Red Hat Enterprise Linux 8

Creación, ejecución y gestión de contenedores

Creación, ejecución y gestión de contenedores Linux en Red Hat Enterprise Linux 8

Last Updated: 2021-02-25

Red Hat Enterprise Linux 8 Creación, ejecución y gestión de contenedores

Creación, ejecución y gestión de contenedores Linux en Red Hat Enterprise Linux 8

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

Legal Notice

Resumen

Esta guía describe cómo trabajar con contenedores Linux en sistemas RHEL 8 utilizando herramientas de línea de comandos como podman, buildah, skopeo y runc.

Table of Contents

PREFACIO

Red Hat clasifica los casos de uso de los contenedores en dos grupos distintos: de un solo nodo y de varios nodos, y los de varios nodos se denominan a veces sistemas distribuidos. OpenShift se construyó para proporcionar despliegues públicos y escalables de aplicaciones en contenedores. Más allá de OpenShift, sin embargo, es útil tener un pequeño y ágil conjunto de herramientas para trabajar con contenedores.

El conjunto de herramientas de contenedores al que nos referimos puede utilizarse en un caso de uso de un solo nodo. Sin embargo, también puede conectar estas herramientas a los sistemas de compilación existentes, a los entornos CI/CD e incluso utilizarlas para abordar casos de uso específicos de la carga de trabajo, como los grandes datos. Para el caso de uso de un solo nodo, Red Hat Enterprise Linux (RHEL) 8 ofrece un conjunto de herramientas para encontrar, ejecutar, construir y compartir contenedores individuales.

Este manual describe cómo trabajar con contenedores Linux en sistemas RHEL 8 utilizando herramientas de línea de comandos como podman, buildah, skopeo y runc. Además de estas herramientas, Red Hat proporciona imágenes base, para actuar como la base de sus propias imágenes. Algunas de estas imágenes base se dirigen a casos de uso que van desde las aplicaciones empresariales (como Node.js, PHP, Java y Python) hasta la infraestructura (como el registro, la recopilación de datos y la autenticación).

HACER QUE EL CÓDIGO ABIERTO SEA MÁS INCLUSIVO

Red Hat se compromete a sustituir el lenguaje problemático en nuestro código, documentación y propiedades web. Estamos empezando con estos cuatro términos: maestro, esclavo, lista negra y lista blanca. Debido a la enormidad de este esfuerzo, estos cambios se implementarán gradualmente a lo largo de varias versiones próximas. Para más detalles, consulte el mensaje de nuestro CTO Chris Wright .

PROPORCIONAR COMENTARIOS SOBRE LA DOCUMENTACIÓN DE RED HAT

Agradecemos su opinión sobre nuestra documentación. Por favor, díganos cómo podemos mejorarla. Para ello:

- Para comentarios sencillos sobre pasajes concretos:
  1. Asegúrese de que está viendo la documentación en el formato *Multi-page HTML*. Además, asegúrese de ver el botón **Feedback** en la esquina superior derecha del documento.
  2. Utilice el cursor del ratón para resaltar la parte del texto que desea comentar.
  3. Haga clic en la ventana emergente **Add Feedback** que aparece debajo del texto resaltado.
  4. Siga las instrucciones mostradas.
- Para enviar comentarios más complejos, cree un ticket de Bugzilla:
  1. Vaya al sitio web de Bugzilla.
  2. Como componente, utilice **Documentation**.
  3. Rellene el campo **Description** con su sugerencia de mejora. Incluya un enlace a la(s) parte(s) pertinente(s) de la documentación.
  4. Haga clic en **Submit Bug**.

CAPÍTULO 1. EMPEZANDO POR LOS CONTENEDORES

Los contenedores Linux han surgido como una tecnología clave de empaquetado y entrega de aplicaciones de código abierto, que combina el aislamiento ligero de las aplicaciones con la flexibilidad de los métodos de despliegue basados en imágenes.

Red Hat Enterprise Linux implementa los contenedores de Linux utilizando tecnologías básicas como:

- Grupos de control (cgroups) para la gestión de recursos
- Espacios de nombres para el aislamiento de procesos
- SELinux para la seguridad
- Multitenencia segura

para reducir el potencial de los exploits de seguridad. Todo esto está pensado para proporcionarle un entorno para producir y ejecutar contenedores de calidad empresarial.

Red Hat OpenShift proporciona potentes herramientas de línea de comandos y de interfaz web para construir, gestionar y ejecutar contenedores en unidades denominadas **pods**. Sin embargo, hay ocasiones en las que puede querer construir y gestionar contenedores individuales e imágenes de contenedores fuera de OpenShift. En esta guía se describen las herramientas proporcionadas para realizar esas tareas que se ejecutan directamente en los sistemas RHEL.

A diferencia de otras implementaciones de herramientas de contenedores, las herramientas descritas aquí no se centran en el motor de contenedores monolítico de Docker y el comando **docker**. En su lugar, proporcionamos un conjunto de herramientas de línea de comandos que pueden funcionar sin un motor de contenedores. Estas incluyen:

- **podman** - Para gestionar directamente los pods y las imágenes de contenedores (run, stop, start, ps, attach, exec, etc.)
- **buildah** - Para construir, empujar y firmar imágenes de contenedores
- **skopeo** - Para copiar, inspeccionar, borrar y firmar imágenes
- **runc** - Para proporcionar funciones de ejecución y construcción de contenedores a podman y buildah

Dado que estas herramientas son compatibles con la Open Container Initiative (OCI), pueden utilizarse para gestionar los mismos contenedores Linux que producen y gestionan Docker y otros motores de contenedores compatibles con la OCI. Sin embargo, son especialmente adecuadas para ejecutarse directamente en Red Hat Enterprise Linux, en casos de uso de un solo nodo.

Para una plataforma de contenedores de múltiples nodos, consulte OpenShift. En lugar de confiar en las herramientas de un solo nodo y sin demonio descritas en este documento, OpenShift requiere un motor de contenedores basado en demonio. Por favor, consulte Uso del motor de contenedores CRI-O para más detalles.

1.1. EJECUCIÓN DE CONTENEDORES SIN DOCKER

Red Hat no sólo eliminó el motor de contenedores Docker de OpenShift. También eliminó el motor de contenedores Docker, junto con el comando **docker**, de Red Hat Enterprise Linux 8 por completo. Para RHEL 8, Docker no está incluido y no es soportado por Red Hat (aunque todavía está disponible en otras fuentes).

La eliminación de Docker refleja un cambio en la forma de pensar de Red Hat sobre el manejo de los contenedores:

- En la empresa, la atención no se centra en la ejecución de contenedores individuales desde la línea de comandos. El lugar principal para ejecutar contenedores es una plataforma basada en Kubernetes, como OpenShift.
- Al reposicionar OpenShift como la plataforma para ejecutar contenedores, los motores de contenedores como Docker se convierten en un componente más abstraído por OpenShift.
- Debido a que el motor de contenedores en OpenShift no está destinado a ser utilizado directamente, se puede implementar con un conjunto de características limitadas que se centran en hacer todo lo que OpenShift necesita, sin tener que implementar muchas características independientes.

Aunque Docker ha desaparecido de RHEL 8, y el motor de contenedores de OpenShift está desconectado de los usos de un solo nodo, la gente todavía quiere usar comandos para trabajar con contenedores e imágenes manualmente. Así que Red Hat se puso a crear un conjunto de herramientas para implementar la mayor parte de lo que hace el comando **docker**.

Herramientas como **podman**, **skopeo**, y **buildah** se desarrollaron para asumir esas características del comando **docker**. Cada herramienta en este escenario puede ser más ligera y centrarse en un subconjunto de características. Y sin necesidad de que se ejecute un proceso daemon para implementar un motor de contenedores, estas herramientas pueden funcionar sin la sobrecarga de tener que trabajar con un proceso daemon.

Si todavía quiere usar Docker en RHEL 8, puede obtener Docker de diferentes proyectos upstream, pero no está soportado en RHEL 8. Debido a que muchas de las características de la línea de comandos de **docker** han sido implementadas exactamente en **podman**, puede configurar un alias para que al escribir **docker** se ejecute podman.

Al instalar el paquete podman-docker se configura un alias de este tipo. Así que cada vez que se ejecuta una línea de comandos **docker**, en realidad se ejecuta **podman** por usted.

1.2. ELECCIÓN DE UNA ARQUITECTURA RHEL PARA CONTENEDORES

Red Hat proporciona imágenes de contenedores y software relacionado con los contenedores para las siguientes arquitecturas informáticas:

- AMD64 e Intel 64 (imágenes base y por capas; no es compatible con arquitecturas de 32 bits)
- PowerPC 8 y 9 de 64 bits (imagen base y la mayoría de las imágenes en capas)
- IBM Z (imagen base y la mayoría de las imágenes en capas)
- ARM 64 bits (sólo imagen base)

Aunque al principio no todas las imágenes de Red Hat eran compatibles con todas las arquitecturas, ahora casi todas están disponibles en todas las arquitecturas de la lista. Consulte Imágenes base universales (UBI): Imágenes , repositorios y paquetes para una lista de imágenes soportadas.

1.3. OBTENCIÓN DE HERRAMIENTAS PARA CONTENEDORES

Para obtener un entorno en el que pueda manipular contenedores individuales, puede instalar un sistema Red Hat Enterprise Linux 8 y, a continuación, añadir un conjunto de herramientas de contenedores para encontrar, ejecutar, construir y compartir contenedores. Aquí hay ejemplos de

herramientas relacionadas con contenedores que puede instalar con RHEL 8:

- **podman** - Herramienta cliente para la gestión de contenedores. Puede reemplazar la mayoría de las funciones del comando **docker** para trabajar con contenedores e imágenes individuales.
- **buildah** - Herramienta cliente para construir imágenes de contenedores compatibles con OCI.
- **skopeo** - Herramienta cliente para copiar imágenes de contenedores desde y hacia los registros de contenedores. Incluye funciones para firmar y autenticar imágenes también.
- **runc** - Cliente de tiempo de ejecución de contenedores para ejecutar y trabajar con contenedores de formato Open Container Initiative (OCI).

Si desea crear imágenes de contenedores utilizando el modelo de suscripción de RHEL, debe registrar y otorgar derechos adecuadamente al equipo anfitrión en el que los construya. Cuando se instalan paquetes, como parte del proceso de construcción de un contenedor, el proceso de construcción tiene automáticamente acceso a los derechos disponibles del host RHEL. Así que puede obtener paquetes RPM de cualquier repositorio habilitado en ese host.

1. **Install RHEL**: Si está listo para empezar, puede comenzar instalando un sistema Red Hat Enterprise Linux.
2. **Register RHEL**: Una vez instalado RHEL, registre el sistema. Se le pedirá que introduzca su nombre de usuario y contraseña. Tenga en cuenta que el nombre de usuario y la contraseña son los mismos que sus credenciales de acceso al Portal del Cliente de Red Hat.

```
# subscription-manager register
Registering to: subscription.rhsm.redhat.com:443/subscription
Username: ********
Password: **********
```

3. **Subscribe RHEL**: Auto-suscripción o determinar el ID del pool de una suscripción que incluye Red Hat Enterprise Linux. Este es un ejemplo de auto-suscripción a una suscripción:

```
# subscription-manager attach --auto
```

4. **Install packages**: Para empezar a construir y trabajar con contenedores individuales, instale el módulo container-tools, que extrae el conjunto completo de paquetes de software para contenedores:

```
# yum module install -y container-tools
```

5. **Install podman-docker (optional)**: Si se siente cómodo con el comando **docker** o utiliza scripts que llaman directamente a **docker**, puede instalar el paquete podman-docker. Este paquete instala un enlace que reemplaza la interfaz de línea de comandos **docker** con los comandos **podman** correspondientes. También enlaza las páginas de manual, por lo que **man docker info** mostrará la página de manual **podman info**.

```
# yum install -y podman-docker
```

1.4. FUNCIONAMIENTO DE LOS CONTENEDORES COMO RAÍZ O SIN RAÍZ

Ejecutar las herramientas de contenedores como **podman**, **skopeo**, o **buildah** como un usuario con

privilegio de superusuario (usuario root) es la mejor manera de asegurar que sus contenedores tengan acceso completo a cualquier característica disponible en su sistema. Sin embargo, con la característica llamada \ "Rootless Containers,\Ndisponible generalmente a partir de RHEL 8.1, puedes trabajar con contenedores como un usuario normal.

Aunque los motores de contenedores, como Docker, permiten ejecutar comandos de **docker** como un usuario normal (no root), el demonio de Docker que lleva a cabo esas peticiones se ejecuta como root. Así que, efectivamente, los usuarios regulares pueden hacer peticiones a través de sus contenedores que dañan el sistema, sin que haya claridad sobre quién hizo esas peticiones. Al establecer usuarios de contenedor sin raíz, los administradores del sistema limitan las actividades potencialmente dañinas de los usuarios regulares, al tiempo que permiten a esos usuarios ejecutar de forma segura muchas características de los contenedores bajo sus propias cuentas.

Esta sección describe cómo configurar tu sistema para utilizar las herramientas de contenedores (Podman, Skopeo y Buildah) para trabajar con contenedores como usuario no root (sin raíz). También describe algunas de las limitaciones que encontrarás porque las cuentas de usuario normales no tienen acceso completo a todas las características del sistema operativo que sus contenedores podrían necesitar para funcionar.

1.4.1. Preparación de los contenedores sin raíces

Es necesario convertirse en usuario root para configurar su sistema RHEL de forma que permita a las cuentas de usuario no root utilizar las herramientas de los contenedores:

1. **Install RHEL**: Instale RHEL 8.1 o actualice a RHEL 8.1 desde RHEL 8.0. Las versiones anteriores de RHEL 7 carecen de las características necesarias para este procedimiento. Si está actualizando desde RHEL 7.6 o una versión anterior, continúe con "Actualizar a contenedores sin raíz" después de realizar este procedimiento.

2. **Install podman and slirp4netns**: Si no está instalado, instale los paquetes podman y slirp4netns:

   ```
   # yum install slirp4netns podman -y
   ```

3. **Increase user namespaces**: Para aumentar el número de espacios de nombres de usuario en el kernel, escriba lo siguiente:

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

4. **Create a new user account** Para crear una nueva cuenta de usuario y añadir una contraseña para esa cuenta (por ejemplo, joe), escriba lo siguiente:

   ```
   # useradd -c "Joe Jones" joe
   # passwd joe
   ```

   El usuario se configura automáticamente para poder utilizar podman sin raíces. En caso de que quiera habilitar un usuario existente para utilizar podman sin raíz, consulte la sección Actualización a contenedores sin raíz.

5. Si desea ejecutar contenedores con systemd, consulte la sección Uso de las imágenes UBI init .

6. **Try a podman command**: Inicie sesión directamente como el usuario que acaba de configurar (no utilice **su** o **su -** para convertirse en ese usuario porque eso no establece las variables de entorno correctas) e intente extraer y ejecutar una imagen:

```
$ podman pull registry.access.redhat.com/ubi8/ubi
$ podman run registry.access.redhat.com/ubi8/ubi cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.1 (Ootpa)"
...
```

7. **Check rootless configuration**: Para comprobar que tu configuración sin raíz está bien configurada, puedes ejecutar comandos dentro del espacio de nombres de usuario modificado con el comando **podman unshare**. Como usuario sin raíz, el siguiente comando le permite ver cómo se asignan los uids al espacio de nombres de usuario:

```
$ podman unshare cat /proc/self/uid_map
         0       1001          1
         1      65537      65536
```

1.4.2. Actualizar a los contenedores sin raíces

Si ha actualizado desde RHEL 7, debe configurar manualmente los valores de subuid y subgid para cualquier usuario existente que desee que pueda utilizar podman sin raíz.

Utilizando un nombre de usuario y un nombre de grupo existentes (por ejemplo, jill), establezca el rango de identificadores de usuario y grupo accesibles que pueden utilizarse para sus contenedores. Aquí hay un par de advertencias:

- No incluya el UID y GID del usuario sin raíz en estos rangos
- Si establece varios usuarios de contenedores sin raíz, utilice rangos únicos para cada usuario
- Recomendamos 65536 UIDs y GIDs para una máxima compatibilidad con las imágenes de contenedores existentes, pero el número puede ser reducido
- No utilice nunca UIDs o GIDs inferiores a 1000 ni reutilice UIDs o GIDs de cuentas de usuario existentes (que, por defecto, comienzan en 1000)
  He aquí un ejemplo:

  ```
  # echo "jill:165537:65536" >> /etc/subuid
  # echo "jill:165537:65536" >> /etc/subgid
  ```

  El usuario/grupo jill tiene ahora asignados 65535 IDs de usuario y grupo, que van desde 165537-231072. Ese usuario debería poder empezar a ejecutar comandos para trabajar con contenedores ahora.

1.4.3. Consideraciones especiales para los desarraigados

Aquí hay algunas cosas a tener en cuenta cuando se ejecutan contenedores como un usuario no root:

- Como usuario no root del contenedor, las imágenes del contenedor se almacenan en su directorio principal (**$HOME/.local/share/containers/storage/**), en lugar de **/var/lib/containers**.
- Los usuarios que ejecutan contenedores sin raíz tienen un permiso especial para ejecutarse como un rango de IDs de usuario y grupo en el sistema anfitrión. Sin embargo, no tienen privilegios de root en el sistema operativo del host.
- Si necesitas configurar tu entorno de contenedores sin raíz, edita los archivos de configuración

en tu directorio de inicio (**$HOME/.config/containers**). Los archivos de configuración incluyen **storage.conf** (para configurar el almacenamiento) y **libpod.conf** (para una variedad de configuraciones de contenedores). También podrías crear un archivo **registries.conf** para identificar los registros de contenedores disponibles cuando uses **podman** para extraer, buscar o ejecutar imágenes.

- Un contenedor que se ejecuta como root en una cuenta sin root puede activar características privilegiadas dentro de su propio espacio de nombres. Pero eso no proporciona ningún privilegio especial para acceder a funciones protegidas en el host (más allá de tener UIDs y GIDs adicionales). Aquí hay ejemplos de acciones del contenedor que se podría esperar que funcionaran desde una cuenta sin raíz y que no funcionarán:
  - Cualquier cosa a la que quieras acceder desde un directorio montado desde el host debe ser accesible por el UID que ejecuta tu contenedor o tu solicitud para acceder a ese componente fallará.
  - Hay algunas características del sistema que no podrás cambiar sin privilegios. Por ejemplo, no puedes cambiar el reloj del sistema simplemente estableciendo una capacidad SYS_TIME dentro de un contenedor y ejecutando el servicio de tiempo de red (ntpd). Tendrías que ejecutar ese contenedor como root, saltándote tu entorno de contenedor sin root y utilizando el entorno del usuario root, para que esa capacidad funcione, como por ejemplo:

    ```
    $ sudo podman run -d --cap-add SYS_TIME ntpd
    ```

    Tenga en cuenta que este ejemplo permite a ntpd ajustar el tiempo para todo el sistema, y no sólo dentro del contenedor.

- Un contenedor sin raíz no tiene capacidad para acceder a un puerto inferior a 1024. Dentro del espacio de nombres del contenedor sin raíz puede, por ejemplo, iniciar un servicio que exponga el puerto 80 de un servicio httpd desde el contenedor, pero no será accesible fuera del espacio de nombres:

  ```
  $ podman run -d httpd
  ```

  Sin embargo, un contenedor necesitaría privilegios de root, de nuevo utilizando el entorno de contenedores del usuario root, para exponer ese puerto al sistema anfitrión:

  ```
  $ sudo podman run -d -p 80:80 httpd
  ```

- El administrador de una estación de trabajo puede configurarla para permitir a los usuarios exponer servicios por debajo de 1024, pero debe entender las implicaciones de seguridad. Un usuario normal podría, por ejemplo, ejecutar un servidor web en el puerto oficial 80 y engañar a los usuarios externos haciéndoles creer que fue configurado por el administrador. Esto generalmente está bien en una estación de trabajo, pero podría no estarlo en un servidor de desarrollo accesible por la red, y definitivamente no debería hacerse en servidores de producción. Para permitir que los usuarios se vinculen a los puertos hasta el puerto 80 ejecute el siguiente comando:

  ```
  # echo 80 > /proc/sys/net/ipv4/ip_unprivileged_port_start
  ```

- Los contenedores sin raíz se basan actualmente en el establecimiento de rangos estáticos de subuid y subgid. Si se utiliza LDAP o Active Directory para proporcionar la autenticación de los usuarios, no hay una forma automatizada de proporcionar esos rangos de UID y GID a los

usuarios. Una solución actual podría ser establecer rangos estáticos en los archivos /etc/subuid y /etc/subgid para que coincidan con los UIDs y GIDs conocidos en uso.

- El almacenamiento del contenedor debe estar en un sistema de archivos local, porque los sistemas de archivos remotos no funcionan bien con los espacios de nombres de los usuarios sin privilegios.

- En Shortcomings of Rootless Podman se encuentra una lista de las deficiencias de ejecutar **podman** y las herramientas relacionadas sin privilegios de root.

CAPÍTULO 2. TRABAJAR CON IMÁGENES DE CONTENEDORES

Las imágenes base de Red Hat Enterprise Linux (RHEL) pueden utilizarse como base para las imágenes de contenedor. Para RHEL 8, todas las imágenes base de Red Hat están disponibles como nuevas imágenes base universales (UBI), lo que significa que puede obtenerlas y redistribuirlas libremente. Estas incluyen versiones de RHEL standard, minimal, init, y Red Hat Software Collections que ahora están disponibles libremente y son redistribuibles. Las imágenes base de RHEL son:

- **Supported**: Soportado por Red Hat para su uso con aplicaciones en contenedores. Contienen los mismos paquetes de software seguros, probados y certificados que se encuentran en Red Hat Enterprise Linux.
- **Cataloged**: Se encuentra en el Catálogo de Contenedores de Red Hat , con descripciones, detalles técnicos y un índice de salud para cada imagen.
- **Updated**: Se ofrece con un calendario de actualizaciones bien definido, para que sepa que está recibiendo el software más reciente (véase Red Hat Container Image Updates).
- **Tracked**: Seguimiento de las erratas para ayudar a entender los cambios que se producen en cada actualización.
- **Reusable**: Las imágenes base deben descargarse y almacenarse en caché en su entorno de producción una vez. Cada imagen base puede ser reutilizada por todos los contenedores que la incluyan como base.

Los UBIs para RHEL 8 proporcionan la misma calidad de software RHEL para la construcción de imágenes de contenedores que sus predecesores no UBIs (**rhel6**, **rhel7**, **rhel-init**, y **rhel-minimal** imágenes de base), pero ofrecen más libertad en cuanto a su uso y distribución.

Para RHEL 8, están disponibles las imágenes base estándar, mínima e init. Red Hat también proporciona un conjunto de imágenes de tiempo de ejecución de lenguajes, basadas en Application Streams, sobre las que se puede construir cuando se crean contenedores para aplicaciones que requieren tiempos de ejecución específicos. Las imágenes de tiempo de ejecución incluyen python, php, ruby, nodejs y otros.

También hay un conjunto de imágenes de RHEL 7 que se pueden ejecutar en sistemas RHEL 8. Para RHEL 7, hay imágenes base UBI (redistribuibles) y no UBI (requieren acceso de suscripción y no son redistribuibles). Estas imágenes incluyen tres imágenes base normales (**rhel7**, **rhel-init**, y **rhel-minimal**) y tres imágenes UBI (**ubi7**, **ubi7-init**, y **ubi7-minimal**).

Aunque Red Hat no ofrece herramientas para ejecutar contenedores en sistemas RHEL 6, sí ofrece imágenes de contenedores RHEL 6 que puede utilizar. Hay imágenes base estándar (**rhel6**) e init (**rhel6-init**) disponibles para RHEL 6, pero no hay una imagen mínima de RHEL 6. Asimismo, no hay imágenes RHEL 6 UBI.

Aunque las imágenes base heredadas de RHEL 7 seguirán siendo soportadas, se recomiendan las imágenes UBI de cara al futuro. Por esta razón, los ejemplos en el resto de este capítulo se realizan con imágenes UBI de RHEL 8. Para una lista de imágenes UBI de Red Hat disponibles, e información asociada sobre los repositorios UBI y el código fuente, vea el artículo Imágenes base universales (UBI): Imágenes , repositorios y paquetes.

2.1. DIFERENCIAS ENTRE LAS IMÁGENES RHEL Y LAS IMÁGENES UBI

Las imágenes UBI fueron creadas para que usted pueda construir sus imágenes de contenedor sobre una base de software oficial de Red Hat que pueda ser compartida e implementada libremente. Desde

una perspectiva técnica, son casi idénticas a las imágenes heredadas de Red Hat Enterprise Linux, lo que significa que tienen una gran seguridad, rendimiento y ciclos de vida, pero se publican bajo un acuerdo de licencia de usuario final diferente. Estos son algunos atributos de las imágenes Red Hat UBI:

- **Built from a subset of RHEL content** Las imágenes de Red Hat Universal Base se construyen a partir de un subconjunto del contenido normal de Red Hat Enterprise Linux. Todo el contenido utilizado para construir las imágenes UBI seleccionadas se libera en un conjunto de repositorios yum disponibles públicamente. Esto le permite instalar paquetes adicionales, así como actualizar cualquier paquete en las imágenes base UBI.

- **Redistributable**: La intención de las imágenes UBI es permitir a los clientes de Red Hat, a los socios, a los ISVs y a otros estandarizar las imágenes UBI para que puedan construir sus imágenes de contenedor sobre una base de software oficial de Red Hat que pueda ser compartido e implementado libremente. Desde una perspectiva técnica, son casi idénticas a las imágenes de Red Hat Enterprise Linux heredadas, lo que significa que tienen una gran seguridad, rendimiento y ciclos de vida, pero se publican bajo un Acuerdo de Licencia de Usuario Final diferente.

- **Base and runtime images** Además de los tres tipos de imágenes base, también están disponibles versiones UBI de varias imágenes de tiempo de ejecución. Estas imágenes de tiempo de ejecución proporcionan una base para las aplicaciones que pueden beneficiarse de los tiempos de ejecución estándar soportados, como python, php, nodejs y ruby.

- **Enabled yum repositories**: Los siguientes repositorios yum están habilitados en cada imagen de RHEL 8 UBI:

  - El repositorio **ubi-8-baseos** contiene el subconjunto redistribuible de paquetes RHEL que puede incluir en su contenedor.

  - El repositorio **ubi-8-appstream** contiene paquetes de flujos de aplicaciones que puede añadir a una imagen UBI para ayudarle a estandarizar los entornos que utiliza con aplicaciones que requieren tiempos de ejecución particulares.

- **Licensing**: Usted es libre de utilizar y redistribuir las imágenes UBI, siempre que se adhiera al Acuerdo de Licencia de Usuario Final de Red Hat Universal Base Image .

- **Adding UBI RPMs**: Puede añadir paquetes RPM a las imágenes UBI desde los repositorios UBI preconfigurados. Si se encuentra en un entorno desconectado, debe permitir la red de entrega de contenidos UBI(https://cdn-ubi.redhat.com) para utilizar esta función. Consulte la solución Connect to https://cdn-ubi.redhat.com para obtener más detalles.

2.2. COMPRENSIÓN DE LAS IMÁGENES BASE ESTÁNDAR DE RED HAT

Las imágenes base de RHEL 8 estándar (**ubi8**) tienen un robusto conjunto de características de software que incluyen lo siguiente:

- **init system**: Todas las características del sistema de inicialización systemd que necesitas para gestionar los servicios systemd están disponibles en las imágenes base estándar. Estos sistemas de init le permiten instalar paquetes RPM preconfigurados para iniciar servicios automáticamente, como un servidor web (**httpd**) o un servidor FTP (**vsftpd**).

- **yum**: El software necesario para instalar paquetes de software se incluye a través del conjunto estándar de comandos **yum** (**yum**, **yum-config-manager**, **yumdownloader**, y así sucesivamente). Para las imágenes base de UBI, tienes acceso a los repositorios gratuitos de yum para añadir y actualizar software.

- **utilities**: La imagen base estándar incluye algunas utilidades para trabajar dentro del contenedor. Las utilidades que están en esta imagen base y que no están en las imágenes mínimas incluyen **tar**, **dmidecode**, **gzip**, **getfacl** (y otros comandos acl), **dmsetup** (y otros comandos de mapeo de dispositivos), y otros.

2.3. COMPRENSIÓN DE LAS IMÁGENES BASE DE RED HAT MÍNIMAS

Las imágenes de **ubi8-minimal** son imágenes de RHEL despojadas para usar cuando se desea una imagen base básica. Si está buscando una imagen base lo más pequeña posible para usarla como parte del ecosistema de Red Hat, puede empezar con estas imágenes mínimas.

