CFGS ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED

BIENVENIDOS!

Manuel Domínguez.

mftienda@gmail.com

@mafradoti

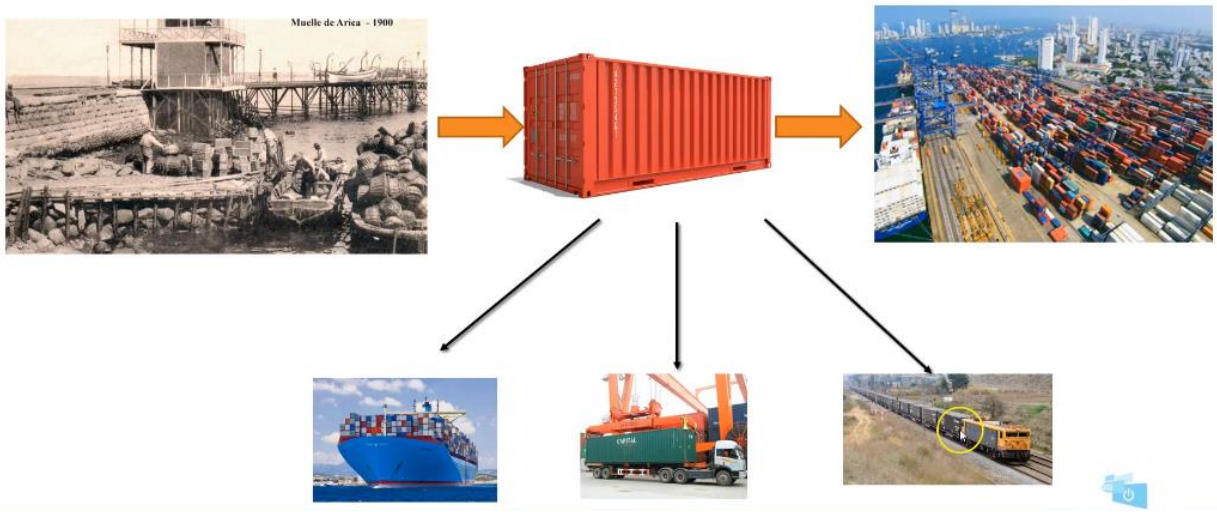
www.linkedin.com/in/mftienda

https://github.com/mftienda

Ud.- ¿Qué es docker?

Índice

1.- Introducción



Si realizas una aplicación en tu ordenador, ¿funcionará en otro ordenador?

2.- Microservicios

Un **microservicio** es un componente de software **que implementa una función** o tarea específica.

Esto nos permite dividir una aplicación grande en un **conjunto de microservicios.**

Se **despliegan de manera independiente**.

Pueden **desarrollarse utilizando diversos lenguajes de programación y distintas BD.**

Se comunican a través de API bien definidas.

2.- Microservicios

API (Application Programming Interface): son un conjunto de protocolos y funciones que se utilizan para comunicar aplicaciones. Nos puede servir para conectarnos con el SO, Bases de datos, etc.

Ejemplo: Cuando compramos una entrada de cine a través de una plataforma y se conecta con nuestro banco para pagar, estamos utilizando la API de nuestro banco.

Ejemplo: Cuando en nuestra aplicación hacemos uso de Google Maps, para localizar una dirección, estamos utilizando la API de Google.

2.- Microservicios

API REST (Interfaz de aplicaciones para transferencia de datos): se ha convertido en el estándar de intercambio y manipulación de datos en los servicios de internet. **Utiliza el protocolo HTTP.**

Ejemplo: Netflix

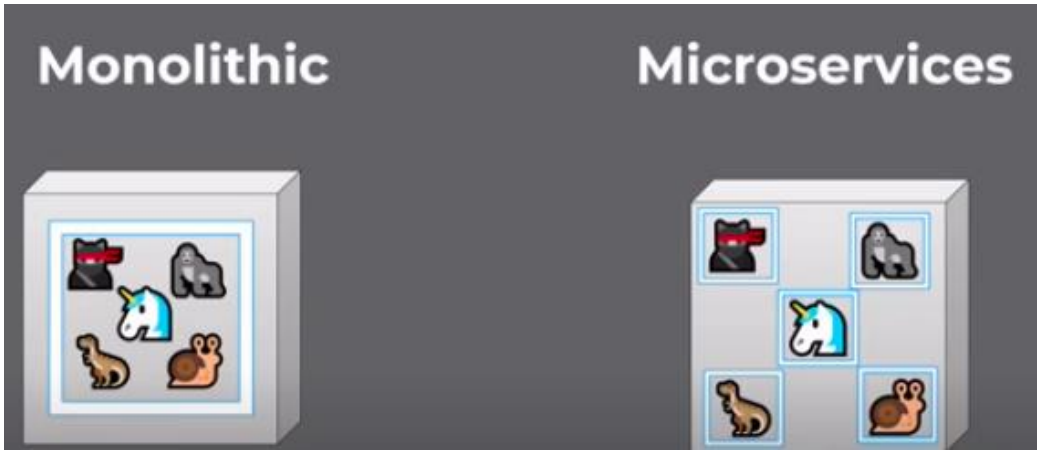
API REST FULL: Tiene más funcionalidades. No solo la de obtener datos, sino que también puede introducir y actualizar datos. Lo utilizan los programadores.

