IMPLEMENTACIÓN DE SERVICIOS CON LA TECNOLOGÍA DE CONTENEDORES PODMAN

MANUAL TÉCNICO

ISBN: 978-958-15-0655-2

SENA

CEAI
Centro de Electricidad y Automatización Industrial Regional Valle

Catalogación en la publicación. SENA Sistema de Bibliotecas

Castillo Ramírez, Mario Germán
Implementación de servicios con la tecnología de contenedores PODMAN / Mario Germán Castillo Ramírez, Luisa Maria Muñoz Valencia. -- Cali : Servicio Nacional de Aprendizaje (SENA). Centro de Electricidad y Automatización Industrial, 2020.

1 recurso en línea (57 páginas : PDF).

Referencias bibliográficas: página 56.
Contenido parcial: Los contenedores -- Diferencias entre una máquina virtual y un contenedor -- Herramienta PODMAN -- Diferencias entre PODMAN y DOCKER -- Instalación de PODMAN en CENTOS -- ¿Que es una imagen? -- Uso de imágenes con HTTPD -- Persistencia de una base de datos MYSQL -- Manipulación de imágenes de contenedores.

ISBN: 978-958-15-0655-2.

1. Sistemas virtuales de computadores 2. Almacenamiento virtual (Computación) 3. Programas para computador I. Muñoz Valencia, Luisa Maria II. Servicio Nacional de Aprendizaje (SENA). Centro de Electricidad y Automatización Industrial.
CDD: 005.43

Manual

IMPLEMENTACIÓN DE SERVICIOS CON LA TECNOLOGÍA DE CONTENEDORES PODMAN

Sistema de Investigación, Desarrollo Tecnológico e Innovación, SENNOVA - SENA

Regional Valle
Centro de Electricidad y Automatización Industrial
Área de Innovación y Competitividad

Calle 52 No 2 Bis 15 Complejo Salomia. Tel 431 5800. Ext. 22581 - 22763
Cali - Colombia

SERVICIO NACIONAL DE APRENDIZAJE, SENA.

Sistema de Investigación, Desarrollo Tecnológico eInnovación, SENNOVA - SENA - Regional Valle.

Centro de Electricidad y Automatización Industrial, CEAI.

Angela Patricia Ibarra Quiroga
Subdirectora

Jose Fernando Perez
Dinamizador SENNOVA

Área de Innovación y Competitividad del CEAI.

Unidad de Investigación Aplicada, Desarrollo Tecnológico e Innovación, UIADTI.

Semillero de Investigación de Electrónica, Instrumentación y Automatización, SIEIA.

Autores:
Mario German Castillo Ramírez
Instructor Investigador

Luisa María Muñoz Valencia
Aprendiz Semillero de Investigación de Teleinformatica - SITI

Editorial:
Servicio Nacional de Aprendizaje, SENA.

ISBN: 978-958-15-0655-2

Primera edición

Diseño y diagramación:
Karen Andrea Saldarriaga Ramírez

Santiago de Cali, marzo de 2020

SENA

CONTENIDO

INTRODUCCIÓN

La complejidad de las aplicaciones va en aumento y la demanda de desarrollos más rápidos es cada vez mayor, debido a esto se debe realizar un mayor esfuerzo por parte de la infraestructura, los equipos de T.I (Tecnologías de la Información) y los procesos. Los contenedores permiten reducir los problemas y lograr interactuar de una manera más rápida en diferentes entornos.

Lo que actualmente llamamos "tecnología de contenedores" ha adquirido una gran relevancia en los últimos años, convirtiéndose en un concepto muy popular dentro del mundo del T.I.
Su objetivo es generar desarrollos con mayor rapidez y satisfacer las necesidades comerciales a media que van surgiendo.

La evolución de este concepto ha estado muy ligada a la evolución del Kernel de Linux, debido a que algunas de sus características han sido grandes habilitadores para el avance en el desarrollo de los contenedores.

En resumen, se puede decir que los contenedores son una nueva tecnología disruptiva y generadores de nuevas formas de desarrollo e implementación de sistemas. Logrando así construir un ecosistema donde aparecen múltiples actores; como tecnologías de orquestación, sistemas operativos dedicados, integradores, herramientas de monitorización y gestión y entornos de computación en la nube.

LOS CONTENEDORES

¿QUÉ ES UN CONTENEDOR?

Un contenedor es un conjunto de uno o más procesos que se ejecutan de forma aislada, dentro de un sistema operativo, al cual se le proporcionan capacidades de ejecución básicas de un runtime. Estas capacidades nos permiten alojar servicios y aplicaciones sobre el contenedor.

Además, se puede definir un contenedor como un paquete ligero, autónomo y ejecutable de una pieza de software, el cual incluye todo lo necesario para ser ejecutado: código, runtime, herramientas y librerías del sistema, así como sus configuraciones. (Arquitecto IT. (s.f.). Recuperado el 20 Noviembre 2020, de http://www.arquitectoit.com/)

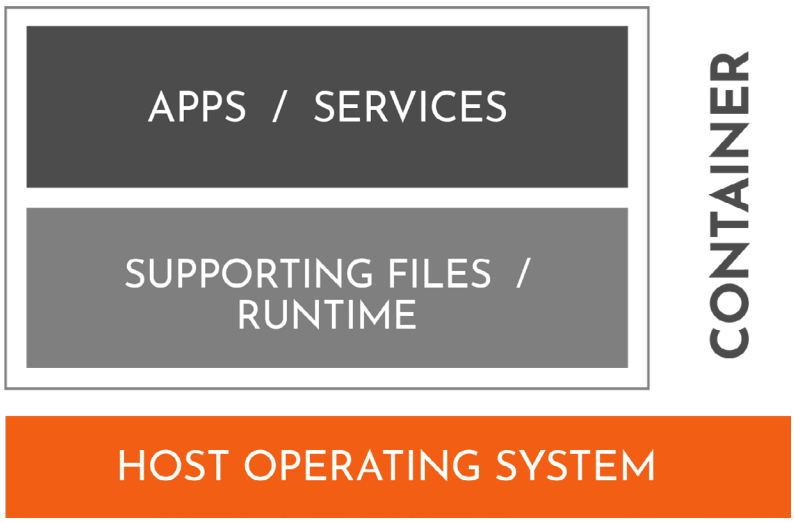

Figura 1. Adaptada de http://www.arquitectoit.com/docker/que-es-un-contenedor/

¿PARA QUÉ SIRVE UN CONTENEDOR?

El uso de contenedores ayuda no solo a la eficiencia, elasticidad y la reutilización de las aplicaciones alojadas, sino también con la portabilidad de las aplicaciones. Los contenedores simplifican el consumo de aplicaciones con todas sus dependencias y archivos de configuración predeterminado.

CARACTERÍSTICAS DE UN CONTENEDOR

Los contenedores se caracterizan por:

- Proporcionar una alta disponibilidad de las aplicaciones.
- Almacenar gran cantidad de información en ellos.
- Los archivos para ejecutarlos provienen de imágenes diferentes.
- Son portables y uniformes en todas sus etapas (desarrollo, prueba y producción).
- Son más rápidos y versátiles.
- Requieren menos recursos informáticos.
- Son importantes en la seguridad de TI debido a su popularidad y facilidad de uso.
- Diseñan aplicaciones nativas de la nube.
- Empaquetar microservicios.

VENTAJAS DE UN CONTENEDOR

El uso de contenedores cuenta con ventajas importantes, las cuales son:

- **Menor tamaño del hardware:** Los contenedores usan las características internas del SO (Sistema Operativo) para crear un entorno aislado, con un enfoque diferente, lo que minimiza la cantidad de CPU y la sobrecarga de la memoria.
- **Aislamiento del entorno:** Los contenedores funcionan en un ambiente cerrado, donde los cambios que se le realicen al SO o aplicaciones del host no afectan el funcionamiento del contenedor. Debido a que las librerías requeridas por un contenedor son autónomas.
- **Implementación rápida:** Los contenedores tienen una implementación rápida, porque no existe la necesidad de instalar el SO subyacente. Al momento de reiniciar un contenedor no se deben detener los servicios del SO del host.
- **Implementación de varios entornos:** En una implementación tradicional, cualquier diferencia entre los entornos puede interrumpir la aplicación. Sin embargo, al usar contenedores, todas las dependencias de la aplicación y las configuraciones del entorno se encuentran encapsuladas en la imagen del contenedor.
- **Reutilización:** El mismo contenedor puede ser reutilizado sin tener que configurar un SO completo. Al usar contenedores ya no es necesario mantener servidores separados para la producción y el desarrollo; se usa una única imagen de contenedor para el servicio.

Además, los contenedores son un enfoque ideal al momento de usar microservicios para el desarrollo de aplicaciones debido a que cada servicio se encapsula en un entorno de contenedor ligero y confiable que puede ser implementado en un entorno de producción o desarrollo.

DIFERENCIAS ENTRE UNA MÁQUINA VIRTUAL Y UN CONTENEDOR

Los contenedores proporcionan beneficios similares a los de las máquinas virtuales, entre los cuales se encuentran la seguridad, el almacenamiento y el aislamiento de la red.
Ambos son entornos informáticos empaquetados que combinan varios elementos de TI y los aíslan del resto del sistema. Las principales diferencias radican en la capacidad de ampliación y la portabilidad.

- Los contenedores se miden en megabytes. El elemento más grande que empaquetan es una aplicación y todos los archivos necesarios para su ejecución. Para que sean ejecutados necesitan de una imagen, la cual incluye todas las bibliotecas y dependencias. La naturaleza ligera de los contenedores y su SO compartido permiten trasladarlos entre los diferentes entornos con mucha facilidad. Requieren muchos menos recursos de hardware y se inician y se finalizan rápidamente.
- Las máquinas virtuales se miden en gigabytes. Necesitan de un software llamado hipervisor, el cual emula el hardware físico para que las máquinas virtuales puedan utilizarlo, este hipervisor consume recursos del sistema. Al iniciar una máquina virtual ella arranca todo el sistema operativo que está virtualizando, es por ello que las imágenes soy muy pesadas (al contener todos los ficheros del sistema operativo).

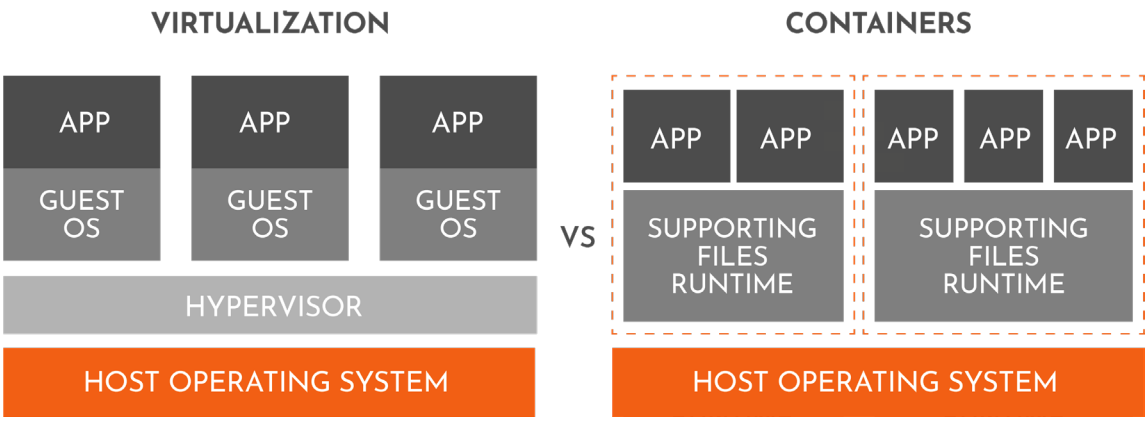

Figura 2. Adaptada de https://www.redhat.com/es/topics/containers/containers-vs-vms

Actualmente existen diferentes tecnologías de contenedores, entre las cuales se encuentran: CoreOS RKT, Mesos Containerizer, LXC, OpenVZ, Docker y Podman.

En este documento nos centraremos en el motor de contenedores Podman desarrollado por Red Hat.

HERRAMIENTA PODMAN

¿QUÉ ES PODMAN?

Podman (Pod Manager) es una herramienta nativa de Linux y la última palabra de moda en tecnología de contenedorización. Fue desarrollada por RedHat como otro motor de contenedores, siendo una solución de código abierto y sin demonios; diseñado para desarrollar, administrar y ejecutar aplicaciones utilizando contenedores e imágenes de contenedor OCI en un sistema Linux.

