Empezando con docker

Docker está de moda, pero ¿qué es docker?

**¿Qué diferencia hay entre docker y Vagrant?**

## ¿Qué es vagrant?

Lo primero que vamos a ver **¿Qué es vagrant?.** Vagrant es una **herramienta** que nos ayuda a **crear y manejar máquinas virtuales** con un mismo entorno de trabajo. Nos permite definir los servicios a instalar así como también sus configuraciones.

A primera vista **Docker vs Vagrant** parecen lo mismo y sirven para algo similar sin embargo la tecnología y el método que utilizan para lograr sus objetivos son bastante diferentes, mientras Docker trabaja con **Contenedores**.

[Vagrant](https://www.vagrantup.com/) en cambio trabaja con **máquinas virtuales**, cada uno tiene sus ventajas y desventajas, para ciertos casos ambos pueden servirnos indistintamente aunque en otras ocasiones puede servirnos uno más que el otro dependiendo que queramos hacer.

Por eso hoy primero vamos a ver que es una Máquina Virtual y que es un Contenedor, para luego adentrarnos en el versus [Docker](https://www.docker.com/) versus Vagrant, explorando sus diferencias y similitudes.

## ¿QUE ES UNA MAQUINA VIRTUAL?

Una Máquina Virtual es un **software que permite emular el hardware de una computadora** para *instalar allí un sistema operativo* y aplicaciones aislándolo por completo del hardware real de la misma forma que lo hacen los emuladores de video juegos, se puede decir que las Máquinas Virtuales actuales con la evolución natural de los emuladores.

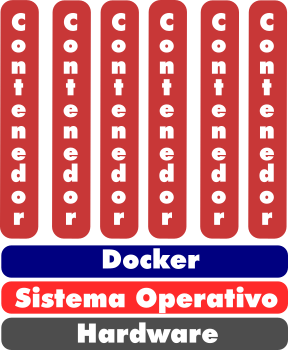
Las Máquinas Virtuales se pueden usar en un servidor o en una computadora de escritorio o una computadora portátil donde además del sistema operativo principal podemos tener una o varias Máquinas Virtuales con diferentes sistemas operativos que pueden ofrecer una variedad servicios o aplicaciones que necesitemos.



Por otro lado también porque debido al tamaño de un Sistema Operativo moderno y sus requisitos, el uso de VM (actuales) **implican un gran consumo de recursos**, de memoria, de CPU y de disco que antes no eran posibles.

## ¿QUE ES UN CONTENEDOR?

Un contenedor puede definirse como un paquete cerrado que dentro contiene todo lo necesario para que una aplicación se ejecute, todo está encapsulado dentro de una sola imagen completamente independiente el servidor o computadora que lo ejecuta, en el contenedor se incluyen tanto los binarios como todos los archivos de configuración y demás ficheros que necesite para ejecutar la aplicación.



Los **contenedores** a diferencia de las máquinas virtuales **no reservan memoria ni limitan el uso del CPU**, las aplicaciones dentro de los contenedores se ejecutan

La **idea** detrás de un contenedor de esto es que sea **ligeros** y **portables**, que se pueda transferir entre diferentes entornos sin contratiempos ya que el funcionamiento interno es completamente **independiente del sistema operativo** que lo aloja.

Con un contenedor nos aseguramos que la aplicación es completamente portable, como es independiente del hardware y del sistema operativo, transferir la imagen entre dos plataformas no representa ningún problema ya que se va a ejecutar siempre de la misma forma.

## CONTENEDORES VS MAQUINAS VIRTUALES

Las máquinas virtuales y los contenedores tienen diferencias sustanciales en cuando a velocidad, en el caso de los contenedores la creación y arranque solo toma segundos, ya que solo carga una aplicación, mientras que en el caso de las máquinas virtuales la creación puede llevar incluso mucho más tiempo, desde minutos hasta horas dependiendo de la complejidad, además el arranque de una máquina virtual puede llevar cinco minutos o incluso más mientras se inicia el sistema operativo y todos sus servicios.

En el uso de CPU los contenedores son más eficientes ya que solo se ejecuta la aplicación mientras que la máquina virtual ejecuta un sistema operativo completo aparte de la aplicación lo que implica un mayor uso del procesador.

El uso de memoria también es mayor en las máquinas virtuales, sin embargo la ventaja de las máquinas virtuales es que la memoria que le asignemos estará garantizada mientras que en los contenedores se asigna solo la que se necesita, esto tiene la ventaja que permitirá que un servidor ejecutar más contenedores que máquinas virtuales sin embargo es posible que cuando un contenedor necesite más memoria resulte que ya no quede RAM disponible.

**Características de Docker**

**Docker** es un proyecto **open source** que ha revolucionado la manera de desarrollar software gracias a la sencillez con la que permite gestionar contenedores. Los contenedores LXC (LinuX Containers) son un concepto relativamente antiguo y utilizado desde hace tiempo por grandes empresas como Amazon o Google, pero cuya gestión era complicada. Sin embargo, Docker define APIs y herramientas de línea de comandos que hacen casi trivial la creación, distribución y ejecución de contenedores. De ahí que el lema de Docker sea: “**Build, Ship and Run. Any application, Anywhere**” (**construir, enviar y ejecutar cualquier aplicación, en cualquier lugar**)

y se haya convertido en una herramienta fundamental tanto para desarrolladores como para administradores de sistemas.

Podríamos definir un contenedor como un **proceso aislado** del resto de los procesos de la máquina gracias a correr sobre **su propio sistema de ficheros**, su propio espacio de usuarios y procesos, sus propias interfaces de red… es por ello que a veces se dice que un contenedor es una máquina virtual ligera.

Las características principales de los contenedores son su portabilidad, su inmutabilidad y su ligereza:

### Portabilidad

Un contenedor es ejecutado por lo que se denomina el Docker Engine, un demonio que es fácilmente instalable en todas las distribuciones Linux y también en Windows. Un contenedor ejecuta una imagen de docker, que es una representación del sistema de ficheros y otros metadatos que el contenedor va a utilizar para su ejecución. Una vez que hemos generado una imagen de Docker, ya sea en nuestro ordenador o vía una herramienta externa, esta imagen podrá ser ejecutada por cualquier Docker Engine, independientemente del sistema operativo y la infraestructura que haya por debajo.

### Inmutabilidad

Una aplicación la componen tanto el código fuente como las librerías del sistema operativo y del lenguaje de programación necesaria para la ejecución de dicho código. Estas dependencias **dependen a su vez del sistema operativo** donde nuestro código va a ser ejecutado, y por esto mismo ocurre muchas veces aquello de que “**no sé, en mi máquina funciona**”. Sin embargo, el proceso de instalación de dependencias en Docker **no depende del sistema operativo**, si no que este proceso se realiza cuando se genera una imagen de docker. Es decir, una imagen de docker (también llamada repositorio por su parecido con los repositorios de git) contiene tanto el código de la aplicación como las dependencias que necesita para su ejecución. Una imagen se genera una vez y puede ser ejecutada las veces que sean necesarias, y siempre ejecutará con las misma versión del código fuente y sus dependencias, por lo que se dice que es **inmutable**. Si unimos inmutabilidad con el hecho de que Docker es portable, decimos que Docker es una herramienta fiable, ya que una vez generada una imagen, ésta se comporta de la misma manera independientemente del sistema operativo y de la infraestructura donde se esté ejecutando.

