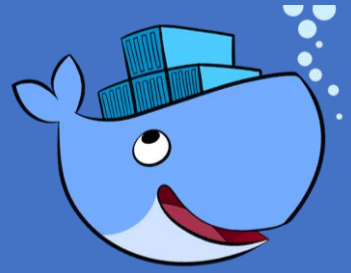
DOCKER DESDE CERO Y PARA TODOS.

DOCKER – DOCKER-COMPOSE – DOCKER SWARM

De novato a experto.

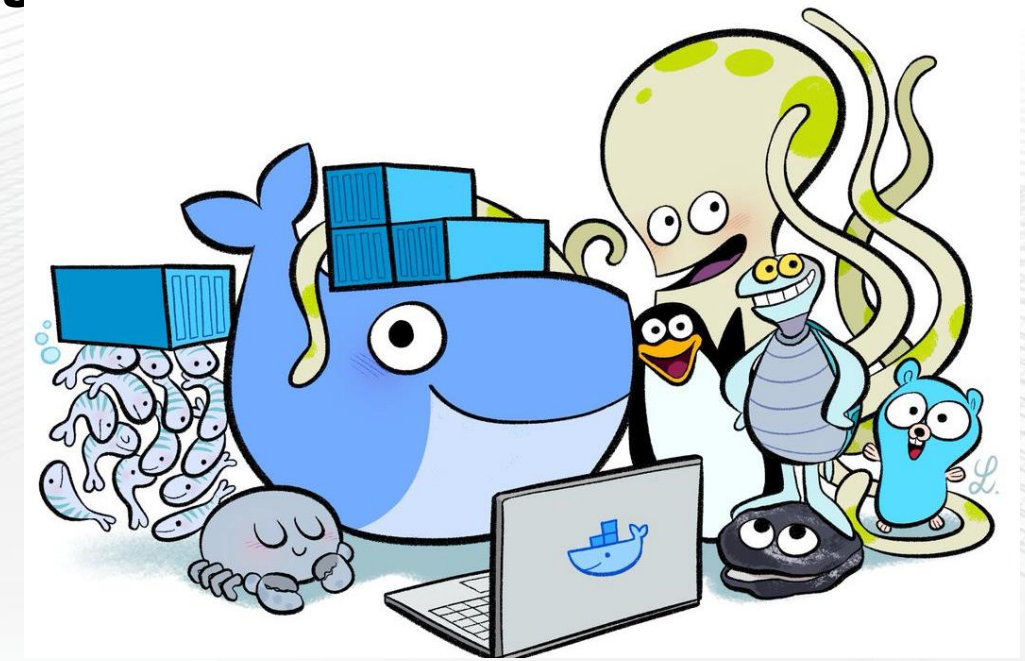


Adrián Hernández Ríos

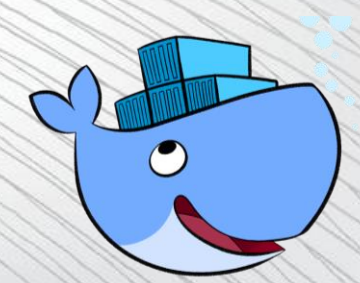
https://github.com/adririos98

¿Qué vamos a ver?

- **Aprenderemos a manejar imágenes y contenedores y cómo aplicarlos para mejorar en la creación, mantenimiento y nuevas soluciones.**
- **Siempre con el objetivo de poder crear un entorno que podamos compartir y facilitar el despliegue de nuestro proyecto**

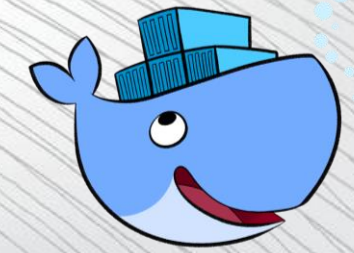


Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/



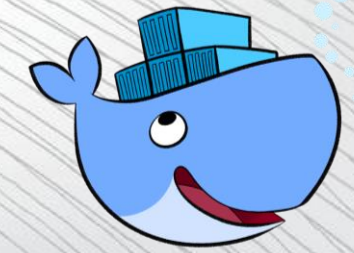
¿Qué es Docker?

- Docker es una plataforma para desarrollar, lanzar y ejecutar aplicaciones. Permite separar las aplicaciones desarrolladas de la infraestructura donde se desarrollan y trabajar así más comoda y rápidamente.
- Docker permite empaquetar y lanzar una aplicación en un entorno totalmente aislado llamado contenedor. Estos contenedores se ejecutan directamente sobre el kernel de la máquina por lo que son mucho más ligeros que las máquinas virtuales.
- Con Docker podemos ejecutar mucho más contenedores para el mismo equipo que si éstos fueran maquina virtuales. De esta manera, podemos probar rápidamente nuestra aplicación web, por ejemplo, en múltiples entornos distintos al de donde nos encontramos desarrollando. Realmente es mucho más rápido que hacerlo en una máquina virtual, puesto que reduce el tiempo de carga y el espacio requerido por cada uno de estos contenedores o máquinas.



