
fedora
CONTAINERS LAB
Fedora México
Alex Callejas
Senior Technical Support Engineer | Red Hat
Abril 2019

About me

Alex Callejas
Senior Technical Support Engineer @Red Hat

@dark_axl

/rootzilopochtli

www.rootzilopochtli.com

Geek by nature, Linux by choice, Fedora of course!

redhat.

Contenedores

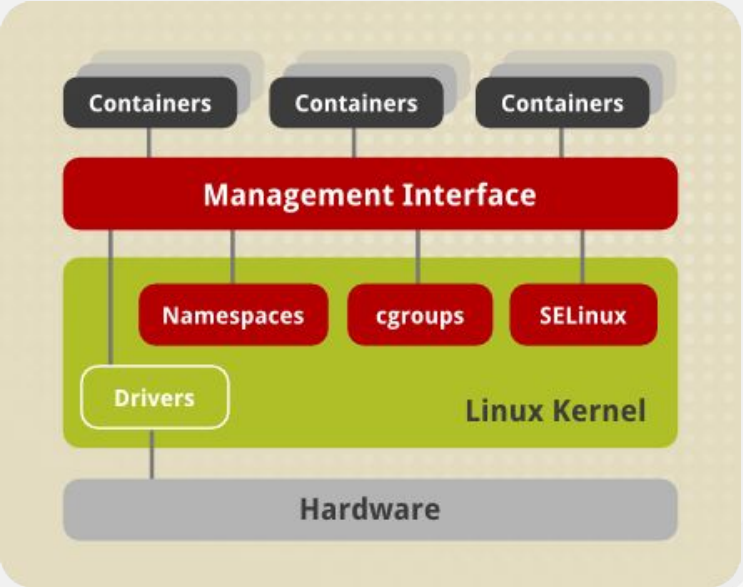
Conceptos Básicos

Un contenedor de Linux es un conjunto de uno o más procesos que se encuentran aislados del resto del sistema.

El *kernel* proporciona sus componentes principales:

- ***Namespaces***: para asegurar el aislamiento de procesos
- ***cgroups***: para el control de los recursos del sistema
- ***SELinux***: para asegurar la separación entre el host y el contenedor, y también entre los contenedores

La *interfaz de administración* interactúa con los componentes del kernel y proporciona herramientas para la construcción y gestión de contenedores.



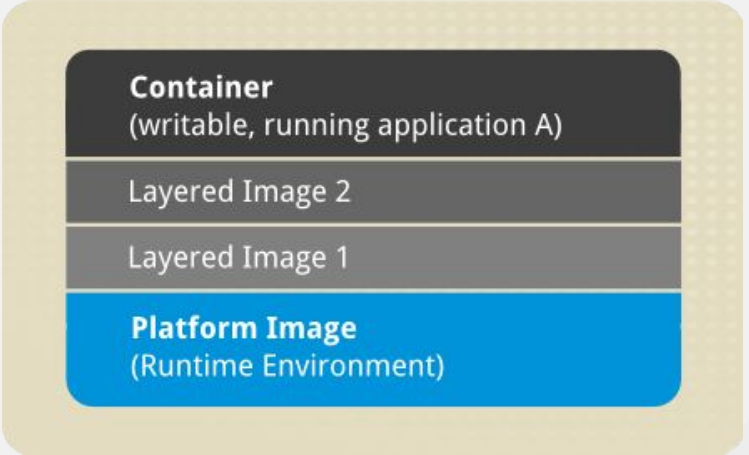
Contenedores

Conceptos Básicos

Todos los archivos que se necesitan para ejecutar un contenedor provienen de una imagen.

- ***Container***: (en sentido estricto) Un componente activo en el que se ejecuta una aplicación. Cada contenedor se basa en una imagen que contiene los datos de configuración necesarios.
- ***Image***: Un *snapshot* estático de la configuración del contenedor. La imagen es una capa *read-only* que nunca se modifica,
- ***Platform Image***: Define el runtime, los paquetes y las utilidades necesarios para que se ejecute una *containerized application*.

