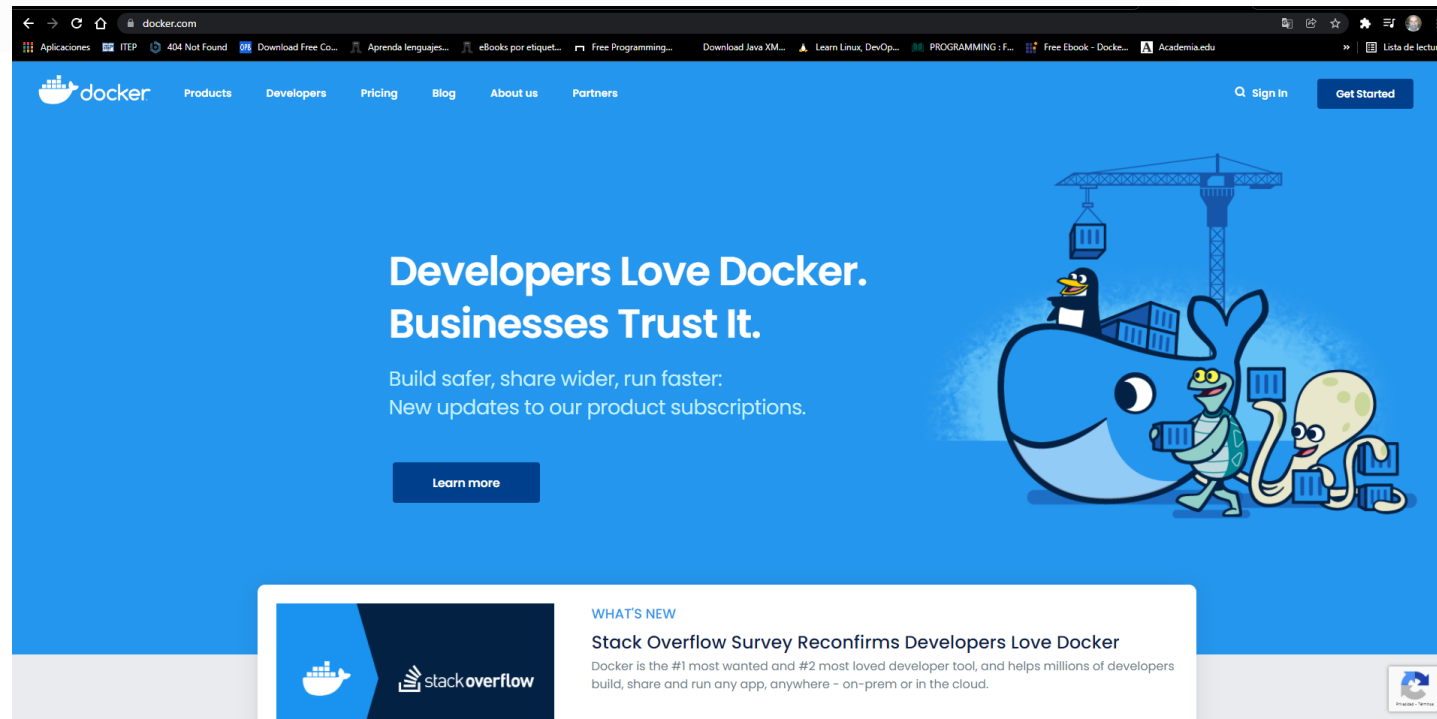


docker®

Docker: descripción general

Docker es un servicio de gestión de contenedores. Las palabras clave de Docker se **desarrollan, envían y ejecutan en** cualquier lugar. La idea general de Docker es que los desarrolladores desarrollen aplicaciones fácilmente, las envíen a contenedores que luego puedan implementarse en cualquier lugar.

El lanzamiento inicial de Docker fue en marzo de 2013 y, desde entonces, se ha convertido en la palabra de moda para el desarrollo del mundo moderno, especialmente frente a los proyectos basados en Agile.



Características de Docker



- Puede implementar contenedores Docker en cualquier lugar, en cualquier máquina física y virtual e incluso en la nube.
- Dado que los contenedores Docker son bastante livianos, son muy fácilmente escalables.

- Docker tiene la capacidad de reducir el tamaño del desarrollo al proporcionar una huella más pequeña del sistema operativo a través de contenedores.
- Con los contenedores, es más fácil para los equipos de diferentes unidades, como desarrollo, control de calidad y operaciones, trabajar sin problemas en todas las aplicaciones.

Componentes de Docker



- **Docker for Mac** : permite ejecutar contenedores Docker en Mac OS.

- **Docker for Linux** : permite ejecutar contenedores Docker en el sistema operativo Linux.

- **Docker for Windows** : permite ejecutar contenedores Docker en el sistema operativo Windows.

- **Docker Engine** : se utiliza para crear imágenes de Docker y crear contenedores de Docker.

- **Docker Hub** : este es el registro que se utiliza para alojar varias imágenes de Docker.

- **Docker Compose** : se utiliza para definir aplicaciones que utilizan varios contenedores Docker.

Instalación de Docker

Desktop

Platform	x86_64 / amd64	arm64 (Apple Silicon)
Docker Desktop for Mac (macOS)	x	x
Docker Desktop for Windows	x	

Instalación de Docker

Server

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	s390x
CentOS	x	x		
Debian	x	x	x	
Fedora	x	x		
Raspbian			x	
RHEL				x
SLES				x
Ubuntu	x	x	x	x
Binaries	x	x	x	

Docker - Hub

Docker Hub es un servicio de registro en la nube que le permite descargar imágenes de Docker creadas por otras comunidades. También puede cargar sus propias imágenes creadas por Docker en Docker Hub.

The screenshot shows the Docker Hub website with the search bar at the top. The search results for 'gradle' are displayed, showing the 'gradle' image with 150 verified content and 61942 community members. A search dropdown menu is open, showing suggestions like 'neo4j', 'gra die', 'arangodb', 'orientdb', 'Mostrar las 150 visitas en Contenido verificado', 'Comunidad (61942)', 'gra ylog / gra ylog', 'gra phiteapp / gra phite-statsd', 'gra die / build-cache-node', 'gra ylog2 / servidor', and 'Mostrar todas las 61942 visitas en Comunidad'. A box on the right contains the command 'docker pull gradle' and a link to 'Ver etiquetas disponibles'. The bottom section of the page shows a 'Referencia rápida' (Quick Reference) section with information about the project's maintenance and support, and a section for 'Etiquetas admitidas y Dockerfile enlaces respectivos' (Accepted tags and respective Dockerfile links).

hub.docker.com/gradle

Explorar > Imágenes

Contenido verificado (150)

Comunidad (61942)

Mostrar las 150 visitas en Contenido verificado

Mostrar todas las 61942 visitas en Comunidad

Copiar y pegar para extraer esta imagen

```
docker pull gradle
```

Ver etiquetas disponibles

Referencia rápida

- **Mantenido por:** Keegan Witt (del Proyecto Groovy) , con la aprobación del Proyecto Gradle
- **Dónde obtener ayuda:** los foros de estibador de la Comunidad , la holgura del estibador Comunidad , o desbordamiento de pila

Etiquetas admitidas y Dockerfile enlaces respectivos

- 7.3.1-jdk8 , 7.3-jdk8 , 7-jdk8 , jdk8

flag needs an argument: 'v' in -v
See 'docker run --help'.

C:\Users\codig\Downloads>docker run -p 8080:8080 -p 50000:50000 jenkins
docker: error during connect: This error may indicate that the docker daemon is not running.: Post "http://%2F%2F.%2Fpipe%2Fdocker_engine/v1.24/containers/create": open //./pipe/docker_engine: El sistema no puede encontrar el archivo especificado.
See 'docker run --help'.

C:\Users\codig\Downloads>docker pull jenkins
Using default tag: latest
error during connect: This error may indicate that the docker daemon is not running.: Post "http://%2F%2F.%2Fpipe%2Fdocker_engine/v1.24/images/create?fromImage=jenkins&tag=latest": open //./pipe/docker_engine: El sistema no puede encontrar el archivo especificado.

C:\Users\codig\Downloads>docker pull jenkins
Using default tag: latest
Error response from daemon: manifest for jenkins:latest not found: manifest unknown: manifest unknown

C:\Users\codig\Downloads>docker pull gradle
Using default tag: latest
latest: Pulling from library/gradle
7b1a6ab2e44d: Pull complete
8329695590e8: Pull complete
9bd6da4468db: Downloading [=====>] 161.4MB/192.9MB
8e07f21656cb: Download complete
ca055f63c612: Download complete
8327f35ed409: Verifying Checksum
9fb8c764d49c: Downloading [=====>] 107.3MB/115.8MB

Ejecute esto desde el directorio del proyecto de Gradle que desea compilar.

docker run --rm -u gradle -v "\$PWD":/home/gradle/project -w /home/gradle/project gradle gradle
<gradle-task>

Docker - Imágenes

Una imagen es una combinación de un sistema de archivos y parámetros

`docker run hello-world`

- El comando Docker es específico y le dice al programa Docker en el sistema operativo que se debe hacer algo.
- El comando de **run** se usa para mencionar que queremos crear una instancia de una imagen, que luego se llama **contenedor**.
- Finalmente, "hello-world" representa la imagen a partir de la cual está hecho el contenedor.

Docker - Imágenes

podemos usar la imagen de CentOS disponible en Docker Hub para ejecutar CentOS en nuestra máquina Ubuntu.

```
sudo docker run -it centos /bin/bash
```

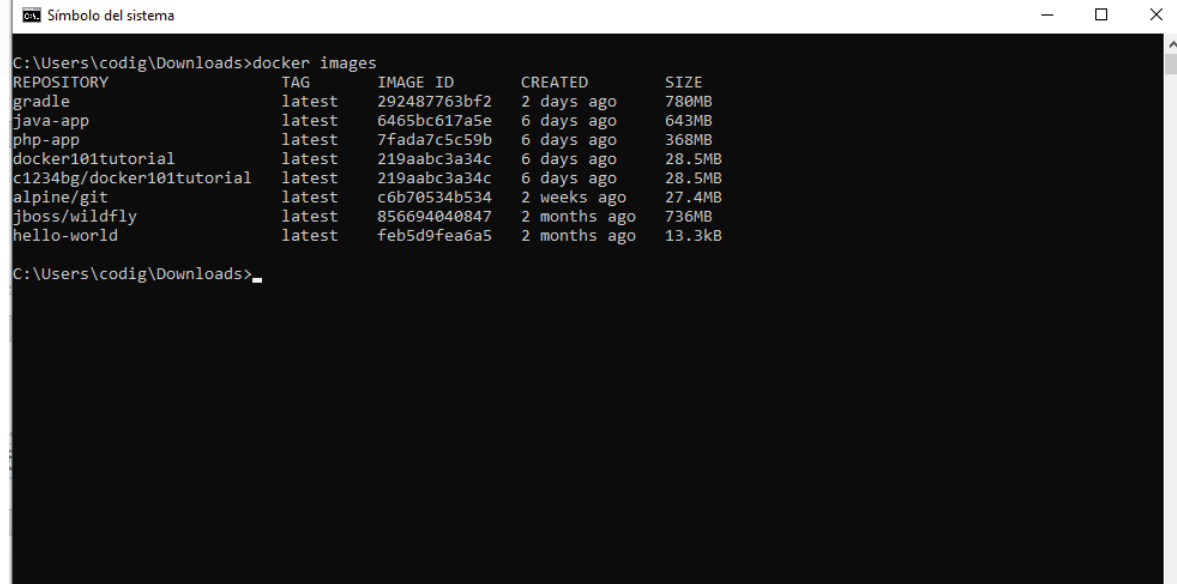
- Estamos usando el comando **sudo** para asegurarnos de que se ejecute con acceso de **root**.
- Aquí, **centos** es el nombre de la imagen que queremos descargar de Docker Hub e instalar en nuestra máquina Ubuntu.
- **-it** usa para mencionar que queremos ejecutar en **modo interactivo**.
- **/ bin / bash** se usa para ejecutar el shell bash una vez que CentOS está en funcionamiento.