### Ligereza

Los contenedores corriendo en la misma máquina comparten entre ellos el sistema operativo, pero cada contenedor es un proceso independiente con su propio sistema de ficheros y su propio espacio de procesos y usuarios. Esto hace que la ejecución de contenedores sea mucho más ligera que otros mecanismos de virtualización. Comparemos por ejemplo con otra tecnología muy utilizada como es Virtualbox. Virtualbox permite del orden de 4 ó 5 máquinas virtuales en un ordenador convencional, mientras que en el mismo ordenador podremos correr cientos de **containers** sin mayor problema, además de que su gestión es mucho más sencilla.

Anuncios

**La jerga de Docker**

Cuando empecé a leer sobre docker, encontré mucha terminología, muchas palabras que no terminaban de cuadrarme, así que aquí dejo un pequeño “diccionario sobre docker”:

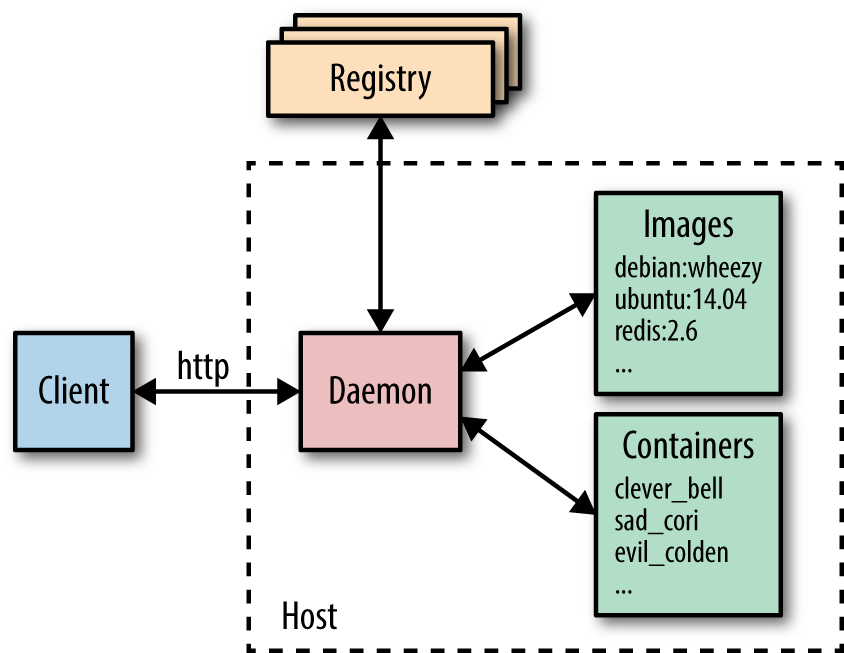
Principalmente hay 2 términos que se manejan en docker, **images** y **containers**(contenedores), además también tenemos otros términos como **links**, o **volumenes**

* **Images**: Una imagen es una especie de “máquina virtual” pero más ligera.  
  Crear una imagen es fácil, aún más fácil es utilizar imágenes de otras personas y modificarlas. Creando así imágenes hijas.  
  [Aquí](https://registry.hub.docker.com/search?q=library&f=official) tenemos el repositorio oficial de imágenes de Docker. Si necesitamos “tunear” una imagen tan solo tenemos que editar su *Dockerfile* para realizar los cambios.
* **Containers**: Es el directorio que contiene toda tu aplicación.
* **Dockerfile**: Script que automatiza el proceso de **construir la imagen**. En este script indicamos que necesitamos instalar en nuestro contenedor.
* **Volumenes**: Son una serie de datos que Docker guarda **fuera del contenedor** para poder persistir los datos de la aplicación. Los volúmenes son diferentes en cada contenedor aunque pueden compartirse, con lo que podríamos compartir datos entre 2 contenedores .
* **Link**: Como hemos comentado antes, los contenedores de Docker pueden compartir información, De la misma forma estos contenedores también **pueden comunicarse**. De esta manera podemos crear un contenedor donde ejecutar una app de Laravel (por ejemplo) y crear otro contenedor donde ejecutar la base de datos.

**Componentes de Docker**

Docker está formado fundamentalmente por tres componentes:

* Docker Engine
* Docker Client
* Docker Registry



### Docker Engine o Demonio Docker:

Es un demonio que corre sobre cualquier distribución de Linux (y ahora también en Windows) y que expone una API externa para la gestión de imágenes y contenedores (y otras entidades que se van añadiendo en sucesivas distribuciones de docker como volúmenes o redes virtuales). Podemos destacar entre sus funciones principales:

* Creación de imágenes docker.
* Publicación de imágenes en un Docker Registry o Registro de Docker (otro componente Docker que se explicará a continuación).
* Descarga de imágenes desde un Registro de Docker
* Ejecución de contenedores usando imágenes locales.

Otra función fundamental del Docker Engine es la gestión de los contenedores en ejecución, permitiendo parar su ejecución, re arrancarla, ver sus logs o sus estadísticas de uso de recursos.

### Docker Registry o Registro Docker

El Registro es otro componente de Docker que suele correr en un servidor independiente y donde se publican las imágenes que generan los Docker Engine de tal manera que estén disponibles para su utilización por cualquier otra máquina. Es un componente fundamental dentro de la arquitectura de Docker ya que permite distribuir nuestras aplicaciones. El Registro de Docker es un proyecto [**open source**](https://github.com/docker/distribution) que puede ser instalado gratuitamente en cualquier servidor, pero Docker ofrece Docker Hub, un sistema SaaS de pago donde puedes subir tus propias imágenes, acceder a imágenes públicas de otros usuarios, e incluso a imágenes oficiales de las principales aplicaciones como son: MySQL, MongoDB, RabbitMQ, Redis, etc.

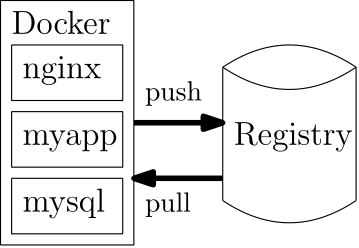
El registro de Docker funciona de una manera muy parecida a git (de la misma manera que Dockerhub y sus métodos de pago funcionan de una manera muy parecida a Github). Cada imagen, también conocida como repositorio, es una sucesión de capas. Es decir, cada vez que hacemos un build en local de nuestra imagen, el Registro de Docker sólo almacena el **diff** respecto de la versión anterior, haciendo mucho más eficiente el proceso de creación y distribución de imágenes.

