/10.15MB
095b6f5b03a7: Loading layer 4.18MB/4.18MB
0196325b5059: Loading layer 25.77MB/25.77MB
9e776ff275e9: Loading layer 3.283MB/3.283MB
11efcd43b7f2: Loading layer 1.536kB/1.536kB
e4f6953e8595: Loading layer 7.68kB/7.68kB
023af46da835: Loading layer 321.1MB/321.1MB
c993c95f45c3: Loading layer 70.66kB/70.66kB
2125bae0c1d7: Loading layer 2.048kB/2.048kB
957b5f570bef: Loading layer 3.072kB/3.072kB
6e96dce9a5cf: Loading layer 18.94kB/18.94kB
292d421ade97: Loading layer 3.072kB/3.072kB
Loaded image: postgres:latest

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
postgres      latest    4bc6cc20ca7a   5 weeks ago    435MB
redis         latest    2724e40d4303   2 months ago   117MB

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ |
```

## Tecnologías de Apoyo en el Entorno Docker

Docker no trabaja de manera aislada, sino que se complementa con tecnologías que amplían sus capacidades y mejoran su funcionalidad en entornos complejos. Algunas de estas tecnologías son:

### Docker Compose

Docker Compose permite gestionar aplicaciones multicontenedor mediante un archivo YAML. Es ideal para configurar y probar entornos en desarrollo.

### Docker Swarm

Docker Swarm proporciona orquestación nativa para gestionar clústeres de contenedores. Permite desplegar y escalar aplicaciones distribuidas.

Ventajas:

1. **Escalabilidad automática:** Añade o elimina réplicas de servicios.
2. **Alta disponibilidad:** Redistribuye servicios automáticamente en caso de fallo de un nodo.

### Kubernetes

Kubernetes es una herramienta avanzada para la orquestación de contenedores. Ofrece escalabilidad y administración automatizada en entornos de producción.

### Docker Machine

Docker Machine automatiza la creación de hosts Docker en diversas plataformas (locales, nube, o virtuales). Es ideal para gestionar múltiples entornos desde un único cliente.

### Docker Trusted Registry (DTR)

DTR es una solución empresarial para almacenar imágenes de Docker en un entorno seguro y privado. Proporciona escaneo de vulnerabilidades y control de acceso basado en roles.

## Introducción a Docker Compose

Docker Compose es una herramienta para definir y ejecutar aplicaciones multicontenedor mediante un archivo YAML (docker-compose.yml). En lugar de iniciar manualmente contenedores individuales y configurarlos, Compose automatiza el proceso.

## Ventajas de Docker Compose

1. **Simplicidad:** Permite definir múltiples servicios en un solo archivo, lo que simplifica la administración de aplicaciones complejas.
2. **Orquestación Local:** Ideal para entornos de desarrollo y pruebas.
3. **Escalabilidad:** Facilita la escalabilidad horizontal mediante el comando `docker-compose up --scale`.
4. **Compatibilidad con Volúmenes y Redes:** Gestiona automáticamente el almacenamiento y la comunicación entre servicios.

## Estructura de un Archivo Docker-Compose

Un archivo típico `docker-compose.yml` incluye:

1. **Version:** Define la versión de Docker Compose.
2. **Services:** Cada servicio corresponde a un contenedor.
3. **Networks:** Configuración de redes personalizadas para la comunicación entre servicios.
4. **Volumes:** Define cómo y dónde se almacenarán los datos persistentes.

Ejemplo Básico:

Archivo `docker-compose.yml`:

```
version: '3.8'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
  app:
    image: python:3.8
    volumes:
      - ./app:/usr/src/app
    command: python app.py
```

Este archivo define dos servicios (web y app) y configura sus redes, volúmenes y puertos.

## Casos de Uso de Docker y Docker Compose

1. **Desarrollo Local:**
  - Docker Compose facilita la configuración de entornos que replican la producción.
  - Ejemplo: Una aplicación web que utiliza Nginx, Node.js y PostgreSQL.
2. **Microservicios:**
  - Docker permite empaquetar cada microservicio en un contenedor independiente, mientras que Compose simplifica la orquestación.

### 3. Integración Continua:

- Con herramientas de CI/CD, Docker permite construir y probar aplicaciones en contenedores aislados.

### 4. Testing:

- Es posible crear entornos temporales para ejecutar pruebas automatizadas.

Ejemplo de uso de Docker Compose utilizando nicolaka/netshoot y mysql

```
$ docker run -d \
--network todo-app --network-alias mysql \
-v /c/Users/mari1/Desktop/multi-container/todo-mysql-data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=secret \
-e MYSQL_DATABASE=todos \
mysql:5.7
```

