Docker

Administración de Sistemas

Unai Lopez Novoa

unai.lopez@ehu.eus

eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Contenido

1. Introducción a Docker
2. Docker Engine
   A. Componentes
   B. Comandos básicos
3. Creando imágenes propias
   A. Dockerfile
   B. Comandos necesarios

Introducción

- ¿Alguna vez os habéis encontrado en la siguiente situación?
  - Necesidad de instalar una aplicación concreta en poco tiempo.

    pero ...
  - El manual no está actualizado / no existe para el SO en uso.
  - Las dependencias no se instalan correctamente.
  - Tras instalarlo, otras aplicaciones dejan de funcionar.

- Los contenedores permiten solucionar este problema.

Introducción

- Los contenedores son conjuntos de herramientas que facilitan la implantación de despliegues software.
- Permiten tener despliegues estancos sin consumir excesivos recursos.
- Se podría ver como una técnica de “virtualización ligera”.
  - Aunque realmente no es una virtualización.

Introducción

- Problemática en un entorno complejo:
  - Aplicaciones de diferentes características
  - Diferentes entornos de despliegue

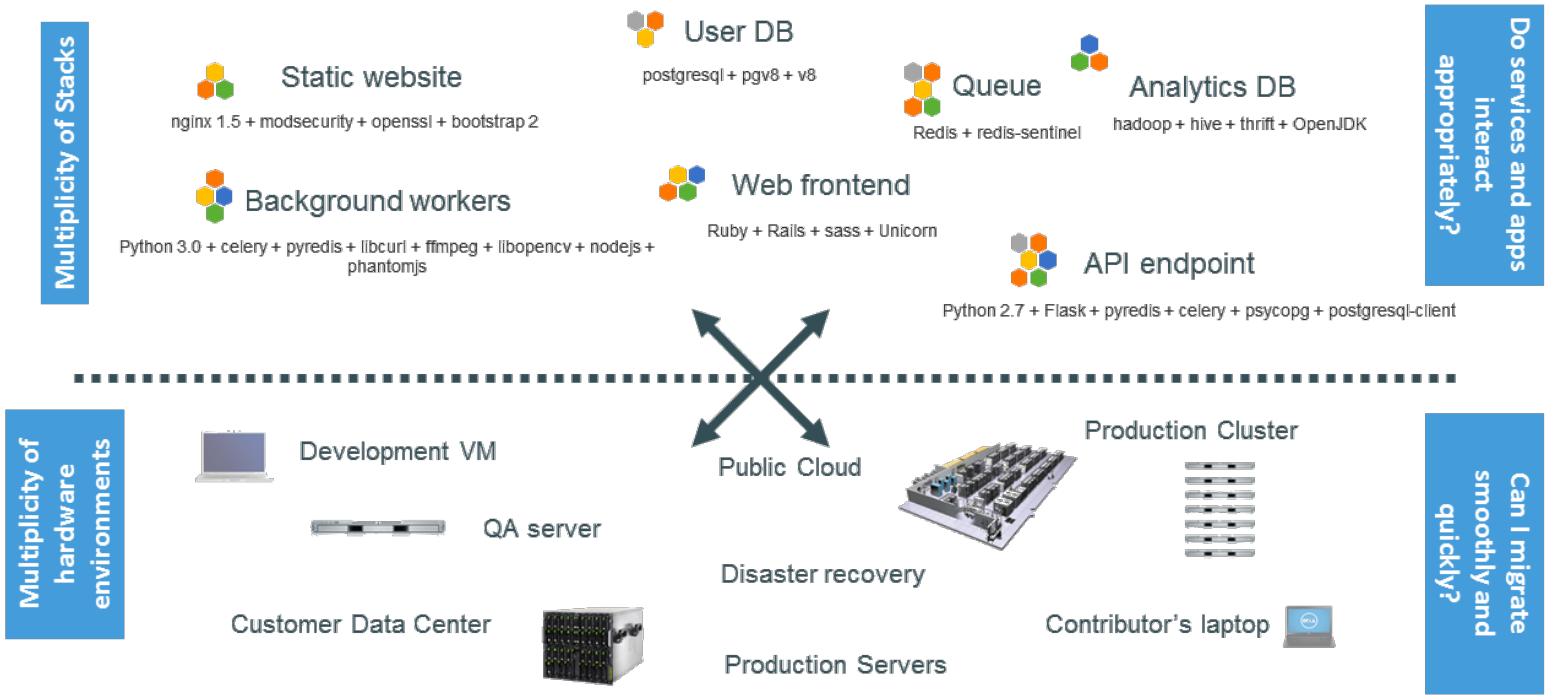


Imagen: John Lewis, 2013.

Introducción

- Solución: utilizar contenedores como herramienta de despliegue.

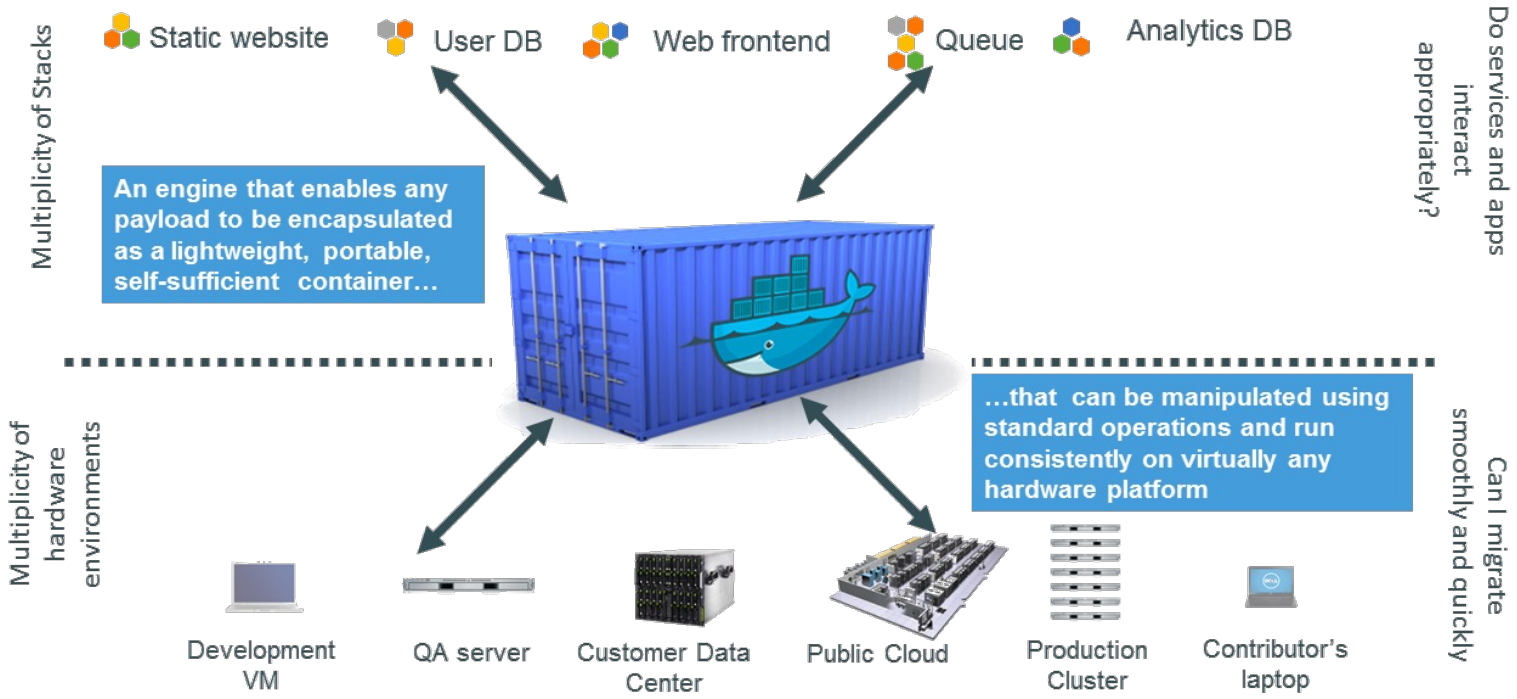


Imagen: John Lewis, 2013.

Introducción

- Ventajas de uso, para desarrolladores:
  - El código va a funcionar en cualquier sistema Linux moderno
  - Permite especificar dependencias y configuraciones por defecto.
  - El código no va a colisionar con otros servicios del sistema.
  - Mayor facilidad para distribuir el código.
  - Mayor rapidez de despliegue comparado con una máquina virtual.
  - ...

