
# Fedora Containers Lab

**Alex Callejas**
Services Content Architect @ Red Hat

@dark_axl
darkaxl017.fedorapeople.org/slides/
rutil.io/social

# ¿Porqué un laboratorio?


*It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong. In that simple statement is the key to science.*

Richard Feynman

# Laboratorio de Pruebas

Mi configuración

## Lenovo ThinkPad P1 Gen 2

Intel(R) Core(TM) i7-9850H CPU @ 2.60GHz

- **Storage**
  `/var/lib/libvirt/images` → LVM 200G
- **Fedora release 33 (ThirtyThree)**
  `5.11.17-200.fc33.x86_64`
- **KVM Packages**
  - qemu-kvm
  - virt-manager
  - virt-viewer
  - libguestfs-tools
  - virt-install
  - genisoimage

# Cloud Images

## KVM y QEMU

**QUEMU** soporta varios tipos de imágenes. El tipo "nativo" y más flexible es ***qcow2***, que admite la copia en escritura, el cifrado, la compresión y los snapshots de VM.

La forma más sencilla de obtener una máquina virtual que funciona con **KVM** es descargar una  imagen que alguien más haya creado:

- Fedora Cloud. Cloud Base Images [**https://alt.fedoraproject.org/cloud/**]
- Fedora CoreOS [https://getfedora.org/en/coreos/download?tab=cloud_launchable&stream=stable]
- OpenStack Docs: Get images [https://docs.openstack.org/image-guide/obtain-images.html]

# Containers Lab

## Creando VM's

Configurar VM:

```
$ sudo virt-customize -a /var/lib/libvirt/images/vmlab01.qcow2 \
  --hostname vmlab01.rootzilopochtli.lab  --root-password password:rootpw \
  --ssh-inject 'root:file:labkey.pub' --uninstall cloud-init --selinux-relabel
```

Importar VM:

```
$ sudo virt-install --name vmlab01 --memory 1024  \
  --vcpus 1 --disk /var/lib/libvirt/images/vmlab01.qcow2  \
  --import --os-type linux --os-variant fedora34 --noautoconsole
```

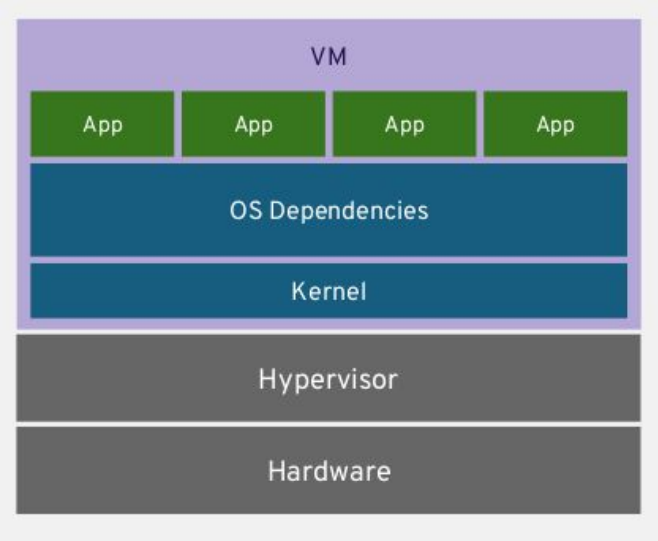
**Fuente:**

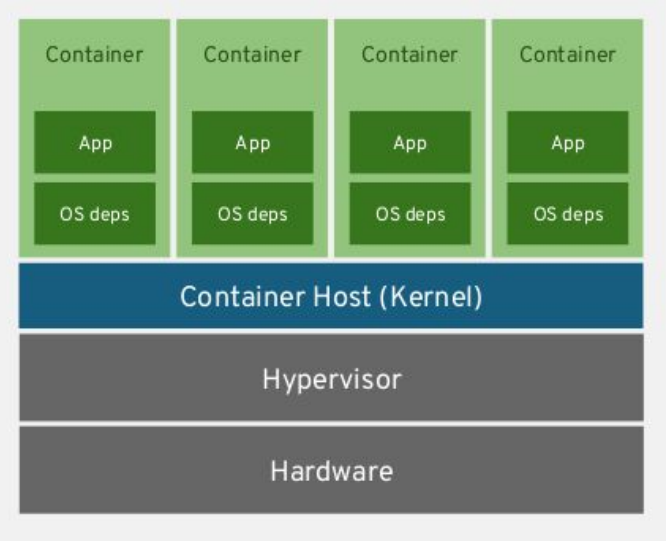
Modifying the Red Hat Enterprise Linux OpenStack Platform Overcloud Image with virt-customize [https://access.redhat.com/articles/1556833]