```
mingw64/c/Users/mari1
maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker run -d \
> --network todo-app --network-alias mysql \
> -v /c/Users/mari1/Desktop/multi-container/todo-mysql-data:/var/lib/mysql \
> -e MYSQL_ROOT_PASSWORD=secret \
> -e MYSQL_DATABASE=todos \
> mysql:5.7^C

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker network create todo-app
c1f05dda53af053fafc5f7720764d885f4a6070cd3588c9be5c0638830a45d05
```

```
op/multi-container/todo-mysql-data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=secret
-e MYSQL_DATABASE=todos mysql:5.7
Unable to find image 'mysql:5.7' locally
5.7: Pulling from library/mysql
20e4dcae4c69: Pull complete
1c56c3d4ce74: Pull complete
e9f03a1c24ce: Pull complete
68c3898c2015: Pull complete
6b95a940e7b6: Pull complete
90986bb8de6e: Pull complete
ae71319cb779: Pull complete
ffc89e9dfd88: Pull complete
43d05e938198: Pull complete
064b2d298fba: Pull complete
df9a4d85569b: Pull complete
Digest: sha256:4bc6bc963e6d8443453676cae56536f4b8156d78bae03c0145cbe47c2aad73bb
Status: Downloaded newer image for mysql:5.7
1c353a1856dc54bd448c17f4f3dc8f81d13936d1b15cc28637fb3c89fd63c972

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
my_app              latest      2b6b1219ed96  2 days ago    179MB
maril1912/my_app    latest      2b6b1219ed96  2 days ago    179MB
my_app              v2         f6418686a9a8  2 days ago    179MB
ubuntu              latest      b1d9df8ab815  6 weeks ago   78.1MB
mysql               5.7        5107333e08a8  12 months ago 501MB

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS
PORTS
1c353a1856dc   mysql:5.7  "docker-entrypoint.s..." 28 seconds ago Up 26 secon
ds          3306/tcp, 33060/tcp  friendly_matsumoto
```

```

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker exec -it 1c353a1856dc mysql -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.44 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| todos |
+-----+
5 rows in set (0.00 sec)

mysql>

```

```

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker run -it --network todo-app nicolaka/netshoot
Unable to find image 'nicolaka/netshoot:latest' locally
latest: Pulling from nicolaka/netshoot
4abcf2066143: Pull complete
f72249ed6705: Pull complete
d21093198226: Pull complete
ff793c57efef: Pull complete
b8cdfec6d24e: Pull complete
b6621d484422: Pull complete
452eb7889eb5: Pull complete
4f4fb700ef54: Pull complete
89065cf5c037: Pull complete
a4b421d4901a: Pull complete
d5c3ad7ea15a: Pull complete
ab073295bbd0: Pull complete
737c1bf9f2ef: Pull complete
097ac21093f8: Pull complete
59e353e0ee74: Pull complete
Digest: sha256:a20c2531bf35436ed3766cd6cfe89d352b050ccc4d7005ce6400adf97503da1b
Status: Downloaded newer image for nicolaka/netshoot:latest
      dP      dP      dP
      88      88      88
88d888b. .d8888b. d8888P .d8888b. 88d888b. .d8888b. .d8888b. d8888P
88' `88 88oooo8 88 Y8oooo. 88' `88 88' `88 88' `88 88
88 88 88. .88 88 88 88 88 88 88 88 88 88
dP dP `88888P' dP `88888P' dP dP `88888P' `88888P' dP

Welcome to Netshoot! (github.com/nicolaka/netshoot)
Version: 0.13

2f764b709228

```

Mediante el alias que le asignamos a la base de datos anteriormente, podemos comprobar que se haya asignado correctamente a la ip.

```
2F764b709228 ➡ dig mysql
; <<> DiG 9.18.25 <<> mysql
; global options: +cmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24081
; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

; QUESTION SECTION:
;mysql.                                IN      A

; ANSWER SECTION:
mysql.                600      IN      A      172.18.0.2

; Query time: 0 msec
; SERVER: 127.0.0.11#53(127.0.0.11) (UDP)
; WHEN: Thu Jan 02 13:10:47 UTC 2025
; MSG SIZE rcvd: 44

2F764b709228 ➡
```

Enlazar dos contenedores entre sí mediante el uso de una red

```
$ docker run -dp 3000:3000 --network todo-app -e MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=secret -e MYSQL_DB=todos my_app
```