Las imágenes mínimas de RHEL proporcionan una base para sus propias imágenes de contenedor que tiene menos de la mitad del tamaño de la imagen estándar, al tiempo que puede recurrir a los repositorios de software de RHEL y mantener cualquier requisito de cumplimiento que tenga su software.

Estas son algunas de las características de las imágenes base mínimas:

- **Small size**: Las imágenes mínimas ocupan unos 92M en disco y 32M comprimidas. Esto hace que tenga menos de la mitad del tamaño de las imágenes estándar.

- **Software installation (microdnf)**: En lugar de incluir la instalación completa **yum** para trabajar con repositorios de software y paquetes de software RPM, las imágenes mínimas incluyen la utilidad **microdnf**. **microdnf** es una versión reducida de **dnf**. Incluye sólo lo necesario para activar y desactivar repositorios, así como para instalar, eliminar y actualizar paquetes. También tiene una opción de limpieza, para limpiar la caché después de instalar los paquetes.

- **Based on RHEL packaging**: Debido a que las imágenes mínimas incorporan paquetes RPM de software RHEL normales, con algunas características eliminadas, como archivos de idioma adicionales o documentación, puede seguir confiando en los repositorios de RHEL para construir sus imágenes. Esto le permite seguir manteniendo los requisitos de cumplimiento que tiene basados en el software RHEL. Las características de las imágenes mínimas las hacen perfectas para probar las aplicaciones que desea ejecutar con RHEL, al tiempo que conllevan la menor cantidad posible de gastos generales. Lo que no se obtiene con las imágenes mínimas es un sistema de inicialización y gestión de servicios (systemd o System V init), un entorno de ejecución de Python y un montón de utilidades de shell comunes.

- **Modules for microdnf are not supported**: Los módulos utilizados con el comando **dnf** le permiten instalar varias versiones del mismo software, cuando están disponibles. La utilidad **microdnf** incluida en las imágenes mínimas no admite módulos. Por lo tanto, si se necesitan módulos, se debe utilizar una imagen base no mínima, que incluya **yum**.

Sin embargo, si su objetivo es sólo tratar de ejecutar algunos binarios simples o software pre-empaquetado que no tiene muchos requisitos del sistema operativo, las imágenes mínimas podrían satisfacer sus necesidades. Si su aplicación tiene dependencias de otro software de RHEL, puede utilizar **microdnf** para instalar los paquetes necesarios en el momento de la compilación.

Red Hat tiene la intención de que usted utilice siempre la última versión de las imágenes mínimas, lo cual está implícito al solicitar **ubi8**/**ubi-minimal** o **ubi8-minimal**. Red Hat no espera dar soporte a versiones anteriores de las imágenes mínimas en el futuro.

2.4. COMPRENSIÓN DE LAS IMÁGENES BASE INIT DE RED HAT

Las imágenes UBI **ubi8-init** contienen el sistema de inicialización de systemd, lo que las hace útiles para construir imágenes en las que se quieren ejecutar servicios de systemd, como un servidor web o un servidor de archivos. El contenido de la imagen init es menor que el que se obtiene con las imágenes

estándar, pero mayor que el de las imágenes mínimas.

NOTA

Dado que la imagen **ubi8-init** se construye sobre la imagen **ubi8**, su contenido es prácticamente el mismo. Sin embargo, hay algunas diferencias críticas. En **ubi8-init**, el Cmd se establece en **/sbin/init**, en lugar de **bash**, para iniciar el servicio systemd Init por defecto. Incluye **ps** y los comandos relacionados con el proceso (paquete **procps-ng**), cosa que ubi8 no hace. También, **ubi8-init** establece **SIGRTMIN 3** como el **StopSignal**, ya que systemd en **ubi8-init** ignora las señales normales de salida (**SIGTERM** y **SIGKILL**), pero terminará si recibe **SIGRTMIN 3**.

Históricamente, las imágenes de contenedor base de Red Hat Enterprise Linux fueron diseñadas para que los clientes de Red Hat ejecutaran aplicaciones empresariales, pero no eran libres de redistribuir. Esto puede crear desafíos para algunas organizaciones que necesitan redistribuir sus aplicaciones. Ahí es donde entran las imágenes base universales de Red Hat.

2.5. UTILIZACIÓN DE LAS IMÁGENES UBI INIT

Este procedimiento muestra cómo construir un contenedor usando un Dockerfile que instala y configura un servidor web (**httpd**) para que se inicie automáticamente por el servicio systemd (**/sbin/init**) cuando el contenedor se ejecuta en un sistema anfitrión.

Procedimiento

1. Cree un Dockerfile con el siguiente contenido en un nuevo directorio:

```
FROM registry.access.redhat.com/ubi8/ubi-init
RUN yum -y install httpd; yum clean all; systemctl enable httpd;
RUN echo "Successful Web Server Test" > /var/www/html/index.html
RUN mkdir /etc/systemd/system/httpd.service.d/; echo -e '[Service]\nRestart=always' >
/etc/systemd/system/httpd.service.d/httpd.conf
EXPOSE 80
CMD [ "/sbin/init" ]
```

   El Dockerfile instala el paquete **httpd**, habilita el servicio **httpd** para que se inicie en el momento del arranque, crea un archivo de prueba (**index.html**), expone el servidor web al host (puerto 80), e inicia el servicio systemd init (**/sbin/init**) cuando se inicia el contenedor.

2. Construye el contenedor:

```
# podman build --format=docker -t mysysd .
```

3. Opcionalmente, si quieres ejecutar contenedores con systemd y SELinux está habilitado en tu sistema, debes establecer la variable booleana **container_manage_cgroup**:

```
# setsebool -P container_manage_cgroup 1
```

4. Ejecute el contenedor llamado **mysysd_run**:

```
# podman run -d --name=mysysd_run -p 80:80 mysysd
```

La imagen **mysysd** se ejecuta como el contenedor **mysysd_run** como un proceso demonio, con el puerto 80 del contenedor expuesto al puerto 80 en el sistema anfitrión.

5. Comprueba que el contenedor está en marcha:

```
# podman ps
a282b0c2ad3d  localhost/mysysd:latest  /sbin/init  15 seconds ago  Up 14 seconds ago
0.0.0.0:80->80/tcp  mysysd_run
```

6. Prueba el servidor web:

```
# curl localhost/index.html
Successful Web Server Test
```

2.6. REDISTRIBUCIÓN DE IMÁGENES UBI

Este procedimiento describe cómo redistribuir las imágenes UBI. Después de extraer una imagen UBI, es libre de enviar una imagen UBI a su propio registro o al de un tercero y compartirla con otros. Puede actualizar o añadir a esa imagen desde los repositorios yum de UBI como desee.

Procedimiento

1. Para extraer la imagen de **ubi** del registro de registry.redhat.io, introduzca:

```
# podman pull registry.redhat.io/ubi8/ubi
```

2. Para añadir un nombre adicional a la imagen **ubi**, introduzca:

```
# podman tag registry.redhat.io/ubi8/ubi registry.example.com:5000/ubi8/ubi
```

3. Para empujar la imagen **ubi** desde su almacenamiento local a un registro, introduzca:

```
# podman push registry.example.com:5000/ubi8/ubi
```

Si bien hay pocas restricciones en la forma de utilizar estas imágenes, hay algunas restricciones sobre cómo puede referirse a ellas. Por ejemplo, no puede llamar a esas imágenes certificadas por Red Hat o soportadas por Red Hat a menos que lo certifique a través del programa Red Hat Partner Connect , ya sea con la certificación Red Hat Container o la certificación Red Hat OpenShift Operator.

2.7. BÚSQUEDA DE IMÁGENES DE CONTENEDORES

El comando **podman search** le permite buscar imágenes en los registros de contenedores seleccionados.

NOTA

También puede buscar imágenes en el Red Hat Container Registry. El Registro de Contenedores de Red Hat incluye la descripción de la imagen, el contenido, el índice de salud y otra información.

Puede encontrar la lista de registros en el archivo de configuración **registries.conf**:

```
[registries.search]
registries = ['registry.access.redhat.com', 'registry.redhat.io', 'docker.io']

[registries.insecure]
registries = []

[registries.block]
registries = []
```

- Por defecto, el comando **podman search** busca imágenes de contenedores en los registros listados en la sección **[registries.search]** en el orden dado. En este caso, el comando **podman search** busca la imagen solicitada en registry.access.redhat.com, registry.redhat.io y docker.io en este orden.
- La sección **[registries.insecure]** añade los registros que no utilizan TLS (un registro inseguro).
- La sección **[registries.block]** impide el acceso al registro desde su sistema local.

Como usuario root, puedes editar el archivo **/etc/containers/registries.conf** para cambiar la configuración de búsqueda por defecto en todo el sistema.

Como usuario normal (sin raíz) de **podman**, puede crear su propio archivo **registries.conf** en su directorio personal (**$HOME/.config/containers/registries.conf**) para anular la configuración de todo el sistema.

Asegúrese de seguir las condiciones al configurar los registros de los contenedores:

- Cada registro debe ir rodeado de comillas simples.
- Si hay varios registros configurados para el **registries = value**, debe separar esos registros con comas.
- Puede identificar los registros por su dirección IP o por su nombre de host.
- Si el registro utiliza un puerto no estándar, distinto de los puertos TCP 443 para el seguro y 80 para el inseguro, introduzca ese número de puerto con el nombre del registro. Por ejemplo: host.example.com:9999.
- El sistema busca los registros en el orden en que aparecen en la lista **registries.search** del archivo **registries.conf**.

A continuación, algunos ejemplos de comandos de **podman search**. El primer ejemplo ilustra la búsqueda infructuosa de todas las imágenes de quay.io. La barra diagonal al final significa buscar en todo el registro todas las imágenes accesibles para usted:

```
# podman search quay.io/
ERRO[0000] error searching registry "quay.io": couldn't search registry "quay.io":
unable to retrieve auth token: invalid username/password
```

Para buscar en el registro de quay.io, inicie sesión primero:

```
# podman login quay.io
Username: johndoe
Password: ***********
Login Succeeded!
# podman search quay.io/
```

```
INDEX    NAME                          DESCRIPTION   STARS   OFFICIAL   AUTOMATED
quay.io  quay.io/test/myquay                         0
quay.io  quay.io/test/redistest                      0
quay.io  quay.io/johndoe/websrv21                    0
quay.io  quay.io/johndoe/mydbtest                    0
quay.io  quay.io/johndoe/newbuild-10                 0
```

Buscar en todos los registros disponibles las imágenes de **postgresql** (se han encontrado más de 40 imágenes):

```
# podman search postgresql-10
INDEX      NAME                                DESCRIPTION                STARS OFFICIAL
AUTOMATED
redhat.io  registry.redhat.io/rhel8/postgresql-10        This container image ...      0
redhat.io  registry.redhat.io/rhscl/postgresql-10-rhel7   PostgreSQL is an advanced ...  0
quay.io    quay.io/mettle/postgresql-database-provisioning
docker.io  docker.io/centos/postgresql-10-centos7        PostgreSQL is an advanced ... 13
...
```

Para limitar la búsqueda de **postgresql** a las imágenes de registry.redhat.io, escriba el siguiente comando. Tenga en cuenta que al introducir el registro y el nombre de la imagen, se puede hacer coincidir cualquier repositorio del registro:

```
# podman search registry.redhat.io/postgresql-10
INDEX      NAME                                DESCRIPTION          STARS   OFFICIAL
AUTOMATED
redhat.io  registry.redhat.io/rhel8/postgresql-10        This container image ...  0
redhat.io  registry.redhat.io/rhscl/postgresql-10-rhel7   PostgreSQL is an  ...     0
```

Para obtener descripciones más largas para cada imagen de contenedor, añada **--no-trunc** al comando:

```
# podman search --no-trunc registry.redhat.io/rhel8/postgresql-10
INDEX      NAME   DESCRIPTION              STARS   OFFICIAL   AUTOMATED
redhat.io  registry.redhat.io/rhel8/postgresql-10
             This container image provides a containerized
             packaging of the PostgreSQL postgres daemon and
             client application. The postgres server daemon
             accepts connections from clients and provides
             access to content from PostgreSQL databases on
             behalf of the clients.   0
```

Para acceder a registros inseguros, añada el nombre completo del registro en la sección **[registries.insecure]** del archivo **/etc/containers/registries.conf**. Por ejemplo:

```
[registries.search]
registries = ['myregistry.example.com']

[registries.insecure]
registries = ['myregistry.example.com']
```

A continuación, busque las imágenes de **myimage**:

```
# podman search myregistry.example.com/myimage
INDEX      NAME  DESCRIPTION                                STARS  OFFICIAL  AUTOMATED
```

```
example.com  myregistry.example.com/myimage
  The myimage container executes the ...   0
```

Ahora puedes sacar la imagen de **myimage**:

```
# podman pull myimage.example.com/myimage
```

2.8. DEFINICIÓN DE LA POLÍTICA DE VERIFICACIÓN DE LA FIRMA DE LA IMAGEN

Red Hat entrega firmas para las imágenes en el Registro de Contenedores de Red Hat. Cuando se ejecuta como root, **/etc/containers/policy.json**, y los archivos YAML en el directorio **/etc/containers/registries.d/** definen la política de verificación de firmas. La política de confianza en **/etc/containers/policy.json** describe un ámbito de registro (registro y o repositorio) para la confianza.

Por defecto, la herramienta del contenedor lee la política de **$HOME/.config/containers/policy.json**, si existe, de lo contrario de **/etc/containers/policy.json**.

La confianza se define mediante tres parámetros:

1. El nombre *registry* o *registry/repository*
2. Una o varias claves GPG públicas
3. Un servidor de firmas

Red Hat sirve firmas desde estos URIs:

```
https://access.redhat.com/webassets/docker/content/sigstore
https://registry.redhat.io/containers/sigstore
```

Procedimiento

1. Muestra el archivo **/etc/containers/policy.json**:

```
# cat /etc/containers/policy.json
{
    "default": [
        {
            "type": "insecureAcceptAnything"
        }
    ],
    "transports":
    {
        "docker-daemon":
        {
            "": [{"type":"insecureAcceptAnything"}]
        }
    }
}
```

2. Para actualizar un ámbito de confianza existente para los registros registry.access.redhat.com y registry.redhat.io, introduzca

```
# podman image trust set -f /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
registry.access.redhat.com
# podman image trust set -f /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
registry.redhat.io
```

3. Para verificar la configuración de la política de confianza, visualice el archivo **/etc/containers/policy.json**:

```
    "docker": {
        "registry.access.redhat.com": [
            {
                "type": "signedBy",
                "keyType": "GPGKeys",
                "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
            }
        ],
        "registry.redhat.io": [
            {
                "type": "signedBy",
                "keyType": "GPGKeys",
                "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
            }
        ]
    },
```

Puede ver que se han añadido las secciones **"registry.access.redhat.com"** y **"registry.redhat.io"**.

4. Cree el archivo **/etc/containers/registries.d/registry.access.redhat.com.yaml** para identificar el almacén de firmas para las imágenes de contenedores desde el registro de registry.access.redhat.com:

```
docker:
    registry.access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

5. Cree el archivo **etc/containers/registries.d/registry.redhat.io.yaml** con el siguiente contenido:

```
docker:
    registry.redhat.io:
        sigstore: https://registry.redhat.io/containers/sigstore
```

6. Para mostrar la configuración de la confianza, introduzca:

```
# podman image trust show                                    default                 accept
registry.access.redhat.com  signedBy              security@redhat.com, security@redhat.com
https://access.redhat.com/webassets/docker/content/sigstore
registry.redhat.io          signedBy              security@redhat.com, security@redhat.com
https://registry.redhat.io/containers/sigstore
                            insecureAcceptAnything
```

7. Para rechazar la política de confianza por defecto, escriba:

```
# podman image trust set -t reject default
```

8. Para verificar la configuración de la política de confianza, muestre la página **/etc/containers/policy.json**:

```
# cat /etc/containers/policy.json
{
    "default": [
    {
        "type": "reject"
    }
...
}
```

Puede ver que la sección **"default"** ha cambiado de **"insecureAcceptAnything"** a **"reject"**.

9. Intente extraer la imagen mínima de Red Hat Universal Base Image 8 (**ubi8-minimal**) del registro de registry.access.redhat.com:

```
# podman --log-level=debug pull registry.access.redhat.com/ubi8-minimal
....
DEBU[0000] Using registries.d directory /etc/containers/registries.d for sigstore configuration
DEBU[0000]  Using "docker" namespace registry.access.redhat.com
DEBU[0000]   Using https://access.redhat.com/webassets/docker/content/sigstore
...
```

Verá que la dirección de almacenamiento de la firma **access.redhat.com/webassets/docker/content/sigstore** coincide con la dirección especificada en **/etc/containers/registries.d/registry.access.redhat.com.yaml**.

10. Inicie sesión en el registro de registry.redhat.io:

```
# podman login registry.redhat.io
Username: username
Password: ***********
Login Succeeded!
```

11. Intente extraer la imagen **support-tools** del registro de registry.redhat.io:

```
# podman --log-level=debug pull registry.redhat.io/rhel8/support-tools
...
DEBU[0000] Using registries.d directory /etc/containers/registries.d for sigstore configuration
DEBU[0000]  Using "docker" namespace registry.redhat.io
DEBU[0000]   Using https://registry.redhat.io/containers/sigstore
...
```

Puede ver que la dirección de almacenamiento de la firma **registry.redhat.io/containers/sigstore** coincide con la dirección especificada en **/etc/containers/registries.d/registry.redhat.io.yaml**.

12. Para listar todas las imágenes que se han extraído de su sistema local, introduzca:

```
# podman images
REPOSITORY                          TAG     IMAGE ID      CREATED       SIZE
registry.redhat.io/rhel8/support-tools   latest  5ef2aab09451  13 days ago   254 MB
registry.access.redhat.com/ubi8-minimal  latest  86c870596572  13 days ago   146 MB
```

Recursos adicionales

- Para más información sobre el **podman image trust**, escriba **man podman-image-trust**.
- Para más información sobre la verificación de imágenes de contenedores, véase el artículo Verificación de la firma de imágenes para Red Hat Container Registry .

2.9. EXTRACCIÓN DE IMÁGENES DE LOS REGISTROS

Para obtener imágenes de contenedores de un registro remoto (como el propio registro de contenedores de Red Hat) y añadirlas a su sistema local, utilice el comando **podman pull**:

```
# podman pull <registry>[:<port>]/[<namespace>/]<name>:<tag>
```

El *<registry>* es un host que proporciona un servicio de registro de contenedores en TCP *<port>*. Juntos, *<namespace>* y *<name>* identifican una imagen particular controlada por *<namespace>* en ese registro. El *<tag>* es un nombre adicional a la imagen almacenada localmente, la etiqueta por defecto es **latest**. Utilice siempre nombres de imagen totalmente cualificados que incluyan: registro, espacio de nombres, nombre de la imagen y etiqueta. Cuando se utilizan nombres cortos, siempre hay un riesgo inherente de suplantación de identidad. Añada registros de confianza, es decir, registros que no permitan a usuarios desconocidos o anónimos crear cuentas con nombres arbitrarios.

Algunos registros también admiten *<name>* en bruto; para ellos, *<namespace>* es opcional. Sin embargo, cuando se incluye, el nivel adicional de jerarquía que proporciona *<namespace>* es útil para distinguir entre imágenes con el mismo *<name>*. Por ejemplo:

Espacio de nombres	Ejemplos (*<namespace>*/*<name>*)
organización	**redhat/kubernetes**, **google/kubernetes**
login (nombre de usuario)	**alice/application**, **bob/application**
papel	**devel/database**, **test/database**, **prod/database**

Los registros que Red Hat proporciona son registry.redhat.io (requiere autenticación), registry.access.redhat.com (no requiere autenticación) y registry.connect.redhat.com (contiene imágenes del programa Red Hat Partner Connect). Para más detalles sobre la transición a registry.redhat.io, consulte Autenticación del Registro de Contenedores de Red Hat. Antes de poder extraer contenedores de registry.redhat.io, es necesario autenticarse. Por ejemplo:

```
# podman login registry.redhat.io
Username: myusername
Password: ************
Login Succeeded!
```

Utilice la opción pull para extraer una imagen de un registro remoto. Para extraer la imagen base de RHEL **ubi** y la imagen de registro **rsyslog** del registro de Red Hat, escriba:

```
# podman pull registry.redhat.io/ubi8/ubi
# podman pull registry.redhat.io/rhel8/rsyslog
```

Una imagen se identifica con un nombre de registro (registry.redhat.io), un nombre de espacio de

nombres (ubi8) y el nombre de la imagen (ubi). También se puede añadir una etiqueta (que por defecto es :latest si no se introduce). El nombre del repositorio **ubi**, cuando se pasa al comando **podman pull** sin el nombre de un registro que lo precede, es ambiguo y podría resultar en la recuperación de una imagen que se origina en un registro no confiable. Si hay varias versiones de la misma imagen, añadir una etiqueta, como **latest** para formar un nombre como **ubi8/ubi:latest**, permite elegir la imagen de forma más explícita.

Para ver las imágenes resultantes del comando **podman pull** anterior, junto con cualquier otra imagen de su sistema, escriba **podman images**:

```
REPOSITORY                      TAG    IMAGE ID      CREATED     SIZE
registry.redhat.io/ubi8/ubi     latest eb205f07ce7d  2 weeks ago 214MB
registry.redhat.io/rhel8/rsyslog  latest 85cfba5cd49c  2 weeks ago 234MB
```

Las imágenes **ubi** y **rsyslog** ya están disponibles en su sistema local para que pueda trabajar con ellas.

2.10. LISTADO DE IMÁGENES

Puede utilizar el comando **podman images** para ver qué imágenes han sido extraídas a su sistema local y están disponibles para su uso.

Procedimiento

- Para listar las imágenes en el almacenamiento local, introduzca:

```
# podman images
REPOSITORY                                TAG      IMAGE ID     CREATED      VIRTUAL SIZE
registry.redhat.io/rhel8/support-tools    latest   b3d6ce4e0043 2 days ago   234MB
registry.redhat.io/ubi8/ubi-init          latest   779a05997856 2 days ago   225MB
registry.redhat.io/ubi8/ubi               latest   a80dad1c1953 3 days ago   210MB
```

2.11. INSPECCIÓN DE IMÁGENES LOCALES

Después de extraer una imagen a su sistema local y antes de ejecutarla, es una buena idea investigar esa imagen. Las razones para investigar una imagen antes de ejecutarla incluyen:

- Entender lo que hace la imagen
- Comprobar qué software hay dentro de la imagen

Procedimiento

El comando **podman inspect** muestra información básica sobre lo que hace una imagen. También tiene la opción de montar la imagen en su sistema anfitrión y utilizar las herramientas del anfitrión para investigar lo que hay en la imagen. Este es un ejemplo de cómo investigar lo que hace una imagen de contenedor antes de ejecutarla.

1. **Inspect an image**: Ejecute **podman inspect** para ver qué comando se ejecuta cuando se ejecuta la imagen del contenedor, así como otra información. Aquí hay ejemplos de examen de las imágenes de contenedor ubi8/ubi y rhel8/rsyslog (con sólo fragmentos de información mostrados aquí):

```
# podman pull registry.redhat.io/ubi8/ubi
# podman inspect registry.redhat.io/ubi8/ubi | less
```

```
...
  "Cmd": [
      "/bin/bash"
  ],
  "Labels": {
      "License": "GPLv3",
      "architecture": "x86_64",
      "authoritative-source-url": "registry.redhat.io",
      "build-date": "2018-10-24T16:46:08.916139",
      "com.redhat.build-host": "cpt-0009.osbs.prod.upshift.rdu2.redhat.com",
      "com.redhat.component": "rhel-server-container",
      "description": "The Red Hat Enterprise Linux Base image is designed to be a fully
supported...
...
```

```
# podman pull registry.redhat.io/rhel8/rsyslog
# podman inspect registry.redhat.io/rhel8/rsyslog
  "Cmd": [
    "/bin/rsyslog.sh"
  ],
  "Labels": {
    "License": "GPLv3",
    "architecture": "x86_64",
...
    "install": "podman run --rm --privileged -v /:/host -e HOST=/host -e IMAGE=IMAGE -e
NAME=NAME IMAGE /bin/install.sh",
...
    "run": "podman run -d --privileged --name NAME --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -e IMAGE=IMAGE -e NAME=NAME --restart=always IMAGE
/bin/rsyslog.sh",
    "summary": "A containerized version of the rsyslog utility
...
```

El contenedor ubi8/ubi ejecutará el shell bash, si no se da ningún otro argumento al iniciarlo con **podman run**. Si se estableciera un Entrypoint, su valor se utilizaría en lugar del valor de Cmd (y el valor de Cmd se utilizaría como argumento del comando Entrypoint).

En el segundo ejemplo, la imagen del contenedor rhel8/rsyslog tiene incorporadas las etiquetas **install** y **run**. Estas etiquetas indican cómo debe configurarse el contenedor en el sistema (instalar) y ejecutarse (ejecutar).

2. **Mount a container**: Usando el comando **podman**, monta un contenedor activo para investigar más a fondo su contenido. Este ejemplo ejecuta y enumera un contenedor **rsyslog** en ejecución, y luego muestra el punto de montaje desde el que se puede examinar el contenido de su sistema de archivos:

```
# podman run -d registry.redhat.io/rhel8/rsyslog
# podman ps
CONTAINER ID IMAGE          COMMAND       CREATED       STATUS     PORTS
NAMES
1cc92aea398d ...rsyslog:latest /bin/rsyslog.sh 37 minutes ago Up 1 day ago     myrsyslog
# podman mount 1cc92aea398d
/var/lib/containers/storage/overlay/65881e78.../merged
```

```
# ls /var/lib/containers/storage/overlay/65881e78*/merged
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr
var
```

Después del **podman mount**, el contenido del contenedor es accesible desde el directorio listado en el host. Utilice **ls** para explorar el contenido de la imagen.

3. **Check the image's package list** Para comprobar los paquetes instalados en el contenedor, indique al comando **rpm** que examine los paquetes instalados en el punto de montaje del contenedor:

```
# rpm -qa --root=/var/lib/containers/storage/overlay/65881e78.../merged
redhat-release-server-7.6-4.el7.x86_64
filesystem-3.2-25.el7.x86_64
basesystem-10.0-7.el7.noarch
ncurses-base-5.9-14.20130511.el7_4.noarch
glibc-common-2.17-260.el7.x86_64
nspr-4.19.0-1.el7_5.x86_64
libstdc++-4.8.5-36.el7.x86_64
```

2.12. INSPECCIÓN DE IMÁGENES REMOTAS

Para inspeccionar una imagen de contenedor antes de llevarla a su sistema, puede utilizar el comando **skopeo inspect**. Con **skopeo inspect**, puede mostrar información sobre una imagen que reside en un registro de contenedores remoto.

Procedimiento

El siguiente comando inspecciona la imagen **ubi8-init** desde el registro de Red Hat.

- Para inspeccionar el **ubi8-init** del registro de registry.redhat.io, introduzca:

```
# skopeo inspect docker://registry.redhat.io/ubi8/ubi-init
{
    "Name": "registry.redhat.io/ubi8/ubi8-init",
    "Digest": "sha256:53dfe24...",
    "RepoTags": [
        "8.0.0-9",
        "8.0.0",
        "latest"
    ],
    "Created": "2019-05-13T20:50:11.437931Z",
    "DockerVersion": "1.13.1",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.access.redhat.com",
        "build-date": "2019-05-13T20:49:44.207967",
        "com.redhat.build-host": "cpt-0013.osbs.prod.upshift.rdu2.redhat.com",
        "com.redhat.component": "ubi8-init-container",
        "description": "The Red Hat Enterprise Linux Init image is designed to be...
```

2.13. ETIQUETADO DE IMÁGENES

Puede añadir nombres a las imágenes para que sea más intuitivo entender lo que contienen. El

etiquetado de imágenes también puede utilizarse para identificar el registro de destino al que está destinada la imagen. Utilizando el comando **podman tag**, esencialmente se añade un alias a la imagen que puede constar de varias partes. Esas partes pueden incluir:

registryhost/username/NAME:tag

Si lo desea, puede añadir sólo *NAME*. Por ejemplo:

```
# podman tag 474ff279782b myrhel8
```

En el ejemplo anterior, la imagen **rhel8** tenía un ID de imagen de 474ff279782b. Usando **podman tag**, el nombre **myrhel8** ahora también está unido al ID de la imagen. Así que podría ejecutar este contenedor por nombre (rhel8 o myrhel8) o por ID de imagen. Observe que sin añadir una :etiqueta al nombre, se le asignó :latest como etiqueta. Podría haber establecido la etiqueta a 8.0 de la siguiente manera:

```
# podman tag 474ff279782b myrhel8:8.0
```

Al principio del nombre, puedes añadir opcionalmente un nombre de usuario y/o un nombre de registro. El nombre de usuario es en realidad el repositorio en Docker.io que se relaciona con la cuenta de usuario que posee el repositorio. El etiquetado de una imagen con un nombre de registro se mostró en la sección "Etiquetado de imágenes" anteriormente en este documento. Este es un ejemplo de cómo añadir un nombre de usuario:

```
# podman tag 474ff279782b jsmith/myrhel8
# podman images | grep 474ff279782b
rhel8          latest  474ff279782b  7 days ago  139.6 MB
myrhel8        latest  474ff279782b  7 months ago  139.6 MB
myrhel8        7.1     474ff279782b  7 months ago  139.6 MB
jsmith/myrhel8  latest  474ff279782b  7 months ago  139.6 MB
```

Arriba, puede ver todos los nombres de las imágenes asignados al ID de la imagen única.

2.14. GUARDAR Y CARGAR IMÁGENES

Si quieres guardar una imagen de contenedor que has almacenado localmente, puedes utilizar **podman save** para guardar la imagen en un archivo o directorio y restaurarla más tarde en otro entorno de contenedores. El archivo que guardes puede estar en cualquiera de los diferentes formatos de imagen de contenedor: docker-archive, oci-archive, oci-dir (directorio con tipo de manifiesto oci), o docker-dir (directorio con tipo de manifiesto v2s2). Después de guardar una imagen, puedes almacenarla o enviarla a otra persona, y luego **load** la imagen más tarde para reutilizarla. Este es un ejemplo de cómo guardar una imagen como un tarball en el formato predeterminado de docker-archive:

```
# podman save -o myrsyslog.tar registry.redhat.io/rhel8/rsyslog:latest
# file myrsyslog.tar
myrsyslog.tar: POSIX tar archive
```

El archivo **myrsyslog.tar** se almacena ahora en su directorio actual. Más tarde, cuando esté listo para reutilizar el tarball como imagen de contenedor, puede importarlo a otro entorno podman de la siguiente manera:

```
# podman load -i myrsyslog.tar
# podman images
REPOSITORY                    TAG    IMAGE ID      CREATED     SIZE
registry.redhat.io/rhel8/rsyslog latest 1f5313131bf0  7 weeks ago 235 MB
```

-

En lugar de utilizar **save** y **load** para almacenar y recargar una imagen, puede hacer una copia de un contenedor utilizando **podman export** y **podman import**.

2.15. ELIMINACIÓN DE IMÁGENES

Para ver una lista de las imágenes que hay en su sistema, ejecute el comando **podman images**. Para eliminar las imágenes que ya no necesita, utilice el comando **podman rmi**, con el ID o el nombre de la imagen como opción. (Debes detener cualquier contenedor que se ejecute desde una imagen antes de poder eliminarla) Este es un ejemplo:

```
# podman rmi ubi8-init
7e85c34f126351ccb9d24e492488ba7e49820be08fe53bee02301226f2773293
```

Puede eliminar varias imágenes en la misma línea de comandos:

```
# podman rmi registry.redhat.io/rhel8/rsyslog support-tools
46da8e23fa1461b658f9276191b4f473f366759a6c840805ed0c9ff694aa7c2f
85cfba5cd49c84786c773a9f66b8d6fca04582d5d7b921a308f04bb8ec071205
```

Si quieres borrar todas las imágenes, puedes utilizar un comando como el siguiente para eliminar todas las imágenes de tu registro local (¡asegúrate de que lo dices en serio antes de hacerlo!):

```
# podman rmi -a
1ca061b47bd70141d11dcb2272dee0f9ea3f76e9afd71cd121a000f3f5423731
ed904b8f2d5c1b5502dea190977e066b4f76776b98f6d5aa1e389256d5212993
83508706ef1b603e511b1b19afcb5faab565053559942db5d00415fb1ee21e96
```

Para eliminar las imágenes que tienen varios nombres (etiquetas) asociados, es necesario añadir la opción de forzar la eliminación. Por ejemplo:

```
# podman rmi -a
A container associated with containers/storage, i.e. via Buildah, CRI-O, etc., may be associated with this image: 1de7d7b3f531

# podman rmi -f 1de7d7b3f531
1de7d7b3f531...
```

CAPÍTULO 3. TRABAJAR CON CONTENEDORES Y PODS

Los contenedores representan un proceso en ejecución o detenido que se genera a partir de los archivos ubicados en una imagen de contenedor descomprimida. En esta sección se describen las herramientas para ejecutar contenedores y trabajar con ellos.

3.1. CONTENEDORES EN FUNCIONAMIENTO

Cuando se ejecuta un comando **podman run**, esencialmente se hace girar y se crea un nuevo contenedor a partir de una imagen de contenedor. El comando que pasas a la línea de comandos **podman run** ve el interior del contenedor como su entorno de ejecución por lo que, por defecto, se puede ver muy poco del sistema anfitrión. Por ejemplo, por defecto, la aplicación en ejecución ve:

- El sistema de archivos proporcionado por la imagen del contenedor.
- Una nueva tabla de procesos desde el interior del contenedor (no se pueden ver los procesos del host).

Si quieres hacer que un directorio del host esté disponible para el contenedor, asignar puertos de red del contenedor al host, limitar la cantidad de memoria que el contenedor puede usar, o expandir los recursos compartidos de la CPU disponibles para el contenedor, puedes hacer esas cosas desde la línea de comandos **podman run**. Aquí hay algunos ejemplos de líneas de comando de **podman run** que habilitan diferentes características.

EXAMPLE #1 (Run a quick command): Este comando podman ejecuta el comando **cat /etc/os-release** para ver el tipo de sistema operativo utilizado como base del contenedor. Después de que el contenedor ejecute el comando, el contenedor sale y se borra (**--rm**).

```
# podman run --rm registry.redhat.io/ubi8/ubi cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.2 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.2"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.2 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8.2:GA"
HOME_URL="https://www.redhat.com/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.2
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.2"
...
```

EXAMPLE #2 (View the Dockerfile in the container): Este es otro ejemplo de ejecución de un comando rápido para inspeccionar el contenido de un contenedor desde el host. Todas las imágenes en capas que Red Hat proporciona incluyen el Dockerfile a partir del cual se construyen en **/root/buildinfo**. En este caso no es necesario montar ningún volumen desde el host.

```
podman run --rm  \
    registry.redhat.io/rhel8/rsyslog  \
    ls /root/buildinfo
```

```
Dockerfile-rhel8-rsyslog-8.2-25
```

Ahora que sabes cómo se llama el Dockerfile, puedes listar su contenido:

```
# podman run --rm registry.redhat.io/rhel8/rsyslog \
    cat /root/buildinfo/Dockerfile-rhel8-rsyslog-8.2-25
FROM sha256:eb205f07ce7d0bb63bfe560...
LABEL maintainer="Red Hat, Inc."

RUN INSTALL_PKGS="\
rsyslog \
rsyslog-gnutls \
rsyslog-gssapi \
rsyslog-mysql \
rsyslog-pgsql \
rsyslog-relp \
" && dnf -y install $INSTALL_PKGS && rpm -V --nosize
    --nofiledigest --nomtime --nomode $INSTALL_PKGS && dnf clean all
LABEL com.redhat.component="rsyslog-container"
LABEL name="rhel8/rsyslog"
LABEL version="8.2"
...
```

EXAMPLE #3 (Run a shell inside the container) El uso de un contenedor para lanzar un shell bash le permite mirar dentro del contenedor y cambiar el contenido. Esto establece el nombre del contenedor como **mybash**. El **-i** crea una sesión interactiva y el **-t** abre una sesión de terminal. Sin **-i**, el shell se abriría y luego saldría. Sin **-t**, el shell permanecería abierto, pero no podrías escribir nada en el shell.

Una vez que ejecute el comando, se le presentará un prompt de shell y podrá comenzar a ejecutar comandos desde el interior del contenedor:

```
# podman run --name=mybash -it registry.redhat.io/ubi8/ubi /bin/bash
[root@ed904b8f2d5c/]#  yum install procps-ng
[root@ed904b8f2d5c/]#  ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 00:46 pts/0    00:00:00 /bin/bash
root        35     1  0 00:51 pts/0    00:00:00 ps -ef
[root@49830c4f9cc4/]# exit
```

Aunque el contenedor ya no se ejecuta una vez que sale, el contenedor sigue existiendo con el nuevo paquete de software aún instalado. Utilice **podman ps -a** para listar el contenedor:

```
# podman ps -a
CONTAINER ID IMAGE                COMMAND   CREATED        STATUS                PORTS NAMES
IS INFRA
1ca061b47bd7 .../ubi8/ubi:latest   /bin/bash 8 minutes ago  Exited 12 seconds ago       musing_brown
false
...
```

Podría iniciar ese contenedor de nuevo usando **podman start** con las opciones de **-ai**. Por ejemplo:

```
# podman start -ai mybash
[root@ed904b8f2d5c/]#
```

EXAMPLE #4 (Bind mounting log files) Una forma de hacer que los mensajes de registro de un contenedor estén disponibles para el sistema anfitrión es montar el dispositivo /dev/log del anfitrión dentro del contenedor. Este ejemplo ilustra cómo ejecutar una aplicación en un contenedor RHEL llamado **log_test** que genera mensajes de registro (sólo el comando logger en este caso) y dirige esos mensajes al dispositivo /dev/log que está montado en el contenedor desde el host. La opción **--rm** elimina el contenedor después de su ejecución.

```
# podman run --name="log_test" -v /dev/log:/dev/log --rm \
    registry.redhat.io/ubi8/ubi logger "Testing logging to the host"
# journalctl -b | grep Testing
Nov 12 20:00:10 ubi8 root[17210]: Testing logging to the host
```

EXAMPLE #5 (Run a service as a daemon with a static IP address) El siguiente ejemplo ejecuta un servicio **rsyslog** como un proceso demonio, por lo que se ejecuta continuamente en segundo plano. También le dice a **podman** que establezca la interfaz de red del contenedor a una dirección IP particular (por ejemplo, 10.88.0.44). Después de eso, puedes ejecutar el comando **podman inspect** para comprobar que la dirección IP se ha establecido correctamente:

```
# podman run -d --ip=10.88.0.44 registry.access.redhat.com/rhel7/rsyslog
efde5f0a8c723f70dd5cb5dc3d5039df3b962fae65575b08662e0d5b5f9fbe85
# podman inspect efde5f0a8c723 | grep 10.88.0.44
          "IPAddress": "10.88.0.44",
```

3.2. INVESTIGAR LOS CONTENEDORES EN FUNCIONAMIENTO Y PARADOS

Después de tener algunos contenedores en ejecución, puedes listar tanto los contenedores que aún están en ejecución como los que han salido o se han detenido con el comando **podman ps**. También puedes usar **podman inspect** para ver información específica dentro de esos contenedores.

3.2.1. Listado de contenedores

Digamos que tienes uno o más contenedores ejecutándose en tu host. Para trabajar con los contenedores desde el sistema anfitrión, puede abrir un shell y probar algunos de los siguientes comandos.

podman ps: La opción **ps** muestra todos los contenedores que se están ejecutando actualmente:

```
# podman run -d registry.redhat.io/rhel8/rsyslog
# podman ps
CONTAINER ID IMAGE              COMMAND         CREATED       STATUS           PORTS NAMES
74b1da000a11 rhel8/rsyslog /bin/rsyslog.sh 2 minutes ago Up About a minute      musing_brown
```

Si hay contenedores que no se están ejecutando, pero no fueron eliminados (opción**--rm**), los contenedores están presentes y pueden ser reiniciados. El comando **podman ps -a** muestra todos los contenedores, en ejecución o detenidos.

```
# podman ps -a
CONTAINER ID IMAGE        COMMAND    CREATED    STATUS              PORTS NAMES     IS
INFRA
d65aecc325a4 ubi8/ubi      /bin/bash  3 secs ago Exited (0) 5 secs ago peaceful_hopper false
74b1da000a11 rhel8/rsyslog rsyslog.sh 2 mins ago Up About a minute     musing_brown    false
```

3.2.2. Inspección de contenedores

Para inspeccionar los metadatos de un contenedor existente, utilice el comando **podman inspect**. Puede mostrar todos los metadatos o sólo los seleccionados para el contenedor. Por ejemplo, para mostrar todos los metadatos de un contenedor seleccionado, escriba:

```
# podman inspect 74b1da000a11
    ...
 "ID": "74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2",
 "Created": "2018-11-13T10:30:31.884673073-05:00",
 "Path": "/bin/rsyslog.sh",
 "Args": [
     "/bin/rsyslog.sh"
 ],
 "State": {
     OciVersion": "1.0.1-dev",
     Status": "running",
     Running": true,
    ...
```

También puede utilizar la función de inspección para extraer determinadas piezas de información de un contenedor. La información se almacena en una jerarquía. Así, para ver la dirección IP del contenedor (IPAddress en NetworkSettings), utilice la opción **--format** y la identidad del contenedor. Por ejemplo:

```
# podman inspect --format='{{.NetworkSettings.IPAddress}}' 74b1da000a11
10.88.0.31
```

Ejemplos de otras piezas de información que puede querer inspeccionar incluyen .Path (para ver el comando que se ejecuta con el contenedor), .Args (argumentos del comando), .Config.ExposedPorts (puertos TCP o UDP expuestos desde el contenedor), .State.Pid (para ver el id de proceso del contenedor) y .HostConfig.PortBindings (mapeo de puertos del contenedor al host). Aquí hay un ejemplo de **.State.Pid** y **.State.StartedAt**:

```
# podman inspect --format='{{.State.Pid}}' 74b1da000a11
19593
# ps -ef | grep 19593
root     19593 19583  0 10:30 ?        00:00:00 /usr/sbin/rsyslogd -n
# podman inspect --format='{{.State.StartedAt}}' 74b1da000a11
2018-11-13 10:30:35.358175255 -0500 EST
```

En el primer ejemplo, se puede ver el ID del proceso del ejecutable en contenedor en el sistema anfitrión (PID 19593). El comando **ps -ef** confirma que es el demonio **rsyslogd** el que se está ejecutando. El segundo ejemplo muestra la fecha y la hora en que se ejecutó el contenedor.

3.2.3. Investigación dentro de un contenedor

Para investigar dentro de un contenedor en ejecución, puede utilizar el comando **podman exec**. Con **podman exec**, puede ejecutar un comando (como **/bin/bash**) para entrar en un proceso de contenedor en ejecución para investigar ese contenedor.

La razón para usar **podman exec**, en lugar de simplemente lanzar el contenedor en un shell bash, es que puedes investigar el contenedor mientras está ejecutando su aplicación prevista. Al adjuntar al contenedor mientras está realizando su tarea prevista, se obtiene una mejor visión de lo que el contenedor realmente hace, sin necesariamente interrumpir la actividad del contenedor.

Aquí hay un ejemplo usando **podman exec** para mirar dentro de un **rsyslog** en funcionamiento, y luego mirar dentro de ese contenedor.

1. **Launch a container**: Lanza un contenedor como la imagen del contenedor **rsyslog** descrita anteriormente. Escriba **podman ps** para asegurarse de que se está ejecutando:

```
# podman ps
CONTAINER ID  IMAGE          COMMAND          CREATED      STATUS       PORTS
NAMES
74b1da000a11  rsyslog:latest "/usr/rsyslog.sh  6 minutes ago Up 6 minutes          rsyslog
```

2. Entre en el contenedor con **podman exec**: Utilice el ID o el nombre del contenedor para abrir un shell bash y acceder al contenedor en ejecución. A continuación, puede investigar los atributos del contenedor de la siguiente manera:

```
# podman exec -it 74b1da000a11 /bin/bash
[root@74b1da000a11 /]# cat /etc/redhat-release
Red Hat Enterprise Linux release 8.0
[root@74b1da000a11 /]# yum install procps-ng
[root@74b1da000a11 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 15:30 ?        00:00:00 /usr/sbin/rsyslogd -n
root         8     0  6 16:01 pts/0    00:00:00 /bin/bash
root        21     8  0 16:01 pts/0    00:00:00 ps -ef
[root@74b1da000a11 /]# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          39G  2.5G   37G   7% /
tmpfs            64M     0   64M   0% /dev
tmpfs           1.5G  8.7M  1.5G   1% /etc/hosts
shm              63M     0   63M   0% /dev/shm
tmpfs           1.5G     0  1.5G   0% /sys/fs/cgroup
tmpfs           1.5G     0  1.5G   0% /proc/acpi
tmpfs           1.5G     0  1.5G   0% /proc/scsi
tmpfs           1.5G     0  1.5G   0% /sys/firmware
[root@74b1da000a11 /]# uname -r
4.18.0-80.1.2.el8_0.x86_64
[root@74b1da000a11 /]# rpm -qa | more
redhat-release-8.0-0.44.el8.x86_64
filesystem-3.8-2.el8.x86_64
basesystem-11-5.el8.noarch
ncurses-base-6.1-7.20180224.el8.noarch
...
bash-4.2# free -m
              total        used        free      shared  buff/cache   available
Mem:           1941         560         139          10        1241        1189
Swap:          1023          15        1008
[root@74b1da000a11 /]# exit
```

Los comandos que se acaban de ejecutar desde el shell bash (que se ejecuta dentro del contenedor) muestran varias cosas.

- El contenedor fue construido a partir de una imagen RHEL release 8.0.
- La tabla de procesos (**ps -ef**) muestra que el comando **/usr/sbin/rsyslogd** es el proceso ID 1.

- Los procesos que se ejecutan en la tabla de procesos del host no se pueden ver desde el contenedor. Aunque el proceso **rsyslogd** puede verse en la tabla de procesos del host (era el proceso ID 19593 en el host).
- No hay un kernel separado que se ejecute en el contenedor (**uname -r** muestra el kernel del sistema anfitrión).
- El comando **rpm -qa** le permite ver los paquetes RPM que están incluidos dentro del contenedor. En otras palabras, hay una base de datos RPM dentro del contenedor.
- La visualización de la memoria (**free -m**) muestra la memoria disponible en el host (aunque lo que el contenedor puede usar realmente puede limitarse usando cgroups).

3.3. ARRANQUE Y PARADA DE CONTENEDORES

Si ejecutó un contenedor, pero no lo eliminó (**--rm**), ese contenedor está almacenado en su sistema local y listo para ejecutarse de nuevo. Para iniciar un contenedor ejecutado previamente que no fue eliminado, utilice la opción **start**. Para detener un contenedor en ejecución, utilice la opción **stop**.

3.3.1. Contenedores de inicio

Un contenedor que no necesita ejecutarse de forma interactiva a veces puede ser reiniciado después de ser detenido con sólo la opción **start** y el ID o nombre del contenedor. Por ejemplo:

```
# podman start myrhel_httpd
myrhel_httpd
```

Para iniciar un contenedor y poder trabajar con él desde el shell local, utiliza las opciones **-a** (adjuntar) y **-i** (interactivo). Una vez que se inicie el shell bash, ejecute los comandos que desee dentro del contenedor y escriba exit para matar el shell y detener el contenedor.

```
# podman start -a -i agitated_hopper
[root@d65aecc325a4 /]#  exit
```

3.3.2. Contenedores de parada

Para detener un contenedor en ejecución que no está unido a una sesión de terminal, utilice la opción de parada y el ID o número del contenedor. Por ejemplo:

```
# podman stop 74b1da000a11
74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2
```

La opción **stop** envía una señal SIGTERM para terminar un contenedor en ejecución. Si el contenedor no se detiene después de un período de gracia (10 segundos por defecto), **podman** envía una señal SIGKILL. También puedes utilizar el comando **podman kill** para matar un contenedor (SIGKILL) o enviar una señal diferente a un contenedor. Aquí hay un ejemplo de enviar una señal SIGHUP a un contenedor (si es soportado por la aplicación, un SIGHUP hace que la aplicación vuelva a leer sus archivos de configuración):

```
# podman kill --signal="SIGHUP" 74b1da000a11
74b1da000a114015886c557deec8bed9dfb80c888097aa83f30ca4074ff55fb2
```

3.4. COMPARTIR ARCHIVOS ENTRE DOS CONTENEDORES

Se pueden utilizar volúmenes para persistir los datos en los contenedores incluso cuando se elimina un contenedor. Los volúmenes pueden utilizarse para compartir datos entre varios contenedores. El volumen es una carpeta que se almacena en la máquina anfitriona. El volumen puede ser compartido entre el contenedor y el host.

Las principales ventajas son:

- Los volúmenes pueden ser compartidos entre los contenedores.
- Los volúmenes son más fáciles de respaldar o migrar.
- Los volúmenes no aumentan el tamaño de los contenedores.

Procedimiento

1. Para crear un volumen, introduzca:

```
$ podman volume create hostvolume
```

2. Para mostrar información sobre el volumen, introduzca:

```
$ podman volume inspect hostvolume
[
    {
        "name": "hostvolume",
        "labels": {},
        "mountpoint":
"/home/username/.local/share/containers/storage/volumes/hostvolume/_data",
        "driver": "local",
        "options": {},
        "scope": "local"
    }
]
```

   Observa que crea un volumen en el directorio de volúmenes. Puede guardar la ruta del punto de montaje en la variable para facilitar su manipulación: **$ mntPoint=$(podman volume inspect hostvolume --format {{.Mountpoint}})**.

   Observe que si ejecuta **sudo podman volume create hostvolume**, el punto de montaje cambia a **/var/lib/containers/storage/volumes/hostvolume/_data**.

3. Crear un archivo de texto dentro del directorio utilizando la ruta se almacena en la variable **mntPoint**:

```
$ echo \ "Hola desde el host" >> $mntPoint/host.txt
```

4. Lista todos los archivos en el directorio definido por la variable **mntPoint**:

```
$ ls $mntPoint/
host.txt
```

5. Ejecute el contenedor llamado **myubi1** y asigne el directorio definido por el nombre del volumen **hostvolume** en el host al directorio **/containervolume1** en el contenedor:

```
$ podman run -it --name myubi1 -v hostvolume:/containervolume1
registry.access.redhat.com/ubi8/ubi /bin/bash
```

Tenga en cuenta que si utiliza la ruta del volumen definida por la variable **mntPoint** (**-v $mntPoint:/containervolume1**), se pueden perder datos al ejecutar el comando **podman volume prune**, que elimina los volúmenes no utilizados. Utilice siempre **-v *hostvolume_name:/containervolume_name***.

6. Enumera los archivos del volumen compartido en el contenedor:

```
# ls /containervolume1
host.txt
```

Puedes ver el archivo **host.txt** que has creado en el host.

7. Cree un archivo de texto dentro del directorio /**containervolume1**:

```
# echo \ "Hola desde el contenedor 1" >> /contenedor1/contenedor1.txt
```

8. Se separa del contenedor con **CTRL p** y **CTRL q**.

9. Liste los archivos en el volumen compartido en el host, debería ver dos archivos:

```
$ ls $mntPoint
container1.rxt  host.txt
```

En este punto, estás compartiendo archivos entre el contenedor y el host. Para compartir archivos entre dos contenedores, ejecute otro contenedor llamado **myubi2**. Los pasos 10 - 13 son análogos a los pasos 5 - 8.

10. Ejecute el contenedor llamado **myubi2** y asigne el directorio definido por el nombre del volumen **hostvolume** en el host al directorio **/containervolume2** en el contenedor:

```
$ podman run -it --name myubi2 -v hostvolume:/containervolume2
registry.access.redhat.com/ubi8/ubi  /bin/bash
```

11. Enumera los archivos del volumen compartido en el contenedor:

```
# ls /containervolume2
container1.txt host.txt
```

Puede ver el archivo **host.txt** que creó en el host y **container1.txt** que creó dentro del contenedor **myubi1**.

12. Cree un archivo de texto dentro del directorio /**containervolume2**:

```
# echo \ "Hola desde el contenedor 2" >> /containervolume2/container2.txt
```

13. Se separa del contenedor con **CTRL p** y **CTRL q**.

14. Enumera los archivos en el volumen compartido en el host, deberías ver tres archivos:

```
$ ls $mntPoint
container1.rxt  container2.txt host.txt
```

15. Para detener y eliminar ambos contenedores, introduzca:

```
$ podman stop myubi1
$ podman stop myubi2
$ podman rm myubi1
$ podman rm myubi2
```

16. Para eliminar el volumen anfitrión, introduzca:

```
$ podman volume rm hostvolume
```

17. Para comprobar que has borrado el volumen, introduce:

```
$ podman volume ls
```

Recursos adicionales

- Para más información sobre el comando **podman volume**, escriba **man podman-volume**.

3.5. RETIRADA DE CONTENEDORES

Para ver una lista de los contenedores que aún se encuentran en tu sistema, ejecuta el comando **podman ps -a**. Para eliminar los contenedores que ya no necesitas, utiliza el comando **podman rm**, con el ID o nombre del contenedor como opción. Deberías detener los contenedores que aún se están ejecutando antes de eliminarlos. Este es un ejemplo:

```
# podman rm goofy_wozniak
```

Puede eliminar varios contenedores en la misma línea de comandos:

```
# podman rm clever_yonath furious_shockley drunk_newton
```

Si quieres borrar todos tus contenedores, puedes usar un comando como el siguiente para eliminar todos los contenedores (no las imágenes) de tu sistema local (¡asegúrate de que lo dices en serio antes de hacerlo!):

```
# podman rm -a
56c496350bd534da7620fe2fa660526a6fc7f1c57b0298291cd2210311fe723b
83ad58c17b20f9e8271171f3023ae094dbfab6ce5708344a68feb121916961ca
a93b696a1f5629300382a8ce860c4ba42f664db98101e82c2dbcc2074b428faf
bee71e61b53bd8b036b2e8cb8f570ef8308403502760a27ee23a4b675d92b93d
```

3.6. CREACIÓN DE VAINAS

Los contenedores son la unidad más pequeña que puedes gestionar con las herramientas de contenedores Podman, Skopeo y Buildah. Un Podman es un grupo de uno o más contenedores. El concepto de Pod fue introducido por Kubernetes. Los pods de Podman son similares a la definición de Kubernetes. Los pods son las unidades de computación más pequeñas que se pueden crear, desplegar y gestionar en entornos OpenShift o Kubernetes. Cada podman incluye un contenedor infra. Este contenedor contiene los espacios de nombres asociados al pod y permite a Podman conectar otros contenedores al pod. Permite iniciar y detener contenedores dentro del pod y el pod seguirá funcionando. El contenedor infra por defecto se basa en la imagen de Kubernetes **k8s.gcr.io/pause**.

Este procedimiento muestra cómo crear un pod con un contenedor.

Procedimiento

1. Cree una vaina vacía:

```
$ podman pod create --name mypod
223df6b390b4ea87a090a4b5207f7b9b003187a6960bd37631ae9bc12c433aff
The pod is in the initial state Created.
```

   La vaina está en el estado inicial Creado.

2. Enumerar todas las vainas:

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED               # OF CONTAINERS   INFRA ID
223df6b390b4  mypod   Created   Less than a second ago  1               3afdcd93de3e
```

   Observa que la vaina tiene un contenedor.

3. Enumerar todos los pods y contenedores asociados a ellos:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                COMMAND  CREATED              STATUS   PORTS
NAMES              POD
3afdcd93de3e  k8s.gcr.io/pause:3.1          Less than a second ago  Created
223df6b390b4-infra  223df6b390b4
```

   Puede ver que el ID del pod del comando **podman ps** coincide con el ID del pod del comando **podman pod ps**. El contenedor infra por defecto está basado en la imagen **k8s.gcr.io/pause**.

4. Para ejecutar un contenedor llamado **myubi** en el pod existente llamado **mypod**, escriba:

```
$ podman run -dt --name myubi --pod mypod registry.access.redhat.com/ubi8/ubi  /bin/bash
5df5c48fea87860cf75822ceab8370548b04c78be9fc156570949013863ccf71
```

5. Enumerar todas las vainas:

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED               # OF CONTAINERS   INFRA ID
223df6b390b4  mypod   Running   Less than a second ago  2               3afdcd93de3e
```

   Puedes ver que la vaina tiene dos contenedores.

