# Docker: una solución al transporte de aplicaciones



**Jueves 28 de Julio, 2016. Universidad Nacional Autónoma de Nicaragua, UNAN, León.**

Mauricio Rigoberto Martínez Romero

Héctor David Guerrero García

Durjan Diomides Alvarado Leiva

José Alberto Alfaro Villatoro

**Universidad de El Salvador**
**Facultad Multidisciplinaria Oriental**

COMPDES 2016

**Jueves 28 de Julio, 2016. Universidad Nacional Autónoma de Nicaragua, UNAN, León.**

# Docker: una solución al transporte de aplicaciones



**Jueves 28 de Julio, 2016. Universidad Nacional Autónoma de Nicaragua, UNAN, León.**

# Objetivo del taller

- Conocer Docker y todos los componentes básicos del mismo. Junto con sus Beneficios.

# ¿Qué es Docker?

Docker containers, envuelve una pieza de software en un sistema de archivos completo que contiene todo lo necesario para **funcionar**: código, tiempo de ejecución, herramientas del sistema, las bibliotecas del sistema - cualquier cosa que se puede instalar en un servidor. **Esto garantiza que el software se ejecutará independientemente de su medio ambiente.**

Estos contenedores de Docker podríamos definirlos como máquinas virtuales ligeras, menos exigentes con los chips y memorias de los equipos donde se ejecutarán.

Las características principales de estos contenedores son la portabilidad, la ligereza y la autosuficiencia:

- **Portabilidad**: El contenedor Docker podremos desplegarlo en cualquier otro sistema (que soporte esta tecnología), con lo que nos ahorraremos el tener que instalar en este nuevo entorno todas aquellas aplicaciones que normalmente usemos.

- **Ligereza:**

El peso de este sistema no tiene comparación con cualquier otro sistema de virtualización más convencional que estemos acostumbrados a usar.

Por poner un ejemplo, una de las herramientas de virtualización más extendida es VirtualBox, y cualquier imagen de Ubuntu que queramos usar en otro equipo pesará entorno a 1Gb si contamos únicamente con la instalación limpia del sistema.

En cambio, un Ubuntu con Apache y una aplicación web, pesa alrededor de 180Mb, lo que nos demuestra un significativo ahorro a la hora de almacenar diversos contenedores que podamos desplegar con posterioridad.

- **Autosuficiencia:**

  Un contenedor Docker no contiene todo un sistema completo, sino únicamente aquellas librerías, archivos y configuraciones necesarias para desplegar las funcionalidades que contenga. Asimismo Docker se encarga de la gestión del contenedor y de las aplicaciones que contenga.

  A parte de ofrecernos un entorno similar a Git para, a base de "capas", controlar cada cambio que se haga en la máquina virtual o contenedor.

Para obtener fluidez Docker extiende **LXC (LinuX Containers), un sistema de virtualización ligero que permite crear múltiples sistemas totalmente aislados entre si sobre la misma máquina o sistema anfitrión.**

Y todo dado que no se emula un sistema operativo completo, sólo las librerías y sistemas de archivos necesarios para la utilización de las aplicaciones que tengamos instaladas en cada contenedor.

# Conociendo la historia de Docker

Docker comenzó como un proyecto interno dentro **dotCloud, empresa enfocado a una plataforma como un servicio (PaaS),** iniciado por **Salomón Hykes.**


Salomón Hykes

Con las contribuciones iniciales de otros ingenieros de dotCloud, incluyendo **Andrea Luzzardi** y **Francois-Xavier Bourlet. Jeff Lindsay** también participó como colaborador independiente.

Docker fue liberado como código abierto en marzo de 2013. **El 13 de marzo de 2014**, con el **lanzamiento de la versión 0.9**, Docker dejó de utilizar LXC como el entorno de ejecución por defecto y lo reemplazó con su propia librería, **libcontainer**, escrito en **Go** (lenguaje de programación).