Creating Guests with virt-install [https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-guest_virtual_machine_installation_overview-creating_guests_with_virt_install]

# Containers Lab

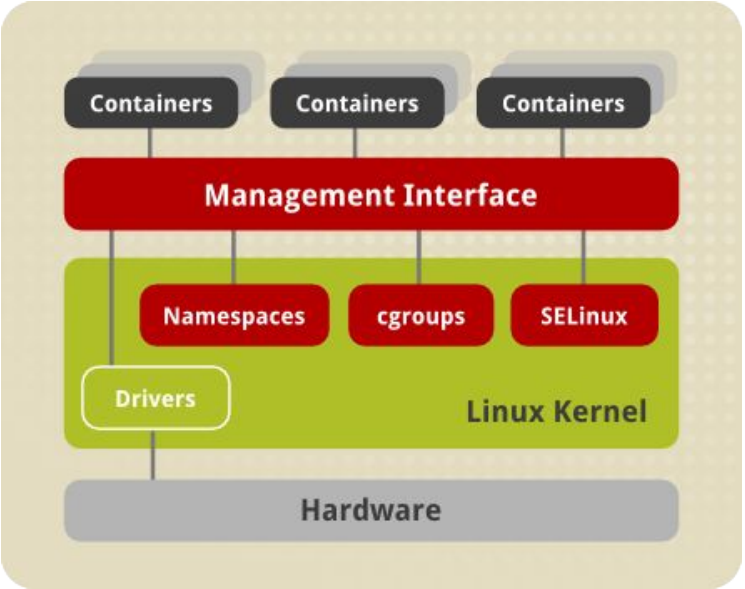
¿Contenedores?



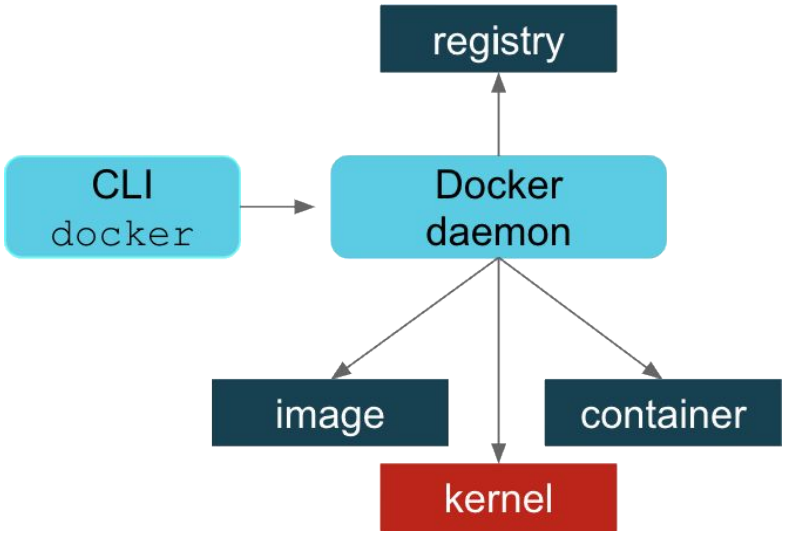
# Containers Lab

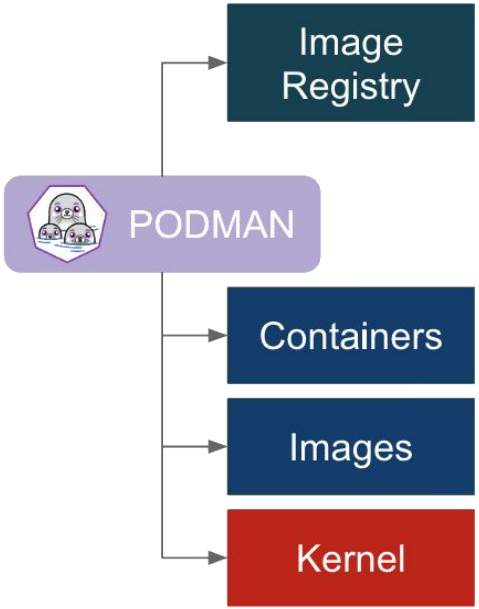
¿Contenedores?