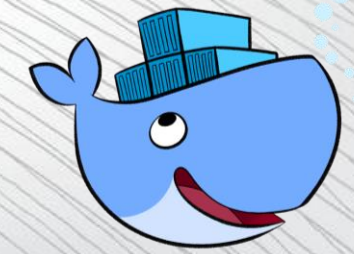
Contenedores

- Los contenedores se distinguen de las máquinas virtuales en que las máquinas virtuales emulan un ordenador físico en el que se instala un sistema operativo completo, mientras que los contenedores usan el kernel del sistema operativo anfitrión pero contienen las capas superiores (sistema de ficheros, utilidades, aplicaciones).
- Al ahorrarse la emulación del ordenador y el sistema operativo de la máquina virtual, los contenedores son más pequeños y rápidos que las máquinas virtuales. Pero al incluir el resto de capas de software, se consigue el aislamiento e independencia entre contenedores que se busca con las máquinas virtuales.
- Docker no es virtualizado, no hay un hipervisor. Los procesos que corren dentro de un contenedor de docker se ejecutan con el mismo kernel que la máquina anfitrión. Linux lo que hace es aislar esos procesos del resto de procesos del sistema, ya sean los propios de la máquina anfitrión o procesos de otros contenedores.
- Además, es capaz de controlar los recursos que se le asignan a esos contenedores (cpu, memoria, etc).

