Docker

1.1. Introducción. Docker para desarrolladores

Su uso **aumenta la productividad** de los desarrolladores y les permite **empaquetar y distribuir sus aplicaciones** y todas sus dependencias en contenedores que pueden ser desplegados con gran facilidad en todo tipo de sistemas operativos, servidores, nubes públicas o privadas etc.

Ejecutaremos contenedores, los configuraremos , crearemos nuestros propios contenedores y para finalizar realizaremos un flujo de despliegue continuo usando contenedores

```
# Esto es un comentario sobre el comando que se va a ejecutar a continuación. NO DEBE SER COPIADO
> echo "Esto si que es un comando válido. Debe ser copiado"
```

1.1 ¿Qué es Docker?

Docker es una tecnología de virtualización "ligera" cuyo elemento básico es la utilización de contenedores en vez de máquinas virtuales y cuyo objetivo principal es el despliegue de aplicaciones encapsuladas en dichos contenedores.

En esa **evolución** nos podemos encontrar, de manera general y simplificada, con tres grandes pasos:

- **Arquitectura de un único servidor**
- **Virtualización**
- **Contenedores**

A continuación describiremos estos tres pasos haciendo especial hincapié en sus ventajas e inconvenientes.

1. Un único servidor

Inicialmente, allá por la prehistoria de la informática las aplicaciones se desplegaban en un único servidor siguiendo un esquema similar al que podemos ver en la imagen:

Figura 1: Juan Diego Pérez Jiménez. Arquitectura de Servidor Físico (Dominio público)

Este enfoque tuvo su momento pero tenía varias **LIMITACIONES:**

- Era un enfoque de **costes elevados** porque era necesaria un máquina de precio elevado.
- El despliegue de aplicaciones era un **proceso lento** que podía suponer en algunos casos una parada del servicio.
- El **escalado de las aplicaciones era costoso y complicado**. Si nuestra máquina llegaba un momento que se quedaba corta no había más remedio que sustituirla por otra más potente.
- La **migración a otro sistema era un proceso complicado**. La configuración del nuevo servidor, de su sistema operativo y de todas las dependencias tenía que ser compatible. Esto era algo complicado de gestionar y en algunos casos difícil de conseguir.
- En muchos momentos, aquellos en los que el servidor no se utilizaba aprovechando su potencia, se estaban **desperdiciando recursos.**
- Había mucha **dependencia del fabricante** del servidor.

2. Virtualización

Con el tiempo y para superar las limitaciones del modelo de un único servidor la tecnología evolucionó hacia servidores con características de virtualización. De una manera simplificada

podríamos decir que para desplegar aplicaciones nos encontrábamos con arquitecturas similares a las siguientes:

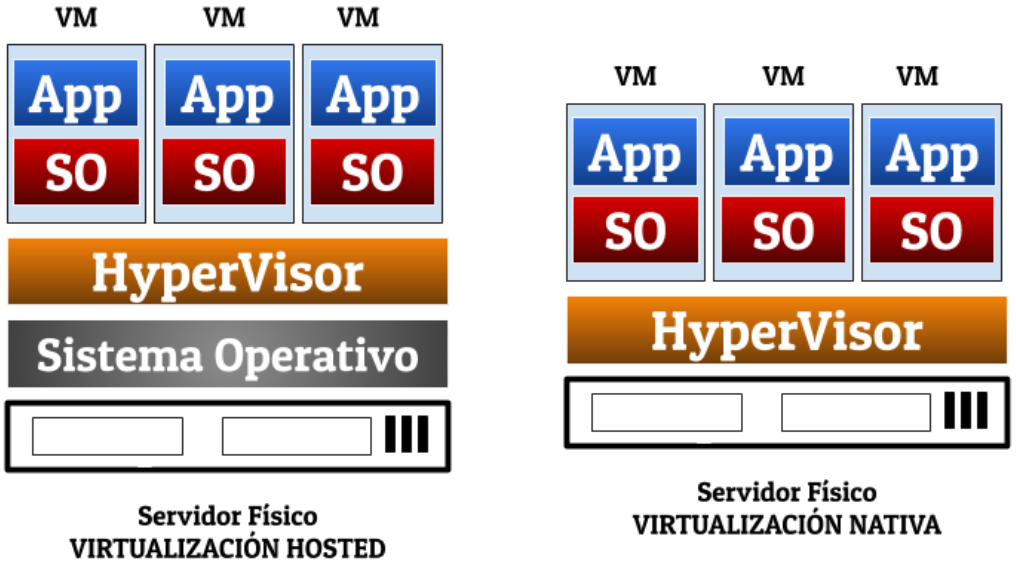


Figura 2: Juan Diego Pérez Jiménez. Arquitecturas basadas en virtualización. (Dominio público)

