CONTAINERS LAB

Fedora México

Alex Callejas
Senior Technical Support Engineer | Red Hat
Febrero 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!

AGENDA

Containers Lab

Kernel-based Virtual Machine

Hipervisor KVM standalone

podman

Utilidad proporcionada como parte de la biblioteca **libpod**, para crear y mantener contenedores.

buildah

Herramienta que facilita la creación de imágenes de contenedores OCI (Open Container Initiative)

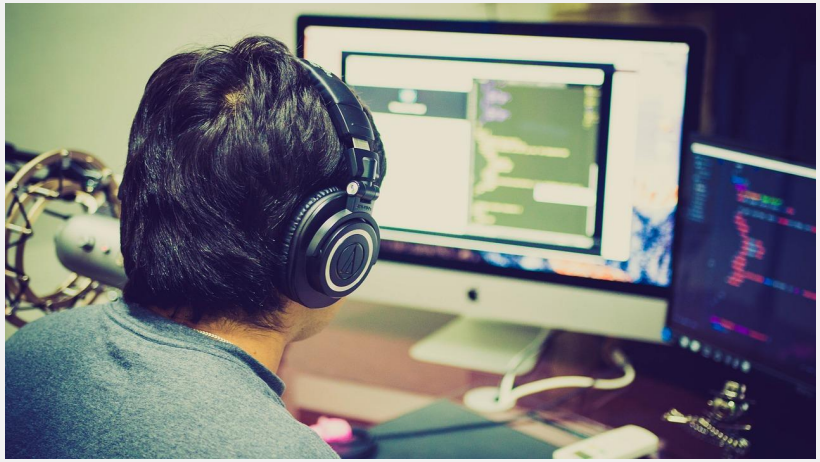
Laboratorio de Pruebas

Laboratorio de Pruebas

Containers Lab

Mi configuración:

- Fedora 28
  - KVM
    - Cloud images



Laboratorio de Pruebas

Containers Lab

Pre-requisitos:

- Procesador Intel con Intel VT-x e Intel 64 virtualization extensions for x86-based systems; ó
- Procesador AMD con AMD-V y AMD64 virtualization extensions

```
# grep -E 'svm|vmx' /proc/cpuinfo
```

Laboratorio de Pruebas

Containers Lab

Paquetes a instalar:

- qemu-kvm
- virt-manager
- virt-viewer
- libguestfs-tools
- virt-install
- genisoimage

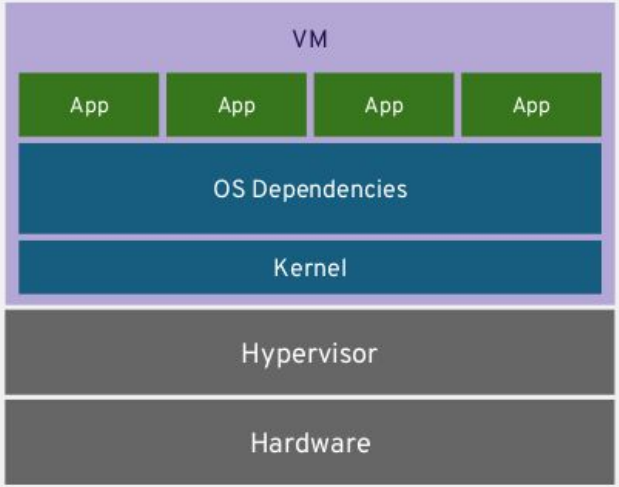
Almacenamiento:

