

Bloque 10. Docker

¿Qué es Docker?	2
Instalación de Docker	3
Docker desktop	3
Descarga de Docker Desktop:	3
Instalación de W2L (Windows Subsystem for Linux)	5
Docker Hub	14
Descargas	17
Etiquetas	17
Descarga	17
Documentación	18
Comandos de Docker	20
Funcionamiento de Docker	22
Trabajando con Docker	24
¿Qué es una imagen de Docker?	24
Ejemplo con imágenes	25
Visualizar Imágenes	26
Buscar una imagen	26
Eliminar Imágenes	27
Creación de nuestras propias imágenes en Docker	28
Creando el Dockerfile	29
Compilación del proyecto	31
Construcción de nuestra imagen	33
Trabajando con contenedores	35
Crear un contenedor	35
Variables de entorno	36
Probando el contenedor	38
Parar y arrancar Contenedores	41
Información del contenedor	42
Eliminando contenedores	43
¿Matar un contenedor?	43
AutoEliminación	44
Redes con Docker	45
Creando Redes	45
Visualizar Redes	45
Eliminar Redes	46
Inspeccionar Redes	46
Docker compose	47
Paso a paso	48
Levantar Docker compose	49
Eliminar el Docker compose	51



Troubleshooting	55
¿Cómo podemos saber qué JRE tenemos en el equipo?	55
Resumen comandos Docker	56



¿Qué es Docker?

Docker es una plataforma de contenedorización de código abierto que revoluciona la forma en que se desarrollan, empaquetan y ejecutan aplicaciones. Permite a los desarrolladores empaquetar aplicaciones en contenedores, que son componentes ejecutables estandarizados que combinan el código fuente de la aplicación con las bibliotecas del sistema operativo y las dependencias necesarias para ejecutar ese código en cualquier entorno.

Funcionamiento de Docker:

Docker funciona como un sistema operativo para contenedores. Similar a cómo una máquina virtual virtualiza el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker está instalado en cada servidor y proporciona comandos simples que se pueden utilizar para crear, iniciar o detener contenedores.

Qué se puede hacer con Docker:

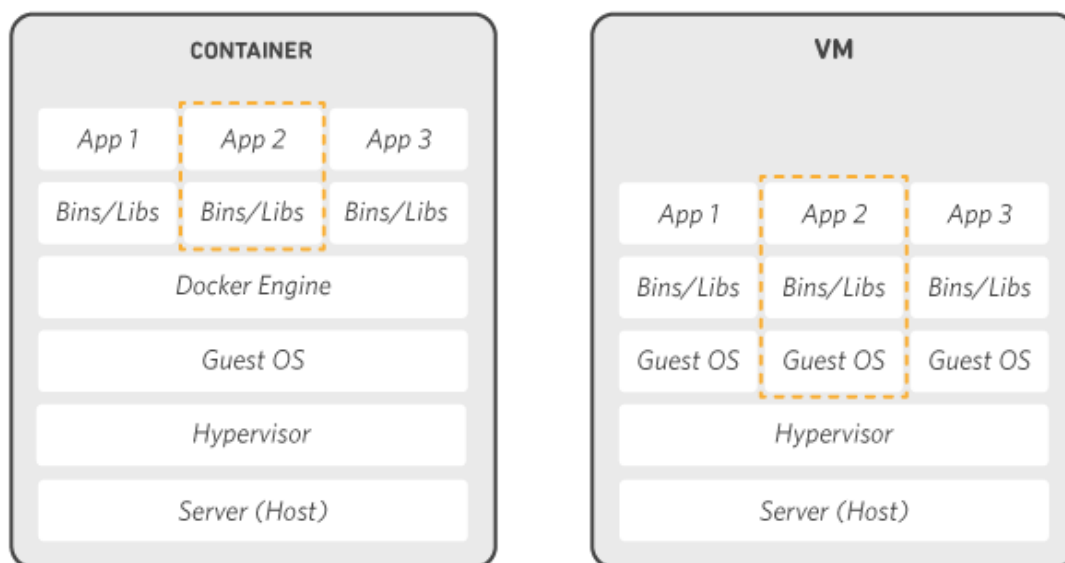
Con Docker, puede implementar y escalar aplicaciones rápidamente en cualquier entorno con la certeza de que su código se ejecutará. Docker empaqueta software en unidades estandarizadas llamadas contenedores, que incluyen todo lo necesario para que el software se ejecute, desde bibliotecas hasta herramientas de sistema y tiempo de ejecución.

Cuándo usar Docker:

Los contenedores Docker son el bloque de construcción principal para crear aplicaciones y plataformas modernas. Facilitan la creación de arquitecturas de microservicios, la implementación con canalizaciones de entrega continua, la creación de sistemas de procesamiento de datos escalables y la creación de plataformas completamente administradas para desarrolladores.

Conceptos Clave:

- Imagen de Docker: Una plantilla de solo lectura que define un contenedor.
- Contenedor de Docker: Almacena código junto con las dependencias necesarias para ejecutar aplicaciones de forma rápida y sencilla.
- Diferencia con Máquina Virtual: Mientras las máquinas virtuales virtualizan el hardware del servidor, los contenedores virtualizan el sistema operativo. Docker actúa como un sistema operativo para contenedores, simplificando la gestión y el despliegue de aplicaciones.



Instalación de Docker

Docker desktop

Para trabajar con Docker en Windows Professional o Enterprise, es necesario cumplir con ciertos requisitos y realizar algunos pasos de configuración. Aquí están los detalles importantes:

Requisitos:

- Windows 10 versión 21H1 o superior.
- Windows 11 versión 21H2 o superior.
- Hyper-V debe estar habilitado en ambos sistemas operativos.
- Los equipos proporcionados por Bosonit ya cumplen con estos requisitos, lo que simplifica el proceso de configuración.

Descarga de Docker Desktop:

Para instalar Docker Desktop, simplemente visita el sitio web oficial de Docker Desktop y descarga la versión correspondiente a tu sistema operativo Windows. Sigue las instrucciones de instalación proporcionadas por el instalador de Docker Desktop.

Una vez instalado, podrás comenzar a utilizar Docker para desarrollar, empaquetar y ejecutar aplicaciones en contenedores en tu entorno de Windows.

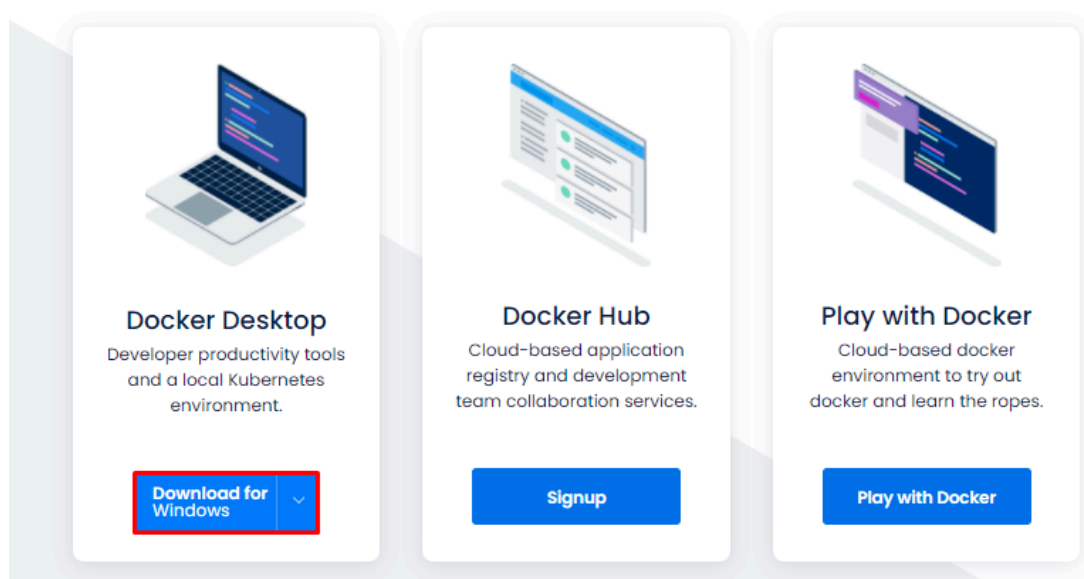


Recuerda que Docker Desktop [Docker Desktop](#) ofrece una interfaz intuitiva y herramientas potentes para la gestión de contenedores en tu entorno de desarrollo. ¡Disfruta de trabajar con Docker en Windows!

Docker viene por defecto instalado en la mayoría de distribuciones de Linux.

Get Started with Docker

We have a complete container solution for you – no matter who you are and where you are on your containerization journey.



Una vez tengamos el instalador descargado realizamos la instalación con el tradicional siguiente, siguiente...

Aunque me gustaría destacar que vamos a marcar las siguientes opciones:



Installing Docker Desktop 3.5.1 (66090)



Configuration

- ☒ Install required Windows components for WSL 2
- ☒ Add shortcut to desktop

Ok

Installing Docker Desktop 3.5.1 (66090)



Docker Desktop 3.5.1

Unpacking files...

```
Unpacking file: resources/docker-desktop.iso
Unpacking file: resources/ddvp.ico
Unpacking file: resources/config-options.json
Unpacking file: resources/componentsVersion.json
Unpacking file: resources/bin/docker-compose
Unpacking file: resources/bin/docker
Unpacking file: resources/.gitignore
Unpacking file: InstallerCli.pdb
Unpacking file: InstallerCli.exe.config
Unpacking file: frontend/vk_swiftshader_icd.json
Unpacking file: frontend/v8_context_snapshot.bin
```

Instalación de W2L (Windows Subsystem for Linux)

Al instalar Docker en Windows, es posible que se requiera la activación del WSL 2 (Windows Subsystem for Linux). A continuación, se detallan los pasos para llevar a cabo esta instalación:



WSL 2 installation is incomplete.



The WSL 2 Linux kernel is now installed using a separate MSI update package. Please click the link and follow the instructions to install the kernel update:

<https://aka.ms/wsl2kernel>.

Press Restart after installing the Linux kernel.

Restart

Cancel

WSL2 no permite ejecutar el kernel nativo de Linux al ejecutarse completamente desde un equipo Windows.


Reinicio del equipo:

- Después de la instalación de Docker, si se pulsa sobre "cerrar", el ordenador se reiniciará. Esto es necesario porque Docker virtualiza a nivel de sistema operativo y requiere el reinicio del equipo para aplicar los cambios correctamente.
- Verificación de WSL 2: Tras la instalación y reinicio, al abrir Docker, es posible que aparezca una ventana indicando que no se tiene instalado el WSL 2.
- Instalación de WSL 2: Para instalar el Windows Subsystem for Linux (WSL) 2, se pueden seguir los siguientes pasos:
 1. Abrir la Configuración de Windows.
 2. Navegar a "Actualización y seguridad" > "Para desarrolladores" y habilitar la función "Modo programador".
 3. Luego, ir a "Activar o desactivar las características de Windows" y activar la función "Subsistema de Windows para Linux".
 4. Una vez habilitado, instalar una distribución de Linux desde la Microsoft Store, como Ubuntu o Debian.
 5. Abrir la distribución instalada y seguir las instrucciones para completar la configuración inicial.
 6. Configuración de Docker para utilizar WSL 2:
- Después de instalar WSL 2, es posible que sea necesario configurar Docker para que lo utilice como backend. Para hacer esto, en la configuración de Docker Desktop, seleccionar "WSL Integration" y habilitar la integración con la distribución de Linux instalada en el paso anterior.



Siguiendo estos pasos, se debería completar la instalación de WSL 2 y configurar Docker para que funcione correctamente en el entorno de Windows.

Guía de instalación del Subsistema de Windows para Linux para Windows 10

07/04/2021 • Tiempo de lectura: 11 minutos • 

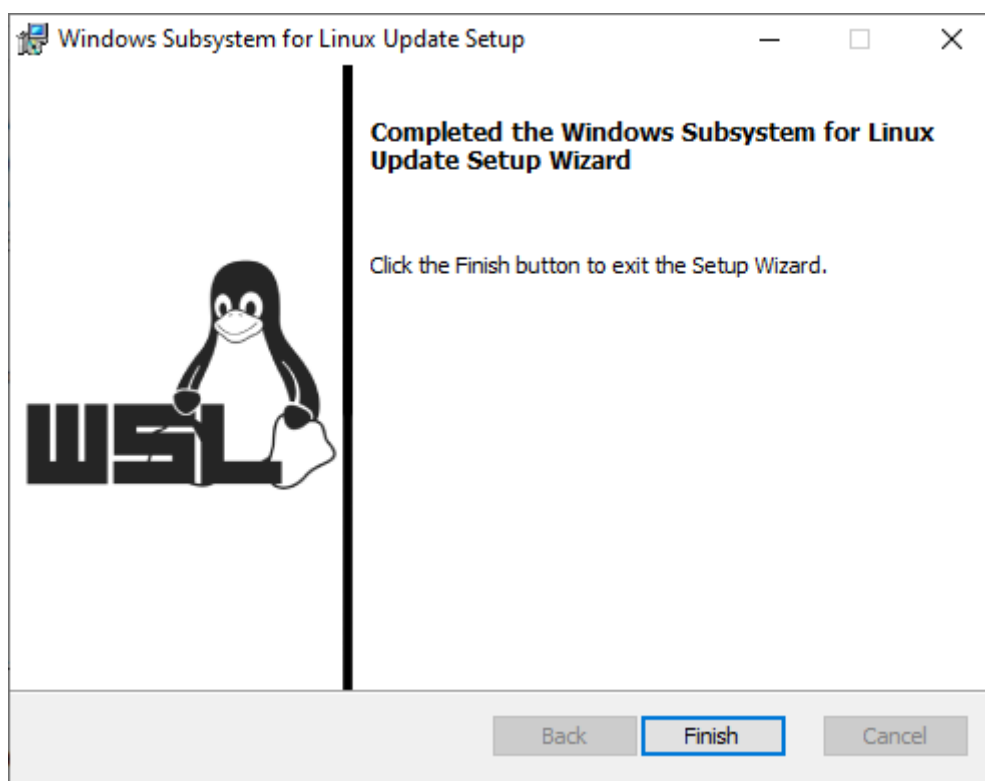
Hay dos opciones disponibles para instalar el Subsistema de Windows para Linux (WSL):

- **Instalación simplificada** (versión preliminar): `wsl --install`

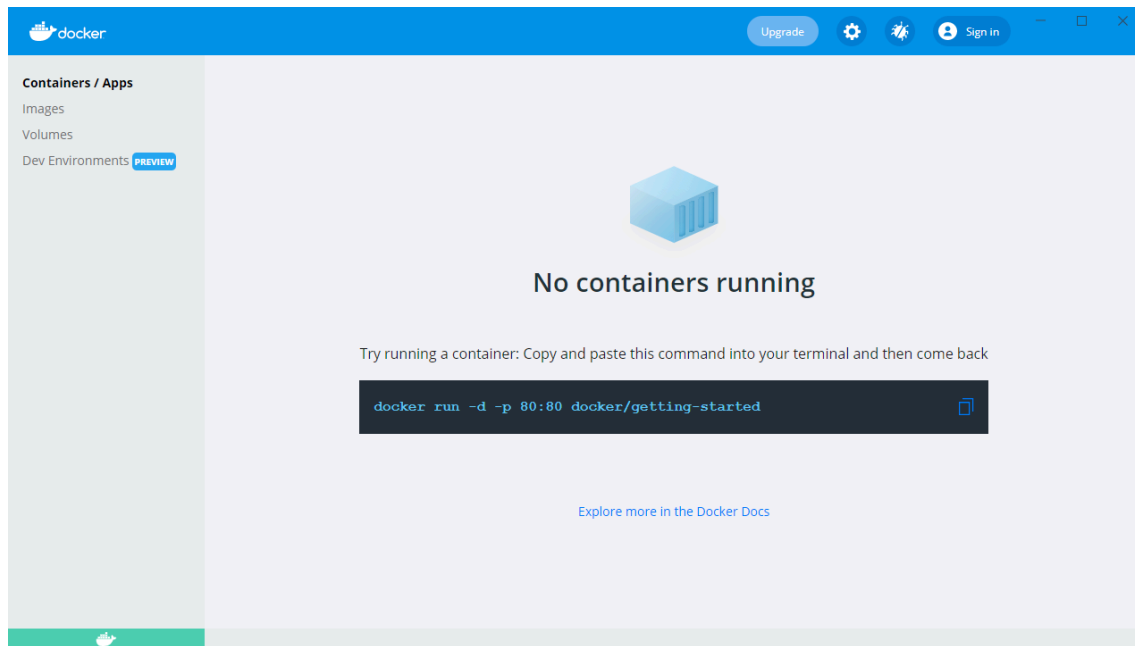
El comando de instalación simplificada `wsl --install` requiere que se una al [Programa Windows Insider](#) e instalar una versión preliminar de Windows 10 (compilación del sistema operativo 20262 o posterior), pero elimina la necesidad de seguir los pasos de instalación manual. Lo único que debe hacer es abrir una ventana Comandos con privilegios de administrador y ejecutar `wsl --install`. Después de un reinicio, estará listo para usar WSL.

