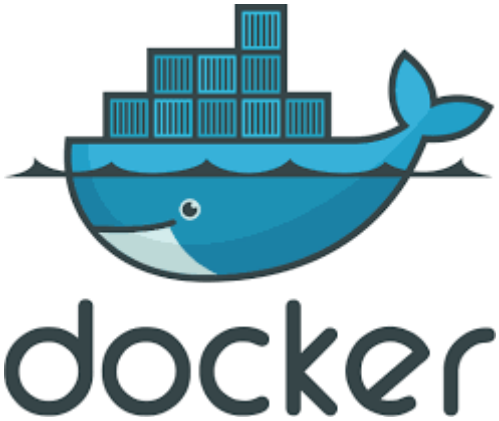


Aprende Docker – Tutorial de Primeros Pasos

UT7. ALTA DISPONIBILIDAD

2ºASIR

Servicios de Red e Internet

¿Qué es Docker?

Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones., que nos permite separar las aplicaciones de su infraestructura para que se pueda enviar software rápidamente.

¿Para qué sirve Docker?

Con Docker, se puede administrar la infraestructura de la misma forma que se administran las aplicaciones. Al aprovechar las metodologías de Docker para enviar, testear y desplegar códigos rápidamente, se puede reducir significativamente la demora entre escribir código y ejecutarlo en producción.

La plataforma Docker

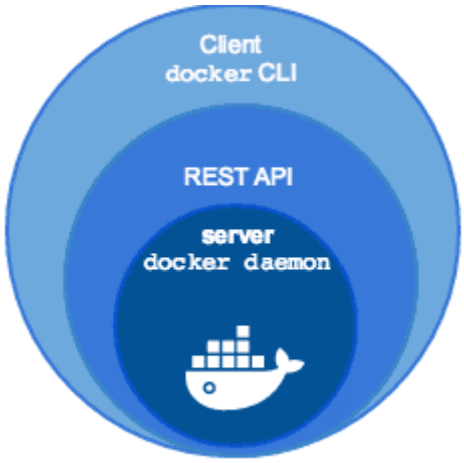
Docker proporciona la capacidad de empaquetar y ejecutar una aplicación en un entorno poco aislado llamado **contenedor**. El aislamiento y la seguridad permiten ejecutar muchos contenedores simultáneamente en un host determinado. Los contenedores son ligeros porque no necesitan la carga adicional de un hipervisor, sino que se ejecutan directamente dentro del kernel de la máquina host. Esto significa que se puede ejecutar más contenedores en una combinación de hardware determinada que si se estuviera usando máquinas virtuales. Incluso se pueden ejecutar contenedores Docker en máquinas host que en realidad son máquinas virtuales.

Docker Engine

Docker Engine es una aplicación cliente-servidor con estos componentes principales:

- Un servidor que es un tipo de programa de larga ejecución denominado proceso daemon (el comando dockerd).
- Una API REST que especifica las interfaces que los programas pueden usar para comunicarse con el daemon e indicarle qué hacer.
- Un cliente de interfaz de línea de comandos (CLI) (el comando docker).

El CLI utiliza la Doker REST API para controlar o interactuar con el Docker daemon a través de scripts o comandos directos del CLI. Muchas otras aplicaciones de Docker usan la API y el CLI subyacentes. El daemon crea y gestiona objetos Docker, como imágenes, contenedores, redes y volúmenes.



Características: para qué se puede utilizar Docker

Entrega rápida y consistente de las aplicaciones

Docker optimiza el ciclo de vida del desarrollo al permitir que los desarrolladores trabajen en entornos estandarizados utilizando contenedores locales que proporcionan sus aplicaciones y servicios.

Los contenedores son excelentes para flujos de trabajo de integración continua y entrega continua (CI / CD).

Consideramos el siguiente escenario de ejemplo:

- Nuestros desarrolladores escriben código localmente y comparten su trabajo con sus compañeros utilizando contenedores Docker.
- Usan Docker para llevar sus aplicaciones a un entorno de testing y ejecutar test automáticos y manuales.
- Cuando los desarrolladores encuentran errores, pueden solucionarlos en el entorno de desarrollo y volver a implementarlos en el entorno de testing para probarlos y validarlos.
- Cuando se completa el test, obtener la solución para el cliente es tan simple como enviar la imagen actualizada al entorno de producción.

Despliegue responsive y escalado

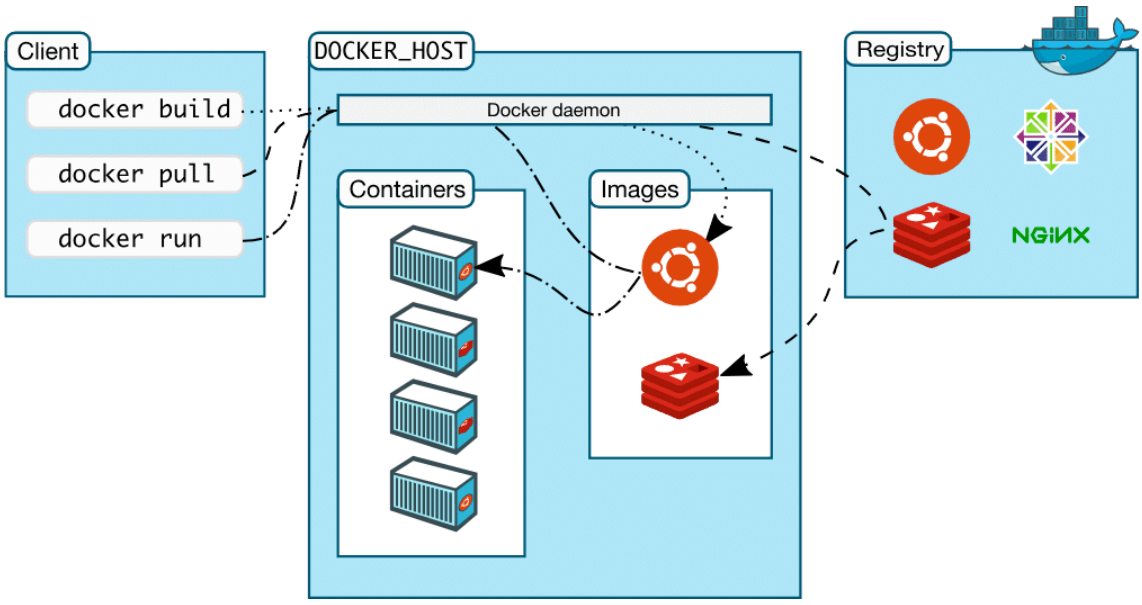
La plataforma basada en contenedores de Docker permite cargas de trabajo altamente portátiles.

Los contenedores Docker se pueden ejecutar en la computadora portátil local de un desarrollador, en máquinas físicas o virtuales en un centro de datos, en proveedores de la nube o en una combinación de entornos.

La portabilidad de Docker y su naturaleza ligera también facilitan la administración dinámica de cargas de trabajo, amplíando o eliminando aplicaciones y servicios según las necesidades del negocio, casi en tiempo real.

Arquitectura de Docker

Docker usa una arquitectura cliente-servidor. El cliente Docker habla con el daemon Docker, que hace el trabajo pesado de construir, ejecutar y distribuir sus contenedores Docker. El cliente Docker y el daemon pueden ejecutarse en el mismo sistema, o se puede conectar un cliente Docker a un daemon Docker remoto. El cliente Docker y el daemon se comunican utilizando una API REST, sobre sockets UNIX o una interfaz de red.