Introducción

- Ventajas de uso, para *sysadmins*:
  - Configurar correctamente el gestor de contenedores y las aplicaciones funcionan.
  - El ciclo de vida es mas consistente y más automatizable.
  - Eliminar inconsistencias entre diferentes entornos.
    - P.e. desarrollo, pruebas, producción, ...
  - Más ligero que un entorno completamente virtualizado.
  - ...

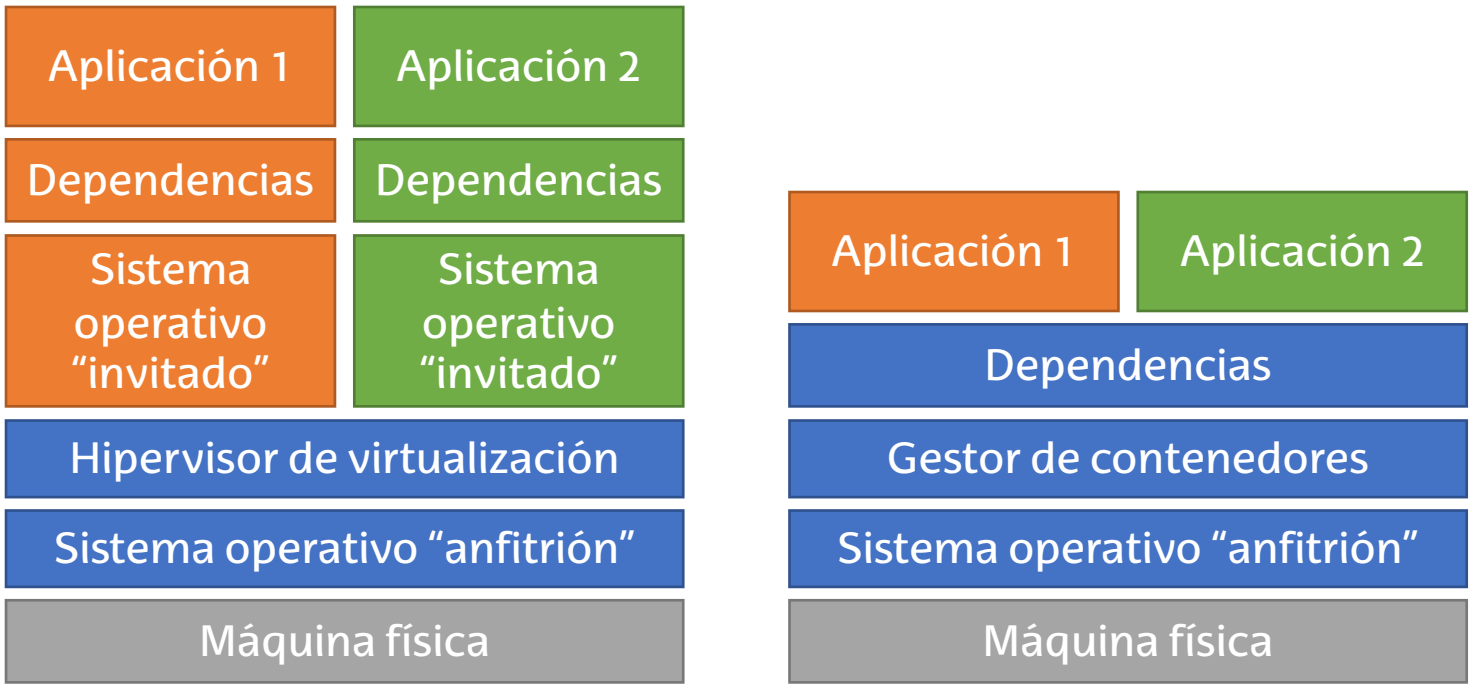
Introducción

- Comparación con la virtualización:

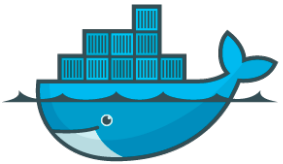
Virtualización

Introducción

- Comparación con la virtualización:



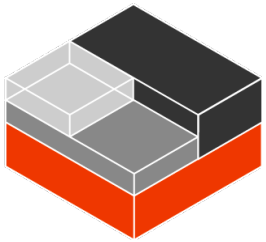
Docker



- Lanzado inicialmente en 2013.
  - Web: www.docker.com
- En la actualidad, sistema de contenedores *de facto*.
- Código fuente *open source*
  - Modelo de negocio *freemium*.
- Utiliza una variedad de técnicas de aislamiento en función del sistema en el que se ejecute.
  - cgroups, espacios de nombres, AppArmor, Netfilter, ...

Logo: https://upload.wikimedia.org/wikipedia/commons/7/79/Docker_%28container_engine%29_logo.png

LXC



- LinuX Containers
  - Web: https://linuxcontainers.org/
- Sistema de contenedores basado en cgroups y espacios de nombres Linux.
- Lanzado inicialmente en 2008
- No es muy popular fuera de entornos Ubuntu
  - LXC se enfocó en sysadmins como público objetivo.
  - Más información: https://www.upguard.com/blog/docker-vs-lxc

Logo: https://upload.wikimedia.org/wikipedia/commons/4/40/Linux_Containers_logo.png

Podman

- Framework para gestionar contenedores OCI
  - Alternativa a Docker
  - https://podman.io/
- Mismos comandos que Docker
  - Uso simplificado:

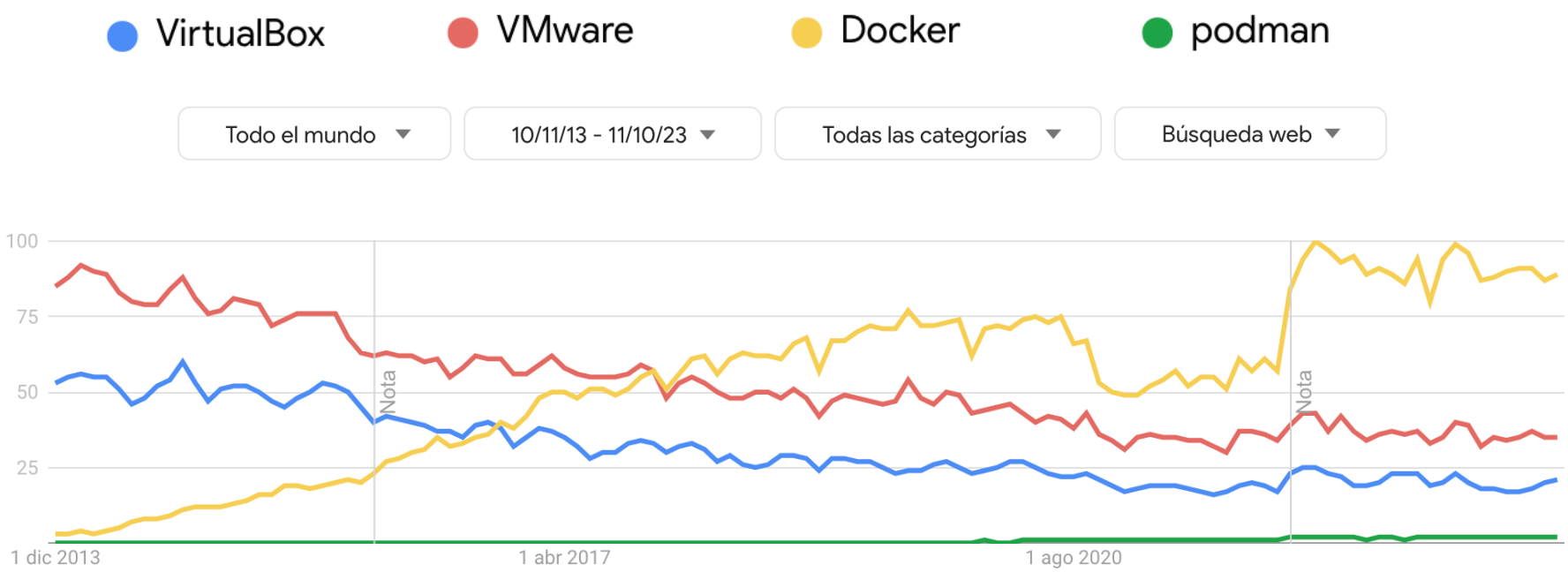
```
alias docker=podman
```

- Ventajas
  - Proceso gestor más ligero que Docker
  - Más integrado con Kubernetes