Visualización de imágenes de Docker

Para ver la lista de imágenes de Docker en el sistema, puede ejecutar el siguiente comando.

`docker images`

Este comando se utiliza para mostrar todas las imágenes instaladas actualmente en el sistema.



```
Símbolo del sistema
C:\Users\codig\Downloads>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
gradle               latest             292487763bf2       2 days ago         780MB
java-app             latest             6465bc617a5e       6 days ago         643MB
php-app              latest             7fada7c5c59b       6 days ago         368MB
docker101tutorial    latest             219aabc3a34c       6 days ago         28.5MB
c1234bg/docker101tutorial latest             219aabc3a34c       6 days ago         28.5MB
alpine/git           latest             c6b70534b534       2 weeks ago        27.4MB
jboss/wildfly        latest             856694040847       2 months ago       736MB
hello-world          latest             feb5d9fea6a5       2 months ago       13.3kB
C:\Users\codig\Downloads>
```

Cada imagen tiene los siguientes atributos:

• **TAG** : se utiliza para etiquetar imágenes de forma lógica.

• **Image ID** : se utiliza para identificar de forma única la imagen.

• **Created** : el número de días desde que se creó la imagen.

• **SIZE (Tamaño virtual)** : el tamaño de la imagen.

Ejemplo

`sudo docker run centos`

si ejecutamos el comando Docker **images** para ver la lista de imágenes en el sistema, también deberíamos poder ver la imagen **centos** .

```
Símbolo del sistema

C:\Users\codig\Downloads>docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
gradle               latest          292487763bf2   2 days ago     780MB
java-app             latest          6465bc617a5e   6 days ago     643MB
php-app              latest          7fada7c5c59b   6 days ago     368MB
docker101tutorial    latest          219aabc3a34c   6 days ago     28.5MB
c1234bg/docker101tutorial latest          219aabc3a34c   6 days ago     28.5MB
alpine/git           latest          c6b70534b534   2 weeks ago    27.4MB
jboss/wildfly         latest          856694040847   2 months ago   736MB
hello-world           latest          feb5d9fea6a5   2 months ago   13.3kB

C:\Users\codig\Downloads>docker run centos
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest

C:\Users\codig\Downloads>_
```

```
Símbolo del sistema

Microsoft Windows [Versión 10.0.19043.1348]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\codig>docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
gradle               latest          292487763bf2   2 days ago     780MB
java-app             latest          6465bc617a5e   6 days ago     643MB
php-app              latest          7fada7c5c59b   6 days ago     368MB
c1234bg/docker101tutorial latest          219aabc3a34c   6 days ago     28.5MB
docker101tutorial    latest          219aabc3a34c   6 days ago     28.5MB
alpine/git           latest          c6b70534b534   2 weeks ago    27.4MB
jboss/wildfly         latest          856694040847   2 months ago   736MB
hello-world           latest          feb5d9fea6a5   2 months ago   13.3kB
centos                latest          5d0da3dc9764   2 months ago   231MB

C:\Users\codig>
```

Eliminar imágenes de Docker

`docker rmi ImageID`

• **ImageID** : este es el ID de la imagen que debe eliminarse.

Ejemplo

`sudo docker rmi 7a86f8ffcb25`

Aquí, **7a86f8ffcb25** es el ID de imagen de la imagen **newcentos**

```
demo@ubuntuserver:~$ sudo docker rmi 7a86f8ffcb25
Untagged: newcentos:latest
Deleted: 7a86f8ffcb258e42c11d971a04b1145151b80122e566bc2b544f8fc3f94caf1e
demo@ubuntuserver:~$
```

Comandos

mágenes de docker -q

devolver solo los ID de imagen

```
sudo docker images -q
```

Docker inspect

se usa para ver los detalles de una imagen o contenedor

```
docker inspect Repository
```

• **Repository** : este es el nombre de la imagen.

Listado de contenedores

`docker ps`

Se pueden enumerar todos los contenedores en la máquina a través del comando **docker ps**. Este comando se usa para devolver los contenedores que se están ejecutando actualmente.

`docker ps -a`

se usa para listar todos los contenedores en el sistema

`docker history ImageID`

puede ver todos los comandos que se ejecutaron con una imagen a través de un contenedor.

- **ImageID** : este es el ID de imagen para el que desea ver todos los comandos que se ejecutaron en él.

Docker: trabajar con contenedores

`docker kill ContainerID`

se usa para matar los procesos en un contenedor en ejecución.

- **ContainerID** : este es el ID de contenedor al que debe matar los procesos en el contenedor.

`docker top ContainerID`

puede ver los procesos principales dentro de un contenedor

- **ContainerID** : este es el ID de contenedor para el que desea ver los procesos principales.

`docker unpause ContainerID`

se utiliza para **reanudar** los procesos en un contenedor en ejecución.

- **ContainerID** : este es el ID de contenedor en el que debe reanudar los procesos en el contenedor.

`docker stop ContainerID`

se usa para detener un contenedor en ejecución.

- **ContainerID** : este es el ID de contenedor que debe detenerse.

`docker pause ContainerID`

se usa para pausar los procesos en un contenedor en ejecución.

- **ContainerID** : este es el ID de contenedor en el que debe pausar los procesos en el contenedor.

`docker rm ContainerID`

se usa para eliminar un contenedor.

- **ContainerID** : este es el ID de contenedor que debe eliminarse.

`docker attach ContainerID`

se usa para adjuntar a un contenedor en ejecución.

- **ContainerID** : este es el ID de contenedor al que debe adjuntar.

`docker stats ContainerID`

se utiliza para proporcionar las estadísticas de un contenedor en ejecución.

- **ContainerID** : este es el ID de contenedor para el que se deben proporcionar las estadísticas.

Docker: ciclo de vida del contenedor



```
graph TD; A[Docker: ciclo de vida del contenedor] --- B[1- Docker: ciclo de vida del contenedor]; A --- C[2- Luego, el contenedor de Docker entra en estado de ejecución cuando se usa el comando de run de Docker .]; A --- D[3- El comando kill de docker se usa para eliminar un contenedor de Docker existente.]; A --- E[4- El comando de pause de Docker se usa para pausar un contenedor de Docker existente.]; A --- F[5- El comando de stop de Docker se usa para pausar un contenedor de Docker existente.]; A --- G[6- El comando de run de Docker se utiliza para devolver un contenedor de un estado detenido a un estado de ejecución .];
```

1- Docker: ciclo de vida del contenedor

2- Luego, el contenedor de Docker entra en estado de ejecución cuando se usa el comando de run de Docker .

3- El comando **kill de docker** se usa para eliminar un contenedor de Docker existente.

4- El comando de **pause de Docker** se usa para pausar un contenedor de Docker existente.

5- El comando de stop de Docker se usa para pausar un contenedor de Docker existente.

6- El comando de run de Docker se utiliza para devolver un contenedor de un estado **detenido** a un estado de **ejecución** .

Docker - Arquitectura



- El servidor es el servidor físico que se utiliza para alojar varias máquinas virtuales.

- El sistema operativo host es la máquina base, como Linux o Windows.

- El hipervisor es VMWare o Windows Hyper V que se utiliza para alojar máquinas virtuales.

- Luego, instalaría varios sistemas operativos como máquinas virtuales sobre el hipervisor existente como SO invitado.

- Luego, alojaría sus aplicaciones en la parte superior de cada sistema operativo invitado.

virtualización habilitada a través de Dockers.

- 1- El servidor es el servidor físico que se utiliza para alojar varias máquinas virtuales. Entonces esta capa sigue siendo la misma.

- 2- El sistema operativo host es la máquina base, como Linux o Windows. Entonces esta capa sigue siendo la misma.

- 3- Ahora llega la nueva generación que es el motor Docker. Esto se usa para ejecutar el sistema operativo que anteriormente solía ser máquinas virtuales como contenedores Docker.

- 4- Todas las aplicaciones ahora se ejecutan como contenedores Docker.

Docker: configuración

```
graph LR; A[Docker: configuración] --- B[service docker stop]; A --- C[service docker start]; B --- D[se utiliza para detener el proceso del demonio de Docker.]; C --- E[se utiliza para iniciar el proceso del demonio de Docker.];
```

`service docker stop`

se utiliza para detener el proceso del **demonio de Docker** .

`service docker start`

se utiliza para iniciar el proceso del demonio de Docker.

Docker - Containers and Shells

```
sudo docker run -it centos /bin/bash
```

Usamos este comando para crear un nuevo contenedor y luego usamos el comando **Ctrl + P + Q** para salir del contenedor. Asegura que el contenedor aún exista incluso después de que salgamos del contenedor.

Podemos verificar que el contenedor todavía existe con el comando **ps de Docker**. Si tuviéramos que salir del contenedor directamente, entonces el contenedor mismo se destruiría.

Ahora hay una manera más fácil de sujetar los contenedores y sacarlos limpiamente sin necesidad de destruirlos. Una forma de lograrlo es mediante el comando **nsenter**.

Antes de ejecutar el comando **nsenter**, primero debe instalar la imagen **nsenter**. Se puede hacer usando el siguiente comando:

```
docker run --rm -v /usr/local/bin:/target jpetazzo/nsenter
```

Antes de usar el comando **nsenter**, necesitamos obtener el ID de proceso del contenedor, porque el comando **nsenter** lo requiere. Podemos obtener el ID de proceso a través del comando de **inspect** de Docker y filtrarlo a través del **Pid**.