- 5 - 10 Gb
  - `/var/lib/libvirt/images/`

Virtualización y Contenedores

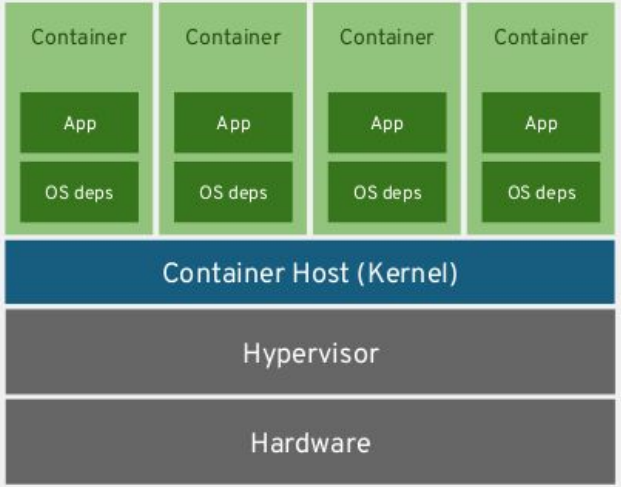
Máquinas Virtuales y Contenedores

Máquinas Virtuales



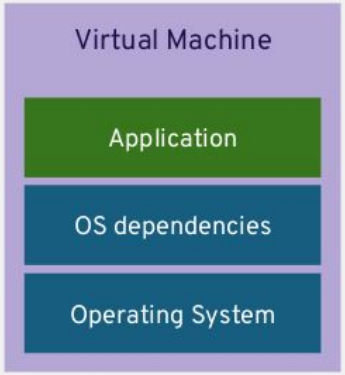
VM aísla el hardware

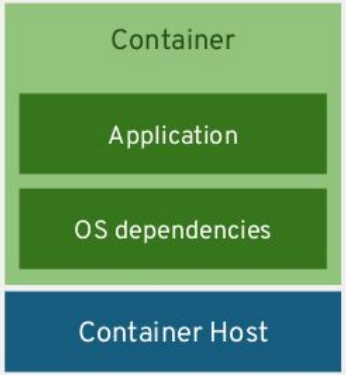
Contenedores



Container aísla el proceso

Máquinas Virtuales y Contenedores

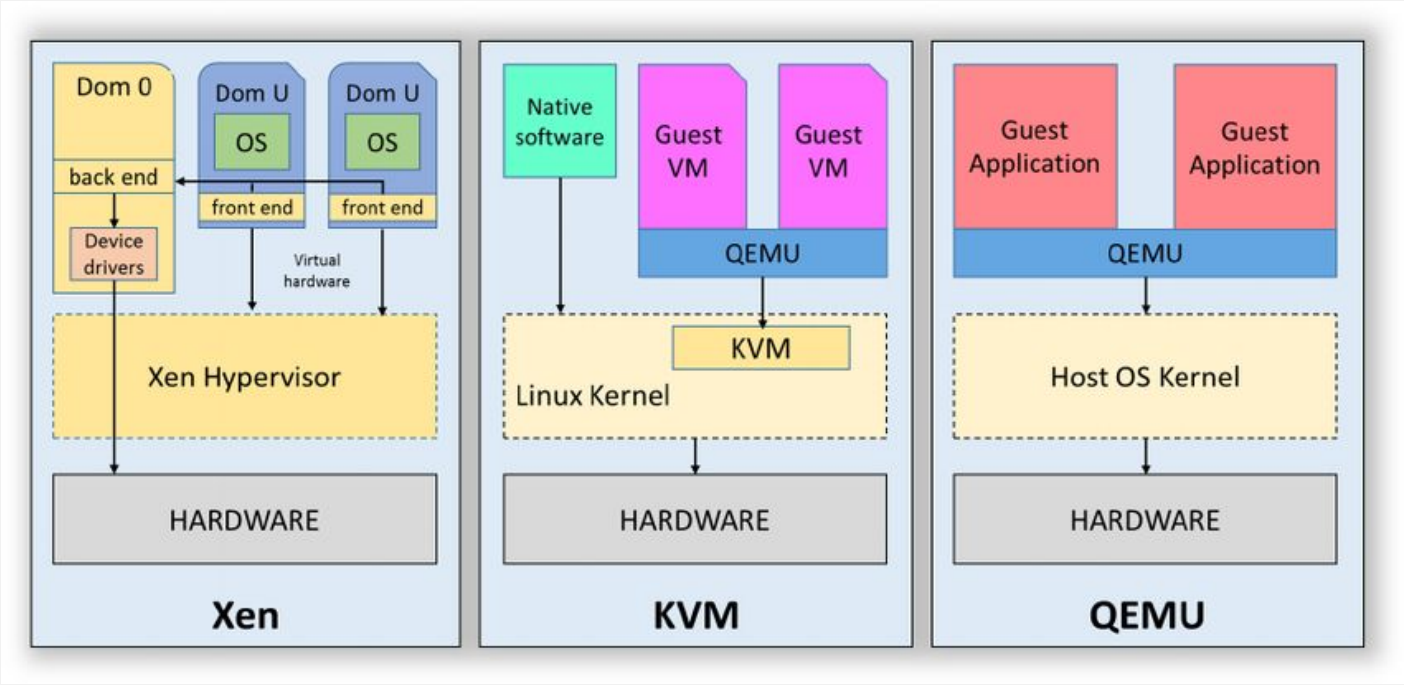




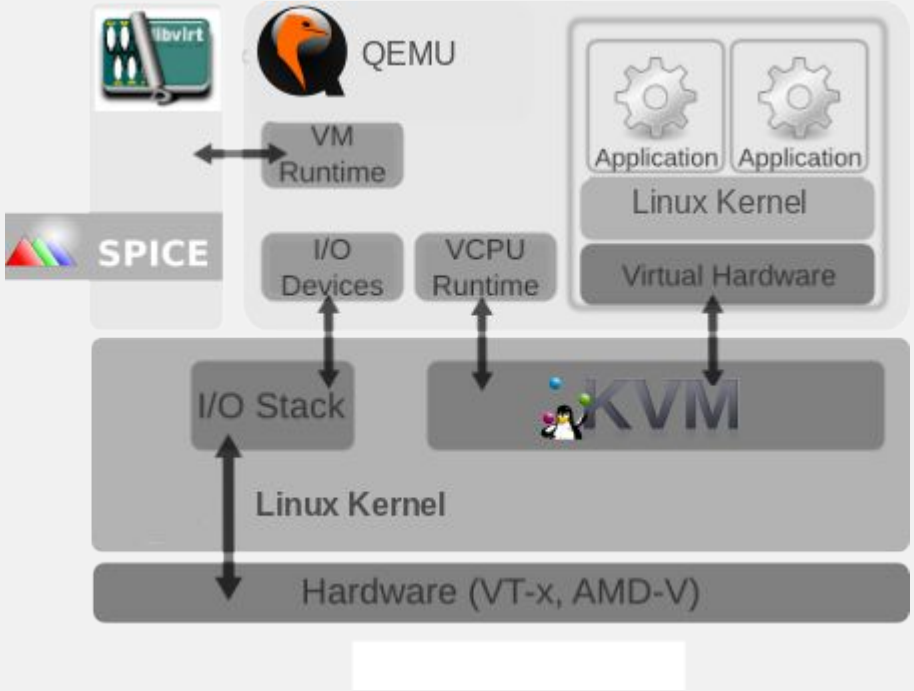
- ✚ Aislamiento de VM
- ➖ OS completo
- ➖ Computo estático
- ➖ Memoria estática
- ➖ Alto uso de recursos

- ✚ Aislamiento de contenedores
- ✚ Kernel compartido
- ✚ Computo escalable
- ✚ Memoria escalable
- ✚ Bajo uso de recursos

Virtualización



Arquitectura KVM



- Management: libvirt (virt tools, C API, bindings, etc)
- qemu-kvm (user-level hardware abstraction layer)
- KVM itself (linux kernel)
- Virtio paravirt drivers (Linux and Windows)
- Spice paravirt drivers (linux and windows

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

Copiar imagen

```
$ sudo cp Fedora-Cloud-Base-29-1.2.x86_64.qcow2 /var/lib/libvirt/images/vmtest01.qcow2
```

Configurar VM

```
$ sudo virt-customize -a /var/lib/libvirt/images/vmtest01.qcow2 \
  --hostname vmtest01.mx.redhat.lab  --root-password password:redhat \
  --ssh-inject 'root:file:labkey.pub' --uninstall cloud-init --selinux-relabel
```

Importar VM

```
$ sudo virt-install --name vmtest01 --memory 1024  \
  --vcpus 1 --disk /var/lib/libvirt/images/vmtest01.qcow2  \
  --import --os-variant fedora29 --noautoconsole
```

Containers Lab

Creando VM's - Fedora

Abrir consola

```
$ sudo virsh console vmtest01
```

Listar / Apagar / Arrancar VM

```
$ sudo virsh list --all

$ sudo virsh [destroy | start] vmtest01
```