### Docker Client o Cliente Docker

Es cualquier herramienta que hace uso de la api remota del Docker Engine, pero suele hacer referencia al comando docker que hace las veces de herramienta de línea de comandos (cli) para gestionar un Docker Engine. La cli de docker se puede configurar para hablar con un Docker Engine local o remoto, permitiendo gestionar tanto nuestro entorno de desarrollo local, como nuestros servidores de producción.

**Ciclo de Desarrollo con Docker**

Docker transforma radicalmente el concepto de entorno local de desarrollo. La siguiente figura muestra un esquema de dicho entorno local de desarrollo:



Los desarrolladores pueden correr contenedores con las dependencias externas de la aplicación que están desarrollando, tales como nginx o mysql. También corren la aplicación que están desarrollando en su propio contenedor. Una vez que han terminado el desarrollo de una nueva funcionalidad y los tests pasan en local se puede hacer push de la nueva imagen que han desarrollado al Registro. La imagen pusheada puede ser descargada en los servidores de producción para desplegar una nueva versión de nuestra aplicación con la garantía de que se comportará de la misma manera que en el entorno local del desarrollador gracias a las propiedades de portabilidad e inmutabilidad de los contenedores. Este ciclo de desarrollo es la gran ventaja que aporta Docker a los ciclos de desarrollo de software, y la razón por la que Docker se ha hecho tan popular.

**Instalación en MacOS/Windows/Linux**

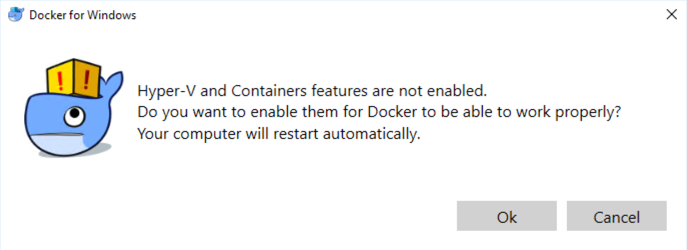
En la instalación de Docker queremos distinguir la instalación para el desarrollo en local, y la instalación en servidores para correr código en producción. En cuanto a la instalación en servidores de producción, la mayoría de proveedores de servicio: AWS, GCE, Azure, Digital Ocean… disponen de máquinas virtuales con versiones de Docker pre-instaladas. En cualquier caso, la instalación es bastante sencilla. Como el proceso varía un poco entre versiones de Docker, preferimos poner un enlace a la documentación oficial de la [**instalación en Ubuntu**](https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/).

Docker para Windows funciona en Windows 10 Professional o Enterprise 64-bit, en los cuales nos enfocaremos en este artículo. Sin embargo, si está usando un sistema legacy de Windows, puede consultar el [Docker Toolbox](https://docs.docker.com/toolbox/toolbox_install_windows/).

El Docker para Windows tiene dos versiones: la de comunidad (Community Edition) y la empresarial (Enterprise Edition). Como no vamos a hacer nada lujoso y me gusta lo gratis, vamos a utilizar la edición de la comunidad.

En nuestra máquina de Windows 10, primero tendremos que descargar la [versión de comunidad de Docker para Windows](https://store.docker.com/editions/community/docker-ce-desktop-windows). Una vez descargado, instálelo adecuadamente. Cuando termine, le solicitara que cierre su sesión en Windows—cierre y vuelva a iniciar.

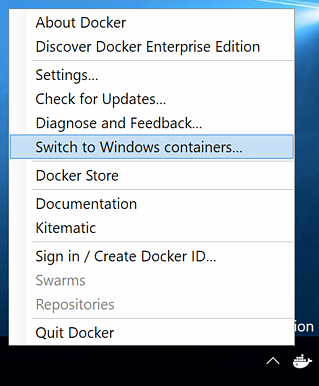
Una vez que haya terminado la instalación, el programa automáticamente iniciara después que la computadora se haya reiniciado. Cuando le aparezca, se le pedirá que instale las funciones de Hyper-V y Containers. Prosiga y luego reinicie.



Haga clic en OK, pero no se preocupe si no pasa nada porque la aplicación está instalando las funciones autorizadas en segundo plano. Espere un minuto más o menos y notará que su computadora se reiniciara. Una vez que su computadora se reinicie, Docker intentará abrir la bandeja del sistema.

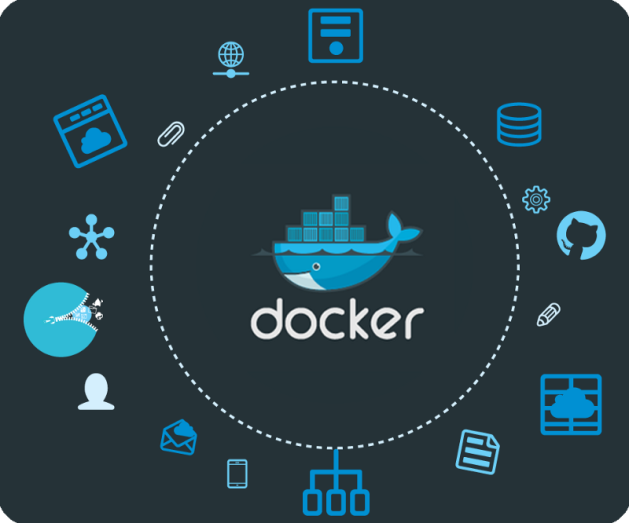
****

***Si Docker no se inicia*** cuando está usando Windows 10 en un entorno ya virtualizado como Parallels, VMWare Fusion en MacOS, o quizás IaaS en la nube, probablemente no podrá darle mensajes de error para notificarle que hubo fallos al intentar iniciar la máquina virtual MobyLinuxVM. De todos modos, como vamos a crear un contenedor de Windows tenemos que cambiarnos de Docker para Windows a contenedores de Windows (*Windows Containers*).



Una vez que se haya cambiado a Windows Containers, Docker debería de funcionar correctamente. El siguiente paso es desplegar una imagen para usar.

# Lo que obtienes y cómo funciona



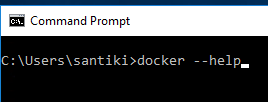
**Docker Toolbox** incluye las siguientes herramientas **Docker**:

* **Docker engine** para manejar las imágenes, contendores, volúmenes, etc.
* **Cliente Docker CLI** para poder comunicarse con el **Docker Engine**.
* **Docker Machine** para que pueda ejecutar comandos de **Docker Engine** desde terminales de Windows.
* **Docker Compose** para ejecutar el comando **docker-compose**.
* **Kitematic**, la interfaz gráfica de usuario de Docker en window.
* El **shell Docker QuickStart** preconfigurado para un entorno de **línea de comandos Docker**.