El **13 de abril de 2015**, el proyecto tenía más de **20.700 estrellas de GitHub** (haciéndolo uno de los proyectos con más estrellas de GitHub, en **20a posición**), más de **4.700 bifurcaciones ("forks")**, y casi **900 colaboradores.**

## Principales contribuyentes de Docker:

- El equipo de Docker
- Red Hat (mayores contribuyentes, aún más que el equipo de Docker en sí)
- IBM
- Google
- Cisco Systems
- Amadeus IT Group

# Arquitectura de Docker

Docker usa una arquitectura **cliente-servidor**. El **cliente** de Docker habla con el **Daemon** de Docker que hace el trabajo de crear, correr y distribuir los contenedores.

Tanto el cliente como el Daemon pueden ejecutarse en el mismo Sistema, o puede conectar un cliente remoto a un daemon de docker.

- Docker Engine
- Docker Client

# Docker Engine

Docker Engine o Demonio Docker es un demonio que corre sobre cualquier distribución de Linux 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).

El usuario nunca interactúa con el demonio directamente, sino que lo hace por medio del cliente.

Es básicamente un motor de contenedores que usa características del Kernel de Linux como espacios de nombres y controles de grupos, para crear contenedores encima del Sistema operativo, y automatizar el despliegue de aplicaciones en estos contenedores.

**Entre sus funciones principales:**

- Creación de imágenes docker.
- Publicación de imágenes en un Docker Registry o Registro de Docker (otro componente Docker ).
- 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, rearrancarla, ver sus logs o sus estadísticas de uso de recursos.