El enfoque principal de Podman es 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. La idea fue buscar facilidad de uso y unificación de conceptos, para ellos se descentralizaron sus componentes, de forma que cada uno es usado solo cuando es necesario. (Tomado de: Callejas, A|rootzilopochtli.com|. (s.f.) Recuperado el 20 Noviembre 2020, de http://www.rootzilopochtli.com/)

Figura 3. Tomada de https://podman.io/

CARACTERÍSTICAS DE PODMAN

Los contenedores se han popularizado rápidamente en los últimos años. Sin embargo, la tecnología detrás de los contenedores ha existido por un tiempo relativamente largo. En 2001, Linux introdujo un proyecto llamado VServer. VServer fue el primer intento de ejecutar conjuntos completos de procesos dentro de un solo servidor con un alto grado de aislamiento.

Comenzado con VServer, la idea de los procesos aislados evolucionó y se formalizó proporcionando el Kernel de Linux las siguientes características:

- **Espacio de nombres (namespaces):** El kernel puede asegurar el aislamiento de procesos, colocándolos en un espacio de nombres, hay que tener en cuenta que dentro del espacio de nombres solo los procesos que son miembros de este espacio pueden observar estos recursos. También pueden incluir recursos como interfaces de red, lista de ID del proceso, puntos de montaje, recursos de IPC y la información del host del sistema.
- **Grupos de control (cgroups):** Particionan procesos y sus elementos secundarios en grupos, para administrar y limitar los recursos consumidos; colocan restricciones sobre la cantidad de recursos del sistema que pueden utilizar los procesos, lo que impide que un proceso use demasiados recursos del host.
- **Seccomp:** Limitan la manera en la que los procesos realizan llamadas al sistema. También determina un perfil de seguridad para procesos y elabora un listado de permitidos con las llamadas al sistema, los parámetros y los descriptores de archivos que pueden usar.
- **SELinux (Security-Enhanced Linux):** Es un sistema de control de acceso obligatorio para los sistemas. El kernel de Linux isa SELinux para proteger los procesos entre sí y proteger el sistema host de sus procesos en ejecución. Los procesos son ejecutados como un tipo de SELinux confinado, que tiene acceso limitado a los recursos del sistema del host.

Todas las innovaciones y características se centran en un concepto básico y es permitir que los procesos se ejecuten de manera aislada y al mismo tiempo accedan a los recursos del sistema. Esto es la base de la tecnología de contenedores y de su implementación. Actualmente, los contenedores son procesos del kernel de Linux que usan características de seguridad para crear un entorno aislado, lo que permite que los procesos aislados no usen indebidamente el sistema y otros recursos de contenedores.

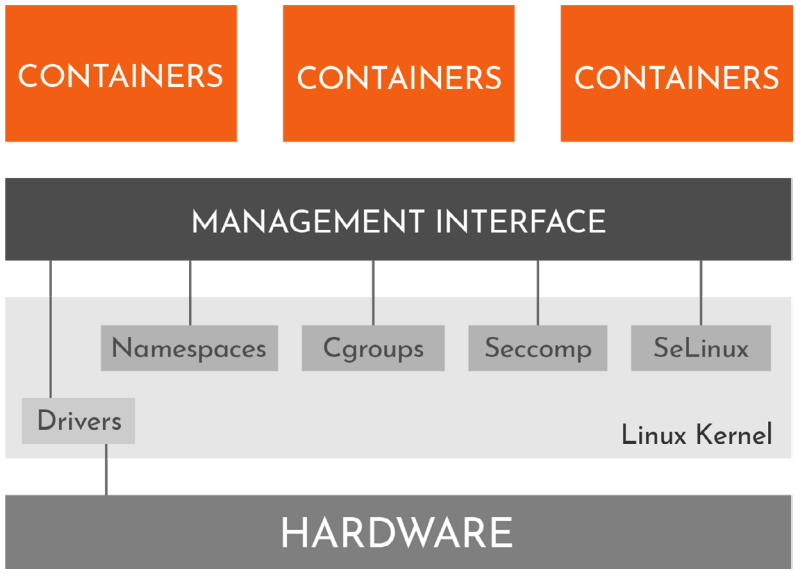

Figura 4. Adaptada de http://www.rootzilopochtli.com/wp-content/uploads/2019/04/Fedora-Containers.pdf

Las características principales de Podman son:

- Los contenedores bajo el control de Podman pueden ser ejecutados como root o por un usuario sin privilegios.
- Rápido y ligero.
- RunC compartido.
- Permite trabajar con Pods y es compatible con Kubernetes.
- No necesita correr un demonio para poder ejecutarlo.
- Es totalmente compatible con Docker.
- Se usa exactamente igual que Docker, pero usando Podman por Docker.
- Proporciona integración de sistemas y aislamiento avanzado de namespace.

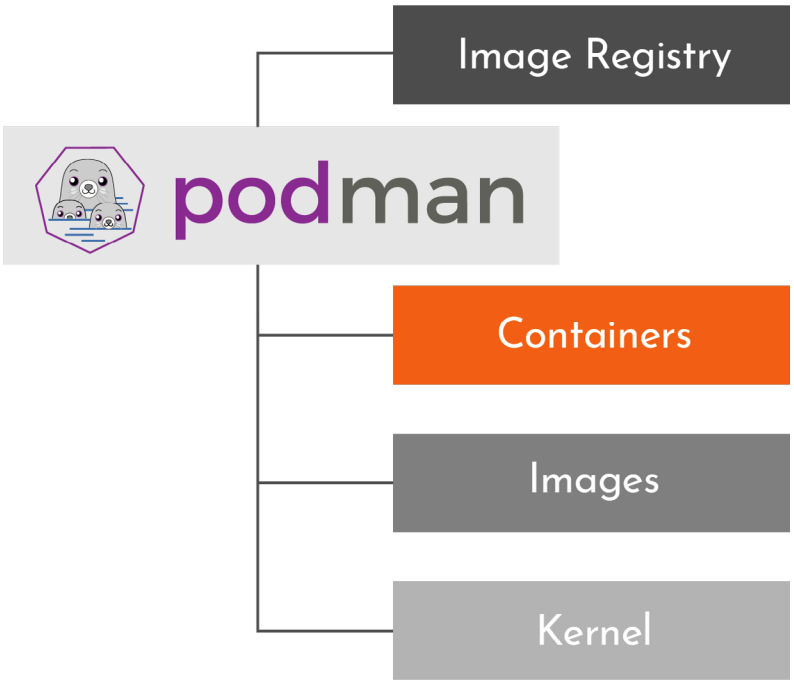

Figura 4.1. Adaptada de http://www.rootzilopochtli.com/wp-content/uploads/2019/04/Fedora-Containers.pdf

DIFERENCIAS ENTRE PODMAN Y DOCKER

La principal diferencia entre Docker y Podman es que Red Hat decidió desarrollar una herramienta sin depender de un servicio, es decir, Podman no necesita de un Daemon para funcionar, debido a que la gestión de contenedores en Podman se ha descentralizado y la han dividido en componentes más pequeños.

Docker utiliza privilegios de root, mientras que Podman puede ejecutar contenedores sin root.

Podman es capaz de ejecutar contenedores exactamente de la misma forma en que lo hace Docker, pero también es capaz de ejecutar Pods. Cuando se habla de Pods se refiere a la unidad de medida mínima en Kubernetes, sin embargo, los Pods pueden alojar más de un contenedor.

INSTALACIÓN DE PODMAN EN CENTOS 8

Lo primero que se debe realizar es actualizar los paquetes del sistema antes de instalar paquetes nuevos, para que después no tengamos inconvenientes con ningún paquete extra.

```
[student@localhost ~]$ sudo dnf update
```

Figura 5

Se instala Podman con el siguiente comando.

```
[student@localhost ~]$ sudo dnf install podman
```

Figura 6

Si la instalación fue exitosa, verificamos la versión de Podman.

```
[student@localhost ~]$ podman --version
podman version 1.6.4
```

Figura 7

Para observar los contenedores que se encuentra corriendo, utilizamos el siguiente comando.

```
[student@localhost ~]$ podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
```

Figura 8

¿QUÉ ES UNA IMAGEN?

Desde la perspectiva del kernel de Linux, un contenedor en un proceso con restricciones. Sin embargo, en lugar de ejecutar un solo archivo binario, ejecuta una **imagen.**

Una imagen es un paquete de sistema de archivos que contiene todas las dependencias necesarias para ejecutar un proceso: archivos del sistema de archivos, paquetes instalados, recursos disponibles, procesos en ejecución y módulos del kernel, lo que facilita la ejecución de la aplicación en diferentes entornos.

Así como los archivos ejecutables son la base para ejecutar procesos, las imágenes son la base para ejecutar contenedores. Como las imágenes son archivos, se pueden administrar mediante sistemas de control de versiones, lo que mejora la automatización del contenedor y el aprovisionamiento de imágenes.

Las imágenes de contenedores deben encontrarse disponibles a nivel local para que el contenedor las ejecute en tiempo real, las imágenes se almacenan y se mantienen en un repositorio de imágenes.

Un repositorio de imágenes es solo un servicio (público o privado) donde las imágenes se pueden almacenar, buscar y recuperar.

Desde Red Hat Enterprise Linux 7 GA, se ofrecen **imágenes de Red Hat Enterprise Linux (RHEL)**, brindando a los clientes contenedores certificados y actualizados de nivel empresarial. La ejecución de imágenes de contenedor RHEL ofrece compatibilidad y portabilidad entre entornos, pero hubo un problema, no puede compartirlo con otros, sin importar si es cliente o socio de Red Hat Enterprise Linux.

Todo esto cambió gracias al lanzamiento de **Red Hat Universal Base Image (UBI)**, ahora se puede aprovechar la mayor confiabilidad, seguridad y rendimiento de las imágenes oficiales de contenedores de Red Hat donde se ejecutan contenedores de Linux compatibles con OCI, ya sea un cliente o no. La imagen de base universal de Red Hat (UBI) permite crear, compartir y ejecutar las aplicaciones en cualquier lugar.

UBI es tres cosas:

- Un conjunto de tres imágenes bases (ubi, ubi-minimal, ubit-init).
- Un conjunto de imágenes en tiempo de ejecución de idiomas (nodejs, ruby, python, php, perl, etc.)
- Un conjunto de paquetes asociados en un repositorio YUM que satisfacen dependencias de aplicaciones comunes.

Red Hat proporciona varias imágenes base que se pueden utilizar como punto de partida para sus propias imágenes. Todas las imágenes de RHEL 8 son imágenes de UBI, lo cual significa que se pueden obtener y redistribuir fácilmente.

Debemos tener en cuenta que dependiendo de la versión de RHEL se le asigna un número a la imagen de base universal de Red Hat (UBI), (RHEL7 → UBI7, RHEL8 → UBI8).

BUSCAR LAS IMÁGENES

Existen muchos repositorios de imágenes diferentes disponibles, cada uno con características diferentes:

- Red Hat Container Catalog [https://registry.redhat.io]
- Docker Hub [https://hub.docker.com]
- Red Hat Quay [https://quay.io/]
- Google Container Registry [https://cloud.google.com/container-registry/]
- Amazon Elastic Container Registry [https://aws.amazon.com/ecr/]

En este manual, se usa el registro de imágenes publicas Quay.

Lo primero que realizamos es la creación de una cuenta en GitHub, lugar donde se almacenan repositorios y permite el control de cambios sobre los contenedores que se están desarrollando.

Se ingresa a la página web https://github.com/login y damos la opción, crear una cuenta.

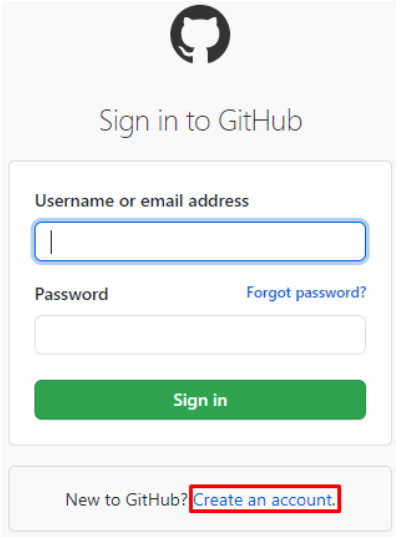

Figura 9

Después de haber creado la cuenta, ingresamos con nuestras credenciales.