Las imágenes del contenedor se almacenan en un registro de imágenes, que contiene todas sus versiones

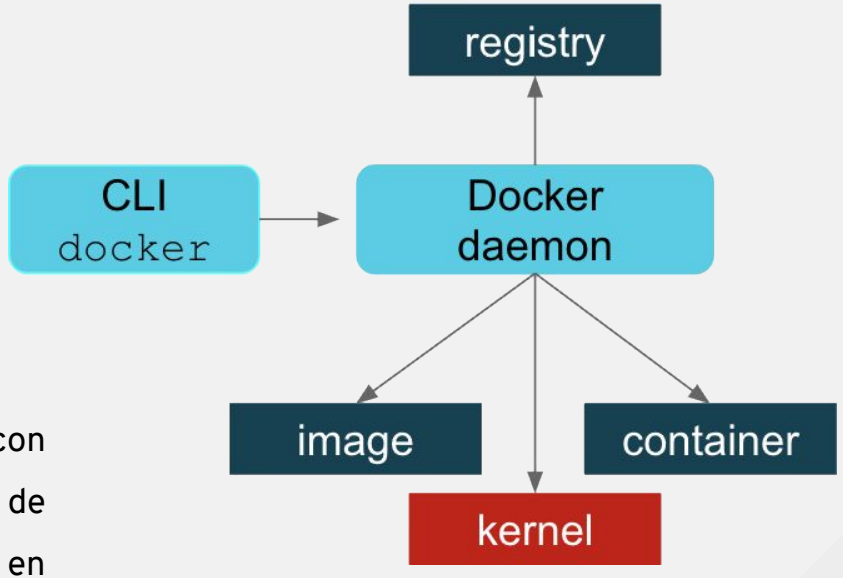


¿Cómo funciona Docker?

Docker proporciona toda la funcionalidad necesaria para:

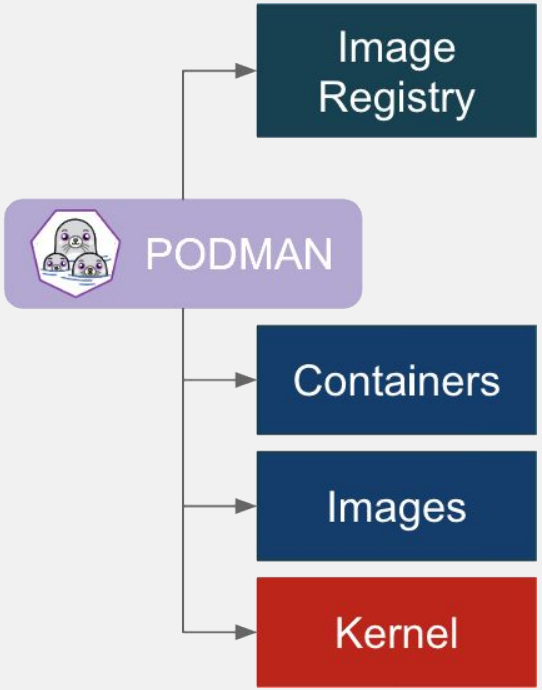
- “Pull & push” de imágenes de un registro de imágenes
- Administrar contenedores locales:
  - Copy, add layers, commit & remove
- Pedir al kernel que ejecute un contenedor con el name-space y cgroup correctos, etc.

Esencialmente, el demonio Docker hace todo el trabajo con registros, imágenes, contenedores y el kernel. La línea de comandos (CLI) de Docker le pide al demonio que haga esto en su nombre.



