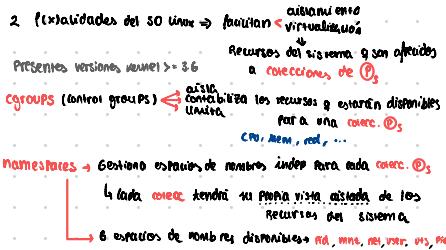


x4

# T10: Contenedores - Docker

## Cgroups y namespaces en Linux



## Namespaces

- PID → en ④ spec. Pueden nombres PID solo de otros ⑤
  - del = espacio de nombres
- NET → los ⑥ rodan dentro su propio espacio de Archivos
  - PILOT\_ROOT, ejecución chroot
- REL → PROTECCIÓN EN BASE DE PID PARA TODOS ⑦ DE UNA ENTRADA
  - interfaces de red, tablas encarr., reglas filtrado, ...
- USER → PROPORCIONA USER ID, y GROUP ID, index (aislamiento usuarios)
  - Lo ⑧ proporciona namespaces, index (aislamiento namespaces)
- IPC → facilita PDA entre ⑨ PDA -> HIF (de ⑩ colecciones)

## Cgroups

- F (familias) ← asignar recursos del sistema → asociados a grupo ⑪
  - asimilar
- se basa en un S.O. genérico en el q pueden ⑫ q se grups de control
  - ↳ grupo 1
  - ↳ grupo 2
  - ↳ grupo 3 → hereda recursos grupo 1
- En cada grupo se pueden establecer los límites al consumo de recursos para todos los grupos de ⑪ q asigan de esa grupo
- Controladores de Recursos:
  - CPU, dispositivos, socket, net\_cls, net\_prio, ...
  - Para agrupar los ⑪ en ⑫ espacios de nombres

## Aislamiento en Linux

### LXC: Linux Containers

Interfaz para acceder a las facilidades de aislamiento de Linux implementación de contenedores.

- Método ⑬ (familias del kernel: namespaces, cgroups, chroots (mediante PILOT\_ROOT), Profiles, SELinux, ...)
- Objetivo: virtualización a nivel del Sist. Op
  - ofrecer a las apps un entorno equivalente a un S.O. Linux sin necesidad de usar VMs
  - ↳ SPEC. N. S.O. Linux → 1. Usando S.O. Linux configuración
  - ↳ Entornos "ligeros" → todos los contenedores acceden a los recursos kernel.
- IMPLEMENTACIONES → UBUNTU - LXC

### OpenVZ → Virtualización a nivel S.O.

- OpenVZ + Virtuozzo (solución comercial) → soluciones de virtualización a nivel S.O.
  - ↳ TB: virtualización basada en contenedores
- S.O. configuración USA → un kernel modificado + complementado por VMs alojadas
  - se comportan como instancias S.O. configuración
  - Efect. ⑪ de forma custodia usando libvirt
- Otras soluciones + "VPS: Virtual Private Server"
  - todas las VPS son Linux
- LIMITACIONES = Kernel del S.O. configuración
  - pueden usar ≠ distrib. Linux
- Otras soluciones: Linux-VServer, Free BSD jails, LXD, Solaris zones

## Ejemplo LXC Red Hat

### Instalación de dependencias y LXC (esta en el repo epel):

```
yum install epel-release debootstrap perl libvirt
yum install lxc lxc-templates
```

Nota importante: libvirt asigna virbr0 a su brido predeterminado, pero LXC utiliza el nombre br0. Para corregir esto en la configuración de LXC [etc/libcemu/default.conf]:

```
net:dev=br0 devname=br0
```

### Arrancamos los servicios:

```
systemctl enable libvirtd_lxc
systemctl start libvirtd_lxc
```

Comprobamos que tenemos un nuevo brido virbr0 (lxcbr0):

```
lxc-checkconfig
```

- En la distribución actual no se incluyen plantillas locales, pero se cuenta con la plantilla "download", que nos muestra un listado amplio de contenedores para elegir.