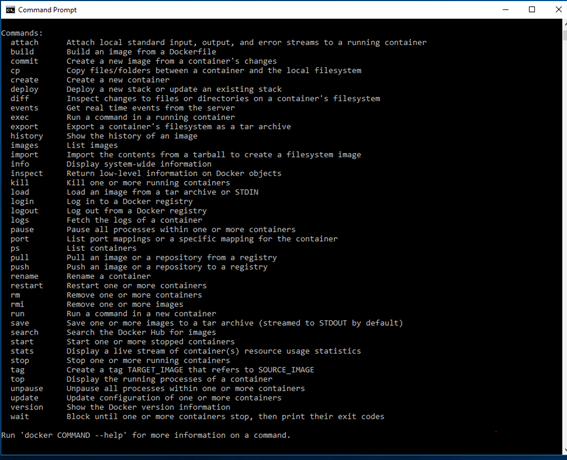
**Iniciando Docker**

Para trabajar con Docker y familiarizarnos con la herramienta, podemos usar la **consola o PowerShell**. En mi caso voy a utilizar la consola CMD para empezar a usar Docker, comentar que existen opciones como Kitematic o Portainer si estamos más cómodos trabajando con herramientas gráficas IU.

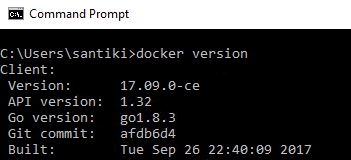
Lo primero, es abrir la consola CMD y ver la lista de comandos disponibles de Docker:



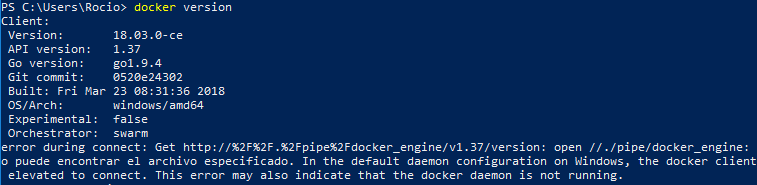
Nos mostrará la lista de comandos que podemos utilizar desde la consola, si queremos información detallada y ejemplos de uso, en la página oficial están todos documentados: <https://docs.docker.com/engine/reference/commandline/docker/#child-commands>.



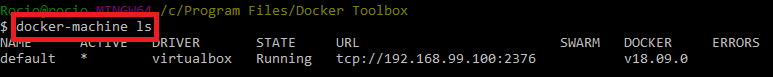
Para utilizar los comandos, escribiremos “docker [comando]”. Por ejemplo, vamos a consultar la versión de Docker instalada con “docker version”:



Utilizando PowerShell



### Si queremos ver un listado de las máquinas virtuales instaladas introducimos el siguiente comando:

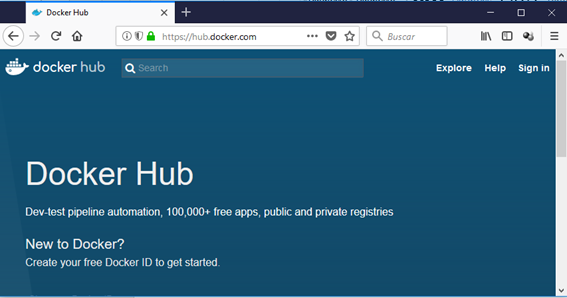


¿Qué es Docker Hub?

[**Docker Hub**](https://hub.docker.com/explore/) es un **repositorio de imágenes para Docker**, en él podemos encontrar gran cantidad de imágenes con el software más popular listo para ser ejecutado. Sería similar a un conjunto de máquinas virtuales ya listas, que nosotros en [**virtual box**](http://www.somosbinarios.es/virtualizacion-de-sistemas-operativos/) bajaríamos e importaríamos. Pero en este caso, como ya se ha comentado, no es necesario que a todo software le acompañe un sistema operativo, sino que podemos tener imágenes con un programa como nginx o MySQL.

**Creando nuestra cuenta en Docker Hub**

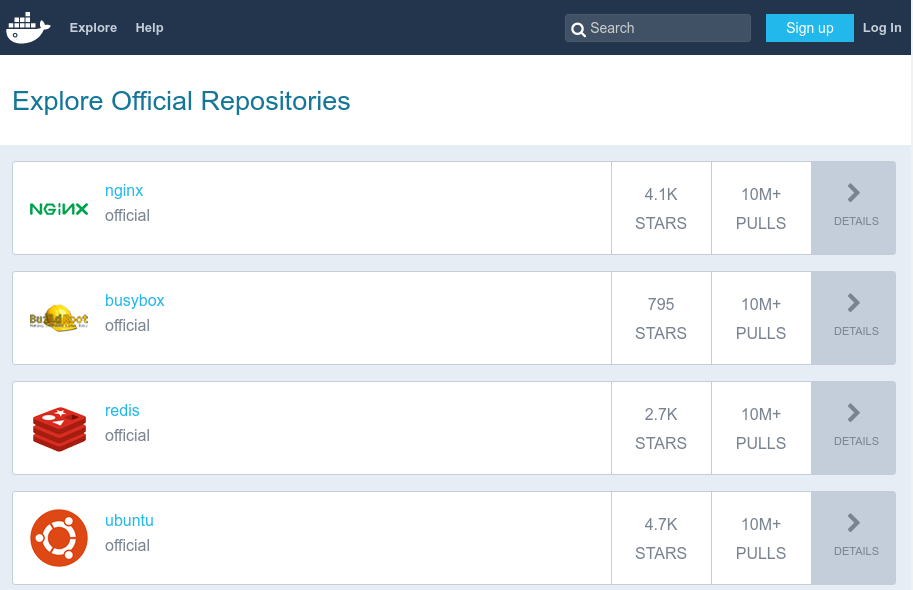
Aunque *NO es obligatorio*, yo siempre recomiendo crear una cuenta [Docker Hub](https://hub.docker.com/) para no tener problemas más tarde a la hora de descargar imágenes, ya que, algunas nos pedirán nuestras credenciales para descargarse en nuestro PC:



El proceso de registro es tan simple que no creo que sea necesario documentarlo: ir a la web de [Docker Hub](https://hub.docker.com/), simplemente registrarse y activar la cuenta desde el enlace que llegará a vuestro email.