Como se ve en la captura de pantalla anterior, primero usamos el comando **docker ps** para ver los contenedores en ejecución. Podemos ver que hay un contenedor en ejecución con el ID de **ef42a4c5e663**.

Luego usamos el comando Docker **inspect** para inspeccionar la configuración de este contenedor y luego usamos el comando **grep** para filtrar simplemente el ID del proceso. Y en la salida, podemos ver que el ID de proceso es **2978**.

```
root@ubuntudemo:~# sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED
STATUS        PORTS     NAMES
ef42a4c5e663   centos:latest  "/bin/bash"             2 minutes ago
Up 2 minutes   stoic_banach

root@ubuntudemo:~# sudo docker inspect ef42a4c5e663 | grep Pid
    "PidMode": "",
    "Pid": 2978,
root@ubuntudemo:~# _
```

nsenter

Este método permite que uno se adhiera a un contenedor sin salir del contenedor.

`nsenter -m -u -n -p -i -t comando containerID`

Opciones

- **-u** se usa para mencionar el **espacio de nombres Uts**
- **-m** se usa para mencionar el **espacio de nombres de montaje**
- **-n** se usa para mencionar el **espacio de nombres de la red**
- **-p** se usa para mencionar el **espacio de nombres del proceso**
- **-i** s para que el contenedor se ejecute en modo interactivo.
- **-t** se utiliza para conectar los flujos de E / S del contenedor al sistema operativo host.
- **containerID** : este es el ID del contenedor.
- **Comando** : este es el comando que se ejecuta dentro del contenedor.

Ejemplo

```
sudo nsenter -m -u -n -p -i -t 2978 /bin/bash
```

Producción

```
root@ubuntudemo:~# sudo nsenter -n -u -n -p -i -t 2978 /bin/bash
bash-4.2# exit
exit
root@ubuntudemo:~# sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED
STATUS        PORTS     NAMES
ef42a4c5e663   centos:latest  "/bin/bash"             9 minutes ago
Up 9 minutes
root@ubuntudemo:~# _
```

A partir de la salida, podemos observar los siguientes puntos:

- El indicador cambia al **shell bash** directamente cuando **emitimos el comando nsenter**.
- Luego emitimos el comando de **exit**. Ahora, normalmente, si no usó el comando **nsenter**, el contenedor se destruiría. Pero notará que cuando ejecutamos el comando **nsenter**, el contenedor todavía está en funcionamiento.

Docker File

Docker también le brinda la capacidad de crear sus propias imágenes de Docker, y puede hacerlo con la ayuda de **Docker Files** . Un archivo Docker es un archivo de texto simple con instrucciones sobre cómo crear sus imágenes.

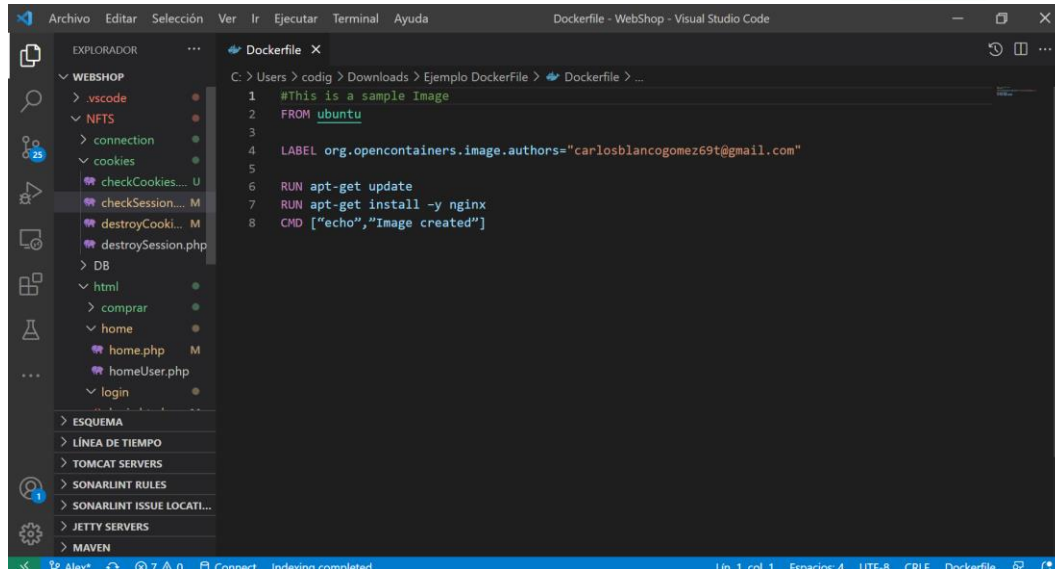
Paso 1 : cree un archivo llamado **Docker File** y **éditelo** usando un editor de texto . Tenga en cuenta que el nombre del archivo debe ser "Dockerfile" con "D" como mayúscula.

Paso 2 : cree su archivo Docker siguiendo las siguientes instrucciones.

Los siguientes puntos deben tenerse en cuenta sobre el archivo anterior:

- La primera línea "# Esta es una imagen de muestra" es un comentario. Puede agregar comentarios al archivo Docker con la ayuda del comando # La primera línea "# Esta es una imagen de muestra" es un comentario. Puede agregar comentarios al archivo Docker con la ayuda del comando #
- La siguiente línea debe comenzar con la palabra clave **FROM** . Le dice a Docker, desde qué imagen base desea basar su imagen. En nuestro ejemplo, estamos creando una imagen a partir de la imagen de **ubuntu** .
- El siguiente comando es la persona que va a mantener esta imagen. Aquí especifica la palabra clave **LABEL** y solo menciona el ID de correo electrónico.
- El comando **RUN** se utiliza para ejecutar instrucciones en la imagen. En nuestro caso, primero actualizamos nuestro sistema Ubuntu y luego instalamos el servidor nginx en nuestra imagen de **ubuntu** .
- El último comando se utiliza para mostrar un mensaje al usuario.

Paso 3 : guarde el archivo.



```
1 #This is a sample Image
2 FROM ubuntu
3
4 LABEL org.opencontainers.image.authors='carlosblancogomez69t@gmail.com'
5
6 RUN apt-get update
7 RUN apt-get install -y nginx
8 CMD ['echo','Image created']
```

Docker creación de archivos

Ahora es el momento de crear el archivo Docker. El archivo Docker se puede compilar con el siguiente comando:

```
docker build
```

Producción

En el resultado, primero verá que la imagen de Ubuntu se descargará desde Docker Hub, porque no hay una imagen disponible localmente en la máquina.

compilación de docker

Este método permite a los usuarios crear sus propias imágenes de Docker.

```
docker build -t ImageName:TagName dir
```

- **-t** - es mencionar una etiqueta a la imagen

- **ImageName** : este es el nombre que desea darle a su imagen.

- **TagName** : esta es la etiqueta que desea asignar a su imagen.

- **Dir** : el directorio donde está presente el archivo Docker.

Ejemplo

```
sudo docker build -t myimage:0.1.
```

Aquí, **myimage** es el nombre que le estamos dando a la imagen y **0.1** es el número de etiqueta que le estamos dando a nuestra imagen.

Dado que el archivo Docker está en el directorio de trabajo actual, usamos "." al final del comando para indicar el directorio de trabajo actual.

```
Símbolo del sistema
C:\Users\codig\Downloads\Ejemplo DockerFile>docker build -t myimage:0.1 .
[+] Building 30.7s (7/7) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 247B                                              0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                    0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest                 3.7s
=> [auth] library/ubuntu:pull token for registry-1.docker.io                    0.0s
=> [1/3] FROM docker.io/library/ubuntu@sha256:626ffa58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322  7.9s
=> => resolve docker.io/library/ubuntu@sha256:626ffa58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322  0.0s
=> => sha256:626ffa58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322  1.42kB / 1.42kB  0.0s
=> => sha256:7cc8576c7c8ec2384de5cbf245f41567e922aab1b075f3e8ad565f508832df17  529B / 529B  0.0s
=> => sha256:ba6accdd2923aee4c2acc6a23780b14ed4b8a5fa4e14e252a23b846df9b6c1  1.46kB / 1.46kB  0.0s
=> => sha256:7b1a6ab2e44dbac178598dabe7cfff59bd67233dba0b27e4fbd1f9d4b3c877a54  28.57MB / 28.57MB  5.5s
=> => extracting sha256:7b1a6ab2e44dbac178598dabe7cfff59bd67233dba0b27e4fbd1f9d4b3c877a54  2.1s
=> [2/3] RUN apt-get update                                                       17.4s
=> ERROR [3/3] RUN apt-get install -y nginx                                     1.5s
-----
> [3/3] RUN apt-get install -y nginx:
#7 0.533 Reading package lists...
#7 1.298 Building dependency tree...
#7 1.470 Reading state information...
#7 1.499 E: Unable to locate package -y
-----
executor failed running [/bin/sh -c apt-get install -y nginx]: exit code: 100
C:\Users\codig\Downloads\Ejemplo DockerFile>
```

```
Símbolo del sistema
C:\Users\codig\Downloads\Ejemplo DockerFile>docker images
REPOSITORY          TAG         IMAGE ID      CREATED      SIZE
c1234bg/docker101tutorial  latest     fdd836a0f463  12 hours ago  28.5MB
docker101tutorial      latest     fdd836a0f463  12 hours ago  28.5MB
alpine/git             latest     c6b70534b534  2 weeks ago   27.4MB
jpetazzo/nsenter       latest     427d3ddc6f9d  18 months ago 377MB

C:\Users\codig\Downloads\Ejemplo DockerFile>
```

Docker repositorio público

Los repositorios públicos se pueden usar para alojar imágenes de Docker que todos los demás pueden usar. Un ejemplo son las imágenes que están disponibles en Docker Hub. La mayoría de las imágenes como Centos, Ubuntu y Jenkins están disponibles públicamente para todos. También podemos hacer que nuestras imágenes estén disponibles publicándolas en el repositorio público de Docker Hub.

Para nuestro ejemplo, usaremos el repositorio **docker101tutorial** integrado en el capítulo "Creación de archivos de Docker" y cargaremos esa imagen en Docker Hub. Primero revisemos las imágenes en nuestro host de Docker para ver qué podemos enviar al registro de Docker.

```
Simbolo del sistema
C:\Users\codig\Downloads\Ejemplo DockerFile>docker images
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE
c1234bg/docker101tutorial  latest      fdd836a0f463  12 hours ago  28.5MB
docker101tutorial      latest      fdd836a0f463  12 hours ago  28.5MB
alpine/git            latest      c6b70534b534  2 weeks ago   27.4MB
jpetazzo/nsenter      latest      427d3ddc6f9d  18 months ago 377MB

C:\Users\codig\Downloads\Ejemplo DockerFile>
```

Los siguientes pasos explican cómo puede cargar una imagen en un repositorio público.