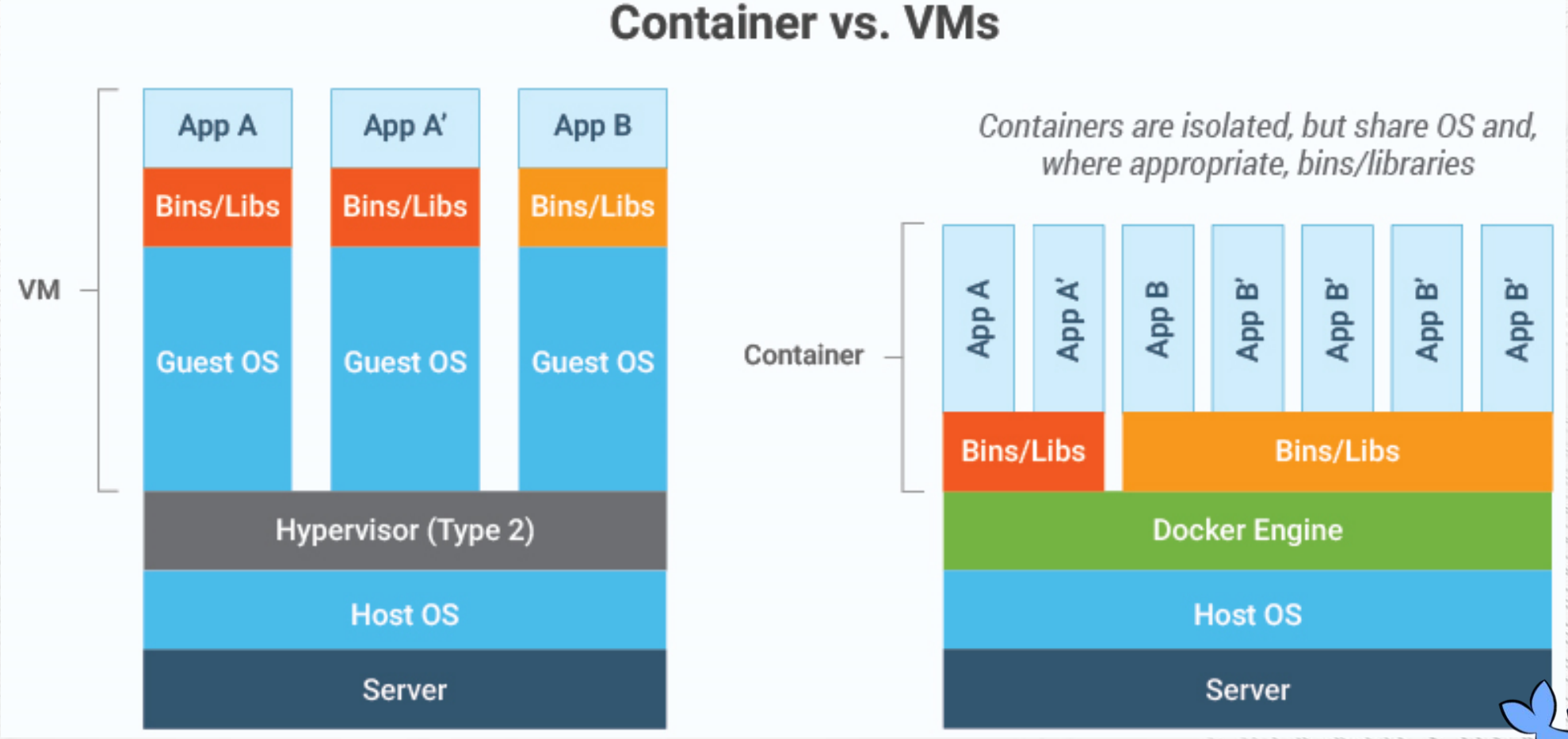


Contenedores

- Internamente, el contenedor no sabe que lo es y a todos los efectos es una distribución GNU/Linux independiente, pero sin la penalización de rendimiento que tienen los sistemas virtualizados.
- Así que, cuando ejecutamos un contenedor, estamos ejecutando un servicio dentro de una distribución construida a partir de una "receta". Esa receta permite que el sistema que se ejecuta sea siempre el mismo, independientemente de si estamos usando Docker en Ubuntu, Fedora o, incluso, sistemas privativos compatibles con Docker.
- De esa manera podemos garantizar que estamos desarrollando o desplegando nuestra aplicación, siempre con la misma versión de todas las dependencias.



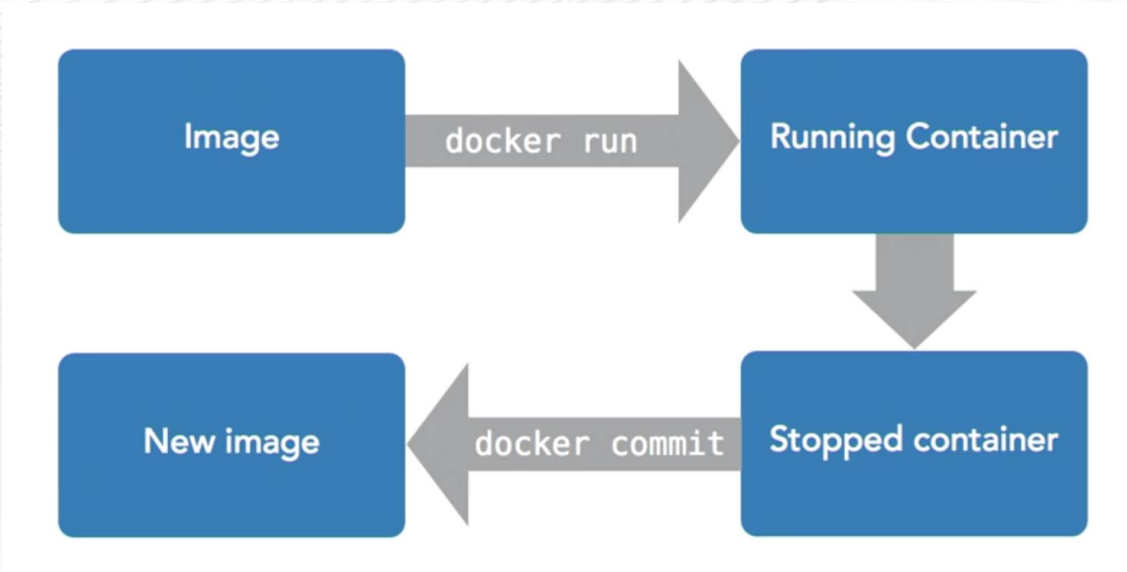
Contenedores

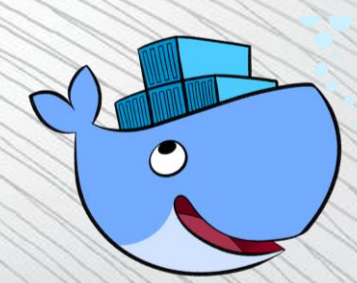


Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

Imágenes

- Una Imagen es una plantilla de solo lectura que contiene las instrucciones para crear un contenedor Docker. Pueden estar basadas en otras imágenes, lo cual es habitual. Para ello usaremos distintos ficheros como el dockerfile.
- Por ejemplo una imagen podría contener un sistema operativo Ubuntu con un servidor Apache y tu aplicación web instalada.