```
MINGW64:/c/Users/mari1
maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker run -dp 3000:3000 --network todo-app -e MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=secret -e MYSQL_DB=todos my_app
0d6ef33a1c69b275fef7529cec49b0998e794894d76aef9cb9f7d7e7fb340f37

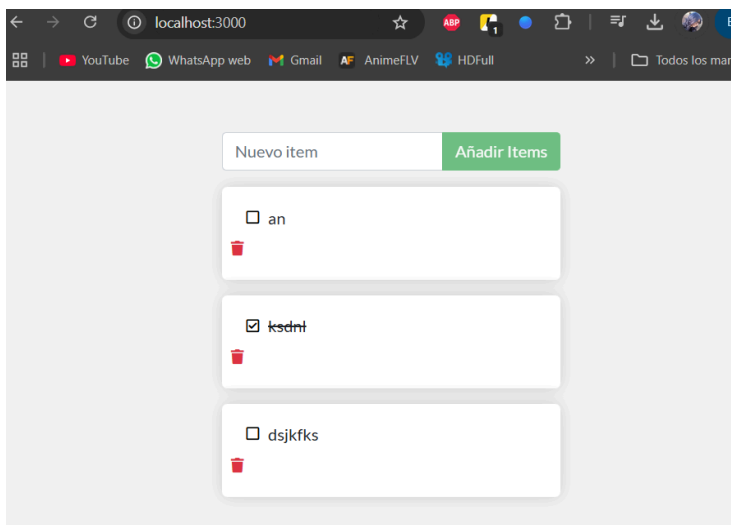
maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
0d6ef33a1c69   my_app        "docker-entrypoint.s..." 4 seconds ago  Up 3 seconds  0.0.0.0:3000->3000/tcp
c353a1856dc    mysql:5.7      "docker-entrypoint.s..." 9 minutes ago  Up 9 minutes  3306/tcp
b, 33060/tcp    friendly_matsumoto

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker stop funny_hermann
funny_hermann

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$ docker start funny_hermann
funny_hermann

maril@DESKTOP-PDLTMM2 MINGW64 ~ (main)
$
```

Al estar asociados a la base de datos creada, los datos de la app de ejemplo se mantienen:





Ejecutar en visual studio los mismos comandos que hicimos en docker

```
version: "3.7"

services:

#docker run -dp 3000:3000 \
#--network todo-app \
#-e MYSQL_HOST=mysql \
#-e MYSQL_USER=root \
#-e MYSQL_PASSWORD=secret \
#-e MYSQL_DB=todos \
#my_app

  app:
    image: my_app
    ports:
      - 3000:3000
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

#docker run -d \
#--network todo-app --network-alias mysql \
#-v /c/Users/mari1/Desktop/multi-container/todo-mysql-data:/var/lib/mysql \
#-e MYSQL_ROOT_PASSWORD=secret \
#-e MYSQL_DATABASE=todos \
#mysql:5.7
  mysql:
    image: mysql:5.7
    volumes:
      - /c/Users/mari1/Desktop/multi-container/todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos
```

Levantamos con docker compose el contenedor para que la aplicación se vuelva a levantar en el host:

```
MINGW64/c:/Users/mari1/Desktop/multi-container

mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/multi-container (main)
$ docker-compose up -d
time="2025-01-02T18:19:58+01:00" level=warning msg="C:\\Users\\mari1\\Desktop\\multi-container\\do
cker-compose.yaml: the attribute 'version' is obsolete, it will be ignored, please remove it to av
oid potential confusion"
[+] Running 2/2
 ✓ Container multi-container-app-1   Started      0.6s
 ✓ Container multi-container-mysql-1 Started      0.6s

mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/multi-container (main)
$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS        PORTS
NAMES
551b4f9385ec   my_app              "docker-entrypoint.s..." 4 minutes ago  Up 3 seconds  0.0.0.0:3000->3
000/tcp      multi-container-app-1
cc277f6ff47a   mysql:5.7           "docker-entrypoint.s..." 4 minutes ago  Up 3 seconds  3306/tcp, 33060
/tcp        multi-container-mysql-1