# Containers Lab

Docker vs Podman



# Podman

```
# dnf install podman buildah skopeo
```

Proporciona una sintaxis “tipo Docker” para manejar los contenedores

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

```
# podman images
```

```
# podman inspect httpd
```

← skopeo

```
# podman run httpd
```

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

# Podman

```
# podman inspect myhttpservice | grep IPAddress
```

← skopeo

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

```
# curl 10.88.0.3:8080
```

## Inmutabilidad

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

```
# podman exec -ti 1b6f2145c74e /bin/bash
```

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

# Podman

## Construyendo contenedores (**buildah**)

Hola Mundo

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

Revisamos la creación de la imagen base

```
# buildah containers
```

Hacemos commit a la imagen local, eliminamos la imagen base y ejecutamos el contenedor

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

```
# buildah images
```

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

```
# podman run -ti localhost/fedora-hola:latest cat /etc/hola.txt
hola mundo
```

# Podman

## Construyendo contenedores

### Dockerfile

```
# Base on the Fedora
FROM registry.fedoraproject.org/fedora
MAINTAINER  darkaxl017 email dark.axl@fakemail.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"]
```

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

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

# Podman

Ejecutamos el contenedor

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

Revisamos el proceso del contenedor

```
# podman ps
```

Revisamos los procesos

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

```
# curl localhost
curl: (7) Failed to connect to localhost port 80: Connection refused
```