Logo: https://podman.io

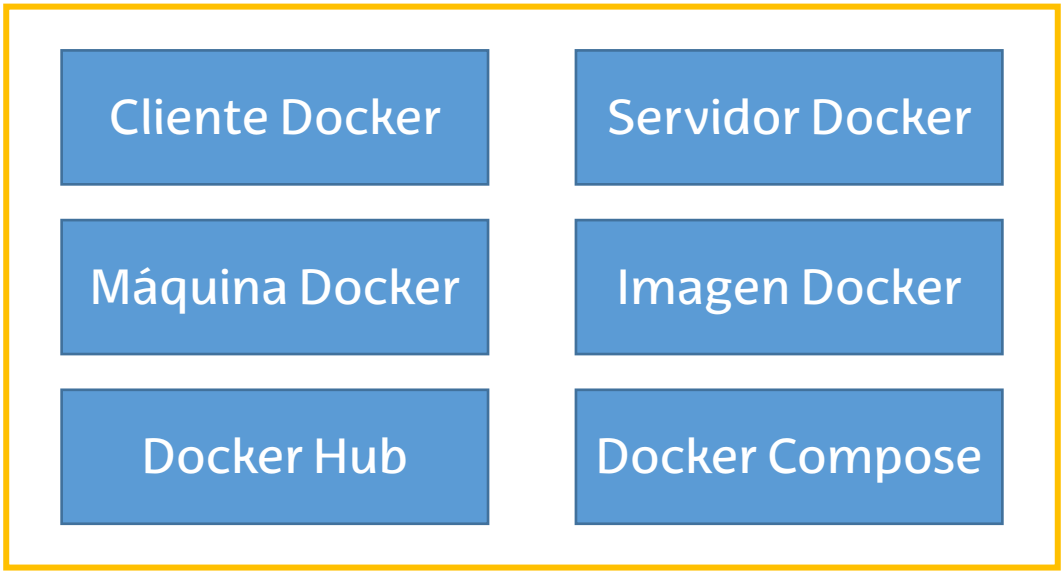
Comparativa de popularidad

- Utilizando Google Trends
  - Fecha: 11 de octubre de 2023



Ecosistema Docker

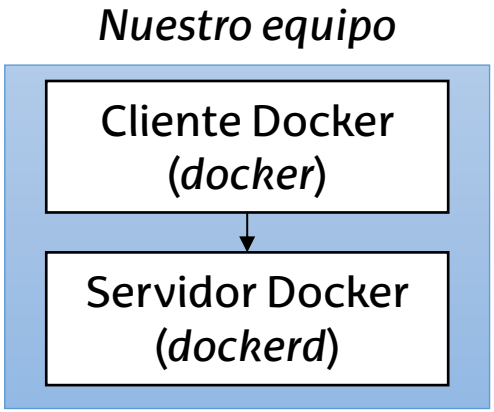
- Docker es una plataforma (o ecosistema) para crear y ejecutar contenedores.
- Algunos de sus componentes:



Docker Engine

- Componente Docker que se ejecuta en nuestro sistema.

- Tiene 3 piezas:
  - *docker*: Cliente de línea de comandos.
  - *dockerd*: Servidor con proceso daemon asociado.
  - API para que aplicaciones de 3ºs se comuniquen con *dockerd*.



Docker Engine

- Instalación en Ubuntu Server:
  - Seguir los pasos "*Install using the repository*" :
    - https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository

- Verificar la instalación:
  - Ejecutar:

```
sudo docker run hello-world
```

  - Entre el texto, debería verse:

```
Hello from Docker!

This message shows that your installation appears to be working correctly.
```

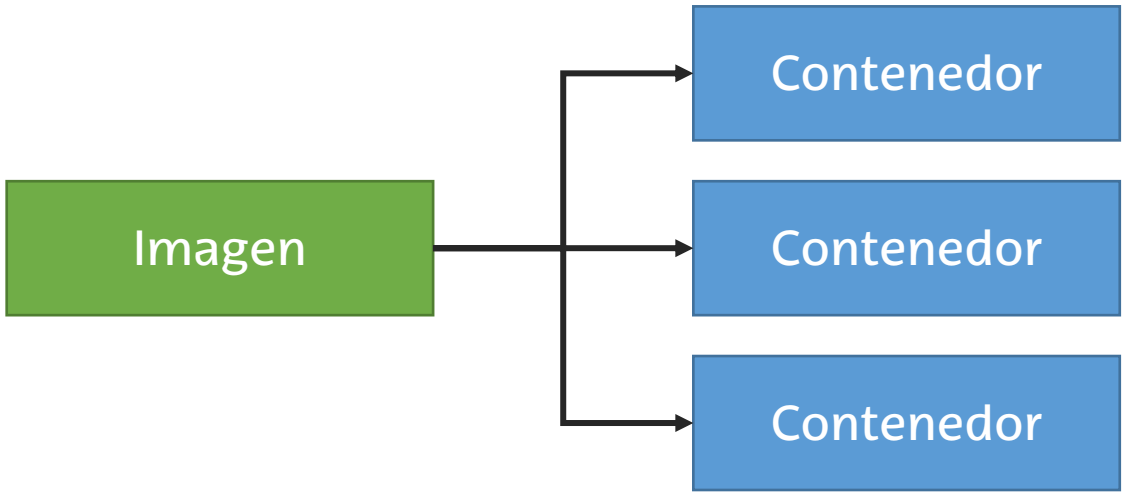
Ejercicio 1

- Instalar Docker Engine en una máquina virtual Ubuntu Server.

- Verificar la instalación.
  - Ejecutar el comando “hello-world”.

Ejecución de un contenedor

- Imagen: fichero único con toda la información necesaria para ejecutar un programa.
  - Incluyendo dependencias y configuración.
- Contenedor: instancia de una imagen.
  - Ejecuta un programa.



Ejecución de un contenedor

- Al ejecutar "run hello-world":

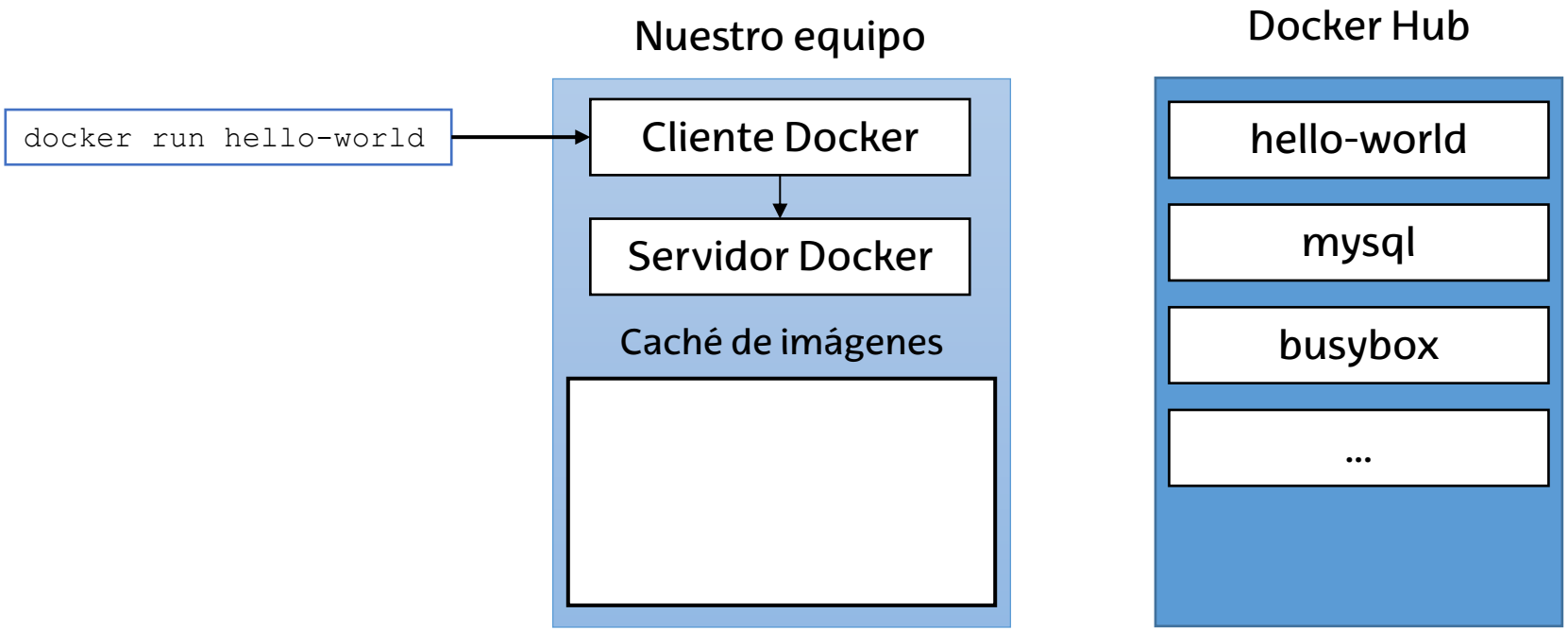


Imagen : Stephen Grider, Udemy, 2020

Ejecución de un contenedor

- Al ejecutar "run hello-world":