**Buscando el Docker Hub**

Lo primero es buscar en [Docker Hub](https://hub.docker.com/) nuestra imagen a descargar,



 CREAR UN CONTAINER (“DOCKER RUN”)

El comando “**docker run**” creará el container usando la imagen que le especifiquemos, y seguidamente iniciará el container. La sintaxis es la siguiente:

docker run [options] [image] [command] [args]

**Hello World, Docker**

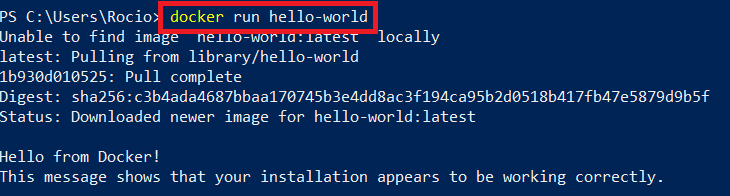
## 1. Puesta en marcha de un contenedor sencillo

Una vez que se ha tratado el tema de la instalación de Docker, es momento de comenzar a explorar la herramienta en sí misma. Como cualquier tecnología informática que se precie, Docker dispone de un ejemplo "Hello World". En nuestro caso, un contenedor que no hace nada salvo mostrar un mensaje de bienvenida. El interés de este tipo de enfoque es que permite validar que la instalación del producto se hace de manera correcta, verificar que el conjunto de la cadena lógica funciona y servir de ayuda al usuario principiante.

*Poner en marcha un contenedor Hello World*

docker run hello-world

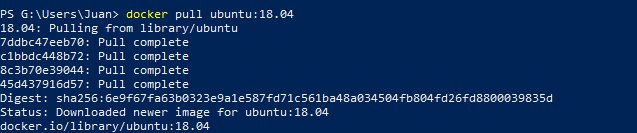
Excepto algún problema de instalación de Docker o de acceso a Internet, la visualización debería ser la siguiente:



## Otro ejemplo crearemos un contendor con ubuntu:

Hoy probaremos Ubuntu 18.04 “Castor Biónico” en un contenedor Docker, en lugar de tener que instalarlo en un equipo real o virtual, demorar un rato para configurarlo, crear cuenta de acceso, claves, zona horaria, particiones de disco… Mejor usamos Docker.

docker **pull** ubuntu:18.04



# docker **images**



docker **exec** -it ubuntu:18.04 /bin/bash



Verificamos el Kernel que tenemos corriendo en el contenedor.

# uname -a

## 

También la versión del shell (bash en nuestro caso)

# bash -version

## 

## Para verificar que usaremos las últimas versiones, hacemos un “Update/Upgrade”

## # apt update

## 

## # apt upgrade

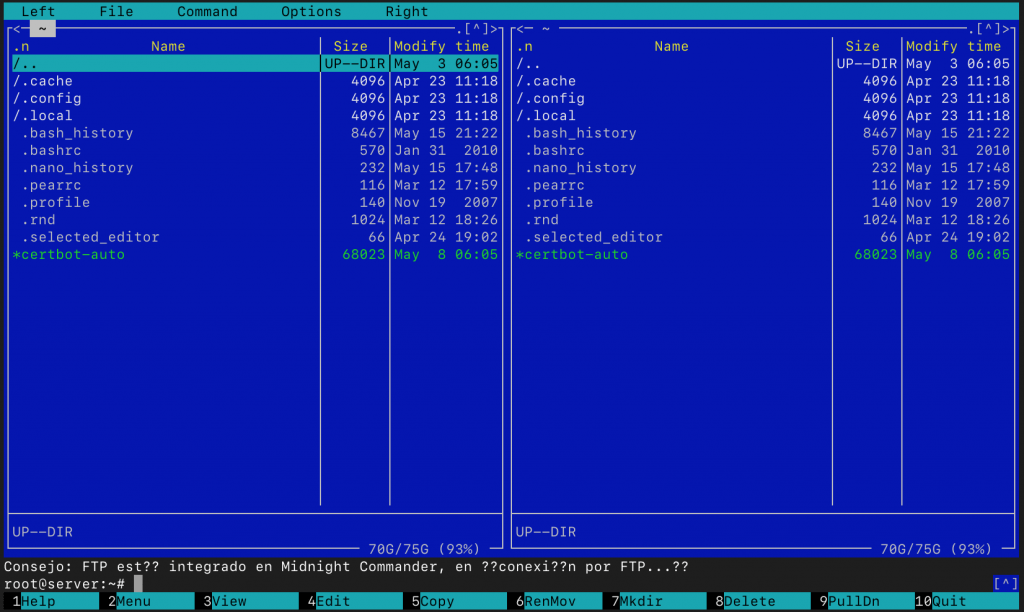
## 

## También instalamos dos paquetes para probar (mc y nmap)

[Midnight Commander](https://midnight-commander.org/) es un **gestor de archivos** para ser usado en el **terminal de linux**. Nos facilita las operaciones de gestión de ficheros, proporcionando un interfaz más amigable que el simple prompt.

apt-get install mc

Una vez instalado, podemos ejecutarlo con el comando mc.



apt-get install nmap

Una vez instalado, podemos ejecutarlo con el comando nmap.

nmap -sP 192.168.0.0/24

Algunos de los comandos más utilizados en Docker

Una vez que tenemos docker corriendo en nuestra máquina, podemos empezar a ejecutar algunos comandos:

## Información sobre la versión de docker que estamos corriendo.

**docker version**

## Información acerca de la cantidad de contenedores e imágenes y plugins instalados

docker info

## Listado de los contenedores que están corriendo en la máquina

docker ps

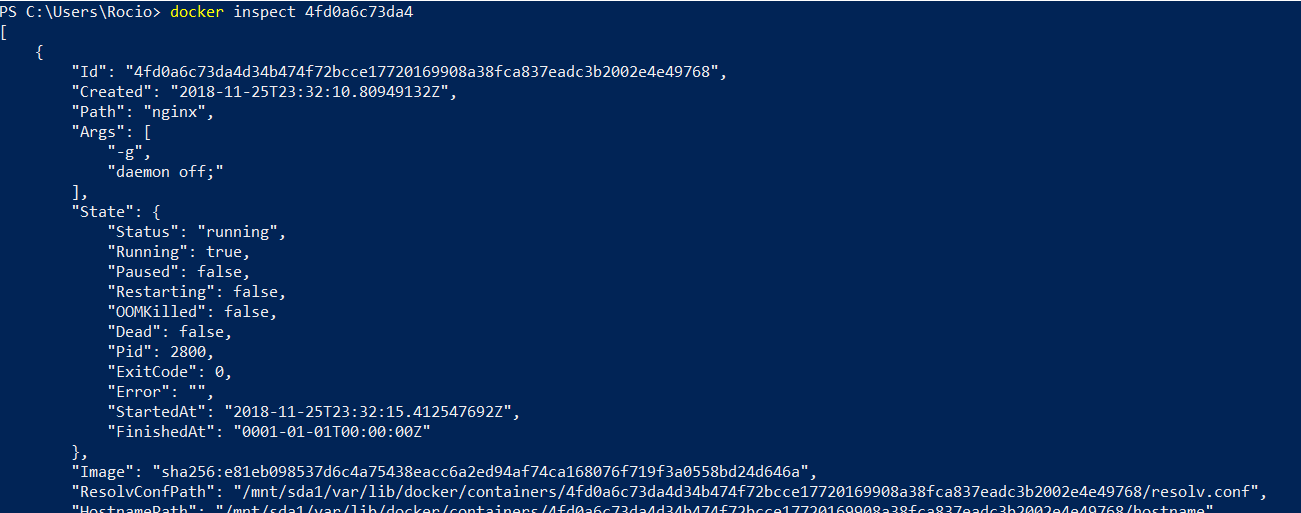
* Con el flag -a muestra también los contenedores que están parados.