- **Instalación manual**: realice los seis pasos que se indican a continuación.

Los pasos de instalación manual de WSL se enumeran a continuación y se pueden usar para instalar Linux en cualquier versión de Windows 10.



Finalmente, y una vez tenemos WSL 2 instalando, si iniciamos nuevamente Docker y vemos la siguiente ventana:



Si ahora abrimos el CMD y ejecutamos `docker --version`, podemos ver que también tenemos configurada la variable de entorno de Windows. Vamos a verlo:

```
C:\> Seleccionar C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Versión 10.0.19042.1052]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\David> docker --version
Docker version 20.10.7, build f0df350

C:\Users\David>
```

Si queremos la versión más detallada podemos hacer un `docker version` la cual nos mostrará las diferentes versiones con las que estamos trabajando tanto en el cliente como en el servidor:



```
C:\WINDOWS\system32\cmd.exe

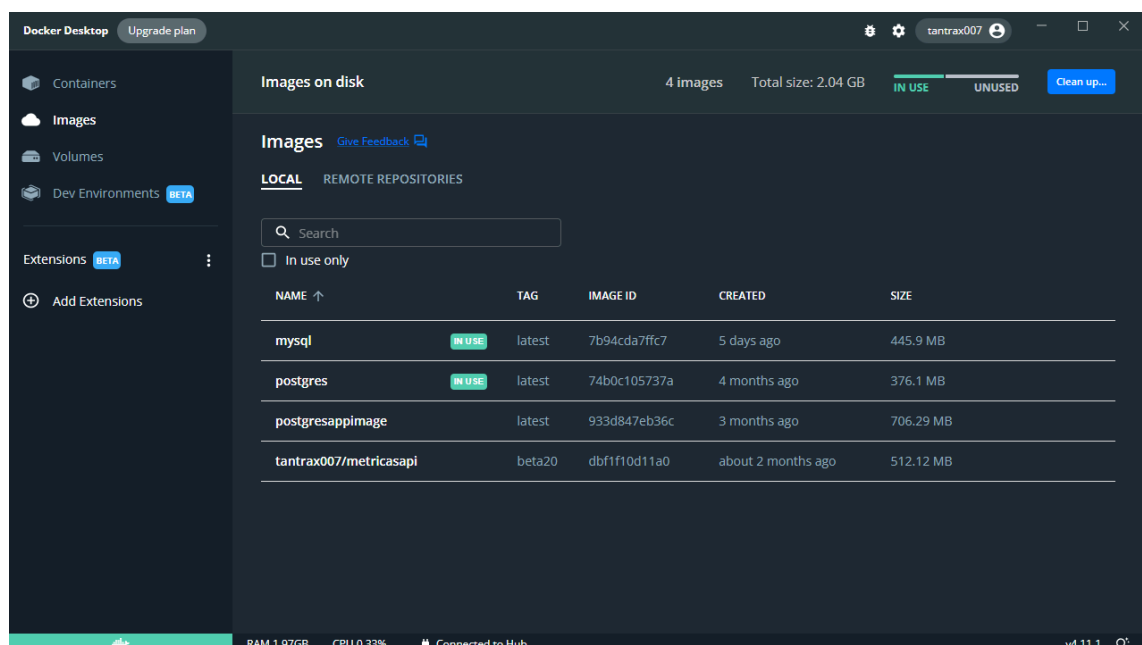
C:\Users\David>docker version
Client:
 Cloud integration: 1.0.17
 Version: 20.10.7
 API version: 1.41
 Go version: go1.16.4
 Git commit: f0df350
 Built: Wed Jun  2 12:00:56 2021
 OS/Arch: windows/amd64
 Context: desktop-linux
 Experimental: true

Server: Docker Engine - Community
 Engine:
  Version: 20.10.7
  API version: 1.41 (minimum version 1.12)
  Go version: go1.13.15
  Git commit: b0f5bc3
  Built: Wed Jun  2 11:54:58 2021
  OS/Arch: linux/amd64
  Experimental: false
 containerd:
  Version: 1.4.6
  GitCommit: d71fcd7d8303cbf684402823e425e9dd2e99285d
 runc:
  Version: 1.0.0-rc95
  GitCommit: b9ee9c6314599f1b4a7f497e1f1f856fe433d3b7
 docker-init:
  Version: 0.19.0
  GitCommit: de40ad0

C:\Users\David>
```

Esta aplicación tiene unas secciones que vamos a ver por encima:

- **Images:** En esta sección podremos ver aquellas imágenes que tengamos descargadas en el equipo. En aquellas imágenes de las cuales hayamos creado contenedores podremos ver un IN USE.



Desde el botón azul “Clean up...” podemos llegar a eliminar imágenes de una forma sencilla. Docker Desktop nos da dos opciones:



4 images Total size: 2.04 GB **IN USE** UNUSED [Clean up...](#)

Clean up images ☐ Unused ⓘ ☐ Dangling ⓘ Space to be reclaimed 0 Bytes [Cancel](#) [Remove](#)

Images [Give Feedback](#) ⓘ

LOCAL REMOTE REPOSITORIES

☐ In use only

	NAME ↑		TAG	IMAGE ID	CREATED	SIZE
<input type="checkbox"/>	mysql	IN USE	latest	7b94cda7ffc7	5 days ago	445.9 MB
<input type="checkbox"/>	postgres	IN USE	latest	74b0c105737a	4 months ago	376.1 MB
<input type="checkbox"/>	postgresappimage		latest	933d847eb36c	3 months ago	706.29 MB
<input type="checkbox"/>	tantrax007/metricasapi		beta20	dbf1f10d11a0	about 2 months ago	512.12 MB

- Opciones:
 - Unused: Como nos podemos imaginar sirve para eliminar aquellas imágenes que no se estén utilizando en ningún contenedor activo o parado.

☒ Unused ⓘ ☐ Dangling ⓘ

An image which is not used by any running or stopped container



Images on disk 4 images Total size: 2.04 GB IN USE UNUSED [Clean up...](#)

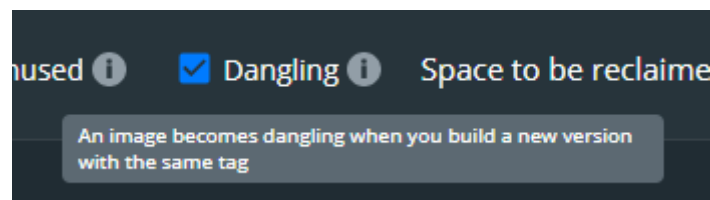
Images [Give Feedback](#)

LOCAL REMOTE REPOSITORIES

☐ In use only

NAME ↑		TAG	IMAGE ID	CREATED	SIZE
mysql	IN USE	latest	7b94cda7ffc7	5 days ago	445.9 MB
postgres	IN USE	latest	74b0c105737a	4 months ago	376.1 MB
postgresappimage		latest	933d847eb36c	3 months ago	706.29 MB
tantrax007/metricasapi		beta20	dbf1f10d11a0	about 2 months ago	512.12 MB

- Dangling: Elimina aquellas imágenes que tienen una versión superior con el mismo tag



- Si ninguna de estas dos opciones nos sirven pues podemos seleccionar manualmente aquellas que no queremos y simplemente aplicar el cambio:



Images on disk 4 images Total size: 2.04 GB IN USE UNUSED [Clean up...](#)

Images [Give Feedback](#)

LOCAL REMOTE REPOSITORIES

☐ In use only

NAME ↑		TAG	IMAGE ID	CREATED	SIZE
mysql	IN USE	latest	7b94cda7ffc7	5 days ago	445.9 MB
postgres	IN USE	latest	74b0c105737a	4 months ago	376.1 MB
postgresappimage		latest	933d847eb36c	3 months ago	706.29 MB
tantrax007/metricasapi		beta20	dbf1f10d11a0	about 2 months ago	512.12 MB

- Containers: En esta sección podremos ver los contenedores que tengamos creados.

Docker Desktop Upgrade plan tantrax007

Containers [Give Feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

Showing 3 items

	NAME	IMAGE	STATUS	PORT(S)	STARTED
<input type="checkbox"/>	competent_khorana b3029a94df57	mysql:latest	Exited (255)	3306	▶ 🗑️
<input type="checkbox"/>	postgres_test 425ccd43d982	postgres:latest	Exited (255)	-	▶ 🗑️
<input type="checkbox"/>	Postgress ae0e9dd76c78	postgres:latest	Exited (255)	5432	▶ 🗑️

RAM 1.92GB CPU 0.11% Connected to Hub v4.11.1

Para activar, parar o reiniciar un contenedor, es tan sencillo como clicar sobre el botón de play:



Containers [Give Feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

Showing 3 items Search

<input type="checkbox"/>		NAME	IMAGE	STATUS	PORT(S)	STARTED	
<input type="checkbox"/>		competent_khorana b3029a94df57	mysql:latest	Exited	3306		
<input type="checkbox"/>		postgres_test 425ccd43d982	postgres:latest	Exited (255)	-		
<input type="checkbox"/>		Postgress ae0e9dd76c78	postgres:latest	Exited (255)	5432		

Y para eliminar un contenedor es también super sencillo, basta con clicar sobre la papelera:

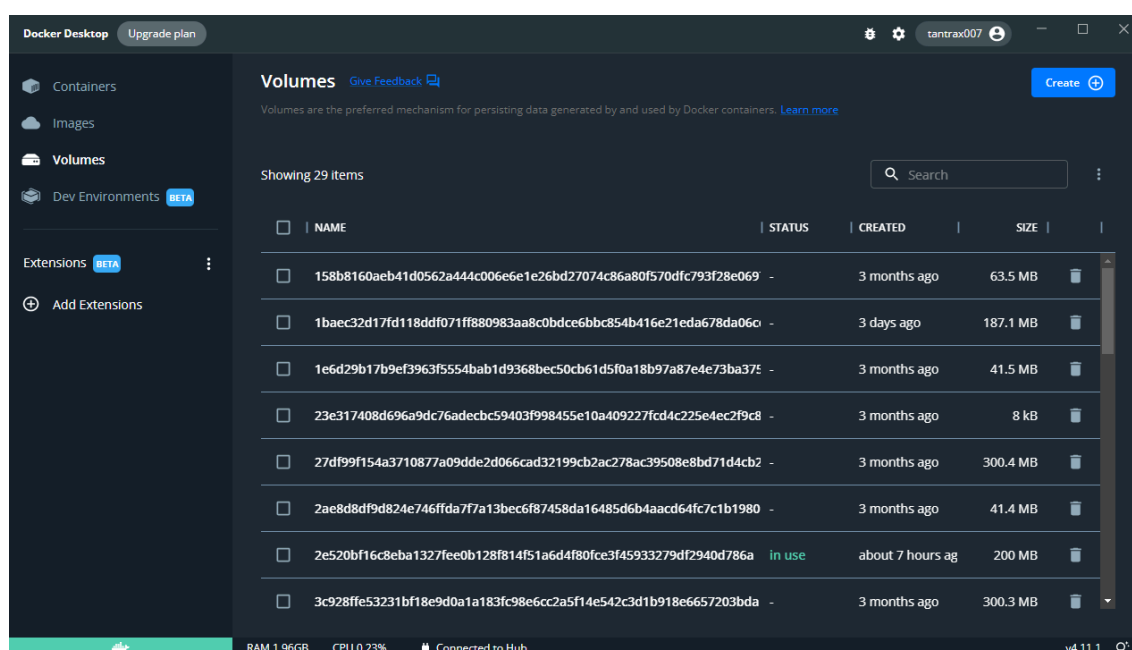
Containers [Give Feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

Showing 4 items Search

<input type="checkbox"/>		NAME	IMAGE	STATUS	PORT(S)	STARTED	
<input type="checkbox"/>		competent_khorana b3029a94df57	mysql:latest	Exited	3306		
<input type="checkbox"/>		postgres_test 425ccd43d982	postgres:latest	Exited (255)	-		
<input type="checkbox"/>		Postgress ae0e9dd76c78	postgres:latest	Exited (255)	5432		
<input type="checkbox"/>		DockerAEliminar a82220101c68	mysql:latest	Exited (1)	-		

- Volumes: Aquí veremos los volúmenes que tenemos y aquellos que estén en uso.



Luego tenemos las opciones en la periferia de la aplicación, en el título de la ventana podemos acceder a la configuración de nuestra cuenta:



Docker Hub

En el ecosistema de Docker, tanto para la descarga de imágenes como para el intercambio de las propias, se utilizan repositorios, que funcionan de manera similar a GitLab, donde almacenamos imágenes y facilitamos su acceso a otros usuarios.

En el mundo Docker, existen dos tipos principales de repositorios:

- **Públicos:** Estos repositorios son accesibles para cualquier persona. Los usuarios pueden buscar, descargar y contribuir a las imágenes almacenadas en ellos. Un ejemplo destacado de repositorio público es Docker Hub.
- **Privados:** Los repositorios privados están restringidos y solo pueden ser accedidos por usuarios autorizados. Estos repositorios son ideales para proyectos privados o empresas que desean mantener el control sobre sus imágenes de Docker.

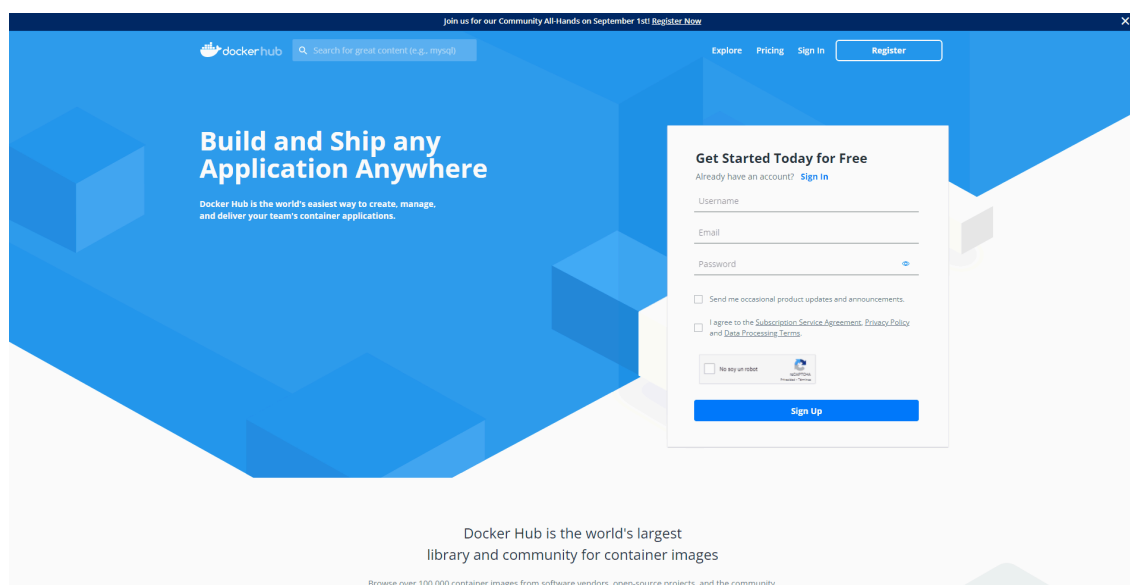
Docker Hub es el repositorio público más reconocido y seguro en el mundo de Docker. En Docker Hub, los usuarios pueden encontrar una amplia gama de



imágenes oficiales, como MySQL, Python, Node.js, entre otras, que son mantenidas por la comunidad y por los propios desarrolladores de Docker.