Eliminar VM

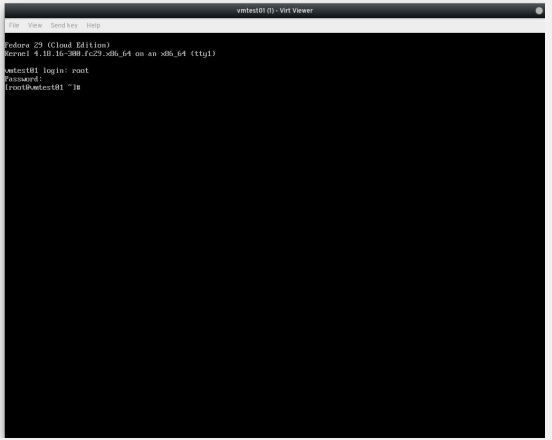
```
$ sudo virsh undefine vmtest01
```

Containers Lab

Creando VM's - Fedora

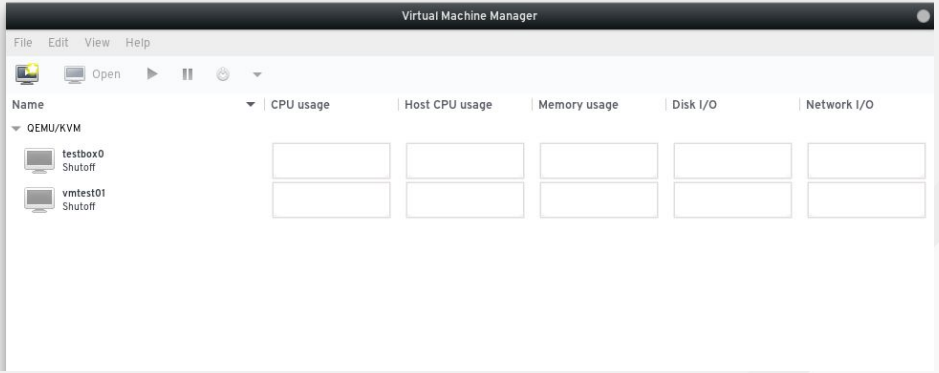
virt-viewer

```
$ virt-viewer --connect qemu:///system vmtest01 &
```



virt-manager

```
$ virt-manager &
```

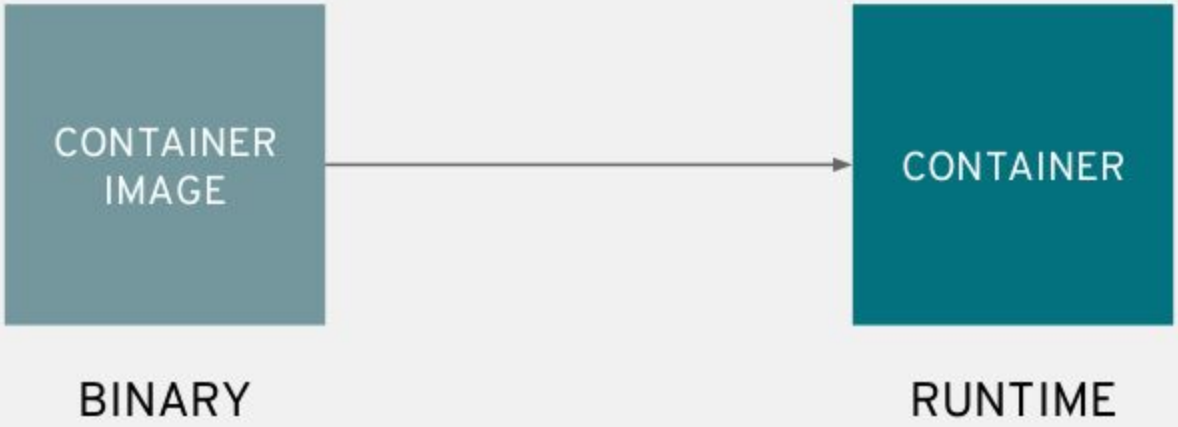


redhat.