El Docker Daemon

El Docker daemon (dockerd) escucha las solicitudes de la API de Docker y gestiona los objetos de Docker, como imágenes, contenedores, redes y volúmenes. Un daemon también puede comunicarse con otros daemons para administrar los servicios de Docker.

El cliente de Docker

El cliente Docker (docker) es la forma principal en que muchos usuarios de Docker interactúan con Docker. Cuando se utilizan comandos como **docker run**, el cliente envía estos comandos a

dockerd, que los lleva a cabo. El comando **docker** usa la API Docker. El cliente de Docker puede comunicarse con más de un daemon.

Registros de Docker

Un registro de Docker almacena imágenes de Docker. Docker Hub y Docker Cloud son registros públicos que cualquiera puede usar, y Docker está configurado para buscar imágenes en Docker Hub de manera predeterminada. Incluso podemos ejecutar nuestro propio registro privado.

Cuando se utilizan los comandos **docker pull** o **docker run**, las imágenes necesarias se extraen del registro configurado. Cuando se utiliza el comando docker push, la imagen se envía a el registro configurado.

La tienda Docker permite comprar y vender imágenes de Docker o distribuirlas gratis. Por ejemplo, se puede comprar una imagen Docker que contenga una aplicación o servicio de un proveedor de software y usar la imagen para implementar la aplicación en nuestros entornos de prueba, preparación y producción. Podemos actualizar la aplicación tirando de la nueva versión de la imagen y volviendo a desplegar los contenedores.

Objetos Docker

Imágenes

Una imagen es una plantilla de 'solo lectura' con instrucciones para crear un contenedor Docker. A menudo, una imagen se basa en otra imagen, con alguna personalización adicional. Por ejemplo, se puede compilar una imagen basada en la imagen ubuntu, pero que instala el

servidor web Apache y nuestra aplicación, así como los detalles de configuración necesarios para ejecutar nuestra aplicación.

Contenedores

Un contenedor es una instancia ejecutable de una imagen. Se puede crear, iniciar, detener, mover o eliminar un contenedor utilizando la API Docker o el CLI. Se puede conectar un contenedor a una o más redes, adjuntarle almacenamiento o incluso crear una nueva imagen en función de su estado actual.

Por defecto, un contenedor está relativamente bien aislado de otros contenedores y su máquina host. Se puede controlar qué tan aislados están la red, el almacenamiento u otros subsistemas subyacentes de otros contenedores o de la máquina host.

Un contenedor se define por su imagen y por las opciones de configuración que se le proporciona cuando se crea o inicia. Cuando se elimina un contenedor, desaparecen los cambios a su estado que no se almacenan en el almacenamiento persistente.

Ejemplo comando docker run

El siguiente comando ejecuta un contenedor ubuntu, se conecta interactivamente a nuestra sesión de línea de comandos local y ejecuta / bin / bash.

```
$ docker run -i -t ubuntu /bin/bash
```

Cuando se ejecuta este comando, sucede lo siguiente (suponiendo que se esté utilizando la configuración de registro predeterminada):

- Si no se tiene la imagen ubuntu localmente, Docker la extrae de nuestro registro configurado, como si se hubiera ejecutado docker pull ubuntu manualmente.
- Docker crea un contenedor nuevo, como si se hubiera ejecutado un docker container create manualmente.

- Docker asigna un sistema de archivos de lectura y escritura al contenedor, como capa final. Esto permite que un contenedor en ejecución cree o modifique archivos y directorios en nuestro sistema de archivos local.
- Docker crea una interfaz de red para conectar el contenedor a la red predeterminada, ya que se no especificó ninguna opción de red. Esto incluye asignar una dirección IP al contenedor. De forma predeterminada, los contenedores se pueden conectar a redes externas utilizando la conexión de red de la máquina host.
- Docker inicia el contenedor y ejecuta / bin / bash. Debido a que el contenedor se está ejecutando de forma interactiva y conectado a nuestra terminal (debido a las opciones -i y -t), se puede proporcionar información usando nuestro teclado mientras la salida está registrada en nuestra terminal.
- Cuando escribe exit para finalizar el comando / bin / bash, el contenedor se detiene pero no se elimina. Puede volver a iniciarlo o eliminarlo.

Servicios

Los servicios permiten escalar contenedores a través de múltiples daemons Docker, que funcionan en conjunto como un swarm con múltiples managers y workers. Cada miembro de un swarm es un Docker daemon, y todos los daemons se comunican usando la API Docker. Un servicio permite definir el estado deseado, como el número de réplicas del servicio que debe estar disponible en un momento dado. De forma predeterminada, el servicio tiene equilibrio de carga en todos los nodos de trabajadores. Es posible que algunos términos no se comprendan por ahora pero se verán en posteriores temas.

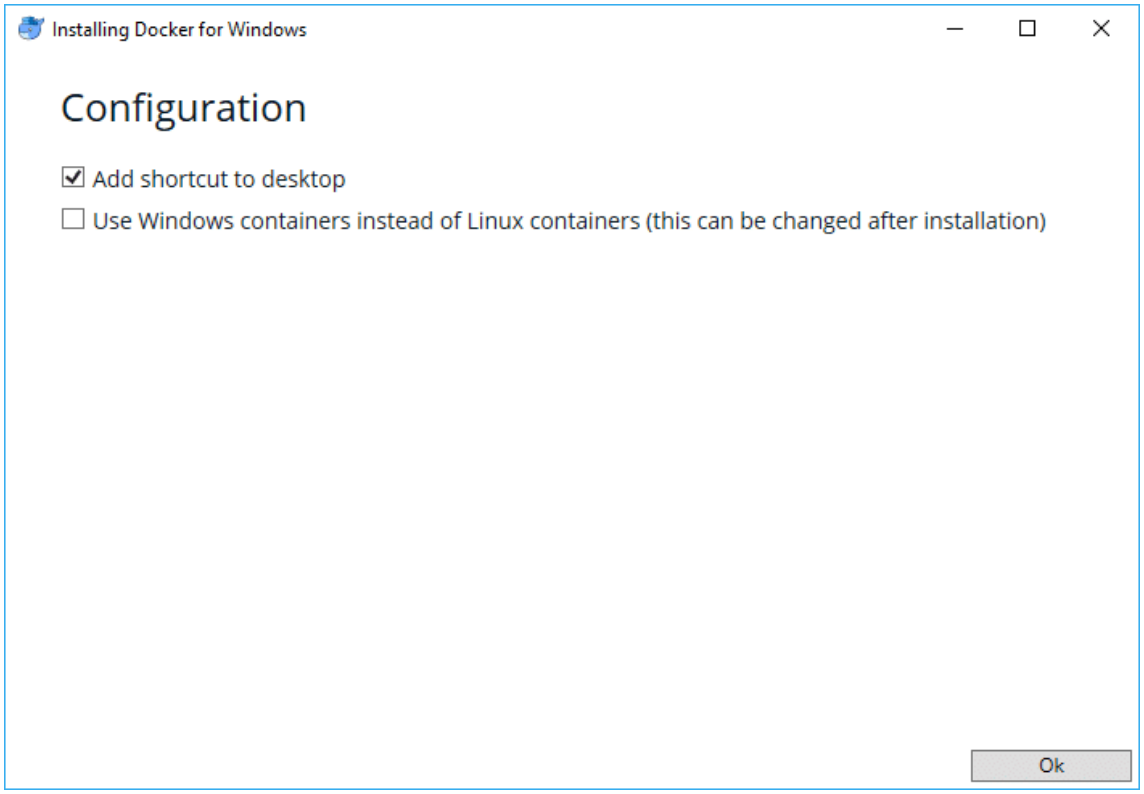
Instalación

Para cualquier instalación de Docker lo primero es registrarse en el **Hub de Docker**.