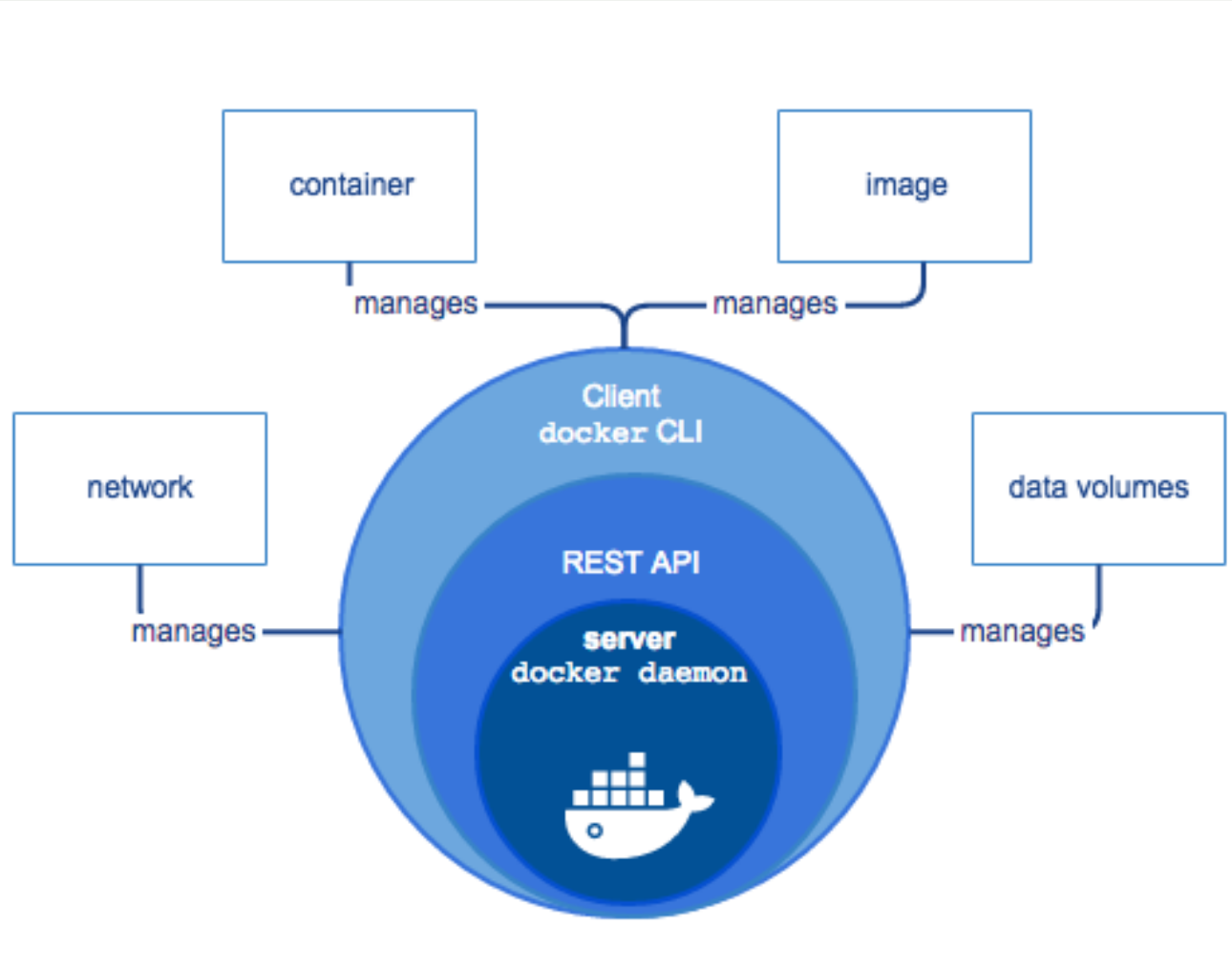
# Docker Client

Binario que constituye la interfaz de usuario entre el cliente y el demonio del servicio. Acepta y procesa las acciones del cliente a través de una serie de comandos.

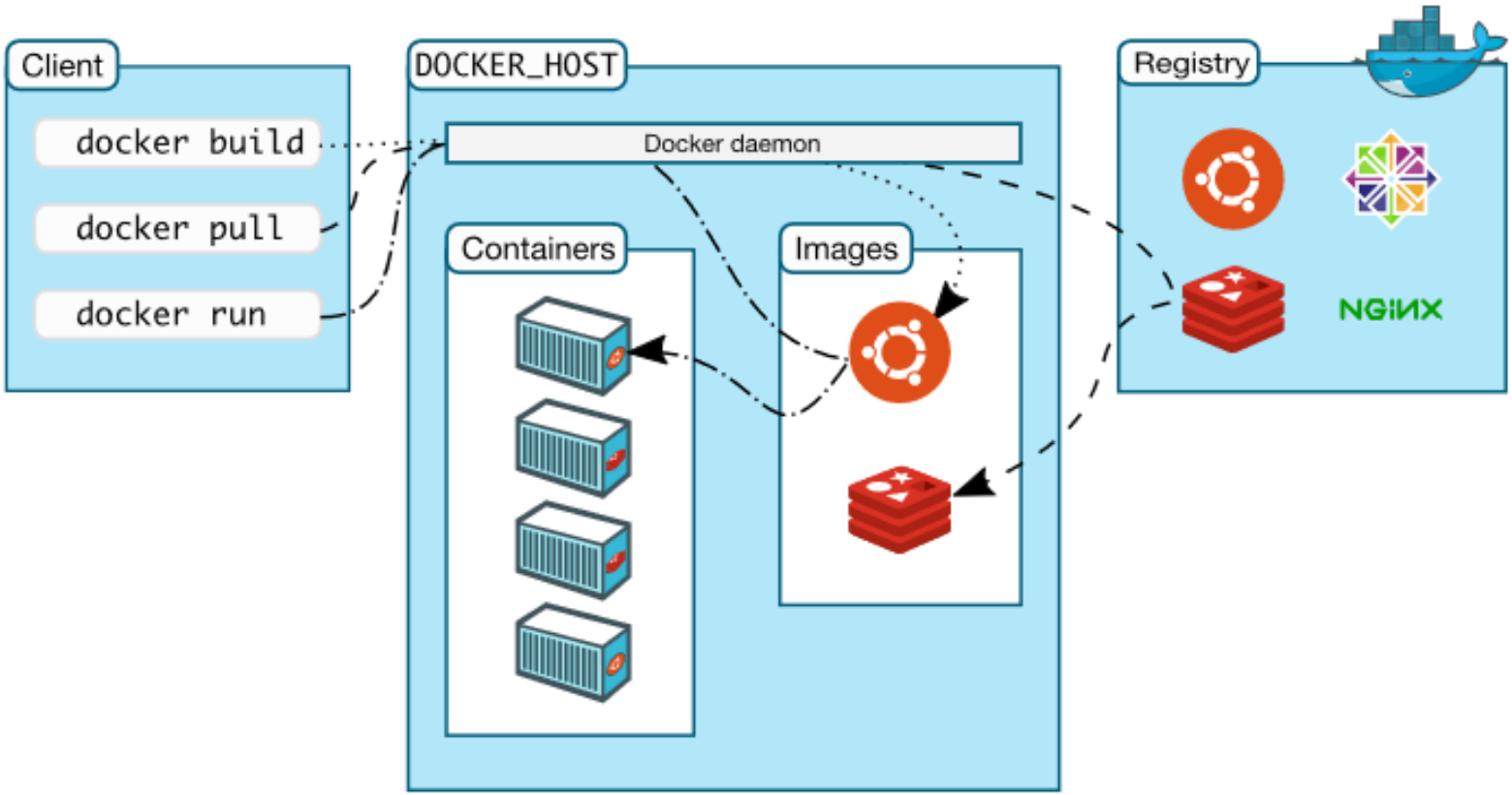
Es cualquier herramienta que hace uso de la api remota del Docker Engine, pero suele hacer referencia al comando docker que hace las veces de herramienta de línea de comandos (cli) para gestionar un Docker Engine.