PODMAN

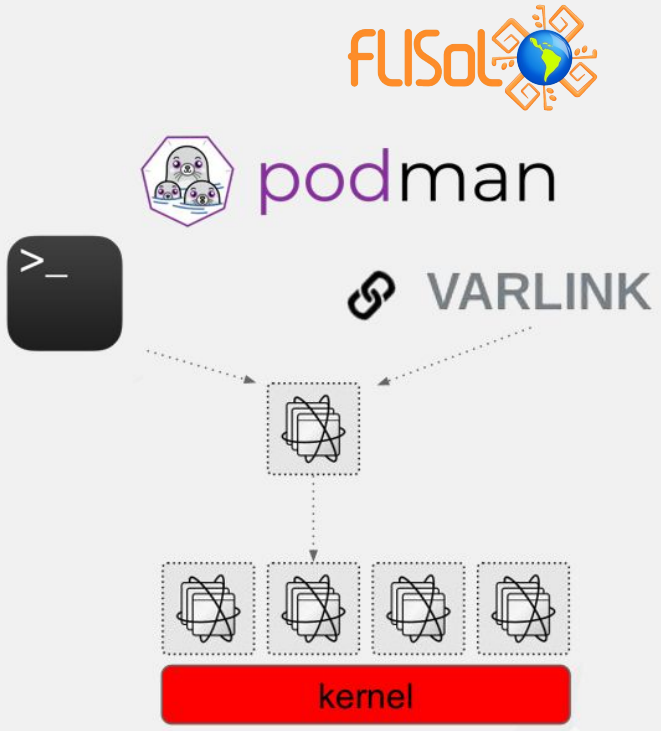
El enfoque de podman es simplemente interactuar directamente con el registro de imágenes, con el contenedor y el almacenamiento de imágenes, y con el kernel de Linux a través del proceso de ejecución (*runC*) del contenedor.



PODMAN

Un CLI/API sin daemon para ejecutar, administrar y depurar contenedores y pods OCI

- Rápido y ligero
- runC compartido
- Proporciona una sintaxis "*tipo docker*" para trabajar con contenedores
- API de gestión remota a través de varlink
- Proporciona integración de sistemas y aislamiento avanzado de namespace



PODMAN

Primeros pasos

podman

Instalar podman y buildah

```
# dnf -y install podman buildah skopeo
```

Bajar una imagen del registro e inspeccionarla

```
# podman pull registry.fedoraproject.org/f29/httpd
```

```
# podman images
```

```
# podman inspect httpd
```


redhat.

PODMAN

Primeros pasos

podman

Ejecutando el contenedor

```
# podman run httpd
```

Revisamos el proceso

```
# systemctl status podman
```

```
# podman ps
```

```
# podman ps -a
```

How does Docker generate default container names?

PODMAN

Primeros pasos

podman

Ejecutando el contenedor en background

```
# podman run --name myhttpservice -d httpd
```

Inspeccionamos el contenedor en busca de ip y puertos expuestos

```
# podman inspect myhttpservice | grep -i ipaddr
```

```
# podman inspect myhttpservice | grep expose-services
```

```
# curl 10.88.0.2:8080
```


redhat.

PODMAN

Inmutabilidad

podman

Ejecutamos el contenedor en background

```
# podman run -d httpd
```

Ejecutamos bash dentro del contenedor (sesión interactiva)

```
# podman exec -ti c94e745f6414 /bin/bash
```

Creamos un archivo dentro del contenedor

```
bash-4.4$ echo "MySecretData" > my.data
```

PODMAN

Inmutabilidad

podman

Detenemos el contenedor

```
# podman kill c94e745f6414
```

Ejecutamos el contenedor nuevamente y entramos en sesión interactiva

```
# podman run -d httpd
```

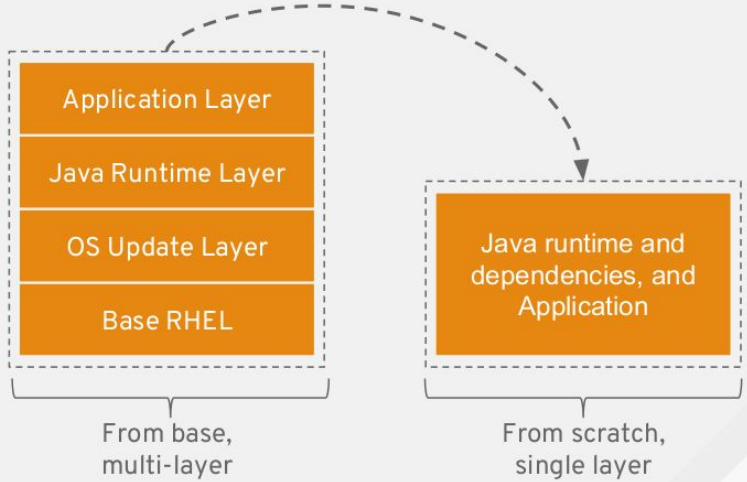
```
# podman exec -ti 40f5b5fd89bb /bin/bash
```

```
bash-4.4$ ls
```

CONSTRUYENDO CONTENEDORES

¿Porqué usar buildah?

