# Contenedores - Docker Compose, Podman, ...

Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

2025

# *Agenda*

1. Docker Compose
2. Podman
3. Estandarización

# Agenda

1. Docker Compose
2. Podman
3. Estandarización

# *¿Qué es Docker Compose?*

- *Docker Compose* es una herramienta para correr aplicaciones que requieren múltiples contenedores.
- Contenedores son llamados servicios en Docker Compose.
- Desarrollado en Orchad (lo llamaba Fig). Es adquirido por Docker en 2014.
- Versión 1 liberada en 2014 y desarrollada en Python.
- Versión 2 liberada en 2020 y escrita en Go.
- Facilita la creación de servicios, almacenamiento y red mediante un archivo YAML.
- El archivo YAML se ubica en el directorio de trabajo y recibe el nombre compose.yaml por default.
- También tiene comandos para iniciar, parar y construir servicios, monitorear los servicios en ejecución, logging, etc.

- docker compose up -d para iniciar todos los servicios. -d para que ejecuten en background (versiones anteriores usaban el comando docker-compose).
- Actualmente no es necesario indicar la versión (pero puede aparecer en antiguos archivos).
- Cada servicio termina siendo un contenedor.
- Se puede indicar la política de reinicio del contenedor si se detiene por algún motivo.
- Permite definir la dependencia de arranque entre contenedores.
- Se puede indicar que el contenedor se debe crear a partir de la imagen creada desde un Dockerfile.
- Por default, compose establece una red default a la que todos los contenedores se unen.
- Es posible definir redes propias para cada compose.

# Docker Compose . . .

```
version: '3.9'
services:
 wordpress:
  container_name: wp-so
  image: wordpress
  ports:
   - 8080:80
  environment:
   - WORDPRESS_DB_NAME=wordpress
   - WORDPRESS_DB_USER=wordpress
   - WORDPRESS_DB_PASSWORD=wordpresspwd
  depends_on:
   - mysql
  networks:
   - backend
 mysql:
  image: mysql:8.0.39
  volumes:
   - /home/docker/mysql:/var/lib/mysql
  environment:
   - MYSQL_ROOT_PASSWORD=wordpressdocker
   - MYSQL_DATABASE=wordpress
   - MYSQL_USER=wordpress
   - MYSQL_PASSWORD=wordpresspwd
  restart: always
  networks:
   - backend
networks:
 backend:
  driver: bridge
```

# Agenda



- Podman es una abreviación de *Pod Manager*.
- Es un container engine *daemonless* para desarrollar, administrar y ejecutar contenedores OCI (Open Container Initiative) en sistemas Linux.
- Un *pod* comprende uno o más contenedores que comparten los mismos namespaces.
- Utiliza prácticamente los mismos comandos que Docker (incluso tiene un podman compose)
- Permite ejecutar imágenes con el formato OCI, tanto como Docker (v1 y v2).
- Soporta todos los runtimes de OCI: runc, crun, etc.
- También pueden ser ejecutados en Windows y MAC.

- No necesita un proceso demonio central para administrar los contenedores.
- Contenedores inician como procesos standard del sistema.
- Basado en la librería *libpod* que contiene toda la lógica necesaria para instrumentar el ciclo de vida de un contenedor:
  - Formato de las imágenes, tanto Docker como OCI. Autenticación, descarga y almacenamiento de imágenes desde una registry, construcción de nuevas imágenes, etc.
  - Ciclo de vida de los contenedores: crear, ejecutar, eliminar, etc. contenedores.
  - Manejo tanto de simple contenedores como de pods.
  - Aislamiento de los contenedores/pods (mediante cgroups a bajo nivel).
  - CLI para administración de los contenedores/pods
  - Soporte de contenedores/pods rootless.
- libpod interactúa con los runtimes.

# ¿Qué es un Pod?

- Concepto que proviene de Kubernetes en el que los contenedores se ejecutan en Pods.
- Uno o mas contenedores trabajando en conjunto con un propósito en común.
- Representa un conjunto de contenedores que comparten almacenamiento y una única IP.
- Servicios en los contenedores dentro del pod se pueden comunicar entre sí usando localhost.
- Pods se pueden crear vacíos y luego agregarle contenedores.
- Ventajas de agrupar dos o mas contenedores:
  - Compartir algunos namespaces y cgroups
  - Compartir el volúmenes para almacenar datos persistentes.
  - Compartir la misma configuración.
  - Compartir el mismo IPC