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.

# Gráfico de la arquitectura de Docker



# Gráfico de la arquitectura de Docker



# Componentes de Docker

- Imágenes de Docker (Docker Images)
- Contenedores de Docker (Docker Containers)
- Registros de Docker (Docker Registries)

# Imágenes de Docker (Docker Images)

Una imagen de Docker es una plantilla de lectura. Por ejemplo, un imagen puede contener un sistema operativo Ubuntu ademas de Apache instalado junto con su aplicación web. Las imagenes se utilizan para crear contenedores Docker.

Docker provee una forma de construir imagenes, o actualizar una existente, ademas de porder hacer uso de otras imagenes creadas por otras personas

Las images es uno de los componentes de construcción de Docker.

# Imágenes de Docker (Docker Images)

Las imágenes se utilizan para crear contenedores, y nunca cambian.

Hay muchas imágenes públicas con elementos básicos como Java, Ubuntu, Apache…etc, que se pueden descargar y utilizar.

# Imágenes de Docker (Docker Images)

Normalmente cuando creas imágenes, partimos de una imagen padre a la que le vamos añadiendo cosas (p.e: una imagen padre con Ubuntu y Apache, que hemos modificado para instalar nuestra aplicación).

Las imágenes se identifican por un ID, y un par nombre-versión, por ejemplo: ubuntu:latest, django:1.6, etc.

# Contenedores de Docker (Docker Containers)

Un contenedor es simplemente un proceso para el sistema operativo que, internamente, contiene la aplicación que queremos ejecutar y todas sus dependencias. La aplicación contenida solamente tiene visibilidad sobre el sistema de ficheros virtual del contenedor y utiliza indirectamente el kernel del sistema operativo principal para ejecutarse.

# Contenedores de Docker (Docker Containers)

Podemos trazar un paralelismo entre el contenedor y una máquina virtual: ambos son sistemas autocontenidos que, en realidad, utilizan un sistema superior para ejecutar sus trabajos. La gran diferencia es que una máquina virtual necesita contener todo el sistema operativo mientras que un contenedor aprovecha el sistema operativo sobre el cual se ejecuta.

# Contenedores de Docker (Docker Containers)

Un contenedor es simplemente un proceso para el sistema operativo que, internamente, contiene la aplicación que queremos ejecutar y todas sus dependencias. La aplicación contenida solamente tiene visibilidad sobre el sistema de ficheros virtual del contenedor y utiliza indirectamente el kernel del sistema operativo principal para ejecutarse.