Estos enfoques tenían una serie de **BENEFICIOS** que derivaban principalmente de superar las limitaciones del modelo de servidor único. A saber:

- Hay un **mejor aprovechamiento de los recursos**. Un servidor grande y potente se puede compartir entre distintas aplicaciones.
- Los **procesos de migración y escalado no son tan dolorosos**, simplemente le doy más recursos a la máquina virtual dentro de mi servidor o bien muevo la máquina virtual a un nuevo servidor, propio o en la nube, más potente y que también tenga características de virtualización.
- Esto además hizo que aparecieran **nuevos modelos de negocio en la nube** que nos permiten en cada momento tener y dimensionar las máquinas virtuales según nuestras necesidades y pagar únicamente por esas necesidades.

No obstante este enfoque también tiene algunos **INCONVENIENTES**:

- Todas las máquinas virtuales siguen teniendo su propia memoria RAM, su almacenamiento y su CPU que será aprovechada al máximo...o no.
- Para arrancar las máquinas virtuales tenemos que arrancar su sistema operativo al completo.
- La portabilidad no está garantizada al 100%.

3. Contenedores

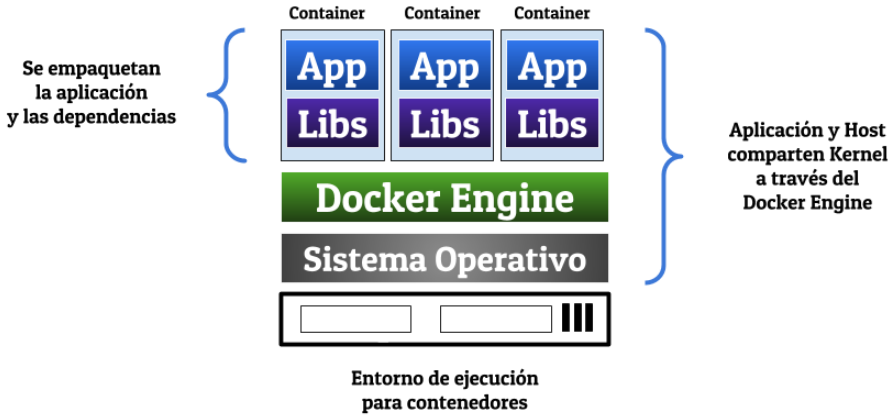


Figura 3: Juan Diego Pérez Jiménez. Entorno de ejecución basado en contenedores (Dominio público)

Y sus principales características son las siguientes:

- Los contenedores utilizan el **mismo Kernel Linux** que la máquina física en la que se ejecutan gracias a la estandarización de los los Kernel y a características como los Cgroups y los Namespaces. Esto **elimina la sobrecarga** que en las máquinas virtuales suponía la carga total del **sistema operativo invitado.**
- Permiten **aislar las distintas aplicaciones** que tenga en los distintos contenedores (salvo que yo estime que deban comunicarse).
- **Facilitan la distribución de las aplicaciones** ya que éstas se empaquetan junto con sus dependencias y pueden ser ejecutadas posteriormente en cualquier sistema en el que se pueda lanzar el contenedor en cuestión.
- Se puede pensar que se añade una capa adicional el **Docker Engine**, pero esta capa **apenas añade sobrecarga** debido a que se hace uso del mismo Kernel.