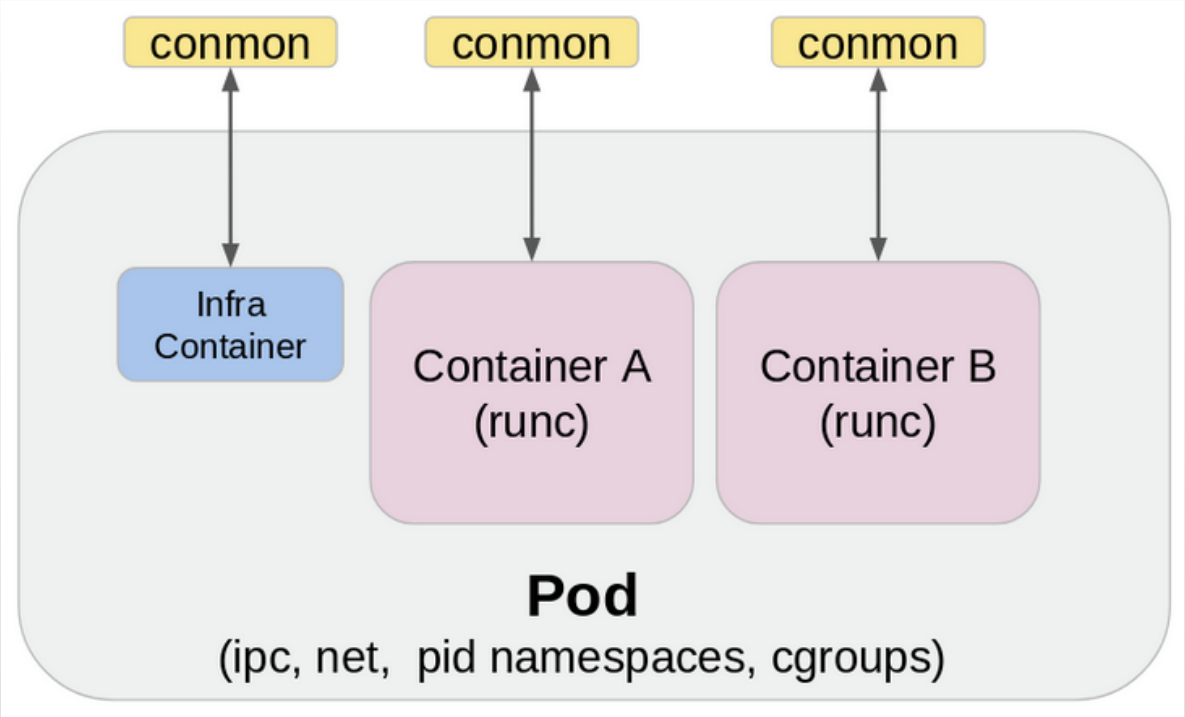
# ¿Qué es un Pod? . . .

- Cada pod incluye un contenedor llamado *infra*.
- También se lo suele llamar el *pause container*.
- Su finalidad es mantener abiertos los namespaces asociados con el pod.
- Al agregarse un contenedor al pod, los procesos comparten varios namespaces del pod.
- Al compartir el *net namespace*, los procesos se comunican usando localhost (127.0.0.1).
- Es posible iniciar/detener un pod y/o un contenedor dentro de un pod.
- Mayoría de los atributos se asignan al contenedor infra: port binding, namespaces, cgroups.
- Si se desea cambiar un atributo se debe regenerar el pod (por ej. agregar un contenedor que escuche en un nuevo puerto).

# ¿Qué es un Pod? . . .

- Por cada contenedor dentro del pod existe un proceso *conmon*.
- *Conmon* es un programa C liviano que monitorea un contenedor hasta que finaliza.
- Es una herramienta de comunicación entre el container engine (Podman) y el OCI runtime (runc o crun).
- Ejecuta el runtime, indicándole donde se encuentra el archivo *OCI spec* y el rootfs (capa que será el punto de montaje en el contenedor).
- Su principal tarea es monitorear el proceso principal del contenedor.
- Salva el código de salida si el contenedor muere.
- Mantiene la tty del contenedor abierta para poder conectarse a él.

- Contenedores comparten los namespaces de tipo PID, networking e IPC.




Ref: https://developers.redhat.com/blog/2019/01/15/podman-managing-containers-pods#podman_pods__what_you_need_to_know

# Pods . . .

```
[root@contenedores /]# podman pod create --name mipod
[root@contenedores /]# podman pod ps
POD ID         NAME        STATUS      CREATED          INFRA ID      # OF
    CONTAINERS
d2e184774bb2   mipod       Running     10 minutes ago   58b581b165eb  1

podman run -dt --pod mipod --name apache registry.access.redhat.com/rhscl/httpd
    -24-rhel7
[root@contenedores /]# podman run -dt --pod mipod --name apache registry.access
    .redhat.com/rhscl/httpd-24-rhel7
Trying to pull registry.access.redhat.com/rhscl/httpd-24-rhel7:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob ea092d7970b2 skipped: already exists
Copying blob 7f2c2c4492b6 skipped: already exists
Copying blob fd77da0b900b done
Copying config 847db19d6c done
Writing manifest to image destination
Storing signatures
5c6bf5cda3c3af6c5dbe93b487b270b6c173bfcb60416f75f2e68a5fe5fb24d9
[root@contenedores /]# podman pod ps
POD ID         NAME        STATUS      CREATED          INFRA ID      # OF
    CONTAINERS
d2e184774bb2   mipod       Running     33 minutes ago   58b581b165eb  2
```