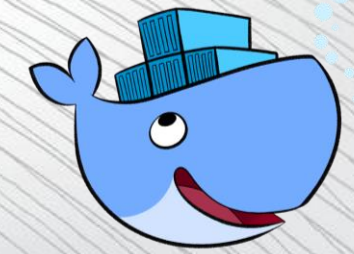




Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

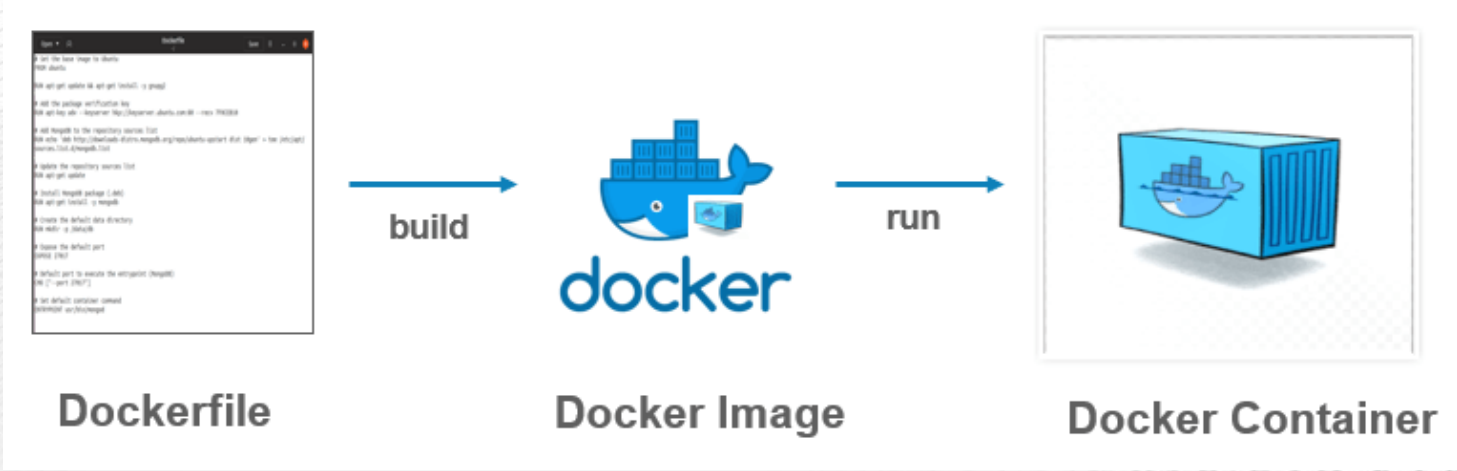
Imágenes y contenedores

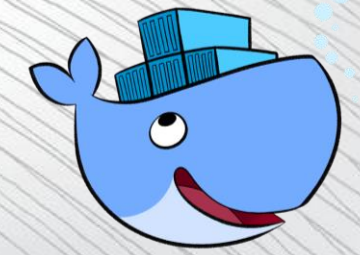
- Por lo tanto, un contenedor es una instancia ejecutable de una imagen.
- Esta instancia puede ser creada, iniciada, detenida, movida o eliminada a través del cliente de Docker o de la API.
- Las instancias se pueden conectar a una o más redes, sistemas de almacenamiento, o incluso se puede crear una imagen a partir del estado de un contenedor.
- Se puede controlar cómo de aislado está el contenedor del sistema anfitrión y del resto de contenedores.
- El contenedor está definido tanto por la imagen de la que procede como de las opciones de configuración que permita.
- Por ejemplo, la imagen oficial de MariaDb permite configurar a través de opciones la contraseña del administrador, de la primera base de datos que se cree, del usuario que la maneja, etc.



Dockerfile

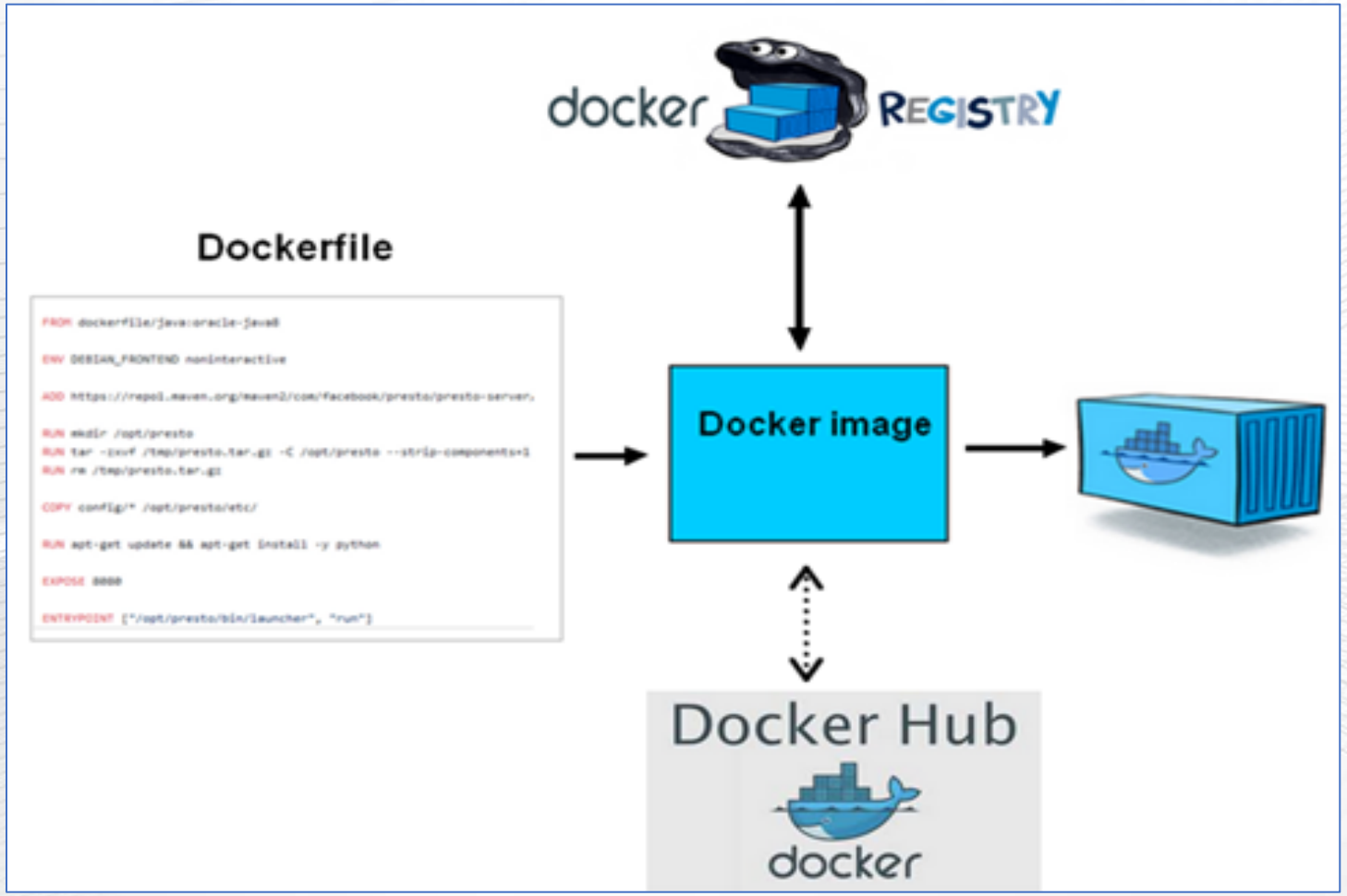
- Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones necesarias para crear una imagen que, posteriormente, se convertirá en los contenedores que ejecutamos en el sistema.
- Es como la receta para definir la imagen, que posteriormente lanzaremos como contenedor.