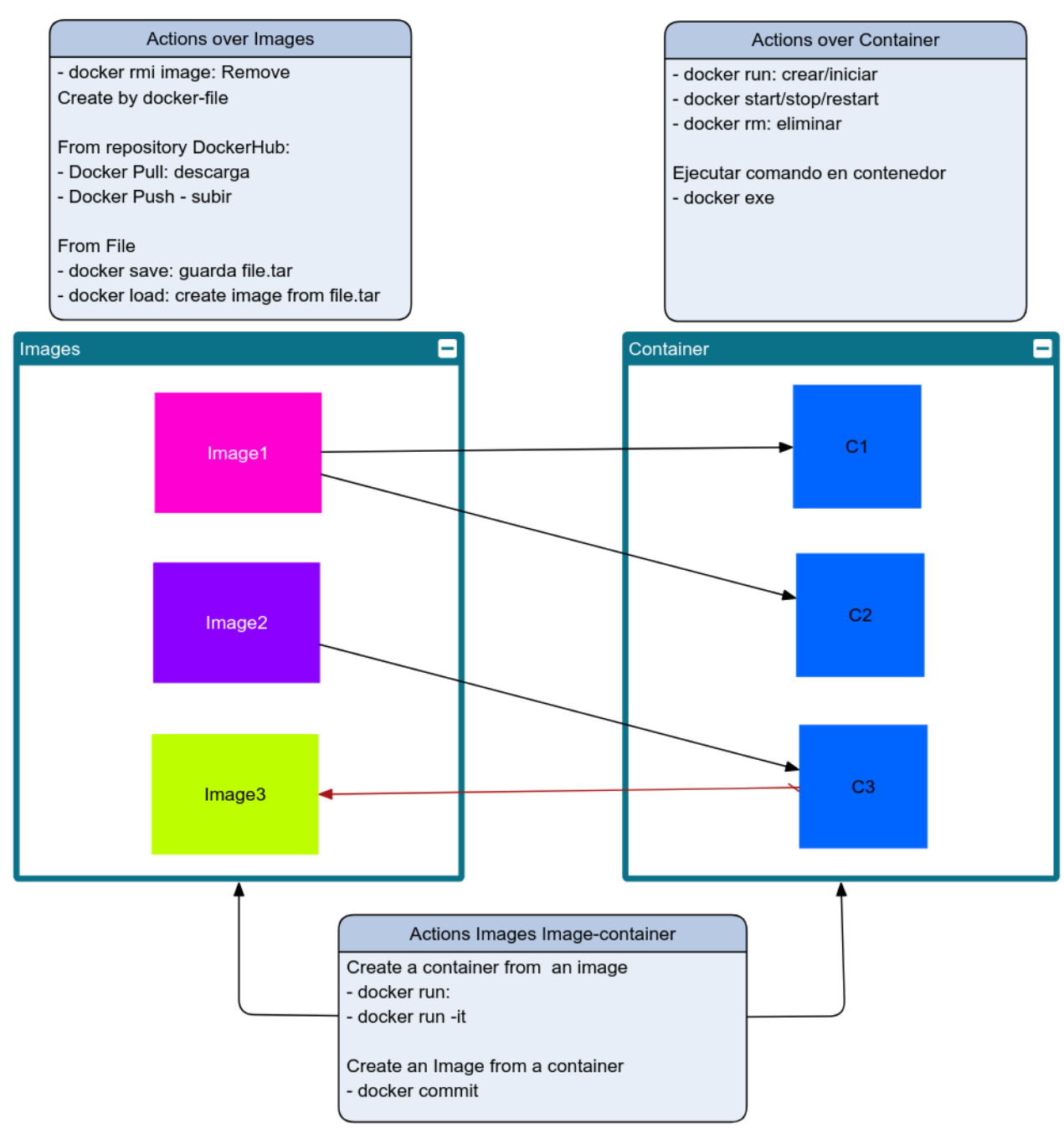
Este enfoque por lo tanto aporta los siguientes **BENEFICIOS:**

- Una **mayor velocidad de arranque,** ya que prescindimos de la carga de un sistema operativo invitado. Estamos hablando de apenas segundos para arrancar un contenedor (a veces menos).
- Un **gran portabilidad,** ya que los contenedores empaquetan tanto las aplicaciones como sus dependencias de tal manera que pueden moverse a cualquier sistema en el que tengamos instalados el Docker Engine, y este se puede ser instalado en casi todos, por no decir todos.

- Una **mayor eficiencia** ya que hay un mejor aprovechamiento de los recursos. Ya no tenemos que reservar recursos, como hacemos con las máquinas virtuales, sin saber si serán aprovechados al máximo o no.

Aunque como todo en esta vida, la tecnología de contenedores tiene algún **INCONVENIENTE:**

- Los contenedores son **más frágiles que las máquinas virtuales** y en ocasiones se quedan en un estado desde el que no podemos recuperarlos. No es algo frecuente pero ocurre y para eso hay soluciones como la orquestación de contenedores que es algo que queda fuera del alcance de este curso que está más orientado a desarrolladores que a sistemas.



Glosario:

Docker Engine

Aplicación cliente-servidor que consta de tres componentes: un servicio **dockerd** para la ejecución de los contenedores, un **API** para que otras aplicaciones puedan comunicarse con ese servicio y una aplicación de línea de comandos **docker cli** que para gestionar los distintos elementos (contenedores, imágenes, redes, volúmenes etc..)

Docker Hub

Registro de repositorios de imágenes de la empresa Docker Inc. Accesible a través de la URL https://hub.docker.com/