# Pods . . .

```
[root@contenedores /]# podman ps
CONTAINER ID  IMAGE
COMMAND               CREATED         STATUS         PORTS NAMES
58b581b165eb  localhost/podman-pause:4.4.1-1682527828
                      48 minutes ago Up 46 minutes       d2e184774bb2-infra
5c6bf5cda3c3  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest
/usr/bin/run-http...  15 minutes ago Up 15 minutes       apache

[root@contenedores /]# lsns
        NS TYPE   NPROCS     PID USER   COMMAND
4026531834 time      266       1 root   /usr/lib/systemd/systemd.....
4026531835 cgroup    148       1 root   /usr/lib/systemd/systemd.....
4026531836 pid       148       1 root   /usr/lib/systemd/systemd.....
4026531837 user      266       1 root   /usr/lib/systemd/systemd.....
4026531838 uts       125       1 root   /usr/lib/systemd/systemd.....
......
4026532257 net        11 1805613 root   /catatonit -P
4026532788 mnt         1 1805613 root   /catatonit -P
4026532789 uts        11 1805613 root   /catatonit -P
4026532790 ipc        11 1805613 root   /catatonit -P
4026532791 pid         1 1805613 root   /catatonit -P
4026532792 cgroup      1 1805613 root   /catatonit -P
4026532793 mnt        10 1805618 1001   httpd -D FOREGROUND
4026532794 pid        10 1805618 1001   httpd -D FOREGROUND
4026532795 cgroup     10 1805618 1001   httpd -D FOREGROUND
```

```
[root@contenedores /]# ls -l /proc/1805618/ns/
total 0
lrwxrwxrwx 1 1001 root 0 may  2 11:01 cgroup -> 'cgroup:[4026532795]'
lrwxrwxrwx 1 1001 root 0 may  2 11:01 ipc -> 'ipc:[4026532790]'
lrwxrwxrwx 1 1001 root 0 may  2 11:01 mnt -> 'mnt:[4026532793]'
lrwxrwxrwx 1 1001 root 0 may  2 11:01 net -> 'net:[4026532257]'
lrwxrwxrwx 1 1001 root 0 may  2 11:01 pid -> 'pid:[4026532794]'
lrwxrwxrwx 1 1001 root 0 may  2 11:02 pid_for_children -> 'pid:[4026532794]'
lrwxrwxrwx 1 1001 root 0 may  2 11:01 time -> 'time:[4026531834]'
lrwxrwxrwx 1 1001 root 0 may  2 11:02 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 1001 root 0 may  2 11:01 user -> 'user:[4026531837]'
lrwxrwxrwx 1 1001 root 0 may  2 11:01 uts -> 'uts:[4026532789]'

[root@contenedores /]# ls -l /proc/1805613/ns/
total 0
lrwxrwxrwx 1 root root 0 may  2 11:01 cgroup -> 'cgroup:[4026532792]'
lrwxrwxrwx 1 root root 0 may  2 11:01 ipc -> 'ipc:[4026532790]'
lrwxrwxrwx 1 root root 0 may  2 11:01 mnt -> 'mnt:[4026532788]'
lrwxrwxrwx 1 root root 0 may  2 11:01 net -> 'net:[4026532257]'
lrwxrwxrwx 1 root root 0 may  2 11:01 pid -> 'pid:[4026532791]'
lrwxrwxrwx 1 root root 0 may  2 11:03 pid_for_children -> 'pid:[4026532791]'
lrwxrwxrwx 1 root root 0 may  2 11:01 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 may  2 11:03 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 may  2 11:01 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 may  2 11:01 uts -> 'uts:[4026532789]'
```

# Pods . . .

```
[root@contenedores /]# podman run -dt --pod mipod --name busypod a416a98b71e2
b79d1376807f51aa2b7e6ebfc4ecd3688749d011eb25b14144e0e3040a3dcc2e
[root@contenedores /]# podman pod ps
POD ID        NAME        STATUS      CREATED       INFRA ID      # OF CONTAINERS
d2e184774bb2  mipod       Running     2 hours ago   58b581b165eb  3
[root@contenedores /]# lsns
NS         TYPE   NPROCS     PID USER             COMMAND
4026532257 net        12 1805613 root             /catatonit -P
4026532788 mnt         1 1805613 root             /catatonit -P
4026532789 uts        12 1805613 root             /catatonit -P
4026532790 ipc        12 1805613 root             /catatonit -P
4026532791 pid         1 1805613 root             /catatonit -P
4026532792 cgroup      1 1805613 root             /catatonit -P
4026532793 mnt        10 1805618 1001             httpd -D FOREGROUND
4026532794 pid        10 1805618 1001             httpd -D FOREGROUND
4026532795 cgroup     10 1805618 1001             httpd -D FOREGROUND
4026532796 mnt         1 1806136 root             sh
4026532797 pid         1 1806136 root             sh
4026532798 cgroup      1 1806136 root             sh
```