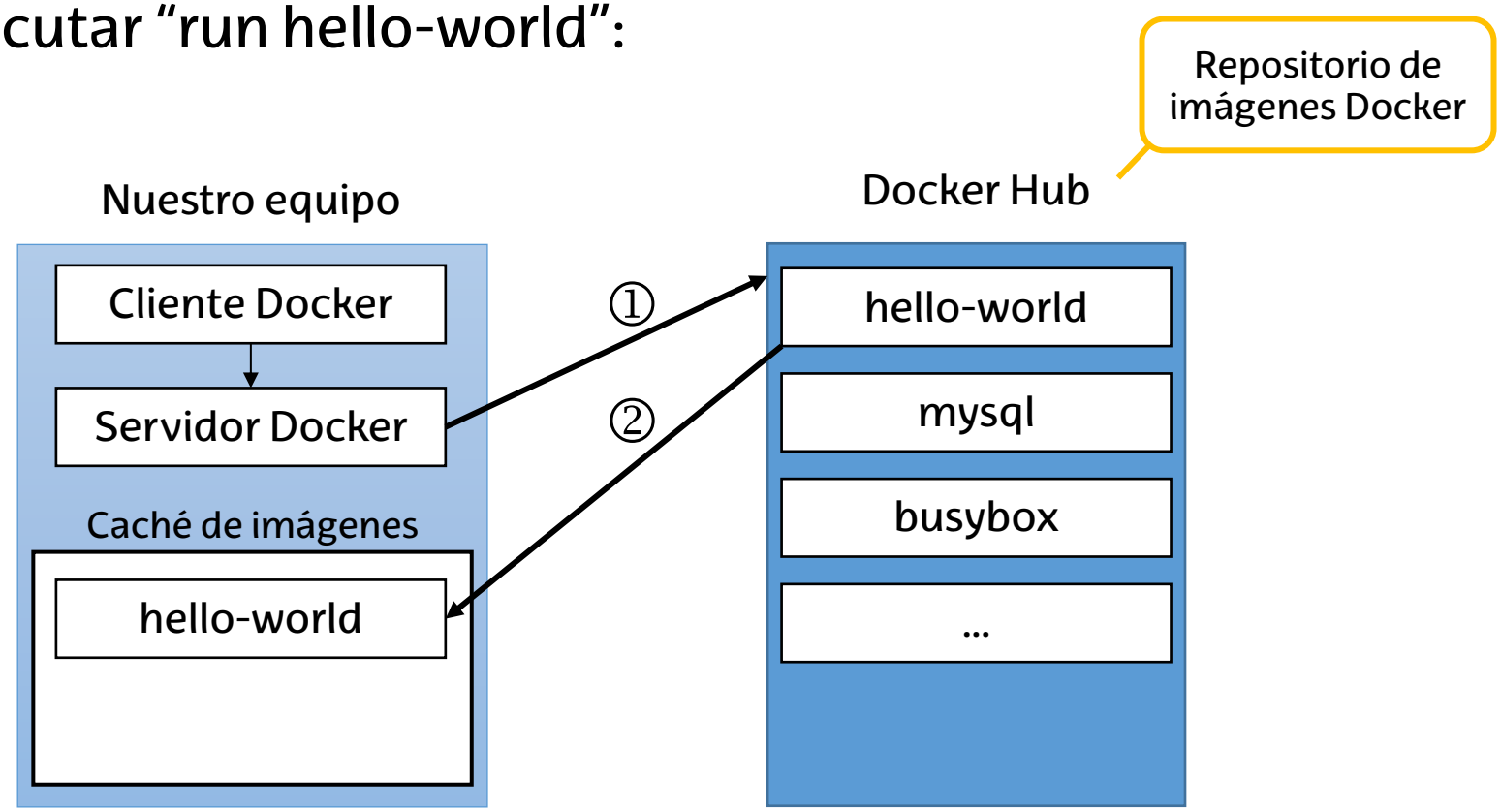


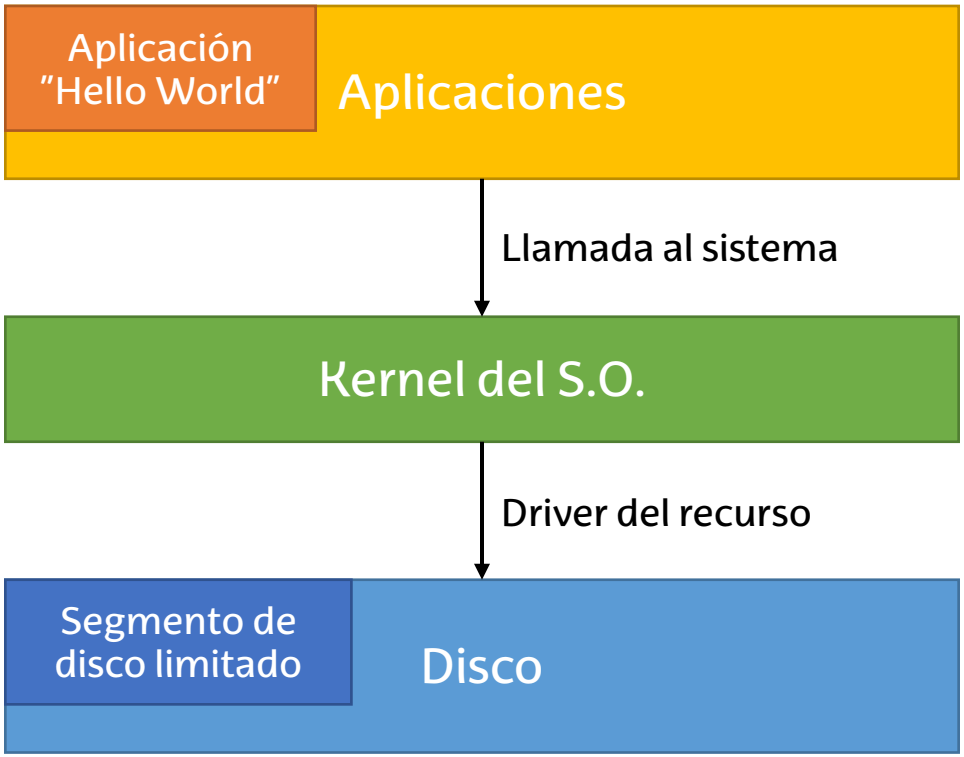
Imagen : Stephen Grider, Udemy, 2020

Ejecución de un contenedor

- Tecnologías necesarias para ejecución:
  - Espacios de nombres (*namespacing*)
    - Permiten aislar recursos por proceso (o grupo de procesos), p.e.
      - Disco duro
      - Red
      - Usuarios
      - Comunicación entre procesos
      - ...
  - Grupos de control (cgroups)
    - Limitar el número de recursos utilizados por un proceso, p.e.
      - Memoria RAM
      - Uso de CPU
      - E/S a disco
      - Ancho de banda de red
      - ...