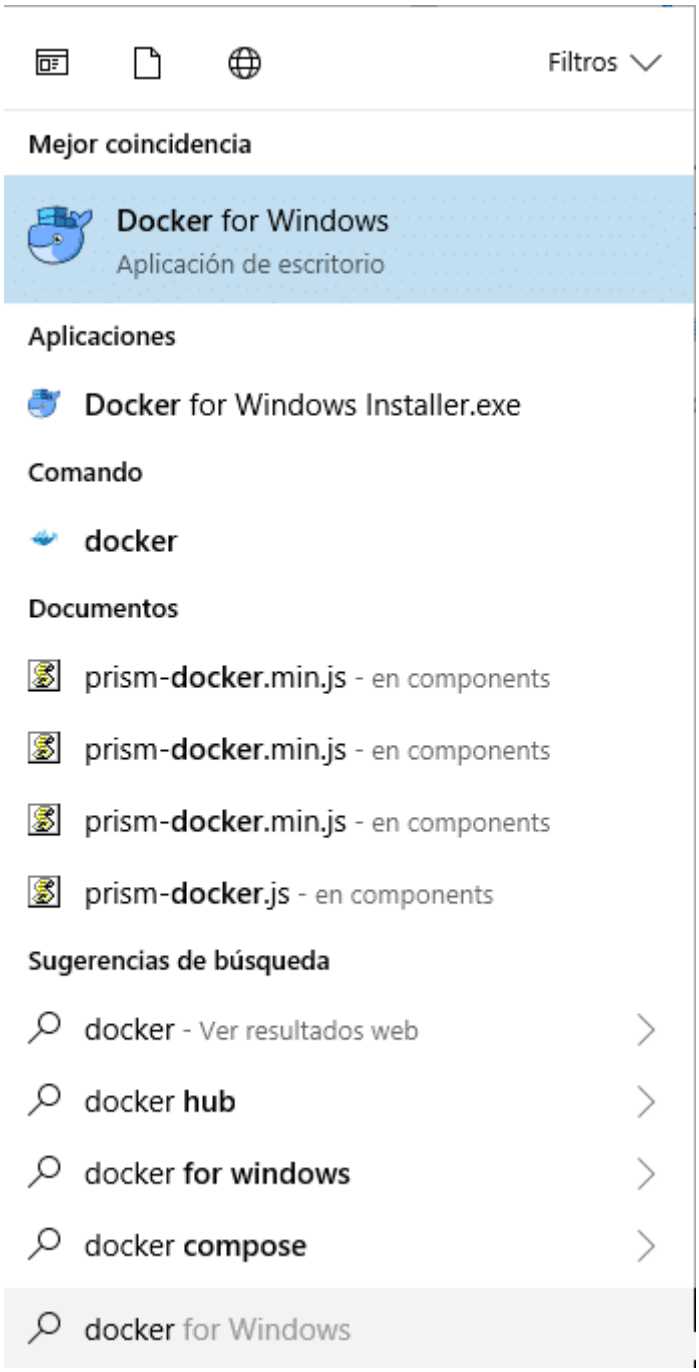
Cómo instalar Docker en Windows

Para instalar Docker en Windows hay que dirigirse a su **página oficial**.

Siga el asistente de instalación para aceptar la licencia, autorizar el instalador y continuar con la instalación. Reiniciaremos el sistema si nos lo pide. Es posible que Docker no se ejecute automáticamente después de su instalación.



Para iniciarlo, buscamos Docker, seleccionamos Docker for Windows en los resultados de búsqueda.



Cuando la ballena en la barra de estado se mantenga estable, Docker estará funcionando y se podrá acceder desde cualquier ventana de terminal.

Si la ballena está oculta en el área de Notificaciones, haríamos clic en la flecha hacia arriba en la barra de tareas para mostrarla.

Si acabamos de instalar la aplicación, también recibiremos un mensaje emergente con los próximos pasos sugeridos. Con esto ya tendríamos instalado Docker en Windows.

Cómo instalar Docker en Linux

Paso 1: Instalar Docker

Importante: Para este tutorial vamos a utilizar la imagen de Ubuntu Desktop, aunque para otras distribuciones el proceso de instalación resulta similar.

Es posible que la versión del paquete de instalación de Docker disponible en el repositorio oficial de Ubuntu no sea la más reciente. Para asegurarnos de contar con la versión más actualizada, instalaremos Docker desde el repositorio oficial de Docker.

Primero, actualizamos la lista de paquetes existentes:

$ sudo apt update

A continuación, añadimos algunos paquetes necesarios para que APT pueda usar paquetes a través de HTTPS:

$ sudo apt install apt-transport-https ca-certificates curl software-properties-common

Luego, añadimos la clave de GPG para el repositorio oficial de Docker en nuestro sistema:

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add

La salida debe ser la siguiente:

```
root@pablo:/home/pablo# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
root@pablo:/home/pablo#
```

Agregamos el repositorio de Docker a las fuentes de APT:

$ sudo add-apt-repository “deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable”

```
root@pablo:/home/pablo# sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
Hit:1 http://es.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://es.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:3 http://es.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu focal InRelease
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Reading package lists... Done
root@pablo:/home/pablo#
```

A continuación, actualizamos el paquete de base de datos con los paquetes de Docker del repositorio que acabamos de agregar:

$ sudo apt update

```
root@pablo:/home/pablo# sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
Hit:1 http://es.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://es.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:3 http://es.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu focal InRelease
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Reading package lists... Done
root@pablo:/home/pablo# sudo apt update
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:2 http://es.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 https://download.docker.com/linux/ubuntu focal InRelease
Hit:4 http://es.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:5 http://es.archive.ubuntu.com/ubuntu focal-backports InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
root@pablo:/home/pablo#
```

Comprobamos la versión que vamos a instalar y como podemos observar el candidato para la instalación es la versión 20.10.5:

$ apt-cache policy docker-ce

```
root@pablo:/home/pablo# apt-cache policy docker-ce
docker-ce:
  Installed: (none)
  Candidate: 5:20.10.5~3-0~ubuntu-focal
  Version table:
     5:20.10.5~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:20.10.4~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:20.10.3~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:20.10.2~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:20.10.1~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:20.10.0~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:19.03.15~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:19.03.14~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:19.03.13~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:19.03.12~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:19.03.11~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:19.03.10~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
     5:19.03.9~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
root@pablo:/home/pablo#
```

Por último, instalamos Docker:

$ sudo apt install docker-ce

Comprobamos que la versión que hemos implantado efectivamente coincide con la versión candidata.

$ docker –version

```
root@pablo:/home/pablo# docker --version
Docker version 20.10.5, build 55c4c88
root@pablo:/home/pablo#
```

Verificamos que el servicio está corriendo correctamente.