2.- Microservicios

Aplicaciones monolíticas frente a microservicios

Monolitica: Todos los componentes o funcionalidades en una unidad.

Microservicios: Cada componente como una unidad independiente.



Emojis: representan las distintas funcionalidades

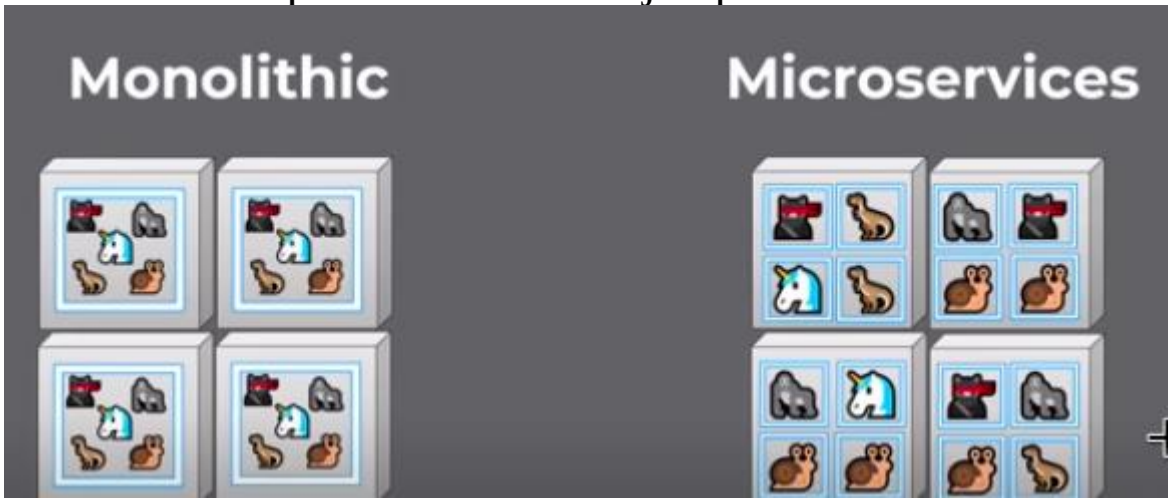
2.- Microservicios

Aplicaciones monolíticas frente a microservicios

Escalar la aplicación. Ha tenido mucho éxito y necesitamos escalarla.

Monolítica: Replicamos la aplicación completa en 4 servidores.

Microservicios: Replicamos los microservicios en 4 servidores, pero en cada servidor replicamos los microservicios que necesitamos. Ejemplo: caracol



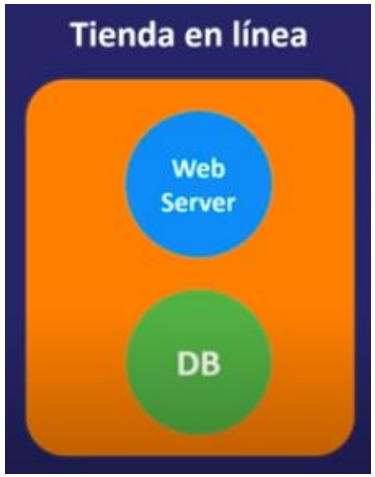
2.- Microservicios

Aplicación monolítica: Tienda online

Nuestra aplicación está compuesta por un solo programa que usa una base de datos.

Servidor web: Para responder a las solicitudes. Tiene 4 funcionalidades: Pedidos, Pagos, inventarios y clientes.

DB: Base de datos para almacenar la información.



2.- Microservicios

Aplicación monolítica: Tienda online

Imaginemos que tenemos que **escalar la aplicación** porque ha tenido mucho éxito.

Colocamos tres servidores repartidos por todo el mundo, para distribuir la carga de trabajo.