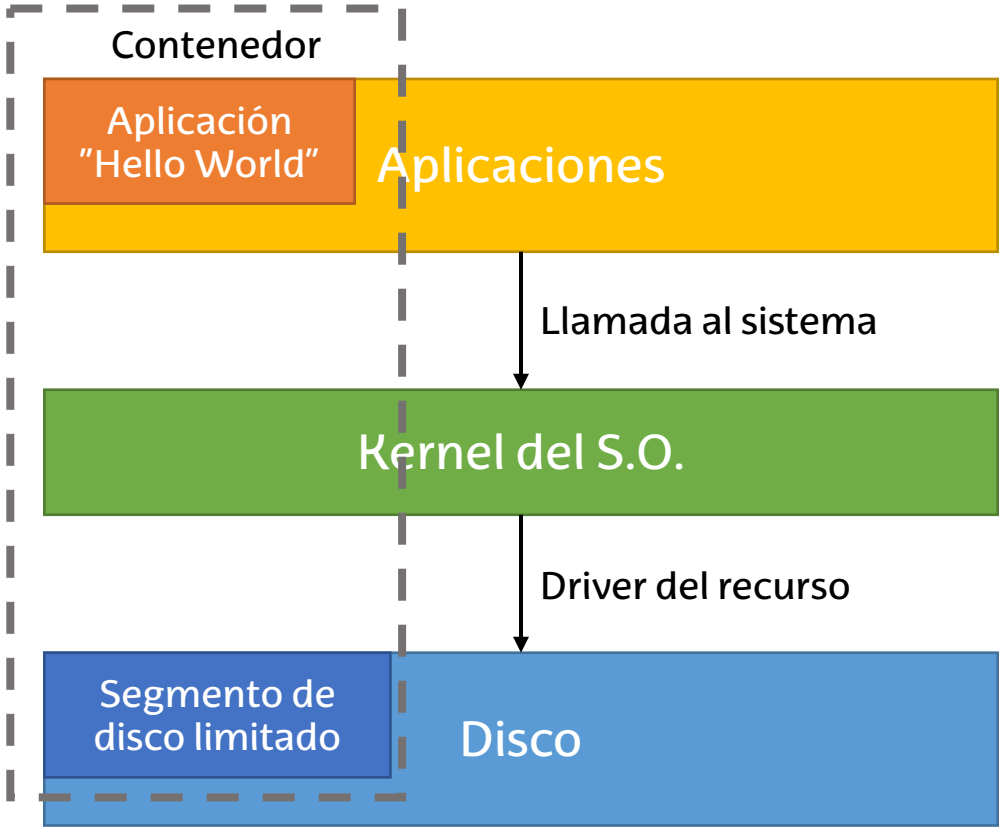
Ejecución de un contenedor

- Dentro del sistema completo:



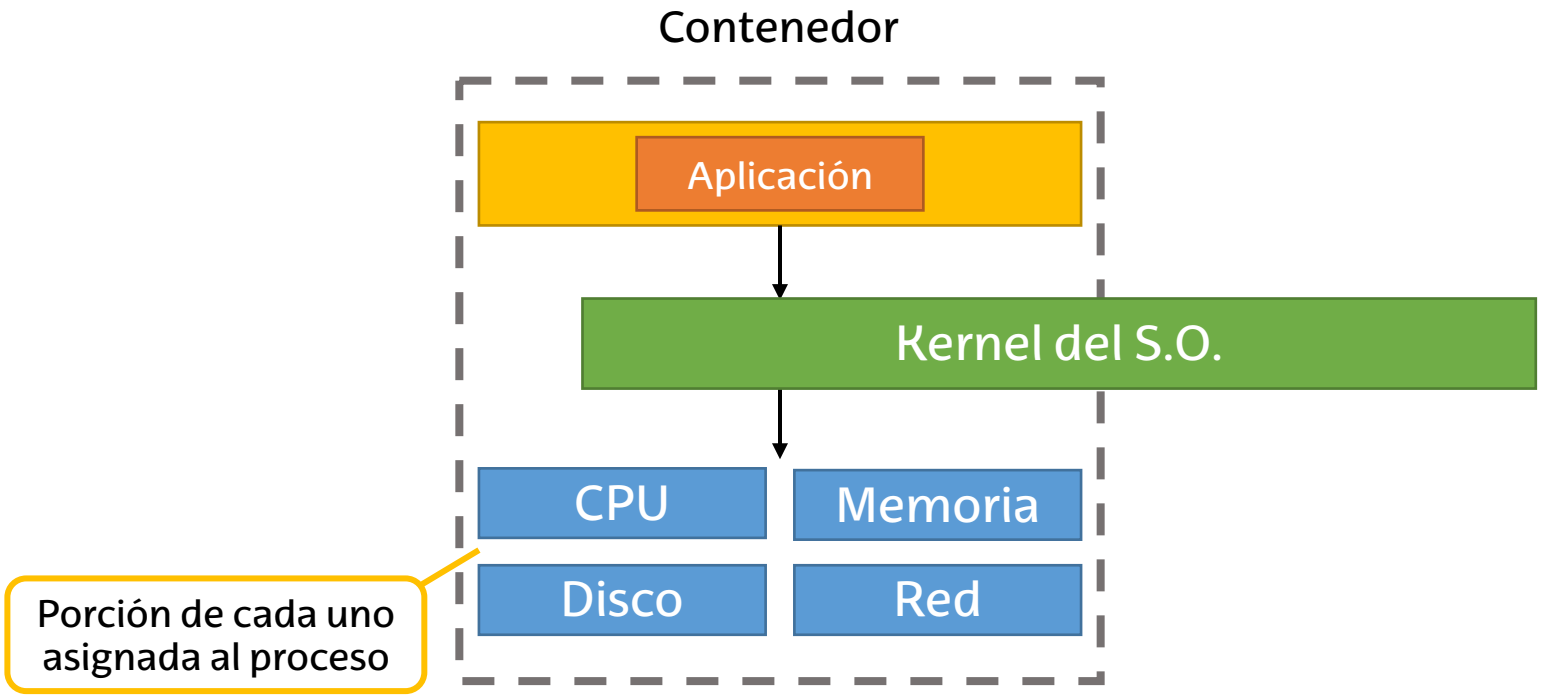
Ejecución de un contenedor

- Dentro del sistema completo:



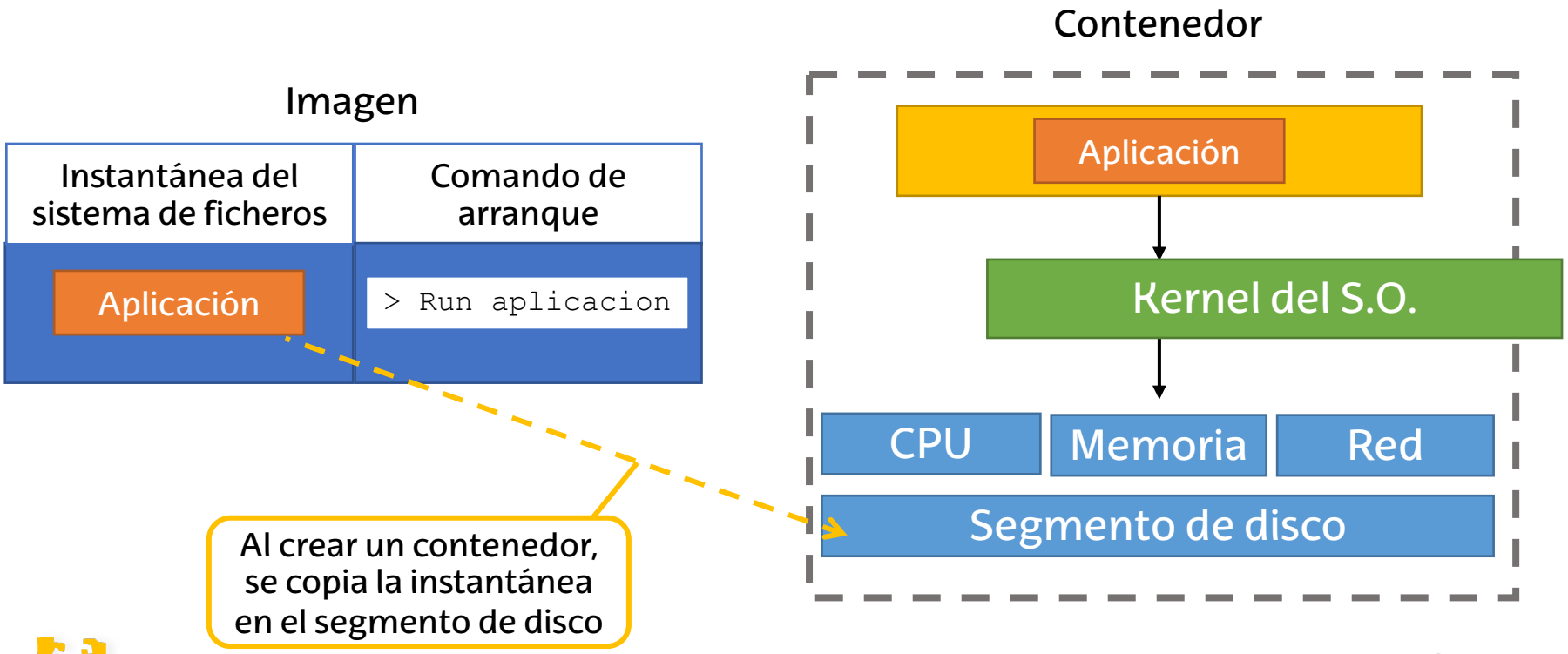
Ejecución de un contenedor

- Un contenedor es un proceso (o grupo de procesos) con un conjunto de recursos específicamente asignado a él.