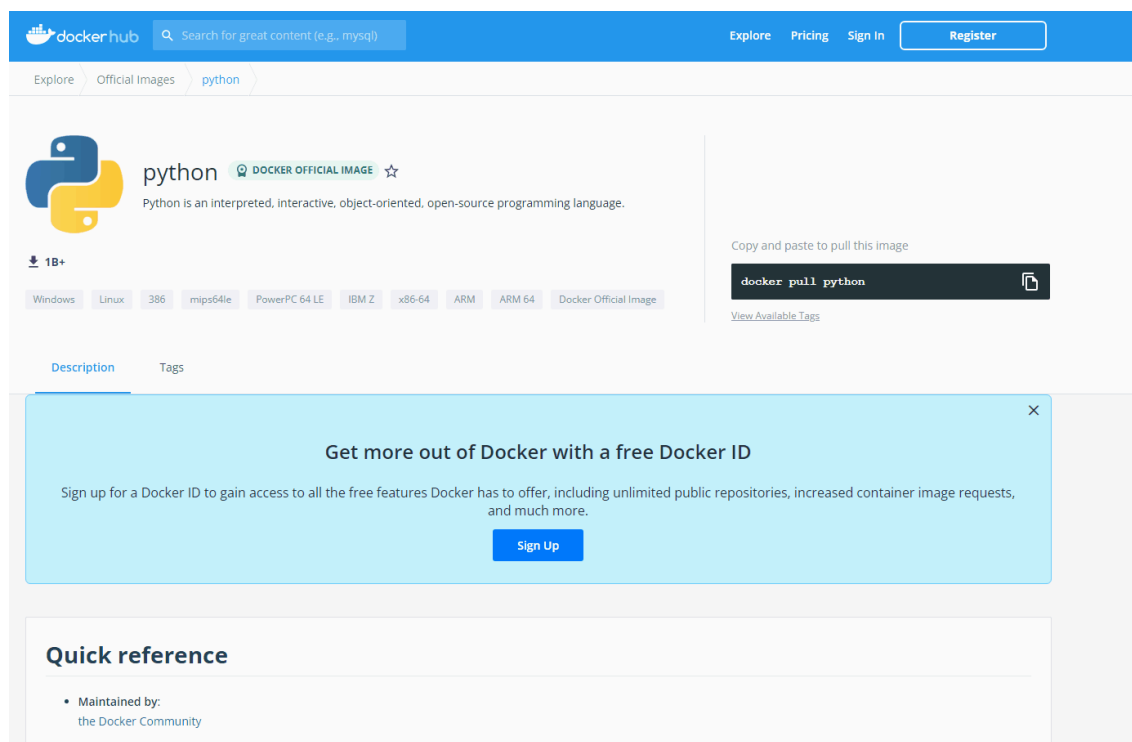
En resumen, los repositorios en Docker desempeñan un papel crucial en la distribución y compartición de imágenes de contenedores, proporcionando un punto centralizado para la colaboración y el intercambio de recursos en el ecosistema de Docker.

Vamos a movernos a [la web de DockerHub](https://hub.docker.com/) para echarle un vistazo:

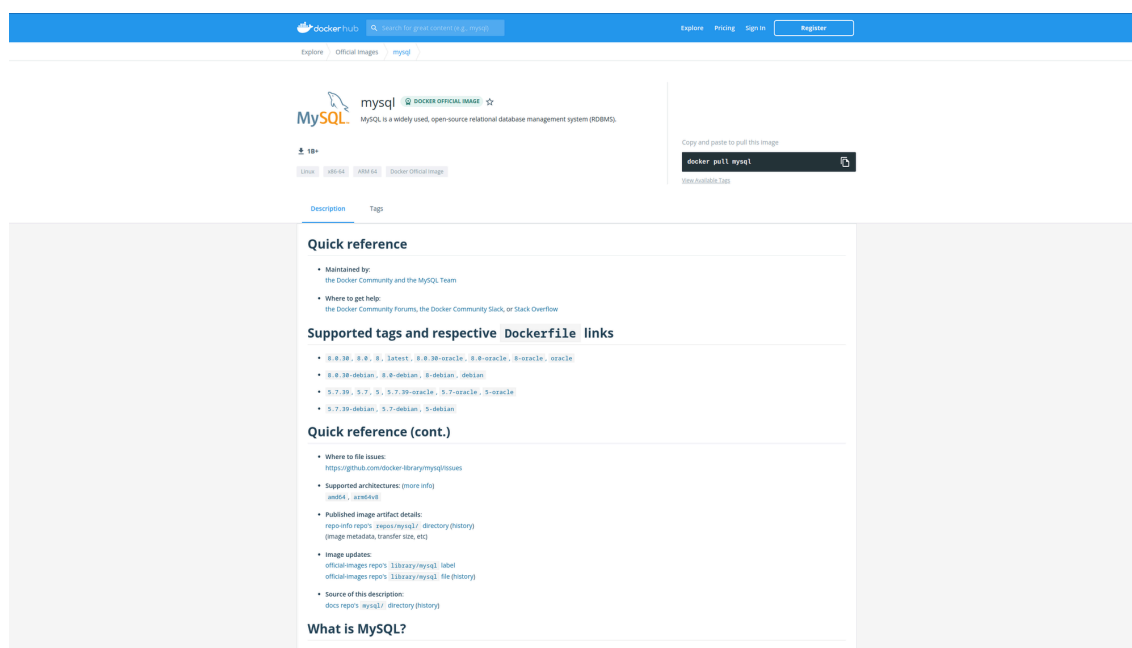


En la imagen actual vemos cómo podemos acceder a las imágenes iniciales que se nos proponen en la landing page y que son aquellas de mayor interés.

De todas maneras también podemos, en su buscador, buscar aquella imagen específica que necesitemos:



Una vez tenemos localizada la imagen que queremos nos situamos en su subpágina para leer la documentación acerca de ella:



Para este ejemplo hemos elegido la imagen de MySQL así que vamos a ver las partes que tiene la documentación:

Si la imagen que hemos seleccionado es oficial podremos ver una insignia de verificación indicándolo:



mysql DOCKER OFFICIAL IMAGE ☆

MySQL is a widely used, open-source relational database management system (RDBMS).

Descargas

Debajo de la descripción de la imagen podemos ver la cantidad estimada de descargas que tiene en total:



mysql DOCKER OFFICIAL IMAGE ☆

MySQL is a widely used, open-source relational database management system (RDBMS).

↓ 1B+

Etiquetas

Lo siguiente que podremos ver serán unas etiquetas de catalogación que no debemos confundir con las etiquetas de versionado:



mysql DOCKER OFFICIAL IMAGE ☆

MySQL is a widely used, open-source relational database management system (RDBMS).

↓ 1B+

Linux ARM 64 x86-64 Docker Official Image

Descarga

Por último y para completar el panel superior podemos ver el comando que necesitaremos utilizar para descargar la imagen:



mysql DOCKER OFFICIAL IMAGE ☆

MySQL is a widely used, open-source relational database management system (RDBMS).

↓ 1B+

Linux ARM 64 x86-64 Docker Official Image

Copy and paste to pull this image

```
docker pull mysql
```

[View Available Tags](#)



Documentación

Después de ver la cabecera de la web con la información principal del contenedor nos vamos a mover ahora al cuerpo, donde vamos a poder ver documentación sobre la imagen en concreto:

The screenshot shows the Docker Hub page for the MySQL official image. At the top, there are navigation links: 'Explore', 'Official Images', and 'mysql'. The main header features the MySQL logo, the text 'mysql', and a 'DOCKER OFFICIAL IMAGE' badge. Below this, it says 'MySQL is a widely used, open-source relational database management system (RDBMS)'. There is a download count of '18+' and tabs for 'Linux', 'ARM 64', 'x86-64', and 'Docker Official Image'. A 'Description' tab is selected. On the right, there is a 'Copy and paste to pull this image' section with a button that says 'docker pull mysql' and a 'View Available Tags' link. The main content area has a 'Quick reference' section with two bullet points: 'Maintained by: the Docker Community and the MySQL Team' and 'Where to get help: the Docker Community Forums, the Docker Community Slack, or Stack Overflow'. Below this is a section titled 'Supported tags and respective Dockerfile links' which contains a list of tags: '8.0.30', '8.0', '8', 'latest', '8.0.30-oracle', '8.0-oracle', '8-oracle', 'oracle', '8.0.30-debian', '8.0-debian', '8-debian', 'debian', '5.7.39', '5.7', '5', '5.7.39-oracle', '5.7-oracle', '5-oracle', and '5.7.39-debian', '5.7-debian', '5-debian'. At the bottom, there is a 'Quick reference (cont.)' section.

Esta sección de la web, a su vez, está dividida en subsecciones donde podremos encontrar información importante relativa a la imagen con la que queramos trabajar.

Por ejemplo, al inicio de todo tenemos la sección de etiquetas en la cual podemos saber qué tags soporta esta imagen:

Supported tags and respective Dockerfile links

- `8.0.30`, `8.0`, `8`, `latest`, `8.0.30-oracle`, `8.0-oracle`, `8-oracle`, `oracle`
- `8.0.30-debian`, `8.0-debian`, `8-debian`, `debian`
- `5.7.39`, `5.7`, `5`, `5.7.39-oracle`, `5.7-oracle`, `5-oracle`
- `5.7.39-debian`, `5.7-debian`, `5-debian`

Luego, por ejemplo, podemos encontrar la sección de “Cómo utilizar esta imagen” en la que nos explicarán junto a comandos, ejemplos de uso:



How to use this image

Start a `mysql` server instance

Starting a MySQL instance is simple:

```
$ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

... where `some-mysql` is the name you want to assign to your container, `my-secret-pw` is the password to be set for the MySQL root user and `tag` is the tag specifying the MySQL version you want. See the list above for relevant tags.

Connect to MySQL from the MySQL command line client

The following command starts another `mysql` container instance and runs the `mysql` command line client against your original `mysql` container, allowing you to execute SQL statements against your database instance:

```
$ docker run -it --network some-network --rm mysql mysql -hsome-mysql -uexample-user -p
```

... where `some-mysql` is the name of your original `mysql` container (connected to the `some-network` Docker network).

This image can also be used as a client for non-Docker or remote instances:

```
$ docker run -it --rm mysql mysql -hsome.mysql.host -usome-mysql-user -p
```

More information about the MySQL command line client can be found in the [MySQL documentation](#)

... via `docker stack deploy` or `docker-compose`

Example `stack.yml` for `mysql`:

```
# Use root/example as user/password credentials
version: '3.1'

services:

  db:
    image: mysql
    # NOTE: use of "mysql_native_password" is not recommended: https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-previous-ser:
    # (this is just an example, not intended to be a production configuration)
    command: --default-authentication-plugin=mysql_native_password
```

Y mucho más que puedes investigar y empezar a leer para descubrir.

Es altamente recomendable crear una cuenta en Docker Hub si planeas subir tus propias imágenes, compartirlas con otros usuarios o simplemente almacenarlas de manera segura. Aquí hay algunas razones por las cuales es beneficioso crear una cuenta en Docker Hub:

- **Almacenamiento Seguro:** Docker Hub proporciona un lugar seguro y confiable para almacenar tus imágenes de contenedores. Esto te permite acceder a ellas desde cualquier lugar y en cualquier momento.



- **Compartir y Colaborar:** Con una cuenta en Docker Hub, puedes compartir tus imágenes con otros usuarios de Docker Hub. Esto facilita la colaboración en proyectos de código abierto o la distribución de software entre equipos de desarrollo.
- **Acceso a Imágenes Oficiales:** Docker Hub alberga una amplia variedad de imágenes oficiales de contenedores para diferentes tecnologías y plataformas. Al tener una cuenta, puedes acceder fácilmente a estas imágenes y utilizarlas en tus proyectos.
- **Gestión de Imágenes:** Con una cuenta en Docker Hub, puedes gestionar tus imágenes de manera más eficiente. Puedes organizarlas en repositorios, etiquetarlas, actualizarlas y eliminarlas según sea necesario.
- **Integración con Herramientas:** Docker Hub se integra con varias herramientas de desarrollo y automatización, lo que facilita la integración continua y la entrega continua (CI/CD) en tus proyectos.

En resumen, crear una cuenta en Docker Hub te brinda una serie de beneficios que mejoran tu experiencia con Docker y te permiten aprovechar al máximo las capacidades de la plataforma. Además, es un paso importante si planeas contribuir a la comunidad de Docker compartiendo tus propias imágenes de contenedores.

Comandos de Docker

Si hacemos `docker help`, podemos ver todos los comandos que tenemos disponibles en Docker. Vamos a verlos:



```
C:\WINDOWS\system32\cmd.exe
```

```
Microsoft Windows [Versión 10.0.19042.1052]  
(c) Microsoft Corporation. Todos los derechos reservados.
```

```
C:\Users\David>docker --help
```

```
Usage: docker [OPTIONS] COMMAND
```

```
A self-sufficient runtime for containers
```

```
C:\WINDOWS\system32\cmd.exe
```

```
Commands:  
  attach      Attach local standard input, output, and error streams to a running container  
  build       Build an image from a Dockerfile  
  commit      Create a new image from a container's changes  
  cp          Copy files/folders between a container and the local filesystem  
  create      Create a new container  
  diff        Inspect changes to files or directories on a container's filesystem  
  events      Get real time events from the server  
  exec        Run a command in a running container  
  export      Export a container's filesystem as a tar archive  
  history     Show the history of an image  
  images      List images  
  import      Import the contents from a tarball to create a filesystem image  
  info        Display system-wide information  
  inspect     Return low-level information on Docker objects  
  kill        Kill one or more running containers  
  load        Load an image from a tar archive or STDIN  
  login       Log in to a Docker registry  
  logout      Log out from a Docker registry  
  logs        Fetch the logs of a container  
  pause       Pause all processes within one or more containers  
  port        List port mappings or a specific mapping for the container  
  ps          List containers  
  pull        Pull an image or a repository from a registry  
  push        Push an image or a repository to a registry  
  rename      Rename a container  
  restart     Restart one or more containers  
  rm          Remove one or more containers  
  rmi         Remove one or more images  
  run         Run a command in a new container  
  save        Save one or more images to a tar archive (streamed to STDOUT by default)  
  search      Search the Docker Hub for images  
  start       Start one or more stopped containers  
  stats       Display a live stream of container(s) resource usage statistics  
  stop        Stop one or more running containers  
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE  
  top         Display the running processes of a container  
  unpause     Unpause all processes within one or more containers  
  update      Update configuration of one or more containers  
  version     Show the Docker version information  
  wait        Block until one or more containers stop, then print their exit codes  
  
Run 'docker COMMAND --help' for more information on a command.  
  
To get more help with docker, check out our guides at https://docs.docker.com/go/guides/
```

Docker info: muestra toda la información del sistema sobre el que se ejecuta docker. La versión del kernel, el número de contenedores y las imágenes.