Europa, EEUU y América Sur

2.- Microservicios

Aplicación monolítica: Tienda online

Problemas:

1.- Hemos replicado toda la aplicación, cuando tal vez haya funciones que no sean necesarias replicar porque no tenga tantas solicitudes. Por ejemplo: clientes No y Pedidos sí.

2.- Si cae un servidor, se cae todos los componentes. Podemos dejar a Europa sin tienda virtual.

2.- Microservicios

Aplicación microservicios: Tienda online

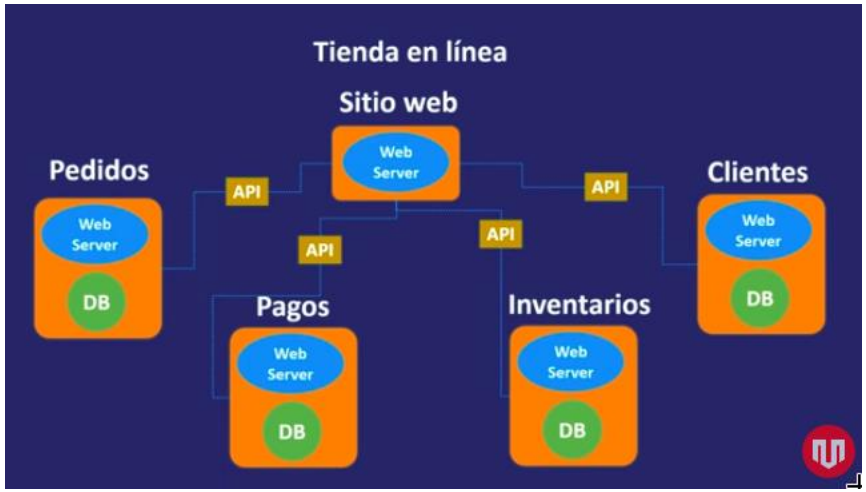
Solución: Separamos la aplicación en componentes de forma que cada componente funcione como un servicio aislado.

Ejemplo: Servicio-usuarios (Altas, modificaciones y bajas de clientes)

2.- Microservicios

Aplicación microservicios: Tienda online

Solución: Cada servicio tiene su propio servidor web y su propia base de datos. A través de API podemos comunicar el servidor web principal (Fronted) con los microservicios y extraer la información que necesitamos.

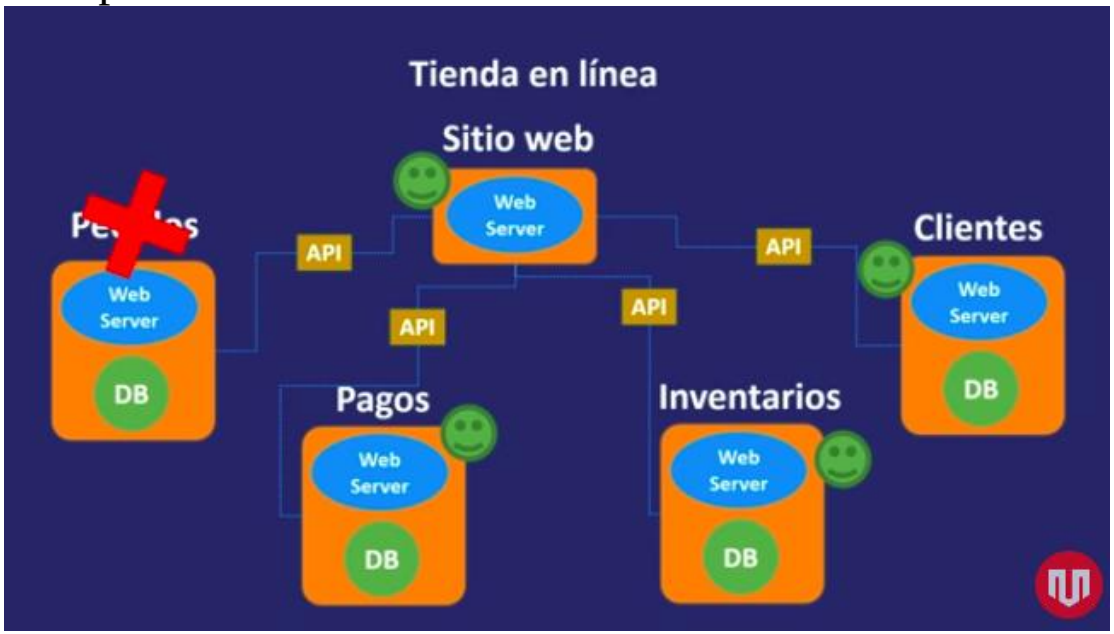


2.- Microservicios

Aplicación microservicios: Tienda online

Ventajas:

1.- Si nos cae un microservicio, por ejemplo Pedidos, el resto de microservicios sigue funcionando. Son independientes.