mari1@DESKTOP-PDLTMM2 MINGW64 ~/Desktop/multi-container (main)
$ |
```

### Limitaciones de Docker

1. **Curva de Aprendizaje:** Configurar Docker y Compose puede ser complejo para principiantes.
2. **Persistencia de Datos:** Aunque los volúmenes resuelven este problema, el manejo de datos sigue siendo un desafío en entornos de producción.
3. **Sobrecarga en Windows/Mac:** Docker utiliza una máquina virtual en estos sistemas, lo que puede aumentar el consumo de recursos.
4. **Escalabilidad Limitada:** Para aplicaciones masivamente escalables, es mejor usar herramientas como Kubernetes.

### Buenas Prácticas

1. **Imágenes Ligeras:** Usar imágenes base minimalistas (por ejemplo, Alpine).
2. **Separación de Preocupaciones:** Dividir aplicaciones en servicios individuales.
3. **Versiónado:** Especificar versiones específicas para imágenes y dependencias.
4. **Seguridad:**
  - Escanear imágenes en busca de vulnerabilidades.
  - Minimizar los permisos de usuario dentro de los contenedores.
5. **Optimización de Capas:** Ordenar instrucciones en Dockerfile para aprovechar la caché.

## Entorno de desarrollo con Eclipse y MySQL utilizando Docker Compose

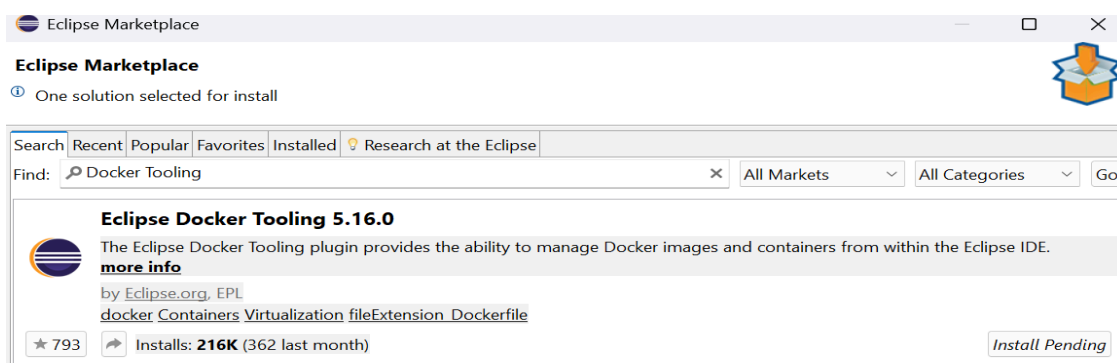
Para empezar a construir con Docker Compose crearemos el archivo `.yaml` en nuestra carpeta `proyecto-eclipse`. Este archivo es muy importante ya que está toda la información con la que se podrá conectar los entornos de eclipse y MySQL entre sí.

```
version: '3.9'

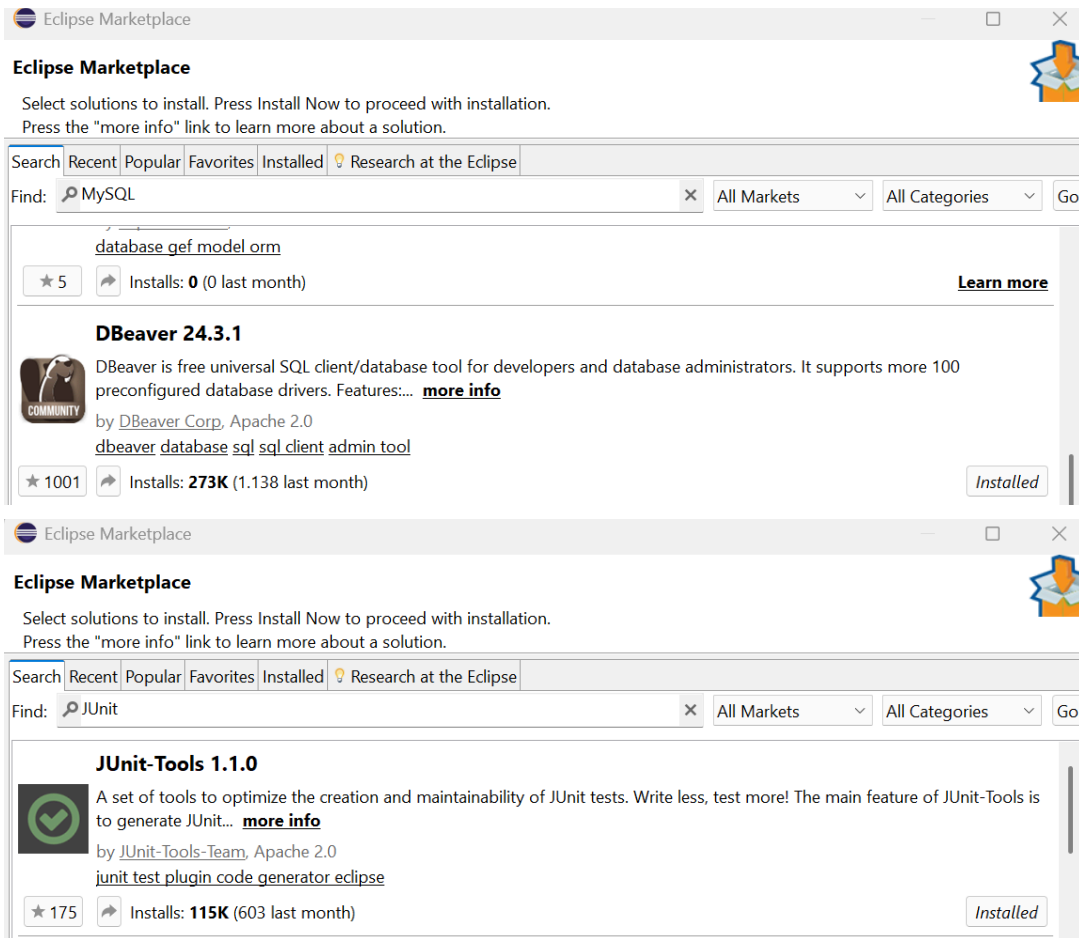
services:
  mysql:
    image: mysql:8.0
    container_name: mysql_dev
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: devdb
      MYSQL_USER: devuser
      MYSQL_PASSWORD: devpassword
    ports:
      - "3306:3306"
    volumes:
      - db_data:/var/lib/mysql
    restart: always