$ sudo systemctl status docker

```
root@pablo:/home/pablo# sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2021-03-22 12:23:04 CET; 3min 11s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 6264 (dockerd)
      Tasks: 16
     Memory: 62.1M
     CGroup: /system.slice/docker.service
             └─6264 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

mar 22 12:23:02 pablo dockerd[6264]: time="2021-03-22T12:23:02.967634757+01:00" level=warni
mar 22 12:23:02 pablo dockerd[6264]: time="2021-03-22T12:23:02.967653119+01:00" level=warni
mar 22 12:23:02 pablo dockerd[6264]: time="2021-03-22T12:23:02.967657919+01:00" level=warni
mar 22 12:23:02 pablo dockerd[6264]: time="2021-03-22T12:23:02.967768798+01:00" level=info
mar 22 12:23:03 pablo dockerd[6264]: time="2021-03-22T12:23:03.604765139+01:00" level=info
mar 22 12:23:04 pablo dockerd[6264]: time="2021-03-22T12:23:04.071856974+01:00" level=info
mar 22 12:23:04 pablo dockerd[6264]: time="2021-03-22T12:23:04.128629286+01:00" level=info
mar 22 12:23:04 pablo dockerd[6264]: time="2021-03-22T12:23:04.128948579+01:00" level=info
mar 22 12:23:04 pablo systemd[1]: Started Docker Application Container Engine.
mar 22 12:23:04 pablo dockerd[6264]: time="2021-03-22T12:23:04.196856434+01:00" level=info
lines 1-21/21 (END)
```

Si la salida del comando devuelve **active (running)** el proceso está funcionando correctamente.

Paso 2 : Ejecutar el comando Docker sin sudo (opcional)

Por defecto, el comando docker solo puede ser ejecutado por el usuario root o un usuario del grupo docker, que se crea automáticamente durante el proceso de instalación de Docker. Si intentamos ejecutar el comando docker sin sudo como prefijo o sin formar parte del grupo docker, obtendremos un resultado como este:

```
pablo@pablo:~$ docker run hello-world
docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.24/containers/create: dial unix /var/run/docker
.sock: connect: permission denied.
See 'docker run --help'.
pablo@pablo:~$
```

Si queremos evitar escribir sudo al ejecutar el comando docker, debemos agregar nuestro nombre de usuario al grupo docker:

$ sudo usermod -aG docker ${USER}

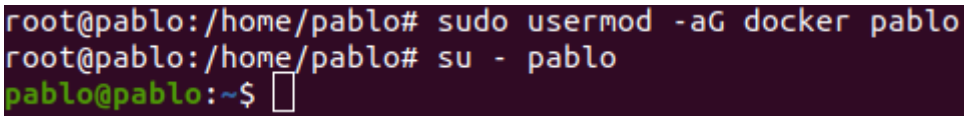
Cambiamos **${USER}** por nuestro nombre de usuario, en mi caso es pablo y quedaría del siguiente modo:

$ sudo usermod -aG docker pablo

```
root@pablo:/home/pablo# sudo usermod -aG docker pablo
root@pablo:/home/pablo#
```

Para aplicar los cambios sin tener que reiniciar el equipo debemos escribir el siguiente comando, al igual que antes cambiamos **${USER}** por nuestro nombre de usuario.

$ su – pablo



Paso 3: Usar el comando docker

El uso de docker consiste en pasar a este una cadena de opciones y comandos seguida de argumentos. La sintaxis adopta esta forma:

$ docker [opciones] [comando] [argumento]

Para ver todos los subcomandos disponibles, escribimos lo siguiente:

$ docker

Si deseamos ver las opciones disponibles para un comando específico, escribimos lo siguiente:

$ docker subcomando-docker –help

Para ver información sobre Docker relacionada con todo el sistema, utilizamos el siguiente comando:

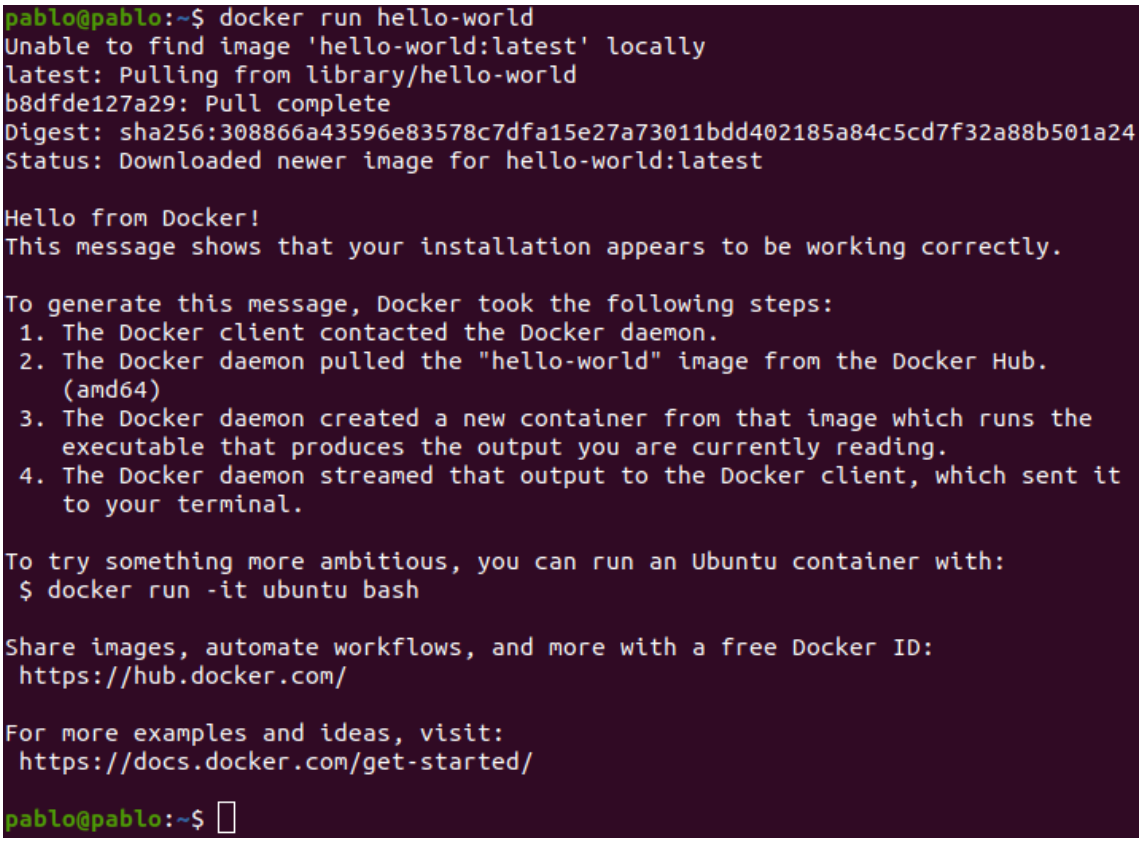
$ docker info

Paso 4: Trabajar con imágenes de Docker

Los contenedores de Docker se construyen con imágenes de Docker. Por defecto, Docker obtiene estas imágenes de Docker Hub, un registro de Docker gestionado por Docker, la empresa responsable del proyecto Docker. Cualquiera puede alojar sus imágenes en Docker Hub, de modo que la mayoría de las aplicaciones y las distribuciones de Linux que necesitaremos tendrán imágenes alojadas allí.