2.- Microservicios

Aplicación microservicios: Tienda online

Ventajas:

2.- A la hora de escalar, solo necesitamos escalar el servicio que está siendo saturado.

Por ejemplo: Inventarios

2.- Microservicios

Aplicación microservicios: Tienda online

Ventajas:

3.- Cada servicio puede estar implementado utilizando una tecnología diferente: Java, .Net, node.js etc. .

2.- Microservicios

En resumen, las ventajas son:

Cada servicio se puede desarrollar, implementar, operar y escalar sin afectar el funcionamiento de otros servicios.

Los servicios no necesitan compartir ninguno de sus códigos o implementaciones con otros servicios.

Cualquier comunicación entre componentes individuales ocurre a través de API bien definidas.

Los inconvenientes son:

Tener muchos microservicios requiere más tareas de administración.

Por ejemplo: es difícil saber en qué host está corriendo ciertos servicios, ya que pueden estar repartidos entre varios servidores

2.- Microservicios

Unas de las empresas que usan microservicios

Netflix es la empresa que más popularizó la arquitectura de microservicios. Ahora tienen miles de servicios que trabajan juntos para que puedas mirar la última serie en *streaming*.

Pero también Ebay, Amazon, Facebook, Uber y muchas empresas más están usando microservicios. Muchas más están migrando su monolito a una arquitectura de microservicios.

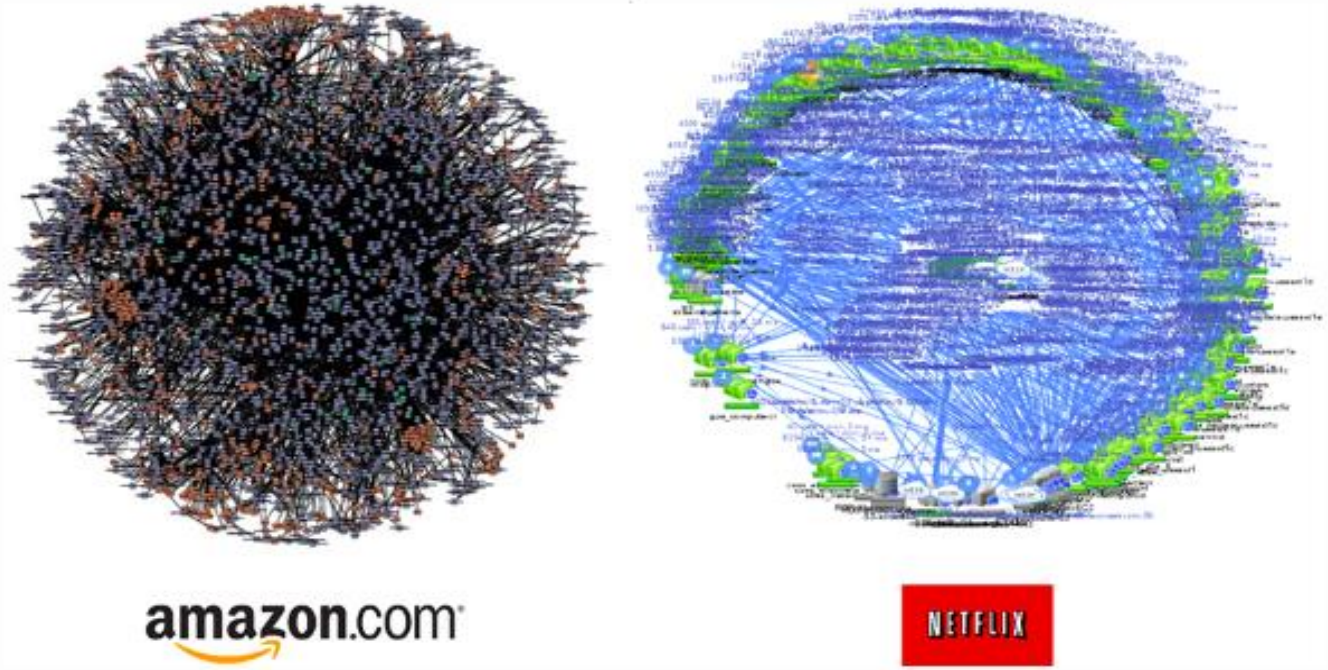


Diagrama de la arquitectura de microservicios de Amazon y de Netflix

2.- Microservicios

Cuestión:

Si tenemos una pequeña aplicación, qué arquitectura utilizaríamos:

Arquitectura monolítica

Arquitectura de microservicios.