Containers

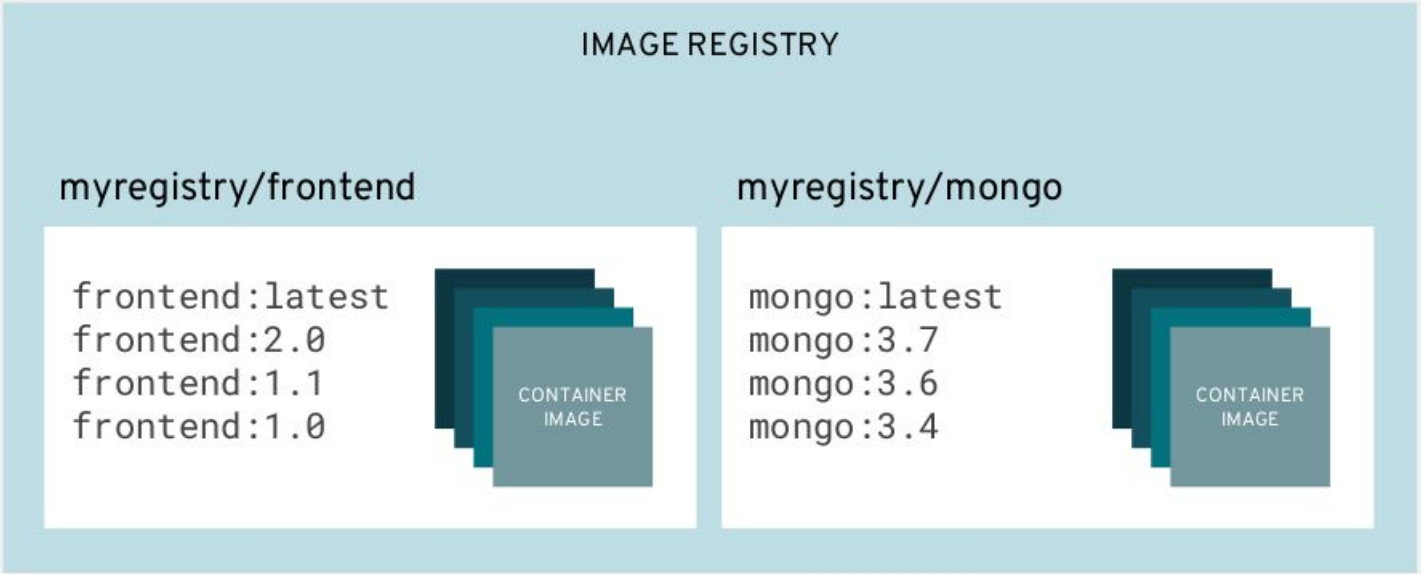
Un contenedor es la unidad de cómputo más pequeña

Los contenedores se crean a partir de imágenes



Containers

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

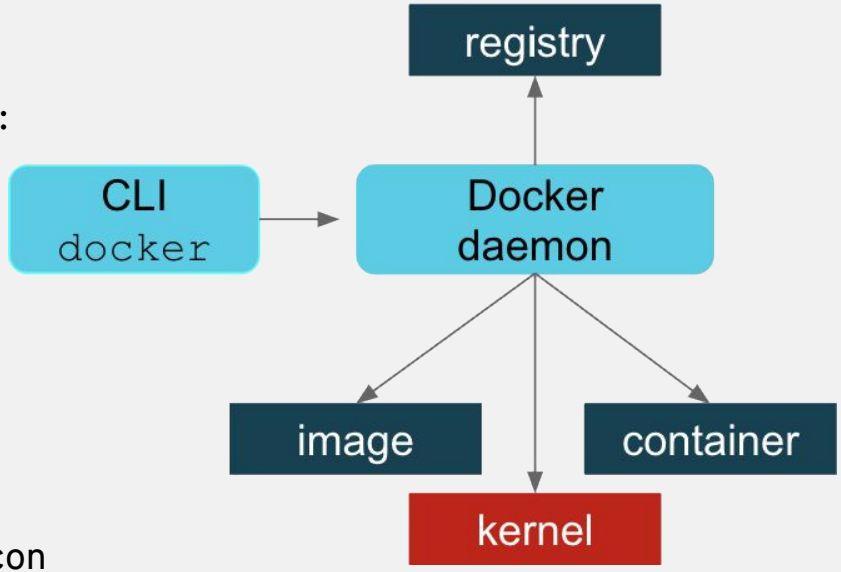


¿Cómo funciona Docker?