Para verificar si puede acceder a imágenes y descargarlas de Docker Hub, escribimos lo siguiente:

$ docker run hello-world



Si la salida que nos devuelve coincide con la plasmada en la imagen anterior, entonces podemos descargar correctamente imágenes de Docker Hub.

Inicialmente, Docker no pudo encontrar la imagen de hello-world a nivel local. Por ello la descargó de Docker Hub, el repositorio predeterminado. Una vez que se descargó la imagen, Docker creó un contenedor a partir de ella y de la aplicación dentro del contenedor ejecutado, y mostró el mensaje.

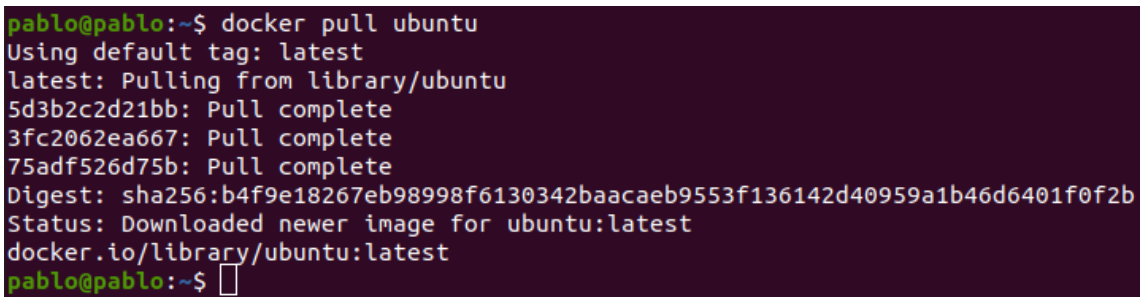
Podemos buscar imágenes disponibles en Docker Hub usando el comando docker con el subcomando search. Por ejemplo, para buscar la imagen de Ubuntu, escribimos lo siguiente:

$ docker search ubuntu

El comando que acabamos de ejecutar buscará en los repositorios de Docker Hub en busca de imágenes que coincidan con el criterio de búsqueda y devuelve los posibles candidatos.

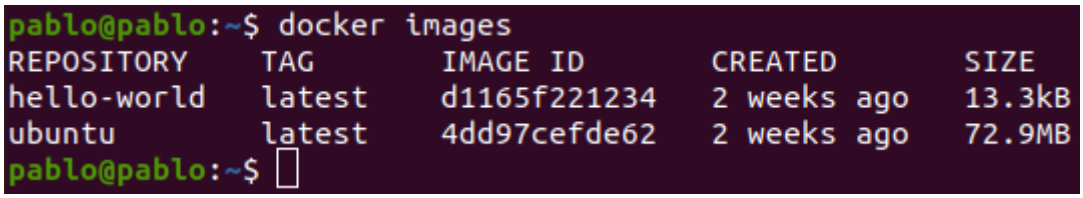
En la columna de **OFFICIAL, OK** indica que la imagen creada está avalada por la empresa responsable del proyecto. Una vez que identifique la imagen que desearía usar, podemos descargarla utilizando el subcomando docker pull.

$ docker pull ubuntu



Para ver las imágenes que tenemos disponibles localmente utilizamos el comando:

$ docker images



Paso 5: Ejecutar un contenedor de Docker

El contenedor hello-world que creamos en el paso anterior es un ejemplo de un contenedor que se ejecuta y se cierra tras emitir un mensaje de prueba. Los contenedores pueden ofrecer una utilidad mucho mayor y ser interactivos

Como ejemplo, ejecutemos un contenedor usando la imagen más reciente de Ubuntu. La combinación de los conmutadores **-i** y **-t** le proporcionan un acceso interactivo del shell al contenedor:

$ docker run -it ubuntu

Nuestros símbolos del sistema deben cambiar para reflejar el hecho de que ahora estamos trabajando dentro del contenedor.

```
pablo@pablo:~$ docker run -it ubuntu
root@cf9cc8313e87:/#
```

Ahora podemos ejecutar cualquier comando dentro del contenedor. Por ejemplo, actualizar los repositorios. No es necesario prefijar ningún comando con sudo, ya que todas las operaciones dentro del contenedor se ejecutan por defecto con el usuario **root**:

$ apt update

```
pablo@pablo:~$ docker run -it ubuntu
root@cf9cc8313e87:/# apt update
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [109 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [689 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [21.6 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [187 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [681 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1093 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [220 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [940 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [29.6 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [4305 B]
Fetched 17.3 MB in 2s (9902 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@cf9cc8313e87:/#
```

Importante: Cuando trabajamos con contenedores debemos tener en cuenta un detalle y es que las imágenes que implementamos en muchas ocasiones pueden no tener instaladas algunas aplicaciones que por defecto si vienen instaladas en otras imágenes. Un claro ejemplo es que el contenedor de Ubuntu que estamos utilizando no tiene el editor de texto nano.

$ nano ficheroprueba

```
root@cf9cc8313e87:/# nano ficheroprueba
bash: nano: command not found
root@cf9cc8313e87:/#
```

El motivo por el que no está instalado por defecto este editor es simple, las imágenes de Docker buscan ser lo más ligeras y modulares posibles para ofrecer servicios de una manera sencilla y rápida. Por lo que

instalar programas innecesarios que harían que el tamaño de la máquina fuese mayor va en contra del propósito por el que fue creada esta tecnología.

Para instalar nano ejecutamos el comando apt seguido del programa que deseamos instalar.

$ apt install nano

```
root@cf9cc8313e87:/# apt install nano
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  hunspell
The following NEW packages will be installed:
  nano
0 upgraded, 1 newly installed, 0 to remove and 3 not upgraded.
Need to get 269 kB of archives.
After this operation, 868 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 nano amd64 4.8-1ubuntu1 [269 kB]
Fetched 269 kB in 0s (712 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package nano.
(Reading database ... 4121 files and directories currently installed.)
Preparing to unpack .../nano_4.8-1ubuntu1_amd64.deb ...
Unpacking nano (4.8-1ubuntu1) ...
Setting up nano (4.8-1ubuntu1) ...
update-alternatives: using /bin/nano to provide /usr/bin/editor (editor) in auto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/editor.1.gz because associated file /usr/share/man/man1/nano.1.gz (of link group editor) doesn't exist
update-alternatives: using /bin/nano to provide /usr/bin/pico (pico) in auto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/pico.1.gz because associated file /usr/share/man/man1/nano.1.gz (of link group pico) doesn't exist
root@cf9cc8313e87:/#
```

Comprobamos que versión hemos instalado.

$ nano –version

```
root@cf9cc8313e87:/# nano --version
 GNU nano, version 4.8
 (C) 1999-2011, 2013-2020 Free Software Foundation, Inc.
 (C) 2014-2020 the contributors to nano
 Email: nano@nano-editor.org    Web: https://nano-editor.org/
 Compiled options: --disable-libmagic --enable-utf8
root@cf9cc8313e87:/#
```

Paso 6: Administrar contenedores de Docker

Para mostrar los contenedores activos utilizamos el comando:

$ docker ps

```
pablo@pablo:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
pablo@pablo:~$
```

El comando no ha devuelto ningún contenedor ¿por qué? Esto se debe a que si no añadimos la opción **-d** al comando docker run, cuando salimos del contenedor estos se paran.

