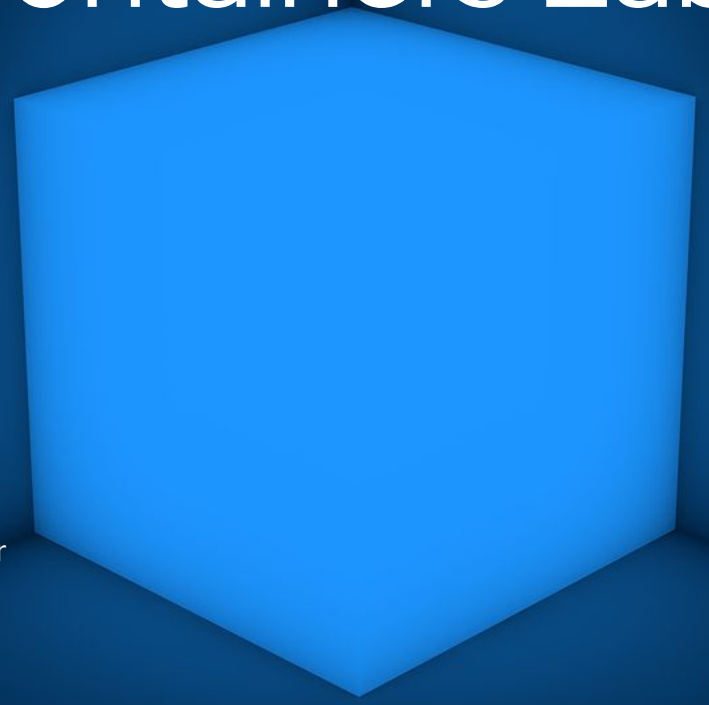
Fedora Containers Lab



Alex Callejas
Senior Technical Support Engineer
April 2020

SG VIRTUAL CONFERENCE

Alex Callejas
Senior Technical Support Engineer @Red Hat

@dark_axl

/rootzilopochtli

www.rootzilopochtli.com

darkaxl017.fedorapeople.org/slides/

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

¿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 T440s

Intel Core i7-4600U CPU @ 2.10GHz + 8G Mem

- **Storage**
  /var/lib/libvirt/images (4.2G) → / fs 50G
- **KVM packages**
  - qemu-kvm
  - virt-manager
  - virt-viewer
  - libguestfs-tools
  - virt-install
  - genisoimage

Cloud Images

KVM y QEMU

QEMU 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 ya haya creado:

- Fedora Cloud. Cloud Base Images [https://alt.fedoraproject.org/cloud/]
- Atomic Host [https://getfedora.org/en/atomic/download/]
- OpenStack: Get images [https://docs.openstack.org/image-guide/obtain-images.html]

Containers Lab

Creando VM's - Fedora 31

Configurar VM:

```
$ sudo virt-customize -a /var/lib/libvirt/images/vmlab01.qcow2 \
  --hostname vmlab01.mx.redhat.lab  --root-password password:redhat \
  --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-variant fedora31 --noautoconsole
```