Docker proporciona toda la funcionalidad necesaria para:

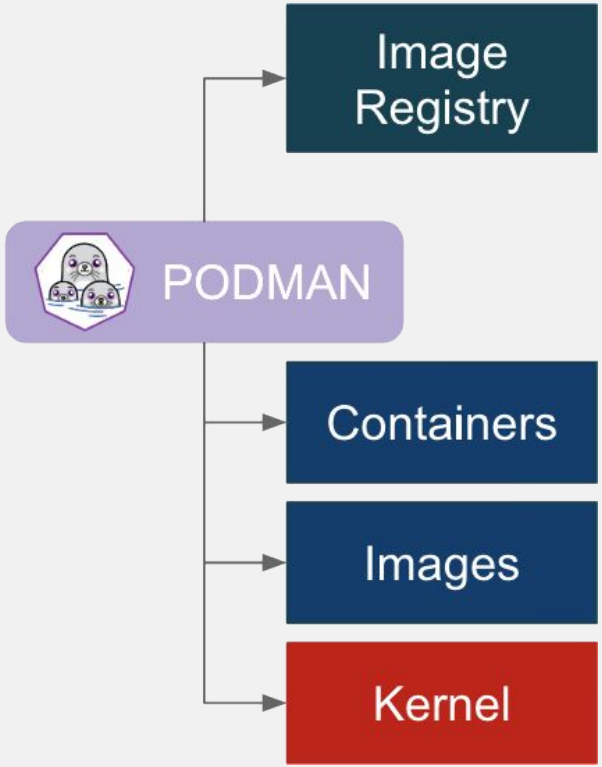
- Pull & push imágenes de un registro de imágenes
- Administrar contenedores locales:
  - Copy, add layers, commit & remove
- Pedir al kernel que ejecute un contenedor con el name-space y cgroup correctos, etc.

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 del contenedor runC (no un demonio)

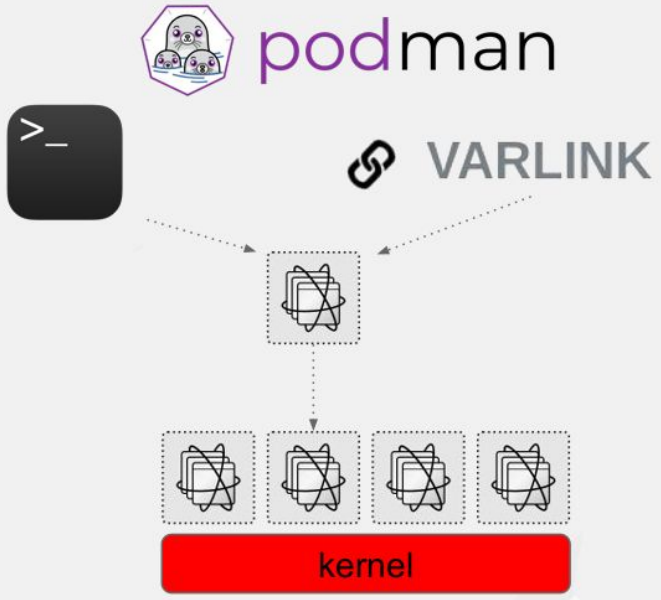


PODMAN

podman está incluido en el último Fedora 29

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

- Rápido y ligero
- runC leverage
- 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

PODMAN

Primeros pasos

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
```

podman

PODMAN

Primeros pasos

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

PODMAN

Primeros pasos

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 10.88.0.3:8080
```