Paso 1 : inicie sesión en Docker Hub y cree su repositorio. Este es el repositorio donde se almacenará su imagen. Vaya a <https://hub.docker.com/> e inicie sesión con sus credenciales.

Paso 2 - Haga clic en el botón "Crear repositorio" en la pantalla de arriba y cree un repositorio con el nombre **demorep**. Asegúrese de que la visibilidad del repositorio sea pública.

Una vez que se crea el repositorio, tome nota del comando **pull** que se adjunta al repositorio.

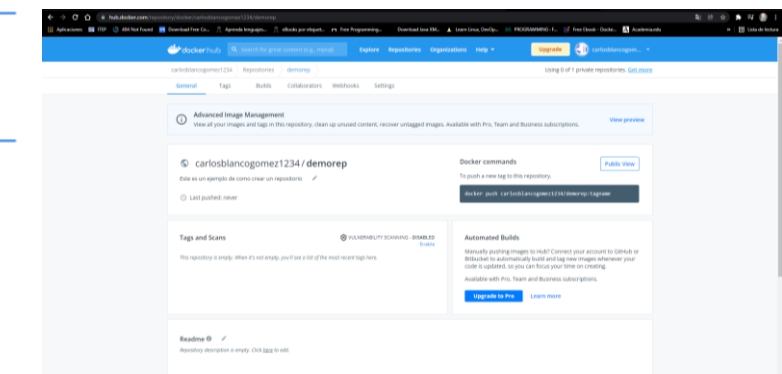
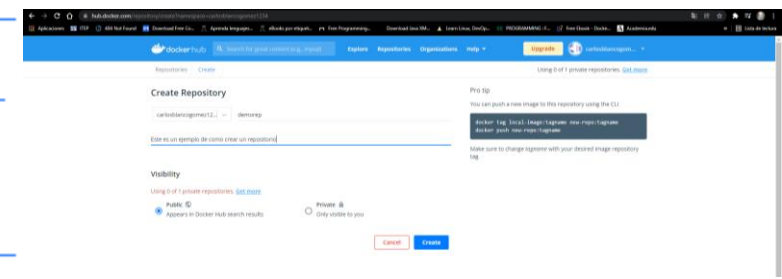
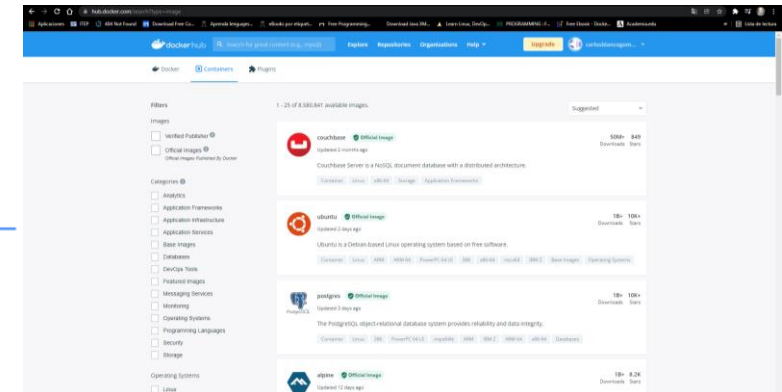
El comando **Pull** que se utilizará en nuestro repositorio es el siguiente:

`docker pull demours/demorep`

Paso 3 : ahora vuelve al host de Docker. Aquí debemos etiquetar nuestra **myimage** en el nuevo repositorio creado en Docker Hub. Podemos hacer esto a través del **comando de etiqueta de Docker**.

Paso 4 : emita el comando de inicio de sesión de Docker para iniciar sesión en el repositorio de Docker Hub desde el símbolo del sistema. El comando de inicio de sesión de Docker le solicitará el nombre de usuario y la contraseña del repositorio de Docker Hub.

Paso 5 : una vez que se haya etiquetado la imagen, ahora es el momento de enviar la imagen al repositorio de Docker Hub. Podemos hacer esto a través del comando **push de Docker**



Docker tag

Este método permite etiquetar una imagen en el repositorio correspondiente.

`docker tag imageID Repositoryname`

- **imageID** : este es el ImageID que debe etiquetarse en el repositorio.
- **Repositoryname** : este es el nombre del repositorio al que se debe etiquetar ImageID.

Ejemplo

`sudo docker tag ab0c1d3744dd demour/demorep:1.0`

A continuación se proporciona un resultado de muestra del ejemplo anterior.

```
demo@ubuntu:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
myimage             0.1                ab0c1d3744dd       6 minutes ago
centos               latest             67591570dd29       2 days ago
ubuntu              latest             104bec311bcd       2 days ago
demo@ubuntu:~$ sudo docker tag ab0c1d3744dd demour/demorep:1.0
demo@ubuntu:~$
```

Docker push

Este método permite enviar imágenes al Docker Hub.

`docker push Repositoryname`

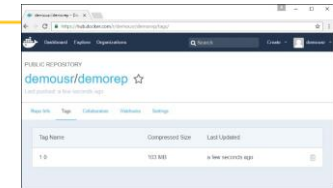
- **RepositoryName** - Este es el nombre del repositorio que necesita ser empujado hasta el Hub del estibador.

Ejemplo

`sudo docker push demour/demorep:1.0`

```
demo@ubuntu:~$ sudo docker push demour/demorep:1.0
The push refers to a repository [docker.io/demour/demorep]
3f3a3db4de69: Layer already exists
ef84b80e23cc: Layer already exists
5972e5e5b524: Layer already exists
3451558844eb: Layer already exists
b6e6ef522791: Layer already exists
37f743c24123: Pushed
32475bc97c41: Layer already exists
1.0: digest: sha256:1bcdac3a9270a95798f02cd287b91956c5a6cf9fae0d02eb3d11f3a22d40d42
3442 size: 1701
demo@ubuntu:~$
```

Si vuelve a la página de Docker Hub y va a su repositorio, verá el nombre de la etiqueta en el repositorio.



Ahora intentemos extraer el repositorio que cargamos en nuestro host Docker. Primero **eliminemos** las imágenes, **myimage: 0.1** y **demour / demorep: 1.0**, del host local de Docker. Usemos el comando de **extracción de Docker** para extraer el repositorio de Docker Hub.

```
demo@ubuntu:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
centos               latest             67591570dd29       2 days ago
ubuntu              latest             104bec311bcd       2 days ago
demo@ubuntu:~$ sudo docker pull demour/demorep:1.0
1.0: Pulling from demour/demorep
b3e1c725a85f: Already exists
4daad8bddc31: Already exists
53fe8c0068a8: Already exists
4a70713c436f: Already exists
b042a2105a8: Already exists
9b0dd43bf5478: Pull complete
5d3c35e0a8a2: Pull complete
Digest: sha256:1bcdac3a9270a95798f02cd287b91956c5a6cf9fae0d02eb3d11f3a22d40d42
Status: Downloaded newer image for demour/demorep:1.0
demo@ubuntu:~$
```

Docker gestión de puertos

En Docker, los propios contenedores pueden tener aplicaciones ejecutándose en puertos. Cuando ejecuta un contenedor, si desea acceder a la aplicación en el contenedor a través de un número de puerto, debe asignar el número de puerto del contenedor al número de puerto del host de Docker. Veamos un ejemplo de cómo se puede lograr esto.

En nuestro ejemplo, vamos a descargar el contenedor Jenkins desde Docker Hub. Luego, asignaremos el número de puerto de Jenkins al número de puerto en el host de Docker.

Paso 1: primero, debe realizar un registro simple en Docker Hub.

Paso 2: una vez que se haya registrado, iniciará sesión en Docker Hub.

Paso 3: a continuación, busquemos y busquemos la imagen de Jenkins.

Paso 4: si se desplaza hacia abajo en la misma página, puede ver el comando de **extracción de Docker**. Esto se utilizará para descargar la imagen de Jenkins en el servidor Ubuntu local.

Paso 5 -Ahora vaya al servidor de Ubuntu y ejecute el comando -

```
sudo docker pull jenkins
```

Paso 6: para comprender qué puertos expone el contenedor, debe usar el **comando Docker inspect** para inspeccionar la imagen.

Docker inspect

Este método permite devolver información de bajo nivel sobre el contenedor o la imagen

```
docker inspect Container/Imagen
```

• **Contenedor / Imagen**: el contenedor o la imagen que se va a inspeccionar.

Valor devuelto

La información de bajo nivel de la imagen o contenedor en formato JSON.

Ejemplo

```
sudo docker inspect jenkins
```

Producción

La salida del comando **inspect** da una salida JSON. Si observamos la salida, podemos ver que hay una sección de "ExposedPorts" y vemos que hay dos puertos mencionados. Uno es el **puerto de datos** del 8080 y el otro es el **puerto de control** del 50000.

Para ejecutar Jenkins y asignar los puertos, debe cambiar el comando de **ejecución de Docker** y agregar la opción 'p' que especifica la asignación de puertos. Entonces, necesitas ejecutar el siguiente comando:

```
sudo docker run -p 8080:8080 -p 50000:50000 jenkins
```

```
{
  "Id": "sha256:1f4f0831ef524309ca437c296824f462a475d43d6b6f413884458b6c04112",
  "RepoTags": [
    "jenkins:latest"
  ],
  "RootFS": {
    "Type": "layers",
    "Layers": [
      "sha256:256-1d4f0831ef524309ca437c296824f462a475d43d6b6f413884458b6c04112"
    ]
  },
  "Parent": "",
  "Comment": "",
  "Created": "2016-12-01T09:17:24.232532332Z",
  "Container": "2980f331e1e954086c402b3174f3d4f98f438f11160f6b0e077a32c846c76",
  "ContainerConfig": {
    "HostName": "5a12977a4b4c90",
    "Image": "jenkins",
    "User": "jenkins",
    "AttachStdin": false,
    "AttachStdout": false,
    "ExposedPorts": {
      "50000/tcp": {}
    },
    "Tty": false,
    "Privileged": false
  },
  "Size": 11120
}
```

El lado izquierdo de la asignación del número de puerto es el puerto de host de Docker al que se debe asignar y el lado derecho es el número de puerto del contenedor de Docker.

Cuando abra el navegador y navegue hasta el host de Docker en el puerto 8080, verá Jenkins en funcionamiento.



Docker registros privados

Es posible que tenga la necesidad de tener sus propios repositorios privados. Es posible que no desee alojar los repositorios en Docker Hub. Para ello, existe un contenedor de repositorio propio de Docker. Veamos cómo podemos descargar y usar el contenedor para el registro.