6. Enumerar todos los pods y contenedores asociados a ellos:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                  COMMAND    CREATED
STATUS                  PORTS  NAMES              POD
5df5c48fea87  registry.access.redhat.com/ubi8/ubi:latest  /bin/bash  Less than a second ago
Up Less than a second ago          myubi              223df6b390b4
3afdcd93de3e  k8s.gcr.io/pause:3.1                        Less than a second ago  Up Less
than a second ago          223df6b390b4-infra  223df6b390b4
```

Recursos adicionales

- Para más información sobre el comando **podman pod create**, escriba **man podman-pod-create**.
- Para más información sobre los pods, véase el artículo Podman: Managing pods and containers in a local container runtime por Brent Baude.

3.7. VISUALIZACIÓN DE LA INFORMACIÓN DEL POD

Esta sección proporciona información sobre cómo mostrar la información del pod.

Requisitos previos

- El pod ha sido creado. Para más detalles, consulte la sección Creación de pods.

Procedimiento

- Muestra los procesos activos que se ejecutan en un pod:
  - Para mostrar los procesos en ejecución de los contenedores de un pod, introduzca:

```
$ podman pod top mypod
USER   PID   PPID   %CPU    ELAPSED         TTY     TIME   COMMAND
0      1     0      0.000   24.077433518s   ?       0s     /pause
root   1     0      0.000   24.078146025s   pts/0   0s     /bin/bash
```

  - Para mostrar un flujo en vivo de las estadísticas de uso de recursos para los contenedores en uno o más pods, introduzca:

```
$ podman pod stats -a --no-stream
ID            NAME             CPU %   MEM USAGE / LIMIT   MEM %   NET IO    BLOCK IO
PIDS
a9f807ffaacd  frosty_hodgkin   --      3.092MB / 16.7GB    0.02%   -- / --   -- / --    2
3b33001239ee  sleepy_stallman  --      -- / --             --      -- / --   -- / --    --
```

  - Para mostrar la información que describe el pod, introduzca:

```
$ podman pod inspect mypod
{
    "Id": "db99446fa9c6d10b973d1ce55a42a6850357e0cd447d9bac5627bb2516b5b19a",
    "Name": "mypod",
    "Created": "2020-09-08T10:35:07.536541534+02:00",
    "CreateCommand": [
        "podman",
        "pod",
        "create",
        "--name",
        "mypod"
    ],
    "State": "Running",
    "Hostname": "mypod",
    "CreateCgroup": false,
    "CgroupParent": "/libpod_parent",
    "CgroupPath":
```

```
"/libpod_parent/db99446fa9c6d10b973d1ce55a42a6850357e0cd447d9bac5627bb2516b5
b19a",
  "CreateInfra": false,
  "InfraContainerID":
"891c54f70783dcad596d888040700d93f3ead01921894bc19c10b0a03c738ff7",
  "SharedNamespaces": [
    "uts",
    "ipc",
    "net"
  ],
  "NumContainers": 2,
  "Containers": [
    {
      "Id":
"891c54f70783dcad596d888040700d93f3ead01921894bc19c10b0a03c738ff7",
      "Name": "db99446fa9c6-infra",
      "State": "running"
    },
    {
      "Id":
"effc5bbcfe505b522e3bf8fbb5705a39f94a455a66fd81e542bcc27d39727d2d",
      "Name": "myubi",
      "State": "running"
    }
  ]
}
```


Puede ver información sobre los contenedores en el pod.

Recursos adicionales

- Para más información sobre el comando **podman pod top**, escriba **man podman-pod-top**.
- Para más información sobre el comando **podman pod stats**, escriba **man podman-pod-stats**.
- Para más información sobre el comando **podman pod inspect**, escriba **man podman-pod-inspect**.

3.8. DETENCIÓN DE VAINAS

Puede detener uno o más pods utilizando el comando **podman pod stop**.

Requisitos previos

- El pod ha sido creado. Para más detalles, consulte la sección Creación de pods.

Procedimiento

1. Para detener la vaina **mypod**, escriba:

```
$ podman pod stop mypod
```

2. Enumerar todos los pods y contenedores asociados a ellos:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                 COMMAND    CREATED            STATUS
PORTS   NAMES              POD ID        PODNAME
5df5c48fea87  registry.redhat.io/ubi8/ubi:latest  /bin/bash  About a minute ago  Exited (0) 7
seconds ago         myubi              223df6b390b4  mypod

3afdcd93de3e  k8s.gcr.io/pause:3.2                  About a minute ago  Exited (0) 7
seconds ago         8a4e6527ac9d-infra  223df6b390b4  mypod
```

Puede ver que el pod **mypod** y el contenedor **myubi** están en estado "Exited".

Recursos adicionales

- Para más información sobre el comando **podman pod stop**, escriba **man podman-pod-stop**.

3.9. RETIRAR LAS VAINAS

Puede eliminar uno o varios pods y contenedores detenidos utilizando el comando **podman pod rm**.

Requisitos previos

- El pod ha sido creado. Para más detalles, consulte la sección Creación de pods.
- El pod ha sido detenido. Para más detalles, consulte la sección Detención de pods.

Procedimiento

1. Para eliminar la vaina **mypod**, escriba:

```
$ podman pod rm mypod
223df6b390b4ea87a090a4b5207f7b9b003187a6960bd37631ae9bc12c433aff
```

Tenga en cuenta que al eliminar el pod se eliminan automáticamente todos los contenedores que hay en su interior.

2. Para comprobar que se han eliminado todos los contenedores y vainas, escriba:

```
$ podman ps
$ podman pod ps
```

Recursos adicionales

- Para más información sobre el comando **podman pod rm**, escriba **man podman-pod-rm**.

CAPÍTULO 4. AÑADIR SOFTWARE A UN CONTENEDOR UBI EN FUNCIONAMIENTO

Las imágenes UBI se construyen a partir del contenido de Red Hat. Estas imágenes UBI también proporcionan un subconjunto de paquetes de Red Hat Enterprise Linux que están disponibles gratuitamente para ser instalados para su uso con UBI. Para añadir o actualizar software, las imágenes UBI están preconfiguradas para apuntar a los repositorios yum disponibles libremente que contienen los RPMs oficiales de Red Hat.

Para añadir paquetes de los repositorios UBI a los contenedores UBI en funcionamiento:

- En las imágenes de **ubi**, el comando **yum** está instalado para permitirle dibujar paquetes.
- En las imágenes de **ubi-minimal**, se incluye el comando **microdnf** (con un conjunto de características más reducido) en lugar de **yum**.

Tenga en cuenta que instalar y trabajar con paquetes de software directamente en contenedores en funcionamiento es sólo para añadir paquetes temporalmente o aprender sobre los repositorios. Consulte la sección "Construir una imagen basada en UBI" para conocer formas más permanentes de construir imágenes basadas en UBI.

A continuación se exponen algunas cuestiones que deben tenerse en cuenta al trabajar con imágenes UBI:

- Cientos de paquetes RPM utilizados en las imágenes de tiempo de ejecución de los flujos de aplicaciones existentes están almacenados en los repositorios yum empaquetados con las nuevas imágenes UBI. Siéntase libre de instalar esos RPMs en sus imágenes UBI para emular el tiempo de ejecución (python, php, nodejs, etc.) que le interese.
- Debido a que algunos archivos de lenguaje y documentación han sido despojados de la imagen mínima de UBI (**ubi8**/**ubi-minimal**), ejecutar **rpm -Va** dentro de ese contenedor mostrará el contenido de muchos paquetes como faltantes o modificados. Si tener una lista completa de archivos dentro de ese contenedor es importante para usted, considere usar una herramienta como **Tripwire** para registrar los archivos en el contenedor y comprobarlo más tarde.
- Después de crear una imagen por capas, utilice **podman history** para comprobar en qué imagen UBI se construyó. Por ejemplo, después de completar el ejemplo del servidor web mostrado anteriormente, escriba **podman history johndoe/webserver** para ver que la imagen sobre la que se construyó incluye el ID de la imagen UBI que añadió en la línea FROM del Dockerfile.

Cuando se añade software a un contenedor UBI, los procedimientos difieren para actualizar las imágenes UBI en un host RHEL suscrito o en un sistema no suscrito (o no RHEL). Estas dos formas de trabajar con imágenes UBI se ilustran a continuación.

4.1. AÑADIR SOFTWARE A UN CONTENEDOR UBI EN EL HOST SUSCRITO

Si está ejecutando un contenedor UBI en un host RHEL registrado y suscrito, el repositorio principal de RHEL Server está habilitado dentro del contenedor UBI estándar, junto con todos los repositorios UBI. Así que el conjunto completo de paquetes de Red Hat está disponible. Desde el contenedor mínimo UBI, todos los repositorios UBI están habilitados por defecto, pero ningún repositorio está habilitado desde el host por defecto.

4.2. AÑADIR SOFTWARE DENTRO DEL CONTENEDOR ESTÁNDAR UBI

Para garantizar que los contenedores que construya puedan ser redistribuidos, desactive los repositorios yum que no sean UBI en la imagen estándar de UBI cuando añada software. Si desactiva todos los repositorios yum excepto los repositorios UBI, sólo se utilizarán los paquetes de los repositorios de libre acceso cuando añada software.

Con un shell abierto dentro de un contenedor de imagen base UBI estándar (**ubi8**/**ubi**) de un host RHEL suscrito, ejecute el siguiente comando para añadir un paquete a ese contenedor (por ejemplo, el paquete **bzip2**):

```
# yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos bzip2
```

Para añadir software dentro de un contenedor UBI estándar que está en el repositorio del servidor RHEL, pero no en los repositorios UBI, no desactive ningún repositorio y simplemente instale el paquete:

```
# yum install zsh
```

Para instalar un paquete que está en un repositorio diferente desde el contenedor UBI estándar, tiene que habilitar explícitamente el repositorio que necesita. Por ejemplo:

```
# yum install --enablerepo=rhel-7-server-optional-rpms zsh-html
```

AVISO

La instalación de paquetes de Red Hat que no están dentro de los repos de Red Hat UBI puede limitar la distribución del contenedor fuera de los hosts suscritos.

4.3. AÑADIR SOFTWARE DENTRO DEL CONTENEDOR MÍNIMO DE UBI

Los repositorios UBI yum están habilitados dentro de la imagen mínima de UBI por defecto.

Para instalar el mismo paquete demostrado anteriormente (**bzip2**) desde uno de esos repositorios UBI yum en un host RHEL suscrito desde el contenedor mínimo UBI, escriba:

```
# microdnf install bzip2
```

Para instalar paquetes dentro de un contenedor UBI mínimo desde repositorios disponibles en un host suscrito que no son parte de un repositorio UBI yum, tendría que habilitar explícitamente esos repositorios. Por ejemplo:

```
# microdnf install --enablerepo=rhel-7-server-rpms zsh
# microdnf install --enablerepo=rhel-7-server-rpms \
      --enablerepo=rhel-7-server-optional-rpms zsh-html
```

AVISO

El uso de repositorios RHEL no UBI para instalar paquetes en sus imágenes UBI podría restringir su capacidad de compartir esas imágenes para ejecutarlas fuera de los sistemas RHEL suscritos.

4.4. AÑADIR SOFTWARE A UN CONTENEDOR UBI EN UN HOST NO SUSCRITO

Para añadir paquetes de software a un contenedor en ejecución que está en un host RHEL no suscrito o en algún otro sistema Linux, no es necesario desactivar ningún repositorio yum. Por ejemplo:

```
# yum install bzip2
```

Para instalar ese paquete en un host RHEL no suscrito desde el contenedor mínimo UBI, escriba:

```
# microdnf install bzip2
```

Como se ha señalado anteriormente, estos dos medios de añadir software a un contenedor UBI en funcionamiento no están pensados para crear imágenes de contenedor permanentes basadas en UBI. Para ello, debes construir nuevas capas sobre las imágenes UBI, como se describe en la siguiente sección.

4.5. CONSTRUIR UNA IMAGEN BASADA EN EL UBI

Puede construir imágenes de contenedores basadas en UBI de la misma manera que construye otras imágenes, con una excepción. Debe desactivar todos los repositorios yum que no sean UBI cuando construya las imágenes, si quiere estar seguro de que su imagen sólo contiene software de Red Hat que pueda redistribuir.

Este es un ejemplo de creación de un contenedor de servidor web basado en UBI a partir de un archivo Docker con la utilidad **buildah**:

NOTA

Para las imágenes de **ubi8/ubi-minimal**, utilice **microdnf** en lugar de **yum**:

```
RUN microdnf update -y && rm -rf /var/cache/yum
RUN microdnf install httpd -y && microdnf clean all
```

1. **Create a Dockerfile**: Agregue un **Dockerfile** con el siguiente contenido a un nuevo directorio:

```
FROM registry.access.redhat.com/ubi8/ubi
USER root
LABEL maintainer="John Doe"
# Update image
RUN yum update --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos
-y && rm -rf /var/cache/yum
```

```
RUN yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-8-baseos
httpd -y && rm -rf /var/cache/yum
# Add default Web page and expose port
RUN echo "The Web Server is Running" > /var/www/html/index.html
EXPOSE 80
# Start the service
CMD ["-D", "FOREGROUND"]
ENTRYPOINT ["/usr/sbin/httpd"]
```

2. **Build the new image**: Estando en ese directorio, utilice **buildah** para crear una nueva imagen con capas UBI:

```
# buildah bud -t johndoe/webserver .
STEP 1: FROM registry.access.redhat.com/ubi8/ubi:latest
STEP 2: USER root
STEP 3: LABEL maintainer="John Doe"
STEP 4: RUN yum update --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-
8-baseos -y
. . .
No packages marked for update
STEP 5: RUN yum install --disablerepo=* --enablerepo=ubi-8-appstream --enablerepo=ubi-
8-baseos httpd -y
Loaded plugins: ovl, product-id, search-disabled-repos
Resolving Dependencies
--> Running transaction check
================================================================
 Package                Arch             Version                      Repository                 Size
================================================================
Installing:
 httpd              x86_64 2.4.37-10
                                        latest-rhubi-8.0-appstream 1.4 M
Installing dependencies:
 apr                x86_64 1.6.3-9.el8        latest-rhubi-8.0-appstream 125 k
 apr-util           x86_64 1.6.1-6.el8        latest-rhubi-8.0-appstream 105 k
 httpd-filesystem   noarch 2.4.37-10
                                        latest-rhubi-8.0-appstream  34 k
 httpd-tools        x86_64 2.4.37-10.
...

Transaction Summary
...
Complete!
STEP 6: RUN echo "The Web Server is Running" > /var/www/html/index.html
STEP 7: EXPOSE 80
STEP 8: CMD ["-D", "FOREGROUND"]
STEP 9: ENTRYPOINT ["/usr/sbin/httpd"]
STEP 10: COMMIT
...
Writing manifest to image destination
Storing signatures
--> 36a604cc0dd3657b46f8762d7ef69873f65e16343b54c63096e636c80f0d68c7
```

3. **Test**: Pruebe la imagen del servidor web con capas UBI:

```
# podman run -d -p 80:80 johndoe/webserver
```

```
bbe98c71d18720d966e4567949888dc4fb86eec7d304e785d5177168a5965f64
# curl http://localhost/index.html
The Web Server is Running
```

4.6. UTILIZACIÓN DE IMÁGENES EN TIEMPO DE EJECUCIÓN DEL FLUJO DE APLICACIONES

El flujo de aplicaciones de Red Hat Enterprise Linux 8 ofrece otro conjunto de imágenes de contenedores que puede utilizar como base para sus construcciones de contenedores. Estas imágenes están construidas sobre las imágenes base estándar de RHEL, con la mayoría ya actualizadas como imágenes UBI. Cada una de estas imágenes incluye software adicional que puede querer utilizar para entornos de ejecución específicos.

Si espera construir múltiples imágenes que requieran, por ejemplo, software de ejecución php, puede utilizar una plataforma más consistente para esas imágenes comenzando con una imagen de flujo de aplicación PHP.

Aquí hay algunos ejemplos de imágenes de contenedores de flujo de aplicaciones construidas sobre imágenes base UBI, que están disponibles en el Registro de Red Hat (registry.access.redhat.com o registry.redhat.io):

- **ubi8/php-72**: Plataforma PHP 7.2 para construir y ejecutar aplicaciones
- **ubi8/nodejs-10**: Plataforma Node.js 10 para construir y ejecutar aplicaciones. Utilizado por las compilaciones de Node.js 10 Source-To-Image
- **ubi8/ruby25**: Plataforma Ruby 2.5 para construir y ejecutar aplicaciones
- **ubi8/python-27**: Plataforma Python 2.7 para construir y ejecutar aplicaciones
- **ubi8/python-36**: Plataforma Python 3.6 para construir y ejecutar aplicaciones
- **ubi8/s2i-core**: Imagen base con bibliotecas y herramientas esenciales utilizadas como base para las imágenes de los constructores como perl, python, ruby, etc
- **ubi8/s2i-base**: Imagen base para las construcciones de origen a imagen

Debido a que estas imágenes UBI contienen el mismo software básico que sus contrapartes de imágenes heredadas, puede aprender sobre esas imágenes en la guía Uso de imágenes de contenedores de Red Hat Software Collections. Asegúrese de utilizar los nombres de las imágenes UBI para extraer esas imágenes.

Las imágenes de contenedores de RHEL 8 Application Stream se actualizan cada vez que se actualizan las imágenes base de RHEL 8. En el caso de RHEL 7, estas mismas imágenes (denominadas imágenes de Red Hat Software Collections) se actualizan según un calendario independiente de las actualizaciones de las imágenes base de RHEL (al igual que las imágenes relacionadas para Dotnet y DevTools). Busque en el Catálogo de Contenedores de Red Hat para obtener detalles sobre cualquiera de estas imágenes. Para más información sobre el calendario de actualizaciones, consulte Actualizaciones de imágenes de contenedores de Red Hat.

4.7. OBTENER EL CÓDIGO FUENTE DE LA IMAGEN DEL CONTENEDOR UBI

El código fuente está disponible para todas las imágenes basadas en Red Hat UBI en forma de contenedores descargables. Antes de continuar, tenga en cuenta los contenedores de código fuente de Red Hat:

- Las imágenes de contenedor de origen no se pueden ejecutar, a pesar de estar empaquetadas como contenedores. Para instalar las imágenes de contenedor de origen de Red Hat en su sistema, utilice el comando **skopeo command**, en lugar de utilizar **podman pull**.
  - Utilice el comando **skopeo copy** para copiar una imagen de contenedor de origen en un directorio de su sistema local.
  - Utilice el comando **skopeo inspect** para inspeccionar la imagen del contenedor de origen.
- Para más detalles sobre el comando **skopeo**, vea la Sección 1.5. Uso de skopeo para trabajar con registros de contenedores.
- Las imágenes del contenedor fuente se nombran en base a los contenedores binarios que representan. Por ejemplo, para un contenedor estándar RHEL UBI 8 en particular **registry.access.redhat.com/ubi8:8.1-397** anexa **-source** para obtener la imagen del contenedor fuente (**registry.access.redhat.com/ubi8:8.1-397-source**).
- Una vez que la imagen del contenedor de origen se ha copiado en un directorio local, puede utilizar una combinación de comandos **tar**, **gzip**, y **rpm** para trabajar con ese contenido.
- Pueden pasar varias horas desde que se publica una imagen de contenedor hasta que su contenedor fuente asociado esté disponible.

Procedimiento

1. Utilice el comando **skopeo copy** para copiar la imagen del contenedor de origen en un directorio local:

```
$ skopeo copy \
docker://registry.access.redhat.com/ubi8:8.1-397-source \
dir:$HOME/TEST
...
Copying blob 477bc8106765 done
Copying blob c438818481d3 done
Copying blob 26fe858c966c done
Copying blob ba4b5f020b99 done
Copying blob f7d970ccd456 done
Copying blob ade06f94b556 done
Copying blob cc56c782b513 done
Copying blob dcf9396fdada done
Copying blob feb6d2ae2524 done
Copying config dd4cd669a4 done
Writing manifest to image destination
Storing signatures
```

2. Utilice el comando **skopeo inspect** para inspeccionar la imagen del contenedor de origen:

```
$ skopeo inspect dir:$HOME/TEST
{
    "Digest":
"sha256:7ab721ef3305271bbb629a6db065c59bbeb87bc53e7cbf88e2953a1217ba7322",
    "RepoTags": [],
```

```
    "Created": "2020-02-11T12:14:18.612461174Z",
    "DockerVersion": "",
    "Labels": null,
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:1ae73d938ab9f11718d0f6a4148eb07d38ac1c0a70b1d03e751de8bf3c2c87fa",
        "sha256:9fe966885cb8712c47efe5ecc2eaa0797a0d5ffb8b119c4bd4b400cc9e255421",
        "sha256:61b2527a4b836a4efbb82dfd449c0556c0f769570a6c02e112f88f8bbcd90166",
        ...
        "sha256:cc56c782b513e2bdd2cc2af77b69e13df4ab624ddb856c4d086206b46b9b9e5f",
        "sha256:dcf9396fdada4e6c1ce667b306b7f08a83c9e6b39d0955c481b8ea5b2a465b32",

"sha256:feb6d2ae252402ea6a6fca8a158a7d32c7e4572db0e6e5a5eab15d4e0777951e"
    ],
    "Env": null
}
```

3. Para desencajar todo el contenido, escriba:

```
$ cd $HOME/TEST
$ for f in $(ls); do tar xvf $f; done
```

4. Para comprobar los resultados, escriba:

```
$ find blobs/ rpm_dir/
blobs/
blobs/sha256
blobs/sha256/10914f1fff060ce31388f5ab963871870535aaaa551629f5ad182384d60fdf82
rpm_dir/
rpm_dir/gzip-1.9-4.el8.src.rpm
```

5. Empezar a examinar y utilizar el contenido.

4.8. RECURSOS ADICIONALES

- Los socios y clientes de Red Hat pueden solicitar nuevas funcionalidades, incluyendo peticiones de paquetes, rellenando un ticket de soporte a través de los métodos estándar. Los clientes que no son de Red Hat no reciben soporte, pero pueden presentar solicitudes a través del Bugzilla estándar de Red Hat para el producto RHEL correspondiente. Para más información, véase Red Hat Bugzilla Queue

- Los socios y clientes de Red Hat pueden presentar tickets de soporte a través de métodos estándar cuando se ejecuta UBI en una plataforma de Red Hat soportada (OpenShift/RHEL). El personal de soporte de Red Hat guiará a los socios y clientes. Para más información, consulte Abrir un caso de soporte

CAPÍTULO 5. EJECUCIÓN DE SKOPEO Y BUILDAH EN UN CONTENEDOR

Con Skopeo, puede inspeccionar imágenes en un registro remoto sin tener que descargar la imagen completa con todas sus capas. También puede utilizar Skopeo para copiar imágenes, firmar imágenes, sincronizar imágenes y convertir imágenes en diferentes formatos y compresiones de capas.

Buildah facilita la construcción de imágenes de contenedores OCI. Con Buildah, puedes crear un contenedor de trabajo, ya sea desde cero o utilizando una imagen como punto de partida. Puedes crear una imagen desde un contenedor en funcionamiento o a través de las instrucciones de un archivo Docker. Puedes montar y desmontar el sistema de archivos raíz de un contenedor en funcionamiento.

Razones para ejecutar Buildah y Skopeo en un contenedor:

- **Skopeo**: Puede ejecutar un sistema CI/CD dentro de Kubernetes o utilizar OpenShift para construir sus imágenes de contenedor, y posiblemente distribuir esas imágenes a través de diferentes registros de contenedores. Para integrar Skopeo en un flujo de trabajo de Kubernetes, es necesario ejecutarlo en un contenedor.

- **Buildah**: Quieres construir imágenes de OCI/contenedor dentro de un sistema de CI/CD de Kubernetes u OpenShift que está constantemente construyendo imágenes. Anteriormente, la gente utilizaba un socket Docker para conectarse al motor de contenedores y realizar un comando **docker build**. Esto era el equivalente a dar acceso de root al sistema sin requerir una contraseña, lo cual no es seguro. Por esta razón, Red Hat recomienda utilizar Buildah en un contenedor.

- **Both**: Estás ejecutando un sistema operativo antiguo en el host pero quieres ejecutar la última versión de Skopeo, Buildah, o ambos. La solución es ejecutar Buildah en un contenedor. Por ejemplo, esto es útil para ejecutar la última versión de Skopeo, Buildah, o ambos proporcionados en RHEL 8 en un host contenedor RHEL 7 que no tiene acceso a las versiones más nuevas de forma nativa.

- **Both**: Una restricción común en los entornos HPC es que los usuarios no root no pueden instalar paquetes en el host. Cuando ejecutas Skopeo, Buildah, o ambos en un contenedor, puedes realizar estas tareas específicas como usuario no root.

5.1. EJECUCIÓN DE SKOPEO EN UN CONTENEDOR

Este procedimiento demuestra cómo inspeccionar una imagen de contenedor remoto utilizando Skopeo. Ejecutar Skopeo en un contenedor significa que el sistema de archivos raíz del contenedor está aislado del sistema de archivos raíz del host. Para compartir o copiar archivos entre el host y el contenedor, tienes que montar archivos y directorios.

Procedimiento

1. Inicie sesión en el registro de registry.redhat.io:

```
$ podman login registry.redhat.io
Username: myuser@mycompany.com
Password: ***********
Login Succeeded!
```

2. Obtenga la imagen del contenedor **registry.redhat.io/rhel8/skopeo**:

```
$ podman pull registry.redhat.io/rhel8/skopeo
```

3. Inspeccionar una imagen de contenedor remoto **registry.access.redhat.com/ubi8/ubi** utilizando Skopeo:

```
$ podman run --rm registry.redhat.io/rhel8/skopeo skopeo inspect
docker://registry.access.redhat.com/ubi8/ubi
{
    "Name": "registry.access.redhat.com/ubi8/ubi",
    ...
    "Labels": {
        "architecture": "x86_64",
        ...
        "name": "ubi8",
        ...
        "summary": "Provides the latest release of Red Hat Universal Base Image 8.",
        "url":
"https://access.redhat.com/containers/#/registry.access.redhat.com/ubi8/images/8.2-347",
        ...
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
    ...
    ],
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "container=oci"
    ]
}
```

La opción **--rm** elimina la imagen **registry.redhat.io/rhel8/skopeo** tras la salida del contenedor.

Recursos adicionales

- Para más información sobre cómo ejecutar Skopeo en un contenedor, consulte el artículo Cómo ejecutar skopeo en un contenedor, de Valentin Rothberg.

5.2. EJECUTAR SKOPEO EN UN CONTENEDOR UTILIZANDO CREDENCIALES

Trabajar con registros de contenedores requiere una autentificación para acceder y alterar los datos. Skopeo admite varias formas de especificar las credenciales.

Con este enfoque puede especificar las credenciales en la línea de comandos utilizando la opción **--cred USERNAME[:PASSWORD]**.

Procedimiento

- Inspeccionar una imagen de contenedor remoto usando Skopeo contra un registro bloqueado:

```
$ podman run --rm registry.redhat.io/rhel8/skopeo inspect --creds $USER:$PASSWORD
docker://$IMAGE
```

Recursos adicionales

- Para más información sobre cómo ejecutar Skopeo en un contenedor, consulte el artículo Cómo ejecutar skopeo en un contenedor, de Valentin Rothberg.

5.3. EJECUTAR SKOPEO EN UN CONTENEDOR UTILIZANDO AUTHFILES

Puede utilizar un archivo de autenticación (authfile) para especificar las credenciales. El comando **skopeo login** entra en el registro específico y almacena el token de autenticación en el authfile. La ventaja de utilizar authfiles es que se evita la necesidad de introducir repetidamente las credenciales.

Cuando se ejecuta en el mismo host, todas las herramientas de contenedor como Skopeo, Buildah y Podman comparten el mismo archivo de autenticación. Cuando se ejecuta Skopeo en un contenedor, hay que compartir el archivo de autenticidad en el host montando el volumen del archivo de autenticidad en el contenedor, o hay que reautenticarse dentro del contenedor.

Procedimiento

- Inspeccionar una imagen de contenedor remoto usando Skopeo contra un registro bloqueado:

```
$ podman run --rm -v $AUTHFILE:/auth.json registry.redhat.io/rhel8/skopeo inspect
docker://$IMAGE
```

  La opción **-v $AUTHFILE:/auth.json** monta un archivo de autenticidad en /auth.json dentro del contenedor. Skopeo puede ahora acceder a los tokens de autenticación en el archivo auth en el host y obtener un acceso seguro al registro.

Otros comandos de Skopeo funcionan de forma similar, por ejemplo:

- Utilice el comando **skopeo-copy** para especificar las credenciales en la línea de comandos para la imagen de origen y de destino utilizando las opciones **--source-creds** y **--dest-creds**. También lee el /**auth.json** authfile.