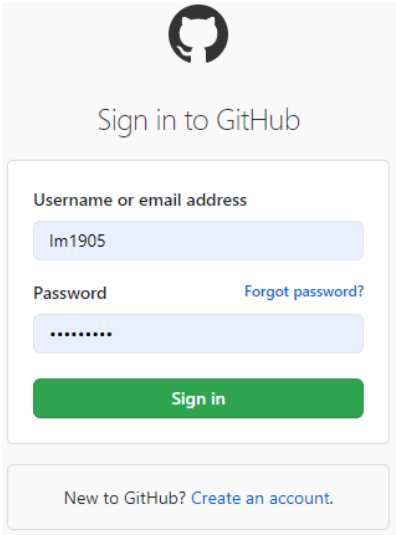

Figura 10

Una vez creada nuestra cuenta en GitHub, nos dirigimos a la página de Quay.io [https://quay.io/] y damos la opción de iniciar sesión.

Podemos observar que nos brinda varias opciones a la hora de iniciar sesión, en este caso iniciamos sesión con GitHub que fue el lugar donde creamos nuestra cuenta anteriormente.

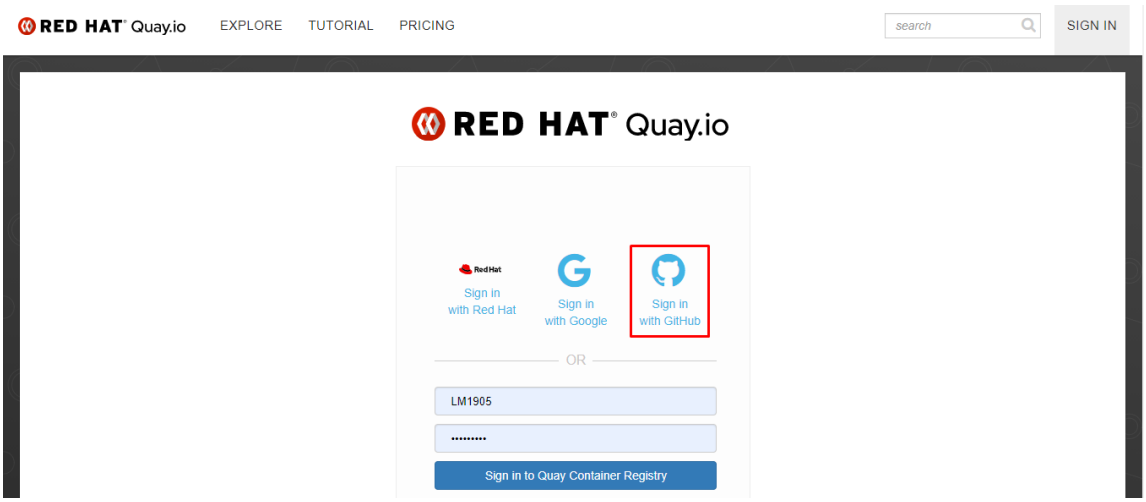

Figura 11

Después de que ingresamos con nuestra cuenta de GitHub, realizamos la creación de un nuevo repositorio, el cual nos será útil para almacenar las imágenes de los contenedores y también permite diseñar, distribuir e implementar contenedores.

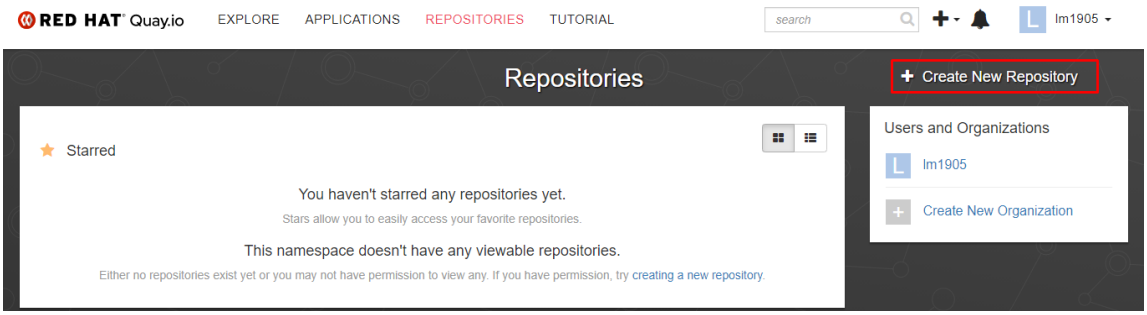

Figura 12

Antes que nada, se recomienda realizar un minitutorial que se encuentra en la página de quay.io y lo podemos observar en la parte superior, en el cual se enseña un paso a paso para la creación de un repositorio.

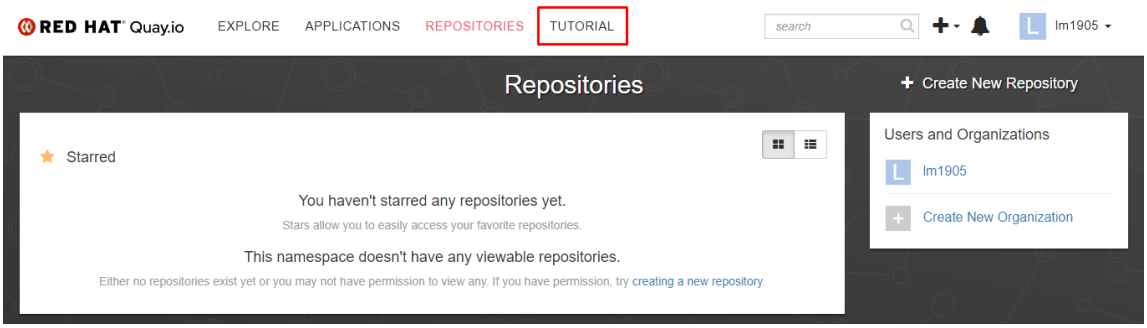

Figura 13

Una vez que damos clic en la pestaña de **tutorial**, se podrá visualizar la bienvenida que nos da a él y se puede realizar de manera opcional, para aprender sobre la creación de un repositorio.

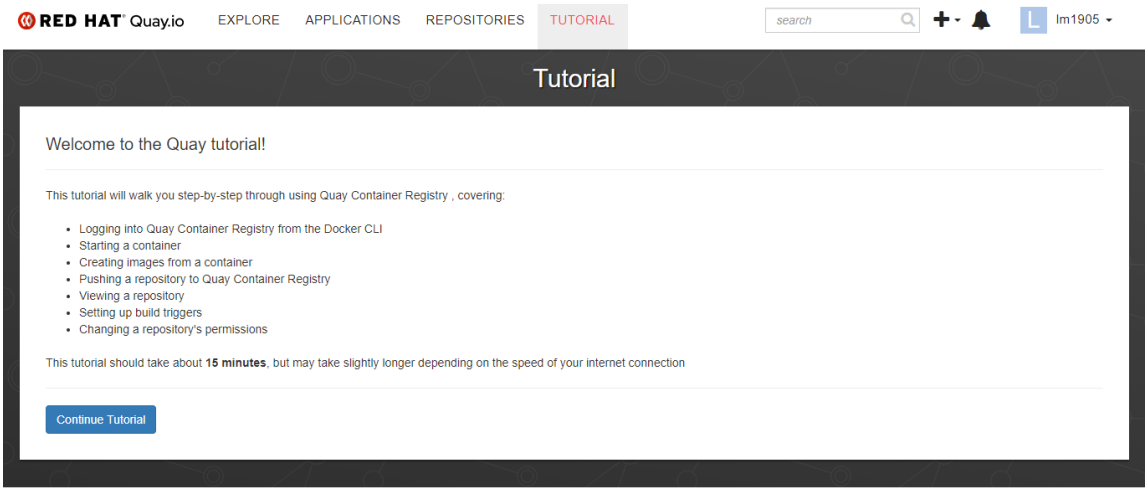

Figura 14

Siguiendo el paso a paso del tutorial, podemos observar que al finalizarlo se creó nuestro repositorio.

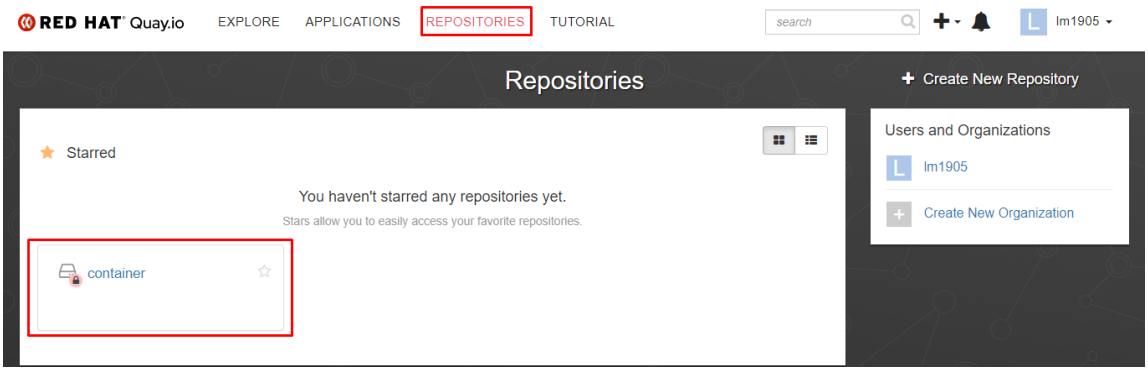

Figura 15

De igual manera cuando ingresamos a nuestro repositorio, podemos observar que nos brinda varias opciones para poder personalizarlo.

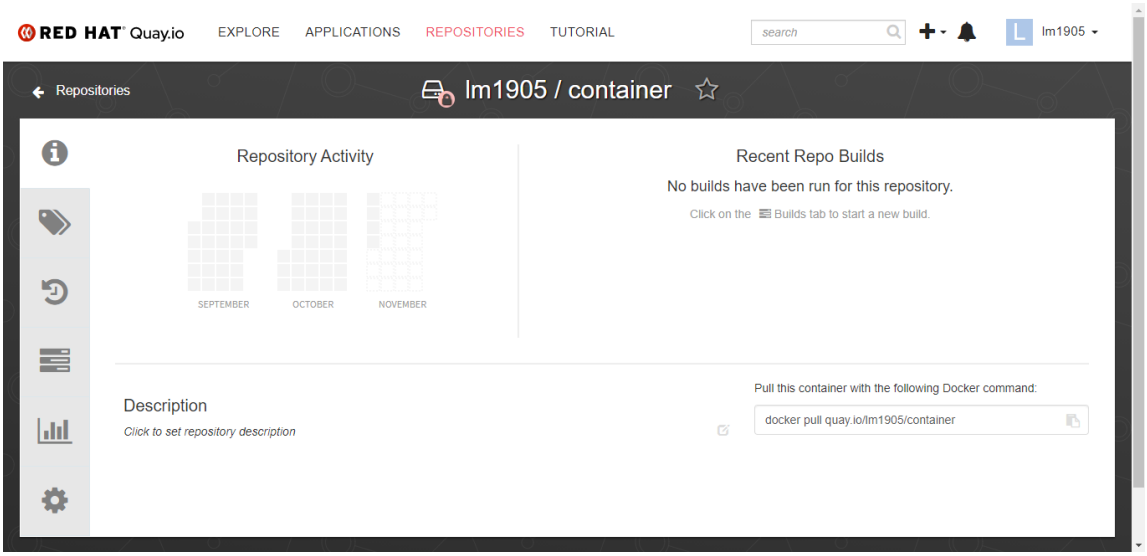
RED HAT Quay.io
EXPLORE
APPLICATIONS
REPOSITORIES
TUTORIAL
search
lm1905
Repositories
lm1905 / container
Repository Activity
SEPTEMBER
OCTOBER
NOVEMBER
Recent Repo Builds
No builds have been run for this repository.
Click on the Builds tab to start a new build.
Description
Click to set repository description
Pull this container with the following Docker command:
docker pull quay.io/lm1905/container

Figura 16

USO DE IMÁGENES CON HTTPD

Lo primero que realizamos es ingresar a la base de datos de Quay.io.

```
[student@localhost ~]$ podman login quay.io
Username: lm1905
Password:
Login Succeeded!
```

Figura 17

Buscamos la imagen que utilizaremos con el subcomando **search** de la siguiente manera y elegimos la imagen con la cual se trabajará. En nuestro caso elegimos la versión **httpd-24-rhel7**.

```
[student@localhost ~]$ sudo podman search httpd-24-rhel7
INDEX       NAME                                              DESCRIPTION               STARS   OFFICIAL   AUTOMATED
redhat.com  registry.access.redhat.com/rhscl/httpd-24-rhel7   Apache HTTP 2.4 Server    0
redhat.io   registry.redhat.io/rhscl/httpd-24-rhel7           Apache HTTP 2.4 Server    0
docker.io   docker.io/guanyebo/httpd-24-rhel7                                           1
docker.io   docker.io/banders2/httpd-24-rhel7-extra                                     0
```

Figura 18

El comando **podman run** ejecuta un contenedor localmente basado en una imagen. Como mínimo, el comando requiere que el nombre de la imagen se ejecute en el contenedor.