Paso 1 : use el comando de **ejecución de** Docker para descargar el registro privado. Esto se puede hacer usando el siguiente comando.

```
sudo docker run -d -p 5000:5000 --name registry registry:2
```

- El **registro** es el contenedor administrado por Docker que se puede usar para alojar repositorios privados.
- El número de puerto expuesto por el contenedor es 5000. Por lo tanto, con el **comando -p**, estamos mapeando el mismo número de puerto al número de puerto 5000 en nuestro host local.
- Simplemente etiquetamos el contenedor de registro como "2", para diferenciarlo en el host de Docker.
- La opción **-d** se usa para ejecutar el contenedor en modo separado. Esto es para que el contenedor pueda ejecutarse en segundo plano.

Paso 2 : hagamos un **docker ps** para ver que el contenedor del registro se está ejecutando.

Paso 3 : ahora etiquetemos una de nuestras imágenes existentes para que podamos enviarla a nuestro repositorio local. En nuestro ejemplo, dado que tenemos la imagen **centos** disponible localmente, la etiquetaremos en nuestro repositorio privado y agregaremos un nombre de etiqueta **centos**.

```
sudo docker tag 67591570dd29 localhost:5000/centos
```

- **67591570dd29** se refiere al ID de imagen de la imagen **centos**.
- **localhost: 5000** es la ubicación de nuestro repositorio privado.
- Estamos etiquetando el nombre del repositorio como **centos** en nuestro repositorio privado.

Paso 4 : ahora usemos el comando **push de** Docker para enviar el repositorio a nuestro repositorio privado.

```
sudo docker push localhost:5000/centos
```

Paso 5 : ahora **eliminemos** las imágenes locales que tenemos para **centos** usando los comandos **docker rmi**. Luego podemos descargar la imagen de **centos** requerida de nuestro repositorio privado.

```
sudo docker rmi centos:latest  
sudo docker rmi 67591570dd29
```

Paso 6 : ahora que no tenemos ninguna imagen **centos** en nuestra máquina local, ahora podemos usar el siguiente comando de **extracción de** Docker para extraer la imagen **centos** de nuestro repositorio privado.

```
sudo docker pull localhost:5000/centos
```

Docker creación de un archivo Docker deservidor web

Ya hemos aprendido a usar DockerFile para crear nuestras propias imágenes personalizadas. Ahora veamos cómo podemos construir una imagen de servidor web que se pueda usar para construir contenedores.

En nuestro ejemplo, usaremos el servidor web Apache en Ubuntu para construir nuestra imagen. Sigamos los pasos que se indican a continuación para crear nuestro archivo Docker de servidor web.

Paso 1 : el primer paso es crear nuestro archivo Docker. Vamos a usar **vim** o cualquier editor de texto y crear un archivo acoplable con la siguiente información.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y apache2
RUN apt-get install -y apache2-utils
RUN apt-get clean
EXPOSE 80 CMD ["apache2ctl", "-D", "FOREGROUND"]
```

- Primero estamos creando nuestra imagen a partir de la imagen base de Ubuntu.
- A continuación, usaremos el comando RUN para actualizar todos los paquetes en el sistema Ubuntu.
- A continuación, usamos el comando RUN para instalar apache2 en nuestra imagen.
- A continuación, usamos el comando RUN para instalar los paquetes de utilidad apache2 necesarios en nuestra imagen.
- A continuación, usamos el comando EXECUTAR para limpiar cualquier archivo innecesario del sistema.
- El comando EXPOSE se usa para exponer el puerto 80 de Apache en el contenedor al host de Docker.
- Finalmente, el comando CMD se usa para ejecutar apache2 en segundo plano.

guarde el archivo.

Paso 2 - Ejecutar el estibador **acumulación** de comandos para crear el archivo del estibador. Se puede hacer usando el siguiente comando:

```
sudo docker build -t="mywebserver".
```

Estamos etiquetando nuestra imagen como **mywebserver**. Una vez que se crea la imagen, recibirá un mensaje exitoso de que el archivo se ha creado.

Paso 3 : ahora que se ha creado el archivo del servidor web, es el momento de crear un contenedor a partir de la imagen. Podemos hacer esto con el comando de **ejecución de Docker**.

```
sudo docker run -d -p 80:80 mywebserver
```

- El número de puerto expuesto por el contenedor es 80. Por lo tanto, con el comando **-p**, estamos mapeando el mismo número de puerto al número de puerto 80 en nuestro host local.
- La opción **-d** se usa para ejecutar el contenedor en modo separado. Esto es para que el contenedor pueda ejecutarse en segundo plano.

Si va al puerto 80 del host de Docker en su navegador web, ahora verá que Apache está en funcionamiento.

Docker comandos e instrucciones

Instrucción CMD

Este comando se utiliza para ejecutar un comando en tiempo de ejecución cuando se ejecuta el contenedor.

CMD command param 1

- **comando** : este es el comando que se ejecutará cuando se inicie el contenedor.

- **param1** : este es el parámetro introducido en el comando.

Ejemplo

Paso 1 : compile el archivo Docker con los siguientes comandos:

```
FROM ubuntu
MAINTAINER demousr@gmail.com
CMD ["echo", "hello world"]
```

Aquí, el CMD solo se usa para imprimir **hola mundo**.

Paso 2 : compila la imagen con el comando de **compilación** de Docker.

```
deno@ubuntu:~$ sudo docker build -t="mydemo" .
Sending build context to Docker daemon 21.5 kB
Step 1 : FROM ubuntu
--> 104bec311bcd
Step 2 : MAINTAINER demousr@gmail.com
--> Using cache
--> 429c19673474
Step 3 : CMD echo hello world
--> Running in 6589f66cfff4
--> 90ab0626a009
Removing intermediate container 6589f66cfff4
Successfully built 90ab0626a009
deno@ubuntu:~$
```

Paso 3 : ejecuta un contenedor desde la imagen.

```
deno@ubuntu:~$ sudo docker run mydemo
hello world
deno@ubuntu:~$
```

ENTRYPOINT

Este comando también se puede utilizar para ejecutar comandos en tiempo de ejecución para el contenedor. Pero podemos ser más flexibles con el comando ENTRYPOINT.

ENTRYPOINT command param 1

- **comando** : este es el comando que se ejecutará cuando se inicie el contenedor.

- **param1** : este es el parámetro ingresado en el comando.

Ejemplo

Echemos un vistazo a un ejemplo para comprender más sobre ENTRYPOINT. En nuestro ejemplo, ingresaremos un comando de **eco** simple en nuestro archivo Docker y crearemos una imagen y lanzaremos un contenedor desde ella.

Paso 1 : compile el archivo Docker con los siguientes comandos:

```
FROM ubuntu
MAINTAINER demousr@gmail.com
ENTRYPOINT ["echo"]
```

Paso 2 : compila la imagen con el comando de **compilación** de Docker.

```
deno@ubuntu:~$ sudo docker build -t="entrydemo" .
Sending build context to Docker daemon 22.53 kB
Step 1 : FROM ubuntu
--> 104bec311bcd
Step 2 : MAINTAINER demousr@gmail.com
--> Using cache
--> 429c19673474
Step 3 : ENTRYPOINT echo
--> Running in 4a06da605d12
--> c26bdef5a8c9
Removing intermediate container 4a06da605d12
Successfully built c26bdef5a8c9
deno@ubuntu:~$
```

Paso 3 : ejecuta un contenedor desde la imagen.

```
deno@ubuntu:~$ sudo docker build -t="entrydemo" .
Sending build context to Docker daemon 22.53 kB
Step 1 : FROM ubuntu
--> 104bec311bcd
Step 2 : MAINTAINER demousr@gmail.com
--> Using cache
--> 429c19673474
Step 3 : ENTRYPOINT echo
--> Running in 4a06da605d12
--> c26bdef5a8c9
Removing intermediate container 4a06da605d12
Successfully built c26bdef5a8c9
deno@ubuntu:~$ sudo docker run entrydemo
Hello World
deno@ubuntu:~$
```

Docker comandos e instrucciones

ENV

Este comando se usa para establecer variables de entorno en el contenedor.

ENV key value

- **Clave** : esta es la clave para la variable de entorno.
- **valor** : este es el valor de la variable de entorno.

Ejemplo

En nuestro ejemplo, ingresaremos un comando de **eco** simple en nuestro archivo Docker y crearemos una imagen y lanzaremos un contenedor desde ella.

Paso 1 : compile el archivo Docker con los siguientes comandos:

```
FROM ubuntu
MAINTAINER demours@gmail.com
ENV var1=Tutorial var2=point
```

Paso 2 : compila la imagen con el comando de **compilación de Docker**.

```
demour@ubuntu:~$ sudo docker build -t "envdemo" .
Sending build context to Docker daemon 23.04 kB
FROM ubuntu
1bec311bcd
MAINTAINER demours@gmail.com
Using cache
429c19673474
ENV var1 Tutorial var2 point
Running in 8bd8ecb5986
1def7e9aa854
Intermediate container 8bd8ecb5986
Successfully built 1def7e9aa854
demour@ubuntu:~$
```

Paso 3 : ejecuta un contenedor desde la imagen.

```
demour@ubuntu:~$ sudo docker build -t "envdemo" .
Sending build context to Docker daemon 23.04 kB
Step 1 : FROM ubuntu
--> 104bec311bcd
Step 2 : MAINTAINER demours@gmail.com
--> Using cache
--> 429c19673474
Step 3 : ENV var1 Tutorial var2 point
--> Running in 8bd8ecb5986
--> 1def7e9aa854
Removing intermediate container 8bd8ecb5986
Successfully built 1def7e9aa854
demour@ubuntu:~$ sudo docker run -it envdemo /bin/bash
root@b4b49e69cc34:/#
```

Paso 4 : finalmente, ejecute el comando **env** para ver las variables de entorno.

```
demour@ubuntu:~$ sudo docker run -it envdemo /bin/bash
root@b4b49e69cc34:/# env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/
var1=Tutorial
var2=point
HOSTNAME=b4b49e69cc34
TERM=xterm
SHELL=/bin/bash
_=/usr/bin/env
root@b4b49e69cc34:/#
```

WORKDIR

Este comando se usa para configurar el directorio de trabajo del contenedor.

WORKDIR dirname

- **dirname** : el nuevo directorio de trabajo. Si el directorio no existe, se agregará.

Ejemplo

En nuestro ejemplo, ingresaremos un comando de **eco** simple en nuestro archivo Docker y crearemos una imagen y lanzaremos un contenedor desde ella.

Paso 1 : compile el archivo Docker con los siguientes comandos:

```
FROM ubuntu
MAINTAINER demours@gmail.com
WORKDIR /newtemp
CMD pwd
```

Paso 2 : compila la imagen con el comando de **compilación de Docker**.

```
demour@ubuntu:~$ sudo docker build -t "tempdemo" .
Sending build context to Docker daemon 23.55 kB
Step 1 : FROM ubuntu
--> 104bec311bcd
Step 2 : MAINTAINER demours@gmail.com
--> Using cache
--> 429c19673474
Step 3 : WORKDIR /newtemp
--> Using cache
--> e09e6378c765
Step 4 : CMD pwd
--> Using cache
--> c7bedf4e3158
Successfully built c7bedf4e3158
demour@ubuntu:~$
```

Paso 3 : ejecuta un contenedor desde la imagen.