Creemos un nuevo contenedor "download":

lxc-create -t download -n contenedor01

Elegimos por ejemplo "ubuntu", "jammmy" y "amd64" (Ubuntu 22.04)

Arrancamos el contenedor (naming como un demonio):

lxc-start -n contenedor01 -d

Visualizamos su estado:

lxc-info -n contenedor01

lxc-ex -t

Poner la directiva DOWNLOAD\_VALIDATE a false en /usr/share/lxc/templates/lc-download

### Abrimos la sesión en el contenedor:

lxc-attach -n contenedor01

Nota: si el Shell no encuentra los comandos básicos (ls, cat, ...) puede ser que lbn no se encuentre en el path: export PATH=\$HOME:\$PATH:\$HOME/.local/bin:\$HOME/.local/share:\$HOME/.local/lib:\$HOME/.local/include:\$HOME/.local/share/glib-2.0:\$HOME/.local/share/aclocal

### Podemos ver la versión del Ubuntu:

cat /etc/os-release

### Salimos del contenedor con exit. Para detener el contenedor:

lxc-stop -n contenedor01

### Para eliminarlo:

lxc-destroy -n contenedor01

## Ejemplo LXC/LXD en Ubuntu

### Instalación de LXC y la API LXD (usar valores por defecto)

```
sudo apt install lxd
sudo lxd init
```

### Listar las imágenes disponibles:

lxc image list images

### Lanzamos un contenedor. Sintaxis de lxc launch:

lxc launch -t [tipo] [images:] [distro:] [version:] [arch:] [nombre]

lxc launch -t t2-micro images:fedoraproject.org:38/amd64 co1

- Podemos usar distintos comandos para gestionar nuestros contenedores: lxc list | exec | stop | start | delete | snapshot ...

```
lxc list
lxc exec co1 bash
```

# T10: Contenedores - Docker

## Docker

Build, ship and Run . Any Application, anywhere

- Docker → despliegue automático de Apps
  - ↳ empaceta y ejec. su código, recursos, dependencias, configuración, etc.
- Mixta las capas de aislamiento de recursos del kernel de Linux
  - ↳ Ejecuta librerías
- Características
  - Portabilidad: puede ser desplegado en cualquier otro sistema operativo
  - Ligereza: consume menos recursos y memoria
  - Autosuficiente



## ¿Qué nos ofrecen los contenedores?

- Encapsulamiento → agrupa todos los componentes suyos. Integridad del entorno.
- Aislamiento → Protección entre aplicaciones.
- Consistencia → entorno para desarrollo y producción.
- Modularidad → facilita desarrollo microservicios.
- Multiplatforma → seguimiento en tiempo real de recursos.
- Monitoreo → interacción entre contenedores.
- Seguridad → alojamiento en la nube.
- Escalabilidad → autoescalamiento en la nube.

## Conceptos clave

**Imagen** → Paquete ejecutable que incluye todo lo necesario para ejecutar una aplicación.

**Container** → Instancia de ejec. de una img.

Por defecto se ejec. completamente aislado del host + otros contenedores. Puede acceder a los archivos del host o comunicarse -> fuera.

**Registro** → es una capa del buelver que permite distribuir imágenes de docker. Alus. usalo a Docker Hub. Mantiene repos de imgs.

## Contenedores con Docker

### Imagen de un Docker

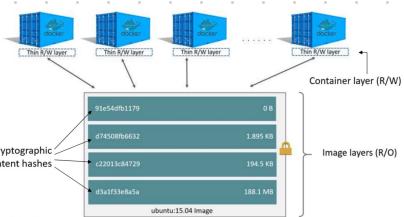
Imagen → conjunto de capas de solo lectura

Representan **diferencias** en el sis. de archivos  
se unen para formar un sis. de archivos raíz  
de un contenedor (overlays, aufs, btrfs, overlayfs,...)

- Al crear **Container** → se crea una capa de lectura/escritura (capa de contenedor)

Todos los **cambios** realizados se escriben  
en la capa de contenedor  
cada cambio es una capa  
se apila

- pero cada una mantiene su propio estado
- Container 1 → imagen 1
  - Container 2 → imagen 1
  - Container 3 → imagen 1
  - Container 4 → imagen 1
- si un **contenedor** se elimina  
solo se elimina la capa de contenedor.