- Si desea especificar archivos automáticos separados para la imagen de origen y la de destino, utilice las opciones **--source-authfile** y **--dest-authfile** y monte esos archivos automáticos desde el host al contenedor.

Recursos adicionales

- Para más información sobre cómo ejecutar Skopeo en un contenedor, consulte el artículo Cómo ejecutar skopeo en un contenedor, de Valentin Rothberg.

5.4. COPIAR IMÁGENES DE CONTENEDORES HACIA O DESDE EL HOST

Skopeo, Buildah y Podman comparten el mismo almacenamiento local de imágenes de contenedores. Si quieres copiar contenedores hacia o desde el almacenamiento del contenedor anfitrión, necesitas montarlo en el contenedor Skopeo.

NOTA

La ruta de acceso al almacenamiento del contenedor anfitrión difiere entre los usuarios root (**/var/lib/containers/storage**) y los no root (**$HOME/.local/share/containers/storage**).

Procedimiento

1. Copie la imagen **registry.access.redhat.com/ubi8/ubi** en su almacenamiento local de contenedores:

```
$ podman run --privileged --rm -v
$HOME/.local/share/containers/storage:/var/lib/containers/storage
registry.redhat.io/rhel8/skopeo skopeo copy docker://registry.access.redhat.com/ubi8/ubi
containers-storage:registry.access.redhat.com/ubi8/ubi
```

   - La opción **--privileged** desactiva todos los mecanismos de seguridad. Red Hat recomienda utilizar esta opción sólo en entornos de confianza.
   - Para evitar desactivar los mecanismos de seguridad, exporte las imágenes a un tarball o cualquier otro transporte de imágenes basado en la ruta y móntelas en el contenedor Skopeo:
     - **$ podman save --format oci-archive -o oci.tar $IMAGE**
     - **$ podman run --rm -v oci.tar:/oci.tar registry.redhat.io/rhel8/skopeo copy oci-archive:/oci.tar $DESTINATION**

2. Para listar las imágenes en el almacenamiento local:

```
$ podman images
REPOSITORY                          TAG     IMAGE ID      CREATED       SIZE
registry.access.redhat.com/ubi8/ubi      latest  ecbc6f53bba0  8 weeks ago   211 MB
```

Recursos adicionales

- Para más información sobre cómo ejecutar Skopeo en un contenedor, consulte el artículo Cómo ejecutar skopeo en un contenedor, de Valentin Rothberg.

5.5. EJECUTAR BUILDAH EN UN CONTENEDOR

El procedimiento demuestra cómo ejecutar Buildah en un contenedor y crear un contenedor de trabajo basado en una imagen.

Procedimiento

1. Inicie sesión en el registro de registry.redhat.io:

```
$ podman login registry.redhat.io
Username: myuser@mycompany.com
Password: ***********
Login Succeeded!
```

2. Extraiga y ejecute la imagen **registry.redhat.io/rhel8/buildah**:

```
# podman run --rm --device /dev/fuse -it registry.redhat.io/rhel8/buildah  /bin/bash
```

- La opción **--rm** elimina la imagen **registry.redhat.io/rhel8/buildah** tras la salida del contenedor.
- La opción **--device** añade un dispositivo anfitrión al contenedor.

3. Cree un nuevo contenedor utilizando una imagen de **registry.access.redhat.com/ubi8**:

```
# buildah --storage-opt=overlay.mount_program=/usr/bin/fuse-overlayfs from
registry.access.redhat.com/ubi8
...
ubi8-working-container
```

- La opción **--storage-opt** establece el controlador de almacenamiento. Esta opción anula todas las opciones configuradas en las variables de entorno **/etc/containers/storage.conf** y **STORAGE_OPTS**.
- **/usr/bin/fuse-overlayfs** es una implementación de FUSE (Filesystem in Userspace) y permite a los usuarios no root crear sus sistemas de archivos sin modificar el código del kernel.

4. Ejecute el comando **ls** / dentro del contenedor **ubi8-working-container**:

```
# buildah --storage-opt=overlay.mount_program=/usr/bin/fuse-overlayfs run  --
isolation=chroot ubi8-working-container ls /
bin  boot  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv
```

5. Para listar todas las imágenes en un almacenamiento local, introduzca:

```
# buildah images
REPOSITORY                       TAG      IMAGE ID       CREATED       SIZE
registry.access.redhat.com/ubi8  latest   ecbc6f53bba0   5 weeks ago   211 MB
```

6. Para listar los contenedores en funcionamiento y sus imágenes base, introduzca:

```
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
0aaba7192762     *     ecbc6f53bba0 registry.access.redhat.com/ub... ubi8-working-container
```

7. Para empujar la imagen de **registry.access.redhat.com/ubi8** al registro local ubicado en **registry.example.com**:

```
# buildah push ecbc6f53bba0 registry.example.com:5000/ubi8/ubi
```

Recursos adicionales

- Para más información sobre cómo ejecutar Buildah en un contenedor, consulta el artículo Best practices for running Buildah in a container de Daniel Walsh.

CAPÍTULO 6. EJECUCIÓN DE IMÁGENES DE CONTENEDORES ESPECIALES

Utilice este capítulo para conocer algunos tipos especiales de imágenes de contenedores. Estos incluyen:

- **Toolbox**: En lugar de sobrecargar un sistema anfitrión instalando las herramientas necesarias para depurar problemas o supervisar funciones, puede ejecutar el comando **toolbox**. Toolbox inicia una imagen de contenedor **support-tools** que contiene herramientas que puede utilizar para ejecutar informes o diagnosticar problemas en el host.
- **Runlabels**: Algunas imágenes de contenedores tienen etiquetas incorporadas que permiten ejecutar esos contenedores con opciones y argumentos preestablecidos. El comando **podman container runlabel <label>**, le permite ejecutar el comando definido en ese **<label>** para la imagen del contenedor. Las etiquetas soportadas son **install**, **run** y **uninstall**.

6.1. SOLUCIÓN DE PROBLEMAS DE HOSTS DE CONTENEDORES CON LA CAJA DE HERRAMIENTAS

En lugar de instalar las herramientas de solución de problemas directamente en su sistema RHEL 8, la utilidad **toolbox** ofrece una forma de añadir temporalmente esas herramientas y luego descartarlas fácilmente cuando haya terminado. La utilidad **toolbox** funciona de la siguiente manera:

- Llevando la imagen de **registry.redhat.io/rhel8/support-tools** a su sistema local.
- Arrancar un contenedor desde la imagen, y luego ejecutar un shell dentro del contenedor desde el que se puede acceder al sistema anfitrión.

El contenedor **support-tools** le permite:

- Ejecutar comandos que pueden no estar instalados en el sistema anfitrión, como **sosreport**, **strace**, o **tcpdump**, de manera que les permita actuar en el sistema anfitrión.
- Instalar más software dentro del contenedor para utilizarlo en el sistema anfitrión.
- Deseche el recipiente cuando haya terminado.

A continuación se ilustra una sesión típica de **toolbox**.

Procedimiento

1. Asegúrese de que los paquetes **toolbox** y **podman** están instalados:

   ```
   # yum module list container-tools
   ```

   Para instalar el conjunto completo de herramientas para contenedores, escriba:

   ```
   # yum module install container-tools -y
   ```

2. Ejecute el comando de la caja de herramientas para extraer y ejecutar la imagen **support-tools** (introduciendo sus credenciales del Portal del Cliente de Red Hat cuando se le solicite):

   ```
   # toolbox
   Trying to pull registry.redhat.io/rhel8/support-tools...
   ```

```
...
Would you like to authenticate to registry: 'registry.redhat.io' and try again? [y/N] y
Username: johndoe
Password: **********
Login Succeeded!
Trying to pull registry.redhat.io/rhel8/support-tools...Getting image source signatures
...
Storing signatures
30e261462851238d38f4ef2afdaf55f1f8187775c5ca373b43e0f55722faaf97
Spawning a container 'toolbox-root' with image 'registry.redhat.io/rhel8/support-tools'
Detected RUN label in the container image. Using that as the default...
command: podman run -it --name toolbox-root --privileged --ipc=host --net=host --pid=host -e
HOST=/host -e NAME=toolbox-root -e IMAGE=registry.redhat.io/rhel8/support-tools:latest -v
/run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v /etc/localtime:/etc/localtime -
v /:/host registry.redhat.io/rhel8/support-tools:latest
```

3. Abra un shell bash para ejecutar comandos dentro del contenedor:

```
# bash-4.4#
```

4. Desde el interior del contenedor, el sistema de archivos raíz del host está disponible en el directorio **/host**. Los otros directorios mostrados están todos dentro del contenedor.

```
# ls /
bin   dev  home  lib lost+found  mnt  proc  run   srv  tmp  var
boot  etc  host  lib64 media     opt  root  sbin  sys  usr
```

5. Intente ejecutar un comando dentro de su contenedor. El comando **sosreport** le permite generar información sobre su sistema para enviarla al soporte de Red Hat:

```
bash-4.4# sosreport

sosreport (version 3.6)
This command will collect diagnostic and configuration information from
this Red Hat Enterprise Linux system and installed applications.

An archive containing the collected information will be generated in
/host/var/tmp/sos.u82evisb and may be provided to a Red Hat support
representative.
...
Press ENTER to continue, or CTRL-C to quit.   <Press ENTER>
...
Your sosreport has been generated and saved in:
  /host/var/tmp/sosreport-rhel81beta-12345678-2019-10-29-pmgjncg.tar.xz
The checksum is: c4e1fd3ee45f78a17afb4e45a05842ed
Please send this file to your support representative.
```

Tenga en cuenta que el comando **sosreport** guarda el informe en el host (**/host/var/tmp/sosreport-<ID>**).

6. Instalar un paquete de software dentro del contenedor, para añadir herramientas que no están ya en el contenedor. Por ejemplo, para obtener un volcado del núcleo de un proceso en ejecución en el host, instale los paquetes **procps** y **gcore**, utilice **ps** para obtener el ID del proceso de un demonio en ejecución y, a continuación, utilice **gcore** para obtener un volcado del núcleo:

```
bash-4.4# yum install procps gdb -y
bash-4.4# ps -ef | grep chronyd
994        809     1  0 Oct28 ?        00:00:00 /usr/sbin/chronyd
bash-4.4# gcore -o /host/tmp/chronyd.core 809
Missing separate debuginfo for target:/usr/sbin/chronyd
Try: dnf --enablerepo='*debug*' install /usr/lib/debug/.build-
id/96/0789a8a3bf28932b093e94b816be379f16a56a.debug
...
Saved corefile /host/tmp/chronyd.core.809
[Inferior 1 (process 809) detached]
```

7. Para salir del contenedor y volver al host, escribe **exit**. El archivo se guarda en **/host/tmp/chronyd.core.809** y está disponible desde **/tmp/chronyd.core.809** en el host.

8. Para eliminar el contenedor toolbox-root, escriba:

```
# podman rm toolbox-root
```

Puedes cambiar el nombre del registro, de la imagen o del contenedor utilizado por toolbox añadiendo lo siguiente:

- REGISTRY: Cambia el registro del que se extrae la imagen de la caja de herramientas. Por ejemplo **REGISTRY=registry.example.com**
- IMAGE: Cambia la imagen que se utiliza. Por ejemplo, **IMAGE=mysupport-tools**
- TOOLBOX_NAME: Cambia el nombre asignado al contenedor en ejecución. Por ejemplo, **TOOLBOX_NAME=mytoolbox**

La próxima vez que ejecute **toolbox**, se utilizarán los nuevos valores del archivo **.toolboxrc**.

6.1.1. Privilegios de apertura para el anfitrión

Cuando se ejecutan otros comandos desde el contenedor **support-tools** (o cualquier contenedor privilegiado), pueden comportarse de manera diferente que cuando se ejecutan en un contenedor no privilegiado. Aunque **sosreport** puede saber cuándo se está ejecutando en un contenedor, otros comandos necesitan que se les diga que actúen en el sistema anfitrión (el directorio **/host**). Aquí hay ejemplos de funciones que pueden o no estar abiertas al host desde un contenedor:

- **Privileges**: Un contenedor privilegiado (**--privileged**) ejecuta aplicaciones como usuario root en el host por defecto. El contenedor tiene esta capacidad porque se ejecuta con un contexto de seguridad SELinux **unconfined_t**. Así que puede, por ejemplo, eliminar archivos y directorios montados desde el host que son propiedad del usuario root.
- **Process tables**: A diferencia de un contenedor normal que sólo ve los procesos que se ejecutan dentro del contenedor, la ejecución de un comando **ps -e** dentro de un contenedor privilegiado (con **--pid=host** configurado) le permite ver todos los procesos que se ejecutan en el host. Puedes pasar un ID de proceso del host a los comandos que se ejecutan en el contenedor privilegiado (por ejemplo, **kill <PID>**). Sin embargo, con algunos comandos pueden surgir problemas de permisos cuando intentan acceder a los procesos desde el contenedor.
- **Network interfaces**: Por defecto, un contenedor sólo tiene una interfaz de red externa y una interfaz de red loopback. Con las interfaces de red abiertas al host (**--net=host**), puedes acceder a esas interfaces de red directamente desde el contenedor.

- **Inter-process communications**: La instalación IPC en el host es accesible desde el contenedor privilegiado. Puedes ejecutar comandos como **ipcs** para ver información sobre colas de mensajes activas, segmentos de memoria compartida y conjuntos de semáforos en el host.

6.2. EJECUCIÓN DE CONTENEDORES CON ETIQUETAS DE EJECUCIÓN

Algunas imágenes de Red Hat incluyen etiquetas que proporcionan líneas de comando preestablecidas para trabajar con esas imágenes. Usando el comando **podman container runlabel <label>**, puede decirle a **podman** que ejecute el comando definido en ese **<label>** para la imagen. Las etiquetas de ejecución existentes incluyen:

- **install**: Configura el sistema anfitrión antes de ejecutar la imagen. Típicamente, esto resulta en la creación de archivos y directorios en el host a los que el contenedor puede acceder cuando se ejecuta más tarde.

- **run**: Identifica las opciones de la línea de comandos de podman que se utilizarán al ejecutar el contenedor. Normalmente, las opciones abrirán privilegios en el host y montarán el contenido del host que el contenedor necesita para permanecer permanentemente en el host.

- **uninstall**: Limpia el sistema anfitrión después de que haya terminado de ejecutar el contenedor.

Las imágenes de Red Hat que tienen una o más etiquetas de ejecución incluyen las imágenes **rsyslog** y **support-tools**. El siguiente procedimiento ilustra cómo utilizar esas imágenes.

6.2.1. Ejecución de rsyslog con runlabels

La imagen de contenedor **rhel8/rsyslog** está hecha para ejecutar una versión en contenedor del demonio **rsyslogd**. Dentro de la imagen **rsyslog** se encuentran las etiquetas de ejecución **install**, **run** y **uninstall**. El siguiente procedimiento te lleva a instalar, ejecutar y desinstalar la imagen **rsyslog**:

Procedimiento

1. Tire de la imagen de **rsyslog**:

```
# podman pull registry.redhat.io/rhel8/rsyslog
```

2. Mostrar (pero no ejecutar) la etiqueta de ejecución **install** para **rsyslog**:

```
# podman container runlabel install --display rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/install.sh
```

   Esto muestra que el comando abrirá privilegios al host, montará el sistema de archivos raíz del host en **/host** en el contenedor, y ejecutará un script **install.sh**.

3. Ejecute la etiqueta de ejecución **install** para **rsyslog**:

```
# podman container runlabel install rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/install.sh
Creating directory at /host//etc/pki/rsyslog
```

```
Creating directory at /host//etc/rsyslog.d
Installing file at /host//etc/rsyslog.conf
Installing file at /host//etc/sysconfig/rsyslog
Installing file at /host//etc/logrotate.d/syslog
```

Esto crea archivos en el sistema anfitrión que la imagen **rsyslog** utilizará posteriormente.

4. Muestra la etiqueta de ejecución **run** para **rsyslog**:

```
# podman container runlabel run --display rhel8/rsyslog
command: podman run -d --privileged --name rsyslog --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -e IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e
NAME=rsyslog --restart=always registry.redhat.io/rhel8/rsyslog:latest /bin/rsyslog.sh
```

Esto muestra que el comando abre privilegios al host y monta archivos y directorios específicos del host dentro del contenedor, cuando lanza el contenedor **rsyslog** para ejecutar el demonio **rsyslogd**.

5. Ejecute la etiqueta de ejecución **run** para **rsyslog**:

```
# podman container runlabel run rhel8/rsyslog
command: podman run -d --privileged --name rsyslog --net=host --pid=host -v
/etc/pki/rsyslog:/etc/pki/rsyslog -v /etc/rsyslog.conf:/etc/rsyslog.conf -v
/etc/sysconfig/rsyslog:/etc/sysconfig/rsyslog -v /etc/rsyslog.d:/etc/rsyslog.d -v /var/log:/var/log
-v /var/lib/rsyslog:/var/lib/rsyslog -v /run:/run -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -e IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e
NAME=rsyslog --restart=always registry.redhat.io/rhel8/rsyslog:latest /bin/rsyslog.sh
28a0d719ff179adcea81eb63cc90fcd09f1755d5edb121399068a4ea59bd0f53
```

El contenedor **rsyslog** abre privilegios, monta lo que necesita del host y ejecuta el demonio **rsyslogd** en segundo plano (**-d**). El demonio **rsyslogd** comienza a recopilar mensajes de registro y a dirigir los mensajes a los archivos del directorio **/var/log**.

6. Muestra la etiqueta de ejecución **uninstall** para **rsyslog**:

```
# podman container runlabel uninstall --display rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/uninstall.sh
```

7. Ejecute la etiqueta de ejecución **uninstall** para **rsyslog**:

```
# podman container runlabel uninstall rhel8/rsyslog
command: podman run --rm --privileged -v /:/host -e HOST=/host -e
IMAGE=registry.redhat.io/rhel8/rsyslog:latest -e NAME=rsyslog
registry.redhat.io/rhel8/rsyslog:latest /bin/uninstall.sh
```

En este caso, el script **uninstall.sh** sólo elimina el archivo **/etc/logrotate.d/syslog**. Tenga en cuenta que no limpia los archivos de configuración.

6.2.2. Ejecutar support-tools con runlabels

La imagen del contenedor **rhel8/support-tools** está hecha para ejecutar herramientas como **sosreport** y **sos-collector** para ayudarle a analizar su sistema anfitrión. Para simplificar la ejecución de la imagen **support-tools**, ésta incluye una etiqueta de ejecución **run**. El siguiente procedimiento describe cómo ejecutar la imagen **support-tools**:

Procedimiento

1. Tire de la imagen de **support-tools**:

```
# podman pull registry.redhat.io/rhel8/support-tools
```

2. Mostrar (pero no ejecutar) la etiqueta de ejecución **run** para **support-tools**:

```
# podman container runlabel run --display rhel8/support-tools
command: podman run -it --name support-tools --privileged --ipc=host --net=host --pid=host -
e HOST=/host -e NAME=support-tools -e IMAGE=registry.redhat.io/rhel8/support-tools:latest
-v /run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -v /:/host registry.redhat.io/rhel8/support-tools:latest
```

   Esto muestra que el comando monta directorios y abre privilegios y espacios de nombres (ipc, net y pid) al sistema anfitrión. Asigna el sistema de archivos raíz del host al directorio /**host** en el contenedor.

3. Ejecute la etiqueta de ejecución **run** para las herramientas de apoyo:

```
# podman container runlabel run rhel8/support-tools
command: podman run -it --name support-tools --privileged --ipc=host --net=host --pid=host -
e HOST=/host -e NAME=support-tools -e IMAGE=registry.redhat.io/rhel8/support-tools:latest
-v /run:/run -v /var/log:/var/log -v /etc/machine-id:/etc/machine-id -v
/etc/localtime:/etc/localtime -v /:/host registry.redhat.io/rhel8/support-tools:latest
bash-4.4#
```

   Esto abre un shell bash dentro del contenedor **support-tools**. Ahora puede ejecutar informes o herramientas de depuración contra el sistema anfitrión (/**host**).

4. Para salir del contenedor y volver al host, escriba **exit**.

```
# exit
```

CAPÍTULO 7. PORTAR CONTENEDORES A OPENSHIFT USANDO PODMAN

Este capítulo describe cómo generar descripciones portátiles de contenedores y pods utilizando el formato YAML (\ "YAML Ain't Markup Language"). El YAML es un formato de texto utilizado para describir los datos de configuración.

Los archivos YAML son:

- Legible.
- Fácil de generar.
- Portátil entre entornos (por ejemplo entre RHEL y OpenShift).
- Es portátil entre los lenguajes de programación.
- Es cómodo de usar (no es necesario añadir todos los parámetros a la línea de comandos).

Razones para utilizar archivos YAML:

1. Puede volver a ejecutar un conjunto local orquestado de contenedores y vainas con una entrada mínima requerida que puede ser útil para el desarrollo iterativo.
2. Puede ejecutar los mismos contenedores y pods en otra máquina. Por ejemplo, para ejecutar una aplicación en un entorno OpenShift y asegurarse de que la aplicación funciona correctamente. Puede utilizar el comando **podman generate kube** para generar un archivo YAML de Kubernetes. A continuación, puede utilizar el comando **podman play** para probar la creación de pods y contenedores en su sistema local antes de transferir los archivos YAML generados al entorno Kubernetes u OpenShift. Utilizando el comando **podman play**, también puede recrear pods y contenedores creados originalmente en entornos OpenShift o Kubernetes.

7.1. GENERACIÓN DE UN ARCHIVO YAML DE KUBERNETES CON PODMAN

Este procedimiento describe cómo crear un pod con un contenedor y generar el archivo YAML de Kubernetes utilizando el comando **podman generate kube**.

Requisitos previos

- El pod ha sido creado. Para más detalles, consulte Creación de pods.

Procedimiento

1. Enumerar todos los pods y contenedores asociados a ellos:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                  COMMAND    CREATED
STATUS                  PORTS  NAMES              POD
5df5c48fea87  registry.access.redhat.com/ubi8/ubi:latest  /bin/bash  Less than a second ago
Up Less than a second ago         myubi              223df6b390b4
3afdcd93de3e  k8s.gcr.io/pause:3.1                        Less than a second ago  Up Less
than a second ago         223df6b390b4-infra  223df6b390b4
```

2. Utilice el nombre o el ID del pod para generar el archivo YAML de Kubernetes:

```
$ podman generate kube mypod > mypod.yaml
```

Tenga en cuenta que el comando **podman generate** no refleja los volúmenes lógicos de Logical Volume Manager (LVM) ni los volúmenes físicos que puedan estar adjuntos al contenedor.

3. Muestra el archivo **mypod.yaml**:

```
$ cat mypod.yaml
# Generation of Kubernetes YAML is still under development!
#
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-1.6.4
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-06-09T10:31:56Z"
  labels:
app: mypod
  name: mypod
spec:
  containers:
  - command:
        - /bin/bash
        env:
        - name: PATH
            value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
        - name: TERM
            value: xterm
        - name: HOSTNAME
        - name: container
            value: oci
        image: registry.access.redhat.com/ubi8/ubi:latest
        name: myubi
        resources: {}
        securityContext:
            allowPrivilegeEscalation: true
            capabilities: {}
            privileged: false
            readOnlyRootFilesystem: false
        tty: true
        workingDir: /
status: {}
```

4. Para detener la vaina **mypod**:

```
$ podman pod stop mypod
```

5. Enumerar todos los pods y contenedores asociados a ellos:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                           COMMAND    CREATED          STATUS
```

```
PORTS   NAMES               POD ID        PODNAME
12fc1ada3d4f  registry.redhat.io/ubi8/ubi:latest  /bin/bash  About a minute ago  Exited (0) 7
seconds ago          myubi               8a4e6527ac9d  mypod

8bb1daaf81fe  k8s.gcr.io/pause:3.2                        About a minute ago  Exited (0) 7 seconds
ago          8a4e6527ac9d-infra  8a4e6527ac9d  mypod
```

Aquí, el pod **mypod** y el contenedor **myubi** están en estado "Exited".

6. Para retirar la vaina **mypod**:

```
$ podman pod rm mypod
8a4e6527ac9d2276e8a6b9c2670608866dbcb5da3efbd06f70ec2ecc88e247eb
```

Tenga en cuenta que al eliminar el pod se eliminan automáticamente todos los contenedores que hay en su interior.

7. Comprobar que se han retirado todos los contenedores y vainas:

```
$ podman ps
$ podman pod ps
```

Recursos adicionales

- La página de manual **podman-generate-kube**.
- Podman: Managing pods and containers in a local container runtime , por Brent Baude.

7.2. GENERACIÓN DE UN ARCHIVO YAML DE KUBERNETES EN EL ENTORNO DE OPENSHIFT

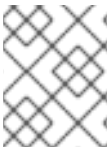
En el entorno OpenShift, utilice el comando **oc create** para generar los archivos YAML que describen su aplicación.

Procedimiento

- Genere el archivo YAML para su aplicación **myapp**:

```
$ oc create myapp --image=me/myapp:v1 -o yaml --dry-run > myapp.yaml
```

El comando **oc create** crea y ejecuta la imagen **myapp**. El objeto se imprime utilizando la opción **--dry-run** y se redirige al archivo de salida **myapp.yaml**.



NOTA

En el entorno de Kubernetes, puede utilizar el comando **kubectl create** con las mismas banderas.

7.3. INICIAR CONTENEDORES Y PODS CON PODMAN

Con los archivos YAML generados, puede iniciar automáticamente contenedores y pods en cualquier entorno. Tenga en cuenta que los archivos YAML no deben ser generados por Podman. El comando **podman play kube** le permite recrear pods y contenedores basándose en el archivo de entrada YAML.