## 

## Obtener información detallada de un contenedor en ejecución

La información es devuelta en formato JSON. Esta información nos devuelve todos los detalles del contenedor, como el ID, la IP donde se encuentra disponible, la MAC address, el hostname, etc.

docker inspect 4fd0a6c73da4



Y si queremos filtrar solo algún tipo de información, lo podemos hacer utilizando una plantilla del lenguaje go.

$ docker inspect -f "Host name: {{.Config.Hostname}}, Identificador: {{.ID}}" 4fd0a6c73da4



## Arrancar un contenedor

$ docker **start** 4fd0a6c73da4

## 

## Parar un contenedor

docker **stop** 4fd0a6c73da4

## 

detener **todos** los contenedores

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

## Borrar un contenedor

Una vez hemos terminado de utilizar un contenedor, lo podemos eliminar si ya no nos hace falta. **Previamente lo tenemos que parar**. Una vez lo hemos borrado, dejaremos de tenerlo en la lista de contenedores.

docker **rm** 4fd0a6c73da4



Te dará este error, por lo que primero deberás parar el contenedor

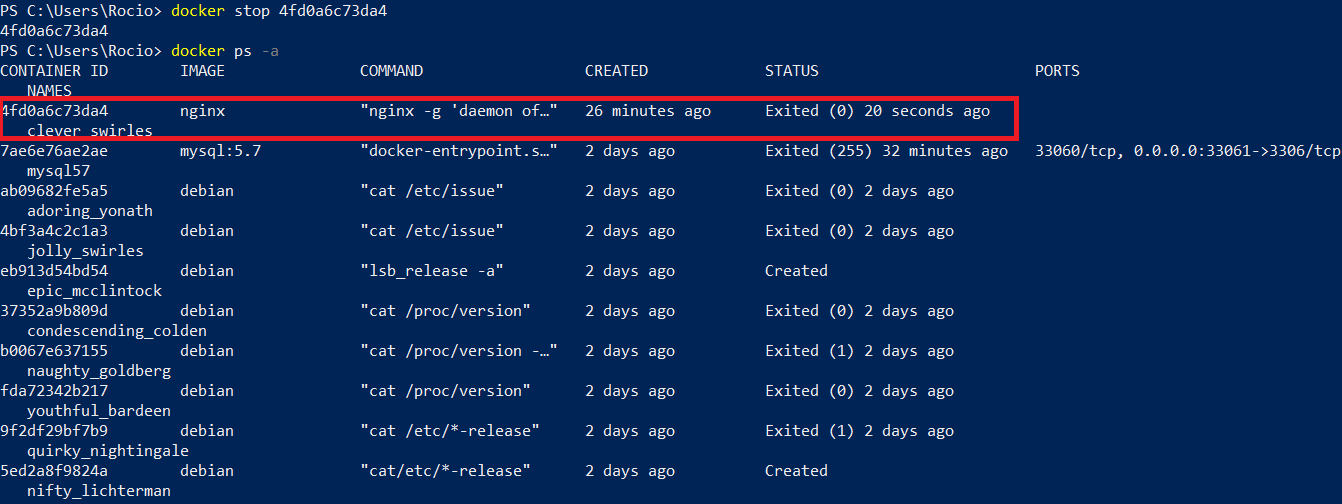
docker **stop** 4fd0a6c73da4

Ahora si podrás borrar el contenedor

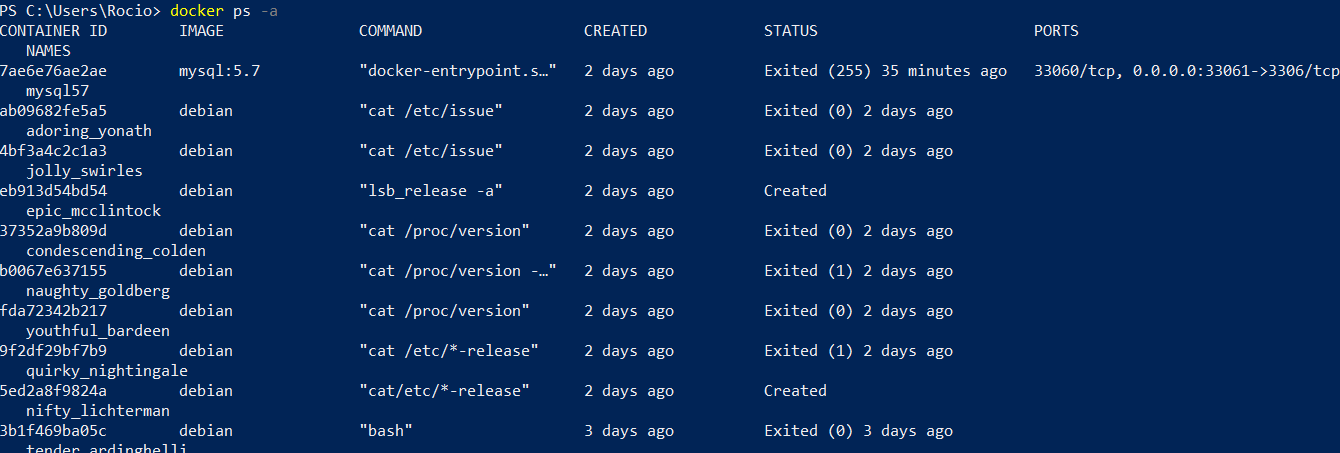


docker **ps** –a

Pantalla de después de pararlo



Pantalla después de borrarlo, verás que ya NO existe.