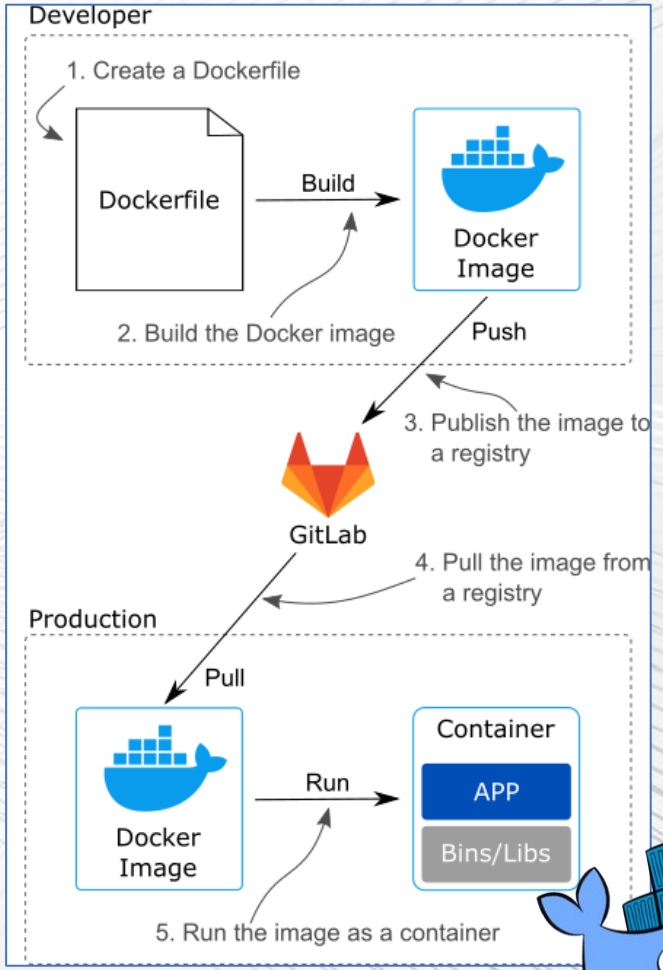




Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

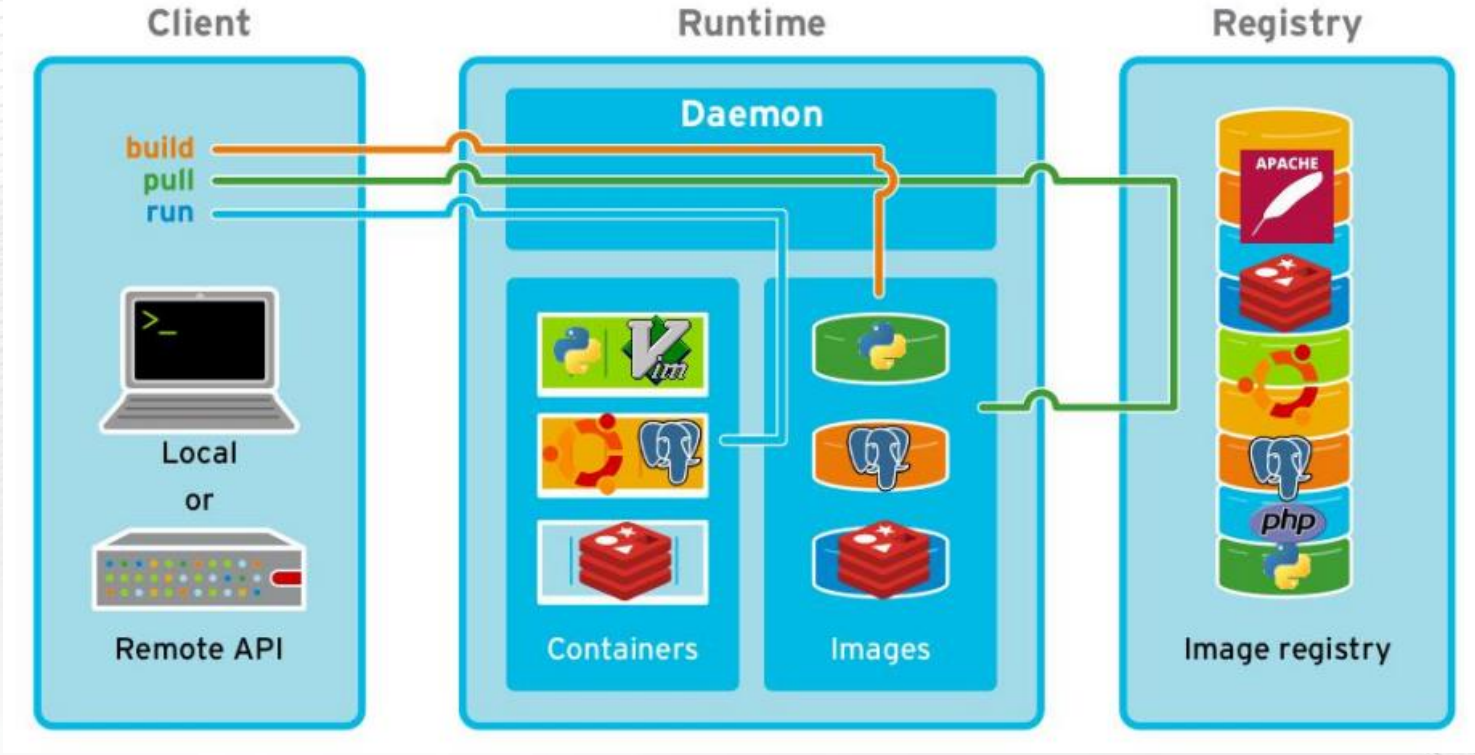
ESQUEMA MENTAL





Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

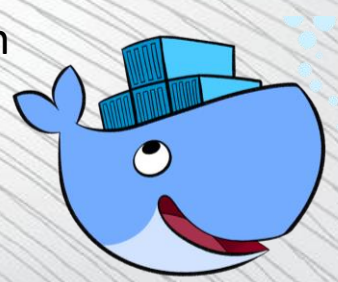
- **DockerHub** es el registro de imágenes donde podemos subir nuestras imágenes o encontrar imágenes ya hechas con la que trabajar sobre ellas. Podemos usarlo de manera muy similar a la que usamos GitHub y nos podemos dar de alta en dicho servicio para almacenar nuestras imágenes.



Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

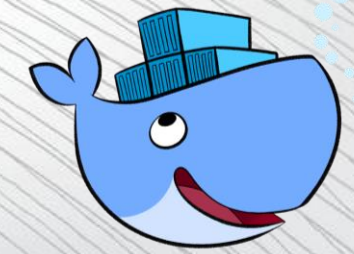
Persistencia de datos

- Por defecto ya hemos indicado que un contenedor está aislado de todo. Hemos visto como podemos conectar el contenedor a un puerto de red para poder acceder a él. Eso incluye al sistema de archivos que contiene. Detal manera que si se elimina el contenedor, se eliminan también sus archivos.
- Si queremos almacenar datos (una web, una base de datos, etc.) dentro de un contenedor necesitamos una manera de almacenarlos sin perderlos.
- Docker ofrece tres maneras:
  - A través de volúmenes, que son objetos de Docker como las imágenes y los contenedores.
  - Montando un directorio de la máquina anfitrión dentro del contenedor (enlazando directorios).
  - Almacenándolo en la memoria del sistema (aunque también se perderían al reiniciar el servidor).
- Lo normal es usar volúmenes, pero habrá ocasiones en que es preferible montar directamente un directorio de nuestro espacio de trabajo. Por ejemplo, para guardar los datos de una base de datos usaremos volúmenes, pero para guardar el código de una aplicación o de una página web montaremos el directorio.
- La razón para esto último es que tanto nuestro entorno de desarrollo como el contenedor tengan acceso a los archivos del código fuente. Los volúmenes, al contrario que los directorios montados, no deben accederse desde la máquina anfitrión.