- Crea imágenes compatibles con OCI
- No daemon - sin socket docker
- No requiere un contenedor en ejecución
- Puede utilizar las suscripciones de hosts y otros secretos
- Control preciso sobre los comandos y el contenido de la capa (s)
- Una sola capa, desde cero, las imágenes se hacen fáciles y aseguran un manifiesto limitado
- Si es necesario, puede mantener el flujo de trabajo basado en Dockerfile



CONSTRUYENDO CONTENEDORES

Probando buildah

Hola Mundo

```
# echo "hello world" > $(buildah mount $(buildah from
registry.fedoraproject.org/fedora-minimal))/etc/hello.txt
```

Revisamos la creación de la imagen base

```
# buildah containers
```

Hacemos commit a la imagen local

```
# buildah commit fedora-minimal-working-container fedora-hello
```

```
# buildah images
```

CONSTRUYENDO CONTENEDORES

Probando buildah

buildah

Eliminamos la imagen base

```
# buildah delete fedora-minimal-working-container
```

Ejecutamos el contenedor

```
# podman run -ti localhost/fedora-hello:latest cat /etc/hello.txt
hello world
```

CONSTRUYENDO CONTENEDORES

Dockerfile

buildah

Creamos el Dockerfile

```
# Base on the Fedora
FROM registry.fedoraproject.org/fedora
MAINTAINER  darkaxl017 email dark.axl@gmail.com # not a real email

# Install httpd on image
RUN echo "Installing httpd"; dnf -y install httpd

# Expose the default httpd port 80
EXPOSE 80

# Run the httpd
CMD ["/usr/sbin/httpd", "-DFOREGROUND"]
```


redhat.

CONSTRUYENDO CONTENEDORES

buildah

Dockerfile

Ejecutamos buildah para la construcción de la imagen

```
# buildah bud -f Dockerfile -t fedora-httpd .
```

Revisamos la creación de la imagen

```
# buildah images
```

Probamos el contenedor

```
# buildah run $(buildah from fedora-httpd) httpd -v
```

CONSTRUYENDO CONTENEDORES

Dockerfile

Ejecutamos el contenedor

```
# podman run -d fedora-httpd
```

Revisamos el proceso del contenedor

```
# podman ps
```

Revisamos los procesos

```
# ps auxf | tail -6
```

CONSTRUYENDO CONTENEDORES

buildah

Dockerfile

Validamos el servicio

```
# curl localhost
curl: Failed to connect to localhost port 80: Connection refused
```

Revisamos los logs del contenedor

```
# podman logs 517495e24317
```

Revisamos los puertos

```
# netstat -tulpn
```

CONSTRUYENDO CONTENEDORES

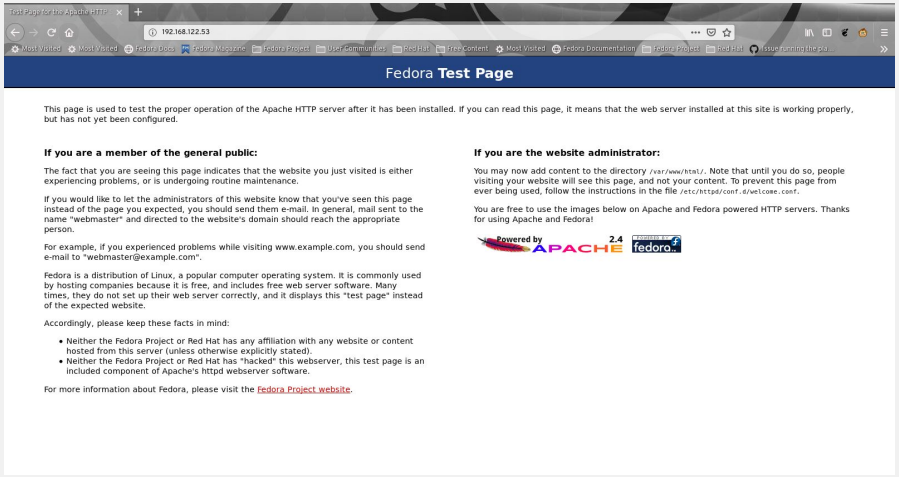
Dockerfile

Detenemos y ejecutamos el contenedor exponiendo el puerto

```
# podman run -d -p 80:80 fedora-httpd
```

Validamos el servicio

```
# curl localhost
```



Persistent Storage

Los contenedores son efímeros...