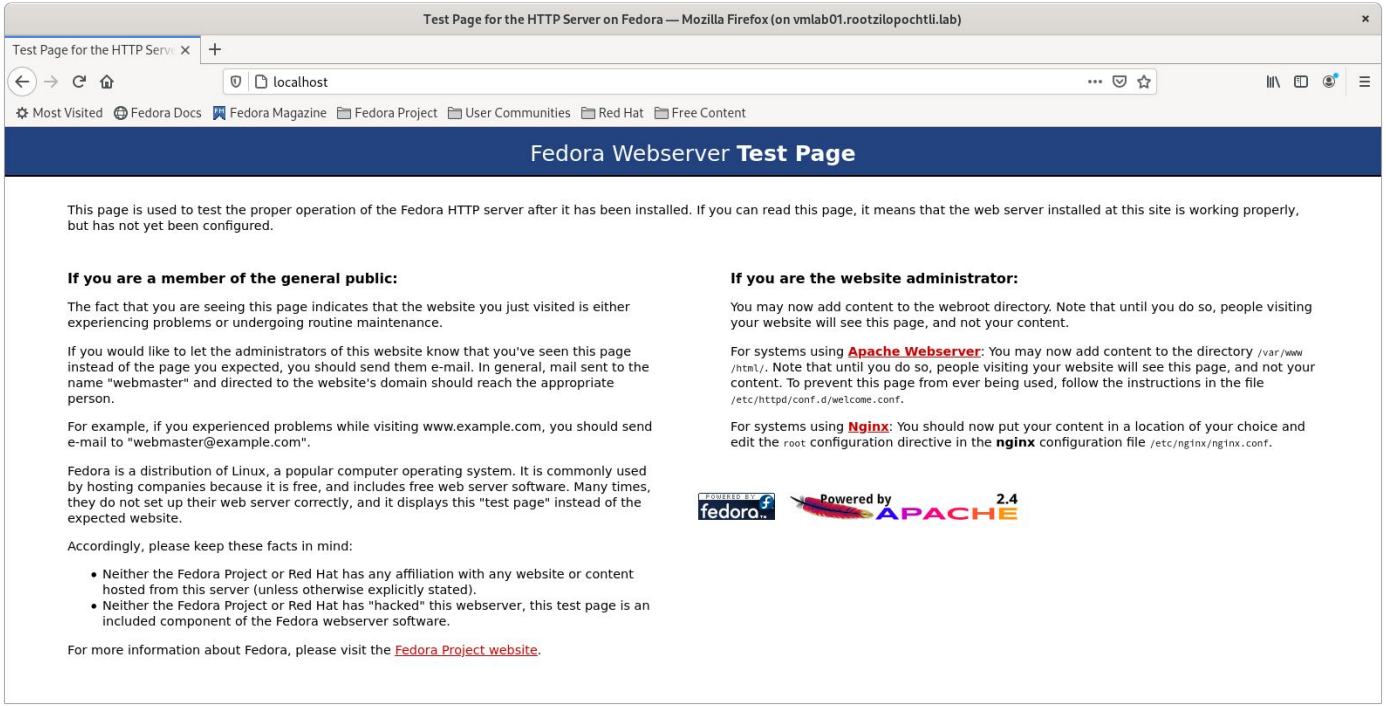
```
# podman logs 5e7c8d39741e
```

# Podman

Detenemos el contenedor y lo ejecutamos exponiendo el puerto 80

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

```
# curl localhost
```



# Podman

## Persistent Storage

Creamos el directorio compartido para el contenedor

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

Creamos el contenido a compartir

```
# echo "Hola Mundo" > index.html
```

Ejecutamos el contenedor compartiendo el puerto y el directorio creado

```
# podman run -d --name myhttpservice -p 8080:8080 -v /opt/var/www/html:/var/www/html:Z httpd
```

Ejecutamos el contenedor con **healthcheck**

```
# podman run -d --name myhttpservice -p 8080:8080 -v /opt/var/www/html:/var/www/html:Z \
  --health-cmd="curl http://localhost:8080 || exit 1" --health-interval=0 httpd
```

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

# Podman

## Construyendo servicios

**/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 --health-cmd 'CMD-SHELL curl
http://localhost:8080 || exit 1' --health-interval=0
registry.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
```

# Podman

## Construyendo servicios

Refrescamos systemd

```
# systemctl daemon-reload
```

Revisamos el status del servicio

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

Iniciamos el servicio

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

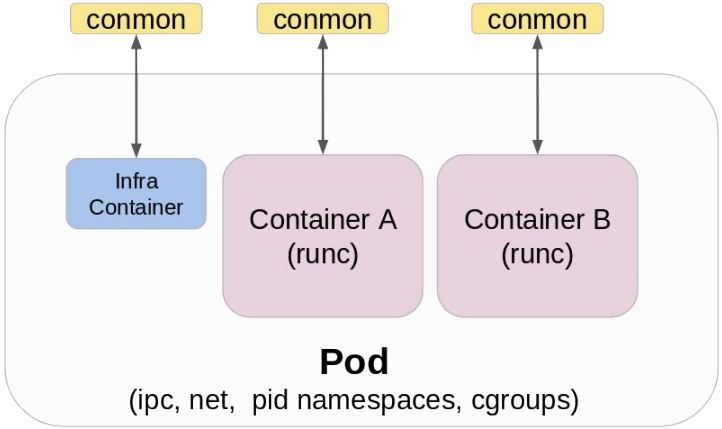
```
# systemctl status myhttpservice.service
```

# Podman Pods

## Lo que necesitas saber

El concepto de **Pod** fue introducido por **Kubernetes**[1]. Los **podman pods** son similares a esa definición.

- 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* de Kubernetes
  - 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 Pod*

[1] **Pods** are the *smallest deployable units* of computing that you can create and manage in Kubernetes. [https://kubernetes.io/docs/concepts/workloads/pods/]

# Podman Pods

## Primeros Pasos

Creamos el pod

```
# podman pod create
```

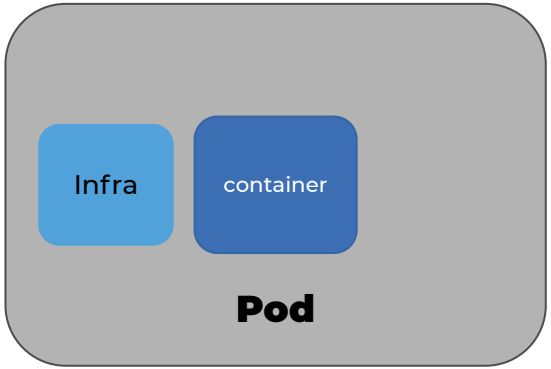
Listar los pod's

```
# podman pod list
```

Agregar el contenedor al pod

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

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



# Podman Pods

## Ejemplo práctico: MariaDB Pod

Creamos el pod

```
# podman run -dt -e MYSQL_ROOT_PASSWORD=x  --pod new:db registry.fedoraproject.org/f31/mariadb:latest
```

Revisamos el status del pod