Para comprobar los contenedores activos e inactivos utilizamos el comando:

$ docker ps -a

```
pablo@pablo:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND       CREATED             STATUS                      PORTS     NAMES
cf9cc8313e87   ubuntu        "/bin/bash"   39 minutes ago      Exited (127) 5 minutes ago            nostalgic_grothendieck
894dded0ed7e   hello-world   "/hello"      About an hour ago   Exited (0) About an hour ago          naughty_hopper
pablo@pablo:~$
```

Para ver el último contenedor que ha sido creado utilizamos la opción -l:

$ docker ps -l

```
pablo@pablo:~$ docker ps -l
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                      PORTS     NAMES
cf9cc8313e87   ubuntu    "/bin/bash"   42 minutes ago   Exited (127) 8 minutes ago            nostalgic_grothendieck
pablo@pablo:~$
```

Para iniciar un contenedor detenido, utilizamos docker start, seguido del nombre del contenedor o el ID. Vamos a iniciar el contenedor basado en Ubuntu con el ID cf9cc8313e87:

$ docker start cf9cc8313e87

```
pablo@pablo:~$ docker start cf9cc8313e87
cf9cc8313e87
pablo@pablo:~$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS         PORTS     NAMES
cf9cc8313e87   ubuntu    "/bin/bash"   45 minutes ago   Up 7 seconds             nostalgic_grothendieck
pablo@pablo:~$
```

Para detener un contenedor, utilizamos docker stop, seguido del ID o nombre del contenedor. Esta vez usaremos el nombre que Docker asignó al contenedor, que es nostalgic_grothendieck:

$ docker stop nostalgic_grothendieck

```
pablo@pablo:~$ docker stop nostalgic_grothendieck
nostalgic_grothendieck
pablo@pablo:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND       CREATED          STATUS                      PORTS     NAMES
cf9cc8313e87   ubuntu        "/bin/bash"   47 minutes ago   Exited (0) 10 seconds ago             nostalgic_grothendieck
894dded0ed7e   hello-world   "/hello"      2 hours ago      Exited (0) 2 hours ago                naughty_hopper
pablo@pablo:~$
```

Una vez que decidamos que ya no necesitamos un contenedor, lo eliminamos con el comando docker rm y el nombre o ID. Utilizamos el comando docker ps -a para visualizar el ID o nombre del contenedor asociado con su imagen y eliminarlo.

$ docker rm nostalgic_grothendieck

```
pablo@pablo:~$ docker rm nostalgic_grothendieck
nostalgic_grothendieck
pablo@pablo:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED       STATUS                     PORTS     NAMES
894dded0ed7e   hello-world   "/hello"   2 hours ago   Exited (0) 2 hours ago             naughty_hopper
pablo@pablo:~$
```

Del mismo modo que hemos borrado el contenedor una vez detenido, también podemos eliminarlo mientras está aún corriendo con la opción **-f**.

$ docker rm -f nombre_contenedor

Podemos iniciar un nuevo contenedor y darle un nombre usando el conmutador **–name**. También podemos usar el conmutador **–rm** para crear un contenedor que se elimine de forma automática cuando se detenga. Consulte el comando docker run help para obtener más información sobre estas y otras opciones.

Paso 7: Creación de una imagen personalizada a partir de un contenedor

Los contenedores pueden convertirse en imágenes que podremos usar para crear contenedores nuevos. Veamos cómo funciona esto.

Después de instalar el editor de texto nano dentro del contenedor de Ubuntu, dispondremos de un contenedor que se ejecuta a partir de una imagen, pero este es diferente de la imagen que utilizamos para crearlo. Sin embargo, quizás deseemos reutilizar este contenedor que ya tiene instalado nano como base de nuevas imágenes posteriormente.

$ docker commit -m «Escribimos una breve descripción del contenedor» -a «Nombre del autor» id_contenedor repositorio/nombre_nueva_imagen

El conmutador **-m** es el mensaje que permite a otros saber qué cambios realizamos, mientras que **-a** se utiliza para especificar el autor. El container_id es el que observamos anteriormente en el tutorial cuando iniciamos la sesión interactiva de Docker. A menos que hayamos creado repositorios adicionales en Docker Hub, el campo repositorio generalmente corresponde a nuestro nombre de usuario de Docker Hub.

Partiendo del siguiente caso vamos a crear una imagen personalizada de ubuntu con el editor de texto preinstalado por nosotros.

```
root@bea6a695cbc3:/# nano --version
 GNU nano, version 4.8
 (C) 1999-2011, 2013-2020 Free Software Foundation, Inc.
 (C) 2014-2020 the contributors to nano
 Email: nano@nano-editor.org    Web: https://nano-editor.org/
 Compiled options: --disable-libmagic --enable-utf8
root@bea6a695cbc3:/#
```

Copiamos el ID del contenedor.

Con el comando exit salimos del contenedor y adaptamos el comando anterior a nuestro propio caso.

$ docker commit -m «Ubuntu con editor de texto» -a «Pablo» bea6a695cbc3/ubuntu_con_nano

```
root@pablo:/home/pablo# docker commit -m "Ubuntu con editor de texto" -a "Pablo" 8ced725285df pjvdockerhub/ubuntu_con_nano
sha256:85a5650e8f68127cd0727ceb846c99c6eb652cc25030b94d718871b88a52822c
```

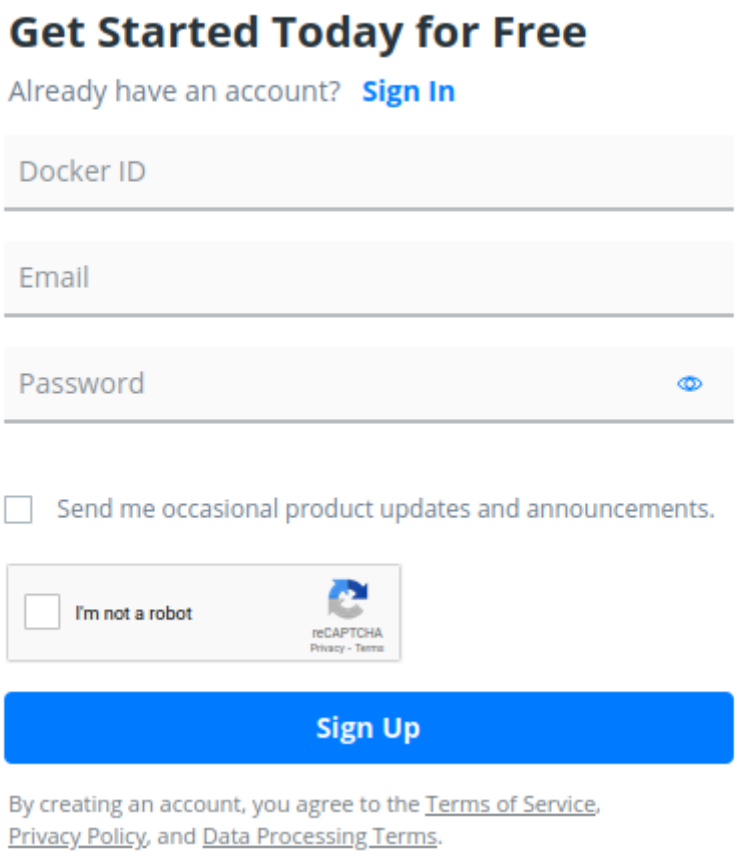
Comprobamos que se ha guardado la imagen localmente con el nombre pjvdockerhub/ubuntu_con_nano

$ docker images

```
root@pablo:/home/pablo# docker images
REPOSITORY                      TAG       IMAGE ID       CREATED          SIZE
pjvdockerhub/ubuntu_con_nano    latest    85a5650e8f68   2 minutes ago    101MB
hello-world                     latest    d1165f221234   2 weeks ago      13.3kB
ubuntu                          latest    4dd97cefde62   2 weeks ago      72.9MB
root@pablo:/home/pablo#
```

Paso 8: Guardar imágenes de Docker en un repositorio de Docker

En primer lugar, nos registramos en Docker Hub accediendo a su página https://hub.docker.com/.