Una forma de borrar todos los contenedores

docker **rm** $(docker **ps** -a -q)

Un pequeño apunte: si una de los contenedores está en ejecución deberemos usar la opción  **-f** para ***forzar la eliminación***, o bien detener la ejecución. **–v** borra los volumenes asociados

docker **rm** **-fv** $(docker **ps** -aq )

Borrar **todos** los contenedores **detenidos**

docker **container** prune

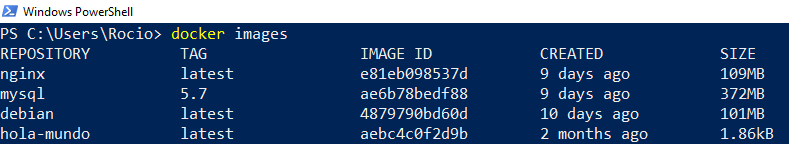
Aparecerá un mensaje para que confirmes si realmente deseas borrarlos. Si no deseas que te pregunte, añade el flag **-f**.

docker **container** prune –f

## Listado de las imágenes descargadas localmente

Todas esta las imágenes las hemos descargado desde [Docker Hub](https://hub.docker.com/).

docker images



## Descargar de una imagen

Cada vez que ejecutamos una imagen que no tenemos, la imagen se **descarga** y se crea el contenedor donde se ejecuta esa imagen. Pero en ocasiones queremos tener ya la imagen descargada para no tener que esperar a su descarga, en ese caso lo hacemos con docker pull.

docker pull elasticsearch:2.1.0

**(Nota:** [Elastic Search, una de las mejores soluciones corporativas que podemos llevar a Cloud)](https://www.arsys.es/blog/soluciones/cloud/cloudbuilder-next/elastic-search-cloud/)

Y una vez la tenemos descargada podemos en cualquier momento crear el contenedor para esa imagen y ejecutarlo. En este caso vamos a ejecutar un contenedor de elasticsearch y verificar su estado.

docker run -dP elasticsearch:2.1.0

## Eliminar de una imagen

Para ver las imágenes que tenemos en el sistema, podemos usar el comando:

docker images -a

Gracias al flag -a veremos todas las imágenes de Docker, incuidas las imágenes intermedias.

Una vez localizada la imagen que queremos borrar, utilizaremos su ID para borrarla:

docker rmi <id\_image1> <id\_image2> <id\_image 3>

### Borrar imágenes “colgadas” (dangling images)

Las imágenes colgadas son las capas de imágenes que no se están utilizando en ningún contenedor, tanto que esté funcionando, como que esté parado. Estas imágenes pueden ser borradas en la gran mayoría de las veces. Podemos ver estás imágenes con el comando:

docker images -f dangling=true

Y las podemos borrar con:

docker images purge

### Borrar todas las imágenes

Si queremos borrar **todas** las imágenes, podemos ejecutar:

docker rmi $(docker images -a -q)

opción  **-f** para ***forzar la eliminación***

## Eliminar de un volumen

Para ver los volúmenes que tenemos en el sistema, podemos usar el comando:

docker volume ls

Cuando localizamos el volumen, lo podemos borrar con:

docker volume rm <nombre\_volumen1> <nombre\_volumen2> <nombre\_volumen3>

### Borrar volúmenes colgados (dangling volumes)

Los volúmenes colgados son aquellos que no están conectados con ningún contenedor. Ojo, ten en cuenta que es posible que haya volúmenes en ese estado y que no tengan que ser borrados.

Para verlos utilizaremos:

docker volume ls -f dangling=true

Localizaremos los que queremos borrar y lo haremos con el mismo comando del punto anterior:

docker volume rm <nombre\_volumen1> <nombre\_volumen2> <nombre\_volumen3>

Si queremos borrar todos los dangling volumes, ejecutaremos:

docker volume prune

## Ejecutar un comando en un contenedor

docker exec

Útil para depurar contenedores en ejecución con las opciones

docker exec -it contenedor bash.

## Listado de los mapeos de puertos de un contenedor

**Mapeo de todos los puertos**

$ docker port 4fd0a6c73da4



**Mapeo de un puerto**

$ docker port 4fd0a6c73da4 80



Otros comandos

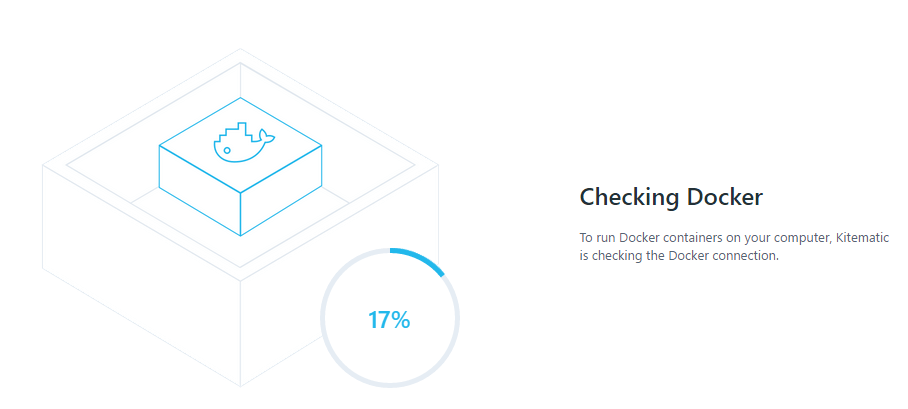
* docker c : copia archivos entre el host y un contenedor.
* docker logs: muestra los logs de un contenedor.
* docker stats: muestras las estadísticas de ejecución de un contenedor.

## Kitematic o cómo desplegar contenedores Docker fácilmente

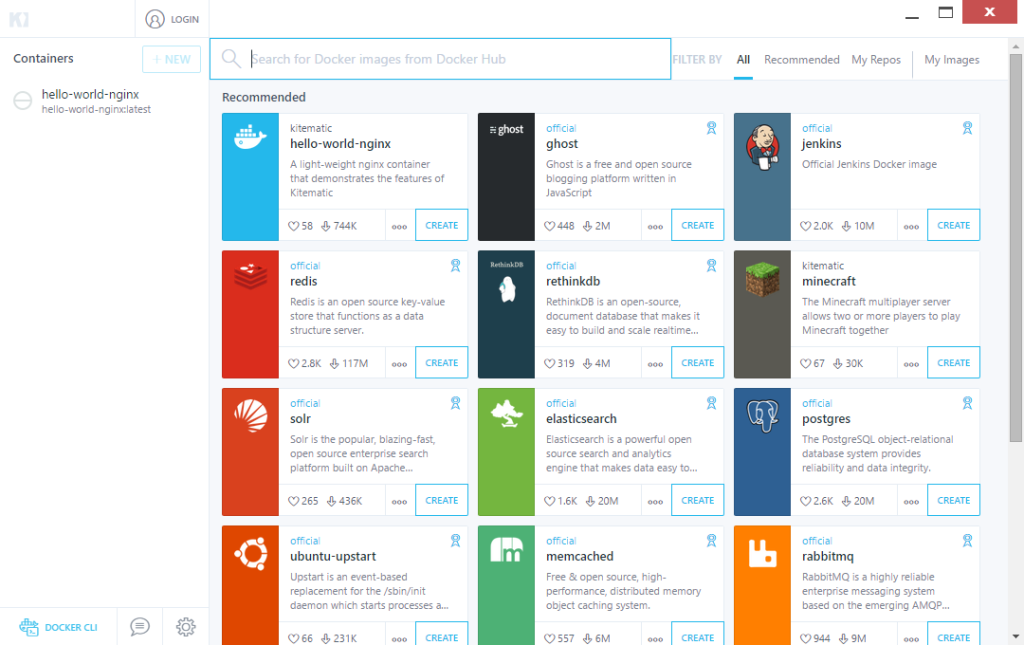
[**Kitematic**](https://kitematic.com/)surge para ayudar a todos aquellos que busquen usar Docker de una manera visual, sin necesidad alguna de utilizar la línea de comandos. Este proyecto está desarrollado bajo la filosofía Open Source y está disponible para [**Mac**](http://www.somosbinarios.es/instalar-mac-os-x-yosemite-en-tu-pc/) OS  y [**Windows**](http://www.somosbinarios.es/category/sistemas-operativos/windows/).