```
# podman pod ps
```

Agregar un contenedor al pod para revisar la DB

```
# 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

## Siguientes Pasos → Kubernetes

Creamos el contenedor y lo validamos

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

```
# curl localhost:8000
podman rulez
```

Generamos un snapshot para crear el archivo YAML de kubernetes

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

Lo transferimos para cargarlo en **minishift**

```
# oc create -f demo.yml -n myproject
```

```
# oc status --suggest
In project My Project (myproject) on server https://192.168.42.219:8443

pod/demo runs quay.io/libpod/alpine_nginx:latest
```

**Major Hayden @ Devconf.cz** 🇨🇿
@majorhayden

Ready to take your container to Kubernetes? Use:

podman generate kube

to create Kubernetes yaml once you have everything working the way you want! #devconfcz

okd

docs.okd.io/minishift/getting-started

# Podman rootless containers

## Lo que necesitas saber

Algunas consideraciones al ejecutar containers como un usuario *non-root*:

- Las imágenes se guardan en el home directory del usuario (`$HOME/.local/share/containers/storage/`) en lugar de `/var/lib/containers`.
- Al ejecutar *rootless containers* se obtiene un permiso especial para ejecutarlos con un rango predefinido de ID's de usuarios y grupos en el host. Sin embargo, éstos no tienen privilegios de root para el sistema operativo del mismo, lo cual puede resultar en algunas situaciones, por ejemplo:
  - Un *rootless container* no tiene capacidad para acceder a un puerto inferior a **1024**. Dentro de su **namespace** puede, por ejemplo, iniciar un servicio que expone el puerto 80 con un servicio **httpd** dentro del contenedor, pero no será accesible fuera del **namespace**.
  - El almacenamiento debe estar en un sistema de archivos local, porque los sistemas de archivos remotos no funcionan bien con **namespaces** de usuarios sin privilegios.

# Podman rootless containers

## Primeros pasos

Incrementar los *user namespaces*

```
# echo "user.max_user_namespaces=28633" > /etc/sysctl.d/userns.conf
# sysctl -p /etc/sysctl.d/userns.conf
```

Creamos un contenedor con el usuario *non-root*

```
$ podman pull ubi8/ubi
```

```
$ podman run ubi8/ubi cat /etc/os-release
```

Checamos la configuración *rootless*

```
$ podman unshare cat /proc/self/uid_map
```

# Podman rootless containers

## Instalando una app: Linkding

Creamos un volumen para el contenedor

```
$ podman volume create linkding_data
```

Creamos el contenedor para la app

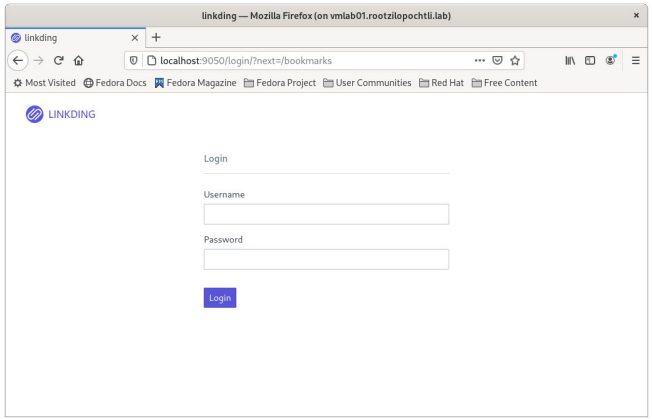
```
$ podman run --name linkding -p 9050:9090 -v linkding_data:/etc/linkding/data -d sissbruecker/linkding
```

Creamos un usuario para la app

```
$ podman exec -it linkding python manage.py createsuperuser --username=admin --email=yo@fakemail.com
```

Como root, agregamos al usuario a systemd

```
# loginctl enable-linger student
```




Creamos el servicio en systemd

```
$ mkdir -p .config/systemd/student & cd .config/systemd/student
$ podman generate systemd --name linkding --files
$ systemctl --user daemon-reload
$ systemctl --user enable --now /home/student/.config/systemd/student/container-linkding.service
```

**Fuente:** How To Setup Root Less Podman Containers!! [https://medium.com/devops-dudes/how-to-setup-root-less-podman-containers-efd109fa4e0d]

# 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

- Fedora Containers Lab Examples

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

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

- Daniel Walsh - @rhatdan



# Gracias!

**Únete a la conversación!**

@fedoramexico