A continuación, iniciamos sesión en Docker Hub con el siguiente comando.

Importante: Se le solicitará autenticarse usando su contraseña de Docker Hub.

$ docker login -u nombre_de_usuario

```
root@pablo:/home/pablo# docker login -u pjvdockerhub
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
root@pablo:/home/pablo#
```

Si nuestro nombre de usuario de registro de Docker es diferente del nombre de usuario local que usamos para crear la imagen, debemos etiquetar la imagen con el nombre de usuario de registro.

Para el ejemplo que se muestra en el último paso, deberemos escribir lo siguiente:

$ docker tag pjvdockerhub/ubuntu_con_nano pjvdockerhub/ubuntu_con_nano

```
root@pablo:/home/pablo# docker tag pjvdockerhub/ubuntu_con_nano pjvdockerhub/ubuntu_con_nano
root@pablo:/home/pablo#
```

Para guardar la imagen de nuestro contenedor en nuestro repositorio de Docker Hub utilizamos el comando.

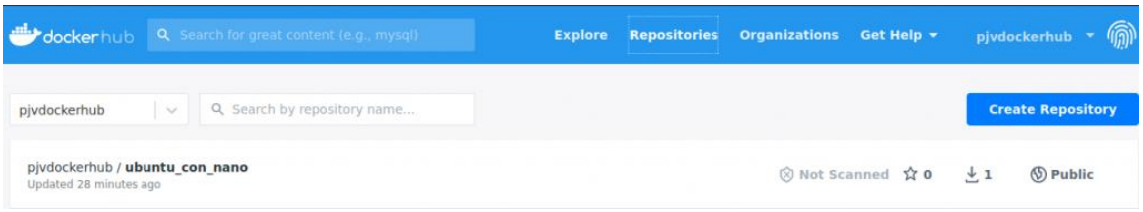
$ docker push nombre_de_usuario/nombre_imagen

Siguiendo el esquema anterior mi comando quedaría así.

$ docker push pjvdockerhub/ubuntu_con_nano

```
root@pablo:/home/pablo# docker push pjvdockerhub/ubuntu_con_nano
Using default tag: latest
The push refers to repository [docker.io/pjvdockerhub/ubuntu_con_nano]
3d07a7e104c8: Pushed
c20d459170d8: Mounted from library/ubuntu
db978cae6a05: Mounted from library/ubuntu
aeb3f02e9374: Mounted from library/ubuntu
latest: digest: sha256:7bf1cdaf0c905d6873b0327035665dd32b31aea3289e13e546742ee1b79a5f19 size: 1155
root@pablo:/home/pablo#
```

Iniciamos sesión en nuestro repositorio de docker y comprobamos que ya tenemos disponible nuestra imagen.



Para utilizar esta imagen en cualquier equipo primero iniciamos sesión en docker con el comando docker login -u nombre_usuario y luego escribimos el siguiente comando:

$ docker pull nombre_de_usuario/imagen

En mi caso el comando queda del siguiente modo:

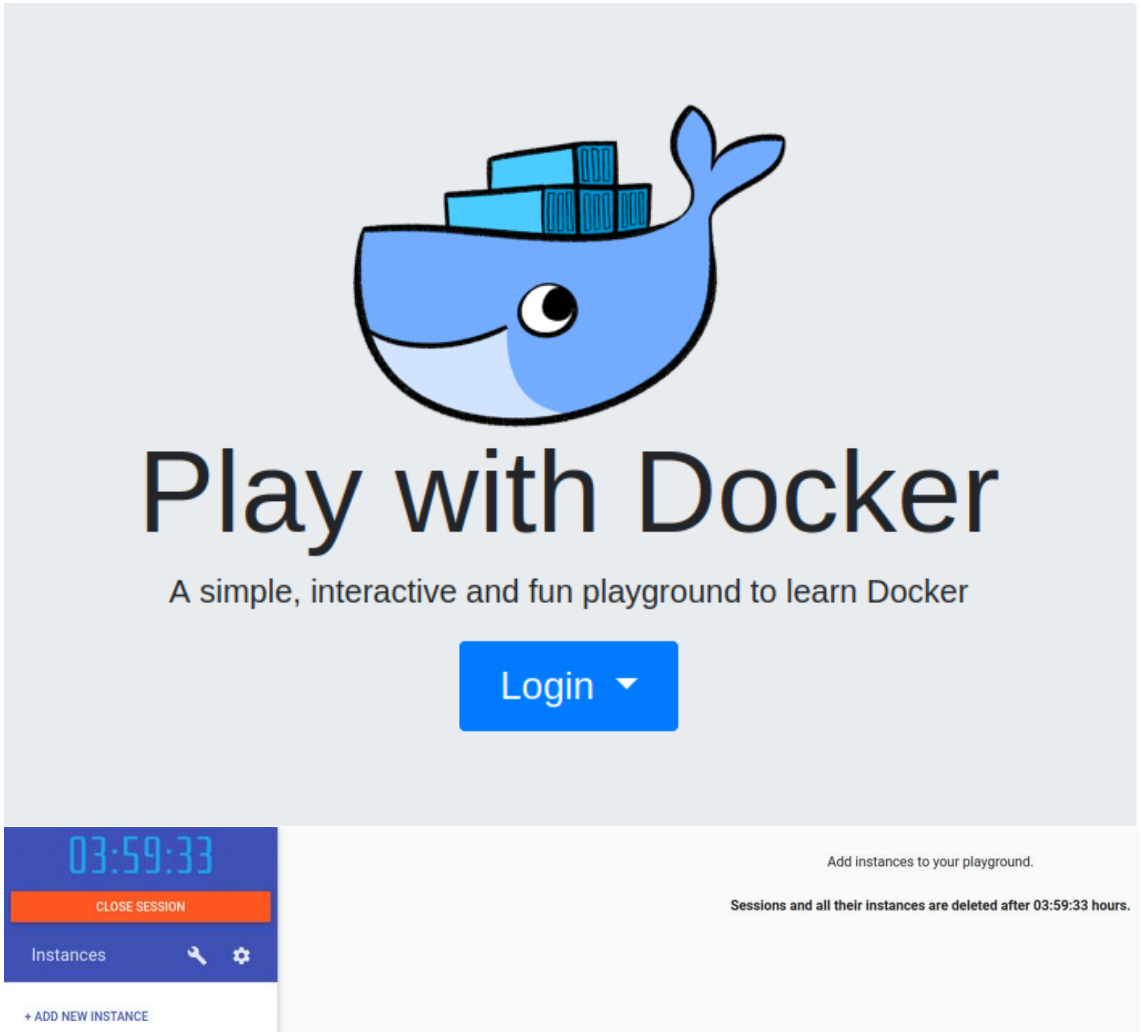
$ docker pull pjvdockerhub/ubuntu_con_nano

```
root@pablo:/home/pablo# docker pull pjvdockerhub/ubuntu_con_nano
Using default tag: latest
latest: Pulling from pjvdockerhub/ubuntu_con_nano
Digest: sha256:7bf1cdaf0c905d6873b0327035665dd32b31aea3289e13e546742ee1b79a5f19
Status: Image is up to date for pjvdockerhub/ubuntu_con_nano:latest
docker.io/pjvdockerhub/ubuntu_con_nano:latest
root@pablo:/home/pablo#
```