Cuando hayamos seleccionado la imagen, lo que hacemos es iniciar la imagen en segundo plano, para ello agregamos la opción **-d** al comando **podman run**.

> **NOTA:** **-d** o en su forma larga **--detach**, significa que el contenedor se ejecuta en segundo plano (independiente).

```
[student@localhost ~]$ sudo podman run -d rhscl/httpd-24-rhel7:2.4-36.8
Trying to pull registry.access.redhat.com/rhscl/httpd-24-rhel7:2.4-36.8...
Getting image source signatures
Copying blob c5d2e9481169 done
Copying blob 258c04e7cfdc done
Copying blob e373541ccf6a done
Copying blob 202f3871a68d done
Writing manifest to image destination
Storing signatures
6fcd6139a2290317e8cf7c438a3014ab44339679955112a1893190c01fa67832
```

Figura 19

Con el ejemplo anterior lo que realizamos fue ejecutar un servidor HTTP Apache contenerizado en segundo plano.

> **NOTA:** Si la imagen que se ejecutará no está disponible localmente, al usar el comando podman run, Podman usa automáticamente pull para descargar la imagen.

Por medio del comando podman images apreciamos las imágenes que descargamos, en la siguiente imagen podemos observar que en nuestro espacio local se encuentra la imagen de HTTP, que acabamos de descargar.

```
[student@localhost ~]$ sudo podman images
REPOSITORY                                  TAG        IMAGE ID       CREATED       SIZE
registry.access.redhat.com/rhscl/httpd-24-rhel7   2.4-36.8   5f9f898a3093   3 years ago   309 MB
```

Figura 20

Con el comando podman inspect -l -f se muestra la dirección IP interna del contenedor, encontrada en los metadatos del contenedor.

```
[student@localhost ~]$ sudo podman inspect -l -f "{{.NetworkSettings.IPAddress}}"
10.88.0.3
```

Figura 21

NOTA: -l (l para el último) como reemplazo para el identificador del contenedor. Este indicador aplica el comando al último contenedor usado en cualquier comando de Podman.

El curl nos ayuda a verificar la conectividad que tenemos con la página web, junto a él debemos colocar la IP obtenida anteriormente y el puerto de conexión.

```
[student@localhost ~]$ curl http://10.88.0.5:8080
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
        <head>
                <title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
                <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
                <style type="text/css">
                        /*<![CDATA[*/
                        body {
                                background-color: #fff;
                                color: #000;
                                font-size: 0.9em;
                                font-family: sans-serif,helvetica;
                                margin: 0;
                                padding: 0;
                        }
                        :link {
                                color: #c00;
                        }
                        :visited {
                                color: #c00;
                        }
                        a:hover {
                                color: #f50;
                        }
                        h1 {
                                text-align: center;
                                margin: 0;
                                padding: 0.6em 2em 0.4em;
                                background-color: #900;
                                color: #fff;
                                font-weight: normal;
                                font-size: 1.75em;
                                border-bottom: 2px solid #000;
                        }
                        h1 strong {
                                font-weight: bold;
```

Figura 22

Con el siguiente comando se inicia una terminal de Bash dentro del contenedor y se ejecutan interactivamente comandos en él.

```
[student@localhost ~]$ sudo podman run -it ubi7/ubi:7.7 /bin/bash
Trying to pull registry.access.redhat.com/ubi7/ubi:7.7...
Getting image source signatures
Copying blob 09dbbf8834d2 done
Copying blob fcd63ccfdd0c done
Copying config 0355cd652b done
Writing manifest to image destination
Storing signatures
[root@3bcd6a84f231 /]#
```

Figura 23

```
[root@3bcd6a84f231 /]# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@3bcd6a84f231 /]# whoami
root
[root@3bcd6a84f231 /]# exit
exit
[student@localhost ~]$
```

Figura 24

Si las imágenes deben interactuar con el usuario con la entrada de la consola, Podman puede redirigir las entradas y salidas del contenedor a la consola. El subcomando **run** requiere los indicadores **-t** y **-i** (en definitiva ¬-it) para habilitar la interactividad.

También podemos modificar el archivo **index.html** de la página web.

```
[student@localhost ~]$ sudo podman exec -it funny_hoover /bin/bash
[sudo] password for student:
bash-4.2$ echo "Hello World" > /var/www/html/index.html
bash-4.2$ exit
exit
```

Figura 25

Y por último se realiza la prueba.

```
[student@localhost ~]$ curl http://10.88.0.24:8080
Hello World
```

Figura 26

Crear e iniciar un contenedor es solo lo básico del ciclo de vida del contenedor, aunque debemos tener en cuenta que el ciclo de vida de un contenedor también incluye detenerlo, reiniciarlo o eliminarlo. Los usuarios también pueden observar el estado y los metadatos del contenedor para fines de depuración, actualización o informes.

- El comando **podman run** crea un nuevo contenedor a partir de una imagen alojada localmente e inicia un proceso dentro del contenedor. Si la imagen no se encuentra disponible de manera local, este comando intenta descargarla usando el repositorio de imágenes configurado.

```
[student@localhost ~]$ sudo podman run rhscl/httpd-24-rhel7
Trying to pull registry.access.redhat.com/rhscl/httpd-24-rhel7...
Getting image source signatures
Copying blob 2bd25ca12457 done
Copying blob 5d011ac93e74 done
Copying blob 18a6e61d9d3e done
Copying blob 1323a241cc06 done
Copying config 610c5665a9 done
Writing manifest to image destination
Storing signatures
=> sourcing 10-set-mpm.sh ...
=> sourcing 20-copy-config.sh ...
=> sourcing 40-ssl-certs.sh ...
---> Generating SSL key pair for httpd...
```

Figura 27

- Si se desea definir el nombre del contenedor, se debe usar la opción - **-name** al ejecutar un contenedor.

```
[student@localhost ~]$ sudo podman run --name my-httpd-container rhscl/httpd-24-rhel7
[sudo] password for student:
=> sourcing 10-set-mpm.sh ...
=> sourcing 20-copy-config.sh ...
=> sourcing 40-ssl-certs.sh ...
---> Generating SSL key pair for httpd...
```

Figura 28

```
[student@localhost ~]$ sudo podman ps --format "{{.ID}} {{.Image}} {{.Names}}"
a6b9cd049fe3 registry.access.redhat.com/rhscl/httpd-24-rhel7:latest my-httpd-container
```

Figura 29

NOTA: El nombre debe ser único, sino Podman arroja un error si el nombre ya esta en uso, incluyendo los contenedores detenidos.

- El comando podman exec inicia un proceso adicional dentro de un contenedor que ya esta en ejecución, en nuestro caso observaremos el ID del contenedor.

```
[student@localhost ~]$ sudo podman exec my-httpd-container cat /etc/hostname
cb14e07541dd[student@localhost ~]$
```

Figura 30

- El comando podman ps muestra los contenedores que se están ejecutando.

```
[student@localhost ~]$ sudo podman ps
CONTAINER ID  IMAGE                                                COMMAND               CREATED        STATUS            PORTS  NAMES
cb14e07541dd  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  8 minutes ago  Up 8 minutes ago         my-httpd-container
```

Figura 31

- **Container ID:** Cada contenedor al crearse, obtiene un ID de Contenedor, que es un numero hexadecimal, diferente y único.
- **Image:** La imagen utilizada para iniciar el contenedor.
- **Command:** El comando que se ejecutó cuando se inició el contenedor.
- **Created:** La fecha y la hora en que se inició.
- **Status:** Tiempo de actividad total de contenedor, si esta en ejecución o si se finalizó.
- **Ports:** Los puertos que expuso el contenedor o el reenvío de puertos si se configuraron.
- **Names:** Nombre del contenedor.

Podman no descarta los contenedores detenidos de inmediato, los conserva para facilitar el análisis post mortem. La opción -a enumera todos los contenedores, incluyendo los detenidos.

```
[student@localhost ~]$ sudo podman ps -a
CONTAINER ID  IMAGE                                                COMMAND               CREATED         STATUS                     PORTS  NAMES
cb14e07541dd  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  51 minutes ago  Up 51 minutes ago                 my-httpd-container
ab53a5799cbd  registry.access.redhat.com/rhscl/httpd-24-rhel7:latest  /usr/bin/run-http...  54 minutes ago  Exited (0) 51 minutes ago         beautiful_hoover
dc9bc97c0f35  docker.io/library/busybox:latest                     echo fun              19 hours ago    Exited (0) 19 hours ago           cranky_pike
3bcd6a84f231  registry.access.redhat.com/ubi7/ubi:7.7              /bin/bash             13 days ago     Exited (0) 13 days ago            hopeful_kapitsa
1a6c1f8a5c21  registry.access.redhat.com/rhscl/httpd-24-rhel7:2.4-36.8  /usr/bin/run-http...  13 days ago     Exited (0) 13 days ago            funny_hoover
```

Figura 32

- El comando podman inspect muestra los metadatos acerca de un contenedor detenido o en ejecución. El comando produce una salida JSON (JavaScript Object Notation).

```
[student@localhost ~]$ sudo podman inspect my-httpd-container
[
    {
        "Id": "cb14e07541dd8b9a225b4a72da7c051c175a20b0f3240678e77b13e42cf4af1c",
        "Created": "2020-11-05T17:13:55.049592303-05:00",
        "Path": "container-entrypoint",
        "Args": [
            "/usr/bin/run-httpd"
        ],
        "State": {
            "OciVersion": "1.0.1-dev",
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 3812,
            "ConmonPid": 3800,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2020-11-05T17:13:59.859500806-05:00",
            "FinishedAt": "0001-01-01T00:00:00Z",
            "Healthcheck": {
                "Status": "",
                "FailingStreak": 0,
                "Log": null
            }
```

Figura 33

Este comando nos brinda muchas opciones específicas, para conocer los diferentes datos del contenedor, en nuestro caso, para observar solo la dirección IP, se realiza de la siguiente manera.

```
[student@localhost ~]$ sudo podman inspect -f '{{ .NetworkSettings.IPAddress }}' my-httpd-container
10.88.0.19
```

Figura 34

- El comando **podman stop** detiene correctamente el contenedor que se encuentra en ejecución.

```
[student@localhost ~]$ sudo podman stop my-httpd-container
cb14e07541dd8b9a225b4a72da7c051c175a20b0f3240678e77b13e42cf4af1c
```

Figura 35

- El comando **podman kill** envía señales de Unix al proceso principal del contenedor. Si nos se especifica ninguna señal, automáticamente envía la señal SIGKILL, que finaliza el proceso principal y el contenedor.

```
[student@localhost ~]$ sudo podman kill my-httpd-container
cb14e07541dd8b9a225b4a72da7c051c175a20b0f3240678e77b13e42cf4af1c
```

Figura 36

Se puede especificar la señal con la opción -s.

```
[student@localhost ~]$ sudo podman kill -s SIGKILL my-httpd-container
cb14e07541dd8b9a225b4a72da7c051c175a20b0f3240678e77b13e42cf4af1c
```

Figura 37

Cualquier señal de Unix puede enviarse al proceso principal. Podman acepta el nombre y el número de la señal.
En la siguiente imagen se muestran las señales que pueden ser enviadas con el comando kill:

```
[student@localhost ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Figura 38

Las señales más utilizadas son las que se ven resaltadas.

- **SIGUP:** Se detectó un bloqueo en el terminal de control o la desactivación del proceso de control.
- **SIGKILL:** Señal de finalización inmediata.
- **SIGTERM:** Señal de terminación.

El comando podman restart reinicia un contenedor anteriormente detenido.