```
C:\Users\David>docker info
Client:
Context:    desktop-linux
Debug Mode: false
Plugins:
 buildx: Build with BuildKit (Docker Inc., v0.5.1-docker)
 compose: Docker Compose (Docker Inc., 2.0.0-beta.4)
 scan: Docker Scan (Docker Inc., v0.8.0)
Server:
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 20.10.7
Storage Driver: overlay2
```

Funcionamiento de Docker

El motivo por el cual nuestro contenedor no está corriendo inmediatamente después de crearlo es una diferencia fundamental entre los contenedores y las máquinas virtuales.

Cuando creamos un contenedor a partir de una imagen, como por ejemplo un contenedor Ubuntu, necesitamos que este contenedor tenga algún tipo de programa o proceso (daemon) que lo mantenga en ejecución de manera continua. De lo contrario, el contenedor se detendrá tan pronto como ejecutemos el comando `docker run`. Esto es porque el propósito de los contenedores es albergar aplicaciones o servicios a los que podamos acceder y obtener un servicio, como por ejemplo una base de datos.

Por lo tanto, necesitamos garantizar que el contenedor no se detenga inmediatamente después de iniciar. Una forma de lograr esto es utilizando la bandera `-it` en el comando `docker run`. Esta bandera nos permite interactuar con el contenedor que estamos creando, lo que evita que se detenga de inmediato y nos brinda la capacidad de interactuar con él a través de la línea de comandos.

En resumen, al utilizar la bandera `-it` al crear un contenedor con el comando `docker run`, podemos evitar que se detenga inmediatamente después de su inicio y podemos interactuar con él de manera efectiva para ejecutar aplicaciones o servicios dentro del contenedor.

Con esta bandera Docker coloca una pseudo-TTY conectada al contenedor de forma que podamos, mediante un Bash poder ejecutar instrucciones.



```
root@2b3f221a147e: /  
C:\Users\nahuel.capuzzi>docker run -it ubuntu:latest  
root@2b3f221a147e:/#
```

```
root@2b3f221a147e: /  
root@2b3f221a147e:/# uname -a  
Linux 2b3f221a147e 5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux  
root@2b3f221a147e:/#
```

Como vemos, ahora gracias a ese parámetro, estamos dentro del contenedor Ubuntu que hemos creado. Con esto lo que hacemos es utilizar la bash de Linux que, en el fondo es un “programa” que mantiene nuestro contenedor ejecutándose.

```
Microsoft Windows [Versión 10.0.19044.1826]  
(c) Microsoft Corporation. Todos los derechos reservados.  
C:\Users\nahuel.capuzzi>docker ps  
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES  
2b3f221a147e   ubuntu:latest "bash"                 3 minutes ago Up 3 minutes          agitated_blackwell  
C:\Users\nahuel.capuzzi>  
root@2b3f221a147e: /  
root@2b3f221a147e:/# uname -a  
Linux 2b3f221a147e 5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux  
root@2b3f221a147e:/#
```

Coacción a los contenedores

¿Y si te digo que hay una forma de hacer que ese ubuntu se quede “obligado” a ejecutarse hasta que nosotros lo paremos? Pues sí, podemos.

Hemos visto cómo funciona la bandera -it para “conectarnos” a un contenedor como con Ubuntu. Pues esta bandera la vamos a utilizar junto a -d para que esa TTY se quede ejecutándose en segundo plano continuamente con el proceso de la bash que crea Docker:

```
C:\Users\nahuel.capuzzi>docker run -itd --name pendiente ubuntu:latest  
65b1d0a87d58f81078653ee6cfe11baa4582ba1aae430b45f78513e78f3cd563  
C:\Users\nahuel.capuzzi>docker ps  
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES  
65b1d0a87d58   ubuntu:latest "bash"                 6 seconds ago Up 4 seconds          pendiente  
C:\Users\nahuel.capuzzi>
```

Como podemos ver en el reporte del docker ps el contenedor pendiente está ejecutándose con el comando bash y eso es justo lo que queríamos.



Trabajando con Docker

Es fundamental comprender los comandos básicos de Docker para poder trabajar de manera efectiva con esta herramienta, independientemente de si utilizamos una aplicación gráfica como Docker Desktop o la línea de comandos en la consola. Los comandos básicos son los mismos en cualquier sistema operativo y nos permiten administrar contenedores, imágenes, volúmenes y otras características de Docker.

A continuación, se presentan algunos de los comandos básicos de Docker que serán útiles para realizar diversas tareas:

- `docker pull <imagen>`: Descarga una imagen de Docker desde un repositorio.
- `docker run <imagen>`: Crea y ejecuta un contenedor a partir de una imagen.
- `docker ps`: Muestra todos los contenedores en ejecución en el sistema.
- `docker ps -a`: Muestra todos los contenedores en el sistema, incluidos los que están detenidos.
- `docker stop <contenedor>`: Detiene un contenedor en ejecución.
- `docker rm <contenedor>`: Elimina un contenedor.
- `docker images`: Muestra todas las imágenes de Docker en el sistema.
- `docker rmi <imagen>`: Elimina una imagen de Docker del sistema.
- `docker exec -it <contenedor> <comando>`: Ejecuta un comando dentro de un contenedor en ejecución.
- `docker logs <contenedor>`: Muestra los registros de un contenedor específico.
- `docker volume ls`: Lista todos los volúmenes de Docker en el sistema.
- `docker volume rm <volumen>`: Elimina un volumen de Docker.

Estos son solo algunos de los comandos básicos de Docker que se utilizan con mayor frecuencia. Es importante familiarizarse con ellos y practicar su uso para poder aprovechar al máximo la potencia de Docker en el desarrollo y despliegue de aplicaciones en contenedores.

¿Qué es una imagen de Docker?

Una imagen de Docker es un paquete ligero y autónomo que contiene todo lo necesario para ejecutar una aplicación: el código, las bibliotecas del sistema, las herramientas y las configuraciones. Esencialmente, una imagen de Docker es como una "plantilla" que se utiliza para crear contenedores en Docker.

Características de las imágenes de Docker:



- Portabilidad: Las imágenes de Docker son portátiles y se pueden ejecutar en cualquier entorno que admita Docker, lo que facilita la implementación y la migración de aplicaciones.
- Compartibilidad: Las imágenes de Docker se pueden compartir y distribuir fácilmente a través de repositorios como Docker Hub, lo que facilita la colaboración y la reutilización de software entre equipos de desarrollo.
- Eficiencia: Las imágenes de Docker son livianas y eficientes en cuanto a recursos, ya que comparten el núcleo del sistema operativo del host y utilizan capas de almacenamiento en caché para optimizar el uso de la memoria y el espacio en disco.

Ejemplo con imágenes

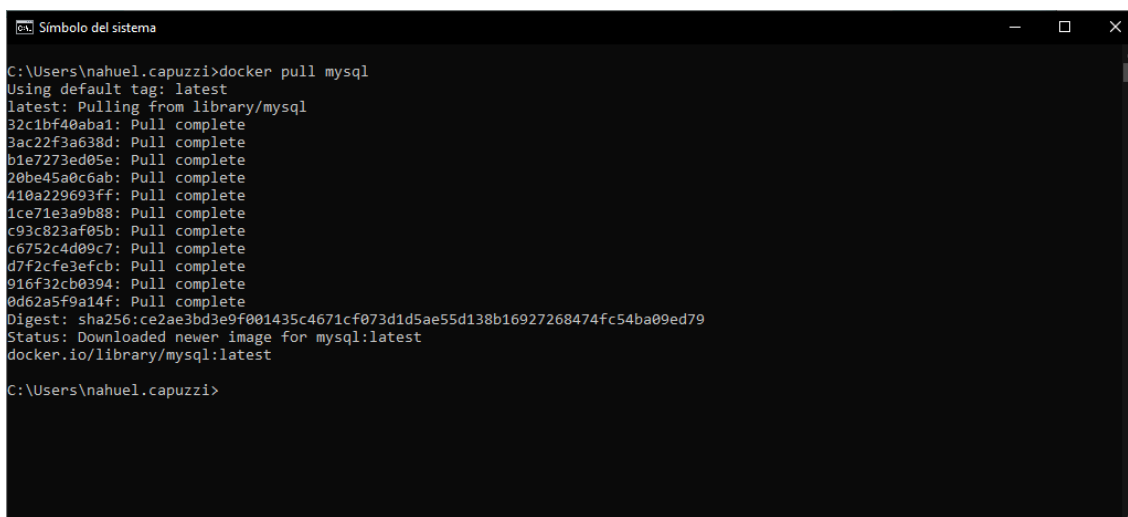
Para trabajar con imágenes de Docker, primero necesitamos descargarlas. Utilizamos el comando *docker pull* seguido del nombre de la imagen que deseamos obtener. La sintaxis del comando es *docker pull <imageName:tag>*. Si no especificamos un tag entonces docker nos descargará por defecto la *latest*:

Por ejemplo, para obtener la imagen de MySQL, ejecutaríamos:

```
docker pull mysql
```

Este comando descarga la imagen de MySQL desde un repositorio de imágenes de Docker, como Docker Hub, y la almacena localmente en nuestra máquina.

Una vez descargada la imagen, podemos utilizarla para crear y ejecutar contenedores de MySQL o para realizar otras operaciones relacionadas con la gestión de contenedores.



```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
32c1bf40aba1: Pull complete
3ac22f3a638d: Pull complete
b1e7273ed05e: Pull complete
20be45a0c6ab: Pull complete
410a229693ff: Pull complete
1ce71e3a9b88: Pull complete
c03c823af05b: Pull complete
c6752c4d09c7: Pull complete
d7f2cfe3efcb: Pull complete
916f32cb0394: Pull complete
0d62a5f9a14f: Pull complete
Digest: sha256:ce2ae3bd3e9f001435c4671cf073d1d5ae55d138b16927268474fc54ba09ed79
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest
C:\Users\nahuel.capuzzi>
```



Visualizar Imágenes

Para ver qué imágenes tenemos descargadas en nuestro equipo, podemos utilizar el comando *docker images* que nos mostrará algo como lo siguiente:

```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker images
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
mysql                latest      7b94cda7ffc7  5 days ago   446MB
tantrax007/metricasapi beta20      dbf1f10d11a0  6 weeks ago  512MB
postgresappimage     latest      933d847eb36c  3 months ago 706MB
postgres             latest      74b0c105737a  3 months ago 376MB
C:\Users\nahuel.capuzzi>
```

Buscar una imagen

Puede ser que, en ocasiones, necesitemos buscar una imagen en el repositorio por el motivo que sea y no nos apetece ir hasta docker hub en la web. Pues para ello contamos con un comando que podremos utilizar para ello:

docker search <imageName>

Este comando nos va a devolver una lista con información crucial acerca de todas las imágenes que haya encontrado:

```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker search mysql
NAME                DESCRIPTION                               STARS     OFFICIAL   AUTOMATED
mysql               MySQL is a widely used, open-source relation... 12988     [OK]
mariadb            MariaDB Server is a high performing open sou... 4977      [OK]
phpmyadmin         phpMyAdmin - A web interface for MySQL and M... 596       [OK]
percona            Percona Server is a fork of the MySQL relati... 583       [OK]
bitnami/mysql      Bitnami MySQL Docker Image                 72        [OK]
linuxserver/mysql-workbench 41
linuxserver/mysql  A Mysql container, brought to you by LinuxSe... 37
ubuntu/mysql       MySQL open source fast, stable, multi-thread... 36
circleci/mysql     MySQL is a widely used, open-source relation... 26
google/mysql       MySQL server for Google Compute Engine       21        [OK]
rapidfort/mysql    RapidFort optimized, hardened image for MySQL 13
bitnami/mysqld-exporter 3
ibmcom/mysql-s390x  Docker image for mysql-s390x                 2
newrelic/mysql-plugin New Relic Plugin for monitoring MySQL databa... 1          [OK]
vitess/mysqlctl    vitess/mysqlctl                                1          [OK]
hashicorp/mysql-portworx-demo 0
docksal/mysql      MySQL service images for Docksal - https://d... 0
mirantis/mysql     0
cimg/mysql         0
drud/mysql         0
silintl/mysql-backup-restore Simple docker image to perform mysql backups... 0          [OK]
corpusops/mysql    https://github.com/corpusops/docker-images/  0
drud/mysql-local-57 ddev mysql local container                   0
drud/mysql-docker-local-57 This repo has been deprecated, new tags are ... 0
drud/mysql-docker-local docker containers for local womysql rk        0          [OK]
C:\Users\nahuel.capuzzi>
```

Vemos que tenemos un listado amplio con varias columnas:

- Nombre
- Descripción
- Estrellas
- Oficial



- Automatizado

Esta es toda la información que necesitaremos para seleccionar aquella imagen que más nos convenga.

Eliminar Imágenes

Podría llegar a darse el caso en el que descarguemos una imagen que no sea necesaria. Para ello tenemos otro maravilloso comando: *docker rmi <imageID>*.

```
C:\Users\nahuel.capuzzi>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mysql                latest              7b94cda7ffc7       5 days ago         446MB
node                 latest              3adbe565b1f0       7 days ago         991MB
tantrax007/metricasapi beta20             dbf1f10d11a0       6 weeks ago        512MB
postgresappimage     latest              933d847eb36c       3 months ago       706MB
postgres             latest              74b0c105737a       3 months ago       376MB

C:\Users\nahuel.capuzzi>docker rmi node
Untagged: node:latest
Untagged: node@sha256:a6f295c2354992f827693a2603c8b9b5b487db4da0714f5913a917ed588d6d41
Deleted: sha256:3adbe565b1f05545a12f2acd51b5e77207cec7f7cf4dd4caa725d4503cd4fe7a
Deleted: sha256:287696ec9de96be43a5a43cacd9a1cf0477ad367188083b3f2ef11949c2414c
Deleted: sha256:2b775516f9788769bf7005bb1b02f252a61242d0fd16871b6a0a6c2b7444aa52
Deleted: sha256:f059818948dcfac265c93f95cf4f373a3eb9ec829aeda5f5142f70e06fa4dfa3
Deleted: sha256:529c085785b21731f3218732cc57a7c44777e323964b63639189b5b5184fed6
Deleted: sha256:836de6cec8cec75a4d73cb3cbcc645598743c587f04b8bb3238fc6760b0c858b
Deleted: sha256:1376fe23991c7bd9ac29c2469f6489e5e68b2311f78191e87c47acd67e846372
Deleted: sha256:935ab298b59cf4955c8a62f40960766ceedee432fde87f22a71d557be7e05d0a
Deleted: sha256:6fa094ba2e615e0fab64e7d1372945f05e70ed3bdf6fd90409153d7ec19d160
Deleted: sha256:9c742cd6c7a5752ee36be8ecb14be45c0885e10e6dd34f26a9ae3eb096c5d492

C:\Users\nahuel.capuzzi>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mysql                latest              7b94cda7ffc7       5 days ago         446MB
tantrax007/metricasapi beta20             dbf1f10d11a0       6 weeks ago        512MB
postgresappimage     latest              933d847eb36c       3 months ago       706MB
postgres             latest              74b0c105737a       3 months ago       376MB

C:\Users\nahuel.capuzzi>
```

¡Importante! No vamos a poder eliminar imágenes *IN USE* es decir, aquellas sobre las cuales tengamos contenedores creados.

```
C:\Users\nahuel.capuzzi>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mysql                latest              7b94cda7ffc7       4 days ago         446MB
ubuntu              latest              df5de72bdb3b       6 days ago         77.8MB
tantrax007/metricasapi beta20             dbf1f10d11a0       6 weeks ago        512MB
postgresappimage     latest              933d847eb36c       3 months ago       706MB
postgres             latest              74b0c105737a       3 months ago       376MB

C:\Users\nahuel.capuzzi>docker rmi mysql
Error response from daemon: conflict: unable to remove repository reference "mysql" (must force) - container b3029a94df57 is using its referenced image 7b94cda7ffc7
C:\Users\nahuel.capuzzi>
```

Como vimos en la sección de Docker Desktop, esta aplicación nos ofrecía una opción para acabar con esas imágenes que no estábamos utilizando. En la terminal podemos utilizar el *docker image prune*.



Creación de nuestras propias imágenes en Docker

Hasta ahora, hemos estado utilizando imágenes preexistentes disponibles en Docker Hub. Sin embargo, también podemos crear nuestras propias imágenes personalizadas, especialmente para aplicaciones que desarrollamos, como las basadas en Spring Boot.

Para demostrar este proceso, vamos a tomar una aplicación de ejemplo de Spring Boot y construiremos nuestra propia imagen Docker basada en ella.

Pasos para crear nuestra propia imagen:

1. Preparación de la aplicación Spring Boot:

Necesitamos tener una aplicación Spring Boot lista para ser empaquetada en una imagen de Docker. Esta aplicación debe estar configurada y lista para ser ejecutada.

2. Creación de un Dockerfile:

Un Dockerfile es un archivo de texto que contiene los pasos necesarios para construir una imagen Docker. En este archivo, especificamos desde qué imagen base vamos a construir nuestra imagen, copiamos los archivos de la aplicación, configuramos el entorno y ejecutamos cualquier comando necesario para preparar la imagen.

El contenido básico de un Dockerfile para una aplicación Spring Boot podría incluir:

```
FROM openjdk:11-jre-slim
WORKDIR /app
COPY target/nombre-del-archivo.jar app.jar
CMD ["java", "-jar", "app.jar"]
```

3. Construcción de la imagen:

Utilizamos el comando *docker build* para construir la imagen a partir del Dockerfile. Copy code:

```
docker build -t nombre-de-la-imagen:tag .
```



- **-t** : etiqueta o nombre de la imagen que estamos creando.
- **nombre-de-la-imagen:tag** : nombre y etiqueta que le asignamos a la imagen.
- **.** : indica que Docker debe buscar el Dockerfile en el directorio actual.

4. Verificación de la imagen:

Una vez completada la construcción, podemos verificar que la imagen se haya creado correctamente utilizando el comando *docker images*.

5. Ejecución de la imagen:

Por último, podemos ejecutar un contenedor basado en nuestra imagen utilizando el comando *docker run*.