# Pods . . .

```
[root@contenedores /]# ls -l /proc/1806136/ns
total 0
lrwxrwxrwx 1 root root 0 may  2 11:23 cgroup -> 'cgroup:[4026532798]'
lrwxrwxrwx 1 root root 0 may  2 11:23 ipc -> 'ipc:[4026532790]'
lrwxrwxrwx 1 root root 0 may  2 11:23 mnt -> 'mnt:[4026532796]'
lrwxrwxrwx 1 root root 0 may  2 11:23 net -> 'net:[4026532257]'
lrwxrwxrwx 1 root root 0 may  2 11:23 pid -> 'pid:[4026532797]'
lrwxrwxrwx 1 root root 0 may  2 12:00 pid_for_children -> 'pid:[4026532797]'
lrwxrwxrwx 1 root root 0 may  2 11:23 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 may  2 12:00 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 may  2 11:23 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 may  2 11:23 uts -> 'uts:[4026532789]'

[root@contenedores /]# podman ps --pod
CONTAINER ID IMAGE                                COMMAND               CREATED
STATUS              PORTS     NAMES                POD ID        PODNAME
58b581b165eb localhost/podman-pause:4.4.1...                            3 hours ago
Up About an hour              d2e184774bb2-infra   d2e184774bb2  mipod
5c6bf5cda3c3 registry.access.redhat.com/...       /usr/bin/run... 2 hours ago
Up About an hour              apache               d2e184774bb2  mipod
b79d1376807f docker.io/library/busybox:latest     sh                    About an hour ago
Up About an hour              busypod              d2e184774bb2  mipod

[root@contenedores /]# podman exec -it b79d1376807f /bin/sh
/ # wget localhost:8080
Connecting to localhost:8080 (127.0.0.1:8080)
wget: server returned error: HTTP/1.1 403 Forbidden
```

# Agenda



- Docker, en sus inicios (2013), usaba LXC como su motor de contenedores.
- Un año más tarde introdujo su propia librería, *libcontainer*, para reemplazar a LXC.
- En 2014/2015, la compañia CoreOS lanzó su propio motor de contenedores, rkt, que era *daemon-less*. Luego comprada por RedHat.
- En 2017, la *Cloud Native Computing Foundation(CNCF)*, cuya meta es coordinar proyectos relacionados a la nube y contenedores, decidió adoptar rkt y containerd (donado por Docker).
- *Containerd* es el runtime usado por Docker Engine (en conjunto con runc).
- En 2015, Docker, en conjunto con RedHat, AWS, Google, etc., inicia el Open Container Iniciative (OCI) auspiciado por la Linux Foundation.

- OCI se encarga de realizar la especificación de runtime y de imágenes.
- También lanzó la primera implementación de un runtime de contenedores que cumplen con esa especificación: *runc*
- Además, la OCI definió la base para una más directa conexión en Kubernetes y el correspondiente engine.
- La comunidad de Kubernetes libera CRI (Container Runtime Interface), plugin que permite la adopción de una amplia variedad de runtimes.
- En 2017, RedHat libera CRI-O que permite el uso de runtimes compatibles con OCI. Es una alternativa liviana a usar Docker, rkt, etc.

# *Contenedores Orquestador - Conceptos*

- Cluster: grupo de nodos interconectados que trabajan en conjunto.
- Permite aprovisionar, desplegar, escalar y administrar automáticamente contenedores sin preocuparse por la infraestructura subyacente.
- Creación de servicios de manera declarativa.
- En general, dos tipos de nodos:
  - **Manager:** encargado de administrar el cluster.
  - **Worker:** encargado de ejecutar las aplicaciones.
- *Service Discovery:* orquestador brinda información para encontrar otro servicio.
- *Routing:* paquetes deben llegar entre servicios ejecutando en diferentes nodos.
- *Load Balancing:* distribuir las cargas de trabajos entre las distintas instancias de un servicio
- *Scaling:* aumentar/disminuir las instancias de un servicios según la carga de trabajo.
- Kubernetes, Docker Swarm, RedHat Openshift, Rancher, etc.

# Referencias

- https://docs.docker.com/engine/reference/commandline/
- https://docs.docker.com/compose/
- https://podman.io/docs
- https://galvarado.com.mx/post/container-runtimes/

¿Preguntas?