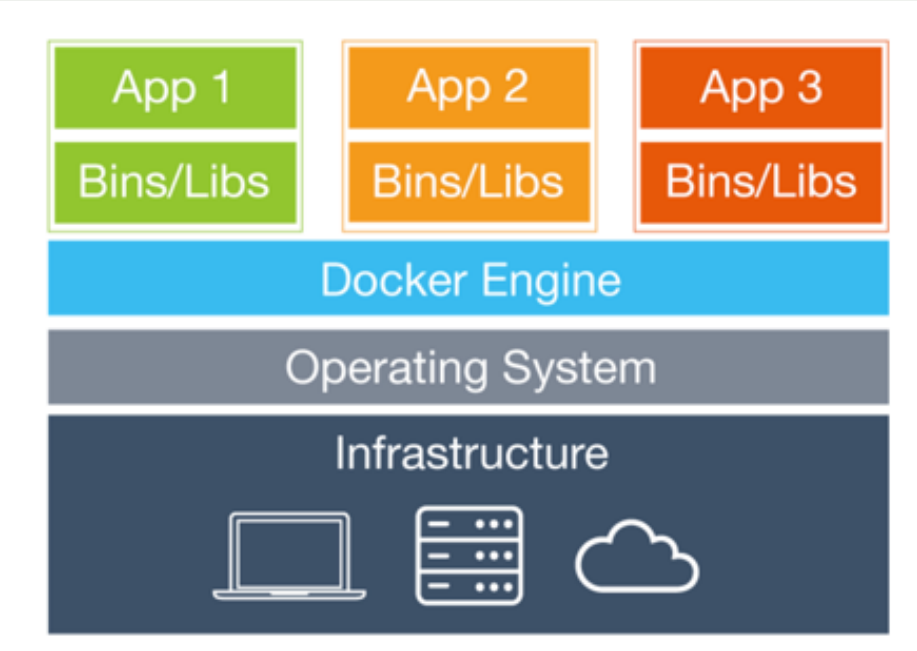
# Registros de Docker (Docker Registries)

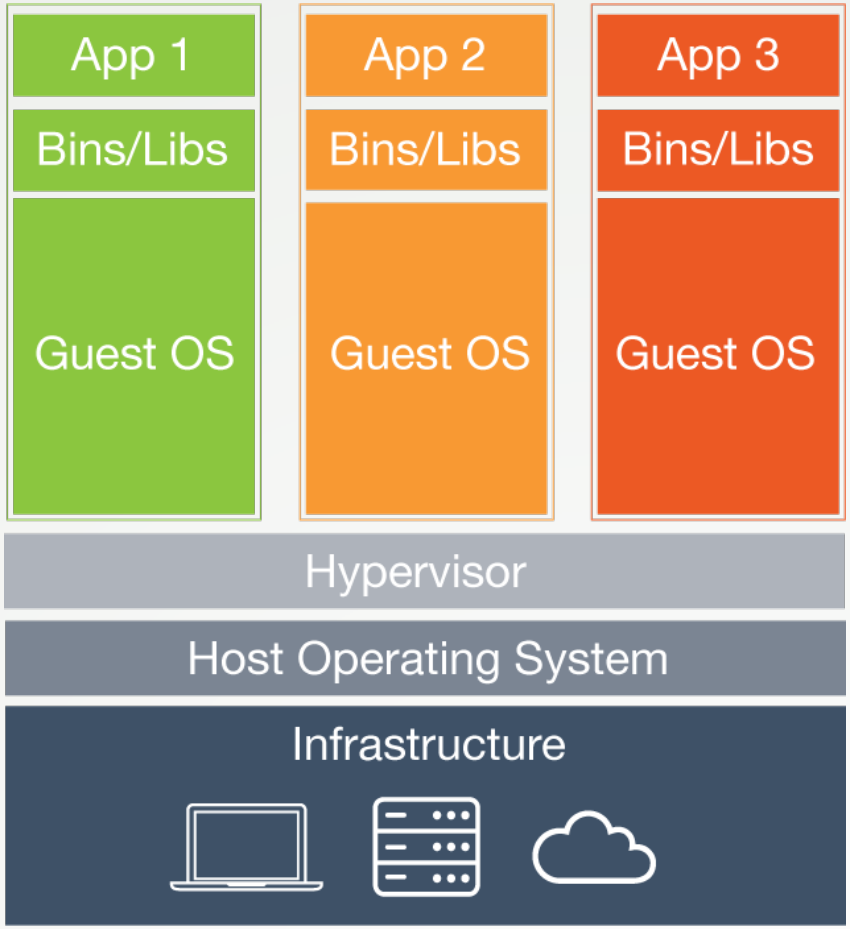
Los registros de Docker guardan las imágenes, estos son repos públicos o privados donde podemos subir o descargar imágenes. El registro público lo provee el Hub de Docker que sirve una colección de imágenes para nuestro uso. Los registros de dockers básicamente son el componente de Distribución de Docker.