### Docker Hub

Repos públicos + Privados

1º Repo es gratuito

## Arquitectura

Docker Engine → Aplicación cliente-servidor para la gestión de contenedores

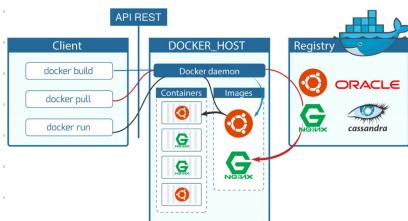


Docker Registry → protocolo para la distribución de imgs. Docker

Docker Compose → herramienta para definir y ejecutar entornos Docker mediante plantillas

Docker Swarm → herramientas de gestión

Docker Machine → aprovisionamiento de VMs para ejec. Docker con VirtualBox



## Docker Registry

Servicio para el almacenamiento de las imágenes docker. Descargar, subir, borrar, facilitar su distrib.

- Proveedores
  - Docker Hub
  - Google Container Registry
  - AWS Container Registry
  - Azure Container Registry
- Repository → colección de imgs. con nombres específicos (tags) ubicados. → Registro identif. de gestión (lates).
- Si el proveedor de almacenamiento de img.:
  - Registry >> Repository >> Img >> Tags

registryhost:5000/repository-name/image-name:tag



# T10: Contenedores - Docker

## (Contenedores con Docker)

### Volumenes de Datos

- Cuando BORRAMOS en **contenedor** desaparece toda la info. & almacenamiento permanece al usar **volumenes de datos**
- Volumen → directorio o archivo que se puede montar en un **contenedor**
  - ↳ creas -v
  - ↳ Puede ser utilizado por N contenedores
- 2. Alternativas
  - Volumenes comunes** → cont. → cont. → directorio → todos apuntan a un directorio
  - Volumenes armados por otros contenedores** → volumen → contenedor 1 → directorio → apunta a los volumenes de datos de otros

### Ejemplo:

- Vamos a mantener los archivos de las bases de datos de mariadb (`/var/lib/mysql`) en un directorio de la máquina que ejecuta el contenedor (en su directorio `/var/pruebas`)

```
[root@server ~]# mkdir /var/pruebas
[root@server ~]# docker run --name mariadb-persistente -v /var/pruebas:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=mypass -d mariadb
```

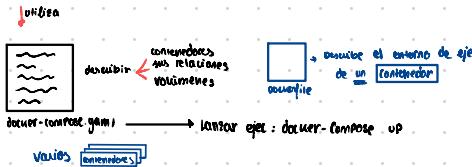
- Identificamos la IP del contenedor y lanzamos un cliente mysql desde otro contenedor:

```
[root@server ~]# docker inspect --format='{{(range .NetworkSettings.Networks)(.IPAddress)}{(.end)}}' mariadb-persistente
[root@server ~]# docker run -it --rm mariadb sh -c 'exec mysql -h172.17.0.2 -P3306 -uroot -pmypass'
```

- Al finalizar la sesión comprobamos el directorio `/var/pruebas`

### Docker Compose

Compose → herramienta para definir ejecutar apps docker & implicar a n contenedores



### Estructura docker-compose.yml

```
version: "3.9" # optional since v1.27.0
services:
  web:
    build:
    ports:
      - "5000:5000"
    volumes:
      - ./code
      - logvolume01:/var/log
    links:
      - redis
    redis:
      image: redis
    volumes:
      logvolume01: {}
```

### Ejemplo WordPress con MySQL y PHPAdmin

```
version: '2'
services:
  cont-wp:
    image: wordpress
    links:
      - cont-bd:mysql
    ports:
      - 8080:80
  cont-bd:
    image: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: mypasswd
    volumes:
      - datos-bd:/var/lib/mysql
  cont-phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - 8181:80
    environment:
      MYSQL_USERNAME: root
      MYSQL_ROOT_PASSWORD: mypasswd
      PMA_HOST: cont-bd
    volumes:
      datos-bd:
        driver: local
```