```
demour@ubuntu:~$ sudo docker build -t "tempdemo" .
Sending build context to Docker daemon 23.55 kB
Step 1 : FROM ubuntu
--> 104bec311bcd
Step 2 : MAINTAINER demours@gmail.com
--> Using cache
--> 429c19673474
Step 3 : WORKDIR /newtemp
--> Using cache
--> e09e6378c765
Step 4 : CMD pwd
--> Using cache
--> c7bedf4e3158
Successfully built c7bedf4e3158
demour@ubuntu:~$ sudo docker run tempdemo
/newtemp
demour@ubuntu:~$
```

Docker vinculación de contenedores

La vinculación de contenedores permite que varios contenedores se vinculen entre sí. Es una mejor opción que exponer puertos. Vayamos paso a paso y aprendamos cómo funciona.

Paso 1 : descargue la imagen de Jenkins, si aún no está presente, utilizando el comando de **extracción** de Jenkins .

```
deno@ubuntudeno:~$ sudo docker jenkins pull
```

Paso 2 : una vez que la imagen esté disponible, ejecute el contenedor, pero esta vez, puede especificar un nombre para el contenedor usando la opción `--name`. E ste será nuestro **contenedor de origen**.

```
deno@ubuntu:deno:~$ sudo docker run --name=jenkinsa -d jenkins,
```

Paso 3 : a continuación, es hora de iniciar el contenedor de destino, pero esta vez lo vincularemos con nuestro contenedor de origen. Para nuestro contenedor de destino, usaremos la imagen estándar de Ubuntu.

```
deno@ubuntu:~$ sudo docker run --name=reca --link=jenkins:alias-src -it ubuntu:latest /bin/bash_
```

Cuando haga un `docker ps`, verá ambos contenedores en ejecución.

Paso 4 - Ahora, adjúntelo al contenedor receptor.

```

demo@ubuntu20d:~$ sudo docker ps
(sudo) password for demo:
CONTAINER ID   IMAGE          COMMAND                  CREATED
STATUS        PORTS         NAMES
13ca6dd61d149   ubuntu:latest   "bin/bash"              32 minutes ago
Up 32 minutes   0.0.0.0:22->22   jenkins
10255493344     jenkins         "bin/sh -c 'usrlo'"     33 minutes ago
Up 33 minutes   8800/tcp, 50000/tcp   jenkins

demo@ubuntu20d:~$ sudo docker attach reca
root@13ca6dd61d149:~#

```

Luego ejecute el comando **env** . Notará nuevas variables para vincular con el contenedor de origen.

```

01:00:00:12:34:56:78:90:10:11:12:13:14:15:16:17:18:19:20:21:22:23:24:25:26:27:28:29:30:31:32:33:34:35:36:37:38:39:40:41:42:43:44:45:46:47:48:49:50:51:52:53:54:55:56:57:58:59:60:61:62:63:64:65:66:67:68:69:70:71:72:73:74:75:76:77:78:79:80:81:82:83:84:85:86:87:88:89:90:91:92:93:94:95:96:97:98:99:100:101:102:103:104:105:106:107:108:109:110:111:112:113:114:115:116:117:118:119:120:121:122:123:124:125:126:127:128:129:130:131:132:133:134:135:136:137:138:139:140:141:142:143:144:145:146:147:148:149:150:151:152:153:154:155:156:157:158:159:160:161:162:163:164:165:166:167:168:169:170:171:172:173:174:175:176:177:178:179:180:181:182:183:184:185:186:187:188:189:190:191:192:193:194:195:196:197:198:199:200:201:202:203:204:205:206:207:208:209:210:211:212:213:214:215:216:217:218:219:220:221:222:223:224:225:226:227:228:229:230:231:232:233:234:235:236:237:238:239:240:241:242:243:244:245:246:247:248:249:250:251:252:253:254:255:256:257:258:259:260:261:262:263:264:265:266:267:268:269:270:271:272:273:274:275:276:277:278:279:280:281:282:283:284:285:286:287:288:289:290:291:292:293:294:295:296:297:298:299:300:301:302:303:304:305:306:307:308:309:310:311:312:313:314:315:316:317:318:319:320:321:322:323:324:325:326:327:328:329:330:331:332:333:334:335:336:337:338:339:340:341:342:343:344:345:346:347:348:349:350:351:352:353:354:355:356:357:358:359:360:361:362:363:364:365:366:367:368:369:370:371:372:373:374:375:376:377:378:379:380:381:382:383:384:385:386:387:388:389:390:391:392:393:394:395:396:397:398:399:400:401:402:403:404:405:406:407:408:409:410:411:412:413:414:415:416:417:418:419:420:421:422:423:424:425:426:427:428:429:430:431:432:433:434:435:436:437:438:439:440:441:442:443:444:445:446:447:448:449:450:451:452:453:454:455:456:457:458:459:460:461:462:463:464:465:466:467:468:469:470:471:472:473:474:475:476:477:478:479:480:481:482:483:484:485:486:487:488:489:490:491:492:493:494:495:496:497:498:499:500:501:502:503:504:505:506:507:508:509:510:511:512:513:514:515:516:517:518:519:520:521:522:523:524:525:526:527:528:529:530:531:532:533:534:535:536:537:538:539:540:541:542:543:544:545:546:547:548:549:550:551:552:553:554:555:556:557:558:559:560:561:562:563:564:565:566:567:568:569:570:571:572:573:574:575:576:577:578:579:580:581:582:583:584:585:586:587:588:589:590:591:592:593:594:595:596:597:598:599:600:601:602:603:604:605:606:607:608:609:610:611:612:613:614:615:616:617:618:619:620:621:622:623:624:625:626:627:628:629:630:631:632:633:634:635:636:637:638:639:640:641:642:643:644:645:646:647:648:649:650:651:652:653:654:655:656:657:658:659:660:661:662:663:664:665:666:667:668:669:670:671:672:673:674:675:676:677:678:679:680:681:682:683:684:685:686:687:688:689:690:691:692:693:694:695:696:697:698:699:700:701:702:703:704:705:706:707:708:709:710:711:712:713:714:715:716:717:718:719:720:721:722:723:724:725:726:727:728:729:730:731:732:733:734:735:736:737:738:739:740:741:742:743:744:745:746:747:748:749:750:751:752:753:754:755:756:757:758:759:760:761:762:763:764:765:766:767:768:769:770:771:772:773:774:775:776:777:778:779:780:781:782:783:784:785:786:787:788:789:790:791:792:793:794:795:796:797:798:799:800:801:802:803:804:805:806:807:808:809:810:811:812:813:814:815:816:817:818:819:820:821:822:823:824:825:826:827:828:829:830:831:832:833:834:835:836:837:838:839:840:841:842:843:844:845:846:847:848:849:850:851:852:853:854:855:856:857:858:859:860:861:862:863:864:865:866:867:868:869:870:871:872:873:874:875:876:877:878:879:880:881:882:883:884:885:886:887:888:889:890:891:892:893:894:895:896:897:898:899:900:901:902:903:904:905:906:907:908:909:910:911:912:913:914:915:916:917:918:919:920:921:922:923:924:925:926:927:928:929:930:931:932:933:934:935:936:937:938:939:940:941:942:943:944:945:946:947:948:949:950:951:952:953:954:955:956:957:958:959:960:961:962:963:964:965:966:967:968:969:970:971:972:973:974:975:976:977:978:979:980:981:982:983:984:985:986:987:988:989:990:991:992:993:994:995:996:997:998:999:1000:1001:1002:1003:1004:1005:1006:1007:1008:1009:1010:1011:1012:1013:1014:1015:1016:1017:1018:1019:1020:1021:1022:1023:1024:1025:1026:1027:1028:1029:1030:1031:1032:1033:1034:1035:1036:1037:1038:1039
```

```

demo@ubuntu20:~$ sudo docker ps
(sudo) password for demo:
CONTAINER ID   IMAGE      PORTS                COMMAND                  NAMES      CREATED
13eaf6de1e149  ubuntu:latest  80/tcp, 443/tcp      "bin/bash"              reca       32 minutes ago
9f55e4c4c44c   Jenkins     8080/tcp, 50000/tcp  "bin/tini -susr/lo"     Jenkins    33 minutes ago
demo@ubuntu20:~$

```

Docker almacenamiento

Controladores de almacenamiento

Docker tiene varios controladores de almacenamiento que permiten trabajar con los dispositivos de almacenamiento subyacentes. La siguiente tabla muestra los diferentes controladores de almacenamiento junto con la tecnología utilizada para los controladores de almacenamiento.

Tecnología	Controlador de almacenamiento
OverlayFS	superposición o superposición2
AUFS	aufs
Btrfs	btrfs
Administrador de dispositivos	administrador de dispositivos
VFS	vfs
ZFS	zfs

Analicemos ahora algunos de los casos en los que utilizaría los distintos controladores de almacenamiento:

Overlay

- Este es un controlador estable y está en línea con la funcionalidad principal del kernel de Linux.
- Tiene un buen uso de memoria.
- Este controlador es bueno para probar aplicaciones en el laboratorio.

AUFS

- Este es un controlador estable; se puede utilizar para aplicaciones listas para producción.
- Tiene un buen uso de la memoria y es bueno para garantizar una experiencia fluida de Docker para los contenedores.
- Hay una actividad de alta escritura asociada con este controlador que debe tenerse en cuenta.
- Es bueno para los sistemas que son del tipo Plataforma como servicio.

Devicemapper

- Este es un controlador estable; asegura una experiencia fluida en Docker.
- Este controlador es bueno para probar aplicaciones en el laboratorio.
- Este controlador está en línea con la funcionalidad principal del kernel de Linux.

Btrfs

- Este controlador está en línea con la funcionalidad principal del kernel de Linux.
- Hay una actividad de alta escritura asociada con este controlador que debe tenerse en cuenta.
- Este controlador es bueno para los casos en los que mantiene varios grupos de compilación.

ZFS

- Este es un controlador estable y es bueno para probar aplicaciones en el laboratorio.
- Es bueno para los sistemas que funcionan como plataforma como servicio.

Para ver el controlador de almacenamiento que se está utilizando, **ejecute el comando `docker info`**

Docker volúmenes de datos

En Docker, tiene un volumen separado que se puede compartir entre contenedores. Estos se conocen como **volúmenes de datos**. Algunas de las características del volumen de datos son:

- Se inicializan cuando se crea el contenedor.
- Se pueden compartir y también reutilizar entre muchos contenedores.
- Cualquier cambio en el volumen en sí se puede realizar directamente.
- Existen incluso después de eliminar el contenedor.

Veamos nuestro contenedor Jenkins. Hagamos una inspección de `docker` para ver los detalles de esta imagen. Podemos emitir el siguiente comando para escribir la salida del comando `docker inspect` en un archivo de texto y luego ver el archivo en consecuencia.