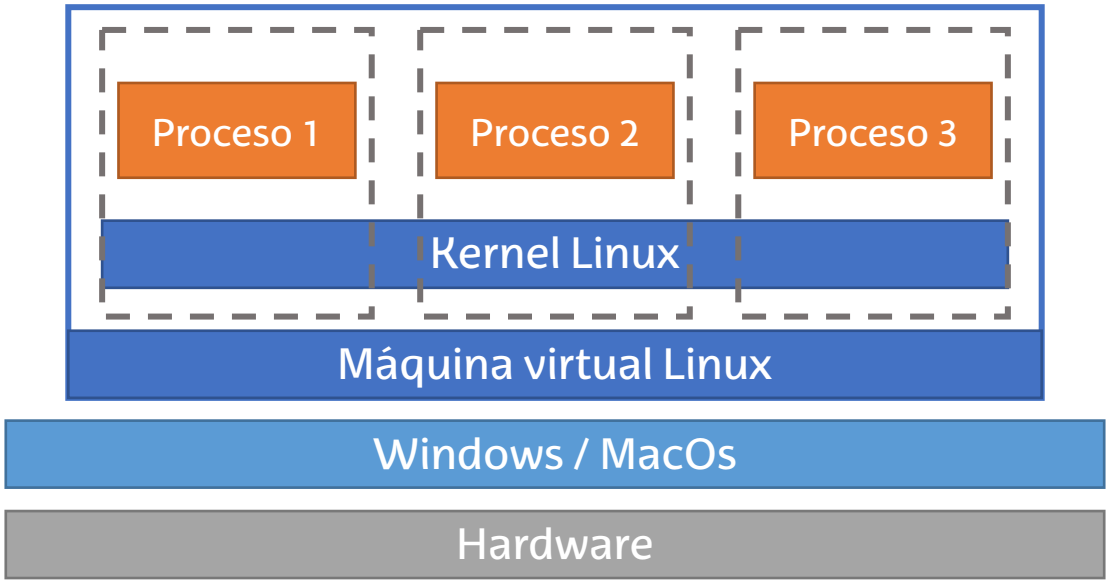
Ejecución de un contenedor

- Una imagen contiene:
  - Archivos y directorios necesarios para arrancar una aplicación
  - Comando de arranque



Ejecución de un contenedor

- Tecnologías como *namespacing* o *cgroups* son exclusivas de Linux.
  - En entornos Windows/Mac, Docker Engine utiliza una máquina virtual Linux para su ejecución.



Ejecución de un contenedor

- Tecnologías como *namespacing* o *cgroups* son exclusivas de Linux.
  - En entornos Windows/Mac, ejecutar:

```
docker version
```

  - Devuelve lo siguiente:
    - En un entorno Mac:

```
Client:
  …
  OS/Arch: darwin/amd64
  …
Server:
  Engine:
    …
    OS/Arch: Linux/amd64
```

Comandos Docker

- Crear y lanzar un contenedor desde una imagen:
  - Comando *run*:

```
docker run <imagen> [comando]
```

  - donde:
    - ‹imagen› es el nombre de una imagen.
    - [comando] es un comando de arranque definido por nosotros (opcional).
  - Internamente:
    - Copia la instantánea del sistema de ficheros al espacio de disco reservado para el contenedor.
    - Ejecuta:
      - Por defecto, el comando de arranque definido en la imagen.
      - Alternativamente, ‹comando› si se ha pasado por parámetro.

Comandos Docker

- Crear y lanzar un contenedor desde una imagen:
  - Comando *run*:

```
docker run <imagen> [comando]
```

  - La funcionalidad del comando de arranque depende de la imagen
  - Ejemplos:

```
docker run hello-word ls
```

    - Devuelve error: "hello-world" no está preparado para recibir el comando "ls"

```
docker run busybox ls
```

    - Devuelve un listado de directorios
      - Busybox[1] es una imagen Linux limitada con comandos básicos.

[1]Busybox : https://hub.docker.com/_/busybox

Comandos Docker

- Listar los contenedores en marcha:
  - Comando *ps*:

```
docker ps [parámetros]
```

  - Muestra:
    - ID de contenedor (valor hexadecimal)
    - Nombre de la imagen
    - Comando actual en ejecución
    - Momento de creación
    - Estado
    - Puertos
    - Nombre generado aleatoriamente
  - El párametro --all muestra el histórico de contenedores ejecutados

Comandos Docker

- Crear un contenedor a partir de una imagen:
  - Comando *create*:

```
docker create <imagen> [comando]
```

  - donde:
    - ‹imagen› es el nombre de una imagen
    - [comando] es un comando de arranque definido por nosotros (opcional).
  - Al crear el contenedor, se fija el comando de arranque
  - Ejemplo:

```
docker create busybox echo Hola
```

    - Devuelve el ID de contenedor creado (una cadena hexadecimal)

Comandos Docker

- Iniciar un contenedor:
  - Comando *start*:

```
docker start [parámetros] <ID-contenedor>
```

    - donde ‹ID› es el ID de un contenedor

  - Por defecto, no muestra la salida estándar del contenedor
    - Parámetro -a: devolver salida estándar del contenedor al terminal.

  - No se permite cambiar el comando de arranque

- *docker run = docker create + docker start*

Comandos Docker

- En una ejecución sin incidencias, un contenedor:
  - Se inicia con *start*
  - Ejecuta su comando de arranque
  - Finaliza de manera normal, estado *Exited*

- Un contenedor en estado *Exited* se puede reutilizar
  - Utilizar el comando *start* con su ID

Comandos Docker

- Ciclo de vida de un contenedor
  - Y comandos asociados

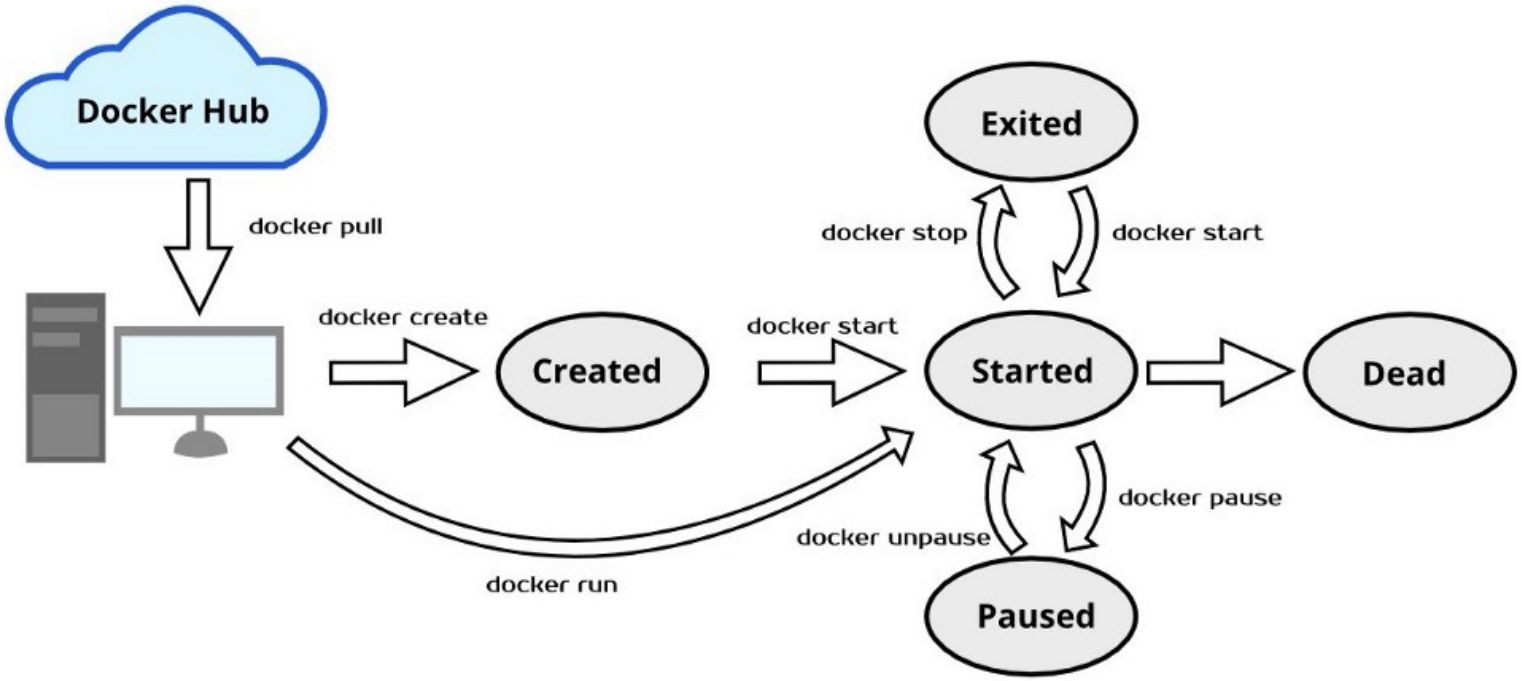


Imagen: https://medium.com/future-vision/docker-lifecycle-tutorial-and-quickstart-guide-c5fd5b987e0d