```
docker run -d -p 8080:8080 nombre-de-la-imagen:tag
```

- **-d** : ejecuta el contenedor en segundo plano.
- **-p 8080:8080** : mapea el puerto 8080 del contenedor al puerto 8080 del host.

Siguiendo estos pasos, podemos crear nuestra propia imagen de Docker basada en una aplicación Spring Boot y ejecutarla como un contenedor Docker. Esto nos permite empaquetar nuestra aplicación de manera consistente y distribuirla en diferentes entornos de manera sencilla y reproducible.

Creando el Dockerfile

El primer paso que debemos seguir una vez hayamos ubicado la aplicación sobre la cual crearemos nuestra imagen es la creación del archivo *Dockerfile*. Este archivo contiene todas las configuraciones necesarias para que Docker pueda construir la imagen.

Es importante destacar que el nombre del archivo debe ser "Dockerfile" sin excepciones.

Vamos a ver lo que contiene este fichero:



```
Dockerfile x
1 FROM openjdk:11
2 EXPOSE 8081
3 COPY /target/*.jar /usr/local/lib/spring.jar
4 ENTRYPOINT ["java","-jar","/usr/local/lib/spring.jar"]
```

La primera instrucción que podemos ver es el `FROM`, esta es la que vamos a utilizar para indicarle a Docker a partir de qué imagen construirá la nueva ya que todas las imágenes tienen que partir de una.

FROM <image>:<tag>

A la imagen le podemos agregar el tag que queramos para seleccionar qué versión utilizar como estamos haciendo en el ejemplo. Estamos utilizando la imagen de `openjdk` y la tag `11`.

La segunda instrucción que tenemos es `EXPOSE` y esta la utilizamos para indicarle a Docker en qué puerto está escuchando nuestro contenedor. Además también podemos configurarlo para indicar si es en UDP o TCP.

`EXPOSE` no abre los puertos sino que funciona a modo de documentación entre la persona que crea la imagen y la persona que crea el contenedor. Esto sigue dependiendo de los puertos que se ponen detrás de la flag `-p`. Lo que quiere decir que aunque en el `EXPOSE` pongamos el puerto `8081` si a la hora de crear el contenedor pones `docker run -p 8080:8080` se nos mapeará el puerto `8080`. Por defecto `EXPOSE` asume que el protocolo es TCP, si quisiéramos utilizar UDP lo haríamos de la siguiente manera:

EXPOSE 80/udp

La tercera instrucción que tenemos es `COPY` que vamos a utilizar para, como podemos intuir por el nombre, copiar archivos o directorios desde la ruta que le indicamos al sistema de ficheros del contenedor.

COPY <src>... <dest>

En el ejemplo estamos cogiendo un fichero que se llame de cualquier forma y acabe en `.jar` y lo estamos moviendo a la ruta `/usr/local/lib/spring.jar`.



Por último tenemos la instrucción `ENTRYPOINT` que la vamos a utilizar para ejecutar un comando en el contenedor.

```
ENTRYPOINT ["executable", "param1", "param2"]
```

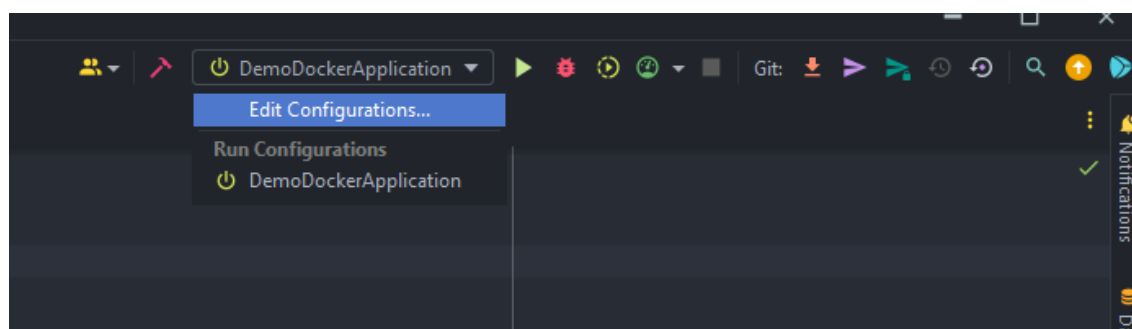
Este se puede sobrescribir al crear el contenedor con la bandera `docker run --entrypoint`.

Utilizar esta instrucción no siempre es buena idea ya que el comando que nosotros pongamos se ejecutará como `"/bin/sh -c"` por ello que no va a poder recibir señales UNIX, lo que quiere decir que no podrá recibir la señal `SIGTERM` del `docker stop`.

Compilación del proyecto

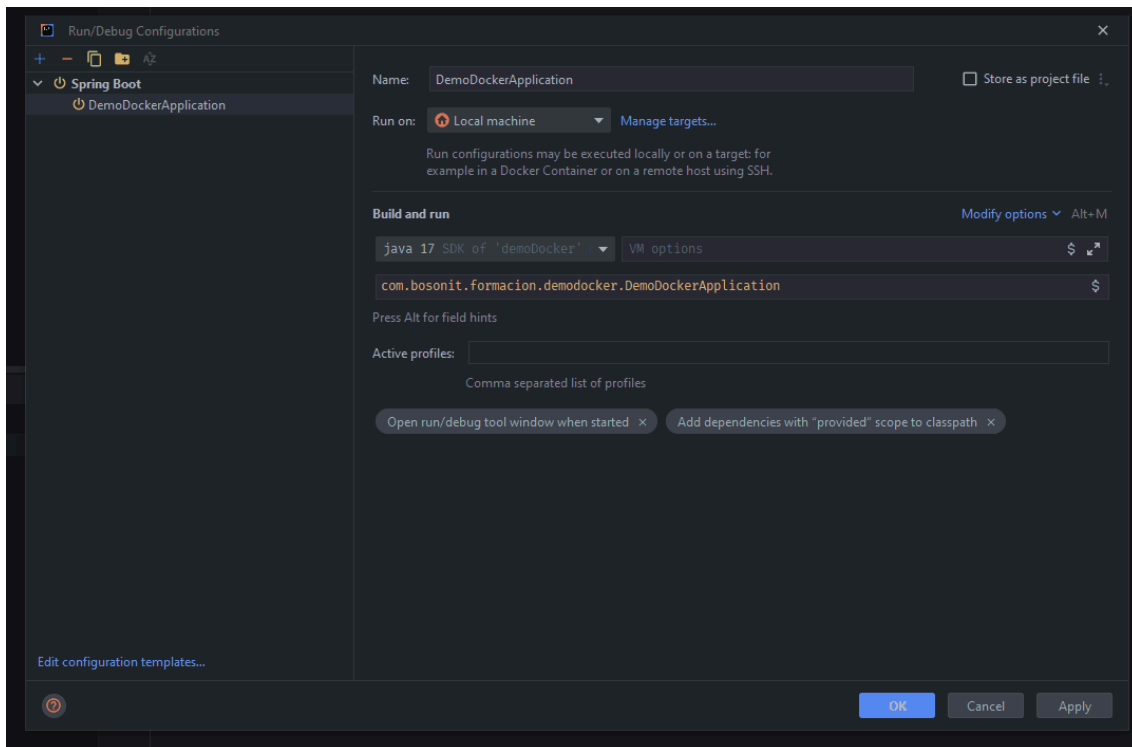
Ya tenemos nuestro `Dockerfile` creado por lo tanto es momento de construir nuestra imagen.

Lo primero que tenemos que hacer es construir el proyecto con Maven, para ello, en IntelliJ vamos a movernos al panel de configuración para añadir una nueva:



Una vez dentro del panel de configuraciones del proyecto vamos a crear una configuración nueva de maven con el siguiente comando:

```
clean install -DskipTests
```

Hemos utilizado la bandera `-D` para pasarle el parámetro `skipTests` de modo que Maven se salte los test unitarios.

Esto solamente se debe utilizar cuando construimos una aplicación que hemos testeado por nuestra cuenta, y que a la hora de ejecutar la compilación de Maven no puede acceder a los recursos necesarios como una BBDD o un fichero o cualquier otra cosa.

Ahora sí, ejecutamos la configuración que hemos creado para que Maven comience a compilar nuestro proyecto:



```
demoDocker - Dockerfile
FROM openjdk:11
EXPOSE 8081
COPY /target/*.jar /usr/local/lib/spring.jar
ENTRYPOINT ["java","-jar","/usr/local/lib/spring.jar"]
```

Si al final vemos algo como lo siguiente es que ya tenemos el JAR creado:

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11.809 s
[INFO] Finished at: 11-08-2022T10:44:51+02:00
[INFO] -----
```

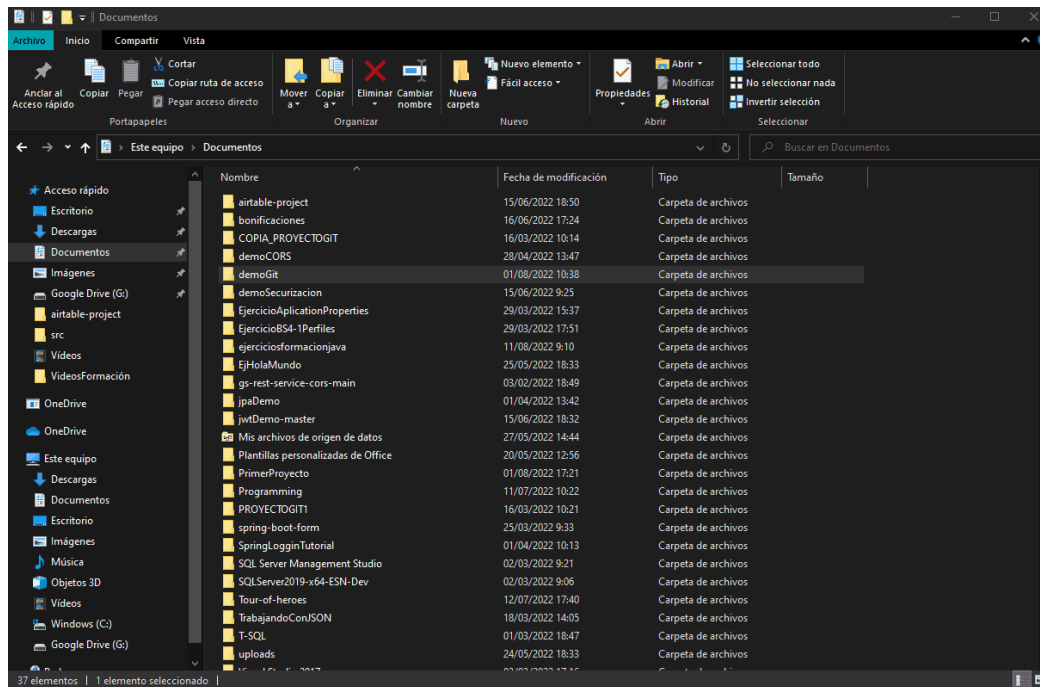
Una vez tenemos el proyecto compilado con el JAR creado estamos listos para construir nuestra imagen.

Construcción de nuestra imagen

Lo primero que tenemos que hacer es movernos a la ubicación de nuestro proyecto desde el cmd, en mi caso va a ser:

C:\\Users\\miusuario\\Documents\\ejerciciosformacionjava\\demoDocker.

Puedes navegar desde el explorador de archivos y una vez dentro escribir en la sección del PATH CMD para que te abra automáticamente una terminal en esa ruta.



Vale, ya tenemos la terminal abierta en la ruta de nuestro proyecto, ahora estamos a un comando de tener nuestra imagen creada.

El comando que tenemos que ejecutar para que se cree dicha imagen es el siguiente:

`docker build -t <name>:<tag> <ruta>`

- name: Hace referencia al nombre que le vamos a dar a la imagen que vamos a crear. (Tiene que estar en minúsculas)
- tag: Etiqueta que tenemos que poner a la imagen
- ruta: Dónde se encuentra el fichero Dockerfile que contiene las instrucciones para la creación de la imagen.



```
C:\Windows\System32\cmd.exe
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>
```

Y ya, tenemos la imagen creada:

```
C:\Windows\System32\cmd.exe
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
miapp                1              e00199ad1467   10 minutes ago  693MB
mysql               latest         7b94cda7ffc7   7 days ago     446MB
nginx               latest         b692a91e4e15   9 days ago     142MB
busybox             latest         7a80323521cc   12 days ago    1.24MB
tantrax007/metricasapi beta20         dbf1f10d11a0   7 weeks ago    512MB
postgresappimage    latest         933d847eb36c   3 months ago   706MB
postgres            latest         74b0c105737a   3 months ago   376MB
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>
```

¡Ya podemos crear contenedores con esa imagen!

Trabajando con contenedores

Hasta ahora hemos estado aprendiendo sobre la teoría detrás de Docker y hemos explorado cómo trabajar con las imágenes. Ahora es momento de adentrarnos en el mundo de los contenedores.

Vamos a trabajar con un ejemplo a lo largo de esta sección para conocer más comandos que podemos utilizar.

Crear un contenedor

Vamos a utilizar la imagen de MySQL. Una vez que tenemos nuestra imagen descargada, crearemos un contenedor con el comando

```
docker run <[imageName / imageId]:tag>
```



```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker run mysql:latest
2022-08-05 11:24:00+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.30-1.el8 started.
2022-08-05 11:24:00+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2022-08-05 11:24:00+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.30-1.el8 started.
2022-08-05 11:24:00+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified
You need to specify one of the following:
- MYSQL_ROOT_PASSWORD
- MYSQL_ALLOW_EMPTY_PASSWORD
- MYSQL_RANDOM_ROOT_PASSWORD
C:\Users\nahuel.capuzzi>
```

Pues sí, nos ha salido mal el run, esto es debido, como podemos intuir por el mensaje de error, a que no hemos digitado ciertas variables de entorno. Estas son para indicar datos adicionales, como por ejemplo, la contraseña de la cuenta Root o la contraseña del usuario... etc.

En este caso con digitar una es suficiente pero... ¿Cómo hacemos eso?

Variables de entorno

Generalmente, al trabajar con imágenes, necesitamos establecer valores específicos que se configuran al crear el contenedor. Estos valores se conocen como variables de entorno y se pasan utilizando la bandera -e.

En el error anterior, Docker ya nos está indicando que nos falta algo y nos ofrece la información:

```
You need to specify one of the following:
- MYSQL_ROOT_PASSWORD
- MYSQL_ALLOW_EMPTY_PASSWORD
- MYSQL_RANDOM_ROOT_PASSWORD
```

Vámonos a la documentación de este contenedor como ya sabemos.