# Contenedores vs Máquinas Virtuales

Cuando se trata de comparar los dos tipos de tecnologías se podría decir que Docker y sus contenedores tienen mucho más potencial que las máquinas virtuales.

Los aspectos fuertes y débiles al usar Docker vs VM, distinguiendo categorías como rendimiento, rapidez, portabilidad, seguridad y administración.


App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Engine
Operating System
Infrastructure


App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure

# Rapidez

Docker y sus contenedores son capaces de compartir un solo núcleo y compartir bibliotecas de aplicaciones, esto ayuda a que los contenedores presenten una carga más baja de sistema que las máquinas virtuales.

En comparación con las máquinas virtuales, los contenedores pueden ser más rápidos y consumirán menos recursos siempre que el usuario está dispuesto a pegarse a una única plataforma para proporcionar el sistema operativo compartido.

Una máquina virtual puede tardar hasta varios minutos para crearse y poner en marcha mientras que un contenedor puede ser creado y lanzado sólo en unos pocos segundos.

Para dar un ejemplo claro, los tiempos de inicio y detención de Docker son menores a 50ms, mientras que las máquinas virtuales inician en 30-45 segundos, y se detienen en 10 segundos o menos.

# Portabilidad

Todas las aplicaciones tienen sus propias dependencias, que incluyen tanto los recursos de software y hardware. Los Contenedores Docker aportarán numerosos beneficios en comparación con las tecnologías existentes.

En términos de tecnología, es bastante interesante en escenarios donde ayuda en la promoción de la portabilidad de la nube mediante la ejecución de las mismas aplicaciones en diferentes entornos virtuales esto es muy útil en el ciclo de vida para el desarrollo de software.

Docker es una plataforma abierta para desarrolladores, es un mecanismo que ayuda a aislar las dependencias por cada aplicación mediante la creación de contenedores. Los contenedores son escalables y seguros si los comparamos con el enfoque anterior del uso de máquinas virtuales.

# Seguridad

Una de las ventajas de la utilización de máquinas virtuales es la abstracción a nivel de hardware físico que se traduce en kernels individuales, que limitan la superficie de ataque al hipervisor (el monitor o núcleo de la virtualización).

En teoría, las vulnerabilidades particulares en las versiones de sistemas operativos no se pueden aprovechar para poner en peligro otras máquinas virtuales que se ejecutan en la misma máquina física.

# Administración

Soluciones tales como Docker hacen más fácil la gestión de contenedores, pero muchos clientes todavía encuentran la gestión de contenedores más un arte que una ciencia. Para varios usuarios que han estado trabajando con Docker recientemente y comparten su experiencia y su frustración de la gestión del Docker en un entorno de producción.

# Datos Adicionales

La gran diferencia es que una máquina virtual necesita contener todo el sistema operativo mientras que un contenedor aprovecha el sistema operativo sobre el cual se ejecuta.