Comandos Docker

- Borrar un contenedor
  - Comando *container rm*:

```
docker container rm <ID-contenedor>
```

    - donde ‹ID› es el ID de un contenedor
- Para limpiar el entorno:
  - Comando *prune*:

```
docker system prune
```

  - Elimina:
    - Contenedores que hayan sido ejecutados
    - Caché de imágenes (descargas de Docker Hub)
  - Muestra la cantidad de espacio en disco liberado
    - Vacía el histórico mostrado con *docker ps --all*

Comandos Docker

- Recuperar la salida estándar de un contenedor
  - Comando *logs*:

```
docker logs <ID-contenedor>
```

    - donde ‹ID› es el ID de un contenedor

  - Muestra por la terminal todo lo emitido por la salida estándar del contenedor.
  - Evita tener que re-ejecutar contenedores.
    - O recuperar información de uno que esté en ejecución.

Comandos Docker

- Parar un contenedor
  - Comando *stop*:

```
docker stop <ID-contenedor>
```

    - donde ‹ID› es el ID de un contenedor

  - Envía una señal SIGTERM al proceso en ejecución del contenedor.
  - Si pasan 10 segundos desde que se envía y el contenedor no ha finalizado, Docker Engine automáticamente envía SIGKILL.

Comandos Docker

- Matar un contenedor
  - Comando kill:

```
docker kill <ID-contenedor>
```

    - donde ‹ID› es el ID de un contenedor

  - Envía una señal SIGKILL al proceso en ejecución del contenedor.
  - El contenedor se finaliza instantáneamente.

Comandos Docker

- Ejecutar un comando en un contenedor en marcha
  - Comando *exec*:

```
docker exec -it <ID-contenedor> <comando>
```

  - donde:
    - El parámetro -*it* es la unión de :
      -i: Abrir el canal STDIN del proceso en el contenedor
      -t: Vincular nuestra terminal a una terminal con el proceso
    - indica la ejecución dentro del contenedor.
    - ‹ID› es el ID de un contenedor
    - ‹comando› es el comando a ejecutar.

Ejercicio 2

- Encontrar:
  - Cuántos usuarios hay creados dentro de la imagen de busybox.
  - Cuántos de esos usuarios tienen /bin/false como Shell de inicio.
  - Con qué usuario se ejecutan los comandos en busybox.

- Crear un contenedor de busybox con el comando de arranque “echo Hola Bilbao”.
  - Cronometrar el tiempo que tarda en ejecutarse.
    - Pista: Utilizar el comando *time* de Linux

- Eliminar todos los contenedores creados.

Dockerfile

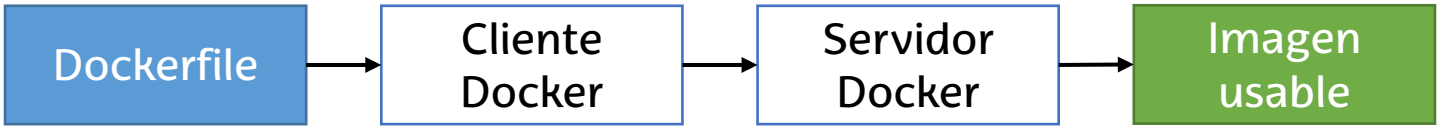
- Hasta ahora hemos utilizado imágenes Docker de terceros disponibles en Docker Hub.
- En esta parte vamos a aprender a crear nuestras propias imágenes.
- La pieza clave va a ser el *Dockerfile*.

Dockerfile

- Fichero de texto plano que contiene instrucciones de configuración.

- Define qué va a contener la imagen:
  - Qué programas y qué ficheros
  - Cuál es el comando de arranque por defecto

- Docker Engine creará la imagen:



Dockerfile

- Componentes de un Dockerfile:
  - Una imagen base
  - Comandos para instalar programas adicionales
  - Un comando de arranque por defecto

- Pasos para crear un imagen:
  1) Crear una carpeta nueva
  2) Dentro de la carpeta, crear un fichero llamado Dockerfile y añadir la información necesaria
  3) Construir la imagen con Docker Engine

Dockerfile

- Estructura general de un Dockerfile:
  - Ejemplo:

```
# Imagen base
FROM ubuntu

# Descargar e instalar dependencias
RUN apt -qq update && apt -qq -y install fortune

# Comando de arranque
CMD /usr/games/fortune
```

  - Los comentarios se añaden con #

Dockerfile

- Todas las líneas de un Dockerfile comienzan con una instrucción, seguida de parámetros.
- Instrucciones más usadas:

FROM	Imagen base (p.e. ubuntu, debian, alpine, ...)
RUN	Ejecutar comando para instalar dependencias
CMD	Comando de arranque por defecto
WORKDIR	Cambiar directorio
COPY	Copiar fichero(s) al contenedor

- Algunas no tienen por qué aparecer sólo 1 vez
  - Puede haber múltiples RUN en un Dockerfile
- Más información:
  - https://docs.docker.com/engine/reference/builder/

Dockerfile

- Imágenes base
  - Preinstalaciones de SSOO con diferentes programas
- En el proceso de creación de una imagen:
  - La imagen puede estar en la caché local
  - Si no está, se descarga de DockerHub
- Para listar las imágenes en el sistema:

```
docker images
```

Creación de una imagen

- Crear imagen desde un Dockerfile:
  - Moverse a la carpeta que contiene el Dockerfile
  - Ejecutar:

```
docker build .
```

Contexto de construcción: carpeta con todos los archivos a utilizar para crear la imagen

  - Si todo ha ido bien, al final se debería mostrar:

```
…
Succesfully built <ID>
```

- Crear contenedor con la imagen:
  - Ejecutar:

```
docker run <ID>
```

    - donde ‹ID› es el ID obtenido al final de la creación de la imagen.

Creación de una imagen

- Se puede asignar un “nombre” a una imagen
  - Nos evita usar el ID cada vez que queremos ejecutarla.

- Utilizar:

```
docker build -t <tag> <directorio>
```

  - donde:
    - -t Indica que se va a asignar un tag (equivalente a un nombre)
    - ‹tag› El tag a asignar
    - ‹directorio› El directorio del contexto de construcción

Creación de una imagen

- Se puede asignar un “nombre” a una imagen
  - Nos evita usar el ID cada vez que queremos ejecutarla.

- Existe una convención para asignar tags:

Nuestro nombre Docker ID	/	Nombre del proyecto	:	Versión

Generalmente “latest” si se está trabajando en la última versión

- Por ejemplo:
  - Para el Dockerfile de la diapositiva 45.

```
ulopeznovoa/fortune:latest
```

Creación de una imagen

- Para generar una imagen desde un Dockerfile, Docker crea una serie de imágenes y contenedores intermedios.

- Para cada instrucción del Dockerfile:
  - Se obtiene la imagen del paso anterior (o la base si es la 1ª).
  - Se crea un contenedor intermedio, en el que se ejecuta la instrucción (provocando cambios en su sistema de ficheros).
  - Se toma una instantánea del sistema de ficheros del contenedor intermedio, y se utiliza para generar una nueva imagen.