3.- Contenedores

Los **contenedores** son la solución al desarrollo de **microservicios.**

Podemos **definir un contenedor** como: una **unidad estándar de software que empaqueta el código** junto a todas de sus dependencias para que el servicio o aplicación se ejecute de forma rápida y fiable de un entorno informático a otro.

La idea de contenedor aparece a finales de los años 90. El kernel de Linux ya tenía implementada las características para el desarrollo de contenedores.

3.- Contenedores

Versátiles:
Del orden de los MB.
Tienen todas las dependencias que necesitan para funcionar correctamente.
Funcionan igual en cualquier lado**:** Nos garantiza que tu aplicación funcionará en cualquier ordenador.

Eficientes:
Solo se ejecutan **procesos,** no un SO completo.
Permite crear máquinas virtuales pero sin virtualizar la capa del S.O.
Se levanta rápidamente.

Aislados:
En principio, lo que ocurra dentro del contenedor no afecta a otro contenedor.

4.- Contenedores contra máquinas virtuales

Wikipedia: una máquina virtual es un software que simula un sistema de computación y puede ejecutar programas como si fuese una computadora real.

“un ordenador corriendo dentro de otro ordenador”

Problemas:

Pesadas: Del orden de GB.

Administración costosa: Actualizaciones, seguridad, etc

Lentas: Hay que arrancar el SO de la MV para que la aplicación funcione.

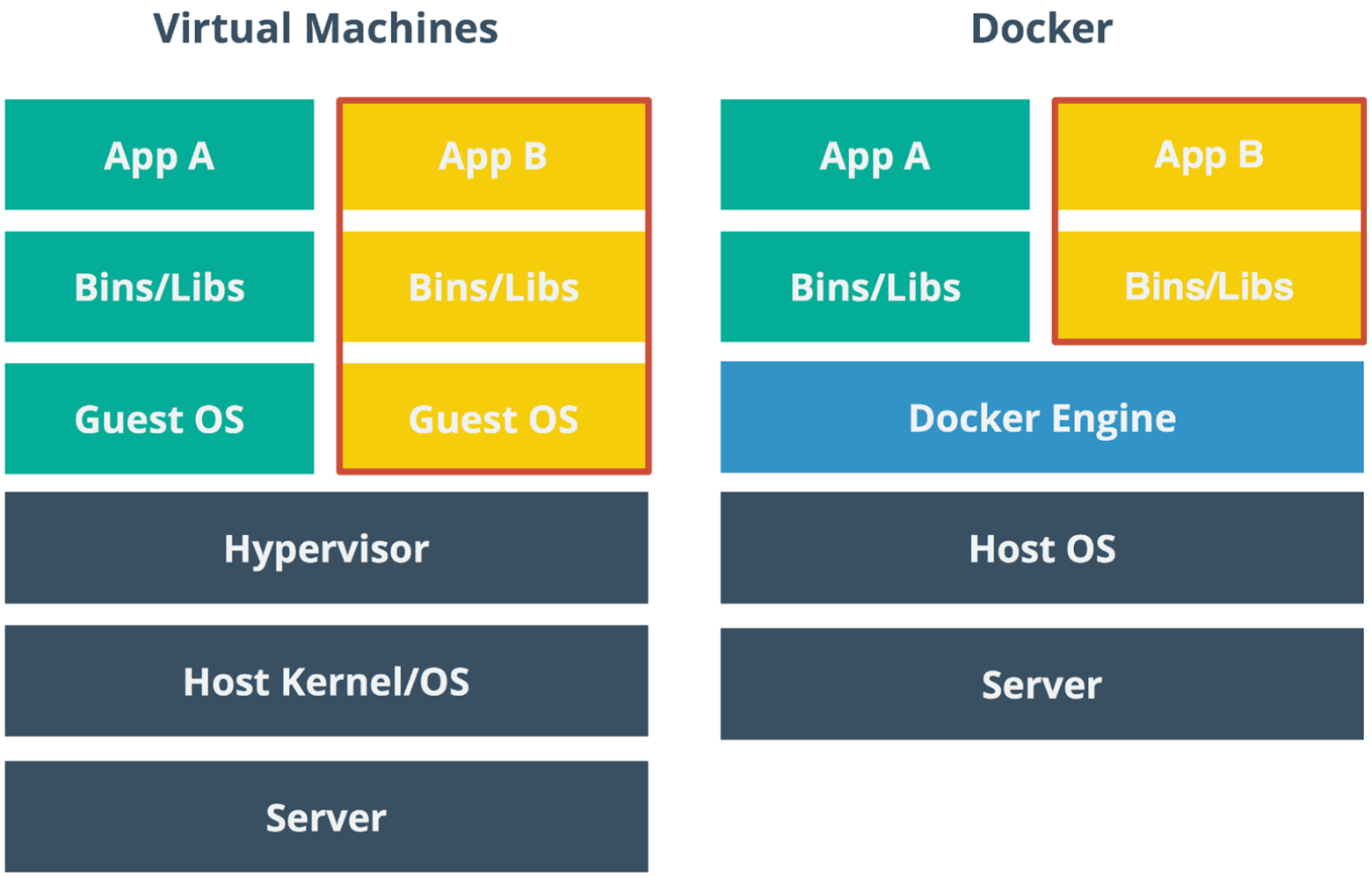
4.- Contenedores contra máquinas virtuales