Por ejemplo, virtualizar una base de datos sobre una máquina virtual, requiere:

- Una máquina física que aporte el hardware
- Un sistema operativo “Host” sobre esta máquina física
- Un sistema de virtualización o hypervisor que gestione las peticiones al hardware virtual y las ejecute sobre el real
- Un sistema operativo “Guest” bajo el hypervisor. Este sistema debe ser completo ya que no puede obtener recursos del kernel de su Host
- Instalar bajo el sistema “Guest” el motor de base de datos y todas sus dependencias

# Datos Adicionales

En cambio, virtualizar la misma base de datos sobre Docker requiere:

- Una máquina física o virtual
- Un sistema operativo sobre esta máquina
- El motor de Docker instalado en esta máquina
- Un contenedor basado en una imagen que contenga el motor de base de datos y todas sus dependencias

# Datos Adicionales

Las diferencias importantes son:

- El sistema de contenedores no exige una máquina física, mientras que no tiene sentido instalar un hypervisor bajo una máquina virtual (el rendimiento se verá penalizado en extremo). Por lo tanto ganamos versatilidad.

- Cada máquina virtual conlleva virtualizar todo un sistema operativo. En el ejemplo anterior, poner en marcha una una segunda base de datos como réplica implicaría destinar recursos a un segundo sistema operativo, mientras que con Docker simplemente lanzaremos un segundo contenedor mucho más ligero. Por lo tanto aprovechamos mejor los recursos.

# Datos Adicionales

- Arrancar un contenedor cuesta, en tiempo, lo mismo que arrancar un proceso en el sistema. En cambio, al arrancar una máquina virtual debemos esperar a que arranque también el sistema operativo subyacente. Por lo tanto ganamos agilidad.

- También es importante destacar el tamaño de las imágenes que podemos conseguir. La imagen oficial de Debian 7.7 en Docker Hub ocupa 85.1 MB. La imagen de Nginx ocupa 91.4 MB. ¿Cuánto ocuparían las imágenes de Virtualbox correspondientes? Nos podemos hacer una idea de lo ágil que resulta realizar cualquier operación de aprovisionamiento o recolocación de contenedores.

# Ventajas de Docker

- La principal ventaja del uso de Docker es que los desarrolladores pueden centrarse en su código sin tener que preocuparse de si dicho código funcionará en la máquina en la que se ejecutará.
- Por ejemplo, si un desarrollador tiene en su ordenador instalado una versión de JAVA 8 y programa una función específica para esa versión, y otro desarrollador tiene instalado en su máquina la versión JAVA 7, cuando el primer desarrollador quiera probar su programa en la máquina del desarrollador que tiene una versión inferior, la aplicación fallará.

# Ventajas de Docker

- Y aquí es donde aparece Docker, el primer desarrollador creará un contenedor de Docker con la aplicación, la versión 8 de Java y el resto de recursos necesarios, se lo pasa al usuario con la versión inferior, y éste con el Docker instalado en su máquina, podrá ejecutar la aplicación a través del contenedor sin tener que instalar absolutamente nada más.
- Estos contenedores tienen unas características principales:
- El contenedor se puede extender en cualquier sistema que soporte la tecnología.

# Ventajas de Docker

- Es ligero. Ocupa mucho menos que cualquier otro sistema de virtualización.
- Los contenedores que se ejecutan en una misma máquina comparten Sistema Operativo.
- Además, existen dos componentes esenciales:
- Imágenes: Plantillas con unos sistemas y configuraciones que podríamos utilizar de base para fabricar nuevos contenedores o entornos.
- Repositorios: Contienen imágenes creadas por los usuarios y pueden ser públicos o privados.

# Ventajas de Docker

- En resumen, si utilizamos Docker en nuestros obtendremos estos beneficios, algo que, sin duda, cualquier desarrollador agradecerá en el día a día de sus proyectos:
- Se acelera el proceso de mantenimiento y desarrollo, de manera que realizar una copia de los sistemas que están en producción y ejecutarlos en otro equipo, sea una tarea sencilla.
- Las aplicaciones se ejecutan tal y como fueron creadas y no importa el equipo ni el ambiente (pruebas o producción).

# Ventajas de Docker

- En un escenario típico el cliente instalaría y configuraría MySQL y posteriormente la aplicación. Con Docker, ejecutando el contenedor estaría todo listo.

- En DevOps, tanto los desarrolladores como los administradores de sistema pueden probar aplicaciones en un entorno seguro y exactamente igual en todos los casos.