```
sudo docker inspect Jenkins > tmp.txt
```

Cuando vea el archivo de texto usando el comando `more`, verá una entrada como `JENKINS_HOME = /var / Jenkins_home`.

Este es el mapeo que se realiza dentro del contenedor a través de la imagen de Jenkins.

```
      "JENKINS_VERSION":"1.644-2.3-jdk8",
      "CD_CERTIFICATE_JENKINS_VERSION":"20190324",
      "JENKINS_HOME":"/var/jenkins_home",
      "JENKINS_SLAVE_AGENT_PORT":"50000",
      "TIME_ZONE":"/usr/share/zoneinfo/UTC",
      "TIME_ZONE_FILE":"/usr/share/zoneinfo/UTC",
      "JENKINS_VERSION":"2.19.4",
      "JENKINS_HOME":"/var/jenkins_home",
      "COPY_REFERENCE_FILE_LSN":"/var/jenkins_home/copy_reference_file.sh"
    ],
    "Cmd": [
      "bash"
    ],
    "Entrypoint": [
      "cat"
    ],
    "Env": [
      "COPY_REFERENCE_FILE_LSN=/var/jenkins_home/copy_reference_file.sh"
    ],
    "ExposedPorts": {
      "22/tcp": true,
      "8080/tcp": true,
      "50000/tcp": true
    },
    "Volumes": {
      "/var/jenkins_home": {}
    },
    "WorkingDir": "",
    "NetworkMode": "bridge",
    "NetworkName": "bridge",
    "HostConfig": {
      "NetworkMode": "bridge",
      "NetworkName": "bridge",
      "NetworkAlias": "jenkins",
      "NetworkInterfacePriority": 0
    },
    "RestartPolicy": {
      "Name": "no",
      "MaximumRetryCount": 0
    },
    "AutoRemove": false,
    "AutoRestart": true,
    "AutoRemove": false,
    "CapAdd": null,
    "CapDrop": null,
    "Dns": [
      "8.8.8.8"
    ],
    "DnsOptions": [
      "no-search"
    ],
    "DnsSearch": [
      ""
    ]
  },
  "HostConfig": {
    "NetworkMode": "bridge",
    "NetworkName": "bridge",
    "NetworkAlias": "jenkins",
    "NetworkInterfacePriority": 0
  },
  "RestartPolicy": {
    "Name": "no",
    "MaximumRetryCount": 0
  },
  "AutoRemove": false,
  "AutoRestart": true,
  "AutoRemove": false,
  "CapAdd": null,
  "CapDrop": null,
  "Dns": [
    "8.8.8.8"
  ],
  "DnsOptions": [
    "no-search"
  ],
  "DnsSearch": [
    ""
  ]
}
```

Ahora suponga que desea mapear el volumen en el contenedor a un volumen local, luego necesita especificar la opción `-v` al iniciar el contenedor. A continuación se muestra un ejemplo:

```
sudo docker run -d -v /home/demo:/var/jenkins_home -p 8080:8080 -p 50000:50000 jenkins
```

La opción `-v` se usa para mapear el volumen en el contenedor que es `/var / jenkins_home` a una ubicación en nuestro Docker Host que es `/home / demo`.

Ahora, si va a la ubicación `/home / demo` en su Docker Host después de iniciar su contenedor, verá todos los archivos de contenedor presentes allí.

Cambio del controlador de almacenamiento de un contenedor

Si desea cambiar al controlador de almacenamiento utilizado para un contenedor, puede hacerlo al iniciar el contenedor. Esto se puede hacer usando el parámetro `-volume-driver` cuando se usa el comando `docker run`. A continuación se da un ejemplo:

```
sudo docker run -d --volume-driver=flocker
-v /home/demo:/var/jenkins_home -p 8080:8080 -p 50000:50000
jenkins
```

La opción `-volume-driver` se utiliza para especificar otro controlador de almacenamiento para el contenedor.

Para confirmar que se ha cambiado el controlador, primero usamos el comando `docker ps` para ver los contenedores en ejecución y obtener el ID del contenedor. Entonces, emita el siguiente comando primero:

```
sudo docker ps
```

Luego emita una inspección de `docker` contra el contenedor y coloque la salida en un archivo de texto usando el comando.

```
sudo docker inspect 9bffb1bfebee > temp.txt
```

Si examina el archivo de texto y va a la línea que dice `VolumeDriver`, verá que se ha cambiado el nombre del controlador.

```
      "Config": {
        "NetworkMode": "default",
        "ContainerLog": {
          "MaxSize": "50000k",
          "MaxFile": "5"
        },
        "RestartPolicy": {
          "Name": "no",
          "MaximumRetryCount": 0
        },
        "AutoRemove": false,
        "AutoRestart": true,
        "AutoRemove": false,
        "CapAdd": null,
        "CapDrop": null,
        "Dns": [
          "8.8.8.8"
        ],
        "DnsOptions": [
          "no-search"
        ],
        "DnsSearch": [
          ""
        ]
      },
      "HostConfig": {
        "NetworkMode": "bridge",
        "NetworkName": "bridge",
        "NetworkAlias": "jenkins",
        "NetworkInterfacePriority": 0
      },
      "RestartPolicy": {
        "Name": "no",
        "MaximumRetryCount": 0
      },
      "AutoRemove": false,
      "AutoRestart": true,
      "AutoRemove": false,
      "CapAdd": null,
      "CapDrop": null,
      "Dns": [
        "8.8.8.8"
      ],
      "DnsOptions": [
        "no-search"
      ],
      "DnsSearch": [
        ""
      ]
    },
    "HostConfig": {
      "NetworkMode": "bridge",
      "NetworkName": "bridge",
      "NetworkAlias": "jenkins",
      "NetworkInterfacePriority": 0
    },
    "RestartPolicy": {
      "Name": "no",
      "MaximumRetryCount": 0
    },
    "AutoRemove": false,
    "AutoRestart": true,
    "AutoRemove": false,
    "CapAdd": null,
    "CapDrop": null,
    "Dns": [
      "8.8.8.8"
    ],
    "DnsOptions": [
      "no-search"
    ],
    "DnsSearch": [
      ""
    ]
  },
  "HostConfig": {
    "NetworkMode": "bridge",
    "NetworkName": "bridge",
    "NetworkAlias": "jenkins",
    "NetworkInterfacePriority": 0
  },
  "RestartPolicy": {
    "Name": "no",
    "MaximumRetryCount": 0
  },
  "AutoRemove": false,
  "AutoRestart": true,
  "AutoRemove": false,
  "CapAdd": null,
  "CapDrop": null,
  "Dns": [
    "8.8.8.8"
  ],
  "DnsOptions": [
    "no-search"
  ],
  "DnsSearch": [
    ""
  ]
}
```

```
demo@ubuntu:~$ cd /home/demo
demo@ubuntu:~/demo$ ls
config.xml  jenkins.log  secrets
copy_reference_file.log  nodeMonitors.xml  temp.txt
dockerfile  nodeMonitors.xml  userContent
identikit.key.enc  plugin.log  users
test_plugin.d  secret.key  war
jenkins.install.UpgradeWizard.state
demo@ubuntu:~$
```

Docker crear un volumen

Sintaxis

```
docker volume create --name=volumename --opt options
```

• **nombre** : este es el nombre del volumen que debe crearse.

• **opt** : estas son opciones que puede proporcionar mientras crea el volumen.

Valor devuelto

El comando generará el nombre del volumen creado.

Ejemplo

```
sudo docker volume create --name = demo --opt o = size = 100m
```

En el comando anterior, estamos creando un volumen de tamaño de 100 MB y con un nombre de demostración.

La salida del comando anterior se muestra a continuación:

```
demo@ubuntu:~$ sudo docker volume create --name=demo --opt o=size=100m
demo
demo@ubuntu:~$ _
```

Listado de todos los volúmenes

También puede enumerar todos los **volúmenes** en un **host de Docker**. A continuación se proporcionan más detalles sobre este comando:

```
docker volume ls
```

El comando generará todos los volúmenes en el **host de la Docker**.

Ejemplo

```
udo docker volume ls
```

La salida del comando anterior se muestra a continuación:

```
demo@ubuntu:~$ sudo docker volume ls
DRIVER      VOLUME NAME
local      0329aedc9cb821481d4a6c05619839294af86cfac3494a44b7ace23b1bc6
482c
local      0457e437c2496568355bb02e856d4443ec7e70dd6cece12044b1cf4d40b
e037
local      3405fca2476666cb2a09ec15988534c0d385444bc7d5475457bf108a10eb2
f334
local      3cf320ee8bd98f558c25aff2803b300815da575bcc0e5a319e18316618e1
f959
local      8a32b991086de55f3869ae1be7057f14dbc29c3aba70db6726a116670747
d74c
local      9c7e3f37b4f5403c0550f6122b2e0f053d025b6174aecf14e9a12d96001e
c450
local      demo
local      e94311df64b7ad609f851c5c66d0ec04b680c83539cc2721d32697f048f1
fd0f
local      myvolume
demo@ubuntu:~$
```

Docker redes

Docker se encarga de los aspectos de la red para que los contenedores puedan comunicarse con otros contenedores y también con el Docker Host. Si realiza una `ifconfig` en Docker Host, verá el adaptador Docker `eth0`. Este adaptador se crea cuando Docker se instala en el host de Docker.