Creamos directorio compartido para el contenedor

```
# mkdir -p /opt/var/www/html
```

Creamos el contenido a compartir

```
# echo 'Hola Mundo' > /opt/var/www/html/index.html
```

Ejecutamos el contenedor compartiendo puerto y directorio

```
# podman run -d --name myhttpservice -p 8080:8080 -v
/opt/var/www/html:/var/www/html:Z registry.fedoraproject.org/f29/httpd
```

Healthcheck

Validando disponibilidad del servicio

Ejecutamos el contenedor con el comando para el healthcheck

```
# podman run -dt --name myhttpservice -p 8080:8080 -v /opt/var/www/html:/var/www/html:Z
--healthcheck-command 'CMD-SHELL curl http://localhost:8080 || exit 1' --healthcheck-interval=0
registry.fedoraproject.org/f29/httpd
```

Validamos el healthcheck del contenedor

```
# podman healthcheck run myhttpservice
healthy
```

[Monitoring container vitality and availability with Podman]

Containerized System Services

Los contenedores son portátiles y están listos para usar: ¿por qué no usarlos como servicios del sistema?

/etc/systemd/system/myhttpservice.service

```
[Unit]
Description=Just a http service with Podman Container

[Service]
Type=simple
TimeoutStartSec=30s
ExecStartPre=-/usr/bin/podman rm "myhttpservice"

ExecStart=/usr/bin/podman run --name myhttpservice -p 8080:8080 -v /opt/var/www/html:/var/www/html:Z
--healthcheck-command 'CMD-SHELL curl http://localhost:8080 || exit 1'
--healthcheck-interval=0registry.fedoraproject.org/f29/httpd

ExecReload=-/usr/bin/podman stop "myhttpservice"
ExecReload=-/usr/bin/podman rm "myhttpservice"
ExecStop=-/usr/bin/podman stop "myhttpservice"
Restart=always
RestartSec=30

[Install]
WantedBy=multi-user.target
```

Containerized System Services

Los contenedores son portátiles y están listos para usar: ¿por qué no usarlos como servicios del sistema?

Refrescamos systemd

```
# systemctl daemon-reload
```

Revisamos status del servicio

```
# systemctl status myhttpservice.service
```

Iniciamos el servicio

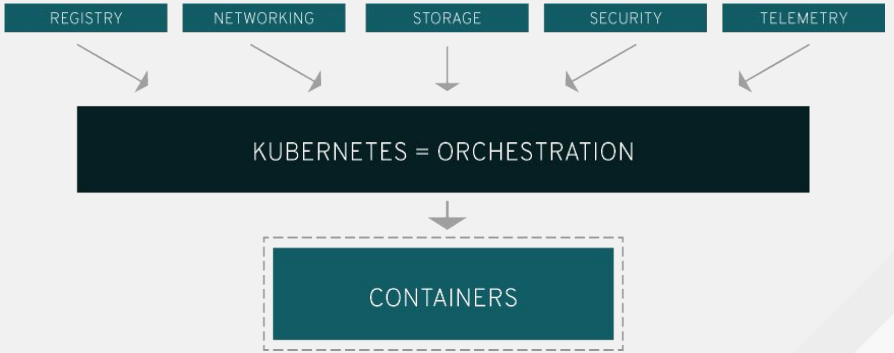
```
# systemctl start myhttpservice.service
```

```
# systemctl status myhttpservice.service
```

Kubernetes

Conceptos básicos

- Kubernetes = Orquestación
  - Elimina muchos de los procesos manuales involucrados en la implementación y escalabilidad de las aplicaciones en contenedores
  - Ayuda a administrar con facilidad y eficacia un clúster de grupos de hosts que ejecutan contenedores de Linux
  - Permite diseñar servicios de aplicaciones que abarcan varios contenedores, programar estos contenedores en un clúster, ampliarlos y gestionar su estado a lo largo del tiempo.