volumes:
  db_data:
```

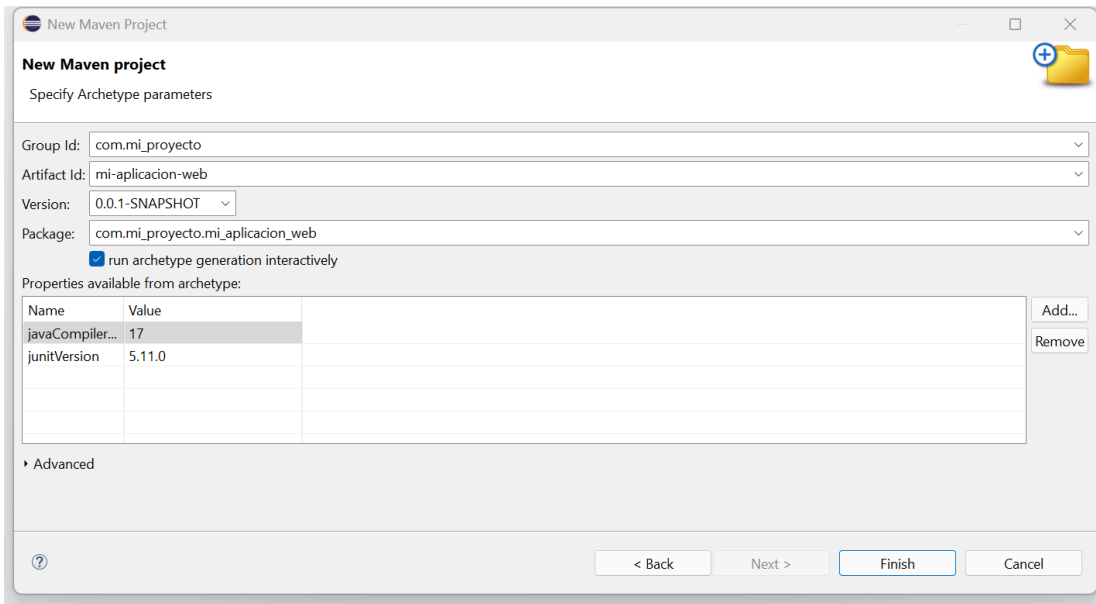
Una vez creado en nuestra carpeta tendremos que abrir eclipse y proceder a la instalación de los plugins de MySQL, Docker y JUnit.



## Entornos de desarrollo - Docker y Docker Compose



Ahora con los plugins instalados correctamente el siguiente paso es crear nuestro proyecto maven y su configuración.