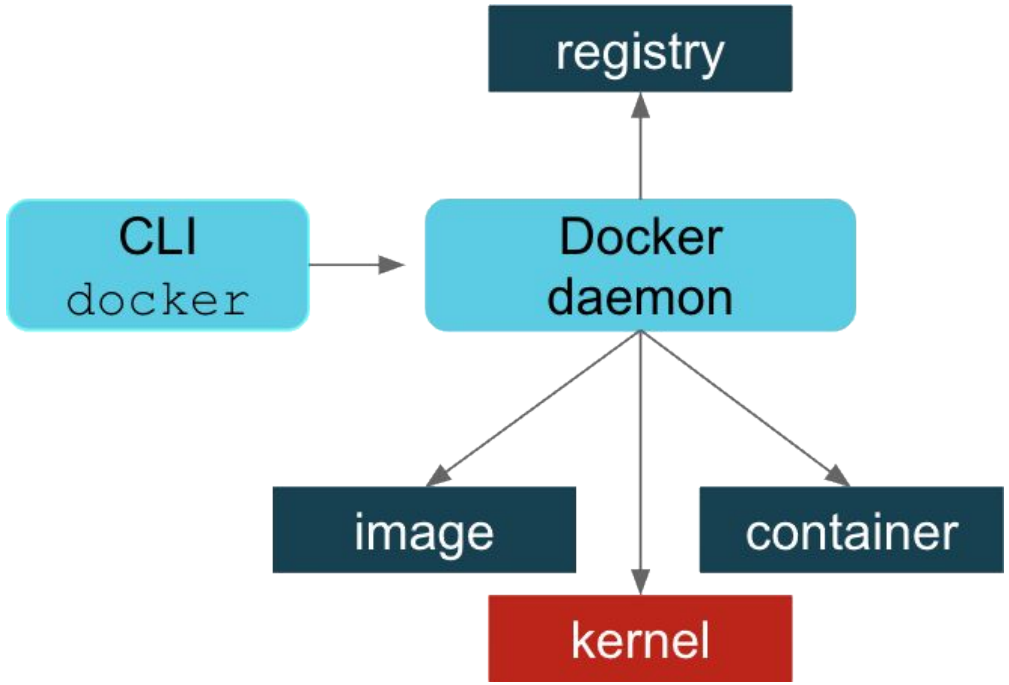
Source:
Modifying the Red Hat Enterprise Linux OpenStack Platform Overcloud Image with virt-customize [https://access.redhat.com/articles/1556833]
Creating Guests with virt-install
[https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/sect-guest_virtual_machine_installation_overview-creating_guests_with_virt_install]

Hacerte invencible significa

conocerte a ti mismo

Sun Tzu

¿Cómo funciona **Docker**?



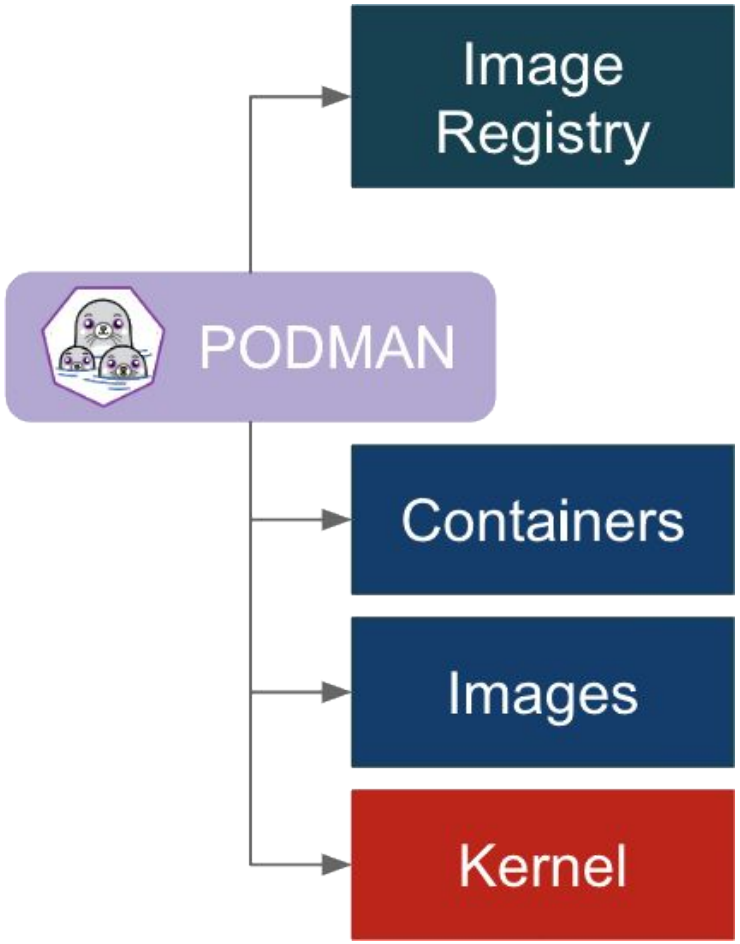
Docker proporciona toda la funcionalidad necesaria para:

- Pull & push 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.