Procedimiento

1. Cree el pod y el contenedor desde el archivo **mypod.yaml**:

```
$ podman play kube mypod.yaml
Pod:
b8c5b99ba846ccff76c3ef257e5761c2d8a5ca4d7ffa3880531aec79c0dacb22
Container:
848179395ebd33dd91d14ffbde7ae273158d9695a081468f487af4e356888ece
```

2. Enumerar todas las vainas:

```
$ podman pod ps
POD ID        NAME    STATUS    CREATED         # OF CONTAINERS   INFRA ID
b8c5b99ba846  mypod   Running   19 seconds ago  2                 aa4220eaf4bb
```

3. Enumerar todos los pods y contenedores asociados a ellos:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                  COMMAND    CREATED            STATUS
PORTS  NAMES              POD
848179395ebd  registry.access.redhat.com/ubi8/ubi:latest  /bin/bash  About a minute ago  Up
About a minute ago        myubi               b8c5b99ba846
aa4220eaf4bb  k8s.gcr.io/pause:3.1                        About a minute ago  Up About a
minute ago        b8c5b99ba846-infra  b8c5b99ba846
```

Los ID de los pods del comando **podman ps** coinciden con los ID de los pods del comando **podman pod ps**.

Recursos adicionales

- La página de manual **podman-play-kube**.
- Podman ahora puede facilitar la transición a Kubernetes y CRI-O artículo de Brent Baude.

7.4. INICIAR CONTENEDORES Y PODS EN EL ENTORNO OPENSHIFT

Puede utilizar el comando **oc create** para crear pods y contenedores en el entorno OpenShift.

Procedimiento

- Cree un pod a partir del archivo YAML en el entorno OpenShift:

```
$ oc create -f mypod.yaml
```

NOTA

En el entorno de Kubernetes, puede utilizar el comando **kubectl create** con las mismas banderas.

CAPÍTULO 8. PORTAR CONTENEDORES A SYSTEMD USANDO PODMAN

Podman (Pod Manager) es un motor de contenedores con todas las funciones que es una herramienta simple sin demonio. Podman proporciona una línea de comandos comparable a Docker-CLI que facilita la transición desde otros motores de contenedores y permite la gestión de vainas, contenedores e imágenes. No fue diseñado originalmente para poner en marcha un sistema Linux completo o gestionar servicios para cosas como el orden de arranque, la comprobación de dependencias y la recuperación de servicios fallidos. Ese es el trabajo de un sistema de inicialización completo como systemd. Red Hat se ha convertido en un líder en la integración de contenedores con systemd, de manera que los contenedores con formato OCI y Docker construidos por Podman pueden ser gestionados de la misma manera que otros servicios y características en un sistema Linux. Puedes utilizar el servicio de inicialización systemd para trabajar con pods y contenedores. Puede utilizar el comando **podman generate systemd** para generar un archivo de unidad systemd para contenedores y pods.

Con los archivos de unidad systemd, puedes:

- Configurar un contenedor o pod para que se inicie como un servicio systemd.
- Definir el orden de ejecución del servicio en contenedor y comprobar las dependencias (por ejemplo, asegurarse de que otro servicio se está ejecutando, un archivo está disponible o un recurso está montado).
- Controla el estado del sistema systemd mediante el comando **systemctl**.

Este capítulo le proporciona información sobre cómo generar descripciones portátiles de contenedores y pods utilizando archivos de unidad systemd.

8.1. HABILITACIÓN DE LOS SERVICIOS SYSTEMD

Al habilitar el servicio, tiene diferentes opciones.

Procedimiento

- Habilitar el servicio:
  - Para habilitar un servicio al inicio del sistema, sin importar si el usuario está conectado o no, introduzca:

    ```
    # systemctl enable <service>
    ```

    Tienes que copiar los archivos de la unidad systemd en el directorio **/etc/systemd/system**.
  - Para iniciar un servicio al iniciar la sesión del usuario y detenerlo al cerrarla, introduzca:

    ```
    $ systemctl --user enable <service>
    ```

    Tienes que copiar los archivos de la unidad systemd en el directorio **$HOME/.config/systemd/user**.
  - Para permitir que los usuarios inicien un servicio al comienzo del sistema y que persista sobre los cierres de sesión, introduzca:

    ```
    # loginctl enable-linger <username>
    ```

Recursos adicionales

- Para más información sobre los comandos **systemctl** y **loginctl**, introduzca **man systemctl** o **man loginctl**, respectivamente.
- Para obtener más información sobre la configuración de los servicios con systemd, consulte el capítulo de la guía Configuración de los ajustes básicos del sistema llamado Gestión de los servicios con systemd.

8.2. GENERACIÓN DE UN ARCHIVO DE UNIDAD SYSTEMD USANDO PODMAN

Podman permite a systemd controlar y gestionar los procesos de los contenedores. Puede generar un archivo de unidad systemd para los contenedores y pods existentes utilizando el comando **podman generate systemd**. Se recomienda utilizar **podman generate systemd** porque los archivos de unidades generados cambian con frecuencia (a través de las actualizaciones de Podman) y el **podman generate systemd** asegura que se obtiene la última versión de los archivos de unidades.

Procedimiento

1. Cree un contenedor (por ejemplo **myubi**):

```
$ podman create -d --name myubi registry.access.redhat.com/ubi8:latest top
0280afe98bb75a5c5e713b28de4b7c5cb49f156f1cce4a208f13fee2f75cb453
```

2. Utilice el nombre o el ID del contenedor para generar el archivo de unidad systemd y dirigirlo al archivo **~/.config/systemd/user/container-myubi.service**:

```
$ podman generate systemd --name myubi > ~/.config/systemd/user/container-myubi.service
```

Pasos de verificación

- Para mostrar el contenido del archivo de unidad systemd generado, introduzca:

```
$ cat ~/.config/systemd/user/container-myubi.service
# container-myubi.service
# autogenerated by Podman 2.0.0
# Tue Aug 11 10:51:04 CEST 2020

[Unit]
Description=Podman container-myubi.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
ExecStart=/usr/bin/podman start myubi
ExecStop=/usr/bin/podman stop -t 10 myubi
ExecStopPost=/usr/bin/podman stop -t 10 myubi
PIDFile=/run/user/1000/containers/overlay-
containers/0280afe98bb75a5c5e713b28de4b7c5cb49f156f1cce4a208f13fee2f75cb453/userdat
a/conmon.pid
```

```
KillMode=none
Type=forking

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

- La línea **Restart=on-failure** establece la política de reinicio e indica a systemd que se reinicie cuando el servicio no pueda iniciarse o detenerse limpiamente, o cuando el proceso salga de forma distinta a cero.
- La línea **ExecStart** describe cómo iniciamos el contenedor.
- La línea **ExecStop** describe cómo detenemos y retiramos el contenedor.

Recursos adicionales

- Ejecución de contenedores con Podman y servicios systemd compartibles artículo de Valentin Rothberg.

8.3. GENERACIÓN AUTOMÁTICA DE UN ARCHIVO DE UNIDAD SYSTEMD USANDO PODMAN

Por defecto, Podman genera un archivo de unidad para los contenedores o pods existentes. Puede generar archivos de unidad systemd más portables utilizando la opción **podman generate systemd --new**. La bandera **--new** indica a Podman que genere archivos de unidad que creen, inicien y eliminen contenedores.

Procedimiento

1. Extraiga la imagen que desea utilizar en su sistema. Por ejemplo, para extraer la imagen **busybox**:

```
# podman pull busybox:latest
```

2. Enumera todas las imágenes disponibles en tu sistema:

```
# podman images
REPOSITORY                TAG     IMAGE ID      CREATED      SIZE
docker.io/library/busybox  latest  c7c37e472d31  3 weeks ago  1.45 MB
```

3. Cree el contenedor **busybox**:

```
# podman create --name busybox busybox:latest
1e12cf95e305435c0001fa7d4a14cf1d52f737c1118328937028c0bd2fdec5ca
```

4. Para verificar que el contenedor ha sido creado, liste todos los contenedores:

```
# podman ps -a
CONTAINER ID  IMAGE                           COMMAND  CREATED        STATUS    PORTS
NAMES
1e12cf95e305  docker.io/library/busybox:latest  sh       7 seconds ago  Created         busybox
```

5. Generar un archivo de unidad systemd para el contenedor **busybox**:

```
# podman generate systemd --new --files --name busybox
/root/container-busybox.service
```

6. Muestra el contenido del archivo de la unidad systemd generado en **container-busybox.service**:

```
# vim container-busybox.services

# container-busybox.service
# autogenerated by Podman 2.0.0-rc7
# Mon Jul 27 11:06:32 CEST 2020

[Unit]
Description=Podman container-busybox.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
ExecStartPre=/usr/bin/rm -f %t/container-busybox.pid %t/container-busybox.ctr-id
ExecStart=/usr/bin/podman run --conmon-pidfile %t/container-busybox.pid --cidfile
%t/container-busybox.ctr-id --cgroups=no-conmon -d --replace --name busybox
busybox:latest
ExecStop=/usr/bin/podman stop --ignore --cidfile %t/container-busybox.ctr-id -t 10
ExecStopPost=/usr/bin/podman rm --ignore -f --cidfile %t/container-busybox.ctr-id
PIDFile=%t/container-busybox.pid
KillMode=none
Type=forking

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

Tenga en cuenta que los archivos de unidad generados con la opción **--new** no esperan que existan contenedores y pods. Por lo tanto, ejecutan el comando **podman run** al iniciar el servicio (ver la línea **ExecStart**) en lugar del comando **podman start**. Por ejemplo, vea la Sección 7.2. Generación de un archivo de unidad systemd usando Podman .

- El comando **podman run** utiliza las siguientes opciones de línea de comandos:
  - La opción **--conmon-pidfile** apunta a una ruta para almacenar el ID del proceso **conmon** que se ejecuta en el host. El proceso **conmon** termina con el mismo estado de salida que el contenedor, lo que permite a systemd informar del estado correcto del servicio y reiniciar el contenedor si es necesario.
  - La opción **--cidfile** apunta a la ruta que almacena el ID del contenedor.
  - El **%t** es la ruta de acceso a la raíz del directorio en tiempo de ejecución, por ejemplo **/run/user/$UserID**.
  - El **%n** es el nombre completo del servicio.

7. Copie los archivos de la unidad en **/usr/lib/systemd/system** para instalarlos como usuario root:

```
# cp -Z container-busybox.service  /usr/lib/systemd/system
```

```
Created symlink /etc/systemd/system/multi-user.target.wants/container-busybox.service
/usr/lib/systemd/system/container-busybox.service.
Created symlink /etc/systemd/system/default.target.wants/container-busybox.service →
/usr/lib/systemd/system/container-busybox.service.
```

Recursos adicionales

- Integración mejorada de Systemd con Podman 2. 0 artículo de Valentin Rothberg y Dan Walsh.
- Para obtener más información sobre la configuración de los servicios con systemd, consulte el capítulo de la guía Configuración de los ajustes básicos del sistema llamado Gestión de los servicios con systemd.

8.4. ARRANQUE AUTOMÁTICO DE CONTENEDORES CON SYSTEMD

Puedes controlar el estado del sistema systemd y del gestor de servicios utilizando el comando **systemctl**. Esta sección muestra el procedimiento general sobre cómo habilitar, iniciar y detener el servicio como usuario no root. Para instalar el servicio como usuario root, omita la opción **--user**.

Procedimiento

1. Recargar la configuración del gestor systemd:

```
# systemctl --user daemon-reload
```

2. Habilitar el servicio **container.service** e iniciarlo en el momento del arranque:

```
# systemctl --user enable container.service
```

3. Para iniciar el servicio inmediatamente:

```
# systemctl --user start container.service
```

4. Comprueba el estado del servicio:

```
$ systemctl --user status container.service
● container.service - Podman container.service
   Loaded: loaded (/home/user/.config/systemd/user/container.service; enabled; vendor
preset: enabled)
   Active: active (running) since Wed 2020-09-16 11:56:57 CEST; 8s ago
     Docs: man:podman-generate-systemd(1)
  Process: 80602 ExecStart=/usr/bin/podman run --conmon-pidfile
//run/user/1000/container.service-pid --cidfile //run/user/1000/container.service-cid -d ubi8-
minimal:>
  Process: 80601 ExecStartPre=/usr/bin/rm -f //run/user/1000/container.service-pid
//run/user/1000/container.service-cid (code=exited, status=0/SUCCESS)
 Main PID: 80617 (conmon)
   CGroup: /user.slice/user-1000.slice/user@1000.service/container.service
           ├─ 2870 /usr/bin/podman
           ├─80612 /usr/bin/slirp4netns --disable-host-loopback --mtu 65520 --enable-sandbox -
-enable-seccomp -c -e 3 -r 4 --netns-type=path /run/user/1000/netns/cni->
           ├─80614 /usr/bin/fuse-overlayfs -o
lowerdir=/home/user/.local/share/containers/storage/overlay/l/YJSPGXM2OCDZPLMLXJOW3N
```

```
RF6Q:/home/user/.local/share/contain>
          ├─80617 /usr/bin/conmon --api-version 1 -c
cbc75d6031508dfd3d78a74a03e4ace1732b51223e72a2ce4aa3bfe10a78e4fa -u
cbc75d6031508dfd3d78a74a03e4ace1732b51223e72>
          └─cbc75d6031508dfd3d78a74a03e4ace1732b51223e72a2ce4aa3bfe10a78e4fa
            └─80626 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1d
```

Puede comprobar si el servicio está activado mediante el comando **systemctl is-enabled container.service**.

Pasos de verificación

- Enumera los contenedores que se están ejecutando o que han salido:

```
# podman ps
CONTAINER ID  IMAGE                                COMMAND  CREATED        STATUS
PORTS  NAMES
f20988d59920  registry.access.redhat.com/ubi8-minimal:latest  top      12 seconds ago  Up 11
seconds ago          funny_zhukovsky
```

NOTA

Para detener **container.service**, introduzca:

```
# systemctl --user stop container.service
```

Recursos adicionales

- Para más información sobre el comando **systemctl**, escriba **man systemctl**.
- Para más información, consulta el artículo Running containers with Podman and shareable systemd services de Valentin Rothberg.
- Para obtener más información sobre la configuración de los servicios con systemd, consulte el capítulo de la guía Configuración de los ajustes básicos del sistema llamado Gestión de los servicios con systemd.

8.5. ARRANQUE AUTOMÁTICO DE PODS MEDIANTE SYSTEMD

Puede iniciar varios contenedores como servicios systemd. Tenga en cuenta que el comando **systemctl** sólo debe utilizarse en el pod y no debe iniciar o detener contenedores individualmente a través de **systemctl**, ya que son gestionados por el servicio del pod junto con el infra-contenedor interno.

Procedimiento

1. Cree un pod vacío, por ejemplo llamado **systemd-pod**:

```
$ podman pod create --name systemd-pod
11d4646ba41b1fffa51c108cbdf97cfab3213f7bd9b3e1ca52fe81b90fed5577
```

2. Enumerar todas las vainas:

```
$ podman pod ps
```

```
POD ID        NAME         STATUS   CREATED         # OF CONTAINERS  INFRA ID
11d4646ba41b  systemd-pod  Created  40 seconds ago  1                8a428b257111
11d4646ba41b1fffa51c108cbdf97cfab3213f7bd9b3e1ca52fe81b90fed5577
```

3. Cree dos contenedores en el pod vacío. Por ejemplo, para crear **container0** y **container1** en **systemd-pod**:

```
$ podman create --pod systemd-pod --name container0 registry.access.redhat.com/ubi8 top
$ podman create --pod systemd-pod --name container1 registry.access.redhat.com/ubi8 top
```

4. Enumerar todos los pods y contenedores asociados a ellos:

```
$ podman ps -a --pod
CONTAINER ID  IMAGE                                COMMAND  CREATED        STATUS
PORTS   NAMES              POD ID        PODNAME
24666f47d9b2  registry.access.redhat.com/ubi8:latest  top      3 minutes ago  Created
container0         3130f724e229  systemd-pod
56eb1bf0cdfe  k8s.gcr.io/pause:3.2                  4 minutes ago  Created
3130f724e229-infra  3130f724e229  systemd-pod
62118d170e43  registry.access.redhat.com/ubi8:latest  top      3 seconds ago  Created
container1         3130f724e229  systemd-pod
```

5. Generar el archivo de unidad systemd para el nuevo pod:

```
$ podman generate systemd --files --name systemd-pod
/home/user1/pod-systemd-pod.service
/home/user1/container-container0.service
/home/user1/container-container1.service
```

Observe que se generan tres archivos de unidad systemd, uno para el pod **systemd-pod** y dos para los contenedores **container0** y **container1**.

6. Mostrar **pod-systemd-pod.service** archivo de la unidad:

```
$ cat pod-systemd-pod.service
# pod-systemd-pod.service
# autogenerated by Podman 2.0.3
# Tue Jul 28 14:00:46 EDT 2020


[Unit]
Description=Podman pod-systemd-pod.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target
Requires=container-container0.service container-container1.service
Before=container-container0.service container-container1.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
ExecStart=/usr/bin/podman start c852fbaba568-infra
ExecStop=/usr/bin/podman stop -t 10 c852fbaba568-infra
ExecStopPost=/usr/bin/podman stop -t 10 c852fbaba568-infra
PIDFile=/run/user/1000/containers/overlay-
```

```
containers/a7ff86382608add27a03ac2166d5d0164199f01eadf80b68b06a406c195105fc/userd
ata/conmon.pid
KillMode=none
Type=forking

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

- La línea **Requires** en la sección **[Unit]** define las dependencias de los archivos de unidad **container-container0.service** y **container-container1.service**. Ambos archivos de unidad se activarán.

- Las líneas **ExecStart** y **ExecStop** de la sección **[Service]** inician y detienen el infra-contenedor, respectivamente.

7. Mostrar **container-container0.service** archivo de la unidad:

```
$ cat container-container0.service
# container-container0.service
# autogenerated by Podman 2.0.3
# Tue Jul 28 14:00:46 EDT 2020

[Unit]
Description=Podman container-container0.service
Documentation=man:podman-generate-systemd(1)
Wants=network.target
After=network-online.target
BindsTo=pod-systemd-pod.service
After=pod-systemd-pod.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
ExecStart=/usr/bin/podman start container0
ExecStop=/usr/bin/podman stop -t 10 container0
ExecStopPost=/usr/bin/podman stop -t 10 container0
PIDFile=/run/user/1000/containers/overlay-
containers/12e85378f2854b8283f791974494a02aa6c92630d76d1050237839b61508a008/user
data/conmon.pid
KillMode=none
Type=forking

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

- La línea **BindsTo** de la sección **[Unit]** define la dependencia del archivo de unidad **pod-systemd-pod.service**

- Las líneas **ExecStart** y **ExecStop** de la sección **[Service]** inician y detienen el **container0** respectivamente.

8. Mostrar **container-container1.service** archivo de la unidad:

```
$ cat contenedor-contenedor1.service
```

9. Copie todos los archivos generados en **$HOME/.config/systemd/user** para instalarlos como usuario no root:

```
$ cp pod-systemd-pod.service container-container0.service container-container1.service
$HOME/.config/systemd/user
```

10. Habilitar el servicio e iniciarlo al iniciar la sesión del usuario:

```
$ systemctl enable --user pod-systemd-pod.service
Created symlink /home/user1/.config/systemd/user/multi-user.target.wants/pod-systemd-
pod.service → /home/user1/.config/systemd/user/pod-systemd-pod.service.
Created symlink /home/user1/.config/systemd/user/default.target.wants/pod-systemd-
pod.service → /home/user1/.config/systemd/user/pod-systemd-pod.service.
```

Tenga en cuenta que el servicio se detiene al cerrar la sesión del usuario.

Pasos de verificación

- Comprueba si el servicio está activado:

```
$ systemctl is-enabled pod-systemd-pod.service
enabled
```

Recursos adicionales

- Para más información sobre el comando **podman create**, escriba **man podman-create**.
- Para más información sobre el comando **podman generate systemd**, escriba **man podman-generate-systemd**.
- Para más información sobre el comando **systemctl**, escriba **man systemctl**.
- Para más información, consulta el artículo Running containers with Podman and shareable systemd services de Valentin Rothberg.
- Para obtener más información sobre la configuración de los servicios con systemd, consulte el capítulo de la guía Configuración de los ajustes básicos del sistema llamado Gestión de los servicios con systemd.

CAPÍTULO 9. CREACIÓN DE IMÁGENES DE CONTENEDORES CON BUILDAH

El comando **buildah** permite crear imágenes de contenedores a partir de un contenedor en funcionamiento, un archivo Docker o desde cero. Las imágenes resultantes son compatibles con OCI, por lo que funcionarán en cualquier tiempo de ejecución de contenedores que cumpla con la especificación de tiempo de ejecución de OCI (como Docker y CRI-O).

Esta sección describe cómo utilizar el comando **buildah** para crear y trabajar con contenedores e imágenes de contenedores.

9.1. COMPRENSIÓN DE BUILDAH

El uso de Buildah se diferencia de la construcción de imágenes con el comando **docker** en los siguientes aspectos:

- **No Daemon!**: ¡Evita el demonio Docker! Así que no se necesita ningún tiempo de ejecución del contenedor (Docker, CRI-O, u otro) para utilizar Buildah.
- **Base image or scratch** Le permite no sólo construir una imagen basada en otro contenedor, sino que también le permite empezar con una imagen vacía (desde cero).
- **Build tools external**: No incluye herramientas de construcción dentro de la propia imagen. Como resultado, Buildah:
  - Reduce el tamaño de las imágenes que construye
  - Hace que la imagen sea más segura al no tener el software utilizado para construir el contenedor (como gcc, make y yum) dentro de la imagen resultante.
  - Crea imágenes que requieren menos recursos para transportarlas (porque son más pequeñas).

Buildah es capaz de operar sin Docker u otros tiempos de ejecución de contenedores almacenando los datos por separado e incluyendo características que te permiten no sólo construir imágenes, sino también ejecutar esas imágenes como contenedores. Por defecto, Buildah almacena las imágenes en un área identificada como **containers-storage** (/var/lib/containers).

NOTA

La ubicación de almacenamiento de contenedores que el comando **buildah** utiliza por defecto es el mismo lugar que el motor de contenedores CRI-O utiliza para almacenar las copias locales de las imágenes. Por lo tanto, las imágenes extraídas de un registro por CRI-O o Buildah, o confirmadas por el comando **buildah**, se almacenarán en la misma estructura de directorios. Actualmente, sin embargo, CRI-O y Buildah no pueden compartir contenedores, aunque sí pueden compartir imágenes.

Hay más de una docena de opciones para utilizar con el comando **buildah**. Algunas de las principales actividades que puede realizar con el comando **buildah** incluyen:

- **Build a container from a Dockerfile** Utiliza un Dockerfile para construir una nueva imagen de contenedor (**buildah bud**).

- **Build a container from another image or scratch** Construye un nuevo contenedor, partiendo de una imagen base existente (**buildah from <imagename>**) o desde cero (**buildah from scratch**)
- **Inspecting a container or image**: Ver los metadatos asociados al contenedor o a la imagen (**buildah inspect**)
- **Mount a container**: Montar el sistema de archivos raíz de un contenedor para añadir o cambiar el contenido (**buildah mount**).
- **Create a new container layer**: Utiliza el contenido actualizado del sistema de archivos raíz de un contenedor como capa del sistema de archivos para confirmar el contenido de una nueva imagen (**buildah commit**).
- **Unmount a container**: Desmontar un contenedor montado (**buildah umount**).
- **Delete a container or an image** Eliminar un contenedor (**buildah rm**) o una imagen de contenedor (**buildah rmi**).

Para más detalles sobre Buildah, consulta la página de GitHub Buildah. La página de GitHub Buildah incluye páginas man y software que podría ser más reciente que el disponible con la versión de RHEL. Aquí hay otros artículos sobre Buildah que podrían interesarte:

- Tutorial de Buildah 1: Creación de imágenes de contenedores OCI
- Tutorial de Buildah 2: Uso de Buildah con registros de contenedores
- Buildah Blocks - Ponerse en forma

9.1.1. Instalación de Buildah

El paquete buildah está disponible con el módulo container-tools en RHEL 8 (**yum module install container-tools**). Puede instalar el paquete buildah por separado escribiendo:

```
# yum -y install buildah
```

Con el paquete buildah instalado, puede consultar las páginas man incluidas en el paquete buildah para obtener detalles sobre su uso. Para ver las páginas man disponibles y otra documentación, abra una página man, escriba:

```
# rpm -qd buildah
# man buildah
buildah(1)         General Commands Manual         buildah(1)

NAME
 Buildah - A command line tool that facilitates building OCI container images.
...
```

Las siguientes secciones describen cómo utilizar **buildah** para obtener contenedores, construir un contenedor a partir de un archivo Docker, construir uno desde cero, y gestionar contenedores de varias maneras.

9.2. OBTENCIÓN DE IMÁGENES CON BUILDAH

Para obtener una imagen de contenedor para utilizar con **buildah**, utilice el comando **buildah from**.

Tenga en cuenta que si está usando RHEL 8.0, puede encontrar problemas con la autenticación en el repositorio, vea el error. A continuación se explica cómo obtener una imagen de RHEL 8 del Registro de Red Hat como contenedor de trabajo para utilizar con el comando **buildah**:

```
# buildah from registry.redhat.io/ubi8/ubi
Getting image source signatures
Copying blob…
Writing manifest to image destination
Storing signatures
ubi-working-container
# buildah images
IMAGE ID      IMAGE NAME                       CREATED AT         SIZE
3da40a1670b5  registry.redhat.io/ubi8/ubi:latest  May 8, 2019 21:55  214 MB
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME         CONTAINER NAME
c6c9279ecc0f     *     3da40a1670b5 ...ubi8/ubi:latest ubi-working-container
```

Observe que el resultado del comando **buildah from** es una imagen (registry.redhat.io/ubi8/ubi:latest) y un contenedor de trabajo que está listo para ejecutarse desde esa imagen (ubi-working-container). A continuación se muestra un ejemplo de cómo ejecutar un comando desde ese contenedor:

```
# buildah run ubi-working-container cat /etc/redhat-release
Red Hat Enterprise Linux release 8.0
```

La imagen y el contenedor están ahora listos para su uso con Buildah.

9.3. CONSTRUIR UNA IMAGEN DESDE UN ARCHIVO DOCKER CON BUILDAH

Con el comando **buildah**, puedes crear una nueva imagen a partir de un archivo Docker. Los siguientes pasos muestran cómo construir una imagen que incluye un sencillo script que se ejecuta cuando la imagen se ejecuta.

Este sencillo ejemplo comienza con dos archivos en el directorio actual: Dockerfile (que contiene las instrucciones para construir la imagen del contenedor) y myecho (un script que hace eco de algunas palabras en la pantalla):

```
# ls
Dockerfile  myecho
# cat Dockerfile
FROM registry.redhat.io/ubi8/ubi
ADD myecho /usr/local/bin
ENTRYPOINT "/usr/local/bin/myecho"
# cat myecho
echo "This container works!"
# chmod 755 myecho
# ./myecho
This container works!
```

Con el Dockerfile en el directorio actual, construya el nuevo contenedor como sigue:

```
# buildah bud -t myecho .
STEP 1: FROM registry.redhat.io/ubi8/ubi
STEP 2: ADD myecho /usr/local/bin
```

```
STEP 3: ENTRYPOINT "/usr/local/bin/myecho"
```

El comando **buildah bud** crea una nueva imagen llamada myecho. Para ver esa nueva imagen, escribe:

```
# buildah images
IMAGE NAME        IMAGE TAG  IMAGE ID      CREATED AT          SIZE
localhost/myecho  latest     a3882af49784  Jun 21, 2019 12:21  216 MB
```

A continuación, puedes ejecutar la imagen para asegurarte de que funciona.

9.3.1. Ejecutar la imagen que has construido

Para comprobar que la imagen que has construido previamente funciona, puedes ejecutar la imagen utilizando **podman run**:

```
# podman run localhost/myecho
This container works!
```

9.3.2. Inspección de un contenedor con Buildah

Con **buildah inspect**, puede mostrar información sobre un contenedor o imagen. Por ejemplo, para construir e inspeccionar la imagen **myecho**, escriba:

```
# buildah from localhost/myecho
# buildah inspect localhost/myecho | less
{
 "Type": "buildah 0.0.1",
 "FromImage": "docker.io/library/myecho:latest",
 "FromImage-ID": "e2b190ac8...",
 "Config": "{\"created\":\"2018-11-13...