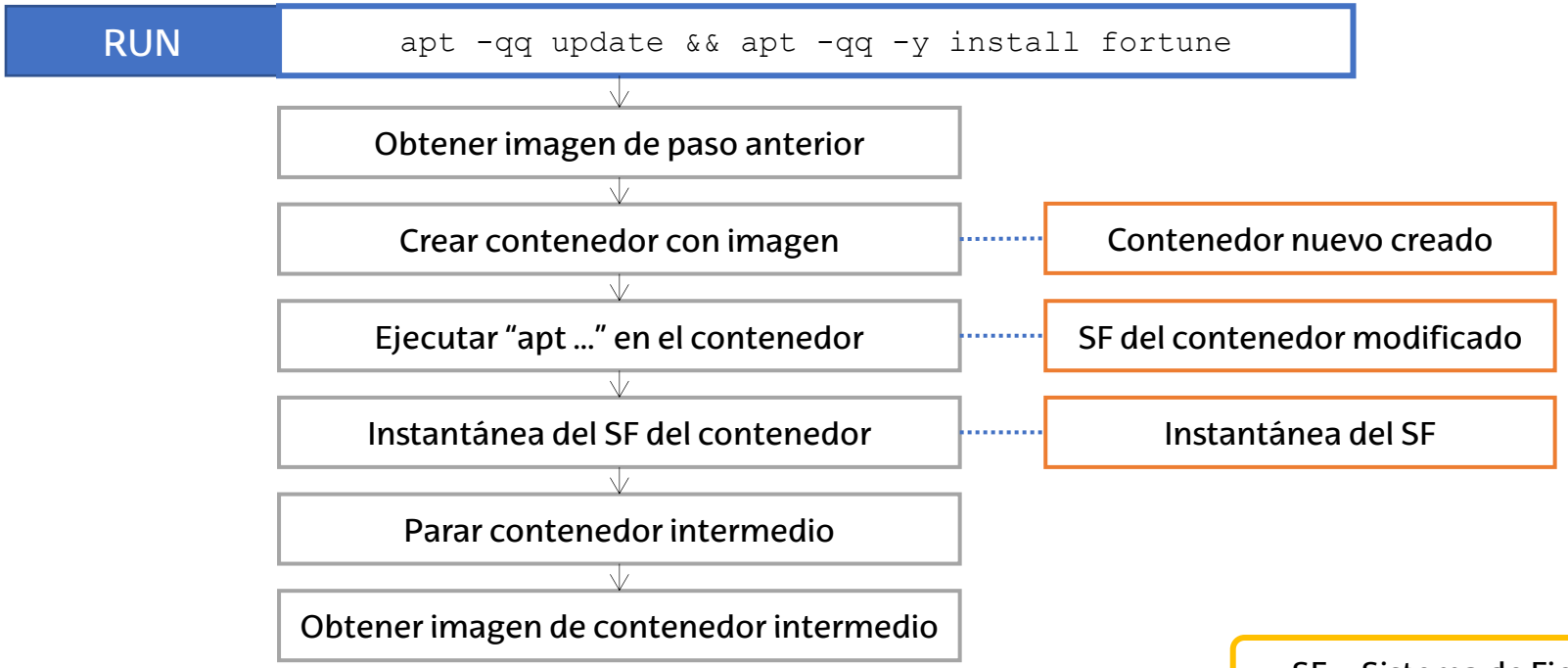
- Cuando no quedan más instrucciones, se devuelve la última imagen generada.

Creación de una imagen

- El proceso completo paso a paso (1/3)
  - Dockerfile de la diapositiva 45 como ejemplo

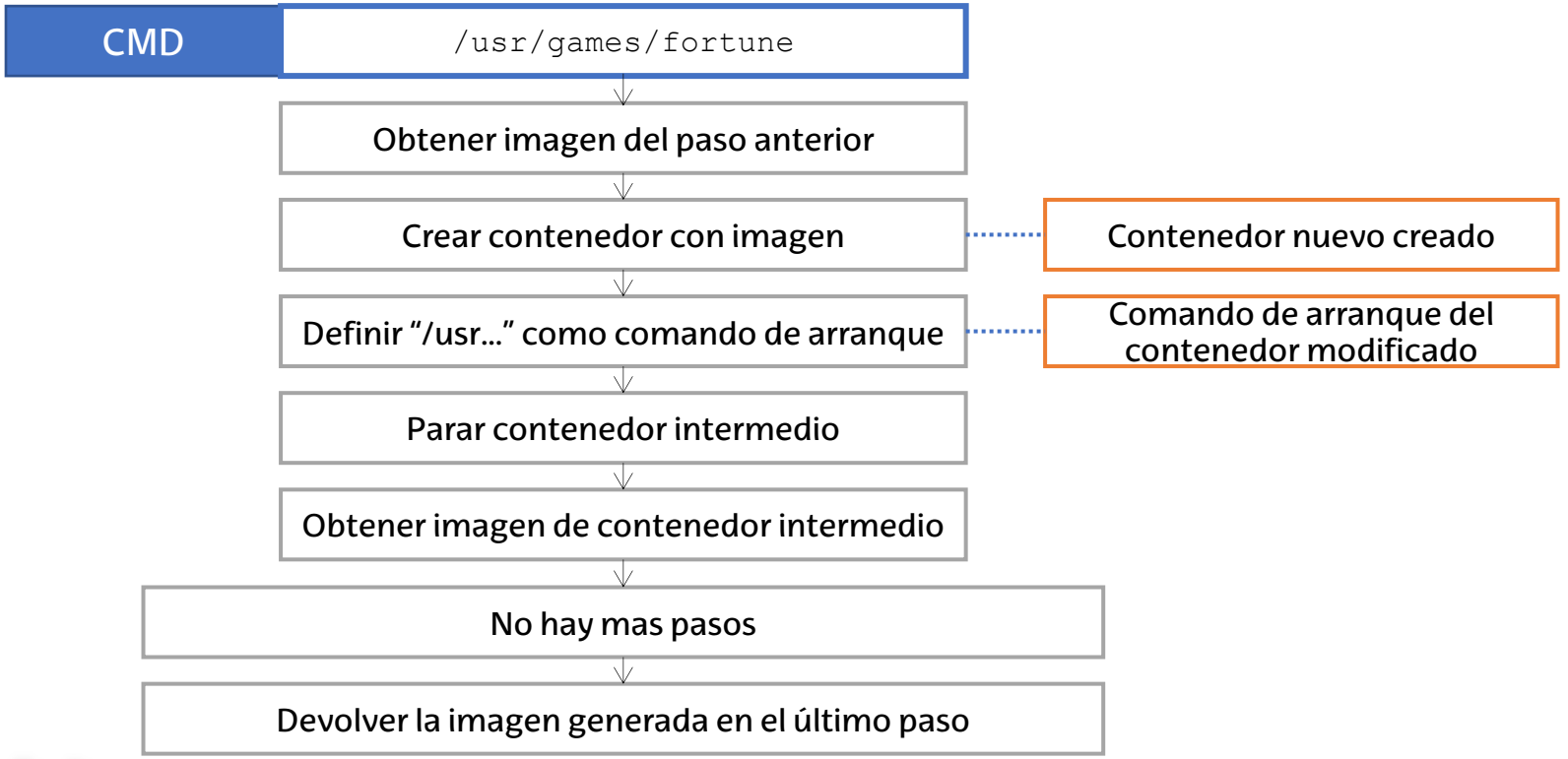
Creación de una imagen

- El proceso completo paso a paso (2/3)
  - Dockerfile de la diapositiva 45 como ejemplo



Creación de una imagen

- El proceso completo paso a paso (3/3)
  - Dockerfile de la diapositiva 45 como ejemplo



Creación de una imagen

- En el proceso de crear una imagen, Docker guarda las imágenes y contenedores intermedios en una caché local.
- Al crear una imagen desde un Dockerfile que ha sido modificado, Docker utiliza esta caché para buscar imágenes/contenedores que pueda reutilizar.
  - Se puede observar en las instrucciones que muestran:

```
---> Using cache
```

- El orden de las operaciones en los Dockerfile es importante.
  - Puede suponer una ganar (o perder) tiempo al reconstruir imágenes.

Crear contenedor desde imagen

- Se puede utilizar un contenedor como base para generar una imagen de manera manual.

- Comando *commit*:

```
docker commit -c <comando> <ID>
```

  - donde:
    - ‹comando› Es el comando de arranque que queremos establecer
    - ‹ID› Es el ID del contenedor a usar de base

- No es la forma habitual de crear imágenes.
  - Por defecto, utilizar Dockerfiles.

Ejercicio 3

- Crear y ejecutar la siguiente imagen Docker:
  - Imagen base: Ubuntu
  - Dependencias: instalar el paquete “stress-ng”
  - Comando de arranque: Stress-ng con 1 hilo durante 5 segundos
  - Asignarle el nombre: ‹vuestro-nombre›/benchmark5s
    - Ejemplo: unai/benchmark5s
- Modificar la imagen para que stress-ng se ejecute durante 90 segundos.
  - Mientras está en marcha, abrir una Shell dentro del contenedor y ejecutar “top” para verificar el uso de CPU.

Bibliografía

- Stephen Grider. “Docker and Kubernetes: The complete guide”, Udemy, 2020.
  - https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide
- John Lewis, “Intro to Docker”, 2013.
  - http://pointful.github.io/docker-intro/#/
- Jean-Philippe Gouigoux. “Docker. Primeros pasos y puesta en práctica de una arquitectura basada en micro-servicios”, 2018.
  - Editorial ENI, ISBN: 2409015891.

- Consultados en octubre 2020