Para ver qué variables de entorno podemos pasarle y cuáles de ellas son obligatorias tenemos que buscar la sección [Environment Variables](#) en la que encontraremos algo como lo siguiente:



Environment Variables

When you start the `mysql` image, you can adjust the configuration of the MySQL instance by passing one or more environment variables on the `docker run` command line. Do note that none of the variables below will have any effect if you start the container with a data directory that already contains a database: any pre-existing database will always be left untouched on container startup.

See also <https://dev.mysql.com/doc/refman/5.7/en/environment-variables.html> for documentation of environment variables which MySQL itself respects (especially variables like `MYSQL_HOST`, which is known to cause issues when used with this image).

`MYSQL_ROOT_PASSWORD`

This variable is mandatory and specifies the password that will be set for the MySQL `root` superuser account. In the above example, it was set to `my-secret-pw`.

`MYSQL_DATABASE`

This variable is optional and allows you to specify the name of a database to be created on image startup. If a user/password was supplied (see below) then that user will be granted superuser access (corresponding to `GRANT ALL`) to this database.

`MYSQL_USER` , `MYSQL_PASSWORD`

These variables are optional, used in conjunction to create a new user and to set that user's password. This user will be granted superuser permissions (see above) for the database specified by the `MYSQL_DATABASE` variable. Both variables are required for a user to be created.

Do note that there is no need to use this mechanism to create the root superuser, that user gets created by default with the password specified by the `MYSQL_ROOT_PASSWORD` variable.

`MYSQL_ALLOW_EMPTY_PASSWORD`

This is an optional variable. Set to a non-empty value, like `yes`, to allow the container to be started with a blank password for the root user. *NOTE:* Setting this variable to `yes` is not recommended unless you really know what you are doing, since this will leave your MySQL instance completely unprotected, allowing anyone to gain complete superuser access.

`MYSQL_RANDOM_ROOT_PASSWORD`

This is an optional variable. Set to a non-empty value, like `yes`, to generate a random initial password for the root user (using `pwgen`). The generated root password will be printed to stdout (`GENERATED ROOT PASSWORD:`).

`MYSQL_ONETIME_PASSWORD`

Sets root (not the user specified in `MYSQL_USER` !) user as expired once init is complete, forcing a password change on first login. Any non-empty value will activate this setting. *NOTE:* This feature is supported on MySQL 5.6+ only. Using this option on MySQL 5.5 will throw an appropriate error during initialization.

Si hemos leído esta sección sabremos que la variable **`MYSQL_ROOT_PASSWORD`** es obligatoria y esta es en la que vamos a especificar la contraseña que se utilizará en la cuenta Root de MySQL.

De modo que, ahora sí, tendríamos que crear nuestro contenedor de la siguiente forma:

```
Selección Smbolo del sistema
C:\Users\nahuel.capuzzi>docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=miContraseja -e MYSQL_DATABASE=miBdd -e MYSQL_USER=usuario -e MYSQL_PASSWORD=contrase;aUsuario mysql:1
atest
b3029a94df57df2569183611b0e75d01540feb00296f95b042f396682d93944b
C:\Users\nahuel.capuzzi>
```

En la captura anterior, a la hora de crear nuestro contenedor, le estamos pasando una serie de banderas para establecer unos parámetros:



- **-d**: Le indicamos a Docker que cree el contenedor en segundo plano
- **-p 3306:3306** : Mapeamos los puertos del contenedor, es decir, el puerto 3306 del contenedor se va a mapear al puerto 3306 de nuestra máquina local de modo que cuando lancemos una petición a nuestro localhost:3306 esta va a llegar al contenedor de docker que hemos levantado.
- Variables de entorno que le hemos especificado:
 - **MYSQL_ROOT_PASSWORD** Contraseña del usuario Root de MySQL.
 - **MYSQL_DATABASE** Nombre que le hemos dado a la BBDD que va a crear MySQL cuando arranque.
 - **MYSQL_USER** Usuario que estamos creando independiente a Root.
 - **MYSQL_PASSWORD** Contraseña del usuario independiente que hemos creado.

Una vez tenemos esto claro, si ejecutamos el comando veremos que Docker nos devuelve el identificador del contenedor creado:

```
Selecciónar Símbolo del sistema
C:\Users\nahuel.capuzzi>docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=miContraseja -e MYSQL_DATABASE=miBdd -e MYSQL_USER=usuario -e MYSQL_PASSWORD=contraseja usuario mysql:latest
b3029a94df57df2569183611b0e75d01540feb00296f95b042f396682d93944b
C:\Users\nahuel.capuzzi>
```

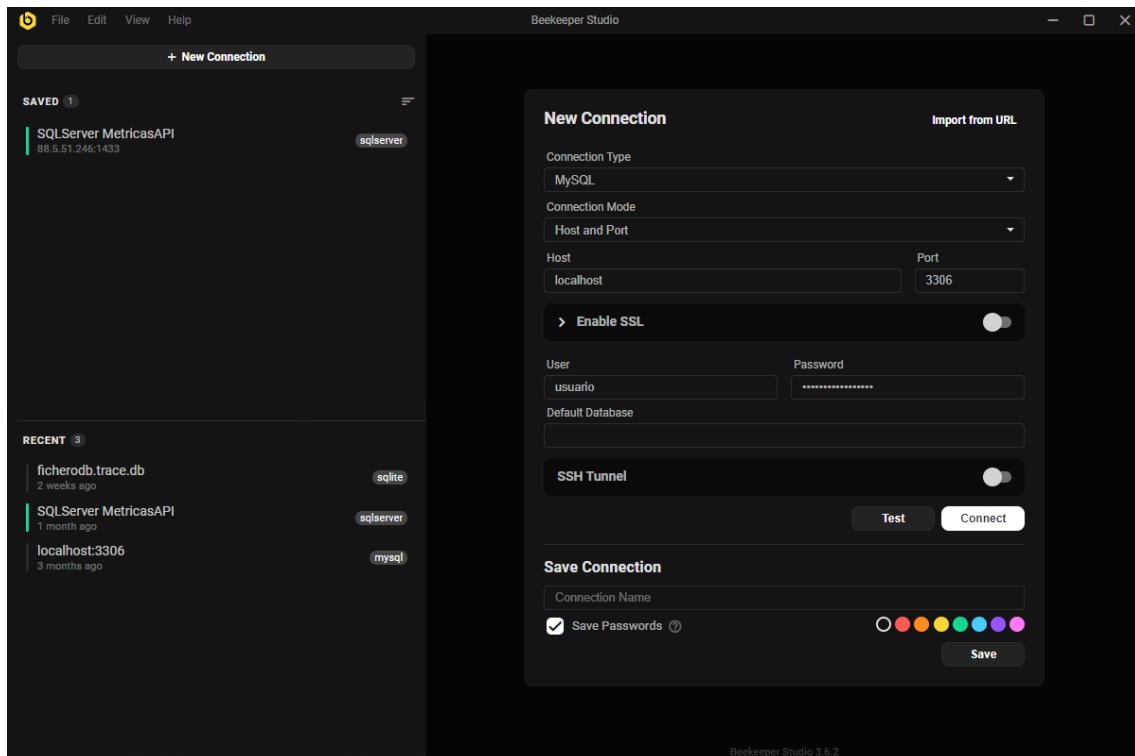
Ahora comprobemos que de verdad este contenedor se está ejecutando:

```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
b3029a94df57   mysql:latest "docker-entrypoint.s..." 7 minutes ago  Up 7 minutes  0.0.0.0:3306->3306/tcp, 33060/tcp  competent_khorana
C:\Users\nahuel.capuzzi>
```

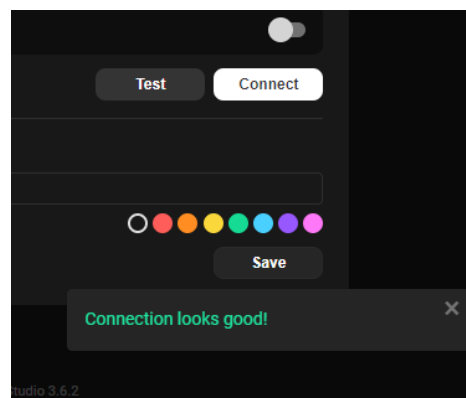
Pues sí, efectivamente tenemos el contenedor de docker funcionando correctamente.

Probando el contenedor

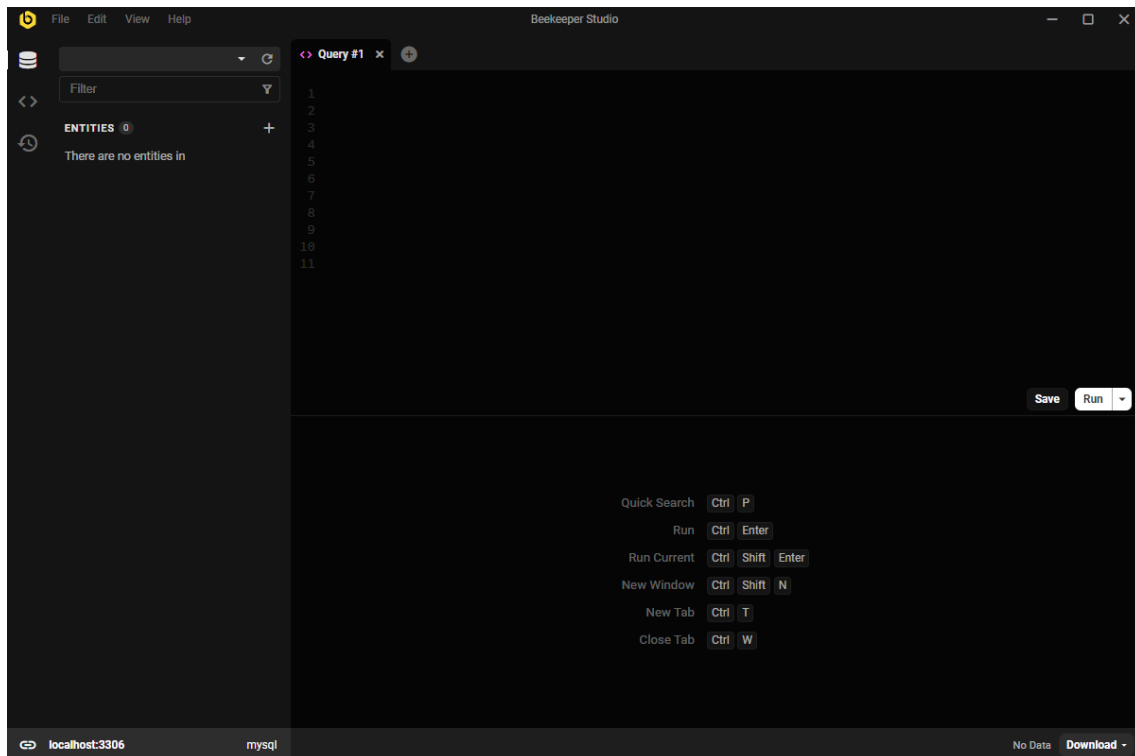
Como es una base de datos vamos a utilizar el cliente SQL [BeekeeperStudio](#) para conectarnos a ella y poder realizar alguna query de demostración:



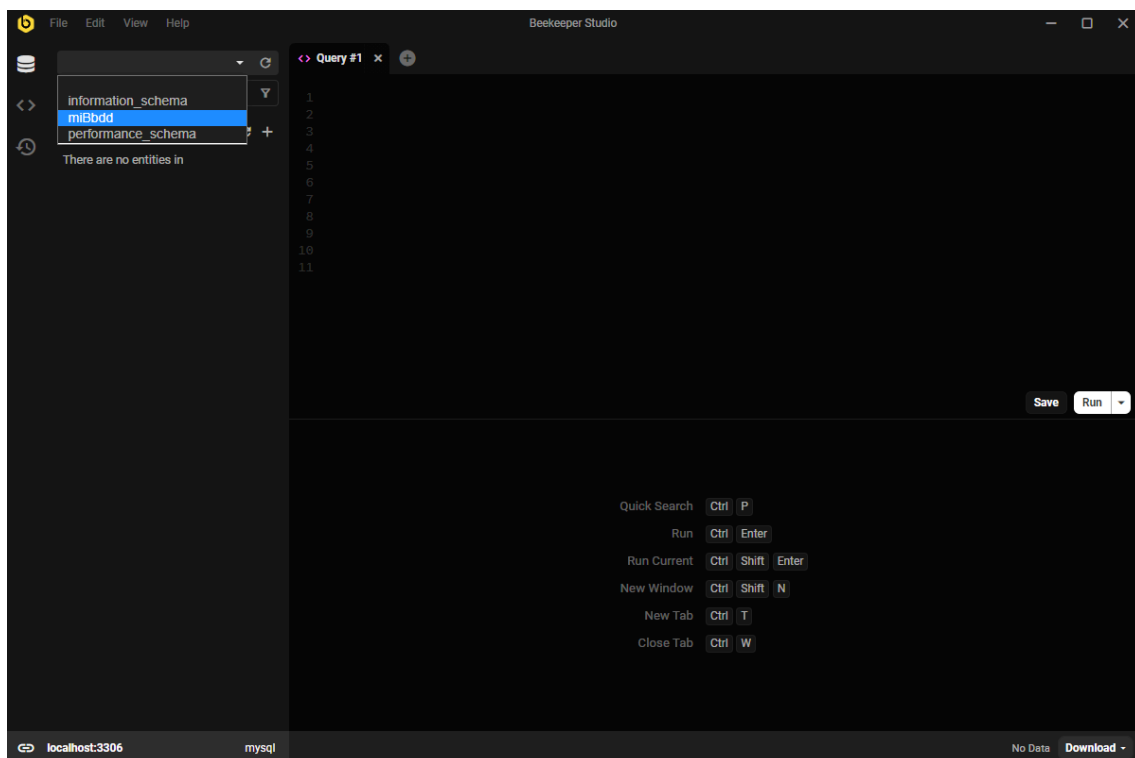
Para saber que la conexión se ha realizado con éxito podemos pulsar sobre el botón Test y deberíamos ver un mensaje como el siguiente:



Una vez nos conectemos veremos una interfaz como la siguiente:



Este es el editor, donde vamos a escribir las queries que vamos a lanzar a la base de datos. Para ello, tenemos que conectarnos a una de las instancias que tenemos. Estas se configuran en el desplegable que tenemos en la parte superior a la izquierda:





Si vemos, tenemos una que se llama *miBbdd*, que es la que pusimos con la variable de entorno **MYSQL_DATABASE** y es la base de datos que vamos a utilizar.

Sobre ella lanzaremos una query para crear una tabla sencilla sobre la que insertamos algún registro:

The screenshot shows the Beekeeper Studio interface. On the left, the 'ENTITIES' panel shows a database named 'miBbdd' with a table named 'Usuario'. The main editor displays a SQL query:

```
1 create table Usuario(  
2   id int(11) primary key not null auto_increment,  
3   nombre varchar(30) not null,  
4   edad int(3)  
5 )  
6  
7 insert into Usuario(nombre, edad) values('Alberto Gonzalez', 20);  
8 insert into Usuario(nombre, edad) values('Berta Garrido', 24);  
9 insert into Usuario(nombre, edad) values('Cristian Perez', 32);  
10 insert into Usuario(nombre, edad) values('Manuel Saenz', 34);  
11  
12 select * from Usuario;
```

 Below the query editor, the results are displayed in a table with columns 'id', 'nombre', and 'edad'. The table contains four rows of data. At the bottom, a status bar indicates 'localhost:3306', 'mysql', '4' rows, '0 affected', and '0.03 s'.

id	nombre	edad
1	Alberto Gonzalez	20
2	Berta Garrido	24
3	Cristian Perez	32
4	Manuel Saenz	34

Pues con este ejemplo ya vemos que nuestro contenedor está funcionando perfectamente.