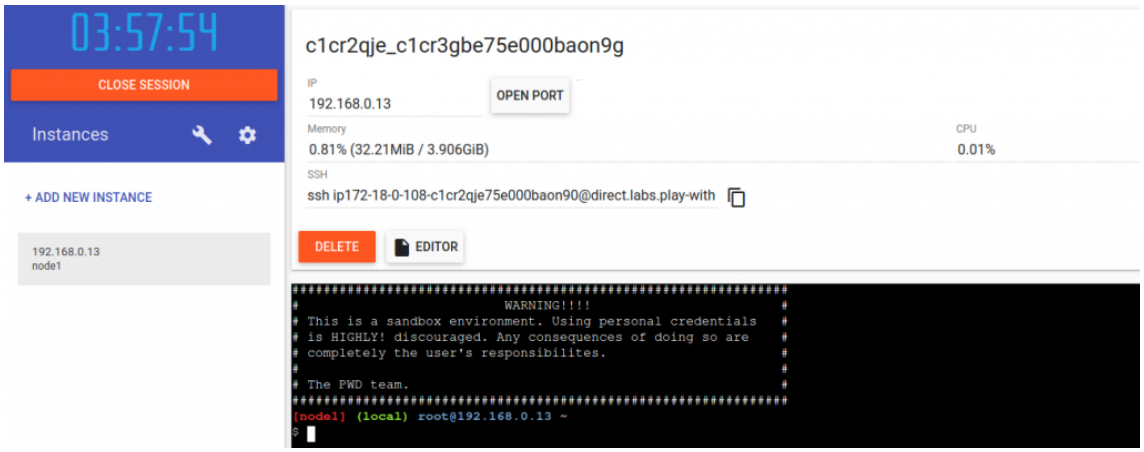
Como podemos observar recibimos un mensaje que dice que la imagen que hemos descargado, ya la tenemos disponible en su última versión.

Paso 9: Practica con Docker Playground

Podemos practicar gratuitamente Docker desde la siguiente página https://labs.play-with-docker.com/



Hacemos clic sobre Añadir nueva instancia y ya podemos comenzar a practicar con Docker.



Comprobamos la versión de Docker que viene instalada.

```
[node1] (local) root@192.168.0.13 ~
$ docker --version
Docker version 20.10.0, build 7287ab3
[node1] (local) root@192.168.0.13 ~
$
```

Ejercicio: Utiliza Docker Playground para instalar un servidor web Apache.

Solución:

En primer lugar, buscamos una imagen de apache en los repositorios de Docker Hub.

$ docker search apache

```
[node1] (local) root@192.168.0.13 ~
$ docker search apache
NAME                                DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
httpd                               The Apache HTTP Server Project                  3417      [OK]
tomcat                              Apache Tomcat is an open source implementati…   2977      [OK]
cassandra                           Apache Cassandra is an open-source distribut…   1251      [OK]
maven                               Apache Maven is a software project managemen…   1173      [OK]
solr                                Solr is the popular, blazing-fast, open sour…   819       [OK]
apache/airflow                      Apache Airflow                                  224
apache/nifi                         Unofficial convenience binaries and Docker i…   205                 [OK]
apache/zeppelin                     Apache Zeppelin                                 144                 [OK]
eboraas/apache-php                  PHP on Apache (with SSL/TLS support), built …   144                 [OK]
eboraas/apache                      Apache (with SSL/TLS support), built on Debi…   92                  [OK]
apacheignite/ignite                 Apache Ignite - Distributed Database            76                  [OK]
nimmis/apache-php5                  This is docker images of Ubuntu 14.04 LTS wi…   69                  [OK]
bitnami/apache                      Bitnami Apache Docker Image                     67                  [OK]
apache/skywalking-oap-server        Apache SkyWalking OAP Server                    55
apachepulsar/pulsar                 Apache Pulsar - Distributed pub/sub messagin…   34
linuxserver/apache                  An Apache container, brought to you by Linux…   27
antage/apache2-php5                 Docker image for running Apache 2.x with PHP…   23                  [OK]
apache/nutch                        Apache Nutch                                    23                  [OK]
webdevops/apache                    Apache container                                15                  [OK]
newdeveloper/apache-php             apache-php7.2                                   8
newdeveloper/apache-php-composer    apache-php-composer                             7
lephare/apache                      Apache container                                6                   [OK]
secoresearch/apache-varnish         Apache+PHP+Varnish5.0                           2                   [OK]
apache/arrow-dev                    Apache Arrow convenience images for developm…   1
jelastic/apachephp                  An image of the Apache PHP application serve…   0
[node1] (local) root@192.168.0.13 ~
$
```

Descargamos la imagen oficial de apache httpd.

$ docker pull httpd

```
[node1] (local) root@192.168.0.13 ~
$ docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
6f28985ad184: Pull complete
3a141a09d1d0: Pull complete
1633384edb75: Pull complete
acb3e3b931b8: Pull complete
f6dc6b8b1d70: Pull complete
Digest: sha256:9625118824bc2514d4301b387c091fe802dd9e08da7dd9f44d93ee65497e7c1c
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
[node1] (local) root@192.168.0.13 ~
$
```

Creamos el contenedor de apache.

$ docker run -dit –name servidor_web -p 80:80 httpd

Parámetros docker run:

-d = permite ejecutar el contenedor en segundo plano

-i = permite que el contenedor sea interactivo

–**t** = proporciona al contenedor una terminal

–-**name** = nombre del contenedor

-p = permite mapear puertos desde el interior del contenedor 80:**80** al exterior del contenedor **80**:80. El puerto que utilizamos ha sido el 80 porque es el puerto por defecto que utiliza apache para conexiones no seguras.

```
[node1] (local) root@192.168.0.13 ~
$ docker run -dit  --name servidor_web -p 80:80 httpd
2aeddc00b71d9b3e1abb0345f10c440d5f8550fc47e4558417c9835de7d8421d
[node1] (local) root@192.168.0.13 ~
$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS                NAMES
2aeddc00b71d   httpd     "httpd-foreground"   9 seconds ago   Up 7 seconds   0.0.0.0:80->80/tcp   servidor_web
[node1] (local) root@192.168.0.13 ~
$
```

Comprobamos que el servidor apache está funcionando correctamente, para ello hacemos clic sobre el puerto mapeado.

Como podemos observar el servidor de apache nos sirve una página que muestra un mensaje de éxito.

Conclusiones

Docker es un entorno que nos permite el despliegue rápido y sencillo de contenedores, estos no son ni más ni menos que máquinas virtuales livianas y modulares con las que podemos crear una aplicación en cuestión de segundos. Esta tecnología tiene una gran comunidad open source que la respalda y sube gran cantidad de documentación sobre el tema. Además, con Docker Hub podemos tener nuestras imágenes

preferidas subidas a nuestro repositorio personal y utilizarlas donde quiera que estemos.