PODMAN

Inmutabilidad

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

PODMAN

Inmutabilidad

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
```

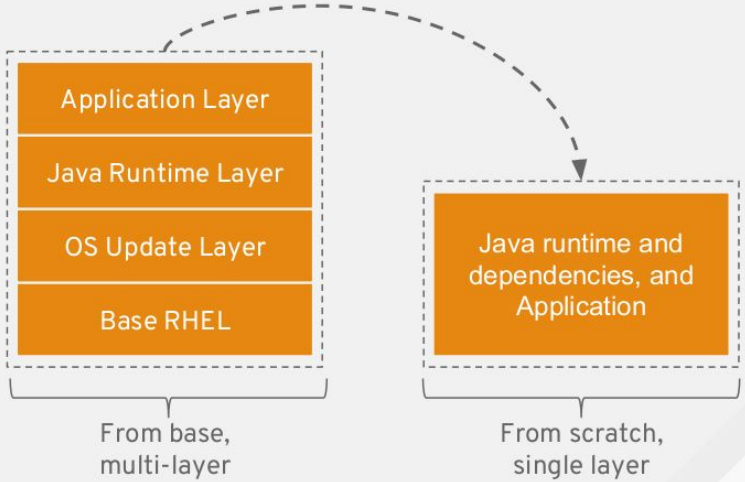
```
bash-4.4$ ls
```

CONSTRUYENDO CONTENEDORES

buildah

¿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



redhat.

buildah

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
```

buildah

CONSTRUYENDO CONTENEDORES

Probando 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
```

buildah

CONSTRUYENDO CONTENEDORES

Dockerfile

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"]
```

buildah

CONSTRUYENDO CONTENEDORES

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
```

buildah

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
```

buildah

CONSTRUYENDO CONTENEDORES

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
```

buildah

CONSTRUYENDO CONTENEDORES

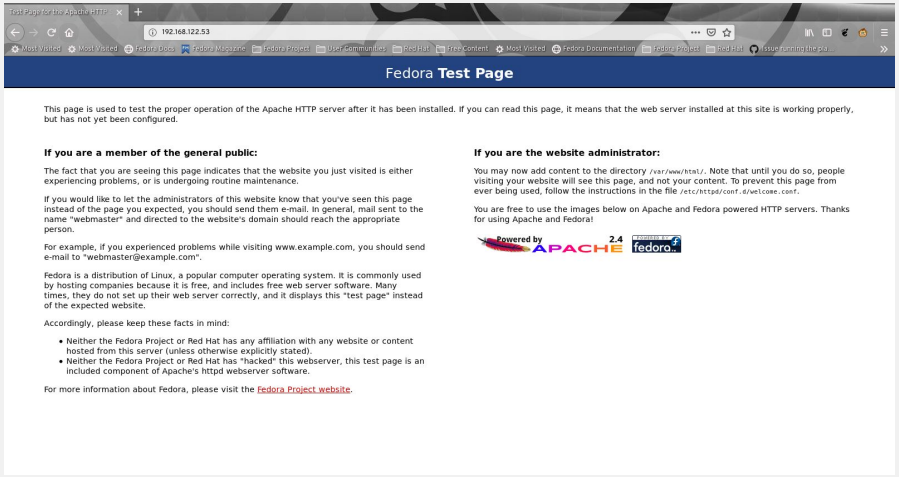
Dockerfile

Detenemos el contenedor y ejecutamos el contenedor exponiendo el puerto

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

Validamos el servicio

```
# curl localhost
```



redhat.

podman

Persistent Storage

Los contenedores son efímeros...

Creamos directorio compartido para el contenedor

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

Creamos el contenido a compartir

```
# wget --page-requisites --convert-links https://registry.fedoraproject.org/
```

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
```

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 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

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
```

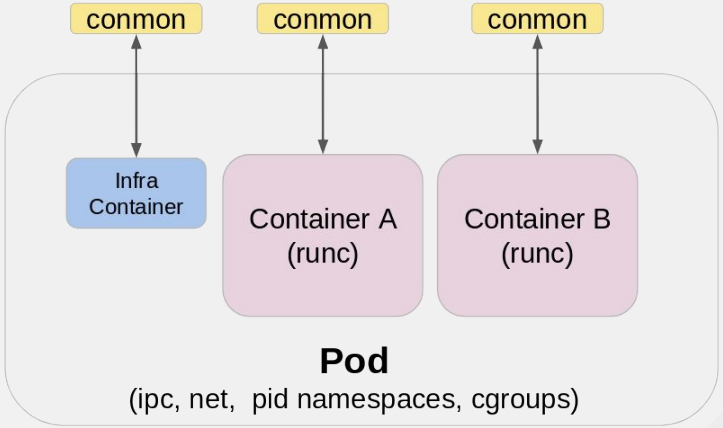
```
# systemctl status myhttpservice.service
```

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

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  container_name docker.io/library/alpine:latest top
```

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

podman

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

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/

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