```
[student@localhost ~]$ sudo podman restart my-httpd-container
[sudo] password for student:
cb14e07541dd8b9a225b4a72da7c051c175a20b0f3240678e77b13e42cf4af1c
```

Figura 39

El comando podman rm elimina un contenedor, además, descarta su estado y sistema de archivos, pero el contenedor se debe encontrar detenido, de lo contrario muestra el siguiente error.

```
[student@localhost ~]$ sudo podman rm my-httpd-container
Error: cannot remove container cb14e07541dd8b9a225b4a72da7c051c175a20b0f3240678e77b13e42cf4af1c
as it is running - running or paused containers cannot be removed without force: container state
 improper
[student@localhost ~]$ sudo podman rm -f my-httpd-container
cb14e07541dd8b9a225b4a72da7c051c175a20b0f3240678e77b13e42cf4af1c
```

Figura 40

Con la opción **-f** del subcomando **rm**, le indica a Podman que elimine el contenedor, aunque no se encuentre detenido. Lo que hace es finalizar el contenedor a la fuerza y luego lo elimina.
La opción **-f** es equivalente a los comandos **podman kill** y **podman rm** juntos.

> **NOTA:** Si desea eliminar todos los contenedores al mismo tiempo, debe de agregar la opción **-a** al comando **podman rm,** pero sin olvidar que primero debe detenerlos todos y lo puede realizar con la opción podman **stop -a.**

CREACIÓN Y ADMINISTRACIÓN DE UNA BASE DE DATOS MYSQL

Lo primero que debemos de realizar es la creación de una instancia de contenedor MySQL.

```
[student@localhost ~]$ sudo podman run --name mysql-basic -e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55
-e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55 -d rhscl/mysql-57-rhel7:5.7-3.14
```

Figura 41

```
scl/mysql-57-rhel7:5.7-3.14
[sudo] password for student:
Trying to pull registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-3.14...
Getting image source signatures
Copying blob c5d2e9481169 skipped: already exists
Copying blob e373541ccf6a skipped: already exists
Copying blob b3949aed10eb done
Writing manifest to image destination
Storing signatures
76aacd7ba4ab4c6e09072877ac2d0bdf1bbdeed47107478ad720af154e7d6cf6
```

Figura 42

El comando anterior descarga la imagen de contenedor MySQL con la etiqueta **5.7-3.14** y luego inicia una imagen basada en contenedores.

Crea una base de datos llamada **items**, que es propiedad de un usuario llamado **user1** con la contraseña **mypa55.**

La contraseña del administrador de la base de datos es **r00tpa55.**
El contenedor se ejecuta en segundo plano.

Se comprueba que el contenedor haya sido iniciado sin errores, en nuestro caso, solo solicitamos los 3 parámetros que observamos en la siguiente imagen.

```
[student@localhost ~]$ sudo podman ps --format "{{.ID}} {{.Image}} {{.Names}}"
76aacd7ba4ab registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-3.14 mysql-basic
```

Figura 43

Se inspeccionan los metadatos del contenedor, para obtener la dirección IP desde la base de datos MySQL.

```
[student@localhost ~]$ sudo podman inspect -f '{{ .NetworkSettings.IPAddress }}' mysql-basic
[sudo] password for student:
10.88.0.25
```

Figura 44

Se accede al espacio aislado del contenedor.

```
[student@localhost ~]$ sudo podman exec -it mysql-basic /bin/bash
bash-4.2$
```

Figura 45

Lo que realiza el comando anterior en iniciar un Shell de Bash, que se ejecuta como el usuario **mysql** dentro del contenedor **MySQL.**

Se procede a agregar datos a la base de datos:
Lo primero es conectarse a MySQL como el usuario administrador de bases de datos (root).

```
bash-4.2$ mysql -uroot

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.16 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Figura 46

El comando **mysql** abre el prompt interactivo de la base de datos MySQL.

Con el siguiente comando se determina la disponibilidad de la base de datos.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| items              |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.38 sec)
```

Figura 47

Ejecutando el siguiente comando se puede acceder a la base de datos items, para después crear una nueva tabla.

```
mysql> use items;
Database changed
```

Figura 48

Se crea una tabla denominada Projects (Proyectos) en la base de datos items (ítems).

```
mysql> CREATE TABLE Projects (id int(11) NOT NULL, name varchar(255) DEFAULT NULL, code varchar(255) DEFAULT NULL, PRIMARY KEY (id));
Query OK, 0 rows affected (0.19 sec)
```

Figura 49

Verificamos que se haya creado la tabla correctamente.

```
mysql> show tables;
+----------------+
| Tables_in_items |
+----------------+
| Projects       |
+----------------+
1 row in set (0.01 sec)
```

Figura 50

Usamos el comando insert para insertar una fila en la tabla.

```
mysql> insert into Projects (id, name, code) values (1,'DevOps','D0180');
Query OK, 1 row affected (0.17 sec)
```

Figura 51

Por medio del comando select verificamos que la información del proyecto haya sido agregada a la tabla.

```
mysql> select * from Projects;
+----+--------+-------+
| id | name   | code  |
+----+--------+-------+
|  1 | DevOps | D0180 |
+----+--------+-------+
1 row in set (0.01 sec)
```

Figura 52

Salimos del prompt de MySQL y del contenedor de MySQL.

```
mysql> exit
Bye
bash-4.2$ exit
exit
[student@localhost ~]$
```

Figura 53

Creamos otro contenedor con la misma imagen de contenedor del contenedor anterior, mediante la ejecución de la Shell de /bin/bash.

```
[student@localhost ~]$ sudo podman run --name mysql-2 -it rhscl/mysql-57-rhel7:5.7-3.14 /bin/bash
[sudo] password for student:
bash-4.2$
```

Figura 54

Cuando nos intentamos conectar a la base de datos MySQL del contenedor nuevo nos aparecerá un error, debido a que no se está ejecutando, porque cuando se creó el contenedor nuevo, se modificó el punto de entrada responsable de iniciar la base de datos a /bin/bash.

```
bash-4.2$ mysql -uroot
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/var/lib/mysql/mysql.sock' (2)
```

Figura 55

Salimos de la Shell bash y se procede a verificar que el contenedor mysql-2 no se esté ejecutando.

```
bash-4.2$ exit
exit
[student@localhost ~]$ sudo podman ps -a --format="table {{.ID}} {{.Names}} {{.Status}}"
[sudo] password for student:
ID             Names              Status
93d669f720ad   mysql-2            Exited (1) About a minute ago
76aacd7ba4ab   mysql-basic        Up About an hour ago
```

Figura 56

Ejecutamos comandos en el contenedor separado. Se usa mypa55 como contraseña.

```
[student@localhost ~]$ sudo podman exec -it mysql-basic /bin/bash -c 'mysql -uuser1 -p -e "select * from items.Projects;"'
Enter password:
+----+--------+-------+
| id | name   | code  |
+----+--------+-------+
|  1 | DevOps | D0180 |
+----+--------+-------+
```

Figura 57

El comando anterior ejecuta un intérprete de **bash** en el contenedor **MySQL-basic**, después el comando le indica a **bash** que ejecute el intérprete **mysql**, el cual recibe la consulta de **SQL** logrando así obtener datos de la base de datos.

PERSISTENCIA DE UNA BASE DE DATOS MYSQL

Se puede decir que el almacenamiento que posee el contenedor es efímero, lo que significa que el contenido existente, no se conserva después de que el contenedor sea eliminado.

El almacenamiento efímero del contenedor no es suficiente para algunas aplicaciones que necesitan conservar los datos, después de los reinicios, un ejemplo claro, son las bases de datos. Para lograr que estas aplicaciones cumplan sus funciones correctamente, el administrador debe proporcionar un contenedor con almacenamiento persistente.

Cuando hablamos de persistencia de datos o información persistente nos referimos a que dichos datos sean almacenados de manera permanente sin importar si se realizan reinicios, logrando así que la información sea recuperada en las siguientes ejecuciones.

A continuación, se realizará un ejemplo de cómo se realiza la persistencia de una base de datos de mysql.

Lo primero que se debe realizar es la creación del directorio **/var/local/mysql** con los permisos correctos de SELinux.

```
[student@localhost ~]$ sudo mkdir -pv /var/local/mysql
[sudo] password for student:
mkdir: created directory '/var/local/mysql'
```

Figura 58

Se agrega el contexto SELinux adecuado al directorio **/var/local/mysql** para permitir que los contenedores accedan a todo su contenido.

```
[student@localhost ~]$ sudo semanage fcontext -a -t container_file_t '/var/local/mysql(/.*)?'
```

Figura 59

Aplicamos la política de SELinux al directorio recién creado.
El contexto SELinux **restorecon** significa Restaurar Contexto SELinux.

```
[student@localhost ~]$ sudo restorecon -R /var/local/mysql
```

Figura 60

Figura 60

Se verifica que el tipo de contexto de SELinux para el directorio **/var/local/mysql sea container_file_t.**

```
[student@localhost ~]$ sudo ls -dZ /var/local/mysql
unconfined_u:object_r:container_file_t:s0 /var/local/mysql
```

Figura 61

Cambiamos el propietario del directorio /var/local/mysql al usuario mysql y al grupo mysql.

```
[student@localhost ~]$ sudo chown -Rv 27:27 /var/local/mysql
changed ownership of '/var/local/mysql' from root:root to 27:27
```

Figura 62

Como podemos observar, anteriormente mencionamos el UID (User ID o Identificador de Usuario) del usuario mysql y el GID (Group ID o Identificador de Grupo) del grupo mysql es 27, de la siguiente manera lo comprobamos:

```
[student@localhost ~]$ sudo id -u mysql
27
[student@localhost ~]$ sudo id -g mysql
27
```

Figura 63

NOTA: El comando **id -u "nombre de usuario"** permite hallar el UID de un usuario especifico.
El comando **id -g "nombre de usuario"** permite hallar GID de un usuario especifico.

Se crea una instancia del contenedor MySQL con almacenamiento persistente. Primero extraemos la imagen de contenedor MySQL desde el registro interno.

```
[student@localhost ~]$ sudo podman pull rhscl/mysql-57-rhel7
Trying to pull registry.access.redhat.com/rhscl/mysql-57-rhel7...
Getting image source signatures
Copying blob 1d2c4ce43b78 done
Copying blob f1e961fe4c51 done
Copying blob 1c9f515fc6ab done
Copying blob 9f1840c3b3bd done
Copying config 60726b33a0 done
Writing manifest to image destination
Storing signatures
60726b33a00a2c3be60e25c3270a34a9b147db86602f05a71988a1c92a70cebc
```

Figura 64

Se crea un nuevo contenedor y se especifica el punto de montaje para almacenar los datos de la base de datos.

```
[student@localhost ~]$ sudo podman run --name persist-db -d -v /var/local/mysql:/var/lib/mysql/data
-e MYSQL_USER=user1 -e MYSQL_PASSWORD=mypa55 -e MYSQL_DATABASE=items -e MYSQL_ROOT_PASSWORD=r00tpa55
 rhscl/mysql-57-rhel7
[sudo] password for student:
3f4a7e0a83bf2d1d161c325dd2d6c5acafd8539301a7a512e2f069db73057b1b
```

Figura 65

El comando anterior monta el directorio /var/local/mysql del host en el directorio /var/lib/mysql/data del contenedor. /var/lib/mysql/data es el directorio donde la base de datos MySQL almacena los datos.

Después verificamos que se haya creado correctamente el contenedor.

```
[student@localhost ~]$ sudo podman ps --format="table {{.ID}} {{.Names}} {{.Status}}"
[sudo] password for student:
ID             Names        Status
3f4a7e0a83bf   persist-db   Up 9 minutes ago
```

Figura 66

Y por último verificamos que el directorio /var/local/mysql contenga un directorio items.