 "Entrypoint": [
     "/usr/local/bin/myecho"
  ],
  "WorkingDir": "/",
  "Labels": {
     "architecture": "x86_64",
     "authoritative-source-url": "registry.access.redhat.com",
     "build-date": "2018-09-19T20:46:28.459833",
```

Para inspeccionar un contenedor de esa misma imagen, escriba lo siguiente:

```
# buildah inspect myecho-working-container | less
{
    "Type": "buildah 0.0.1",
    "FromImage": "docker.io/library/myecho:latest",
    "FromImage-ID": "e2b190a...",
    "Config": "{\"created\":\"2018-11-13T19:5...
...
    "Container": "myecho-working-container",
    "ContainerID": "c0cd2e494d...",
    "MountPoint": "",
    "ProcessLabel": "system_u:system_r:svirt_lxc_net_t:s0:c89,c921",
    "MountLabel": "",
```

Observe que la salida del contenedor ha añadido información, como el nombre del contenedor, el id del contenedor, la etiqueta del proceso y la etiqueta de montaje a lo que había en la imagen.

9.4. MODIFICACIÓN DE UN CONTENEDOR PARA CREAR UNA NUEVA IMAGEN CON BUILDAH

Hay varias maneras de modificar un contenedor existente con el comando **buildah** y confirmar esos cambios en una nueva imagen de contenedor:

- Montar un contenedor y copiar archivos en él
- Utilice **buildah copy** y **buildah config** para modificar un contenedor

Una vez que haya modificado el contenedor, utilice **buildah commit** para confirmar los cambios en una nueva imagen.

9.4.1. Utilizando buildah mount para modificar un contenedor

Después de obtener una imagen con **buildah from**, puede utilizar esa imagen como base para una nueva imagen. El siguiente texto muestra cómo crear una nueva imagen montando un contenedor de trabajo, añadiendo archivos a ese contenedor, y luego confirmando los cambios en una nueva imagen.

Escriba lo siguiente para ver el contenedor de trabajo que utilizó anteriormente:

```
# buildah containers
CONTAINER ID BUILDER IMAGE ID     IMAGE NAME  CONTAINER NAME

dc8f21af4a47   *     1456eedf8101 registry.redhat.io/ubi8/ubi:latest
            ubi-working-container
6d1ffccb557d   *     ab230ac5aba3 docker.io/library/myecho:latest
            myecho-working-container
```

Monte la imagen del contenedor y establezca el punto de montaje en una variable ($mymount) para que sea más fácil de manejar:

```
# mymount=$(buildah mount myecho-working-container)
# echo $mymount
/var/lib/containers/storage/devicemapper/mnt/176c273fe28c23e5319805a2c48559305a57a706cc7ae7b
ec7da4cd79edd3c02/rootfs
```

Añade contenido al script creado anteriormente en el contenedor montado:

```
# echo 'echo \ "Incluso lo modificamos.\N" >> $mymount/usr/local/bin/myecho
```

Para confirmar el contenido que has añadido para crear una nueva imagen (llamada myecho), escribe lo siguiente:

```
# buildah commit myecho-working-container containers-storage:myecho2
```

Para comprobar que la nueva imagen incluye sus cambios, cree un contenedor de trabajo y ejecútelo:

```
# buildah images
```

```
IMAGE ID     IMAGE NAME     CREATED AT          SIZE
a7e06d3cd0e2 docker.io/library/myecho2:latest
                        Oct 12, 2017 15:15  3.144 KB
# buildah from docker.io/library/myecho2:latest
myecho2-working-container
# podman run docker.io/library/myecho2
This container works!
We even modified it.
```

Puede ver que el nuevo comando **echo** añadido al script muestra el texto adicional.

Cuando haya terminado, puede desmontar el contenedor:

```
# buildah umount myecho-working-container
```

9.4.2. Utilizando **buildah copy** y **buildah config** para modificar un contenedor

Con **buildah copy**, puedes copiar archivos a un contenedor sin montarlo primero. Aquí hay un ejemplo, usando el **myecho-working-container** creado (y desmontado) en la sección anterior, para copiar un nuevo script al contenedor y cambiar la configuración del contenedor para ejecutar ese script por defecto.

Crea un script llamado **newecho** y hazlo ejecutable:

```
# cat newecho
echo "I changed this container"
# chmod 755 newecho
```

Crear un nuevo contenedor de trabajo:

```
# buildah from myecho:latest
myecho-working-container-2
```

Copie **newecho** a /usr/local/bin dentro del contenedor:

```
# buildah copy myecho-working-container-2 newecho /usr/local/bin
```

Cambie la configuración para utilizar el script **newecho** como nuevo punto de entrada:

```
# buildah config --entrypoint "/bin/sh -c /usr/local/bin/newecho "myecho-working-container-2
```

Ejecute el nuevo contenedor, lo que debería dar lugar a la ejecución del comando **newecho**:

```
# buildah run myecho-working-container-2
I changed this container
```

Si el contenedor se ha comportado como esperabas que lo hiciera, puedes confirmarlo en una nueva imagen (mynewecho):

```
# buildah commit myecho-working-container-2 containers-storage:mynewecho
```

9.5. CREACIÓN DE IMÁGENES DESDE CERO CON BUILDAH

En lugar de comenzar con una imagen base, puede crear un nuevo contenedor que no tenga contenido y sólo una pequeña cantidad de metadatos del contenedor. Esto se conoce como un contenedor **scratch**. Aquí hay algunos aspectos a tener en cuenta cuando se opta por crear una imagen partiendo de un contenedor base con el comando **buildah**:

- Cuando se construye un contenedor scratch se pueden copiar ejecutables sin dependencias en la imagen scratch y hacer unos pocos ajustes de configuración para que funcione un contenedor mínimo.
- Para utilizar herramientas como **yum** o **rpm** paquetes para poblar el contenedor scratch, es necesario al menos inicializar una base de datos RPM en el contenedor y añadir un paquete de liberación. El ejemplo siguiente muestra cómo hacerlo.
- Si acaba añadiendo muchos paquetes RPM, considere la posibilidad de utilizar las imágenes base de **ubi** o **ubi-minimal** en lugar de una imagen de cero. Estas imágenes base han sido recortadas en cuanto a documentación, paquetes de idiomas y otros componentes, lo que puede hacer que tu imagen sea más pequeña.

Este ejemplo añade un servicio web (httpd) a un contenedor y lo configura para que se ejecute. Para empezar, crea un contenedor de cero:

```
# buildah from scratch
working-container
```

Esto crea sólo un contenedor vacío (sin imagen) que puede montar de la siguiente manera:

```
# scratchmnt=$(buildah mount working-container)
# echo $scratchmnt
/var/lib/containers/storage/devicemapper/mnt/cc92011e9a2b077d03a97c0809f1f3e7fef0f29bdc6ab5e86
b85430ec77b2bf6/rootfs
```

Inicialice una base de datos RPM dentro de la imagen scratch y añada el paquete redhat-release (que incluye otros archivos necesarios para que los RPM funcionen):

```
# yum install -y --releasever=8 --installroot=$scratchmnt redhat-release
```

Instale el servicio httpd en el directorio scratch:

```
# yum install -y --setopt=reposdir=/etc/yum.repos.d \
    --installroot=$scratchmnt \
    --setopt=cachedir=/var/cache/dnf httpd
```

Añade algún texto a un archivo index.html en el contenedor, para poder probarlo más tarde:

```
# echo \ "Su contenedor httpd desde cero funcionó.\N- > $scratchmnt/var/www/html/index.html
```

En lugar de ejecutar httpd como un servicio init, establezca algunas opciones de **buildah config** para ejecutar el demonio httpd directamente desde el contenedor:

```
# buildah config --cmd "/usr/sbin/httpd -DFOREGROUND" working-container
# buildah config --port 80/tcp working-container
# buildah commit working-container localhost/myhttpd:latest
```

Por ahora, puede utilizar el ID de la imagen para ejecutar la nueva imagen como un contenedor con el comando **podman**:

```
# podman images
REPOSITORY          TAG              IMAGE ID          CREATED            SIZE
localhost/myhttpd   latest           47c0795d7b0e      9 minutes ago      665.6 MB
# podman run -p 8080:80 -d --name httpd-server 47c0795d7b0e
# curl localhost:8080
Your httpd container from scratch worked.
```

9.6. ELIMINACIÓN DE IMÁGENES O CONTENEDORES CON BUILDAH

Cuando haya terminado con determinados contenedores o imágenes, puede eliminarlos con **buildah rm** o **buildah rmi**, respectivamente. He aquí algunos ejemplos.

Para eliminar el contenedor creado en la sección anterior, puedes escribir lo siguiente para ver el contenedor montado, desmontarlo y eliminarlo:

```
# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER NAME
05387e29ab93     *     c37e14066ac7 docker.io/library/myecho:latest  myecho-working-container
# buildah mount
05387e29ab93 /var/lib/containers/storage/devicemapper/mnt/9274181773a.../rootfs
# buildah umount 05387e29ab93
# buildah rm 05387e29ab93
05387e29ab93151cf52e9c85c573f3e8ab64af1592b1ff9315db8a10a77d7c22
```

Para eliminar la imagen creada anteriormente, puedes escribir lo siguiente:

```
# buildah rmi docker.io/library/myecho:latest
untagged: docker.io/library/myecho:latest
ab230ac5aba3b5a0a7c3d2c5e0793280c1a1b4d2457a75a01b70a4b7a9ed415a
```

9.7. USO DE REGISTROS DE CONTENEDORES CON BUILDAH

Con Buildah, puedes empujar y sacar imágenes de contenedores entre tu sistema local y los registros de contenedores públicos o privados. Los siguientes ejemplos muestran cómo hacerlo:

- Empuje los contenedores hacia y desde un registro privado con buildah.
- Empuje y tire del contenedor entre su sistema local y el Registro Docker.
- Utiliza las credenciales para asociar tus contenedores con una cuenta de registro cuando los empujes.

Utilice el comando skopeo, junto con el comando **buildah**, para consultar los registros en busca de información sobre las imágenes de los contenedores.

9.7.1. Envío de contenedores a un registro privado

El envío de contenedores a un registro de contenedores privado con el comando **buildah** funciona de forma muy similar al envío de contenedores con el comando **docker**. Es necesario:

- Configure un registro privado (OpenShift proporciona un registro de contenedores o puede configurar un registro de contenedores de Red Hat Quay).
- Cree o adquiera la imagen del contenedor que desea impulsar.
- Utilice **buildah push** para enviar la imagen al registro.

Para empujar una imagen desde su almacenamiento local de contenedores Buildah, compruebe el nombre de la imagen, y luego empuje utilizando el comando **buildah push**. Recuerda identificar tanto el nombre de la imagen local como un nuevo nombre que incluya la ubicación. Por ejemplo, un registro que se ejecuta en el sistema local que está escuchando en el puerto TCP 5000 se identificaría como localhost:5000.

```
# buildah images
IMAGE ID     IMAGE NAME                       CREATED AT          SIZE
cb702d492ee9 docker.io/library/myecho2:latest Nov 12, 2018 16:50     3.143 KB

# buildah push --tls-verify=false myecho2:latest localhost:5000/myecho2:latest
Getting image source signatures
Copying blob sha256:e4efd0...
...
Writing manifest to image destination
Storing signatures
```

Utilice el comando **curl** para listar las imágenes en el registro y **skopeo** para inspeccionar los metadatos sobre la imagen:

```
# curl http://localhost:5000/v2/_catalog
{"repositories":["myatomic","myecho2"]}
# curl http://localhost:5000/v2/myecho2/tags/list
{"name":"myecho2","tags":["latest"]}
# skopeo inspect --tls-verify=false docker://localhost:5000/myecho2:latest | less
{
    "Name": "localhost:5000/myecho2",
    "Digest": "sha256:8999ff6050...",
    "RepoTags": [
        "latest"
    ],
    "Created": "2017-11-21T16:50:25.830343Z",
    "DockerVersion": "",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.redhat.io",
```

En este punto, cualquier herramienta que pueda extraer imágenes de contenedores de un registro de contenedores puede obtener una copia de tu imagen empujada. Por ejemplo, en un sistema RHEL 7 podrías iniciar el demonio docker y tratar de extraer la imagen para que pueda ser utilizada por el comando **docker** de la siguiente manera:

```
# systemctl start docker
# docker pull localhost:5000/myecho2
# docker run localhost:5000/myecho2
This container works!
```

9.7.2. Envío de contenedores al Docker Hub

Puedes usar tus credenciales del Docker Hub para empujar y sacar imágenes del Docker Hub con el comando **buildah**. Para este ejemplo, sustituye el nombre de usuario y la contraseña (testaccountXX:My00P@sswd) por tus propias credenciales del Docker Hub:

```
# buildah push --creds testaccountXX:My00P@sswd \
    docker.io/library/myecho2:latest docker://testaccountXX/myecho2:latest
```

Al igual que con el registro privado, puede obtener y ejecutar el contenedor desde el Docker Hub con el comando **podman**, **buildah** o **docker**:

```
# podman run docker.io/textaccountXX/myecho2:latest
This container works!
# buildah from docker.io/textaccountXX/myecho2:latest
myecho2-working-container-2
# podman run myecho2-working-container-2
This container works!
```

CAPÍTULO 10. CONTROL DE LOS CONTENEDORES

Este capítulo se centra en los comandos útiles de Podman que le permiten gestionar un entorno Podman, incluyendo la determinación de la salud del contenedor, la visualización de la información del sistema y del pod, y la supervisión de los eventos de Podman.

10.1. REALIZAR UNA COMPROBACIÓN DE LA SALUD DE UN CONTENEDOR

El healthcheck permite determinar la salud o la preparación del proceso que se ejecuta dentro del contenedor. Un healthcheck consta de cinco componentes básicos:

- Comando
- Reintentos
- Intervalo
- Periodo de inicio
- Tiempo de espera

A continuación se describen los componentes del chequeo.

Comando

Podman ejecuta el comando dentro del contenedor de destino y espera el código de salida.

Los otros cuatro componentes están relacionados con la programación del chequeo y son opcionales.

Reintentos

Define el número de comprobaciones de salud fallidas consecutivas que deben producirse antes de que el contenedor se marque como "sin salud". Un chequeo exitoso reinicia el contador de reintentos.

Intervalo

Describe el tiempo entre la ejecución del comando healthcheck. Tenga en cuenta que los intervalos pequeños hacen que el sistema pase mucho tiempo ejecutando healthchecks. Los intervalos grandes causan problemas con la captura de los tiempos de espera.

Periodo de inicio

Describe el tiempo que transcurre entre el inicio del contenedor y el momento en que se desea ignorar los fallos de healthcheck.

Tiempo de espera

Describe el periodo de tiempo que debe completar el chequeo antes de considerarse fallido.

Los Healthchecks se ejecutan dentro del contenedor. Las comprobaciones de salud sólo tienen sentido si se conoce el estado de salud del servicio y se puede diferenciar entre una comprobación de salud satisfactoria y una no satisfactoria.

Procedimiento

1. Definir un chequeo de salud:

```
$ sudo podman run -dt --name hc1 --health-cmd='curl http://localhost || exit 1' --health-
interval=0 quay.io/libpod/alpine_nginx:latest
D25ee6faaf6e5e12c09e734b1ac675385fe4d4e8b52504dd01a60e1b726e3edb
```

- La opción **--health-cmd** establece un comando de healthcheck para el contenedor.
- La opción **-health-interval=0** con valor 0 indica que se quiere ejecutar el healthcheck manualmente.

2. Ejecute el chequeo manualmente:

```
$ sudo podman healthcheck run hc1
Healthy
```

3. Opcionalmente, puede comprobar el estado de salida del último comando:

```
$ echo $?
0
```

El valor "0" significa éxito.

Recursos adicionales

- Para más información sobre el comando **podman run**, escriba **man podman-run**.
- Para más información, consulte el artículo Supervisión de la vitalidad y disponibilidad de los contenedores con Podman, de Brent Baude.

10.2. VISUALIZACIÓN DE LA INFORMACIÓN DEL SISTEMA PODMAN

El comando **podman system** le permite gestionar los sistemas Podman. Esta sección proporciona información sobre cómo mostrar la información del sistema Podman.

Procedimiento

- Muestra la información del sistema Podman:
  - Para mostrar el uso del disco de Podman, introduzca:

```
$ podman system df
TYPE           TOTAL   ACTIVE   SIZE    RECLAIMABLE
Images         3       1        255MB   255MB (100%)
Containers     1       0        0B      0B (0%)
Local Volumes  0       0        0B      0B (0%)
```

  - Para mostrar información detallada sobre el uso del espacio, introduzca:

```
$ podman system df -v
Images space usage:

REPOSITORY                  TAG      IMAGE ID       CREATED        SIZE     SHARED
SIZE   UNQUE SIZE   CONTAINERS
docker.io/library/alpine    latest   e7d92cdc71fe   3 months ago   5.86MB   0B
5.86MB       0
```

```
registry.access.redhat.com/ubi8/ubi   latest   8121a9f5303b   6 weeks ago    240MB   0B
240MB       1
quay.io/libpod/alpine_nginx           latest   3ef70f7291f4   18 months ago   9.21MB   0B
9.21MB      0

Containers space usage:

CONTAINER ID   IMAGE   COMMAND    LOCAL VOLUMES   SIZE   CREATED
STATUS   NAMES
ff0167c6c271   8121    /bin/bash  0               0B     10 seconds ago   exited   playTest

Local Volumes space usage:

VOLUME NAME   LINKS   SIZE
```

- Para mostrar información sobre el host, las estadísticas de almacenamiento actuales y la compilación de Podman, introduzca:

```
$ podman system info
host:
  arch: amd64
  buildahVersion: 1.15.0
  cgroupVersion: v1
  conmon:
    package: conmon-2.0.18-1.module+el8.3.0+7084+c16098dd.x86_64
    path: /usr/bin/conmon
    version: 'conmon version 2.0.18, commit:
7fd3f71a218f8d3a7202e464252aeb1e942d17eb'
  cpus: 1
  distribution:
    distribution: '"rhel"'
    version: "8.3"
  eventLogger: file
  hostname: localhost.localdomain
  idMappings:
    gidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
    uidmap:
    - container_id: 0
      host_id: 1000
      size: 1
    - container_id: 1
      host_id: 100000
      size: 65536
  kernel: 4.18.0-227.el8.x86_64
  linkmode: dynamic
  memFree: 69713920
  memTotal: 1376636928
  ociRuntime:
    name: runc
    package: runc-1.0.0-66.rc10.module+el8.3.0+7084+c16098dd.x86_64
```

```
    path: /usr/bin/runc
    version: 'runc version spec: 1.0.1-dev'
  os: linux
  remoteSocket:
    path: /run/user/1000/podman/podman.sock
  rootless: true
  slirp4netns:
    executable: /usr/bin/slirp4netns
    package: slirp4netns-1.1.1-1.module+el8.3.0+7084+c16098dd.x86_64
    version: |-
      slirp4netns version 1.1.1
      commit: bbf27c5acd4356edb97fa639b4e15e0cd56a39d5
      libslirp: 4.3.0
      SLIRP_CONFIG_VERSION_MAX: 3
  swapFree: 1833693184
  swapTotal: 2147479552
  uptime: 145h 19m 14.55s (Approximately 6.04 days)
registries:
  search:
  - registry.access.redhat.com
  - registry.redhat.io
  - docker.io
store:
  configFile: /home/user/.config/containers/storage.conf
  containerStore:
    number: 1
    paused: 0
    running: 0
    stopped: 1
  graphDriverName: overlay
  graphOptions:
    overlay.mount_program:
      Executable: /usr/bin/fuse-overlayfs
      Package: fuse-overlayfs-1.1.1-1.module+el8.3.0+7121+472bc0cf.x86_64
      Version: |-
        fuse-overlayfs: version 1.1.0
        FUSE library version 3.2.1
        using FUSE kernel interface version 7.26
  graphRoot: /home/user/.local/share/containers/storage
  graphStatus:
    Backing Filesystem: xfs
    Native Overlay Diff: "false"
    Supports d_type: "true"
    Using metacopy: "false"
  imageStore:
    number: 15
  runRoot: /run/user/1000/containers
  volumePath: /home/user/.local/share/containers/storage/volumes
version:
  APIVersion: 1
  Built: 0
  BuiltTime: Thu Jan  1 01:00:00 1970
  GitCommit: ""
  GoVersion: go1.14.2
  OsArch: linux/amd64
  Version: 2.0.0
```

- Para eliminar todos los contenedores, imágenes y datos de volumen no utilizados, introduzca:

```
$ podman system prune
WARNING! This will remove:
        - all stopped containers
        - all stopped pods
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
```

  - El comando **podman system prune** elimina todos los contenedores no utilizados (tanto los colgados como los no referenciados), los pods y, opcionalmente, los volúmenes del almacenamiento local.
  - Utilice la opción **--all** para eliminar todas las imágenes no utilizadas. Las imágenes no utilizadas son imágenes colgantes y cualquier imagen que no tenga ningún contenedor basado en ella.
  - Utilice la opción **--volume** para podar volúmenes. Por defecto, los volúmenes no se eliminan para evitar que se borren datos importantes si actualmente no hay ningún contenedor que utilice el volumen.

Recursos adicionales

- Para más información sobre el comando **podman system df**, escriba **man podman-system-df**.
- Para más información sobre el comando **podman system info**, escriba **man podman-system-info**.
- Para más información sobre el comando **podman system prune**, escriba **man podman-system-prune**.

10.3. TIPOS DE EVENTOS DE PODMAN

Puede supervisar los eventos que se producen en Podman. Existen varios tipos de eventos y cada tipo de evento informa de diferentes estados.

El tipo de evento *container* informa de los siguientes estados:

- adjuntar
- punto de control
- limpieza
- comprometerse
- crear
- exec
- exportar
- importar

- init
- matar
- monte
- pausa
- ciruela pasa
- eliminar
- reiniciar
- restaurar
- iniciar
- detener
- sincronizar
- desmontar
- desactivar

El tipo de evento *pod* informa de los siguientes estados:

- crear
- matar
- pausa
- eliminar
- iniciar
- detener
- desactivar

El tipo de evento *image* informa de los siguientes estados:

- ciruela pasa
- empuje
- tirar de
- guardar
- eliminar
- etiqueta
- desmarcarse

El tipo *system* informa de los siguientes estados:

- actualizar
- renumerar

El tipo *volume* informa de los siguientes estados:

- crear
- ciruela pasa
- eliminar

Recursos adicionales

- Para más información sobre el comando **podman events**, escriba **man podman-events**.

10.4. SEGUIMIENTO DE LOS EVENTOS DE PODMAN

Puede supervisar e imprimir los eventos que se producen en Podman. Cada evento incluirá una marca de tiempo, un tipo, un estado, un nombre (si procede) y una imagen (si procede).

Procedimiento

- Muestra los eventos de Podman:
  - Para mostrar todos los eventos de Podman, introduzca:

```
$ podman events
2020-05-14 10:33:42.312377447 -0600 CST container create 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:46.958768077 -0600 CST container init 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:46.973661968 -0600 CST container start 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:50.833761479 -0600 CST container stop 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
2020-05-14 10:33:51.047104966 -0600 CST container cleanup 34503c192940
(image=registry.access.redhat.com/ubi8/ubi:latest, name=keen_colden)
```

    Para salir del registro, pulse CTRL c.

  - Para mostrar sólo los eventos de creación de Podman, introduzca:

```
$ podman events --filter event=create
2020-05-14 10:36:01.375685062 -0600 CST container create 20dc581f6fbf
(image=registry.access.redhat.com/ubi8/ubi:latest)
2019-03-02 10:36:08.561188337 -0600 CST container create 58e7e002344c
(image=registry.access.redhat.com/ubi8/ubi-minimal:latest)
2019-03-02 10:36:29.978806894 -0600 CST container create d81e30f1310f
(image=registry.access.redhat.com/ubi8/ubi-init:latest)
```

Recursos adicionales

- Para más información sobre el comando **podman events**, escriba **man podman-events**.

CAPÍTULO 11. USO DE LA CLI DE CONTAINER-TOOLS

11.1. PODMAN

El comando **podman** (que significa Pod Manager) permite ejecutar contenedores como entidades independientes, sin necesidad de que intervengan Kubernetes, el tiempo de ejecución de Docker o cualquier otro tiempo de ejecución de contenedores. Es una herramienta que puede actuar como reemplazo del comando **docker**, implementando la misma sintaxis de línea de comandos, mientras que agrega aún más características de administración de contenedores. Las características de **podman** incluyen:

- **Based on the Docker interface**: Dado que la sintaxis de **podman** es un reflejo del comando **docker**, la transición a **podman** debería ser fácil para quienes estén familiarizados con **docker**.
- **Managing containers and images**: Tanto las imágenes de contenedores compatibles con Docker como con OCI pueden utilizarse con **podman** para:
  - Ejecutar, detener y reiniciar contenedores
  - Creación y gestión de imágenes de contenedores (push, commit, configuración, build, etc.)
- **Managing pods**: Además de ejecutar contenedores individuales, **podman** puede ejecutar un conjunto de contenedores agrupados en un pod. Un pod es la unidad de contenedores más pequeña que gestiona Kubernetes.
- **Working with no runtime**: **podman** no utiliza ningún entorno de ejecución para trabajar con contenedores.

Estas son algunas de las características de implementación de Podman que debes conocer:

- Podman, Buildah y el motor de contenedores CRI-O utilizan el mismo directorio de almacenamiento back-end, **/var/lib/containers**, en lugar de utilizar la ubicación de almacenamiento de Docker (**/var/lib/docker**), por defecto.
- Aunque Podman, Buildah y CRI-O comparten el mismo directorio de almacenamiento, no pueden interactuar con los contenedores del otro. Sin embargo, estas herramientas pueden compartir imágenes. Eventualmente esas funciones podrán compartir contenedores.
- El comando **podman**, al igual que el comando **docker**, puede construir imágenes de contenedores desde un archivo Docker.
- El comando **podman** puede ser una herramienta útil para solucionar problemas cuando el servicio **CRI-O** no está disponible.
- Las opciones del comando **docker** que no son soportadas por **podman** incluyen network, node, plugin (**podman** no soporta plugins), rename (use rm y create para renombrar contenedores con **podman**), secret, service, stack, y swarm (**podman** no soporta Docker Swarm). Las opciones de contenedor e imagen se utilizan para ejecutar subcomandos que se utilizan directamente en **podman**.
- Para interactuar programáticamente con podman, puede utilizar la API RESTful de Podman v2.0, que funciona tanto en un entorno con raíces como sin ellas. Para más información, consulte el capítulo Uso de la API de herramientas de contenedores .

11.1.1. Uso de los comandos de podman

Si está acostumbrado a utilizar el comando **docker** para trabajar con contenedores, encontrará que la mayoría de las funciones y opciones coinciden con las de **podman**. La Tabla 1 muestra una lista de comandos que puedes utilizar con **podman** (escribe **podman -h** para ver esta lista):

Tabla 11.1. Comandos soportados por podman

comando podman	Descripción	comando podman	Descripción
attach	Adjuntar a un contenedor en funcionamiento	**commit**	Crear una nueva imagen a partir de un contenedor modificado
build	Construir una imagen utilizando las instrucciones de Dockerfile	**create**	Crear, pero no iniciar, un contenedor
diff	Inspeccionar los cambios en los sistemas de archivos del contenedor	**exec**	Ejecutar un proceso en un contenedor en funcionamiento
export	Exportar el contenido del sistema de archivos del contenedor como un archivo tar	**help, h**	Muestra una lista de comandos o la ayuda de un comando
history	Mostrar el historial de una imagen especificada	**images**	Lista de imágenes en el almacenamiento local
import	Importación de un tarball para crear una imagen del sistema de archivos	**info**	Mostrar información del sistema
inspect	Mostrar la configuración de un contenedor o imagen	**kill**	Enviar una señal específica a uno o varios contenedores en funcionamiento
load	Cargar una imagen desde un archivo	**login**	Acceder a un registro de contenedores
logout	Cierre de sesión de un registro de contenedores	**logs**	Obtener los registros de un contenedor
mount	Montar el sistema de archivos raíz de un contenedor en funcionamiento	**pause**	Pone en pausa todos los procesos de uno o varios contenedores

ps	Lista de contenedores	**port**	Lista de mapeos de puertos o un mapeo específico para el contenedor
pull	Sacar una imagen de un registro	**push**	Enviar una imagen a un destino especificado
restart	Reiniciar uno o varios contenedores	**rm**	Eliminar uno o más contenedores del host. Añadir **-f** si se está ejecutando.
rmi	elimina una o varias imágenes del almacenamiento local	**run**	ejecutar un comando en un nuevo contenedor
save	Guardar la imagen en un archivo	**search**	buscar imagen en el registro
start	Iniciar uno o más contenedores	**stats**	Mostrar el porcentaje de CPU, memoria, E/S de red, E/S de bloque y PIDs para uno o más contenedores
stop	Detener uno o varios contenedores	**tag**	Añadir un nombre adicional a una imagen local
top	Mostrar los procesos en ejecución de un contenedor	**umount, unmount**	Desmontar el sistema de archivos raíz de un contenedor en funcionamiento
unpause	Desactivar los procesos de uno o varios contenedores	**version**	Mostrar la información de la versión de podman
wait	Bloqueo en uno o varios contenedores		

11.1.2. Creación de políticas SELinux para contenedores

Para generar políticas SELinux para contenedores, utilice la herramienta UDICA. Para más información, consulte Introducción al generador de políticas SELinux udica.

11.1.3. Uso de podman con MPI

Puede utilizar Podman con Open MPI (Message Passing Interface) para ejecutar contenedores en un entorno de computación de alto rendimiento (HPC).

El ejemplo se basa en el programa ring.c tomado de Open MPI. En este ejemplo, un valor es pasado por todos los procesos en forma de anillo. Cada vez que el mensaje pasa por el rango 0, el valor se decrementa. Cuando cada proceso recibe el mensaje 0, lo pasa al siguiente proceso y luego sale. Al pasar el 0 primero, todos los procesos reciben el mensaje 0 y pueden salir normalmente.

Procedimiento

1. Instalar Open MPI:

```
$ sudo yum install openmpi
```

2. Para activar los módulos de entorno, escriba:

```
$ . /etc/profile.d/modules.sh
```

3. Cargue el módulo **mpi/openmpi-x86_64**:

```
$ module load mpi/openmpi-x86_64
```

   Opcionalmente, para cargar automáticamente el módulo **mpi/openmpi-x86_64**, añada esta línea al archivo **.bashrc**:

```
$ echo \ "module load mpi/openmpi-x86_64" >> .bashrc
```

4. Para combinar **mpirun** y **podman**, cree un contenedor con la siguiente definición:

```
$ cat Containerfile
FROM registry.access.redhat.com/ubi8/ubi

RUN yum -y install openmpi-devel wget && \
    yum clean all

RUN wget https://raw.githubusercontent.com/open-mpi/ompi/master/test/simple/ring.c && \
    /usr/lib64/openmpi/bin/mpicc ring.c -o /home/ring && \
    rm -f ring.c
```

5. Construye el contenedor:

```
$ podman build --tag=mpi-ring .
```

6. Inicie el contenedor. En un sistema con 4 CPUs este comando inicia 4 contenedores:

```
$ mpirun \
  --mca orte_tmpdir_base /tmp/podman-mpirun \
  podman run --env-host \
    -v /tmp/podman-mpirun:/tmp/podman-mpirun \
    --userns=keep-id \
    --net=host --pid=host --ipc=host \
    mpi-ring /home/ring
Rank 2 has cleared MPI_Init
```