Parar y arrancar Contenedores

Para terminar vamos a ver cómo parar el contenedor con el comando *docker stop <containerID / containerName>*, *docker pause <containerID containerName>* y cómo podemos volver a arrancarlo con *docker start <containerID / containerName>*.



```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
b3029a94df57   mysql:latest "docker-entrypoint.s..." 2 days ago Up 2 days 0.0.0.0:3306->3306/tcp, 33060/tcp competent_khorana

C:\Users\nahuel.capuzzi>docker stop b3029a
b3029a

C:\Users\nahuel.capuzzi>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES

C:\Users\nahuel.capuzzi>docker start b3029a
b3029a

C:\Users\nahuel.capuzzi>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
b3029a94df57   mysql:latest "docker-entrypoint.s..." 2 days ago Up 1 second 0.0.0.0:3306->3306/tcp, 33060/tcp competent_khorana

C:\Users\nahuel.capuzzi>
```

Si un contenedor se detiene, se liberan todos sus recursos asignados, mientras que un contenedor en pausa no libera memoria, pero sí la CPU. En ese caso, el proceso se suspende. Para ver diferencias podemos utilizar el comando *docker inspect*.

Información del contenedor

Como lo adelantamos anteriormente, si queremos ver los detalles de algún contenedor vamos a poder utilizar el comando *docker inspect <containerID / containerName>*.

Si lo aplicamos sobre el contenedor que creamos de MySQL veremos que nos imprime en la consola un JSON bastante largo con toda la información acerca de ese contenedor:

```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker inspect b302
[
  {
    "Id": "b3029a94df57df2569183611b0e75d01540fb00296f95b042f396682d93944b",
    "Created": "2022-08-05T11:58:09.95175152Z",
    "Path": "docker-entrypoint.sh",
    "Args": [
      "mysql"
    ],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 0,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2022-08-08T08:15:14.498379251Z",
      "FinishedAt": "2022-08-08T08:15:14.498379251Z"
    },
    "Image": "sha256:7b94cd27ffc7c59b03668e63f48b0f4ee3d10b427cc0b846193b65db671e9fa2",
    "ResolvConfPath": "/var/lib/docker/containers/b3029a94df57df2569183611b0e75d01540fb00296f95b042f396682d93944b/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/b3029a94df57df2569183611b0e75d01540fb00296f95b042f396682d93944b/hostname",
    "HostPath": "/var/lib/docker/containers/b3029a94df57df2569183611b0e75d01540fb00296f95b042f396682d93944b/hosts",
    "LogPath": "/var/lib/docker/containers/b3029a94df57df2569183611b0e75d01540fb00296f95b042f396682d93944b/hosts",
    "Name": "/competent_khorana",
    "RestartCount": 0,
    "Driver": "overlay2",
    "Platform": "linux",
    "MountLabel": "",
    "ProcessLabel": "",
    "AppArmorProfile": "",
    "ExecID": null,
    "HostConfig": {
      "Binds": null,
      "ContainerIDFile": "",
      "LogConfig": {
        "type": "json-file",
        "config": {}
      },
      "NetworkMode": "default",
      "PortBindings": {
        "3306/tcp": [
          {
            "HostIp": "",
            "HostPort": "3306"
          }
        ]
      },
      "RestartPolicy": {
        "Name": "no",
        "MaximumRetryCount": 0
      },
      "AutoRemove": false,
      "VolumeDriver": "",
      "VolumesFrom": null,
      "CapAdd": null,
      "CapDrop": null,
      "CgroupnsMode": "host",
      "Dns": [],
      "DnsOptions": [],
      "DnsSearch": [],
      "ExtraHosts": null,

```

Este comando nos puede ser muy útil en ciertos casos como el de comparar el uso de *docker stop* y *docker pause*.



Eliminando contenedores

Ya tenemos un contenedor creado, funcionando y sabemos como manejarlo mínimamente.

Veamos ahora cómo podríamos eliminar un contenedor que no necesitáramos o que, por el motivo que sea, ya no queremos tener. Para esto tenemos el comando `docker rm <containerID / containerName>`.

Supongamos que, sin querer, he creado un contenedor de MySQL de más y no me hace falta, por ello quiero eliminarlo así que lo primero que hago es listar todos los contenedores que tengo y, posteriormente, aplico el comando de eliminación:

```
Simbolo del sistema
C:\Users\nahuel.capuzzi>docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
f0e65690970c   mysql:latest   "docker-entrypoint.s..." 3 minutes ago   Exited (0) 6 seconds ago   5432/tcp                hardcore_euler
b3029a94d457   mysql:latest   "docker-entrypoint.s..." 2 days ago     Exited (0) 5 minutes ago   5432/tcp                competent_khorana
425cd43d982    postgres       "docker-entrypoint.s..." 3 months ago   Exited (255) 2 months ago   5432/tcp                postgres_test
ae0e9dd76c78    postgres       "docker-entrypoint.s..." 3 months ago   Exited (255) 2 months ago   0.0.0.0:5432->5432/tcp   Postgress

C:\Users\nahuel.capuzzi>docker rm f0e656
f0e656
C:\Users\nahuel.capuzzi>docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
b3029a94d457   mysql:latest   "docker-entrypoint.s..." 2 days ago     Exited (0) 6 minutes ago   5432/tcp                competent_khorana
425cd43d982    postgres       "docker-entrypoint.s..." 3 months ago   Exited (255) 2 months ago   5432/tcp                postgres_test
ae0e9dd76c78    postgres       "docker-entrypoint.s..." 3 months ago   Exited (255) 2 months ago   0.0.0.0:5432->5432/tcp   Postgress

C:\Users\nahuel.capuzzi>
```

¡Ojo! No vamos a poder eliminar contenedores que estén ejecutándose:

```
Simbolo del sistema
C:\Users\nahuel.capuzzi>docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
f0e65690970c   mysql:latest   "docker-entrypoint.s..." 34 seconds ago   Exited (0) 4 seconds ago   5432/tcp                hardcore_euler
b3029a94d457   mysql:latest   "docker-entrypoint.s..." 2 days ago     Exited (0) 2 minutes ago   5432/tcp                competent_khorana
425cd43d982    postgres       "docker-entrypoint.s..." 3 months ago   Exited (255) 2 months ago   5432/tcp                postgres_test
ae0e9dd76c78    postgres       "docker-entrypoint.s..." 3 months ago   Exited (255) 2 months ago   0.0.0.0:5432->5432/tcp   Postgress

C:\Users\nahuel.capuzzi>docker start f0e656
f0e656
C:\Users\nahuel.capuzzi>docker rm f0e656
Error response from daemon: You cannot remove a running container f0e65690970c49b3e789fe8370e6fae9bacf8934416c8ab18c829ae476d70b49. Stop the container before attempting removal or force remove
C:\Users\nahuel.capuzzi>
```

¿Matar un contenedor?

Otra opción que nos da Docker es la de matar un contenedor. Posiblemente te estés preguntando qué diferencia hay entre utilizar `docker stop <containerID / containerName>` y `docker kill <containerID / containerName>` pero no te preocupes porque aquí vamos a resolver esa duda pero antes vamos a explicar cómo funciona `docker kill`.

El `docker kill` termina con el proceso del contenedor de forma brusca pudiendo llegar a causar una parada no segura, por ejemplo, en el caso de que tengamos un contenedor corriendo que tenga montado un volumen de datos puede ser que si matamos el proceso en medio de una operación este quede dañado.

La estructura del comando es `docker kill <containerID / containerName>`.



La principal diferencia entre `docker stop` y `docker kill` radica en cómo detienen los contenedores y manejan sus procesos:

- `docker stop`:

Envía una señal `SIGTERM` al contenedor, permitiéndole tiempo para finalizar sus procesos correctamente.

Los procesos dentro del contenedor pueden manejar esta señal, realizar limpieza y terminar sus tareas antes de detenerse.

Es una forma más amigable de detener un contenedor, ya que permite que los procesos internos tengan un cierre ordenado.

- `docker kill`:

Envía una señal `SIGKILL` al contenedor, lo que fuerza la terminación inmediata de todos los procesos del contenedor.

Esta señal no puede ser manejada ni ignorada por los procesos internos del contenedor.

Puede resultar en la terminación abrupta de procesos, lo que podría dejar recursos no liberados o datos corruptos.

En resumen, `docker stop` es una forma más suave de detener un contenedor, mientras que `docker kill` es más drástico y puede resultar en la finalización abrupta de procesos, lo que podría tener consecuencias no deseadas, como dejar procesos hijo sin control.

AutoEliminación

Otra cosa más que Docker nos permite realizar es, a la hora de crear nuestro contenedor indicarle que este se elimine en el momento en el que lo paremos. Para ello vamos a utilizar la bandera `-rm`.

En la captura anterior queda reflejado, creamos un contenedor con la bandera `-rm` y, a la hora de parar ese contenedor este se elimina automáticamente.

¿Cuándo será útil? dejamos abierta la respuesta sin contestar para que cada uno piense en qué momento puede ser útil.



Redes con Docker

Una de las características sorprendentes de Docker es su capacidad para interconectar contenedores entre sí, aislarlos, conectarlos con otros, y crear mapeos hacia el host para establecer conexiones. Pero su utilidad no se limita solo a eso; las redes de Docker también pueden conectar contenedores con cargas de trabajo que no están en Docker.

Los contenedores y servicios Docker ni siquiera necesitan saber que están desplegados en Docker, o si sus compañeros son también cargas de trabajo Docker o no. No entraremos en detalles sobre aspectos del sistema operativo como Iptables en Linux o Firewall en Windows.

Docker ofrece por defecto hasta 7 formas distintas de configurar conexiones, pero antes de explorarlas, es importante entender algunos conceptos fundamentales.

Creando Redes

Para crear redes en docker vamos a utilizar el comando `docker network create <nombreRed>`.

Si el comando se ejecuta exitosamente nos devolverá el identificador de la red que se ha creado:

```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker network create miRed
d9fc5ee5095b1b9889a5ad7080e9642b5ae904124190bf8c8072b9b609ec75d3
C:\Users\nahuel.capuzzi>
```

Podremos crear tantas redes como necesitemos pero que no tengan el mismo nombre:

```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker network create miRed
Error response from daemon: network with name miRed already exists
C:\Users\nahuel.capuzzi>
```

La red por defecto se crea de tipo bridge pero si lo necesitamos podemos especificarle que sea de el tipo que queramos con la bandera `-d <networkType>`

Visualizar Redes

Una vez hemos creado nuestra primera red vamos a ver cómo podemos visualizar todas las redes con las que contamos. Para ello vamos a utilizar el comando `docker network ls`:



```
Selecionar Símbolo del sistema
C:\Users\nahuel.capuzzi>docker network create miRed
Error response from daemon: network with name miRed already exists

C:\Users\nahuel.capuzzi>docker network ls
NETWORK ID        NAME        DRIVER        SCOPE
a74791cf19e2     bridge     bridge        local
b3ccfd2b8176     host       host          local
d9fc5ee5095b     miRed      bridge        local
150000901bb7     none      null          local

C:\Users\nahuel.capuzzi>
```

Este comando nos devuelve la siguiente tabla con todas las redes que tenemos en nuestro equipo junto con los siguientes datos:

- NETWORK ID: Identificador de la red
- NAME: Nombre de la red
- DRIVER: Tipo de red que estamos utilizando
- SCOPE: Alcance de la red

Eliminar Redes

Por último vamos a ver cómo podemos eliminar las redes que no vayamos a seguir utilizando. El comando que en este caso podemos utilizar es *docker network rm <imageName / imageID>*.

```
Símbolo del sistema
C:\Users\nahuel.capuzzi>docker network create miRed
Error response from daemon: network with name miRed already exists

C:\Users\nahuel.capuzzi>docker network ls
NETWORK ID        NAME        DRIVER        SCOPE
a74791cf19e2     bridge     bridge        local
b3ccfd2b8176     host       host          local
d9fc5ee5095b     miRed      bridge        local
150000901bb7     none      null          local

C:\Users\nahuel.capuzzi>docker network rm miRed
miRed

C:\Users\nahuel.capuzzi>docker network ls
NETWORK ID        NAME        DRIVER        SCOPE
a74791cf19e2     bridge     bridge        local
b3ccfd2b8176     host       host          local
150000901bb7     none      null          local

C:\Users\nahuel.capuzzi>
```

Inspeccionar Redes

También podemos inspeccionar las redes de Docker de forma que podamos ver información acerca de ellas como los contenedores que contiene.

Para realizar esta inspección posiblemente ya te estés imaginando el comando que vamos a utilizar y efectivamente es *docker network inspect <networkName / networkID>*.



```
C:\Users\nahuel.capuzzi>docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "a74791cf19e2bda0509274c6020c20243abfdb9f664d50dc8206630d4841760a",
    "Created": "2022-08-09T11:41:48.743248885Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "efcce4024e42735d1b22f258f5e9ffed5bc0bc82f378f74a3d84cf50eea59595": {
        "Name": "cliente",
        "EndpointID": "e235833f086f66b381e121265f97547b286031ad0790f8ddc0ff96b8f6a14303",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
      },
      "ff40f7ef203b0582cbc0fe9c2aa5f91e7c54af5dd5c5d6d223bbf2105075ade": {
        "Name": "server",
        "EndpointID": "50a906843221ec975047e32b9f54dcf37278509fb5fc5d6d492581c718d08357",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]
```

En la captura anterior tenemos lo que nos devuelve el comando cuando lo aplicamos sobre una red. En este caso se puede apreciar que en la red bridge tengo dos contenedores conectados.

Los contenedores aparecen porque están arrancados y corriendo, sino, por mucho que los hubiéramos configurado con esta red si están parados no nos van a salir en el docker network inspect.

Docker compose

Al trabajar con contenedores, hemos experimentado cómo para cada uno de ellos debemos pasar parámetros, configurar la red, entre otras tareas. Manejar múltiples contenedores de esta manera puede convertirse en un dolor de cabeza. Afortunadamente, Docker ofrece una forma mucho más sencilla y organizada de abordar esta tarea.



Docker Compose es un archivo de configuración de contenedores que nos permite crear las configuraciones necesarias para dichos contenedores de manera ordenada. Este archivo se escribe utilizando el formato YAML.

Paso a paso

Exactamente en la misma ruta en la que creamos el fichero Dockerfile es donde vamos a poder crear nuestro fichero *docker-compose.yml*.

Vamos a ver cómo configurar el fichero *docker-compose.yml* con dos contenedores:

```
version: "3.2" #Version de Docker compose que estamos utilizando
services: #Servicios, es decir, los contenedores que vamos a crear
  mispringboot: #Nombre del contenedor
    build: . #Indicamos que este contenedor se va a construir a traves del Dockerfile
    ports: #Vamos a indicar una lista de los puertos que vamos a mapear para el
contenedor
    - "8080:8080" #PuertoDelHost:PuertoDelContenedor
    links: #Lista con los contenedores a los que el contenedor se va a conectar
    - server #Este es el nombre del contenedor
  server: #Nombre del contenedor
    image: mysql #Imagen desde la cual vamos a construir el contenedor
    ports: #Lista de puertos que vamos a mapear del contenedor
    - "3306:3306" #PuertoDelHost:PuertoDelContenedor
    environment: #Variables de entorno que hay que configurar en la maquina
      MYSQL_USER: miUsuario
      MYSQL_PASSWORD: passwordUsuario
      MYSQL_DATABASE: miBaseDeDatos
      MYSQL_ROOT_PASSWORD: password
```

```
1  version: "3.2" #Version de Docker compose que estamos utilizando
2  services: #Servicios, es decir, los contenedores que vamos a crear
3    mispringboot: #Nombre del contenedor
4      build: . #Indicamos que este contenedor se va a construir a traves del Dockerfile
5      ports: #Vamos a indicar una lista de los puertos que vamos a mapear para el contenedor
6        - "8080:8080" #PuertoDelHost:PuertoDelContenedor
7      links: #Lista con los contenedores a los que el contenedor se va a conectar
8        - server #Este es el nombre del contenedor
9    server: #Nombre del contenedor
10     image: mysql #Imagen desde la cual vamos a construir el contenedor
11     ports: #Lista de puertos que vamos a mapear del contenedor
12       - "3306:3306" #PuertoDelHost:PuertoDelContenedor
13     environment: #Variables de entorno que hay que configurar en la maquina
14       MYSQL_USER: miUsuario
15       MYSQL_PASSWORD: passwordUsuario
16       MYSQL_DATABASE: miBaseDeDatos
17       MYSQL_ROOT_PASSWORD: password
18
```




Incluso, si miramos las imágenes que tenemos de Docker debería de haber una nueva ya que, si recordamos, para crear el contenedor *mispringboot* tiene que construir la imagen a partir del dockerfile:

```
C:\Windows\System32\cmd.exe
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
demodocker_mispringboot latest             5ce48eacdb3e       3 hours ago        510MB
mysql                latest             7b94cda7ffc7       7 days ago         446MB
nginx                latest             b692a91e4e15       9 days ago         142MB
busybox              latest             7a80323521cc       12 days ago        1.24MB
tantrax007/metricasapi beta20             dbf1f10d11a0       7 weeks ago        512MB
postgresappimage     latest             933d847eb36c       3 months ago       706MB
postgres             latest             74b0c105737a       3 months ago       376MB
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>
```

```
C:\Windows\System32\cmd.exe
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
demodocker_mispringboot latest             5ce48eacdb3e       3 hours ago        510MB
mysql                latest             7b94cda7ffc7       7 days ago         446MB
nginx                latest             b692a91e4e15       9 days ago         142MB
busybox              latest             7a80323521cc       12 days ago        1.24MB
tantrax007/metricasapi beta20             dbf1f10d11a0       7 weeks ago        512MB
postgresappimage     latest             933d847eb36c       3 months ago       706MB
postgres             latest             74b0c105737a       3 months ago       376MB
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>
```

Y correcto, tenemos la imagen *demodocker_mispringboot*.