```
[student@localhost ~]$ ls -l /var/local/mysql
total 41036
-rw-r-----. 1 27 27        2 Nov  6 08:09 3f4a7e0a83bf.pid
-rw-r-----. 1 27 27       56 Nov  6 08:09 auto.cnf
-rw-------. 1 27 27     1676 Nov  6 08:09 ca-key.pem
-rw-r--r--. 1 27 27     1112 Nov  6 08:09 ca.pem
-rw-r--r--. 1 27 27     1112 Nov  6 08:09 client-cert.pem
-rw-------. 1 27 27     1676 Nov  6 08:09 client-key.pem
-rw-r-----. 1 27 27      673 Nov  6 08:09 ib_buffer_pool
-rw-r-----. 1 27 27 12582912 Nov  6 08:09 ibdata1
-rw-r-----. 1 27 27  8388608 Nov  6 08:09 ib_logfile0
-rw-r-----. 1 27 27  8388608 Nov  6 08:09 ib_logfile1
-rw-r-----. 1 27 27 12582912 Nov  6 08:09 ibtmp1
drwxr-x---. 2 27 27       20 Nov  6 08:09 items
drwxr-x---. 2 27 27     4096 Nov  6 08:09 mysql
-rw-r--r--. 1 27 27        6 Nov  6 08:09 mysql_upgrade_info
drwxr-x---. 2 27 27     8192 Nov  6 08:09 performance_schema
-rw-------. 1 27 27     1680 Nov  6 08:09 private_key.pem
-rw-r--r--. 1 27 27      452 Nov  6 08:09 public_key.pem
-rw-r--r--. 1 27 27     1112 Nov  6 08:09 server-cert.pem
-rw-------. 1 27 27     1680 Nov  6 08:09 server-key.pem
drwxr-x---. 2 27 27     8192 Nov  6 08:09 sys
```

Figura 67

Este directorio lo que hace es almacenar datos relacionados con la base de datos items que se creó en este contenedor.
Si este directorio no está disponible, el punto de montaje no se definió correctamente en la creación del contenedor.

MANIPULACIÓN DE IMÁGENES DE CONTENEDORES

GUARDADO Y CARGA DE IMÁGENES

Las imágenes que se encuentran en el almacenamiento local de Podman se pueden guardar en un archivo **.tar** con el comando **podman save.** El archivo que se genera no es una colección de archivos TAR normal; porque este contiene metadatos de la imagen y conserva las capas de la imagen original. Con este archivo, Podman puede recrear la imagen original exactamente como estaba.

La sintaxis general del subcomando **save** es la siguiente:

[student@localhost ~]$ sudo podman save [-o FILE_NAME] IMAGE_NAME[:TAG]

Para evitar que Podman envíe la imagen generada a la salida estándar como datos binarios, se usa la opción **-o**.

En el siguiente ejemplo, se guarda la imagen de contenedor de MySQL descargada anteriormente en el archivo **mysql.tar.**

```
[student@localhost ~]$ sudo podman save -o mysql.tar registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-3.14
```

Figura 68

Use los archivos .tar generados por el subcomando save con fines de copia de seguridad.

Para restaurar la imagen del contenedor, se utiliza el comando podman load. La sintaxis general del comando es la siguiente:

[student@localhost ~]$ sudo podman load [-i FILE_NAME]

El siguiente comando cargaría una imagen guardada en un archivo llamado **mysql.tar.**

```
[student@localhost ~]$ sudo podman load -i mysql.tar
[sudo] password for student:
Getting image source signatures
Copying blob acb06e825743 skipped: already exists
Copying blob 5444fe2e6b50 skipped: already exists
Copying blob d4d408077555 skipped: already exists
Copying config 4ae3a3f4f4 done
Writing manifest to image destination
Storing signatures
Loaded image(s): registry.access.redhat.com/rhscl/mysql-57-rhel7:5.7-3.14
```

Figura 69

Si el archivo .tar no es una imagen de contenedor con metadatos, el comando podman load fallará.

ELIMINACIÓN DE IMÁGENES

Podman mantiene todas las imágenes descargadas en su almacenamiento local, incluso las que no se están ejecutando actualmente en ningún contenedor. Sin embargo, las imágenes se pueden volver obsoletas y deben ser reemplazadas.

> **NOTA:** Las imágenes no se actualizan automáticamente. La imagen debe primero eliminarse y después descargar la versión más reciente, garantizando que el almacenamiento local se encuentre actualizado.

Para eliminar una imagen del almacenamiento local, se ejecuta el comando podman rmi.
La sintaxis del comando es la siguiente:

[student@localhost ~]$ sudo podman rmi [OPTIONS] IMAGE [IMAGE...]

Se puede eliminar la imagen usando su nombre o ID. Podman no puede eliminar una imagen si los contenedores se encuentran usándola; primero se deben detener y eliminar todos los contenedores que usan la imagen antes de eliminarla.

Si desea evitar lo mencionado anteriormente, el subcomando **rmi** tiene la opción **- -force.** Esta opción obliga a eliminar una imagen, incluso si se encuentra en varios contenedores. Podman detiene y elimina todos los contenedores que usan la imagen eliminada a la fuerza antes de eliminarla.

En nuestro caso, primero observamos las imágenes que tenemos con el comando **podman images** y luego procedemos a eliminar la imagen seleccionada, se realiza utilizando el ID de la imagen.

```
[student@localhost ~]$ sudo podman images
REPOSITORY                                      TAG         IMAGE ID       CREATED         SIZE
<none>                                          <none>      48b64ca2b212   5 days ago      215 MB
quay.io/lm1905/container                        latest      279f8af7b4a7   12 days ago     1.46 MB
docker.io/library/busybox                       latest      f0b02e9d092d   4 weeks ago     1.45 MB
registry.access.redhat.com/rhscl/httpd-24-rhel7 latest      610c5665a9e6   8 weeks ago     328 MB
registry.access.redhat.com/ubi7/ubi             7.7         0355cd652bd1   8 months ago    215 MB
registry.access.redhat.com/rhscl/mysql-57-rhel7 latest      60726b33a00a   13 months ago   448 MB
registry.access.redhat.com/rhscl/httpd-24-rhel7 2.4-36.8    5f9f898a3093   3 years ago     309 MB
registry.access.redhat.com/rhscl/mysql-57-rhel7 5.7-3.14    4ae3a3f4f409   3 years ago     418 MB
[student@localhost ~]$ sudo podman rmi --force 610c5665a9e6
Untagged: registry.access.redhat.com/rhscl/httpd-24-rhel7:latest
Deleted: 610c5665a9e62ba1f5f7a0a1af9f4715d82c007d3d1a5d38fcefa7328ae93bca
```

Figura 70

> NOTA: Si desea eliminar todas las imágenes que no son usadas por ningún contenedor, use el comando **podman rmi -a.**

MODIFICACIÓN DE IMÁGENES

Lo ideal sería que todas las imágenes de contenedores se compilen con Dockerfile, para generar un conjunto de capas de imágenes limpio y reducido. Sin embargo, algunos usuarios proporcionan imágenes de contenedores como se encuentran, sin un Dockerfile.

El comando **podman commit** nos permite la creación de nuevas imágenes, modificando un contenedor en ejecución en el lugar y guardando las capas, para crear una nueva imagen de contenedor.

La sintaxis del comando podman commit es la siguiente:

*[student@localhost ~]$ sudo podman commit [OPTIONS] CONTAINER *
> [REPOSITORY[:PORT]/]IMAGE_NAME[:TAG]

En la siguiente descripción, podemos observar las opciones más importantes disponibles para el comando podman commit:

- **- -autor ""**→ Identifica quien creó la imagen del contenedor.
- **- -message ""**→ Incluye un mensaje de confirmación en el registro. (Esta opción no se encuentra disponible en el formato de contenedor OCI predeterminado).
- **- -format ""**→ Selecciona el formato de la imagen. Las opciones validas son **oci** y **docker**.

Con el tiempo, los administradores pueden personalizar una imagen y establecer el contenedor en el estado que deseen.
Para identificar los archivos modificados, creados o eliminados desde el inicio del contenedor, se debe usar el subcomando **diff**. Este subcomando solo necesita el nombre o id del contenedor.

```
[student@localhost ~]$ sudo podman diff mysql-basic
C /etc
C /etc/opt/rh/rh-mysql57
C /etc/opt/rh/rh-mysql57/my.cnf.d
C /etc/opt
C /etc/opt/rh
A /etc/opt/rh/rh-mysql57/my.cnf.d/base.cnf
A /etc/opt/rh/rh-mysql57/my.cnf.d/paas.cnf
A /etc/opt/rh/rh-mysql57/my.cnf.d/tuning.cnf
C /tmp
C /var
C /var/lib
C /var/lib/mysql
A /var/lib/mysql/.bash_history
A /var/lib/mysql/.mysql_history
C /var/tmp
```

Figura 71

El subcomando **diff** etiqueta cualquier archivo agregado con una **A**, cualquier archivo modificado con una **C** y cualquier archivo eliminado con una **D**.

NOTA: Los archivos que son montados en un contenedor en ejecución no se consideran parte del sistema de archivos del contenedor.
Se recomienda usar el comando **podman inspect** para recuperar la lista de archivos y directorios montados para un contenedor en ejecución:
sudo podman inspect -f "{{range .Mounts}}{{println .Destination}}{{end}}" container_name/id

Para confirmar los cambios a otra imagen, se ejecuta el siguiente comando.

```
[student@localhost ~]$ sudo podman commit mysql-basic mysql-custom
Getting image source signatures
Copying blob d4d408077555 skipped: already exists
Copying blob 5444fe2e6b50 skipped: already exists
Copying blob acb06e825743 skipped: already exists
Copying blob f51044a70e44 done
Copying config 6f973879a7 done
Writing manifest to image destination
Storing signatures
6f973879a7d0248657fac5306a42270668ac1b980c4456e4e4ca407856ff3d20
```

Figura 72

ETIQUETADO DE IMÁGENES

Los registros de imágenes de contenedor admiten etiquetas para distinguir varias versiones del mismo proyecto. Se proporcionan etiquetas para identificar una versión con facilidad.

Para etiquetar una imagen se utiliza el comando podman tag, la sintaxis general es la siguiente:

*[student@localhost ~]$ sudo podman tag [OPTIONS] IMAGE[:TAG] *
>[REGISTRYHOST/][USERNAME/]NAME[:TAG]

Podman supone que se usa la ultima versión, como se observa en la etiqueta latest, este nombre también se observa si no se configura el valor de la etiqueta.

```
[student@localhost ~]$ sudo podman tag mysql-custom devops/mysql
```

Figura 73

Para usar un nombre de etiqueta diferente se realiza de la siguiente manera:

```
[student@localhost ~]$ sudo podman tag mysql-custom devops/mysql:snapshot
[student@localhost ~]$ sudo podman images
REPOSITORY                                  TAG         IMAGE ID       CREATED          SIZE
localhost/senaceai/mysql                    1.0         813a2bd1ea3c   21 hours ago     393 MB
localhost/mysql-custom                      latest      6f973879a7d0   23 hours ago     418 MB
localhost/devops/mysql                      latest      6f973879a7d0   23 hours ago     418 MB
localhost/devops/mysql                      snapshot    6f973879a7d0   23 hours ago     418 MB
docker.io/library/centos                    centos7     8652b9f0cb4c   4 days ago       212 MB
<none>                                      <none>      48b64ca2b212   6 days ago       215 MB
quay.io/lm1905/container                    latest      279f8af7b4a7   13 days ago      1.46 MB
docker.io/library/busybox                   latest      f0b02e9d092d   5 weeks ago      1.45 MB
registry.access.redhat.com/ubi7/ubi         7.7         0355cd652bd1   8 months ago     215 MB
registry.access.redhat.com/rhscl/mysql-57-rhel7   latest      60726b33a00a   13 months ago    448 MB
registry.access.redhat.com/rhscl/httpd-24-rhel7   2.4-36.8    5f9f898a3093   3 years ago      309 MB
registry.access.redhat.com/rhscl/mysql-57-rhel7   5.7-3.14    4ae3a3f4f409   3 years ago      418 MB
```

Figura 74

Eliminación de etiquetas de imágenes

Se pueden asignar diferentes etiquetas a una misma imagen con el comando **podman tag**, si se desean eliminarlas se debe usar el comando **podman rmi** como se menciona anteriormente.