### Desplegando un contenedor Docker con Kitematic

En mi caso para esta demostración, he utilizado como sistema operativo Windows 10.

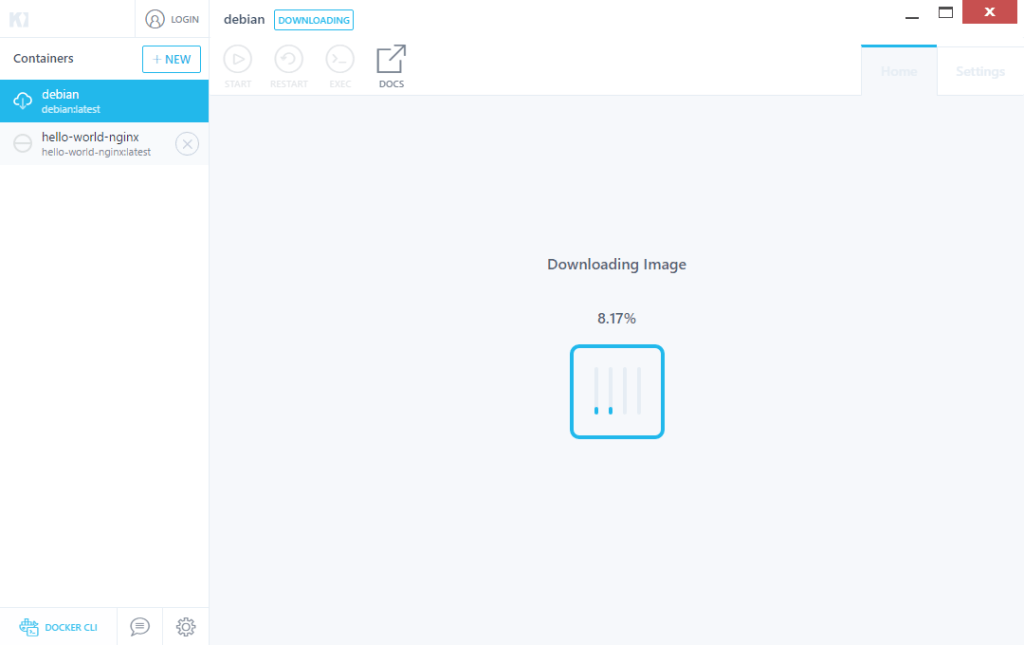


Lo primero que hace Kitematic nada más arrancar es intentar conectarse con Docker para empezar a desplegar los contenedores.

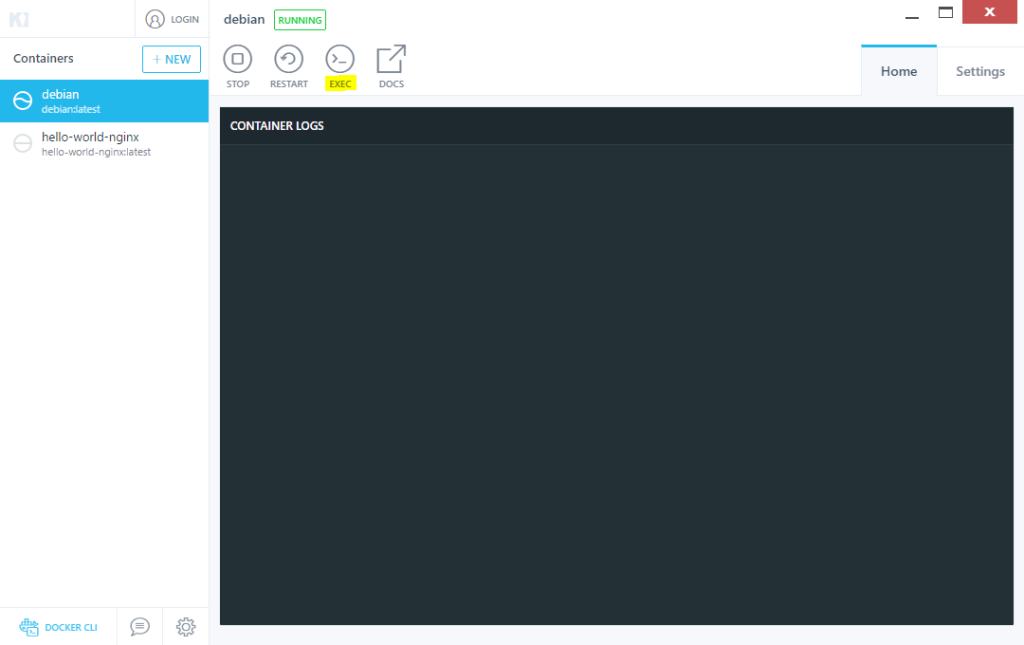


Una vez ya ha conectado, pasamos a ver la pantalla principal de Kitematic, aquí podemos buscar la imagen que vamos a desplegar en un contenedor Docker. Como podemos ver a la izquierda nos muestra nuestros contenedores, la opción de acceso con nuestra cuenta a Docker Hub, al acceso a Docker vía terminal y las opciones del programa.

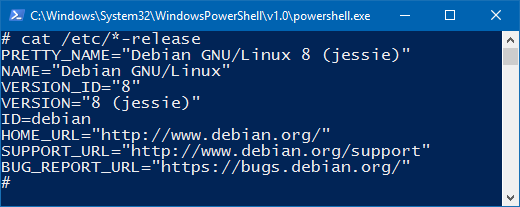
En nuestro caso vamos a desplegar un Debían, como hicimos en el artículo anterior. Para ello buscamos la imagen en el menú superior y le damos a Create.



Como se puede ver en la imagen superior, Docker está descargando desde el repositorio correspondiente la imagen que vamos a utilizar. Este procedimiento es similar a cuando usábamos Docker desde la terminal.



Una vez que nuestra imagen está lista, es desplegada en un contenedor Docker. Ahora si hacemos click en **Exec** obtendremos una terminal de nuestro Debian desplegado.



# [Play with Docker](http://jorgeblom.com/2017/06/27/play-with-docker/)

[Play-with-docker.com](http://play-with-docker.com/) es una plataforma online que nos ofrece de forma gratuita y durante un tiempo limitado (4 horas), instancias de máquinas donde podremos ejecutar nuestros Docker.