```
Rank 2 has completed ring
Rank 2 has completed MPI_Barrier
Rank 3 has cleared MPI_Init
Rank 3 has completed ring
Rank 3 has completed MPI_Barrier
Rank 1 has cleared MPI_Init
Rank 1 has completed ring
Rank 1 has completed MPI_Barrier
Rank 0 has cleared MPI_Init
Rank 0 has completed ring
Rank 0 has completed MPI_Barrier
```

Como resultado, **mpirun** inicia 4 contenedores Podman y cada contenedor está ejecutando una instancia del binario **ring**. Los 4 procesos se comunican entre sí a través de MPI.

Las siguientes opciones de **mpirun** se utilizan para iniciar el contenedor:

- la línea**--mca orte_tmpdir_base /tmp/podman-mpirun** indica a Open MPI que cree todos sus archivos temporales en **/tmp/podman-mpirun** y no en **/tmp**. Si se utiliza más de un nodo, este directorio tendrá un nombre diferente en los otros nodos. Esto requiere montar el directorio completo **/tmp** en el contenedor, lo que es más complicado.

El comando **mpirun** especifica el comando a iniciar, el comando **podman**. Las siguientes opciones de **podman** se utilizan para iniciar el contenedor:

- **run** comando ejecuta un contenedor.
- **--env-host** opción copia todas las variables de entorno del host en el contenedor.
- **-v /tmp/podman-mpirun:/tmp/podman-mpirun** indica a Podman que monte el directorio donde Open MPI crea sus directorios y archivos temporales para que estén disponibles en el contenedor.
- la línea**--userns=keep-id** asegura el mapeo de la identificación del usuario dentro y fuera del contenedor.
- la línea**--net=host --pid=host --ipc=host** establece los mismos espacios de nombres de red, PID e IPC.
- **mpi-ring** es el nombre del contenedor.
- **/home/ring** es el programa MPI en el contenedor.

Para más información, véase el artículo Podman in HPC environments de Adrian Reber.

11.1.4. Creación y restauración de puntos de control de contenedores

Checkpoint/Restore In Userspace (CRIU) es un software que permite establecer un punto de control en un contenedor en ejecución o en una aplicación individual y almacenar su estado en el disco. Puedes utilizar los datos guardados para restaurar el contenedor después de un reinicio en el mismo punto en el que se hizo el checkpoint.

11.1.4.1. Creación y restauración de un punto de control del contenedor de forma local

Este ejemplo se basa en un servidor web basado en Python que devuelve un único número entero que se incrementa después de cada petición.

Procedimiento

1. Crear un servidor basado en Python:

```
# cat counter.py
#!/usr/bin/python3

import http.server

counter = 0

class handler(http.server.BaseHTTPRequestHandler):
   def do_GET(s):
      global counter
        s.send_response(200)
        s.send_header('Content-type', 'text/html')
        s.end_headers()
        s.wfile.write(b'%d\n' % counter)
        counter += 1


server = http.server.HTTPServer(('', 8088), handler)
server.serve_forever()
```

2. Cree un contenedor con la siguiente definición:

```
# cat Containerfile
FROM registry.access.redhat.com/ubi8/ubi

COPY counter.py /home/counter.py

RUN useradd -ms /bin/bash counter

RUN yum -y install python3 && chmod 755 /home/counter.py

USER counter
ENTRYPOINT /home/counter.py
```

   El contenedor está basado en la Imagen Base Universal (UBI 8) y utiliza un servidor basado en Python.

3. Construye el contenedor:

```
# podman build . --tag counter
```

   Los archivos **counter.py** y **Containerfile** son la entrada para el proceso de construcción del contenedor (**podman build**). La imagen construida se almacena localmente y se etiqueta con la etiqueta **counter**.

4. Inicie el contenedor como root:

```
# podman run --name criu-test --detach counter
```

5. Para listar todos los contenedores en ejecución, introduzca:

```
# podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED   STATUS  PORTS NAMES
e4f82fd84d48  localhost/counter:latest  5 seconds ago  Up 4 seconds ago  criu-test
```

6. Muestra la dirección IP del contenedor:

```
# podman inspect criu-test --format "{{.NetworkSettings.IPAddress}}"
10.88.0.247
```

7. Enviar solicitudes al contenedor:

```
# curl 10.88.0.247:8080
0
# curl 10.88.0.247:8080
1
```

8. Crear un punto de control para el contenedor:

```
# podman container checkpoint criu-test
```

9. Reinicia el sistema.

10. Restaurar el contenedor:

```
# podman container restore --keep criu-test
```

11. Enviar solicitudes al contenedor:

```
# curl 10.88.0.247:8080
2
# curl 10.88.0.247:8080
3
# curl 10.88.0.247:8080
4
```

El resultado ahora no comienza de nuevo en **0**, sino que continúa en el valor anterior.

De este modo, puede guardar fácilmente el estado completo del contenedor mediante un reinicio.

Para más información, consulta el artículo Adding checkpoint/restore support to Podman de Adrian Reber.

11.1.4.2. Reducción del tiempo de arranque mediante la restauración de contenedores

Puede utilizar la migración de contenedores para reducir el tiempo de arranque de los contenedores que requieren un cierto tiempo para inicializarse. Utilizando un punto de control, puede restaurar el contenedor varias veces en el mismo host o en diferentes hosts. Este ejemplo se basa en el contenedor de la sección Crear y restaurar un punto de control del contenedor localmente .

Procedimiento

1. Cree un punto de control del contenedor y exporte la imagen del punto de control a un archivo **tar.gz**:

```
# podman container checkpoint criu-test --export /tmp/chkpt.tar.gz
```

2. Restaurar el contenedor desde el archivo **tar.gz**:

```
# podman container restore --import /tmp/chkpt.tar.gz --name counter1
# podman container restore --import /tmp/chkpt.tar.gz --name counter2
# podman container restore --import /tmp/chkpt.tar.gz --name counter3
```

La opción **--name** (**-n**) especifica un nuevo nombre para los contenedores restaurados desde el punto de control exportado.

3. Muestra el ID y el nombre de cada contenedor:

```
# podman ps -a --format "{{.ID}} {{.Names}}"
a8b2e50d463c counter3
faabc5c27362 counter2
2ce648af11e5 counter1
```

4. Muestra la dirección IP de cada contenedor:

```
#  podman inspect counter1 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.248

#  podman inspect counter2 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.249

#  podman inspect counter3 --format "{{.NetworkSettings.IPAddress}}"
10.88.0.250
```

5. Enviar solicitudes a cada contenedor:

```
#  curl 10.88.0.248:8080
4
#  curl 10.88.0.249:8080
4
#  curl 10.88.0.250:8080
4
```

Tenga en cuenta que el resultado es **4** en todos los casos, porque está trabajando con diferentes contenedores restaurados desde el mismo punto de control.

Utilizando este enfoque, puede iniciar rápidamente réplicas con estado del contenedor inicialmente verificado.

Para más información, consulte el artículo Container migration with Podman on RHEL de Adrian Reber.

11.1.4.3. Migración de contenedores entre sistemas

Este procedimiento muestra la migración de contenedores en ejecución de un sistema a otro, sin perder el estado de las aplicaciones que se ejecutan en el contenedor. Este ejemplo se basa en el contenedor de la sección Crear y restaurar un punto de control del contenedor localmente etiquetado con **counter**.

Requisitos previos

Los siguientes pasos no son necesarios si el contenedor es empujado a un registro ya que Podman descargará automáticamente el contenedor desde un registro si no está disponible localmente. En este ejemplo no se utiliza un registro, hay que exportar el contenedor previamente construido y etiquetado (ver sección Creación y restauración de un punto de control del contenedor localmente) localmente e importar el contenedor en el sistema de destino de esta migración.

- Exportar un contenedor previamente construido:

```
# podman save --output counter.tar counter
```

- Copiar la imagen del contenedor exportado al sistema de destino (***other_host***):

```
# scp contador.tar other_host:
```

- Importar el contenedor exportado en el sistema de destino:

```
# ssh other_host podman load --input counter.tar
```

Ahora el sistema de destino de esta migración de contenedores tiene la misma imagen de contenedor almacenada en su almacenamiento local de contenedores.

Procedimiento

1. Inicie el contenedor como root:

```
# podman run --name criu-test --detach counter
```

2. Muestra la dirección IP del contenedor:

```
# podman inspect criu-test --format "{{.NetworkSettings.IPAddress}}"
10.88.0.247
```

3. Enviar solicitudes al contenedor:

```
# curl 10.88.0.247:8080
0
# curl 10.88.0.247:8080
1
```

4. Cree un punto de control del contenedor y exporte la imagen del punto de control a un archivo **tar.gz**:

```
# podman container checkpoint criu-test --export /tmp/chkpt.tar.gz
```

5. Copie el archivo de puntos de control en el host de destino:

```
# scp /tmp/chkpt.tar.gz other_host:/tmp/
```

6. Restaurar el punto de control en el host de destino (***other_host***):

```
# podman container restore --import /tmp/chkpt.tar.gz
```

7. Enviar una solicitud al contenedor en el host de destino (***other_host***):

```
# curl 10.88.0.247:8080
2
```

Como resultado, el contenedor con estado se ha migrado de un sistema a otro sin perder su estado.

Para más información, consulte el artículo Container migration with Podman on RHEL de Adrian Reber.

11.2. RUNC

El tiempo de ejecución de contenedores "runC" es una implementación ligera y portátil de la especificación de tiempo de ejecución de contenedores de la Iniciativa de Contenedores Abiertos (OCI). El runC reúne muchas de las características de bajo nivel que hacen posible la ejecución de contenedores. Comparte mucho código de bajo nivel con Docker, pero no depende de ninguno de los componentes de la plataforma Docker.

runc es compatible con los espacios de nombres de Linux, la migración en vivo, y tiene perfiles de rendimiento portátiles. También proporciona soporte completo para las características de seguridad de Linux como SELinux, grupos de control (cgroups), seccomp, y otros. Puedes construir y ejecutar imágenes con runc, o puedes ejecutar imágenes compatibles con OCI con runc.

11.2.1. Ejecución de contenedores con runc

Con runc, los contenedores se configuran utilizando paquetes. Un paquete para un contenedor es un directorio que incluye un archivo de especificación llamado **config.json** y un sistema de archivos raíz. El sistema de archivos raíz contiene el contenido del contenedor.

Para crear un paquete, ejecute:

```
$ runc spec
```

Este comando crea un archivo **config.json** que sólo contiene una estructura básica que tendrá que editar. Lo más importante es que tendrás que cambiar el parámetro **args** para identificar el ejecutable a ejecutar. Por defecto, **args** está configurado como **sh**.

```
  "args": [
    "sh"
  ],
```

Como ejemplo, puede descargar la imagen base de Red Hat Enterprise Linux (**ubi8/ubi**) usando podman y luego exportarla, crear un nuevo paquete para ella con runc, y editar el archivo **config.json** para que apunte a esa imagen. Luego puede crear la imagen del contenedor y ejecutar una instancia de esa imagen con runc. Utilice los siguientes comandos:

```
# podman pull registry.redhat.io/ubi8/ubi
# podman export $(podman create registry.redhat.io/ubi8/ubi) > rhel.tar
# mkdir -p rhel-runc/rootfs
# tar -C rhel-runc/rootfs -xf rhel.tar
# runc spec -b rhel-runc
# vi rhel-runc/config.json   Change any setting you like
# runc create -b rhel-runc/ rhel-container
# runc start rhel-container
sh-4.2#
```

En este ejemplo, el nombre de la instancia del contenedor es **rhel-container**. Ejecutar ese contenedor, por defecto, inicia un shell, por lo que puedes empezar a mirar y ejecutar comandos desde dentro de ese contenedor. Escriba **exit** cuando haya terminado.

El nombre de una instancia de contenedor debe ser único en el host. Para iniciar una nueva instancia de un contenedor:

```
# runc start <nombre_del_contenedor>
```

Puede proporcionar el directorio del paquete utilizando la opción **-b**. Por defecto, el valor del bundle es el directorio actual.

Necesitarás privilegios de root para iniciar contenedores con runc. Para ver todos los comandos disponibles para runc y su uso, ejecute **runc --help**.

11.3. SKOPEO

Con el comando **skopeo**, puedes trabajar con imágenes de contenedores desde registros sin usar el demonio docker o el comando **docker**. Los registros pueden incluir el Registro Docker, sus propios registros locales, los registros de Red Hat Quay u OpenShift. Las actividades que puedes hacer con skopeo incluyen:

- **inspect**: La salida de un comando **skopeo inspect** es similar a la de un comando **docker inspect**: información de bajo nivel sobre la imagen del contenedor. Esa salida puede estar en formato json (por defecto) o en formato raw (usando la opción **--raw**).
- **copy**: Con **skopeo copy** puede copiar una imagen de contenedor de un registro a otro registro o a un directorio local.
- **layers**: El comando **skopeo layers** permite descargar las capas asociadas a las imágenes para que se almacenen como bolas de tar y archivos de manifiesto asociados en un directorio local.

Al igual que el comando **buildah** y otras herramientas que dependen de la biblioteca containers/image, el comando **skopeo** puede trabajar con imágenes de áreas de almacenamiento de contenedores distintas a las asociadas a Docker. Los transportes disponibles a otros tipos de almacenamiento de contenedores incluyen: containers-storage (para imágenes almacenadas por **buildah** y CRI-O), ostree (para contenedores atómicos y de sistema), oci (para contenido almacenado en un directorio compatible con OCI), y otros. Consulte la página man de skopeo para más detalles.

Para probar skopeo, puedes configurar un registro local, y luego ejecutar los comandos que siguen para inspeccionar, copiar y descargar capas de imágenes. Si quieres seguir los ejemplos, empieza por hacer lo siguiente:

- Instale un registro local (como Red Hat Quay. El software de registro de contenedores disponible en el paquete de distribución de Docker para RHEL 7, no está disponible para RHEL 8.
- Extraiga la última imagen de RHEL a su sistema local (**podman pull ubi8/ubi**).
- Vuelva a etiquetar la imagen RHEL y póngala en su registro local de la siguiente manera:

```
# podman tag ubi8/ubi localhost/myubi8
# podman push localhost/myubi8
```

El resto de esta sección describe cómo inspeccionar, copiar y obtener capas de la imagen RHEL.

NOTA

La herramienta **skopeo** requiere por defecto una conexión TLS. Falla cuando intenta utilizar una conexión no cifrada. Para anular el valor predeterminado y utilizar un registro http, añada **http:** a la cadena **<registry>/<image>**.

11.3.1. Inspección de imágenes de contenedores con skopeo

Cuando se inspecciona una imagen de contenedor desde un registro, es necesario identificar el formato del contenedor (como docker), la ubicación del registro (como docker.io o localhost) y el repositorio/imagen (como ubi8/ubi).

El siguiente ejemplo inspecciona la imagen del contenedor mariadb desde el Registro Docker:

```
# skopeo inspect docker://docker.io/library/mariadb
{
    "Name": "docker.io/library/mariadb",
    "Tag": "latest",
    "Digest": "sha256:d3f56b143b62690b400ef42e876e628eb5e488d2d0d2a35d6438a4aa841d89c4",
    "RepoTags": [
        "10.0.15",
        "10.0.16",
        "10.0.17",
        "10.0.19",
...
    "Created": "2018-06-10T01:53:48.812217692Z",
    "DockerVersion": "1.10.3",
    "Labels": {},
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
...
```

Asumiendo que has empujado una imagen de contenedor con la etiqueta **localhost/myubi8** a un registro de contenedores que se ejecuta en tu sistema local, el siguiente comando inspecciona esa imagen:

```
# skopeo inspect docker://localhost/myubi8
{
    "Name": "localhost/myubi8",
    "Tag": "latest",
    "Digest": "sha256:4e09c308a9ddf56c0ff6e321d135136eb04152456f73786a16166ce7cba7c904",
    "RepoTags": [
        "latest"
    ],
    "Created": "2018-06-16T17:27:13Z",
    "DockerVersion": "1.7.0",
    "Labels": {
        "Architecture": "x86_64",
        "Authoritative_Registry": "registry.access.redhat.com",
        "BZComponent": "rhel-server-docker",
        "Build_Host": "rcm-img01.build.eng.bos.redhat.com",
        "Name": "myubi8",
        "Release": "75",
        "Vendor": "Red Hat, Inc.",
```

```
        "Version": "8.0"
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf"
    ]
}
```

11.3.2. Copiar imágenes de contenedores con skopeo

Este comando copia la imagen del contenedor myubi8 desde un registro local a un directorio del sistema local:

```
# skopeo copy docker://localhost/myubi8 dir:/root/test/
INFO[0000] Downloading
myubi8/blobs/sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf
# ls /root/test
16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf.tar manifest.json
```

El resultado del comando **skopeo copy** es un tarball (16d*.tar) y un archivo manifest.json que representa la imagen que se está copiando en el directorio que has identificado. Si hubiera varias capas, habría varios tarballs. El comando **skopeo copy** también puede copiar imágenes a otro registro. Si necesita proporcionar una firma para escribir en el registro de destino, puede hacerlo añadiendo una opción **--sign-by=** a la línea de comandos, seguida de la clave-id requerida.

11.3.3. Obtención de capas de imagen con skopeo

El comando **skopeo layers** es similar a **skopeo copy**, con la diferencia de que la opción **copy** puede copiar una imagen a otro registro o a un directorio local, mientras que la opción **layers** sólo deja las capas (tarballs y archivo manifest.json) en el directorio actual. Por ejemplo

```
# skopeo layers docker://localhost/myubi8
INFO[0000] Downloading
myubi8/blobs/sha256:16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf
# find .
./layers-myubi8-latest-698503105
./layers-myubi-latest-698503105/manifest.json
./layers-myubi8-latest-
698503105/16dc1f96e3a1bb628be2e00518fec2bb97bd5933859de592a00e2eb7774b6ecf.tar
```

Como puede ver en este ejemplo, se crea un nuevo directorio (**layers-myubi8-latest-698503105**) y, en este caso, se copian en ese directorio un tarball de una capa y un archivo **manifest.json**.

CAPÍTULO 12. USO DE LA API DE LAS HERRAMIENTAS PARA CONTENEDORES

La nueva API de Podman 2.0 basada en REST sustituye a la antigua API remota de Podman que utilizaba la librería varlink. La nueva API funciona tanto en un entorno rootful como en uno sin root.

La API RESTful de Podman v2.0 consiste en la API Libpod que proporciona soporte para Podman, y la API compatible con Docker.

Con esta nueva API REST, puedes llamar a Podman desde plataformas como cURL, Postman, el cliente REST avanzado de Google y muchas otras.

12.1. HABILITACIÓN DE LA API DE PODMAN MEDIANTE SYSTEMD EN MODO ROOT

Este procedimiento muestra cómo hacer lo siguiente:

1. Utiliza systemd para activar el socket de la API de Podman.
2. Utilice un cliente Podman para realizar comandos básicos.

Requisitos previos

- El paquete **podman-remote** está instalado.

```
# yum install podman-remote
```

Procedimiento

1. Configurar el archivo de unidad systemd para el socket Podman:

```
# cat /usr/lib/systemd/system/podman.socket

[Unit]
Description=Podman API Socket
Documentation=man:podman-api(1)

[Socket]
ListenStream=%t/podman/podman.sock
SocketMode=0660

[Install]
WantedBy=sockets.target
```

2. Recarga la configuración del gestor systemd:

```
# systemctl daemon-reload
```

3. Inicie el servicio inmediatamente:

```
# systemctl enable --now podman.socket
```

4. Para habilitar el enlace a **var/lib/docker.sock** utilizando el paquete **docker-podman**:

```
# yum install podman-docker
```

Pasos de verificación

- Muestra la información del sistema de Podman:

```
# podman-remote info
```

- Verifique el enlace:

```
# ls -al /var/run/docker.sock
lrwxrwxrwx. 1 root root 23 Nov  4 10:19 /var/run/docker.sock -> /run/podman/podman.sock
```

Recursos adicionales

- Para más información sobre la API de Podman 2.0, consulte la documentación de la API RESTful de Podman v2.0.
- Para más ejemplos sobre cómo utilizar la API de Podman 2. 0, consulte el artículo A First Look At Podman 2.0 API de Scott McCarty.
- Para ver más ejemplos de cómo utilizar la API de Podman 2.0, consulta el artículo Sneak peek: La nueva API REST de Podman, por Tom Sweeney.

12.2. HABILITACIÓN DE LA API DE PODMAN MEDIANTE SYSTEMD EN MODO SIN RAÍZ

Este procedimiento muestra cómo utilizar systemd para activar el socket de la API Podman y el servicio de la API Podman.

Requisitos previos

- El paquete **podman-remote** está instalado.

```
# yum install podman-remote
```

Procedimiento

1. Crear un socket sin raíces para Podman:

```
$ vim .config/systemd/user/podman.socket

[Unit]
Description=Podman API Socket
Documentation=man:podman-api(1)

[Socket]
ListenStream=/home/username/podman.sock
SocketMode=0660
```

```
[Install]
WantedBy=sockets.target
```

2. Crear un servicio sin raíces para Podman:

```
$ vim .config/systemd/user/podman.service
[Unit]
Description=Podman API Service
Requires=podman.socket
After=podman.socket
Documentation=man:podman-api(1)
StartLimitIntervalSec=0

[Service]
Type=oneshot
Environment=REGISTRIES_CONFIG_PATH=/etc/containers/registries.conf
ExecStart=/usr/bin/podman system service unix:///home/username/podman.sock
TimeoutStopSec=30
KillMode=process

[Install]
WantedBy=multi-user.target
Also=podman.socket
```

   - La línea **After** en las secciones **[Unit]** define la dependencia del archivo de la unidad **podman.socket**. La unidad **podman.socket** se inició antes que la configurada **podman.service**.

3. Recarga la configuración del gestor systemd:

```
$ systemctl --user daemon-reload
```

4. Habilitar e iniciar el servicio inmediatamente:

```
$ systemctl --user enable --now podman.socket
```

5. Para permitir que los programas que utilizan Docker interactúen con el socket Podman sin raíz:

```
$ export DOCKER_HOST=unix:///var/run/<username>/podman.sock
```

Pasos de verificación

- Muestra la información del sistema de Podman:

```
$ podman-remote info
```

Recursos adicionales

- Para más información sobre la API de Podman 2.0, consulte la documentación de la API RESTful de Podman v2.0.

- Para más ejemplos sobre cómo utilizar la API de Podman 2. 0, consulte el artículo A First Look At Podman 2.0 API de Scott McCarty.
- Para ver más ejemplos de cómo utilizar la API de Podman 2.0, consulta el artículo Sneak peek: La nueva API REST de Podman, por Tom Sweeney.
- Para ver ejemplos de uso de la API de Podman 2.0 con Python y Bash, consulta el artículo Exploring Podman RESTful API using Python and Bash de Jhon Honce.

12.3. EJECUCIÓN MANUAL DE LA API DE PODMAN

Este procedimiento describe cómo ejecutar la API de Podman. Esto es útil para depurar las llamadas a la API, especialmente cuando se utiliza la capa de compatibilidad de Docker.

Requisitos previos

- El paquete **podman-remote** está instalado.

```
# yum install podman-remote
```

Procedimiento

1. Ejecute el servicio para la API REST:

```
# podman system service -t 0 --log-level=debug
```

   - El valor 0 significa que no hay tiempo de espera. El punto final por defecto para un servicio rootful es **unix:/run/podman/podman.sock**.
   - La opción **--log-level <level>** establece el nivel de registro. Los niveles de registro estándar son **debug**, **info**, **warn**, **error**, **fatal** y **panic**.

2. En otro terminal, muestra la información del sistema de Podman. El comando **podman-remote**, a diferencia del comando normal **podman**, se comunica a través del socket de Podman:

```
# podman-remote info
```

3. Para solucionar los problemas de la API de Podman y mostrar las peticiones y respuestas, utilice el comando **curl**. Para obtener la información sobre la instalación de Podman en el servidor Linux en formato JSON:

```
# curl -s --unix-socket /run/podman/podman.sock http://d/v1.0.0/libpod/info | jq
   {
 "host": {
   "arch": "amd64",
   "buildahVersion": "1.15.0",
   "cgroupVersion": "v1",
   "conmon": {
     "package": "conmon-2.0.18-1.module+el8.3.0+7084+c16098dd.x86_64",
     "path": "/usr/bin/conmon",
     "version": "conmon version 2.0.18, commit:
7fd3f71a218f8d3a7202e464252aeb1e942d17eb"
   },
   …
```

```
  "version": {
    "APIVersion": 1,
    "Version": "2.0.0",
    "GoVersion": "go1.14.2",
    "GitCommit": "",
    "BuiltTime": "Thu Jan  1 01:00:00 1970",
    "Built": 0,
    "OsArch": "linux/amd64"
  }
}
```

La utilidad **jq** es un procesador JSON de línea de comandos.

4. Extraiga la imagen del contenedor **registry.access.redhat.com/ubi8/ubi**:

```
# curl -XPOST --unix-socket /run/podman/podman.sock -v 'http://d/v1.0.0/images/create?
fromImage=registry.access.redhat.com%2Fubi8%2Fubi'
*   Trying /run/podman/podman.sock...
* Connected to d (/run/podman/podman.sock) port 80 (#0)
> POST /v1.0.0/images/create?fromImage=registry.access.redhat.com%2Fubi8%2Fubi
HTTP/1.1
> Host: d
> User-Agent: curl/7.61.1
> Accept: /
>
< HTTP/1.1 200 OK
< Content-Type: application/json
< Date: Tue, 20 Oct 2020 13:58:37 GMT
< Content-Length: 231
<
{"status":"pulling image () from registry.access.redhat.com/ubi8/ubi:latest,
registry.redhat.io/ubi8/ubi:latest","error":"","progress":"","progressDetail":
{},"id":"ecbc6f53bba0d1923ca9e92b3f747da8353a070fccbae93625bd8b47dbee772e"}
* Connection #0 to host d left intact
```

5. Muestra la imagen extraída:

```
# curl --unix-socket /run/podman/podman.sock -v 'http://d/v1.0.0/libpod/images/json' | jq
*   Trying /run/podman/podman.sock...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to d
(/run/podman/podman.sock) port 80 (0) > GET /v1.0.0/libpod/images/json HTTP/1.1 > Host: d
> User-Agent: curl/7.61.1 > Accept: / > < HTTP/1.1 200 OK < Content-Type: application/json
< Date: Tue, 20 Oct 2020 13:59:55 GMT < Transfer-Encoding: chunked < { [12498 bytes
data] 100 12485 0 12485 0 0 2032k 0 --:--:-- --:--:-- --:--:-- 2438k * Connection #0 to host d
left intact [ { "Id":
"ecbc6f53bba0d1923ca9e92b3f747da8353a070fccbae93625bd8b47dbee772e",
"RepoTags": [ "registry.access.redhat.com/ubi8/ubi:latest", "registry.redhat.io/ubi8/ubi:latest"
], "Created": "2020-09-01T19:44:12.470032Z", "Size": 210838671, "Labels": { "architecture":
"x86_64", "build-date": "2020-09-01T19:43:46.041620", "com.redhat.build-host": "cpt-
1008.osbs.prod.upshift.rdu2.redhat.com", ... "maintainer": "Red Hat, Inc.", "name": "ubi8", ...
"summary": "Provides the latest release of Red Hat Universal Base Image 8.", "url":
"https://access.redhat.com/containers//registry.access.redhat.com/ubi8/images/8.2-347",
      ...
    },
```

```
    "Names": [
      "registry.access.redhat.com/ubi8/ubi:latest",
      "registry.redhat.io/ubi8/ubi:latest"
    ],
    ...
    ]
  }
]
```

Recursos adicionales

- Para más información sobre la API de Podman 2.0, consulte la documentación de la API RESTful de Podman v2.0.
- Para ver más ejemplos de cómo utilizar la API de Podman 2.0, consulta el artículo Sneak peek: La nueva API REST de Podman, por Tom Sweeney.
- Para ver ejemplos de uso de la API de Podman 2.0 con Python y Bash, consulta el artículo Exploring Podman RESTful API using Python and Bash de Jhon Honce.
- Para más información sobre el comando **podman system service**, consulte la página de manual **podman-system-service**.

CAPÍTULO 13. RECURSOS ADICIONALES

- Buildah - una herramienta para construir imágenes de contenedores OCI
- Podman - una herramienta para ejecutar y gestionar contenedores
- Skopeo - una herramienta para copiar e inspeccionar imágenes de contenedores