```
[student@localhost ~]$ sudo podman rmi devops/mysql:snapshot
[sudo] password for student:
Untagged: localhost/devops/mysql:snapshot
```

Figura 75

> NOTA: Teniendo en cuenta que varias etiquetas pueden apuntar a la misma imagen, si desea eliminar una imagen que posee diferentes etiquetas, primero debe eliminar cada etiqueta de forma individual antes de eliminar la imagen.

PUBLICACIÓN DE IMÁGENES EN UN REGISTRO

Para publicar una imagen en un registro, esa imagen debe encontrarse en el almacenamiento local de Podman y ser etiquetada para lograr identificarla.

Para enviar la imagen a su registro, la sintaxis del subcomando **push** es la siguiente:

[student@localhost ~]$ sudo podman push [OPTIONS] IMAGE [DESTINATION]

En nuestro caso, publicaremos la imagen en nuestro registro de Quay.io de la siguiente manera:

```
[student@localhost ~]$ sudo podman push localhost/senaceai/mysql:1.0 quay.io/lm1905/container
Getting image source signatures
Copying blob 9024d18f3d6d done
Copying blob 83cddb25830f done
Copying blob 11e6dd86fed3 done
Copying blob 174f56854903 done
Copying blob e8443d37bd94 done
Copying config 813a2bd1ea done
Writing manifest to image destination
Copying config 813a2bd1ea done
Writing manifest to image destination
Storing signatures
```

Figura 76

NOTA: Se debe tener en cuenta que el repositorio en el cual se publica la imagen debe de estar público.

Por último, verificamos que la imagen se encuentre disponible en Quay.io, en nuestro caso ingresamos de manera gráfica a Quay.io y podemos observarla:

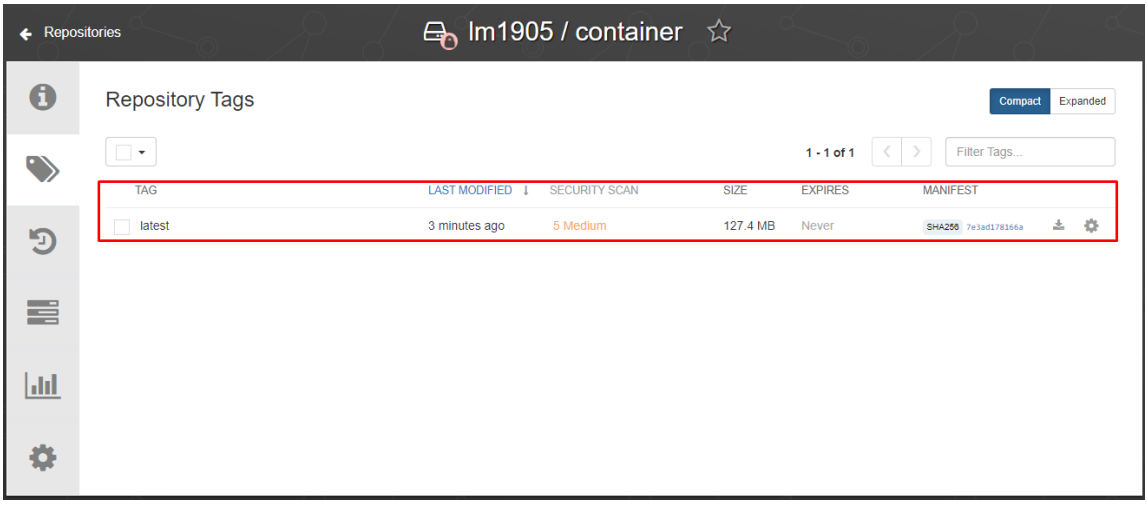

Figura 77

COMPILACIÓN DE IMÁGENES DE CONTENEDORES PERSONALIZADAS CON DOCKERFILES

¿QUÉ ES UN DOCKERFILE?

Es un archivo de texto que debe existir en el directorio de trabajo. En este archivo se encuentran las instrucciones necesarias para la compilación de una imagen de contenedor.

La compilación de una imagen a partir de un Dockerfile es un proceso de tres pasos:

1. Crear un directorio de trabajo.
2. Escribir el Dockerfile.
3. Compilar la imagen con Podman.

Crear un directorio de trabajo

El directorio de trabajo es el directorio donde se alojan todos los archivos que son necesarios para compilar la imagen. El crear un directorio de trabajo vacío es lo más recomendable, debido a que así se evita incorporar archivos innecesarios a la imagen. Por seguridad, nunca debería usarse el directorio **root** ni / como directorio de trabajo para compilaciones de imágenes.

Escribir el Dockerfile

La sintaxis básica de un Dockerfile es:

En el siguiente ejemplo se muestra un Dockerfile para compilar un contenedor de servidores web simple de Apache.

```
1 # This is a comment line
2 FROM ubi7/ubi:7.7
3 LABEL description="This is a custom httpd container image"
4 MAINTAINER John Doe <jdoe@xyz.com>
5 RUN yum install -y httpd
6 EXPOSE 80
7 ENV LogLevel "info"
8 ADD http://someserver.com/filename.pdf /var/www/html
9 COPY ./src/ /var/www/html/
10 USER apache
11 ENTRYPOINT ["/usr/sbin/httpd"]
12 CMD ["-D", "FOREGROUND"]
```

Figura 78

1. Las líneas que comienzan con numeral (#) son comentarios.

2. **FROM** declara que la nueva imagen del contenedor se extiende a la imagen base del contenedor ubi7/ubi:7.7. Dockerfile también puede usar cualquier otra imagen de contenedor como imagen base.

3. **LABEL** es responsable de agregar metadatos genéricos a una imagen.

4. **MAINTAINER** indica el campo **Author** (Autor) de los metadatos de la imagen del contenedor generada.

5. **RUN** ejecuta comando en una nueva capa sobre la imagen actual. La Shell usada para ejecutar comando es **/bin/sh.**

6. **EXPOSE** indica que el contenedor escucha en el puerto de red especificado en tiempo de ejecución.

7. **ENV** es responsable de definir las variables de entorno que están disponibles en el contenedor. Se pueden declarar varias instrucciones ENV dentro de un Dockerfile.

8. **ADD** copia archivos o carpetas de una fuente local o remota y los agrega al sistema de archivos del contenedor. Esta instrucción desempaqueta los archivos de .tar locales en el directorio de la imagen de destino.

9. **COPY** copia archivos desde el directorio de trabajo y los agrega al sistema de archivos del contenedor. No es posible copiar un archivo remoto mediante su URL con esta instrucción Dockerfile.

10. **USER** especifica el nombre de usuario o ID que se usará cuando se ejecute la imagen de contenedor para las instrucciones **RUN**, **CMD** y **ENTRYPOINT**. Se recomiendo definir un usuario diferente a root por motivos de seguridad.

11. **ENTRYPOINT** especifica el comando predeterminado para ejecutarse cuando se ejecuta la imagen en un contenedor. Si se omite, el valor predeterminado de ENTRYPOINT es **/bin/sh -c.**

12. **CMD** proporciona los argumentos predeterminados para la instrucción ENTRYPOINT. Si aplica el valor predeterminado de ENTRYPOINT (/bin/sh -c), CMD forma un comando ejecutable y los parámetros que se ejecutan al inicio del contenedor.

Compilar la imagen con PODMAN

El comando podman build procesa el Dockerfile y compila una nueva imagen, sobre la base de las instrucciones que contiene.
La sintaxis del comando es la siguiente:

[student@localhost ~]$ sudo podman build -t NAME:TAG DIR

- DIR es la ruta del directorio de trabajo, en el cual se encuentra el Dockerfile. Si el directorio de trabajo es el actual se distingue con un punto (.).

- NAME:TAG es el nombre con una etiqueta que se le asigna a una nueva imagen. Si TAG no se especifica, la imagen se etiqueta automáticamente como latest (más reciente).

CREACIÓN DE UNA IMAGEN DE CONTENEDOR DE MARIADB BÁSICA

Lo primero que se realiza es loguearse en Quay.io.

```
[root@localhost ~]# podman login quay.io
Username: lm1905
Password:
Login Succeeded!
```

Figura 79

Se realiza la descarga de un archivo tarball que contiene archivos de apoyo.

[root@localhost ~]$ wget https://access.redhat.com/webassets/avalon/d/Red_Hat_Enterprise_Linux_Atomic_Host-7-Getting_Started_with_Containers-en-US/files/mariadb_cont_2.tgz

```
[root@localhost ~]# wget https://access.redhat.com/webassets/avalon/d/Red_Hat_Enterprise_
Linux_Atomic_Host-7-Getting_Started_with_Containers-en-US/files/mariadb_cont_2.tgz
--2020-11-19 15:36:18--  https://access.redhat.com/webassets/avalon/d/Red_Hat_Enterprise_
Linux_Atomic_Host-7-Getting_Started_with_Containers-en-US/files/mariadb_cont_2.tgz
Resolving access.redhat.com (access.redhat.com)... 104.92.193.20
Connecting to access.redhat.com (access.redhat.com)|104.92.193.20|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: 'mariadb_cont_2.tgz'

mariadb_cont_2.tgz          [ <=>                   ] 137.16K  --.-KB/s    in 0.1s

2020-11-19 15:36:25 (1.15 MB/s) - 'mariadb_cont_2.tgz' saved [140448]
```

Figura 80

Se crea el directorio de trabajo, que contendrá los archivos de configuración.

```
[root@localhost ~]# mkdir ~/mydbcontainer
[root@localhost ~]# ls
anaconda-ks.cfg  mariadb_cont_2.tgz  mydbcontainer
```

Figura 81

Se copia el archivo descargado al directorio de trabajo creado anteriormente e ingresamos a él.

```
[root@localhost ~]# cp mariadb_cont*.tgz ~/mydbcontainer
[root@localhost ~]# cd ~/mydbcontainer
[root@localhost mydbcontainer]# ls
mariadb_cont_2.tgz
```

Figura 82

Se descomprime el archivo, el cual contiene archivos de apoyo.

```
[root@localhost mydbcontainer]# tar xvf mariadb_cont*.tgz
gss_db.sql
Dockerfile
```

Figura 83

Se edita el archivo Dockerfile según sea necesario, se puede observar en la figura 85.

```
[root@localhost mydbcontainer]# vim Dockerfile
```

Figura 84

```
# Database container with simple data for a Web application
# Using RHEL 7 base image and MariahDB database
# Version 1

# Pull the rhel image from the local repository
FROM centos:centos7
USER root

MAINTAINER SENA-CEAI

# Update image
RUN yum update -y

# Add Mariahdb software
RUN yum -y install net-tools mariadb-server

# Set up Mariahdb database
ADD gss_db.sql /tmp/gss_db.sql
RUN /usr/libexec/mariadb-prepare-db-dir
RUN test -d /var/run/mariadb || mkdir /var/run/mariadb; \
    chmod 0777 /var/run/mariadb; \
    /usr/bin/mysqld_safe --basedir=/usr & \
    sleep 10s && \
    /usr/bin/mysqladmin -u root password 'redhat' && \
    mysql --user=root --password=redhat < /tmp/gss_db.sql && \
    mysqladmin shutdown --password=redhat

# Expose Mysql port 3306
EXPOSE 3306

# Start the service
CMD test -d /var/run/mariadb || mkdir /var/run/mariadb; chmod 0777 /var/run/mariadb;/usr/bin/mysqld_safe --basedir=/usr
```

Figura 85

Se compila la imagen para realizar la construcción de la imagen de contenedor a partir del Dockerfile.

```
[root@localhost mydbcontainer]# podman build -t senaceai/mydb .
```

Figura 86

Se muestra el análisis que se realiza con cada una de las líneas del Dockerfile.

```
[root@localhost mydbcontainer]# podman build -t senaceai/mydb .
STEP 1: FROM centos:centos7
Getting image source signatures
Copying blob 2d473b07cdd5 done
Copying config 8652b9f0cb done
Writing manifest to image destination
Storing signatures
STEP 2: USER root
a61ebf684f7a577757ddfc6727f05b4004f180cbc3b9668a81c8949893aaf8da
STEP 3: MAINTAINER SENA-CEAI
82591bf1780e341d78becd77d2389e220832d7e4cd90361493993f42ad3fc494
STEP 4: RUN yum update -y
Loaded plugins: fastestmirror, ovl
Determining fastest mirrors
 * base: centos.ufes.br
 * extras: centos.ufes.br
 * updates: mirror.unimagdalena.edu.co
base                                                     | 3.6 kB     00:00
extras                                                   | 2.9 kB     00:00
updates                                                  | 2.9 kB     00:00
```

Figura 87

NOTA: La imagen de contenedor que se encuentra en la instrucción FROM, solo se descarga si no se encuentra disponible en el almacenamiento local.