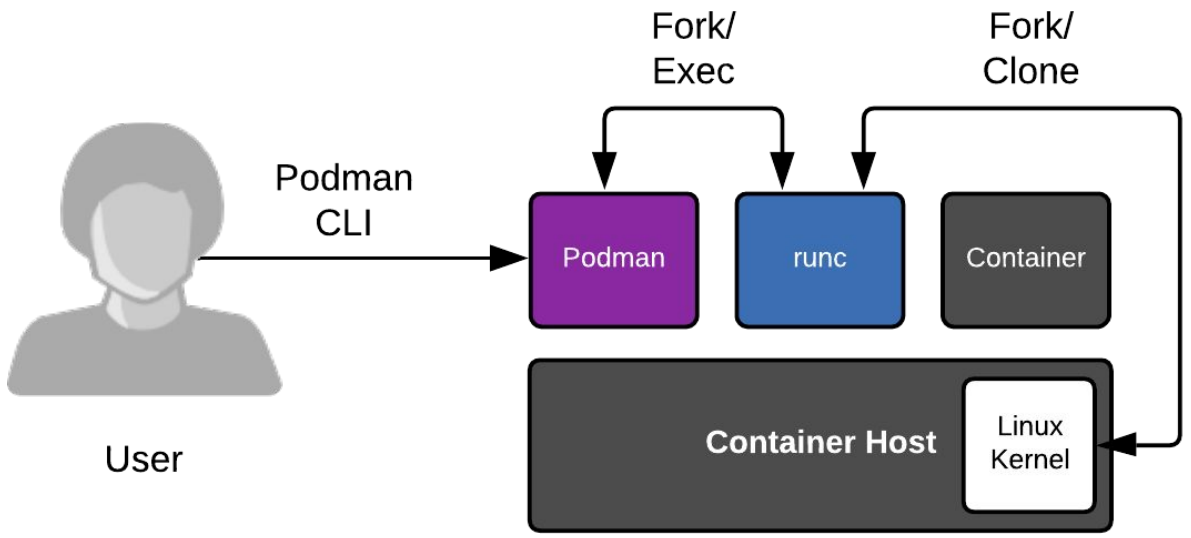
Esencialmente, el demonio **Docker** hace todo el trabajo con registros, imágenes, contenedores y el kernel. La línea de comandos (CLI) de **Docker** le pide al demonio que haga esto en su nombre.

Podman

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



Podman



How containers run with a container engine

Podman

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

Proporciona una sintaxis "tipo **Docker**" para trabajar con 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 -i ipaddr
```
← skopeo

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

```
# curl 10.88.0.2:8080
```

Inmutabilidad

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

```
# podman exec -ti 467ee8ae4f59 /bin/bash
```

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

Podman

Construyendo contenedores (**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, eliminamos la imagen base y ejecutamos el contenedor

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

```
# buildah images
```

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

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

Podman

Construyendo contenedores

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"]
```

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

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

Podman

Construyendo contenedores

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

```
# podman logs 2f41db203183
```