```

C:\Users\Vidal_P\2\proyectos\ejemplo\src\main\java\com\mi_proyecto\mi_aplicacion_web
[INFO] Parameter: artifactId, Value: mi-aplicacion-web
[INFO] Parameter: version, Value: 0.0.1-SNAPSHOT
[INFO] Parameter: package, Value: com.mi_proyecto.mi_aplicacion_web
[INFO] Parameter: packagingPathFormat, Value: com/mi_proyecto/mi_aplicacion_web
[INFO] Parameter: junitVersion, Value: 5.11.0
[INFO] Parameter: package, Value: com.mi_proyecto.mi_aplicacion_web
[INFO] Parameter: groupId, Value: com.mi_proyecto
[INFO] Parameter: artifactId, Value: mi-aplicacion-web
[INFO] Parameter: javaCompilerVersion, Value: 17
[INFO] Parameter: version, Value: 0.0.1-SNAPSHOT
[WARNING] Don't override file C:\Users\Vidal_P\2\proyectos\ejemplo\src\main\java\com\mi_proyecto\mi_aplicacion_web
[WARNING] Don't override file C:\Users\Vidal_P\2\proyectos\ejemplo\src\test\java\com\mi_proyecto\mi_aplicacion_web
[WARNING] CP Don't override file C:\Users\Vidal_P\2\proyectos\ejemplo\src\main\java\com\mi_proyecto\mi_aplicacion_web
[INFO] Project created from Archetype in dir: C:\Users\Vidal_P\2\proyectos\ejemplo
[INFO] BUILD SUCCESS
[INFO] Total time: 13.817 s
[INFO] Finished at: 2025-01-04T18:37:07+01:00
[INFO]

```

Con el maven creado tendremos que añadir las dependencias en el archivo pom de JUnit y MySQL para que nuestro archivo.java pueda importar las librerías correctamente. Una vez añadidas actualizaremos el pom para que el cambio se efectúe.

```

<dependencies>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.0.33</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>

```

Seguidamente levantaremos un contenedor docker MySQL y crearemos la tabla con el nombre ejemplo para poder guardar la información desde eclipse.

```

Vidal@Vidal_PC MINGW64 ~/OneDrive/Desktop/proyecto-eclipse
$ docker-compose up -d
time="2025-01-04T18:49:24+01:00" level=warning msg="C:\\Users\\Vidal\\OneDrive\\Desktop\\proyecto-eclipse\\docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
  ✓ Network proyecto-eclipse_default    Created      0.0s
  ✓ Volume "proyecto-eclipse_db_data"   Created      0.0s
  ✓ Container mysql_dev                 Started      0.5s

Vidal@Vidal_PC MINGW64 ~/OneDrive/Desktop/proyecto-eclipse
$ docker exec -it mysql_dev mysql -u devuser -pdevpassword
mysql: [warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE devdb;
Database changed
mysql> SHOW TABLES;
Empty set (0.00 sec)

mysql> CREATE TABLE ejemplo (
->   id INT AUTO_INCREMENT PRIMARY KEY,
->   nombre VARCHAR(255) NOT NULL
-> );
Query OK, 0 rows affected (0.04 sec)