Desarrollo de aplicaciones

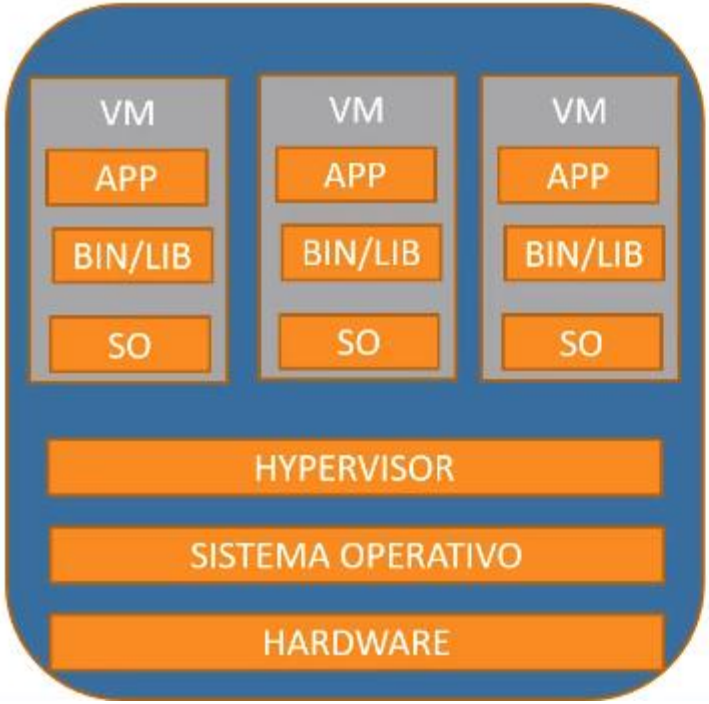
Máquinas virtuales: Es difícil simular un entorno de producción.
Debemos tener las mismas versiones, actualizaciones, etc, en los distintos entornos.

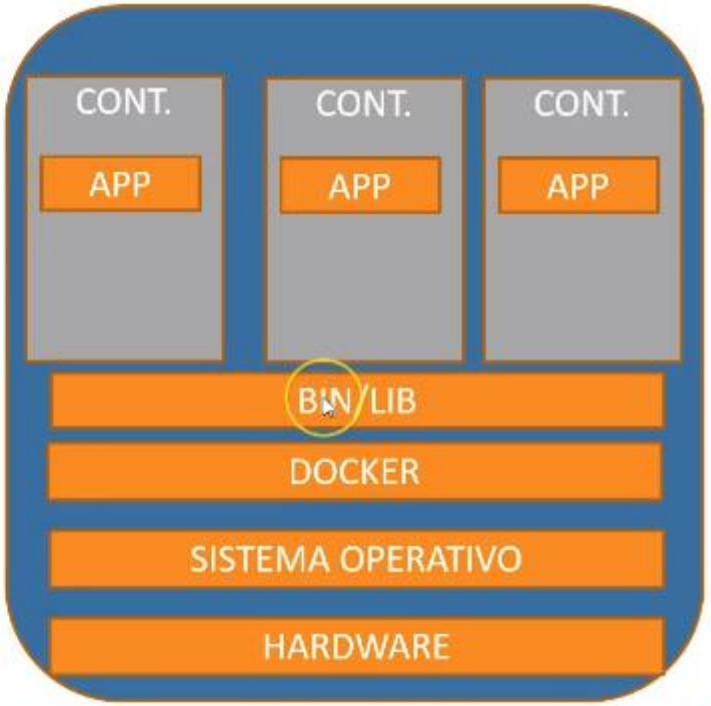
Contenedores: Es fácil tener un entorno exactamente igual en desarrollo, que en pruebas, que en producción.
En el contenedor se enjaula todo lo que se necesita, independiente del SO anfitrión.