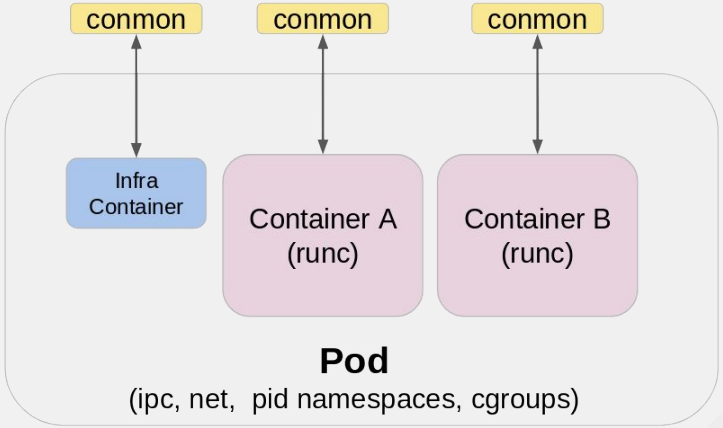


Podman pods

Lo que necesitas saber

El concepto de **Pod** fue introducido por Kubernetes: un grupo de uno o más contenedores implementados en un nodo único.

- Cada podman pod incluye un contenedor "infra"
  - Mantiene los namespaces asociados con el pod y permite a podman conectarse a los otros contenedores
  - Se basa en la imagen `k8s.gcr.io/pause`
  - Se le asignan los port bindings, cgroup-parent values, y kernel namespaces del pod
  - Una vez que se crea el pod, estos atributos se asignan al contenedor "infra" y no se pueden cambiar
- Cada contenedor tiene su propio monitor (**conmon**)
  - Monitorea el proceso primario del contenedor y guarda el exit code si se termina o muere el contenedor
  - Permite que podman se ejecute en modo detached (background)



Podman pods

Primeros pasos

Creamos el podman pod

```
# podman pod create
```

Listamos los pod's

```
# podman pod list
```

Agregamos un contenedor al pod y lo revisamos

```
# podman run -dt --pod  [pod id] docker.io/library/alpine:latest top
```

```
# podman ps -a --pod
```

Podman pods

Ejemplo práctico: MariaDB container

Creando el podman pod

```
# podman run -dt -e MYSQL_ROOT_PASSWORD=x --pod new:db
registry.fedoraproject.org/f28/mariadb:latest
```

Revisamos el status de los pod's

```
# podman pod ps
```

Agregamos un contenedor al pod y revisamos la base de datos

```
# podman run -it --rm --privileged  --pod db docker.io/library/alpine:latest /bin/sh
```

```
/ # apk add mariadb-client
```

```
/ # mysql -u root -P 3306 -h 127.0.0.1 -p
```

Podman pods

Siguientes pasos?

Podman pods

Siguientes pasos: ejemplo práctico

Creamos un contenedor demo y lo validamos

```
# podman run -dt -p 8000:80 --name demo quay.io/libpod/alpine_nginx:latest
```

```
# podman ps
```

```
# curl http://localhost:8000
```

Generamos snapshot para crear el archivo YAML de Kubernetes

```
# podman generate kube demo > demo.yml
```

Con el archivo yml podemos recrear el contenedor/pod en kubernetes

```
# kubectl create -f demo.yml
```

Kubernetes

Configuración inicial

- **Fedora (Single Node) -** https://kubernetes.io/docs/getting-started-guides/fedora/fedora_manual_config/
- **Introduction to Kubernetes with Fedora -** https://fedoramagazine.org/introduction-kubernetes-fedora/
- **Clustered computing on Fedora with Minikube -** https://fedoramagazine.org/minikube-kubernetes/


AVENGE THE FALLEN
FLISoL
redhat.

Referencias

Links y documentación

- Fedora Classroom: Containers 101 with Podman
- Getting Started with Buildah
- Managing containerized system services with Podman
- Podman: Managing pods and containers in a local container runtime
- Podman can now ease the transition to Kubernetes and CRI-O

- https://registry.fedoraproject.org/
- https://registry.centos.org/containers/

- https://podman.io/
- https://github.com/containers/buildah

- Daniel Walsh - @rhatdan


fedora
Gracias!
fedoracommunity.org/latam
https://t.me/fedoralat
https://t.me/fedoramexico