Persistencia de datos

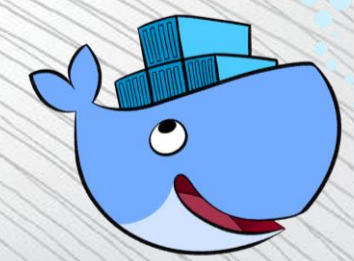
- **Volúmenes**
- En vez de guardar los datos persistentes en la máquina host, Docker dispone de unos elementos llamados volúmenes que podemos asociar también a directorios del contenedor, de manera que cuando el contenedor lea o escriba en su directorio, donde leerá o escribirá será en el volumen.
- Los volúmenes son independientes de los contenedores, por lo que también podemos conservar los datos aunque se destruya el contenedor, reutilizarlos con otro contenedor, etc. La ventaja frente a los directorios enlazados es que pueden ser gestionados por Docker. Otro detalle importante es que el acceso al contenido de los volúmenes sólo se puede hacer a través de algún contenedor que utilice el volumen.
- Los volúmenes son entidades independientes de los contenedores, pero para acceder al contenido del volumen hay que hacerlo a través contenedor, más exactamente a través del directorio indicado al crear el contenedor.



Docker compose

- El cliente de Docker es engorroso para crear contenedores, así como para crear el resto de objetos y vincularlos entre sí a base de Dockerfiles.
- Para automatizar la creación, inicio y parada de un contenedor o un conjunto de ellos, Docker proporciona una herramienta llamada Docker Compose.
- Compose es una herramienta para definir y ejecutar aplicaciones multi-contenedor. Con un solo comando podremos crear e iniciar todos los servicios que necesitamos para nuestra aplicación.
- Los casos de uso más habituales para docker-compose son: Entornos de desarrollo Entornos de testeo automáticos (integración contínua) Despliegue en host individuales (no clusters)
- Compose tiene comandos para manejar todo el ciclo de vida de nuestra aplicación: Iniciar, detener y rehacer servicios. Ver el estado de los servicios. Visualizar los logs. Ejecutar un comando en un servicio
- **EJ:**

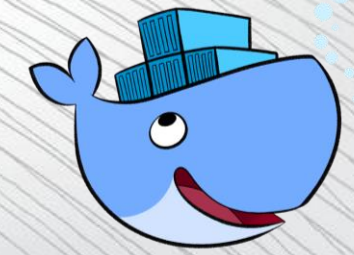
```
version: '2'
services:
    [Name_Service]:
        image: ubuntu
        command: [/bin/echo, 'Hola compañeros, esto es Docker Compose']
```



Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

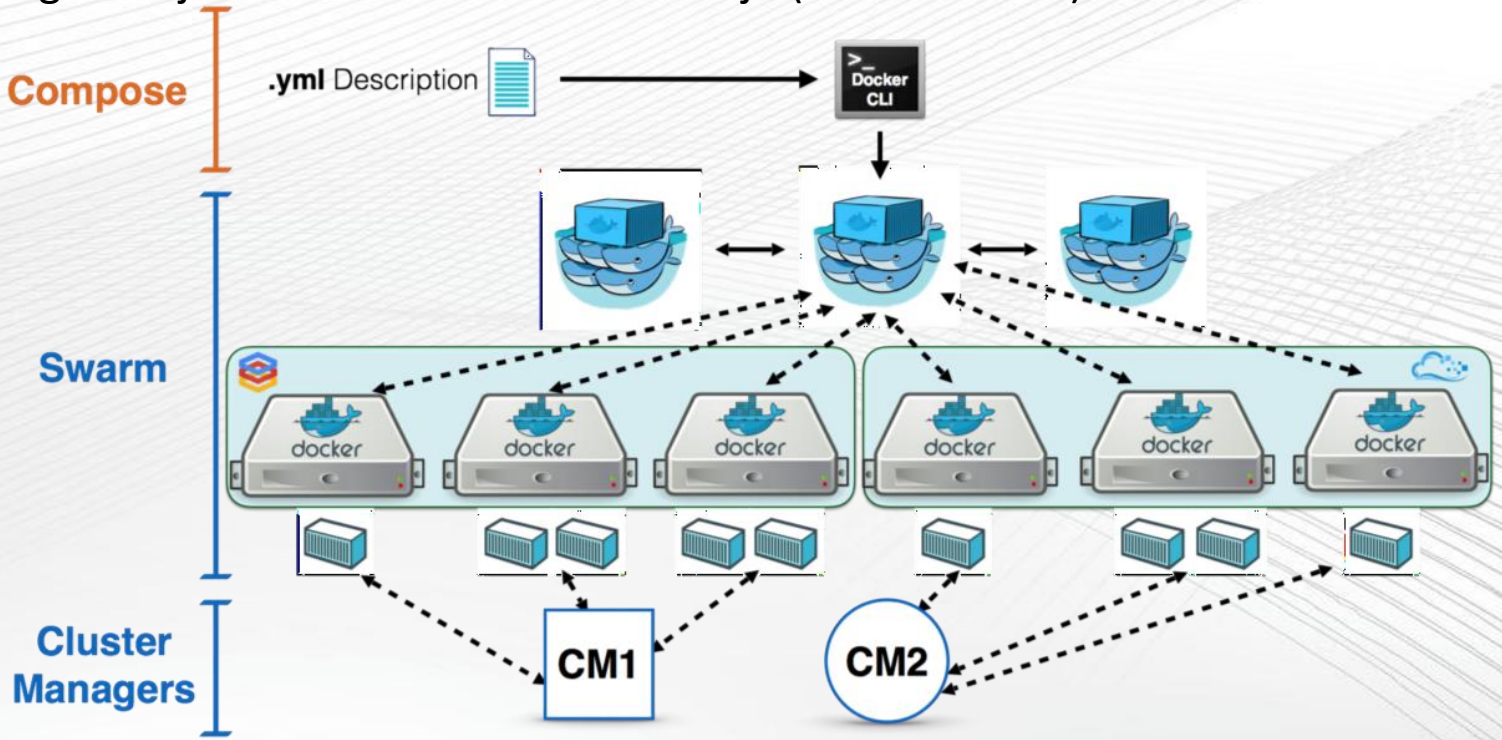
Docker Swarm

- Hasta ahora, todo se está ejecutando en una sola máquina y con una sola réplica por contenedor, lo cual hace que tu aplicación no sea ni lo suficientemente escalable ni tolerante a fallos.
- Vamos a dar un paso más viendo cómo puedes trabajar con aplicaciones en Docker repartidas por varias máquinas de un clúster, utilizando Swarm.
- Swarm, en español enjambre, se trata de una herramienta que nos ayuda a gestionar y orquestar contenedores en un clúster. Gracias a que en Docker también podemos trabajar con este concepto, podremos asegurarnos de que nuestras aplicaciones soportan la demanda, además de ser tolerantes a fallos, teniendo más de una copia del mismo contenedor en más de una máquina o nodo.
- Dentro de este clúster, existirán dos roles: manager y worker. El primero de ellos es el que es capaz de mandar órdenes al resto, para administrar el clúster. Los workers simplemente hospedan los contendores.
- Podemos tener distintas máquinas físicas, virtuales o hacerlo todo en la misma máquina.
- Aunque al día de hoy la alternativa por excelencia de Kubernetes, Swarm es una buena alternativa si no se quiere salir del universo Docker y de las tecnologías que ya conoces.



Docker Swarm

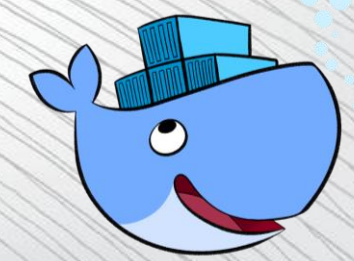
- Docker Swarm se basa en una arquitectura maestro-esclavo.
- Cada clúster de Docker está formado al menos por un nodo maestro (también llamado administrador o manager) y tantos nodos esclavos (llamados de trabajo o workers) como sea necesario.
- Mientras que el maestro de Swarm es responsable de la gestión del clúster y la delegación de tareas,el esclavo se encarga de ejecutar las unidades de trabajo (tasks o tareas).



Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

Docker Swarm

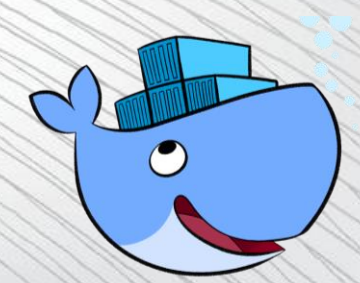
- **¿Por qué se debería poco a poco usar Dockers?**
  - Es open source.
  - Facilita el testing y facilita la tarea.
  - Ahorra tiempo: te evita instalar diferentes softwares para ejecutar una App.
  - Ambiente común para equipos de desarrollos, la misma imagen y contenedor puede por el equipo sin riesgo de fallar en paquetes, librerías, instalaciones de recursos, etc.
  - Es muy sencillo crear y eliminar contenedores.
  - Los contenedores se vuelven muy livianos: permite manejar varios dentro de una misma máquina.
  - Es menos costoso: requiere menos espacio, menos máquinas y menos ordenadores.
  - Da libertad de tener todo en un único lugar lo necesario para hacer correr una app
  - Portabilidad. Al almacenar los contenedores en discos duros, se pueden transportar de un lugar a otro sin problemas.
  - Repositorios Docker. “Banco de imágenes docker” creadas por usuarios a las cuales podemos tener acceso.
  - Se acelera el proceso de mantenimiento y desarrollo gracias a las facilidades para generar copias.
  - Las aplicaciones se ejecutan sin variaciones. Sin importar el equipo ni el ambiente.
  - Facilita las visualizaciones al cliente gracias a que no tiene que instalar nada más que docker en su ordenador.
  - Despliegue en la nube basado en dockers.
  - Facilidad en tener un ambiente de desarrollo y otro de producción basado en contenedores.
  - Es un entorno seguro y no ofrece variaciones.
  - Ideal para el uso de microservicios y para el CI/CD.



Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

¿Jugamos?

https://github.com/adririos98/Curso-Docker



Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/

¡Muchas gracias!

Adrián Hernández Ríos. https://github.com/adririos98 https://www.linkedin.com/in/adrian-hern%C3%A1ndez-r%C3%ADos/