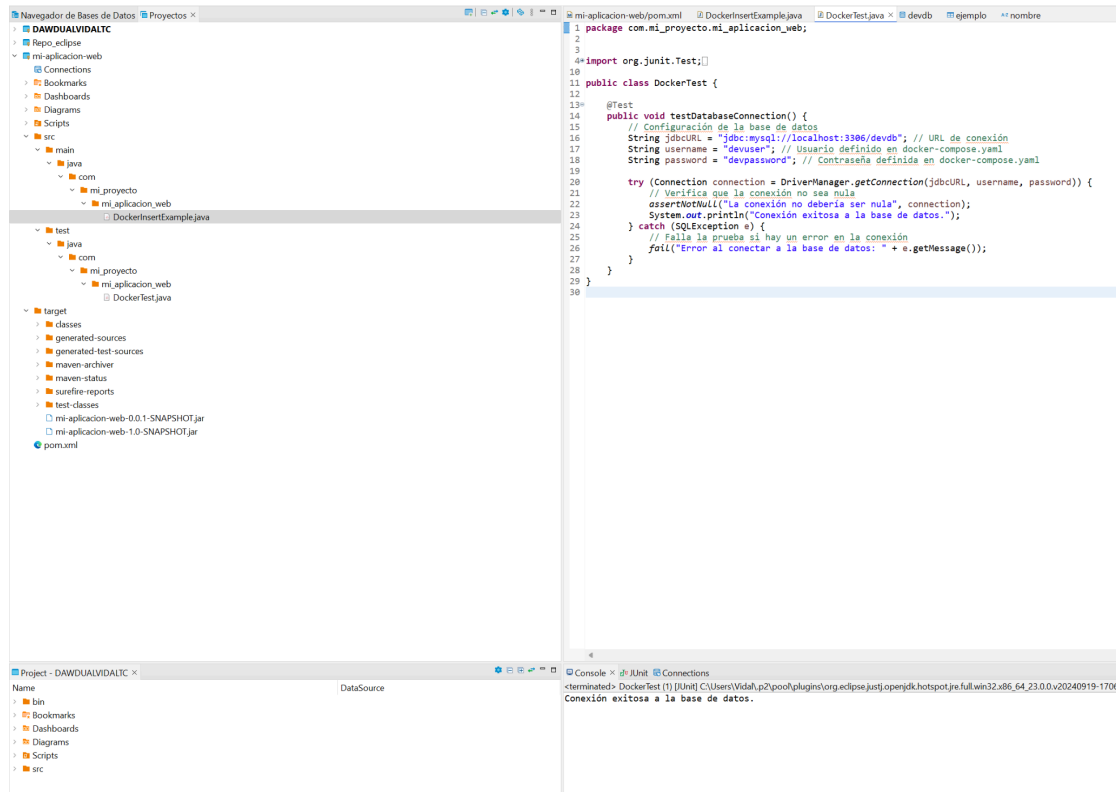
mysql> docker ps
-> AC
mysql> EXIT
Bye

Vidal@Vidal_PC MINGW64 ~/OneDrive/Desktop/proyecto-eclipse
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS
5d7e1e25a6be   mysql:8.0  "docker-entrypoint.s..." About a minute ago Up About a minute
0.0.0.0:3306->3306/tcp, 33060/tcp   mysql_dev

```

## Entornos de desarrollo - Docker y Docker Compose

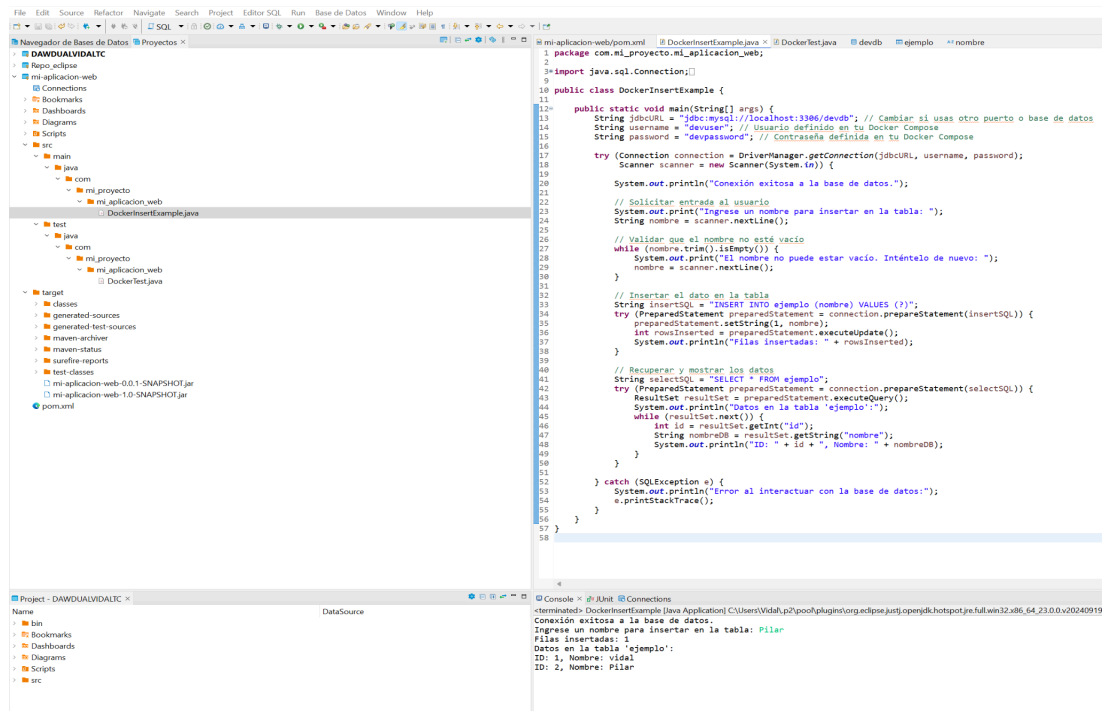
Con todo montado crearemos la clase `DockerTest.java` en la carpeta `Test` para hacer la verificación de conexión.



The screenshot shows an IDE with a project named 'mi-aplicacion-web'. The file explorer on the left shows the project structure, including a 'test' directory containing 'DockerTest.java'. The main editor displays the code for 'DockerTest.java', which is a JUnit test class. The code imports 'org.junit.Test' and defines a test method 'testDatabaseConnection()' that uses 'DriverManager.getConnection()' to connect to a database. The console output at the bottom shows the test passing, with the message 'Conexión exitosa a la base de datos.'