```
Complete!
3f8f37ddd5e60880bf4fcc160e6ce53f935ac4b8208139e0727041b0bf66f1ef
STEP 5: RUN yum -y install net-tools mariadb-server
Loaded plugins: fastestmirror, ovl
Loading mirror speeds from cached hostfile
 * base: centos.ufes.br
 * extras: centos.ufes.br
 * updates: mirror.unimagdalena.edu.co
```

Figura 88

```
Complete!
8b9c15cbc0ba22d3064f3b59acd7be85c47c663413509901b7090a1b66714290
STEP 6: ADD gss_db.sql /tmp/gss_db.sql
d3ae018ba76fb8973831be184dd85563679dc4f62695b457e75deda8d71ad3b8
STEP 7: RUN /usr/libexec/mariadb-prepare-db-dir
Failed to get D-Bus connection: Operation not permitted
Failed to get D-Bus connection: Operation not permitted
Initializing MariaDB database
201119 21:16:07 [Note] /usr/libexec/mysqld (mysqld 5.5.68-MariaDB) starting as process 72 ...
201119 21:16:10 [Note] /usr/libexec/mysqld (mysqld 5.5.68-MariaDB) starting as process 80 ...
```

Figura 89

NOTA: Como podemos observar en la imagen anterior en el paso 7 (step 7), el servicio no se puede iniciar en el contenedor Podman de CentOS7, debido a que no se puede usar systemctl.

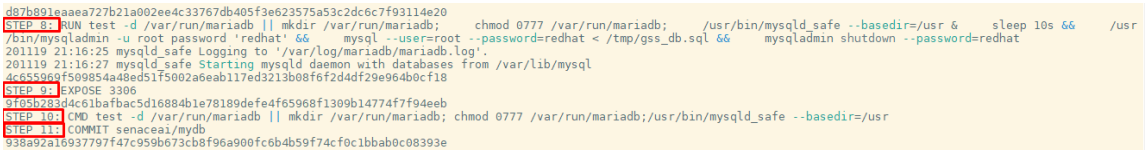

Figura 90

Se ejecuta el comando podman images para observar la nueva imagen en el repositorio de imágenes.

```
[root@localhost mydbcontainer]# podman images
REPOSITORY                                    TAG        IMAGE ID       CREATED          SIZE
localhost/senaceai/mydb                       latest     938a92a16937   20 minutes ago   643 MB
localhost/mysql-custom                        latest     6f973879a7d0   2 days ago       418 MB
localhost/devops/mysql                        latest     6f973879a7d0   2 days ago       418 MB
docker.io/library/centos                      centos7    8652b9f0cb4c   5 days ago       212 MB
docker.io/library/busybox                     latest     f0b02e9d092d   5 weeks ago      1.45 MB
registry.access.redhat.com/ubi7/ubi           7.7        0355cd652bd1   8 months ago     215 MB
registry.access.redhat.com/rhscl/mysql-57-rhel7   latest     60726b33a00a   13 months ago    448 MB
registry.access.redhat.com/rhscl/httpd-24-rhel7   2.4-36.8   5f9f898a3093   3 years ago      309 MB
registry.access.redhat.com/rhscl/mysql-57-rhel7   5.7-3.14   4ae3a3f4f409   3 years ago      418 MB
```

Figura 91

Se inicia un contenedor, teniendo como imagen, la creada anteriormente. También se le define el nombre, el puerto y que sea ejecutado en segundo plano.

```
[root@localhost ~]# podman run -d -p 3306:3306 --name=mydb_container senaceai/mydb
a6f1e0a88c1cac69d0486905b0d9540efa1871fa8ad62dd64d976df2fa12ad80
```

Figura 92

Una vez que se encuentre corriendo el contenedor, se verifica el ID, la imagen, el puerto y el nombre.

```
[root@localhost ~]# podman ps --format "{{.ID}} {{.Image}} {{.Ports}} {{.Names}}"
a6f1e0a88c1c localhost/senaceai/mydb:latest 0.0.0.0:3306-&gt;3306/tcp mydb_container
```

Figura 93

Se verifica la IP del contenedor.

```
[root@localhost ~]# podman inspect --format '{{ .NetworkSettings.IPAddress }}' a6f1e0a88c1c
10.88.0.48
```

Figura 94

Se verifica que, con esa IP, el puerto se encuentre a la escucha.

```
[root@localhost ~]# nc -v 10.88.0.48 3306
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Connected to 10.88.0.48:3306.
R
5.5.68-MariaDBk=/qlLu,▒▒t*otw{"5$T\Omysql_native_password^C
```

Figura 95

NOTA: Recuerde instalar el paquete nc (dnf install nc -y), para la ejecución del comando anterior.

Se ingresa al /bin/bash del contenedor y después al servicio de mysql definiendo un usuario, el cual fue creado anteriormente.

```
[root@localhost ~]# podman exec -it mydb_container /bin/bash
[root@a6f1e0a88c1c /]# mysql -u mysqladmin
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 5.5.68-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Figura 96

Se observan las bases de datos que se encuentran creadas.

```
MariaDB [(none)]> show databases ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| test               |
+--------------------+
2 rows in set (0.09 sec)
```

Figura 97

Por último, salimos de mysql, después del contenedor y se verifica que el contenedor aún se encuentra corriendo y cuenta con las mismas características.

```
MariaDB [(none)]> exit
Bye
[root@a6f1e0a88c1c /]# exit
exit
[root@localhost ~]# podman ps
CONTAINER ID  IMAGE                          COMMAND               CREATED         STATUS             PORTS                   NAMES
a6f1e0a88c1c  localhost/senaceai/mydb:latest  /bin/sh -c test -...  14 minutes ago  Up 14 minutes ago  0.0.0.0:3306->3306/tcp  mydb_container
```

Figura 98

GLOSARIO

CentOS: Community Enterprise Operating System. Es una distribución del sistema operativo Linux utilizada sobre todo en el entorno del servidor.

Comprimir: Reducir el tamaño de un archivo o información mediante técnicas de compresión, lo que facilita su archivo y su manejo. La extensión de archivos .tar (Tape ARchive) es una herramienta de software utilizada para recopilar varios archivos en un solo archivo.

Contenerización: O contenedorización, es un método de virtualización a nivel de sistema operativo que se utiliza para implementar y ejecutar aplicaciones distribuidas sin iniciar una máquina virtual.

CoreOS RKT: Es un motor de contenedores de aplicaciones desarrollado para entornos nativos de la nube de producción moderna. Cuenta con el formato de imagen nativo (ACI) de rkt y el entorno de ejecución / ejecución (pods).

CPU: La unidad central de procesamiento o CPU (por el acrónimo en inglés de central processing unit), es el componente del computador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.

Daemon: Un demonio en Linux, y de manera general en cualquier sistema tipo UNIX, es un proceso que se ejecuta en segundo plano y es autónomo, de manera que no necesita interacción por parte de un usuario del sistema para arrancar y funcionar.

Docker: Es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software de Linux.

Elasticidad: Es la capacidad de ampliar o reducir rápidamente los recursos informáticos de procesamiento, memoria y almacenamiento para satisfacer demandas variables.

Git Hub: Es un sistema de gestión de proyectos y control de versiones de código, permite trabajar en colaboración con otras personas de todo el mundo, planificar proyectos y realizar un seguimiento del trabajo.

Hardware: Conjunto de componentes materiales de un sistema informático. Cada

una de las partes físicas que forman un ordenador, incluidos sus periféricos. Maquinaria y equipos (CPU, discos, cintas, módem, cables, etc.).

Herramientas de monitorización: Son sistemas de diagnóstico para telecomunicaciones, servidores o redes que buscan componentes defectuosos o lentos, con el fin de informar a los administradores mediante correo electrónico, sms, entre otros.

Hipervisor: Es un software que crea y ejecuta máquinas virtuales, aísla el sistema operativo y los recursos del hipervisor de las máquinas virtuales, y permite crearlas y gestionarlas.

Host: Computador central o principal en un entorno de procesamiento distribuido y/o sistema informático complejo.

HTTP: Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertextos). Es un protocolo de acceso para las páginas web a través de Internet.

Integradores: Se encargan de unir componentes independientes, para crear un sistema único, adecuado a la realidad y tareas particulares de una compañía.

IPC: (Inter-Process Communication). Comunicación entre procesos.

Kernel: Núcleo. Parte fundamental de un programa, por lo general de un sistema operativo, que reside en memoria todo el tiempo, provee los servicios básicos y es el encargado de hacer interactuar el software con el hardware.

Kubernetes: Es una plataforma open source que automatiza las operaciones de los contenedores de Linux. Elimina muchos de los procesos manuales involucrados en la implementación y escalabilidad de las aplicaciones en contenedores.

LXC: Es una tecnología de virtualización en el nivel de sistema operativo para Linux. LXC permite que un servidor físico ejecute múltiples instancias de sistemas operativos aislados.

Máquina virtual: Es un software que simula un sistema de computación y puede ejecutar programas como si fuese una computadora real.

Mesos Containerizer: Proporciona contenedores ligeros y aislamiento de recursos de los ejecutores utilizando funciones específicas de Linux, como grupos de control y espacios de nombres. Permite que las tareas se ejecuten con una serie de aisladores conectables proporcionados por Mesos.

MYSQL: Es un sistema de gestión de base de datos relacional de código abierto. Puede ser ejecutado prácticamente todas las plataformas, incluyendo Linux, UNIX y Windows.

Open Shift: Es una plataforma de desarrollo, de computación en la nube que ofrece Red Hat.

OpenVZ: Es un software para virtualizar entornos en Linux, específicamente, servidores privados virtuales (VPS) y entornos virtuales (EV).

Pod: Es un conjunto de uno o varios contenedores de Linux y constituye la unidad más pequeña de las aplicaciones de Kubernetes.

Podman: Es un motor de contenedores, desarrollado por Red Hat y que puede ser un sustituto para Docker.

Proceso: Es la ejecución de diversas instrucciones por parte del microprocesador, de acuerdo a lo que indica un programa.
Red Hat: Es la plataforma empresarial Linux líder en el mundo. Se trata de un sistema operativo open source. Es lo que permite ampliar las aplicaciones actuales e implementar tecnologías nuevas en equipos sin sistema operativo, entornos virtuales, contenedores y todo tipo de entornos de nube.

Reutilización: Es el proceso de creación de sistemas de software a partir de un software existente, en lugar de tener que rediseñar desde el principio.

Root: Usuario de un sistema UNIX encargado de la administración de la máquina. El usuario root opera sin ningún tipo de restricción, por lo tanto, únicamente debe utilizarse para tareas administrativas.

Runtime: Tiempo de ejecución. Tiempo en el que un programa se ejecuta en un sistema operativo.

Shell: Parte fundamental de un sistema operativo encargada de ejecutar las órdenes básicas para el manejo del sistema.

Sistema Operativo: Conjunto de programas fundamentales sin los cuales no sería posible hacer funcionar el ordenador con los programas de aplicación que se desee utilizar.

Tecnología de Orquestación: Crea una infraestructura de aplicaciones alineadas que pueden ser escaladas hacia arriba o hacia abajo en función de las necesidades de cada aplicación.

Tecnología disruptiva: Cualquier tecnología o innovación que deja obsoleta la tecnología anterior.

Virtual Box: Es un software de virtualización para arquitecturas x86/amd64.

Actualmente es desarrollado por Oracle Corporation como parte de su familia de productos de virtualización.

Virtualización: Es una tecnología que permite crear servicios de TI útiles mediante recursos que están ligados tradicionalmente al hardware.

VServer: Linux Virtual Server. Es una implementación de servidor privado virtual realizada por Linux.

BIBLIOGRAFÍA

Arquitecto IT. (2020). Retrieved 20 November 2020, from http://www.arquitectoit.com/

Callejas, A., & Islas, X. (2020). |rootzilopochtli.com|. Retrieved 20 November 2020, from http://www.rootzilopochtli.com/

Matthias, K., & Kane, S. P. (2015). Docker: Up & Running: Shipping Reliable Containers in Production. " O'Reilly Media, Inc.".

Fugaro, L., & Vocale, M. (2019). Hands-On Cloud-Native Microservices with Jakarta EE. [Place of publication not identified]: Packt Publishing.

Schenker, G. N. (2018). Learn Docker-Fundamentals of Docker 18. x: Everything you need to know about containerizing your applications and running them in production. Packt Publishing.

Bullington-McGuire, R., Dennis, A., & Schwartz, M. (2020). Docker for Developers. [S.l.]: Packt Publishing.

Diccionario informático. (2020). Retrieved 20 November 2020, from https://www.lawebdelprogramador.com/diccionario/