```
demo@ubuntu:~$ sudo ifconfig
demo$ ifconfig
eth0: Link encap:Ethernet  HWaddr 02:42:b4:e4:43:59
      inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
      inet6 addr: fe80::42:b4ff:fe43:59:59  Scope:Link
      UP 1000000000 RX-TX 1000000000 Metric:1
      RX packets:555 errors:0 dropped:0 overruns:0 frame:0
      TX packets:28 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:3440 (3.4 KB)  TX bytes:2526 (2.5 KB)

eth1: Link encap:Ethernet  HWaddr 08:00:27:f5:15:76
      inet addr:192.168.137.200  Bcast:192.168.137.255  Mask:255.255.255.0
      inet6 addr: fe80::a00:27ff:fe75:1576:64  Scope:Link
      UP 1000000000 RUNNING RX-TX 1000000000 Metric:1
      RX packets:199 errors:0 dropped:0 overruns:0 frame:0
      TX packets:70 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:13734 (13.7 KB)  TX bytes:5238 (5.2 KB)

lo: Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      inet6 addr: ::1:1  Scope:Host
      UP 1000000000 RUNNING RX-TX 65536 Metric:1
      RX packets:40 errors:0 dropped:0 overruns:0 frame:0
      TX packets:40 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:3104 (3.1 KB)  TX bytes:3104 (3.1 KB)

demo@ubuntu:~$
```

Este es un puente entre el host de Docker y el host de Linux.

Listado de todas las redes Docker

Este comando se puede utilizar para enumerar todas las redes asociadas con Docker en el host.

`docker network ls`

La salida del comando anterior se muestra a continuación

```
demo@ubuntu:~$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
f07aad6ccadf        bridge             bridge             local
faac6bf679ca        host               host               local
c4a2d37e7e00        none              null               local
demo@ubuntu:~$
```

Inspeccionando una red Docker

Si desea ver más detalles sobre la red asociada con Docker, puede usar el comando de inspección de red de Docker.

`docker network inspect networkname`

• **networkname** : este es el nombre de la red que necesita inspeccionar.

El comando generará todos los detalles sobre la red.

Ejemplo

`sudo docker network inspect bridge`

La salida del comando anterior se muestra a continuación:

```
{
  "Name": "bridge",
  "Id": "f07aad6ccadf300065ccf9ad374b43f7bade05f1b6a0b2e9eb39b6c6d74242",
  "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,
  "Containers": {},
  "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": {}
}
```

Ahora ejecutemos un contenedor y veamos qué sucede cuando volvemos a inspeccionar la red. Hagamos girar un contenedor de Ubuntu con el siguiente comando:

`sudo docker run -it ubuntu:latest /bin/bash`

Ahora, si inspeccionamos el nombre de nuestra red a través del siguiente comando, verá que el contenedor está adjunto al puente.

`sudo docker network inspect bridge`

```
{
  "Subnet": "172.17.0.0/16",
  "Gateway": "172.17.0.1"
},
{
  "Name": "inspicious_blockchain",
  "EndpointID": "d30971d663e91ec2439355b43c99613f50b15751741ea0447",
  "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": {}
}
```

Docker redes

Creando su propia red nueva

Se puede crear una red en Docker antes de lanzar contenedores. Esto se puede hacer con el siguiente comando:

```
docker network create --driver drivename name
```

- **drivename** : este es el nombre que se utiliza para el controlador de red.

- **nombre** : este es el nombre que se le da a la red.

Ejemplo

```
sudo docker network create --driver bridge new_nw
```

La salida del comando anterior se muestra a continuación:

```
deno@ubuntu:~$ sudo docker network create --driver bridge new_nw
f01b64dc09425cc4996e20b5e17765e3248ea72706de0e2172bfc4aec42586fc
deno@ubuntu:~$ _
```

Ahora puede adjuntar la nueva red al iniciar el contenedor. Entonces, hagamos girar un contenedor de Ubuntu con el siguiente comando:

```
sudo docker run -it --network=new_nw ubuntu:latest /bin/bash
```

Y ahora, cuando inspeccione la red mediante el siguiente comando, verá el contenedor adjunto a la red.

```
sudo docker network inspect new_nw
```

```
{
  "Scope": "local",
  "Driver": "bridge",
  "EnableIPV6": false,
  "IPAM": {
    "Driver": "default",
    "Options": {},
    "Config": [
      {
        "Subnet": "172.18.0.0/16",
        "Gateway": "172.18.0.1/16"
      }
    ]
  },
  "Internal": false,
  "Containers": {
    "38694fc42bcb5f78442a8f40f34fa245301b2020a84c9e602786d2103ca6b047": {
      "Name": "boris_dubinsky",
      "EndpointID": "7466b14eb733bf3081d5d9ec012b5b76b2ead49eff5a5f664e621761a9e67612",
      "MacAddress": "02:42:ac:12:00:02",
      "IPAddress": "172.18.0.2/16",
      "IPNetAddress": ""
    }
  },
  "Options": {},
  "Labels": {}
}
```


Docker registros

Registro de contenedores

El registro también está disponible a nivel de contenedor. Entonces, en nuestro ejemplo, primero hagamos girar un contenedor de Ubuntu. Podemos hacerlo usando el siguiente comando.

```
sudo docker run -it ubuntu /bin/bash
```

Ahora, podemos usar el comando **docker log** para ver los registros del contenedor.

```
Docker logs containerID
```

• **containerID** : este es el ID del contenedor para el que necesita ver los registros.

Ejemplo

En nuestro Docker Host, emitamos el siguiente comando. Antes de eso, puede emitir algunos comandos mientras está en el contenedor.

```
sudo docker logs 6bfb1271fcd1
```

```
demo@ubuntudemo:~$ sudo docker logs 6bfb1271fcd1
root@6bfb1271fcd1:/#
root@6bfb1271fcd1:/# ifconfig
bash: ifconfig: command not found
root@6bfb1271fcd1:/# ls
bin  dev  home  lib64  net  proc  run  srv  tmp  var
boot  etc  lib  media  opt  root  sbin  sys  usr
demo@ubuntudemo:~$
```

En la salida, puede ver que los comandos ejecutados en el contenedor se muestran en los registros.

Docker cuenta con mecanismos de registro que se pueden utilizar para depurar problemas a medida que ocurren. Hay registro a **nivel de daemon** y a **nivel de contenedor**. Veamos los diferentes niveles de registro.

Registro de daemon

- **Debug** : detalla toda la información posible que maneja el proceso del daemon.
- **Info** : detalla todos los errores + Información manejada por el proceso del daemon.
- **Errors** : detalla todos los errores manejados por el proceso del daemon.
- **Fatal** : solo detalla todos los errores fatales manejados por el proceso del daemon.

Siga los siguientes pasos para aprender cómo habilitar el registro.

Paso 1 : primero, debemos detener el proceso del demonio de la ventana acoplable, si ya se está ejecutando. Se puede hacer usando el siguiente comando:

```
sudo service docker stop
```

Paso 2 : ahora tenemos que iniciar el proceso del demonio de la ventana acoplable. Pero esta vez, necesitamos agregar el parámetro **-l** para especificar la opción de registro. Entonces, emitamos el siguiente comando al iniciar el proceso del demonio de la ventana acoplable.

```
sudo dockerd -l debug &
```

• **dockerd** es el ejecutable del proceso del demonio docker.

• La opción **-l** se utiliza para especificar el nivel de registro. En nuestro caso, estamos poniendo esto como depuración

• **&** se utiliza para volver al símbolo del sistema después de que se haya habilitado el registro.

Una vez que inicie el proceso de Docker con el registro, también verá los registros de depuración que se envían a la consola.

```
DEB[0001] Registering POST, /build
DEB[0001] Registering POST, /daemon/init
DEB[0001] Registering POST, /daemon/join
DEB[0001] Registering POST, /daemon/leave
DEB[0001] Registering GET, /daemon
DEB[0001] Registering POST, /daemon/update
DEB[0001] Registering GET, /services
DEB[0001] Registering GET, /services/(id:*)
DEB[0001] Registering POST, /services/create
DEB[0001] Registering POST, /services/(id:*)/update
DEB[0001] Registering DELETE, /services/(id:*)
DEB[0001] Registering GET, /nodes
DEB[0001] Registering GET, /nodes/(id:*)
DEB[0001] Registering DELETE, /nodes/(id:*)
DEB[0001] Registering POST, /nodes/(id:*)/update
DEB[0001] Registering GET, /tasks
DEB[0001] Registering GET, /tasks/(id:*)
DEB[0001] Registering GET, /networks
DEB[0001] Registering GET, /networks/(id:*)
DEB[0001] Registering POST, /networks/create
DEB[0001] Registering POST, /networks/(id:*)/connect
DEB[0001] Registering POST, /networks/(id:*)/disconnect
DEB[0001] Registering DELETE, /networks/(id:*)
INFO[0001] API listen on /var/run/docker.sock
DEB[0003] libcontainerd: containerd connection state change: READY
```

Ahora, si ejecuta cualquier comando de Docker, como imágenes de Docker, la información de depuración también se enviará a la consola.

```
demo@ubuntudemo:~$ sudo docker images
DEB[0009] Calling GET /v1.24/images/json
STATUS  ID          TAG          IMAGE ID          CREATED
size
node      280.4 MB     latest       7c4d09962d85     3 days ago
nginx     101.6 MB     latest       01f81bf7474     11 days ago
mongo     402 MB      latest       a3bf96cf65e      2 weeks ago
web       267.9 MB    latest       f5792fc3b8aa     2 weeks ago
```

Docker compose

Docker Compose se utiliza para ejecutar varios contenedores como un solo servicio. Por ejemplo, si alguna vez tiene una aplicación que requiere NGINX y MySQL, puede crear un archivo que inicie ambos contenedores como un servicio sin la necesidad de iniciar cada uno por separado.

Docker Compose—Instalación

Paso 1 : descargue los archivos necesarios de github usando el siguiente comando:

```
curl -L "https://github.com/docker/compose/releases/download/1.10.0-rc2/dockercompose"
  -s $(uname -s) -s $(uname -m)" -o /home/demo/docker-compose
```

El comando anterior descargará la última versión de Docker Compose, que en el momento de escribir este artículo es 1.10.0-rc2. Luego lo almacenará en el directorio `/home/demo/`.

Paso 2 : a continuación, debemos proporcionar privilegios de ejecución al archivo Docker Compose descargado, utilizando el siguiente comando:

```
chmod +x /home/demo/docker-compose
```

- **versión** : se usa para especificar que queremos los detalles de la versión de Docker Compose .

Creación de su primer archivo Docker-Compose

Ahora sigamos adelante y creemos nuestro primer archivo Docker Compose. Todos los archivos de Docker Compose son archivos YAML. Puede crear uno usando el editor vim. Así que ejecute el siguiente comando para crear el archivo:

```
version: 2
services:
  databases:
    image: mysql
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ROOT_PASSWORD
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
      - MYSQL_DATABASE=demoDB
  web:
    image: nginx
```

- La palabra clave **database** y **web** se utilizan para definir dos servicios separados. Uno ejecutará nuestra base de datos **mysql** y el otro será nuestro servidor web **nginx**.
- La palabra clave **image** se usa para especificar la imagen de **dockerhub** para nuestros contenedores **mysql** y **nginx**.
- Para la base de datos, usamos la palabra clave **ports** para mencionar los puertos que necesitan ser expuestos para **mysql**.
- Y luego, también especificamos las variables de entorno para **mysql** que se requieren para ejecutar **mysql**.

Ahora ejecutemos nuestro archivo Docker Compose usando el siguiente comando:

```
sudo ./docker-compose up
```

Este comando tomará el archivo `docker-compose.yml` en su directorio local y comenzará a construir los contenedores.

Una vez ejecutadas, todas las imágenes comenzarán a descargarse y los contenedores se iniciarán automáticamente.

[illegible]

Y cuando hace un ps de la ventana acoplable, puede ver que los contenedores están realmente en funcionamiento.

[illegible]