4.- Contenedores contra máquinas virtuales



4.- Contenedores contra máquinas virtuales





5.- Despliegue de contenedores en la nube

Google Kubernetes Engine (GKE) es la plataforma empresarial de Google

Azure Container Service es la plataforma empresarial de Microsoft.

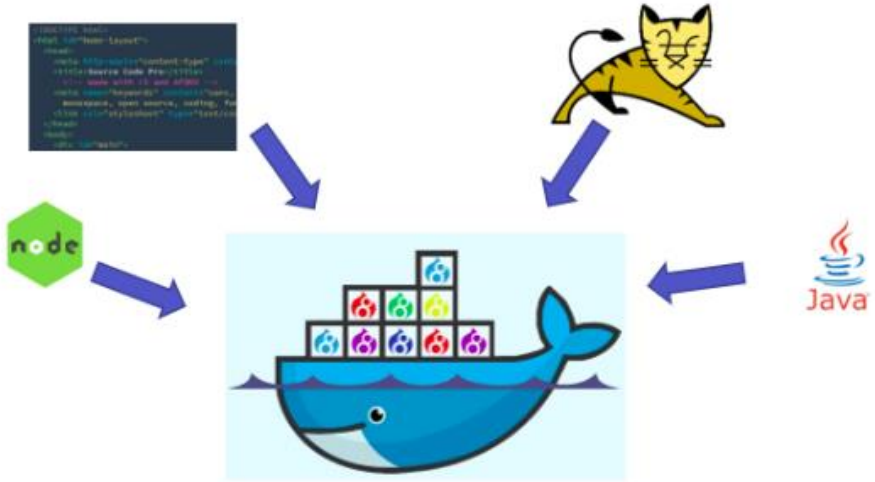
Amazon Elastic Compute Cloud (Amazon EC2) es la plataforma empresarial de Amazon.

6.-¿Qué es docker?

Docker es sin lugar a dudas, la solución de contenedores más utilizada en todo el mundo. Open-source.
Ha conseguido s**implificar al máximo el proceso de crear contenedores, generar imágenes, y compartirlas.**

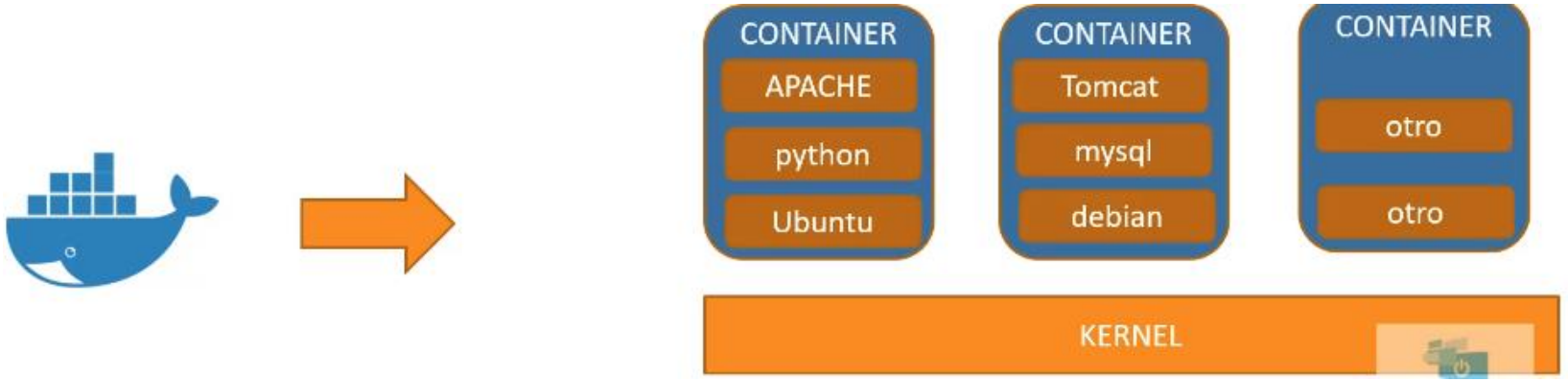
Metemos en un contenedor, todo lo que la aplicación necesita para funcionar.

Docker format



6.-¿Qué es docker?

Estos contenedores comparten la capa de kernel del SO anfitrión para acceder a los recursos de la máquina.



6.-¿Qué es docker?