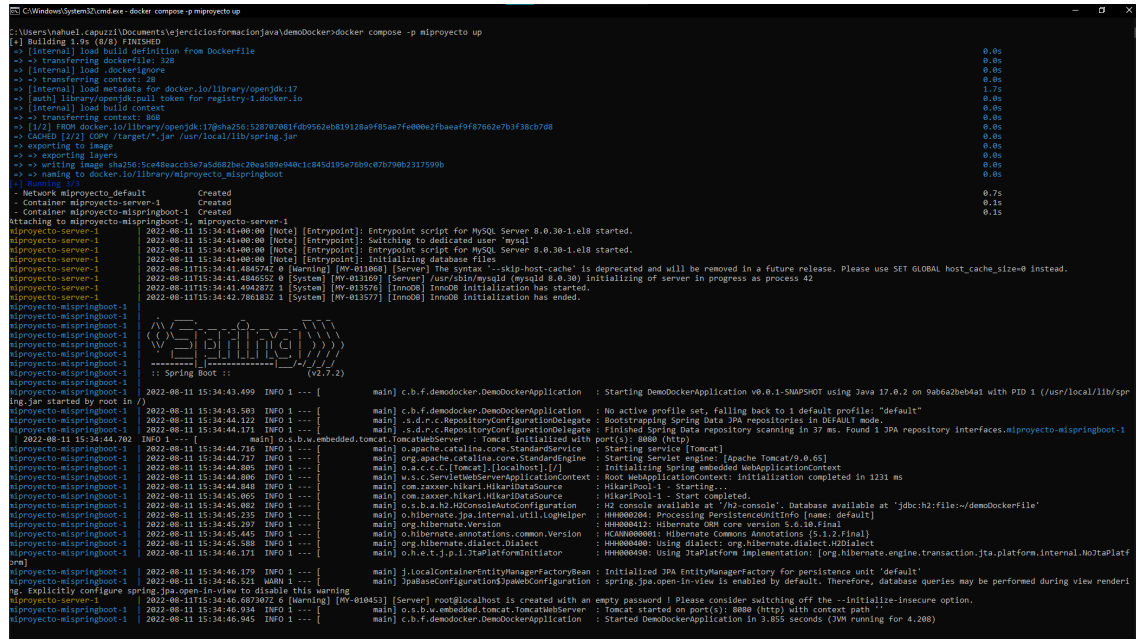
Para detener la ejecución de Docker compose tenemos que pulsar la combinación de teclas *Ctrl + C*. Esto detendrá de una forma correcta y segura los contenedores que ha creado. Si quisiéramos forzar el parado tendríamos que pulsar esa combinación de teclas dos veces. Si queremos volver a levantar el Docker compose este volverá a utilizar los contenedores que ya ha creado.

Ahora bien, posiblemente nos hayamos dado cuenta de que tanto la imagen como los contenedores no se están nombrando como nosotros lo hemos indicado, esto se debe a que realmente cuando nosotros pusimos esto en el fichero *docker-compose.yml*:

```
version: "3.2" #Version de Docker compose que estamos utilizando
services: #Servicios, es decir, los contenedores que vamos a crear
  mispringboot: #Nombre del contenedor <-- ¡FALSO!
```

No estábamos realmente estableciendo el nombre al contenedor, en parte sí pero en parte no ya que, acorde con la documentación oficial este está cogiendo el nombre del proyecto como prefijo del nombre para los contenedores y las imágenes.

Esto lo podemos cambiar si lo necesitamos, para ello, como también se especifica en la documentación oficial, podemos utilizar la bandera *-p* para establecer un nombre de proyecto propio:



```
C:\Windows\System32\cmd.exe

C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker ps

CONTAINER ID   IMAGE                                COMMAND                  STATUS          PORTS
9ab6a2be641    miproyecto_mispringboot            "java -jar /usr/loca..." About a minute ago Up About a minute    0.0.0.0:8080->8080/tcp, 8081/tcp
7c31b3b4634a    mysql                               "docker-entrypoint.s..." About a minute ago Up About a minute    0.0.0.0:3306->3306/tcp, 33060/tcp
miproyecto-mispringboot-1
miproyecto-server-1

C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker images

REPOSITORY    TAG       IMAGE ID      CREATED       SIZE
miproyecto_mispringboot    latest   3ce48eacbc3e   3 hours ago   510MB
mysql                latest   7b9dcd7ffc7f   7 days ago    446MB
nginx                latest   b692a91e4e15   9 days ago    142MB
busybox              latest   7a80323521cc   12 days ago    1.24MB
tantrax007/metricasapi    beta20   dbf1f10d11a0   7 weeks ago    512MB
postgresappimage         latest   933d847eb36c   3 months ago   786MB
postgres              latest   740c105737a    3 months ago   376MB
```

Eliminar el Docker compose

Posiblemente estés pensando en ir a identificar cada uno de los contenedores y eliminarlos con `docker rm`. Posteriormente tendríamos que ir también a cargarnos también las imágenes que este utilizó con `docker rmi`.

Esto muy seguramente nos llevará tiempo y trabajo pero no te preocupes, los desarrolladores de Docker compose pensaron en esto y crearon un comando que nos va a realizar el trabajo por nosotros. El comando es *docker compose down*.

Vamos a ver dos casos en los que podemos utilizar este comando:

- Ejemplo 1:

Levantamos *docker compose up*:

[illegible]

Nos crea los siguientes contenedores y redes:

```
- Network demodocker_default      Created
- Container demodocker-server-1   Created
- Container demodocker-mispringboot-1 Created
```

Ahora vamos a pararlo:

```
- Container demodocker-mispringboot-1 Stopped
- Container demodocker-server-1 Stopped
canceled
```

Veamos cómo tenemos los contenedores creados y parados:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d92903e7133d	demodocker_mispringboot	"java -jar /usr/local..."	About a minute ago	Exited (143) 37 seconds ago		demodocker-mispringboot-1
d9d207d29d06	mysql	"docker-entrypoint.s..."	About a minute ago	Exited (0) 35 seconds ago		demodocker-server-1

Además también tenemos la imagen creada:

```
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE	
demodocker	mispringboot	latest	5ce48eaccb3e	4 hours ago	510MB

Y por último también tenemos la red:

```
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
a1d0636fa171	bridge	bridge	local
7ebca7bb25a1	demodocker_default	bridge	local

Si utilizamos el comando `docker compose down` teóricamente, como os hemos comentado hace nada, Docker debería de eliminar los contenedores, la imagen y la red que ha creado así que vamos a ejecutarlo:

```
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker compose down
[+] Running 3/3
 - Container demodocker-mispringboot-1 Removed
 - Container demodocker-server-1 Removed
 - Network demodocker_default Removed

C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>
```

Como vemos, el comando nos ha eliminado los dos containers y la red pero... ¿Qué ha pasado con nuestra imagen? Pues muy sencillo, no se ha eliminado. Para que este comando nos elimine las imágenes vamos a tener que agregarle la bandera `--rm`.

```
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker compose down --rmi all
[+] Running 5/5
 - Container demodocker-mispringboot-1   Removed
 - Container demodocker-server-1         Removed
 - Image mysql                           Removed
 - Image demodocker_mispringboot         Removed
 - Network demodocker_default            Removed

C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>
```

Como vemos, ahora sí que se ha eliminado absolutamente todo lo que se utilizó por el docker compose up.

- Ejemplo 2:

Vamos a levantar otro `docker compose -p miProyecto up` pero ahora con la bandera `-p` para especificar un nombre de proyecto propio:

```

C:\Users\mahuel.capuzzi\Documents\ejercicioformacionjava\demodocker>docker compose -p proyectopropio up
[+] Running 17/17
   server Pulled
   3ac22f3a638d Pull complete
   3ac22f3a638d Pull complete
   51e727fde85e Pull complete
   20ba5a8c3ab Pull complete
   410a229693ff Pull complete
   1c73fca980b Pull complete
   c93c21a5f85b Pull complete
   67523c498c7 Pull complete
   d7f6cfe1efcb Pull complete
   916f32c0394 Pull complete
   6dd5a50a4f Pull complete
[+] Building 1.0s (7/7) FINISHED
[+] Internal: load build definition from Dockerfile
[+] > transferring Dockerfile: 32B
[+] Internal: load .dockerignore
[+] > transferring context: 2B
[+] Internal: load metadata for docker.io/library/openjdk:17
[+] Internal: load build context
[+] > transferring context: 80B
[+] [1/2] FROM docker.io/library/openjdk:17@sha256:528707801fd0952a08b112d9f95ae7f000a2f0aeaf9f87062e7b3f38c7d0
[+] CACHED [2/2] COPY /src/main/resources/jar /usr/local/lib/jar
[+] exporting layers
[+] > exporting image sha256:5c6c8a4c3e7a5d5d2bec20a59e948c1c845d195670b9c87070b02317599b
[+] > naming to docker.io/library/proyectopropio_mispringboot
[+] Running 6/6
   0. Network proyectopropio_default Created
   0. Container proyectopropio-server-1 Created
   0. Container proyectopropio-mispringboot-1 Created
Attaching to proyectopropio-mispringboot-1, proyectopropio-server-1
proyectopropio-server-1 2022-08-11 16:38:30:000 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.30-1.el8 started.
proyectopropio-server-1 2022-08-11 16:38:30:000 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
proyectopropio-server-1 2022-08-11 16:38:30:000 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.30-1.el8 started.
proyectopropio-server-1 2022-08-11 16:38:30:000 [Note] [Entrypoint]: Initializing database files
proyectopropio-server-1 2022-08-11 17:16:38:30:4496232 [Warning] [NW-01808] [Server] The syntax '--skip-host-cache' is deprecated and will be removed in a future release. Please use SET GLOBAL host_cache_size= instead.
proyectopropio-server-1 2022-08-11 17:16:38:30:4496804 [System] [NW-013169] [Server] /usr/sbin/mysqld (mysqld 8.0.30) initializing of server in progress as process 42
proyectopropio-server-1 2022-08-11 17:16:38:30:4508372 [Info] [NW-013570] [InnoDB] InnoDB initialization has started.
proyectopropio-server-1 2022-08-11 17:16:38:30:4508372 [System] [NW-013571] [InnoDB] InnoDB initialization has ended.
proyectopropio-mispringboot-1 2022-08-11 16:38:30:353 INFO 1 --- [main] c.b.f.demodocker.DemoDockerApplication : Starting DemoDockerApplication v0.0.1-SNAPSHOT using Java 17.0.2 on 2071cf6c565 with PID 1 (/usr/local/lib
proyectopropio-mispringboot-1 2022-08-11 16:38:30:356 INFO 1 --- [main] c.b.f.demodocker.DemoDockerApplication : No active profile set, falling back to 1 default profile: 'default'

```




Lo primero que está haciendo Docker es volver a descargarse la imagen de MySQL ya que, en el ejemplo anterior, dejamos todo limpio y sin nada.

Incluso, si observamos con atención veremos el proceso que sigue el contenedor de MySQL para establecer la base de datos propia y el usuario que digitamos en el fichero `docker-compose.yml`:

```
projectopropio-server-1 2022-08-11 16:38:59+00:00 [Note] [Entrypoint]: Creating database miBaseDeDatos
projectopropio-server-1 2022-08-11 16:38:59+00:00 [Note] [Entrypoint]: Creating user miUsuario
projectopropio-server-1 2022-08-11 16:38:59+00:00 [Note] [Entrypoint]: Giving user miUsuario access to schema miBaseDeDatos
```

Bien, hasta aquí lo de siempre, ya hemos ejecutado el Docker compose y ha levantado todo lo que necesitamos para trabajar. Lo siguiente que vamos a hacer es parar el proceso como ya sabemos y lo vamos a limpiar todo.

Si utilizamos el comando `docker compose down --rmi all` para limpiar todo lo que levantamos no vamos a conseguir nada:

Lo único que está haciendo Docker es intentar eliminar la imagen de MySQL pero no nos está mencionando por ningún lado ni los contenedores, ni la imagen de Spring Boot ni siquiera la red que ha creado. Más aún, nos está diciendo que no puede eliminar la imagen de MySQL porque hay un contenedor utilizándola ¿Qué está pasando?

Pues bien, como sabemos, al ejecutar el Docker compose anterior le indicamos con la bandera `-p` que el nombre del proyecto que queríamos utilizar era `projectopropio` en vez del padre (proyecto donde se ubica el fichero `docker-compose.yml`) `demoDocker` pues esto es lo que nos falta por especificarle a la hora de eliminarlo. Por defecto Docker compose sabemos que pilla el nombre del proyecto en el que se encuentra ubicado, en este caso `demoDocker` pero, al nosotros especificarle uno diferente, lógicamente no lo encuentra, por lo que se ciñe en ir a por lo único que sí conoce y ni siquiera puede eliminarla porque está ocupada.

Para poder eliminar todo lo que creamos en ese momento vamos a tener que ejecutar el comando `docker compose -p projectopropio down --rmi all` que básicamente tiene la bandera `-p` en la que especificamos el nombre del proyecto que queremos limpiar:

```
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>docker compose -p projectopropio down --rmi all
[+] Running 5/5
 - Container projectopropio-mispringboot-1 Removed
 - Container projectopropio-server-1       Removed
 - Image mysql                             Removed
 - Image projectopropio_mispringboot       Removed
 - Network projectopropio_default          Removed
C:\Users\nahuel.capuzzi\Documents\ejerciciosformacionjava\demoDocker>
```



¡Y sí! A funcionado perfectamente, hemos conseguido limpiar todo lo que se refiere al proyecto proyectopropio.

Troubleshooting

Si obtenemos el error `Class has been compiled by a more recent version of the Java Environment` se puede deber muy probablemente a que en el equipo tenemos configurado un JRE (Java Runtime Environment) diferente al que se utilizó por Docker para la compilación de la imagen.

¿Cómo podemos saber qué JRE tenemos en el equipo?

Tanto en Linux como en Windows tenemos el comando `java -XshowSettings:properties -version` que podemos utilizar para ver las propiedades de nuestro sistema. Una de ellas será el JRE y el JDK.

¿Cómo lo puedo solucionar?

Lo que tenemos que hacer es eliminar la imagen que hemos instalado y compilarla de nuevo pero con una versión compatible a la aplicación que hemos programado.

En la siguiente tabla se puede ver la clase a la que corresponde cada versión de Java:

Class	Java Version
49	5
50	6
51	7
52	8
53	9
54	10
55	11
56	12
57	13
58	14



59	15
60	16
61	17
62	18
63	19

Resumen comandos Docker

[docker-cheat-sheet](#)