```
1 package com.mi_proyecto.mi_aplicacion_web;
2
3
4 import org.junit.Test;
5
6 public class DockerTest {
7
8     @Test
9     public void testDatabaseConnection() {
10         // Configuración de la base de datos
11         String jdbcURL = "jdbc:mysql://localhost:3306/devdb"; // URL de conexión
12         String username = "devuser"; // Usuario definido en docker-compose.yaml
13         String password = "devpassword"; // Contraseña definida en docker-compose.yaml
14
15         try (Connection connection = DriverManager.getConnection(jdbcURL, username, password)) {
16             // Verifica que la conexión no sea nula
17             assertNotNull("La conexión no debería ser nula", connection);
18             System.out.println("Conexión exitosa a la base de datos.");
19         } catch (SQLException e) {
20             // Falla la prueba si hay un error en la conexión
21             fail("Error al conectar a la base de datos: " + e.getMessage());
22         }
23     }
24 }
```

El siguiente paso es crear en el main nuestro programa `DockerInsertExample.java` para guardar la información en la base de datos.



The screenshot shows the same IDE with the 'DockerInsertExample.java' file. The code imports 'java.sql.Connection' and defines a 'main' method that prompts the user for a name to insert into the database. It uses 'DriverManager.getConnection()' to connect to the database and 'PreparedStatement' to execute the insert query. The console output shows the successful insertion of two records: 'Pilar' and 'Pilar'.

```
1 package com.mi_proyecto.mi_aplicacion_web;
2
3 import java.sql.Connection;
4
5 public class DockerInsertExample {
6
7     public static void main(String[] args) {
8         String jdbcURL = "jdbc:mysql://localhost:3306/devdb"; // Cambiar si usas otro puerto o base de datos
9         String username = "devuser"; // Usuario definido en tu Docker Compose
10        String password = "devpassword"; // Contraseña definida en tu Docker Compose
11
12        try (Connection connection = DriverManager.getConnection(jdbcURL, username, password);
13             Scanner scanner = new Scanner(System.in)) {
14            System.out.println("Conexión exitosa a la base de datos.");
15
16            // Solicitar entrada al usuario
17            System.out.print("Ingresa un nombre para insertar en la tabla: ");
18            String nombre = scanner.nextLine();
19
20            // Validar que el nombre no esté vacío
21            while (nombre.trim().isEmpty()) {
22                System.out.print("El nombre no puede estar vacío. Inténtelo de nuevo: ");
23                nombre = scanner.nextLine();
24            }
25
26            // Insertar el dato en la tabla
27            String insertSQL = "INSERT INTO ejemplo (nombre) VALUES (?)";
28            try (PreparedStatement preparedStatement = connection.prepareStatement(insertSQL)) {
29                preparedStatement.setString(1, nombre);
30                int rowsInserted = preparedStatement.executeUpdate();
31                System.out.println("Filas insertadas: " + rowsInserted);
32            }
33
34            // Recuperar y mostrar los datos
35            String selectSQL = "SELECT * FROM ejemplo";
36            try (PreparedStatement preparedStatement = connection.prepareStatement(selectSQL)) {
37                ResultSet resultSet = preparedStatement.executeQuery();
38                System.out.println("Datos en la tabla 'ejemplo':");
39                while (resultSet.next()) {
40                    int id = resultSet.getInt("id");
41                    String nombreDB = resultSet.getString("nombre");
42                    System.out.println("ID: " + id + ", Nombre: " + nombreDB);
43                }
44            }
45        } catch (SQLException e) {
46            System.out.println("Error al interactuar con la base de datos:");
47            e.printStackTrace();
48        }
49    }
50 }
```

## Entornos de desarrollo - Docker y Docker Compose

Ahora nos iremos a nuestra base de datos para verificar que los datos se han guardado en la tabla.

The screenshot shows a database management interface. On the left, a tree view displays the database structure, including a table named 'ejemplo' with columns 'id' (int) and 'nombre' (varchar(255)). The main area shows the data in the 'ejemplo' table:

id	nombre
1	vidal
2	Pilar

At the bottom, the console output shows the following messages:

```
<terminated> DockerInsertExample [Java Application] C:\Users\Vidal\p2\pool\plugins\org.eclipse.justj.openjdk.hot
Conexión exitosa a la base de datos.
Ingreso un nombre para insertar en la tabla: Pilar
Filas insertadas: 1
Datos en la tabla 'ejemplo':
ID: 1, Nombre: vidal
ID: 2, Nombre: Pilar
```