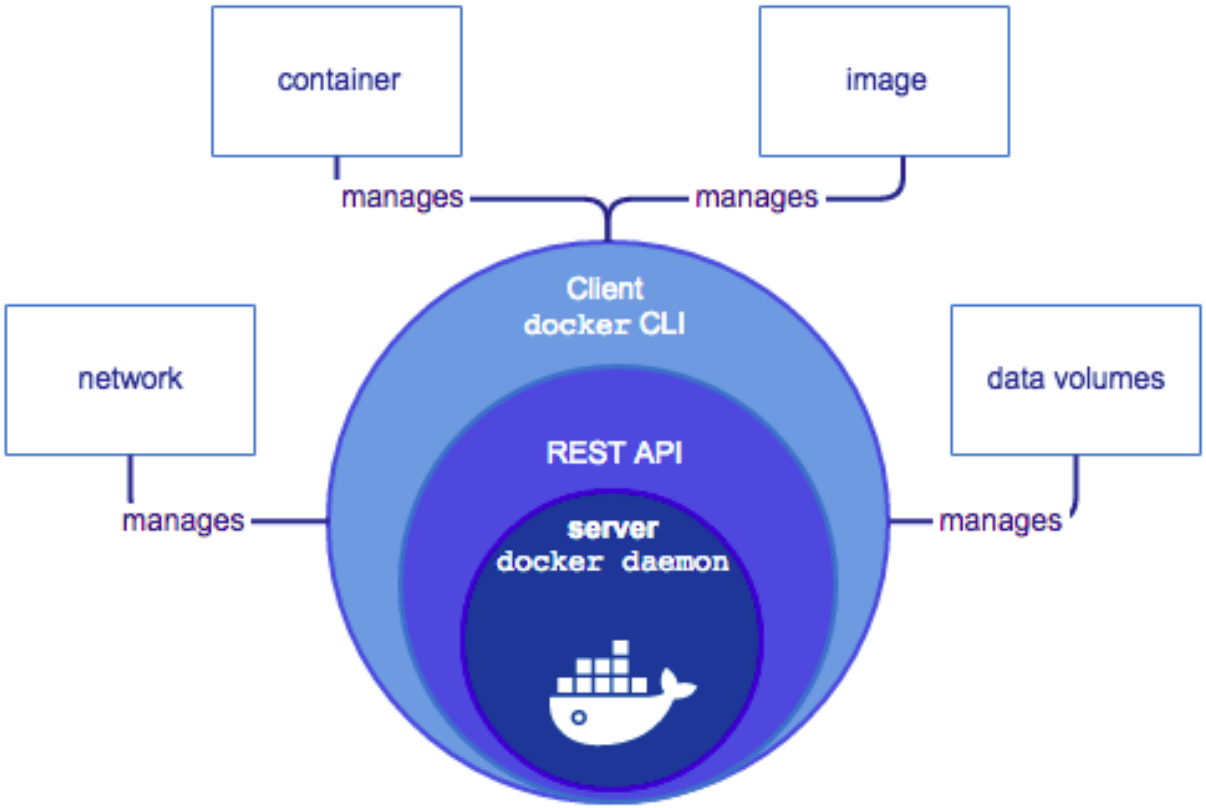
- Poner en un contenedor un determinado entorno y que este pueda ser usado en cualquier plataforma si tener que cambiar nada

6.-¿Qué es docker?

Componentes principales de Docker

- Docker Engine: es la aplicación cliente-servidor que contiene los siguientes elementos
  - Un servidor que se ejecuta como un demonio en el sistema (dockerd)
  - Un interfaz REST API que sirve de acceso para ejecutar comandos en el servidor
  - Un cliente que utiliza el usuario para interactuar con el servidor(docker)
- Image: es el paquete de software ejecutable que contiene todo lo necesario para ejecutar una aplicación
- Container: es una instancia en ejecución de una imagen.
- Network: los contenedores se comunican con otros dockers o con recursos externos a través de la red:
- Data Volume: permiten almacenar datos de forma persistente.

6.-¿Qué es docker?



6.-¿Qué es docker?

Docker **necesita del kernel de Linux.**

Ahora bien, también se puede instalar en Windows y MAC, pero previamente hay que virtualizar el kernel de Linux.

Empresas que colaboran

Resumen

1.- ¿Qué es un microservicio?

2.- ¿Qué es una API Rest?

3.- ¿Qué es un contenedor?

4.- Diferencias de los contenedores con las máquinas virtuales.

5.- ¿Qué es Docker?

Sugerencias/mejoras del tema

Sugerencias /mejoras del tema

Dudas

Referencias

- Sebastián Gómez, Albert Coronado, Guido Vilariño, #CafeConRivas, https://robertoorayen.eu/
- docker.com
- Apasoft Training (