Podman

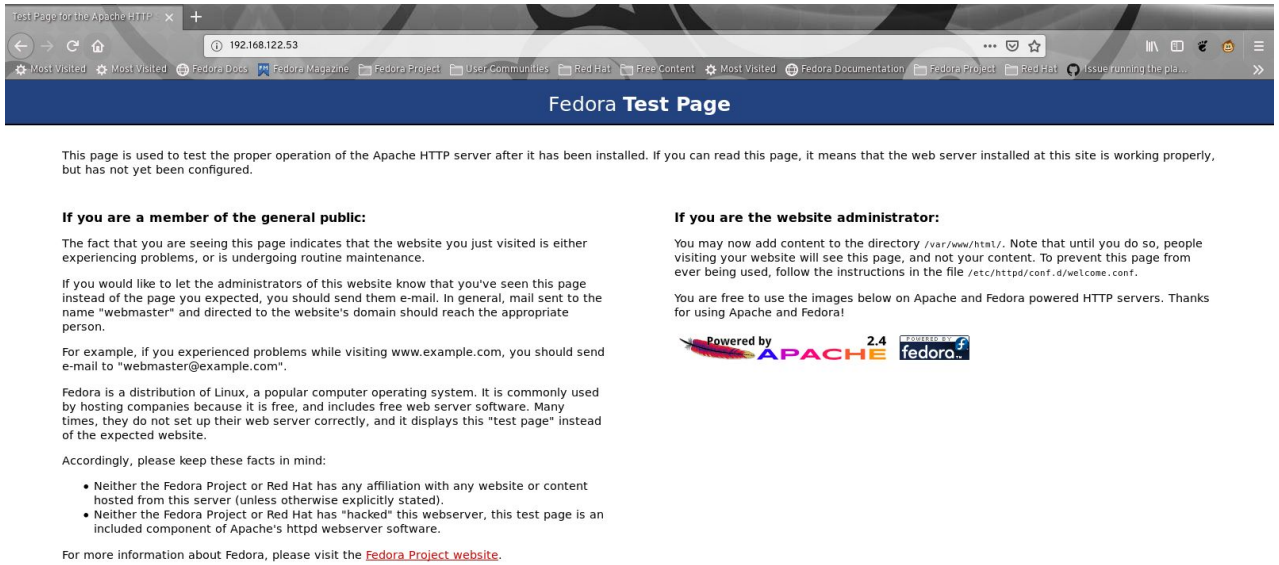
Construyendo contenedores

Detenemos el contenedor y ejecutamos el contenedor exponiendo el puerto

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

Validamos el servicio

```
# curl localhost
```



Podman

Persistent Storage

Creamos 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 puerto y directorio

```
# podman run -d --name myhttpservice -p 8080:8080 -v /opt/var/www/html:/var/www/html:Z httpd-24-rhel7
```

Healthcheck

```
# podman run -dt --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 registry.fedoraproject.org/f29/httpd
```



```
# podman healthcheck run myhttpservice
```

Monitoring container vitality and availability with Podman

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="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 status del servicio

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

Iniciamos el servicio

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

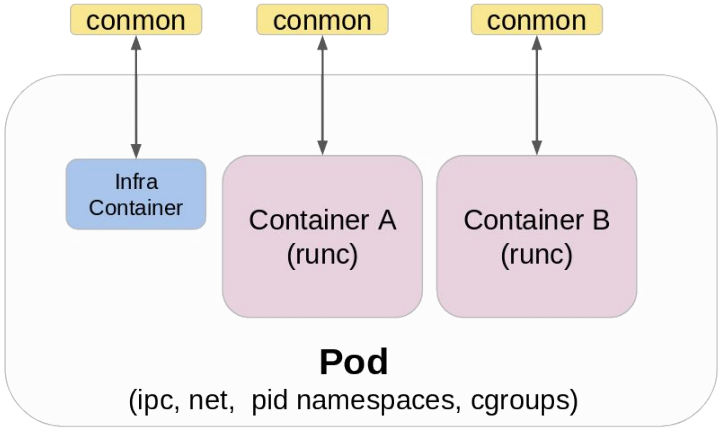
```
# systemctl status myhttpservice.service
```

Podman Pods

Lo que necesitas saber

El concepto de **Pod** fue introducido por Kubernetes. 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
  - 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
```

Listar los pod's

```
# podman pod list
```

Agregar contenedor al pod

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

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

Podman Pods

Ejemplo práctico: MariaDB Container

Creamos 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
```

Agregar contenedor al pod y 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 Pods

¿Siguientes pasos?

Podman Pods

Ejemplo práctico

Creamos un contenedor 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
```

El archivo yml puede recrear el contenedor o 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/

Podman rootless containers

Lo que necesitas saber

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

- Las imágenes se guardan en el *home directory* (`$HOME/.local/share/containers/storage/`) en lugar de `/var/lib/containers`.
- Al ejecutar *rootless containers* se obtiene un permiso especial para ejecutarse como un rango de ID de usuarios y grupos en el host. Sin embargo, no tienen privilegios de root para el sistema operativo del mismo, 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 desde un servicio httpd desde el 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 usuario 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
```

Probar los comandos de podman con un usuario sin privilegios

```
$ podman pull registry.fedoraproject.org/fedora-minimal
$ podman run registry.fedoraproject.org/fedora-minimal cat /etc/redhat-release
```

Checamos la configuración *rootless*

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

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

Comunidades en México

Únete!

- **Fedora México**
  - https://www.meetup.com/es-ES/Fedora-Mexico/
  - https://t.me/fedoramexico
  - https://fedoramx.fedorapeople.org/
- **SysArmy México**
  - https://www.meetup.com/es/Sysarmy-Mexico/
  - https://t.me/sysarmymx

Gracias!

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat