

# **CURSO CLOUD Y CONTAINERS.**

**Versión 1.0**

# Contenido

<b>Contenido .....</b>	<b>2</b>
<b><a href="#">Capítulo 1.</a> Introducción y descripción general de los contenedores .....</b>	<b>5</b>
Introducción a los contenedores .....	5
Objetivos .....	5
Descripción de los contenedores .....	5
Comparación de contenedores y máquinas virtuales .....	6
Desarrollo para contenedores.....	8
Introducción a Kubernetes y OpenShift .....	10
Descripción general de Kubernetes .....	10
Descripción general de Red Hat OpenShift Container Platform .....	11
<b>Capítulo 2. Conceptos básicos de Docker.....</b>	<b>13</b>
Creación de contenedores con Docker .....	13
Introducción a Docker .....	13
Trabajar con Docker .....	13
Docker Desktop.....	18
Ejercicio guiado: Creación de contenedores con Docker .....	21
Conceptos básicos sobre redes de contenedores .....	24
Conceptos básicos sobre redes de contenedores .....	24
Gestión de redes Docker .....	24
Habilitación de la resolución de nombres de dominio .....	26
Conexión de contenedores.....	26
Acceso a servicios de redes contenerizados .....	27
Reenvío de puertos .....	27
Redes en contenedores.....	28
Ejercicio guiado: Acceso a servicios de redes contenerizados .....	29
Acceso a los contenedores.....	34
Transparencia del contenedor .....	34
Introducción a las capas del contenedor .....	34
Iniciar procesos en contenedores .....	35
Abrir una sesión interactiva en contenedores .....	36
Copiar archivos dentro y fuera de contenedores .....	38
Ejercicio guiado: Acceso a los contenedores .....	39

Gestión del ciclo de vida del contenedor.....	42
Ciclo de vida del contenedor.....	42
Enumerar contenedores .....	43
Inspección de contenedores .....	44
Detención de contenedores sin problemas .....	46
Detención de contenedores de manera forzosa .....	47
Pausa de contenedores.....	47
Reinicio de contenedores.....	47
Eliminación de contenedores.....	48
Ejercicio guiado: Gestión del ciclo de vida del contenedor.....	49
<b>Capítulo 3. Container Images .....</b>	<b>53</b>
Registros de imágenes de contenedor .....	53
Registros de contenedor .....	53
Registro de Red Hat.....	53
Quay.io .....	56
Gestionar registros con Docker .....	56
Utilizar la CLI de Docker .....	57
Gestionar credenciales de registros con Docker .....	58
Ejercicio guiado: Registros de imágenes de contenedor.....	60
Administración de imágenes.....	62
Gestión de imágenes.....	62
Ejercicio guiado: Administración de imágenes .....	69
<b>Capítulo 4. Imágenes de contenedores personalizadas .....</b>	<b>73</b>
Crear imágenes con Containerfiles .....	73
Creación de imágenes con Containerfiles .....	73
Elección de una imagen base.....	73
Instrucciones de Containerfile .....	74
Etiquetas de imágenes de contenedores .....	77
Ejercicio guiado: Crear imágenes con Containerfiles .....	78
Compilación de imágenes con instrucciones avanzadas de Containerfile .....	86
Instrucciones avanzadas de Containerfile .....	86
La instrucción ENV .....	86
La instrucción ARG .....	87

La instrucción VOLUME .....	88
Las instrucciones ENTRYPOINT y CMD .....	89
Compilaciones de varias etapas .....	91
Examinar las capas de datos del contenedor .....	93
Ejercicio guiado: Compilación de imágenes con instrucciones avanzadas de Containerfile .....	95
<b>Capítulo 5. Persistencia de datos .....</b>	<b>100</b>
Montaje de volúmenes .....	100
Sistema de archivos de copia en escritura (COW).....	100
Almacenar datos en la máquina host.....	103
Almacenamiento de datos con montajes de enlace .....	104
Almacenamiento de datos con volúmenes .....	105
Almacenamiento de datos con un montaje tmpfs .....	105
Ejercicio guiado: Montaje de volúmenes .....	106
Trabajo con bases de datos.....	109
Contenedores de bases de datos con estado.....	109
Buenas prácticas para contenedores de bases de datos.....	110
Importar datos de la base de datos .....	110
Exportar datos de la base de datos.....	112
Contenedores de bases de datos de Red Hat .....	112
Ejercicio guiado: Trabajo con bases de datos.....	115
<b>Capítulo 6. Aplicaciones con múltiples contenedores con Compose.....</b>	<b>127</b>
Descripción general y casos de uso de Compose.....	127
Orqueste contenedores con Docker Compose.....	127
El archivo Compose .....	129
Redes .....	132
Volúmenes .....	133
Crear entornos de desarrollador con Compose .....	136
Docker Compose y Docker .....	136
Entornos de desarrollador de varios contenedores con Compose .....	136
Ejercicio guiado: Crear entornos de desarrollador con Compose.....	138

# Introducción y descripción general de los contenedores

## Introducción a los contenedores

### Objetivos

- Describir los conceptos básicos de los contenedores y en qué se diferencian de las máquinas virtuales.

### Descripción de los contenedores

En computación, un *contenedor* es un proceso encapsulado que incluye las dependencias de tiempo de ejecución necesarias para ejecutar el programa. En un contenedor, las librerías específicas de la aplicación son independientes de las librerías del sistema operativo del host. Librerías y funciones que no son especificadas por las aplicaciones en contenedores son provistas por el sistema operativo y el kernel. Las librerías y funciones proporcionadas ayudan a garantizar que el contenedor permanezca compacto y que pueda ejecutarse y detenerse rápidamente según sea necesario.

Un motor de contenedores crea un sistema de archivos de unión fusionando capas de imágenes de contenedores. Debido a que las capas de imágenes de contenedores son inmutables, un motor contenedor agrega una capa de escritura para modificaciones de archivos en tiempo de ejecución. Los contenedores son *efímeros* de forma predeterminada, lo que significa que el motor de contenedores elimina la capa de escritura cuando se elimina el contenedor.

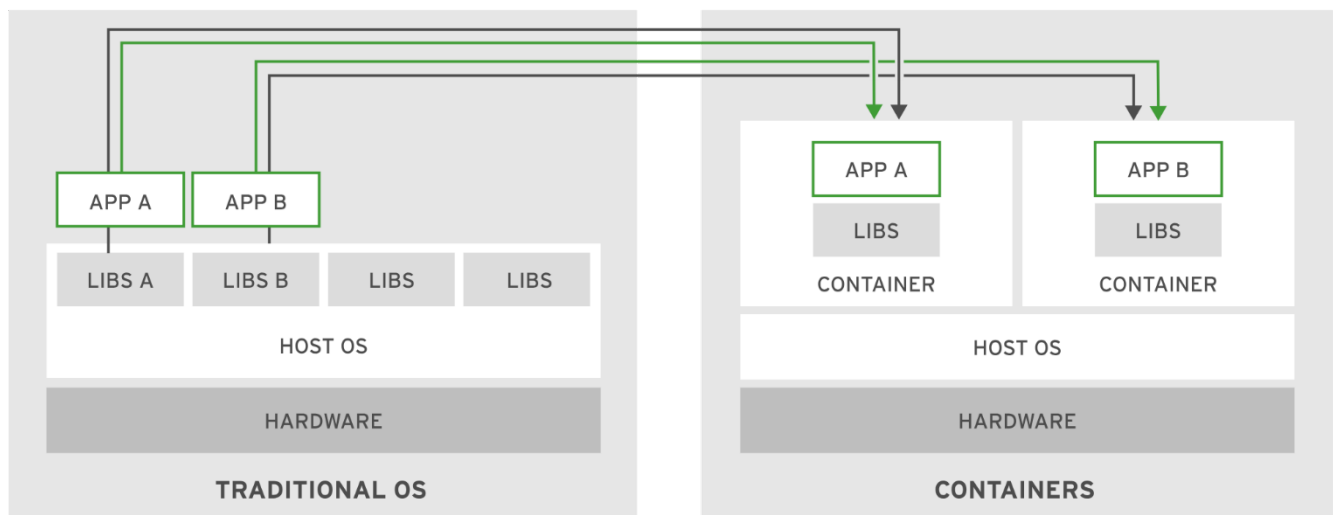


Figura 1.1: Aplicaciones en contenedores contra el sistema operativo host

Los contenedores usan características del kernel Linux, como espacios de nombres y grupos de control (cgroups). Por ejemplo, los contenedores usan grupos de control para la gestión de recursos, como la asignación de tiempo de CPU y la memoria del sistema. Los

espacios de nombres, en particular, brindan la funcionalidad para aislar los procesos dentro de los contenedores entre sí y del sistema host. Como tal, el entorno dentro de un contenedor está basado en Linux, independientemente del sistema operativo del host. Cuando se usan contenedores en sistemas operativos que no son Linux, estas características específicas de Linux a menudo se virtualizan mediante la implementación del motor de contenedores.

La contenerización se originó a partir de tecnologías como chroot, un método para aislar parcial o totalmente un entorno, y evolucionó a *Open Container Initiative (OCI)*, que es una organización de gestión que define estándares para crear y ejecutar contenedores. La mayoría de los motores de contenedores se ajustan a las especificaciones de OCI, por lo que los desarrolladores pueden crear con confianza sus artefactos de destino implementables para que se ejecuten como contenedores de OCI.

### ***Imágenes contra instancias***

Los contenedores se pueden dividir en dos ideas similares pero distintas: *imágenes de contenedor* e *instancias de contenedor*. Una *imagen de contenedor* contiene datos efectivamente inmutables que definen una aplicación y sus librerías. Puede usar las imágenes de contenedor para crear *instancias de contenedor*, que son versiones ejecutables de la imagen que incluyen referencias a redes, discos y otras necesidades de tiempo de ejecución.

Puede usar una sola imagen de contenedor varias veces para crear muchas instancias de contenedor distintas. También puede ejecutar estas instancias en varios hosts. La aplicación dentro de un contenedor es independiente del entorno del host.

#### **NOTA**

Las imágenes de contenedor de OCI se definen mediante la especificación *image-spec*, mientras que las instancias de contenedor de OCI se definen mediante la especificación *runtime-spec*.

Otra forma de pensar en las *imágenes de contenedor* frente a las *instancias de contenedor* es que una instancia se relaciona con una imagen como un objeto se relaciona con una clase en la programación orientada a objetos.

### **Comparación de contenedores y máquinas virtuales**

Los contenedores generalmente cumplen un rol similar a las *máquinas virtuales* (VM), donde una aplicación reside en un entorno autónomo con redes virtualizadas para la comunicación. Aunque este caso de uso inicialmente parece ser el mismo, los contenedores tienen un tamaño más pequeño y se inician y detienen más rápido que una máquina virtual.

Tanto para la memoria como para el uso del disco, las máquinas virtuales a menudo se miden en gigabytes, mientras que los contenedores se miden en megabytes.

Una máquina virtual es útil cuando se requiere un entorno de computación completo adicional, como cuando una aplicación requiere hardware específico y dedicado. Además, es preferible una máquina virtual cuando una aplicación requiere un sistema operativo que no sea Linux o un kernel diferente al del host.

### ***Máquinas virtuales contra contenedores***

Las máquinas virtuales y los contenedores usan un software diferente para la gestión y la funcionalidad. Los hipervisores, como KVM, Xen, VMware e Hyper-V, son aplicaciones que proporcionan la funcionalidad de virtualización para las máquinas virtuales. El equivalente en contenedor de un hipervisor es un motor de contenedores, como Docker.

	<b>Máquinas virtuales</b>	<b>Contenedores</b>
Funcionalidad a nivel de máquina	Hipervisor	Motor de contenedores
Gestión	Interfaz de gestión de VM	Motor de contenedores o software de orquestación
Nivel de virtualización	Entorno totalmente virtualizado	Solo partes relevantes
Tamaño	Medido en gigabytes	Medido en megabytes
Portabilidad	Generalmente solo el mismo hipervisor	Cualquier motor compatible con OCI

Puede gestionar hipervisores con software adicional de gestión, que se puede incluir con el hipervisor o ser externo, como Virtual Machine Manager con KVM. Por el contrario, puede gestionar los contenedores directamente a través del propio motor de contenedores. Además, puede usar herramientas de organización de contenedores, como Red Hat OpenShift Container Platform (RHOCP) y Kubernetes, para ejecutar y gestionar contenedores a escala. RHOCP gestiona tanto los contenedores como las máquinas virtuales desde una interfaz común.

Con las máquinas virtuales, la interoperabilidad es poco común. Por lo general, no se garantiza que una máquina virtual que se ejecuta en un hipervisor se ejecute en uno diferente. Por el contrario, los contenedores que siguen la especificación OCI no requieren un motor de contenedores en particular para funcionar. Muchos motores de contenedores pueden funcionar como reemplazos directos entre sí.

## ***Implementación a escala***

Tanto los contenedores como las máquinas virtuales pueden funcionar bien en diversas escalas. Debido a que un contenedor requiere muchos menos recursos que una máquina virtual, los contenedores tienen beneficios de rendimiento y recursos a mayor escala. Un método común en entornos a gran escala es usar contenedores que se ejecutan dentro de las máquinas virtuales. Esta configuración aprovecha los puntos fuertes de cada tecnología.

## **Desarrollo para contenedores**

La contenerización proporciona muchas ventajas para el proceso de desarrollo, como pruebas e implementación más sencillas, al proporcionar herramientas para la estabilidad, la seguridad y la flexibilidad.

## ***Pruebas y flujos de trabajo***

Una de las mayores ventajas de los contenedores para los desarrolladores es la capacidad de escalar. Un desarrollador puede escribir software, probarlo localmente y, luego, implementar la aplicación terminada en un servidor en la nube o en un clúster dedicado con pocos cambios o ninguno. Este flujo de trabajo es especialmente útil al crear microservicios, que son contenedores pequeños y efímeros que están diseñados para acelerar y desacelerar según sea necesario. Además, los desarrolladores que usan contenedores pueden aprovechar las tuberías (pipelines) de *Integración continua/Desarrollo continuo (CI/CD)* para implementar contenedores en diversos entornos. En particular, RHOCP ofrece diversas características de integración teniendo en cuenta las tuberías (pipelines) y los flujos de trabajo de CI/CD.

## ***Estabilidad***

Como se mencionó anteriormente, las imágenes de contenedores son un objetivo estable para los desarrolladores. Las aplicaciones de software requieren versiones de librerías específicas disponibles para la implementación, lo que puede dar lugar a problemas de dependencia o requisitos específicos del sistema operativo. Debido a que las librerías están incluidas en la imagen del contenedor, un desarrollador puede estar seguro de que no habrá problemas de dependencia en una implementación. Tener las librerías integradas dentro del contenedor elimina la variabilidad entre los entornos de prueba y de producción. Por ejemplo, un contenedor con una versión específica de Python garantiza que se use la misma versión de Python en todos los entornos de prueba o implementación.



## ***Aplicaciones con múltiples contenedores***

Una aplicación de múltiples contenedores se distribuye en varios contenedores. Puede ejecutar los contenedores de la misma imagen para réplicas de *alta disponibilidad (HA)* o de varias imágenes diferentes. Por ejemplo, un desarrollador puede crear una aplicación que incluya un contenedor de base de datos que se ejecuta por separado del contenedor de la API web de la aplicación. Una aplicación puede confiar en el software de gestión de contenedores para proporcionar réplicas de HA con opciones de contenedores múltiples, como Docker Pods o la especificación compose-spec.

### **REFERENCIAS**

[Tema de Red Hat para contenedores](#)

[Tema de Red Hat para la virtualización](#)

[Acerca de las especificaciones de OCI](#)

[Documentación oficial de Kubernetes](#)

# Introducción a Kubernetes y OpenShift

## Objetivos

- Describir la orquestación de contenedores y las características de Red Hat OpenShift.

## Descripción general de Kubernetes

Kubernetes es un servicio de orquestación que simplifica la implementación, la administración y el escalamiento de las aplicaciones contenerizadas. Gestiona conjuntos (pools) complejos de recursos, como CPU, RAM, almacenamiento y redes. Kubernetes proporciona un alto tiempo de actividad y tolerancia a fallas para las implementaciones de aplicaciones contenerizadas, lo que elimina la preocupación que pueden tener los desarrolladores con respecto a cómo sus aplicaciones usan los recursos.

La unidad gestionable más pequeña en Kubernetes es un pod, que representa una sola aplicación y consta de uno o más contenedores, incluidos los recursos de almacenamiento y una dirección IP.

## *Características de Kubernetes*

Los clústeres de Kubernetes proporcionan una plataforma de contenedores moderna que aborda las preocupaciones y los desafíos de ejecutar aplicaciones a escala. Independientemente del tamaño de la implementación, las implementaciones de Kubernetes ofrecen una infraestructura robusta y facilidad de gestión:

## **Detección de servicios y balanceo de carga**

Kubernetes habilita la comunicación entre servicios al asignar una sola entrada de DNS a cada conjunto de contenedores. Para permitir que el clúster cambie la ubicación y la dirección IP del contenedor, el servicio solicitante debe conocer el nombre DNS del objetivo. Como resultado, se puede balancear la carga de la solicitud en el conjunto de contenedores que brindan el servicio. Por ejemplo, Kubernetes puede dividir de manera uniforme las solicitudes entrantes a un servidor web NGINX teniendo en cuenta la disponibilidad de los pods NGINX.

## **Escalamiento horizontal**

Las aplicaciones pueden escalarse hacia arriba o hacia abajo de forma manual o automática con la configuración establecida con la interfaz de línea de comandos de Kubernetes o con la interfaz de usuario web.

## **Reparación automática**

Kubernetes puede usar controles de estado definidos por el usuario para monitorear los pods, reiniciarlos y reprogramarlos en caso de falla.

## **Implementación automatizada**

Kubernetes puede implementar gradualmente actualizaciones a los contenedores de su aplicación mientras comprueba su estado. Si algo sale mal durante la implementación, Kubernetes puede revertir a la versión anterior de la implementación.

## **Gestión de secretos y configuración**

Puede administrar los ajustes de configuración y los secretos de sus aplicaciones sin modificar los contenedores. Los secretos de las aplicaciones pueden ser nombres de usuario, contraseñas y extremos de servicio o cualquier configuración que deba mantenerse privada.

## **Operadores**

Los operadores son aplicaciones de Kubernetes empaquetadas que también llevan el conocimiento del ciclo de vida de la aplicación al clúster de Kubernetes. Las aplicaciones empaquetadas como operadores usan la API de Kubernetes para actualizar el estado del clúster y reaccionar a los cambios en el estado de la aplicación.

## **Descripción general de Red Hat OpenShift Container Platform**

Red Hat OpenShift Container Platform (RHOCP) es un conjunto de componentes y servicios modulares desarrollados sobre la base de una infraestructura de contenedores de Kubernetes. RHOCP agrega las capacidades para una plataforma de producción, como gestión remota, arquitectura multiinquilino (multitenant), mayor seguridad, monitoreo y auditoría, administración del ciclo de vida de la aplicación e interfaces de autoservicio para desarrolladores.

## ***Características de Red Hat OpenShift Container Platform***

RHOCP agrega las siguientes características a un clúster de Kubernetes:

### **Flujo de trabajo de desarrollador**

Integra un registro de contenedores incorporado, canalizaciones de *integración continua/entrega continua (CI/CD)* y *fuentes a imagen (S2I)*, una herramienta para crear artefactos desde repositorios de origen hasta imágenes de contenedores.

### **Rutas**

Expone los servicios al mundo exterior.

**Métricas y registro**

Incluye un servicio de métricas incorporado y con autoanálisis, y registro agregado.

**Interfaz de usuario unificada**

Ofrece herramientas unificadas y una interfaz de usuario integrada para gestionar las diferentes capacidades.

## Capítulo 2. Conceptos básicos de Docker

### Creación de contenedores con Docker

#### Objetivos

- Ejecutar un servicio contenerizado con Docker.

#### Introducción a Docker

Docker es una herramienta de código abierto que puede usar para gestionar sus contenedores localmente. Con Docker, puede buscar, ejecutar, compilar o implementar contenedores e imágenes de contenedores OCI (Open Container Initiative).

Docker viene en forma de interfaz de línea de comandos (CLI), que es compatible en varios sistemas operativos. Junto con la CLI, Docker proporciona dos formas adicionales de interactuar con sus contenedores y automatizar procesos, la API RESTful y una aplicación de escritorio llamada *Docker Desktop*.

#### Trabajar con Docker

Después de instalar Docker, puede usarlo ejecutando el comando `Docker`. El siguiente comando muestra la versión que está usando.

```
[user@host ~]$ docker -v
Docker version 24.0.5, build ced0996
```

#### *Extracción y visualización de imágenes*

Antes de poder ejecutar su aplicación en un contenedor, debe crear una imagen de contenedor.

Con Docker, obtiene imágenes de contenedores de registros de imágenes mediante el comando `Docker pull`. Por ejemplo, el siguiente comando obtiene una versión contenerizada de Red Hat Enterprise Linux 7 del registro de Red Hat.

```
[user@host ~]$ docker pull registry.redhat.io/rhel7/rhel:7.9
Trying to pull registry.redhat.io/rhel7/rhel:7.9...
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
b85986059f7663c1b89431f74cdcb783f6540823e4b85c334d271f2f2d8e06d6
```

Se hace referencia a una imagen de contenedor con el formato *NAME: VERSION*. En el ejemplo anterior, obtiene la versión 7.9 de la imagen `registry.redhat.io/rhel7/rhel`.

Después de la ejecución del comando `pull`, la imagen se almacena localmente en su sistema. Puede enumerar las imágenes en su sistema con el comando `Docker images`.

```
[user@host ~]$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
registry.redhat.io/rhel7/rhel	7.9	52617ef413bd	4 weeks ago	216 MB

### ***Contenedor e imágenes del contenedor***

Un contenedor es un entorno de tiempo de ejecución aislado donde las aplicaciones se ejecutan como procesos aislados. El aislamiento del entorno de ejecución garantiza que no interrumpan otros contenedores o procesos del sistema.

Una imagen de contenedor contiene una versión empaquetada de su aplicación, con todas las dependencias necesarias para que la aplicación se ejecute. Las imágenes pueden existir sin contenedores, pero los contenedores dependen de las imágenes porque usan imágenes de contenedor para crear un entorno de tiempo de ejecución para ejecutar aplicaciones.

### ***Ejecución y visualización de contenedores***

Con la imagen almacenada en su sistema local, puede usar el comando `Docker run` para crear un nuevo contenedor que use la imagen. La imagen RHEL de ejemplos anteriores acepta comandos Bash como argumento. Estos comandos se proporcionan como un argumento y se ejecutan dentro de un contenedor RHEL.

```
[user@host ~]$ docker run registry.redhat.io/rhel7/rhel:7.9 echo 'Red Hat'
```

Red Hat

En el ejemplo anterior, el comando `echo 'Red Hat'` se proporciona como un argumento para el comando `Docker run`. Docker ejecuta el comando `echo` dentro del contenedor RHEL y muestra la salida del comando.

#### **NOTA**

Si ejecuta un contenedor desde una imagen que no está almacenada en su sistema, Docker intenta extraer la imagen antes de ejecutar el contenedor. Por lo tanto, no es necesario ejecutar primero el comando `pull`.

Cuando el contenedor finaliza la ejecución de `echo`, el contenedor se detiene porque ningún otro proceso lo mantiene en ejecución. Puede enumerar los contenedores en ejecución mediante el comando `Docker ps`.

```
[user@host ~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

De forma predeterminada, el comando `Docker ps` enumera los siguientes detalles para sus contenedores.

- el ID de contenedor
- el nombre de la imagen que está usando el contenedor
- el comando que está ejecutando el contenedor
- la hora en que se creó el contenedor
- el estado del contenedor
- los puertos expuestos en el contenedor
- el nombre del contenedor.

Sin embargo, *detener* un contenedor no es lo mismo que *removerlo*. Aunque el contenedor está detenido, Docker no lo remueve. Puede enumerar todos los contenedores (en ejecución y detenidos) agregando el indicador `--all` al comando `Docker ps`.

```
[user@host ~]$ docker ps --all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
20236410bcef	registry.redhat.io/rhel7/rhel:7.9	echo Red Hat	1 second ago	Exited (0) 1 s
	hungry_mclaren			

También puede eliminar automáticamente un contenedor cuando sale con la opción `--rm` al comando `Docker run`.

```
[user@host ~]$ docker run --rm registry.redhat.io/rhel7/rhel:7.9 echo 'Red Hat'
```

```
Red Hat
```

```
[user@host ~]$ docker ps --all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			

El ciclo de vida del contenedor se trata más adelante en este curso.

Si no se proporciona un nombre durante la creación de un contenedor, Docker genera un nombre de cadena aleatorio para el contenedor. Es importante definir un nombre único para facilitar la identificación de sus contenedores a la hora de gestionar su ciclo de vida.

Puede asignar un nombre a sus contenedores agregando el indicador `--name` al comando `Docker run`.

```
[user@host ~]$ docker run --name Docker_rhel7 \
registry.redhat.io/rhel7/rhel:7.9 echo 'Red Hat'
Red Hat
```

Docker puede identificar los contenedores por el identificador corto *identificador único universal (UUID)*, que está compuesto por doce caracteres alfanuméricos, o por el identificador largo UUID, que está compuesto por 64 caracteres alfanuméricos, como se muestra en el ejemplo.

```
[user@host ~]$ docker ps --all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
20236410bcef second ago	registry.redhat.io/rhel7/rhel:7.9 Docker_rhel7	echo Red Hat	1 second ago	Exited (0) 1

En este ejemplo, `20236410bcef` es el ID del contenedor o el identificador corto UUID. Además, `Docker_rhel7` aparece como el nombre del contenedor.

Si desea recuperar el ID de contenedor largo de UUID, puede agregar el indicador `--format=json` al comando `Docker ps --all`.

```
[user@host ~]$ docker ps --all --format=json
{
  "AutoRemove": false,
  "Command": [
    "echo",
    "Red Hat"
  ],
  ...output omitted...

  "Id": "2023...b0b2",
  "Image": "registry.redhat.io/rhel7/rhel:7.9",
```



```
...output omitted...
```

ID de contenedor largo UUID.

Puede recuperar información sobre un contenedor en formato JSON o como plantilla de Go.

### ***Exposición de contenedores***

Muchas aplicaciones, como los servidores web o las bases de datos, siguen ejecutándose indefinidamente a la espera de las conexiones. Por lo tanto, los contenedores para estas aplicaciones deben ejecutarse indefinidamente. Al mismo tiempo, suele ser necesario que se acceda a estas aplicaciones de forma externa a través de un protocolo de red.

Puede usar la opción `-p` para asignar un puerto en su máquina local a un puerto dentro del contenedor. De esta manera, el tráfico en su puerto local se reenvía al puerto dentro del contenedor, lo que le permite acceder a la aplicación desde su computadora.

El siguiente ejemplo crea un nuevo contenedor que ejecuta un servidor HTTP Apache al mapear el puerto 8080 en su máquina local al puerto 8080 dentro del contenedor.

```
[user@host ~]$ docker run -p 8080:8080 \
registry.access.redhat.com/ubi8/httpd-24:latest
...output omitted...
[Thu Apr 21 12:58:57.048491 2022] [ssl:warn] [pid 1:tid 140257793613248] AH01909: 10.0.2.10
0:8443:0 server certificate does NOT include an ID which matches the server name
[Thu Apr 21 12:58:57.048899 2022] [[:notice] [pid 1:tid 140257793613248] ModSecurity for Apa
che/2.9.2 (http://www.modsecurity.org/) configured.
...output omitted...
[Thu Apr 21 12:58:57.136272 2022] [mpm_event:notice] [pid 1:tid 140257793613248] AH00489: A
pache/2.4.37 (Red Hat Enterprise Linux) OpenSSL/1.1.1k configured -- resuming normal operat
ions
[Thu Apr 21 12:58:57.136332 2022] [core:notice] [pid 1:tid 140257793613248] AH00094: Comman
d line: 'httpd -D FOREGROUND'
```

Puede acceder al servidor HTTP en `localhost:8080`.

Si desea que el contenedor se ejecute en segundo plano, para evitar que se bloquee el terminal, puede usar la opción `-d`.

```
[user@host ~]$ docker run -d -p 8080:8080 \
registry.access.redhat.com/ubi8/httpd-24:latest
```

b7eb467781106e4f416ba79cede91152239bfc74f6a570c6d70baa4c64fa636a

Las capacidades de red de Docker se tratan más adelante en este curso.

### ***Uso de variables del entorno***

Las variables de entorno son variables usadas en sus aplicaciones que se establecen fuera del programa. El sistema operativo o el entorno donde se ejecuta la aplicación proporciona el valor de la variable. Puede acceder a la variable de entorno en su aplicación en el tiempo de ejecución. Por ejemplo, en Node.js, puede acceder a las variables del entorno mediante `process.env.VARIABLE_NAME`.

Las variables de entorno son una forma útil y segura de insertar valores de configuración específicos del entorno en su aplicación. Por ejemplo, su aplicación puede usar un nombre de host de base de datos que sea diferente para cada entorno de aplicación, como los nombres de host `database.local`, `database.stage` o `database.test`.

Puede pasar variables de entorno a un contenedor con la opción `-e`. En el siguiente ejemplo, se pasa una variable de entorno denominada `NAME` con el valor `Red Hat`. Luego, la variable del entorno se imprime con el comando `printenv` dentro del contenedor.

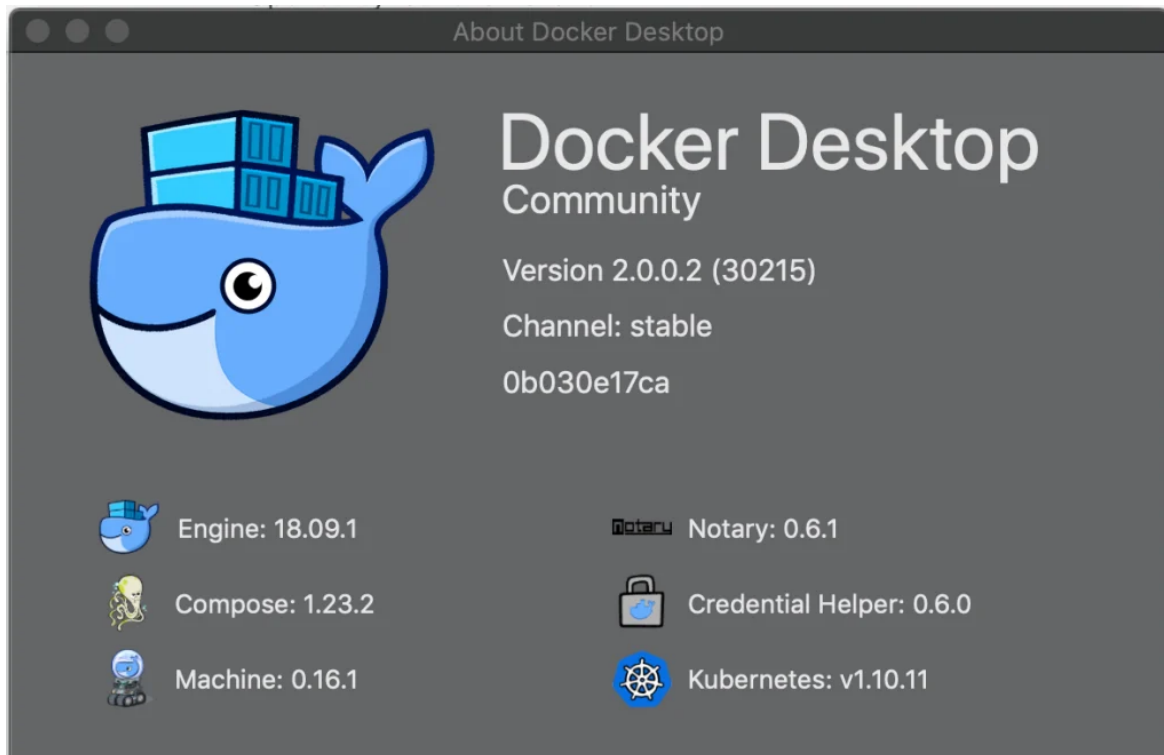
```
[user@host ~]$ docker run -e NAME='Red Hat' \
registry.redhat.io/rhel7/rhel:7.9 printenv NAME
Red Hat
```

Las variables del entorno se tratan más adelante en este curso.

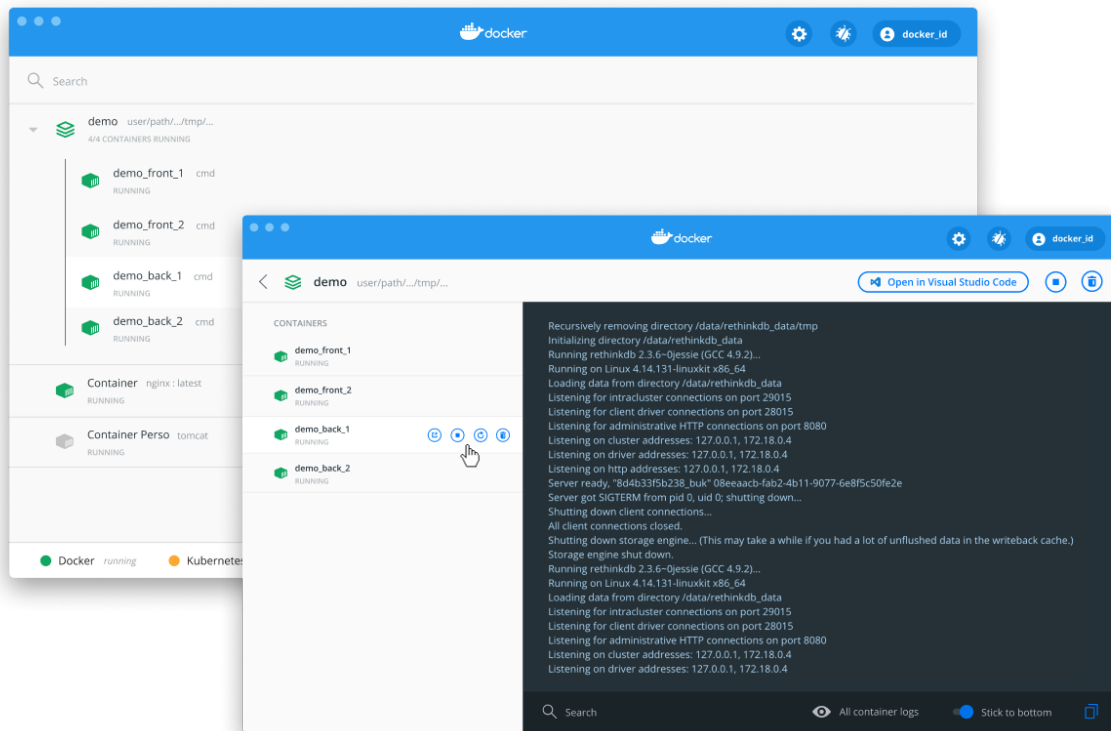
### **Docker Desktop**

*Docker Desktop* es una interfaz gráfica de usuario que se utiliza para administrar e interactuar con contenedores en entornos locales. Utiliza el motor Docker de forma predeterminada y también es compatible con otros motores de contenedores, como Docker.

Al iniciarse, Docker Desktop muestra un panel con información sobre el estado del motor Docker. El panel puede mostrar advertencias o errores, por ejemplo, si el motor Docker no está instalado o si la compatibilidad con Docker no está completamente configurada.



Con Docker Desktop, puede realizar muchas de las tareas que puede realizar con la CLI Docker, como extraer imágenes y crear contenedores. Por ejemplo, puede crear, enumerar y ejecutar contenedores desde la sección Containers, como muestra la siguiente imagen:



De manera similar, puede extraer, enumerar imágenes y crear contenedores a partir de esas imágenes en la sección Images.

Docker Desktop es una adición a la CLI Docker, en lugar de un reemplazo. Para los comandos u opciones más avanzados, que se tratan más adelante en el curso, se requiere la CLI. Aun así, Docker Desktop puede ser útil para usuarios que prefieren entornos gráficos para tareas comunes y para principiantes que están aprendiendo sobre contenedores.

Docker Desktop es modular y extensible. Puede usar y crear extensiones que brinden capacidades adicionales. Por ejemplo, con la extensión *Red Hat OpenShift*, Docker Desktop puede implementar contenedores en Red Hat OpenShift Container Platform.

Docker Desktop está disponible para Linux, MacOS y Windows. Para obtener instrucciones de instalación específicas, consulte la documentación de Docker Desktop.

## REFERENCIAS

[Documentación oficial de Docker](#)

## Ejercicio guiado: Creación de contenedores con Docker

Crear varios contenedores de Docker con diferentes opciones.

### Resultados

Debería poder ejecutar contenedores en Docker con el comando `docker run`.

### Procedimiento 2.1. Instrucciones

1. Use la imagen `registry.ocp4.example.com:8443/ubi8/ubi-minimal:8.5`, para crear un nuevo contenedor que imprima el texto `Hello Red Hat`.

Use el comando `docker pull` para obtener la imagen del registro.

```
[student@workstation ~]$ docker pull \
registry.ocp4.example.com:8443/ubi8/ubi-minimal:8.5
Trying to pull registry.ocp4.example.com:8443/ubi8/ubi-minimal:8.5...
...output omitted...
Writing manifest to image destination
Storing signatures
```

```
a0c247bdbd39ebaf1d6c8a2b42573311b3052ed0d67cc8eb7518b47f5e0eeca6
```

1. Use el comando `docker images` para verificar que la imagen esté disponible localmente.

```
[student@workstation ~]$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
registry.access.redhat.com/ubi8/ubi-minimal	8.5	a0c247bdbd39	4 weeks ago
			107 MB

2. En una terminal de línea de comandos, use `docker run` para crear un nuevo contenedor. Proporcione un comando `echo` para que se ejecute dentro del contenedor.

```
[student@workstation ~]$ docker run --rm \
registry.ocp4.example.com:8443/ubi8/ubi-minimal:8.5 echo 'Hello Red Hat'
Hello Red Hat
```

3. Verifique que el contenedor no esté ejecutándose después de que finalice la ejecución.

```
[student@workstation ~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

2. Explore las variables de entorno de configuración e impresión mediante el uso de la imagen del contenedor `registry.ocp4.example.com:8443/ubi8/ubi-minimal:8.5`.

1. Use la opción `-e` en el comando `docker run` para configurar las variables de entorno. Use el comando `printenv` que está empaquetado en la imagen del contenedor para imprimir los valores de las variables del entorno.

```
[student@workstation ~]$ docker run --rm -e GREET=Hello -e NAME='Red Hat' \
registry.ocp4.example.com:8443/ubi8/ubi-minimal:8.5 printenv GREET NAME
Hello
```

```
Red Hat
```

2. Verifique que el contenedor no esté ejecutándose después de que finalice la ejecución.

```
[student@workstation ~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

3. Al usar la imagen `registry.ocp4.example.com:8443/ubi8/httpd-24`, ejecute un nuevo contenedor que cree un servidor HTTP de Apache.

1. Use el comando `docker run` para extraer la imagen del contenedor e iniciar el contenedor `httpd`. Use la opción `-p` en el comando `docker run` para redirigir el tráfico desde el puerto `8080` en su máquina al puerto `8080` dentro del contenedor.

```
[student@workstation ~]$ docker run --rm -p 8080:8080 \
registry.ocp4.example.com:8443/ubi8/httpd-24
```

```
[Thu Apr 21 12:58:57.048491 2022] [ssl:warn] [pid 1:tid 140257793613248] AH0190
9: 10.0.2.100:8443:0 server certificate does NOT include an ID which matches th
e server name
```

```
[Thu Apr 21 12:58:57.048899 2022] [:notice] [pid 1:tid 140257793613248] ModSecurity for Apache/2.9.2 (http://www.modsecurity.org/) configured.
```

*...output omitted...*

```
[Thu Apr 21 12:58:57.136272 2022] [mpm_event:notice] [pid 1:tid 140257793613248] AH00489: Apache/2.4.37 (Red Hat Enterprise Linux) OpenSSL/1.1.1k configured -  
- resuming normal operations
```

```
[Thu Apr 21 12:58:57.136332 2022] [core:notice] [pid 1:tid 140257793613248] AH00094: Command line: 'httpd -D FOREGROUND'
```

El contenedor sigue ejecutándose y los registros se muestran en su terminal.

2. En un explorador web, diríjase a `http://localhost:8080`. Verifique que el servidor HTTP se esté ejecutando en el puerto 8080.
3. Regrese a su terminal de línea de comandos. Presione `Ctrl + C` para detener el contenedor.

*...output omitted...*

^C

```
[Thu Apr 21 13:00:52.516503 2022] [mpm_event:notice] [pid 1:tid 140481583287744] AH00491: caught SIGTERM, shutting down
```

4. Cree el contenedor nuevamente agregando la opción `-d`. El contenedor se ejecuta en segundo plano.

```
[student@workstation ~]$ docker run --rm -d \
-p 8080:8080 registry.ocp4.example.com:8443/ubi8/httpd-24
```

```
c1db10ec1ba61afcc7bb482b8d9cd42995473a4f4523ae15f3253e896a45d61a
```

5. Verifique que el contenedor esté ejecutándose en segundo plano.

```
[student@workstation ~]$ docker ps
```

```
c1db10ec1ba6 registry.ocp4.example.com:8443/ubi8/httpd-24:latest /usr/bin
/run-http... About a minute ago Up About a minute ago 0.0.0.0:8080->8080
/tcp distracted_lumiere
```

## Finalizar

- Eliminar todos los containers que se crearon temporalmente.

## Conceptos básicos sobre redes de contenedores

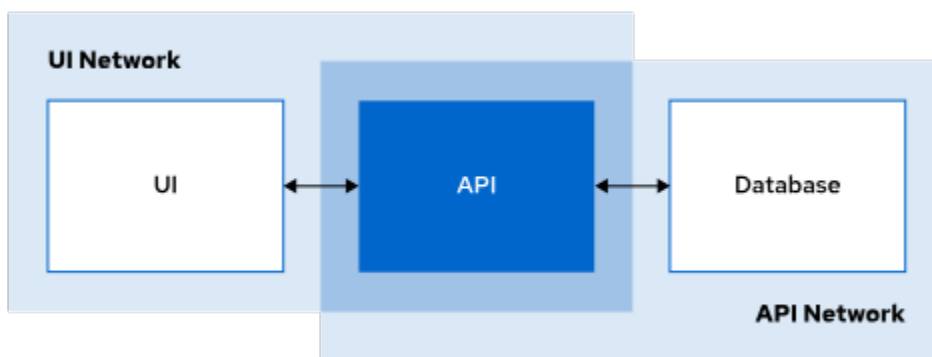
### Objetivos

- Describir cómo los contenedores se comunican entre sí.

### Conceptos básicos sobre redes de contenedores

Docker viene con una red llamada `docker`. De forma predeterminada, los contenedores están conectados a esta red y pueden usarla para comunicarse entre sí.

Sin embargo, a menudo necesitará crear una nueva red docker para adaptarse mejor a las crecientes necesidades de comunicación de la mayoría de las aplicaciones. Por ejemplo, los contenedores que ejecutan una aplicación API y una base de datos pueden usar una red Docker separada para aislar su comunicación de otros contenedores. De manera similar, ese mismo contenedor de API puede usar otra red para aislar la comunicación con un tercer contenedor que aloja la interfaz de usuario de la aplicación.



Ejemplo de comunicación aislada mediante el uso de redes docker

En el diagrama de ejemplo anterior, los contenedores de la interfaz de usuario y la API están conectados a la red Docker `ui-network`. Los contenedores de la base de datos y la API están conectados a la red Docker `api-network`.

### Gestión de redes Docker

La gestión de la red Docker se realiza mediante el subcomando `docker network`. Este subcomando incluye las siguientes operaciones:

`docker network create`

Crea una nueva red Docker. Este comando acepta diversas opciones para configurar las propiedades de la red, incluida la dirección de la puerta de enlace, la máscara de subred y si se debe usar IPv4 o IPv6.

`docker network ls`



Enumera las redes existentes y un breve resumen de cada una. Las opciones para este comando incluyen diversos filtros y un formato de salida para enumerar otros valores para cada red.

```
docker network inspect
```

Genera un objeto JSON detallado que contiene datos de configuración para la red.

```
docker network rm
```

Elimina una red.

```
docker network prune
```

Elimina las redes que no están actualmente en uso por ningún contenedor en ejecución.

```
docker network connect
```

Conecta un contenedor que ya se está ejecutando hacia o desde una red existente. Alternativamente, conecte contenedores a una red Docker en la creación de contenedores con la opción `--net`. El comando `disconnect` desconecta un contenedor de una red.

Por ejemplo, el siguiente comando crea una nueva red Docker llamada `example-net`:

```
[user@host ~]$ docker network create example-net
```

Para conectar un nuevo contenedor a esta red Docker, use la opción `--net`. El siguiente comando de ejemplo crea un nuevo contenedor llamado `my-container`, que está conectado a la red `example-net`.

```
[user@host ~]$ docker run -d --name my-container \
--net example-net container-image:latest
```

Cuando crea nuevos contenedores, puede conectarlos a varias redes especificando los nombres de las redes en una lista separada por comas. Por ejemplo, el siguiente comando crea un nuevo contenedor llamado `double-connector` que se conecta a ambas redes: `postgres-net` y `redis-net`.

```
[user@host ~]$ docker run -d --name double-connector \
--net postgres-net,redis-net \
container-image:latest
```

Alternativamente, si el contenedor `my-container` ya se está ejecutando, ejecute el siguiente comando para conectarlo a la red `example-net`:

```
[user@host ~]$ docker network connect example-net my-container
```

## Habilitación de la resolución de nombres de dominio

Cuando usa la red Docker predeterminada, el sistema de nombres de dominio (DNS) para otros contenedores en esa red está deshabilitado. Para habilitar la resolución DNS entre contenedores, cree una nueva red Docker y conecte sus contenedores a esa red.

Cuando se usa una red con DNS habilitado, el nombre de host de un contenedor es el nombre asignado al contenedor. Por ejemplo, si un contenedor se inicia con el siguiente comando, los otros contenedores en la red `test-net` pueden realizar solicitudes usando el nombre de host `basic-container`. El nombre de host `basic-container` se resuelve en la dirección IP actual del contenedor `basic-container`.

```
[user@host ~]$ docker run --net test-net --name basic-container example-image
```

## Conexión de contenedores

Puede conectar los contenedores a una o más redes docker. Luego de que un contenedor se conecta a una red, puede comunicarse con otros contenedores en esa red. Sin embargo, aunque los contenedores sean accesibles entre sí, otros componentes pueden impedir las conexiones. Por ejemplo, las reglas de firewall pueden bloquear una conexión proveniente de otro contenedor. De forma predeterminada, un contenedor está disponible dentro de cualquier red a la que se conecte.

Por ejemplo, considere un contenedor en ejecución llamado `nginx-host` que usa la red `example-net`. El contenedor expone un servidor HTTP en el puerto 8080. Dentro de otro contenedor que usa la red `example-net`, el siguiente comando `curl` se resuelve en la root del servidor HTTP.

```
[user@host ~]$ curl http://nginx-host:8080
```

## Acceso a servicios de redes contenerizados

### Objetivos

- Exponer puertos para acceder a servicios contenerizados.

### Reenvío de puertos

El espacio de nombres de red de un contenedor está aislado, lo que significa que solo puede accederse a una aplicación en red dentro del contenedor. *El reenvío de puertos* mapea un puerto desde la máquina host donde el contenedor se ejecuta a un puerto dentro de un contenedor.

La opción `-p` del comando `docker run` reenvía un puerto. La opción acepta la forma `HOST_PORT:CONTAINER_PORT`.

Por ejemplo, el siguiente comando asigna el puerto 80 dentro del contenedor al puerto 8075 en la máquina host.

```
[user@host ~]$ docker run -p 8075:80 my-app
```

Sin un host especificado, al contenedor se le asigna la dirección de difusión (0.0.0.0). Esto significa que se puede acceder al contenedor desde todas las redes de la máquina host.

Para publicar un contenedor en un host específico y limitar las redes desde las que puede accederse, use la siguiente forma.

```
[user@host ~]$ docker run -p 127.0.0.1:8075:80 my-app
```

El puerto 80 en el contenedor `my-app` está disponible desde el puerto 8075 únicamente desde la máquina host, a la que puede accederse a través de la dirección IP 127.0.0.1 de localhost.

### ***Enumerar mapeos de puertos***

Para enumerar los mapeos de puertos para un contenedor, use el comando `docker port`. Por ejemplo, el siguiente comando revela que el puerto 8010 de la máquina host está mapeado al puerto 8008 dentro del contenedor.

```
[user@host ~]$ docker port my-app  
8008/tcp -> 0.0.0.0:8010
```

La opción `--all` muestra una lista de los mapeos de puertos de todos los contenedores.

```
[user@host ~]$ docker port --all
```

```
1aacd9cf1c76      8008/tcp -> 0.0.0.0:8010
```

## NOTA

En la salida del ejemplo anterior, 1aacd9cf1c76 se refiere al ID del contenedor.

## Redes en contenedores

A los contenedores conectados a las redes Docker se les asignan direcciones IP privadas para cada red. Otros contenedores de la red pueden realizar solicitudes a esta dirección IP.

Por ejemplo, un contenedor llamado `my-app` se conecta a la red `apps`. El siguiente comando recupera la dirección IP privada del contenedor dentro de la red `apps`.

```
[user@host ~]$ docker inspect my-app \
  -f '{{.NetworkSettings.Networks.apps.IPAddress}}'
10.89.0.2
```

## Ejercicio guiado: Acceso a servicios de redes containerizados

Ejecutar dos aplicaciones que se comunican mediante el sistema DNS de Docker.

### Resultados

Debería poder comprender cómo funciona el DNS en las redes Docker.

Este ejercicio usa dos aplicaciones: `times` y `cities`. La aplicación `times` devuelve la hora actual de una ciudad determinada y la aplicación `cities` devuelve información sobre una ciudad específica.

Para obtener la hora actual de la ciudad, la aplicación `cities` obtiene datos de la aplicación `times`.

Clonar el repositorio de ejemplos:

```
git clone https://github.com/arrsvjes2/cdiputadosdocker.git
```

### Procedimiento 2.2. Instrucciones

1. Analice el código fuente de las aplicaciones.
  1. Navegue al directorio `cd cdiputadosdocker/basics-exposing/`, donde están disponibles las aplicaciones.

```
[student@workstation ~]$ cd cdiputadosdocker/basics-exposing/  
no output expected
```

2. Vea el contenido del archivo `docker-info-times/app/main.go`.

```
[student@workstation basics-exposing]$ cat docker-info-times/app/main.go  
...output omitted...  
    http.ListenAndServe(":8080", r)  
...output omitted...
```

La aplicación expone un servidor HTTP en el puerto 8080 que devuelve la hora actual de una ciudad.

3. Vea el archivo `docker-info-cities/app/main.go`.

```
[student@workstation basics-exposing]$ cat docker-info-cities/app/main.go  
...output omitted...  
    http.ListenAndServe(":8090", r)  
...output omitted...
```

La aplicación expone un servidor HTTP en el puerto 8090 que devuelve información de una ciudad.

4. Diríjase al directorio de inicio. El resto del ejercicio no involucra archivos locales.

```
[student@workstation basics-exposing]$ cd ~  
no output expected
```

2. Cree una red Docker con DNS habilitado.

1. Observe que DNS está deshabilitado en la red Docker predeterminada.

```
[student@workstation ~]$ docker network inspect docker  
...output omitted...  
  "dns_enabled": false,  
...output omitted...
```

2. Cree una red con el comando `docker network create`.

```
[student@workstation ~]$ docker network create cities  
cities
```

3. Inspeccione la red `cities` para verificar que el DNS esté habilitado.

```
[student@workstation ~]$ docker network inspect cities  
...output omitted...  
  "dns_enabled": true,  
...output omitted...
```

3. Inicie y pruebe la aplicación `times` conectada a la red `cities`.

1. Cree un contenedor para la aplicación `times` que reenvía el puerto 8080 desde el contenedor al host. Conecte el contenedor a la red `cities`.

```
[student@workstation ~]$ docker run --name times-app \  
--network cities -p 8080:8080 -d \  
quay.io/piracarter1/docker-info-times:v0.1  
...output omitted...
```

```
256...ee0
```

2. Inspeccione el contenedor times-app para encontrar su IP privada dentro de la red Docker.

```
[student@workstation ~]$ docker inspect times-app \
-f '{{.NetworkSettings.Networks.cities.IPAddress}}'
172.19.0.2
```

Copie la dirección IP.

3. Use la dirección IP privada para realizar una solicitud al extremo de la API /times desde otro contenedor. Ejecute el comando curl dentro de un contenedor ubi-minimal para obtener la hora actual de Bangkok, que tiene el código BKK. Conecte el contenedor a la red cities.

Reemplace IP\_ADDRESS con la dirección IP del paso anterior.

```
[student@workstation ~]$ docker run --rm --network cities \
quay.io/piracarter1/ubi-minimal:8.5 \
curl -s http://IP_ADDRESS:8080/times/BKK && echo
2023-08-30T04:39:58.047Z
```

```
268/
```

## NOTA

Sin proporcionar argumentos, el comando echo imprime un carácter de nueva línea, lo que mejora el formato de salida.

4. Use el nombre de DNS para realizar una solicitud al extremo de la API /times desde otro contenedor.

```
[student@workstation ~]$ docker run --rm --network cities \
quay.io/piracarter1/ubi-minimal:8.5 \
curl -s http://times-app:8080/times/BKK && echo
2023-08-30T05:06:44.776Z
```

Tenga en cuenta que `times-app` se usa como nombre de host en lugar de la dirección IP.

4. Inicie y pruebe la aplicación `cities` conectada a la red `cities`.

1. Cree un contenedor denominado `cities-app` e inicie la aplicación en el puerto `8090`. Conecte el contenedor a la red `cities`.

Defina la variable de entorno `TIMES_APP_URL` en la URL de la aplicación `times`. Observe que la URL usa el nombre del contenedor `times-app`.

```
[student@workstation ~]$ docker run --name cities-app \
--network cities -p 8090:8090 -d \
-e TIMES_APP_URL=http://times-app:8080/times \
quay.io/piracarter1/docker-info-cities:v0.1
...output omitted...
dcc...1fd
```

2. Obtenga la información de la ciudad de Madrid con el código de ciudad `MAD`.

```
[student@workstation ~]$ curl -s http://localhost:8090/cities/MAD | jq
{
  "name": "Madrid",
  "time": "2022-06-01T12:34:56.123Z",
  "population": 3223000,
  "country": "Spain"
}
```

### NOTA

Pasar la salida del comando `curl` al comando `jq` proporciona un mejor formato de la respuesta JSON en comparación con la visualización de la salida sin formato.

3. Obtenga la información de la ciudad de San Diego con el código de ciudad `SAN`.

```
[student@workstation ~]$ curl -s http://localhost:8090/cities/SAN | jq
{
```



```
"name": "San Diego",  
"time": "2022-06-01T12:34:56.123Z",  
"population": 1415000,  
"country": "United States of America"  
}
```

## Acceso a los contenedores

### Objetivos

- Explorar la ejecución de contenedores.

### Transparencia del contenedor

Los desarrolladores suelen empaquetar aplicaciones como contenedores para aislar el proceso de aplicación, entre otros beneficios. No obstante, el aislamiento de procesos también significa que los desarrolladores pierden visibilidad inmediata del estado del proceso containerizado y su entorno.

Para recuperar esa visibilidad, las herramientas de containerización como Docker proporcionan una forma de iniciar nuevos procesos dentro de los contenedores en el estado de ejecución. Esto es útil, por ejemplo, cuando desea leer un archivo de registro, verificar el valor de una variable de entorno o depurar un proceso.

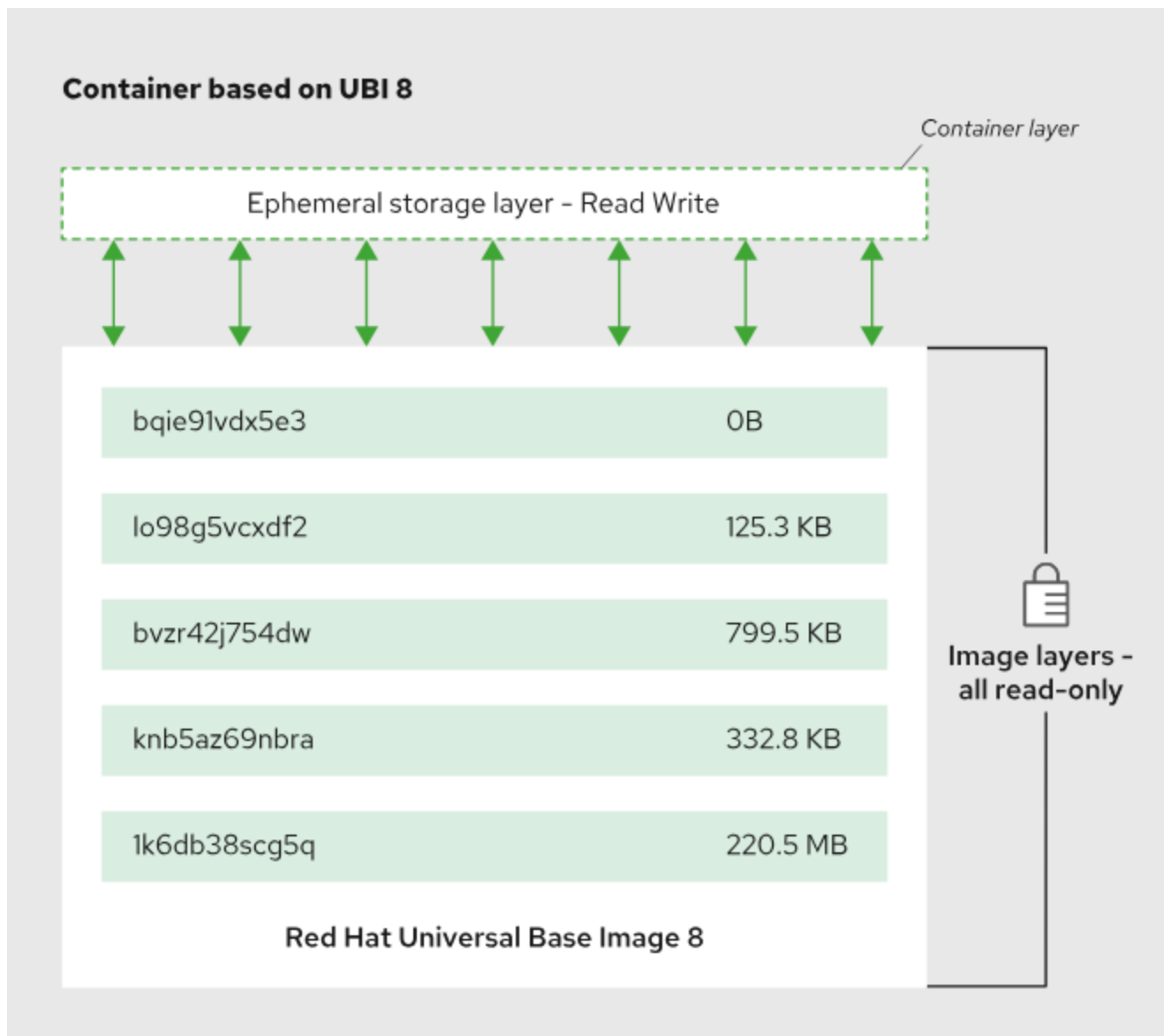
### Introducción a las capas del contenedor

Las imágenes de contenedores se caracterizan por ser *inmutables* y *en capas*. Cada capa de imagen consta de un conjunto de diferencias del sistema de archivos, o *diffs*. Un diff indica un cambio en el sistema de archivos de la capa anterior, como agregar o modificar un archivo.

Cuando inicia un contenedor, el contenedor crea una nueva capa efímera sobre sus capas de imagen de contenedor base denominada *capa del contenedor*. Esta capa es el almacenamiento de solo lectura-escritura disponible para el contenedor de forma predeterminada y se usa para cualquier operación del sistema de archivos del tiempo de ejecución, como crear archivos de trabajo, archivos temporales y archivos de registro.

Los archivos que se crean en la capa del contenedor se consideran volátiles, lo que significa que se eliminan cuando elimina el contenedor. La capa del contenedor es exclusiva del contenedor en ejecución, de modo que si crea otro contenedor a partir de la misma imagen base, el nuevo contenedor crea otra capa del contenedor. Esto garantiza que los datos del tiempo de ejecución de cada contenedor estén aislados de otros contenedores.

El almacenamiento efímero del contenedor no es suficiente para las aplicaciones que necesitan conservar los datos durante los reinicios, como las bases de datos. Puede usar el almacenamiento persistente para admitir dichas aplicaciones. El almacenamiento persistente del contenedor se trata más adelante en este curso.



## Iniciar procesos en contenedores

Use el comando `docker exec` para iniciar un nuevo proceso en un contenedor en ejecución. El comando usa la siguiente sintaxis:

```
docker exec [options] container [command ...]
```

### NOTA

La mayoría de los comandos de Docker aceptan un nombre de contenedor o un ID de contenedor al identificar contenedores. Puede ver los ID de todos los contenedores en ejecución al ejecutar el comando `docker ps`.

En la explicación de sintaxis anterior, las partes del comando entre corchetes, como `[options]`, son opcionales. Por ejemplo, el siguiente comando imprime el archivo `/etc/httpd/conf/httpd.conf` con el comando `cat` en un contenedor en ejecución denominado `httpd`:

```
[user@host ~]$ docker exec httpd cat /etc/httpd/conf/httpd.conf
```

Además, `docker exec` proporciona una serie de opciones, como:

- Use `--env` o `-e` para especificar variables de entorno.
- Use `--interactive` o `-i` para indicarle al contenedor que acepte la entrada.
- Use `--tty` o `-t` para asignar un pseudoterminal.
- Use `--latest` o `-l` para ejecutar el comando en el último contenedor creado.

### NOTA

Los indicadores `--latest` y `-l` no están disponibles cuando se usa el cliente remoto de Docker. Esto incluye cuando se ejecuta a través de Docker Machine en macOS y Windows, excepto cuando se usa WSL2 en este último.

El siguiente comando establece la variable `ENVIRONMENT` y luego ejecuta el comando `env` para imprimir todas las variables del entorno. En este ejemplo, el nombre del contenedor no es necesario porque se usa la opción `-l`

```
[user@host ~]$ docker exec -e ENVIRONMENT=dev -l env
```

Una vez finalizado el proceso `env`, la variable `ENVIRONMENT` no se establece. Para que la variable `ENVIRONMENT` sea persistente, detenga y elimine el contenedor en ejecución y vuelva a ejecutar el comando `docker run` con la opción `-e ENVIRONMENT=dev`.

### Abrir una sesión interactiva en contenedores

Use la combinación de las opciones `--tty` y `--interactive` para abrir una shell interactiva en un contenedor en ejecución.

Si abre un programa de shell como Bash o PowerShell en un contenedor sin proporcionar ninguna opción, el comando `docker exec` abre el programa de shell, no recibe ninguna entrada y sale correctamente:

```
[user@host ~]$ docker exec -l /bin/bash  
[user@host ~]$
```

Si abre un programa de shell con la opción `--interactive`, el `docker exec` ejecuta el programa de shell y recibe su entrada, pero la entrada no es capturada por un terminal:

```
[user@host ~]$ docker exec -il /bin/bash

pwd

ls
```

Los usuarios pueden escribir comandos, pero los comandos no se ejecutan.

### NOTA

El ejemplo anterior combina varias opciones. El comando `docker exec -il` es idéntico a `docker exec -i -l`.

Si abre un programa de shell con la opción `--tty`, el `docker exec` ejecuta el programa de shell y abre un seudoterminal, pero no recibe entrada:

```
[user@host ~]$ docker exec -tl /bin/bash

bash-4.4$
```

El programa de shell se abre en un seudoterminal, pero no puede pasar ninguna entrada al shell.

Por último, para abrir una shell interactiva en un contenedor en ejecución, combine las opciones `--interactive` y `--tty`.

```
[user@host ~]$ docker exec -til /bin/bash

bash-4.4$ pwd
/opt/app-root/src

bash-4.4$ exit

[user@host ~]$
```

Ejecutar el comando `pwd` para imprimir el directorio de trabajo actual:

Ejecute el comando `exit` para salir de la sesión interactiva.

El contenedor sale correctamente.

## Copiar archivos dentro y fuera de contenedores

Algunos contenedores no proporcionan los programas necesarios para depurar un proceso en ejecución. Por ejemplo, para imprimir el contenido de un archivo, puede usar la utilidad `cat`. Sin embargo, en los contenedores listos para producción, una práctica recomendada es empaquetar solo las librerías requeridas por el tiempo de ejecución de la aplicación. Es posible que dicho contenedor no incluya utilidades básicas, como `cat`, o un editor como `vi`.

Puede solucionar el problema de varias maneras, una ellas es copiar el archivo que desea modificar en su máquina `host`, modificarlo y luego reemplazar el archivo original.

Use el comando `docker cp` para copiar archivos desde y hacia un contenedor en ejecución. El comando usa la siguiente sintaxis:

```
docker cp [options] [container:]source_path [container:]destination_path
```

Use el siguiente comando para copiar el archivo `/tmp/logs` de un contenedor con ID `a3bd6c81092e` al directorio actual:

```
[user@host ~]$ docker cp a3bd6c81092e:/tmp/logs .
```

### NOTA

El punto `(.)` en el comando anterior significa el directorio de trabajo actual.

Use el siguiente comando para copiar el archivo `nginx.conf` al directorio `/etc/nginx/` en un contenedor denominado `nginx`:

```
[user@host ~]$ docker cp nginx.conf nginx:/etc/nginx
```

El comando anterior supone que el archivo `nginx.conf` existe en su directorio de trabajo.

Use el siguiente comando para copiar el archivo `nginx.conf` desde el contenedor `nginx-test` al contenedor `nginx-proxy`:

```
[user@host ~]$ docker cp nginx-test:/etc/nginx/nginx.conf nginx-proxy:/etc/nginx
```

## Ejercicio guiado: Acceso a los contenedores

Usar los comandos `docker exec` y `docker cp` para depurar y corregir la configuración del contenedor.

### Resultados

Usted deberá ser capaz de realizar lo siguiente:

- Use el comando `docker exec` para ejecutar comandos dentro de un contenedor.
- Use el comando `docker cp` para copiar archivos desde y hacia un contenedor.

### Procedimiento 2.3. Instrucciones

1. Cree un contenedor nuevo con los siguientes parámetros:

- Nombre del contenedor: `nginx`
- Imagen de contenedor: `quay.io/piracarter1/docker-nginx-helloworld`

Enrute el tráfico desde el puerto 8080 en su máquina al puerto 8080 dentro del contenedor. Use la opción `-d` para ejecutar el contenedor en segundo plano.

```
[student@workstation ~]$ docker run --name nginx -d -p 8080:8080 \
    quay.io/piracarter1/docker-nginx-helloworld
Trying to pull registry.ocp4.example.com:8443/redhattraining/docker-nginx-hellowor
ld:latest...
...output omitted...
3c2b...fa84
```

2. En un explorador web, diríjase a `localhost:8080`. Se le presenta una página de error 404 Not Found.

```
curl localhost:8080
```

```
404 page not found
```

3. Solucione el problema.

- Use el comando `docker cp` para copiar el archivo de registro `/var/log/nginx/error.log` del contenedor `nginx` a su máquina local.

En su terminal, ejecute el siguiente comando:

```
[student@workstation ~]$ docker cp nginx:/var/log/nginx/error.log error.log
```

*Successfully copied 2.05kB to /home/student/error.log*

- Ve a el contenido del archivo de registro.

```
[student@workstation ~]$ gedit error.log  
  
2022/04/26 12:19:17 [error] 2#0: *2 "/usr/share/nginx/html/public/index.html" is not found (2: No such file or directory), client: 10.0.2.100, server: _, request: "GET / HTTP/1.1", host: "localhost:8080"  
  
2022/04/26 12:19:17 [error] 2#0: *2 open() "/usr/share/nginx/html/public/404.html" failed (2: No such file or directory), client: 10.0.2.100, server: _, request: "GET / HTTP/1.1", host: "localhost:8080"
```

El servidor no puede leer el archivo `/usr/share/nginx/html/public/index.html`.

- Verifique el contenido del directorio `/usr/share/nginx/html/public`.

```
[student@workstation ~]$ docker exec nginx ls /usr/share/nginx/html/public  
ls: cannot access '/usr/share/nginx/html/public': No such file or directory
```

El directorio no existe en el contenedor.

- Verifique el contenido del directorio `/usr/share/nginx/html`.

```
[student@workstation ~]$ docker exec nginx ls /usr/share/nginx/html  
404.html  
50x.html  
index.html  
nginx-logo.png  
poweredby.png
```

La página `index.html` existe en el directorio `/usr/share/nginx/html`.

#### 4. Corrija la configuración del servidor.

Copie el archivo `/etc/nginx/nginx.conf` del contenedor a su máquina.

```
[student@workstation ~]$ docker cp nginx:/etc/nginx/nginx.conf .  
  
no output expected
```

- Abra el archivo `nginx.conf` en un editor de texto, como VSCode o Gedit.



En la directiva `server`, cambie el parámetro `root` al valor `/usr/share/nginx/html/`.

```
[student@workstation ~]$ gedit nginx.conf
...file omitted...
    server {
        listen      8080 default_server;
        server_name  _;
        root         /usr/share/nginx/html;
    }
...file omitted...
```

Guarde los cambios en el archivo `nginx.conf`.

- Reemplace el archivo `/etc/nginx/nginx.conf` en el contenedor con el archivo `nginx.conf` modificado.

```
[student@workstation ~]$ docker cp nginx.conf nginx:/etc/nginx
no output expected
```

## 5. Verifique la funcionalidad del contenedor en ejecución.

- Vuelva a cargar la configuración del servidor.

```
[student@workstation ~]$ docker exec nginx nginx -s reload
no output expected
```

- En un explorador web, diríjase a `localhost:8080`. Se le presenta el contenido de la página `index.html`.

## Gestión del ciclo de vida del contenedor

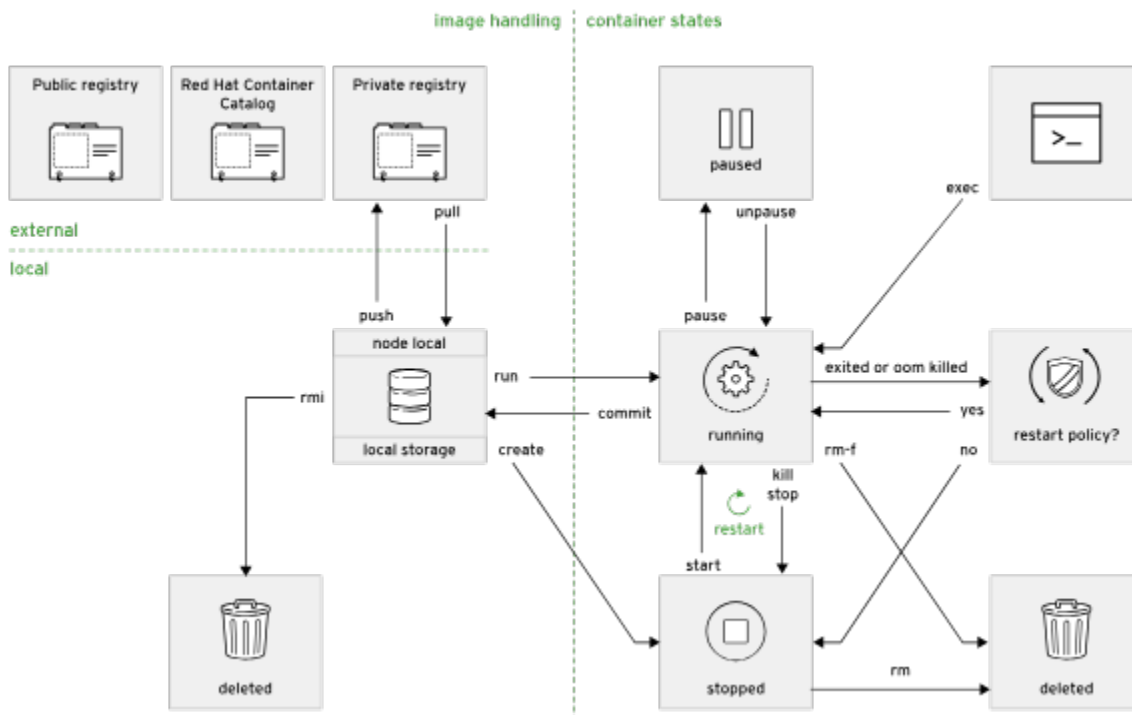
### Objetivos

- Enumerar, detener y eliminar contenedores con Docker.

### Ciclo de vida del contenedor

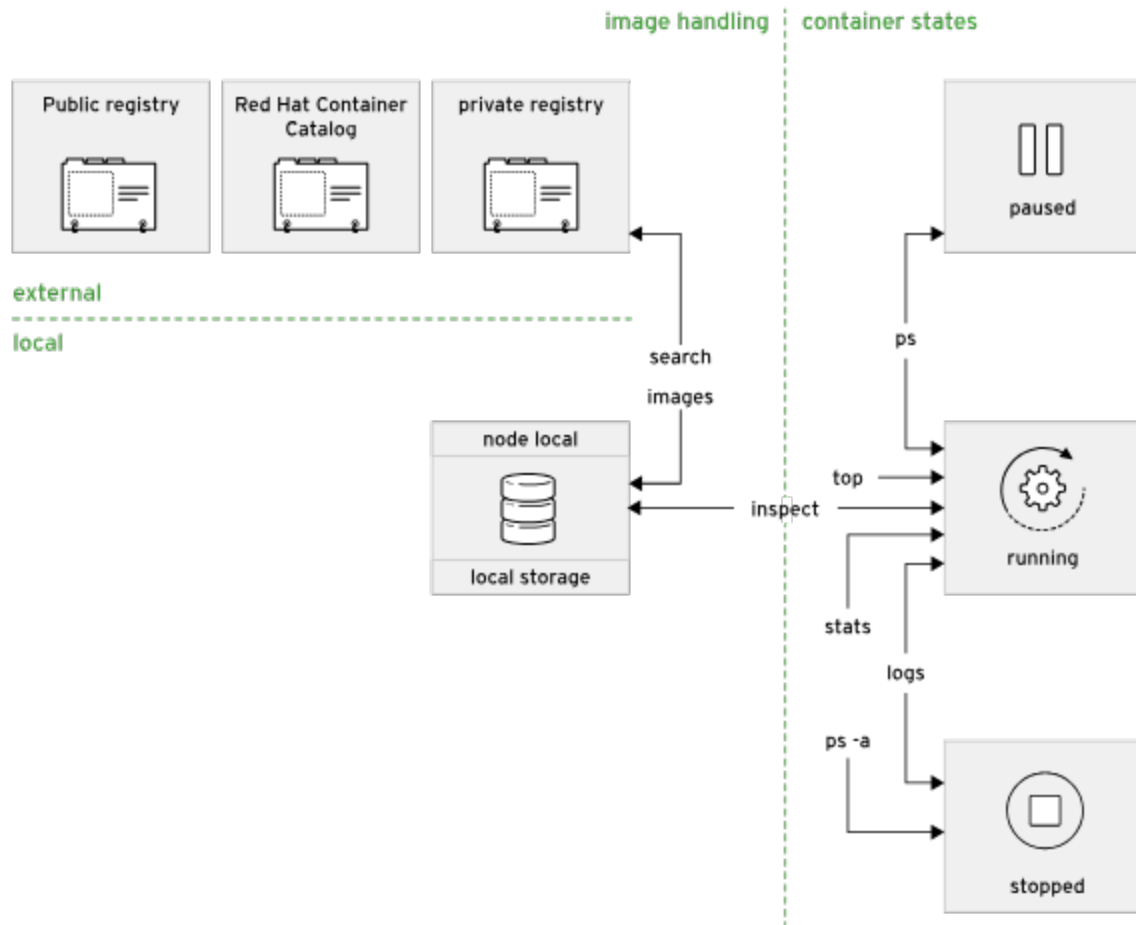
Docker proporciona un conjunto de subcomandos para crear y gestionar contenedores. Puede usar esos subcomandos para gestionar el ciclo de vida de los contenedores y de las imágenes de contenedores.

En las siguientes figuras, se muestra un resumen de los subcomandos Docker usados más comúnmente que cambian el estado del contenedor y la imagen:



Docker también proporciona un conjunto de subcomandos para obtener información acerca de los contenedores en ejecución y detenidos.

Puede usar estos subcomandos para extraer información de contenedores e imágenes con fines de depuración, actualización o informes. En la siguiente figura, se muestra un resumen de los subcomandos usados más comúnmente que consultan información en los contenedores y las imágenes:



Esta clase cubre las operaciones básicas que puede usar para gestionar contenedores. Los comandos explicados en esta clase aceptan un ID de contenedor o el nombre del contenedor.

## Enumerar contenedores

Puede enumerar los contenedores en ejecución mediante el comando `docker ps`.

```
[user@host ~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0ae7be593698	...server	/bin/sh -c python...	...ago	Up...	...8000/tcp	httpd
c42e7dca12d9	...helloworld	/bin/sh -c nginx...	...ago	Up...	...8080/tcp	nginx

Cada fila describe información sobre el contenedor, como la imagen usada para iniciar el contenedor, el comando ejecutado cuando se inició el contenedor y el tiempo de actividad del contenedor.

Puede incluir contenedores detenidos en la salida agregando el indicador `--all` o `-a` al comando `docker ps`.

```
[user@host ~]$ docker ps --all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0ae7be593698	...server	/bin/sh -c python...	...ago	Up...	...8000/tcp	httpd
bd5ada1b6321	...httpd-24	/usr/bin/run-http...	...ago	Exited...	...8080/tcp	upbeat...
c42e7dca12d9	...helloworld	/bin/sh -c nginx	...ago	Up...	...8080/tcp	nginx

## Inspección de contenedores

En el contexto de Docker, inspeccionar un contenedor significa recuperar la información completa del contenedor. El comando `docker inspect` devuelve una matriz JSON con información sobre diferentes aspectos del contenedor, como la configuración de red, el uso de la CPU, las variables de entorno, el estado, el mapeo de puertos o los volúmenes. El siguiente fragmento es un ejemplo de salida del comando.

```
[user@host ~]$ docker inspect 7763097d11ab
```

```
[
  {
    "Id": "7763...cbc0",
    "Created": "2022-05-04T10:00:32.988377257-03:00",
    "Path": "container-entrypoint",
    "Args": [
      "/usr/bin/run-httpd"
    ],
    "State": {
      "OciVersion": "1.0.2-dev",
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 9746,
      ...output omitted...
      "Image": "d2b9...fa0a",
```

```

        "ImageName": "registry.access.redhat.com/ubi8/httpd-24:latest",
        "Rootfs": "",
...output omitted...
        "Env": [
            "PATH=/opt/app-root/src/bin:/opt/app-root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
            "TERM=xterm",
            "container=oci",
            "HTTPD_VERSION=2.4",
...output omitted...
        "CpuCount": 0,
        "CpuPercent": 0,
        "IOMaximumIOps": 0,
        "IOMaximumBandwidth": 0,
        "CgroupConf": null
    }
}
]

```

La salida JSON anterior omite la mayoría de los campos. Debido a que el JSON completo es largo, es posible que le resulte difícil encontrar los campos específicos que desea recuperar. Puede pasar un indicador `--format` al comando `docker inspect` para filtrar la salida del comando. El indicador `--format` espera una expresión de plantilla Go como parámetro, que puede usar para seleccionar campos específicos en el JSON.

Las expresiones de plantilla Go usan anotaciones, que están delimitadas por llaves dobles (`{{` y `}}`), para hacer referencia a elementos, como campos o claves, en una estructura de datos. Los elementos de la estructura de datos están separados por un punto (.) y deben comenzar en mayúsculas.

En el siguiente comando, el campo `status` del objeto `state` se recupera para un contenedor llamado `redhat`.

```

[user@host ~]$ docker inspect \
  --format='{{.State.Status}}' redhat
running

```

En el ejemplo anterior, el comando `docker inspect` usa la plantilla Go proporcionada para navegar por la matriz JSON del contenedor `redhat`. La plantilla devuelve el valor del campo `status` del objeto `State` de la matriz JSON.

Consulte la sección de referencias para obtener más información sobre el lenguaje de plantillas de Go.

### Detención de contenedores sin problemas

Puede detener un contenedor sin problemas mediante el comando `docker stop`. Cuando ejecuta el comando `docker stop`, Docker envía una señal `SIGTERM` al contenedor. Los procesos usan la señal `SIGTERM` para implementar procedimientos de limpieza antes de detenerse.

El siguiente comando detiene un contenedor con un ID de contenedor `1b982aeb75dd`.

```
[user@host ~]$ docker stop 1b982aeb75dd
1b982aeb75dd
```

Puede detener todos los contenedores en ejecución mediante el indicador `--all` o `-a`. En el siguiente ejemplo, el comando detiene tres contenedores.

```
[user@host ~]$ docker stop --all
4aea...0c2a
6b18...54ea
7763...cbc0
```

Si un contenedor no responde a la señal `SIGTERM`, Docker envía una señal `SIGKILL` para detener el contenedor a la fuerza. Por defecto, Docker espera 10 segundos antes de enviar la señal `SIGKILL`. Puede cambiar el comportamiento predeterminado mediante el uso del indicador `--time`.

```
[user@host ~]$ docker stop --time=100
```

En el ejemplo anterior, Docker le da al contenedor un período de gracia de 100 segundos antes de enviar la señal de eliminación.

Detener un contenedor no es lo mismo que eliminarlo. Los contenedores detenidos están presentes en la máquina host y se pueden enumerar mediante el comando `docker ps --all`.

Además, cuando finaliza el proceso de containerización, el contenedor entra en el estado de salida. Un proceso de este tipo puede finalizar por varias razones, como un error, un

estado de OOM (memoria insuficiente) o una finalización satisfactoria. Docker enumera los contenedores en estado de salida con otros contenedores detenidos.

### Detención de contenedores de manera forzosa

Puede enviar la señal SIGKILL al contenedor mediante el comando `docker kill`. En el siguiente ejemplo, un contenedor llamado `httpd` se detiene a la fuerza.

```
[user@host ~]$ docker kill httpd
httpd
```

### Pausa de contenedores

Los comandos `docker stop` y `docker kill` eventualmente envían una señal SIGKILL al contenedor. El comando `docker pause` suspende todos los procesos en el contenedor al enviar la señal SIGSTOP.

```
[user@host ~]$ docker pause 4f2038c05b8c
4f2038c05b8c
```

El comando `docker unpause` reanuda un contenedor en pausa.

```
[user@host ~]$ docker unpause 4f2038c05b8c
4f2038c05b8c
```

### NOTA

El comando `docker pause` requiere grupos de control v2. Debido a que Red Hat Enterprise Linux 8 (RHEL 8) usa grupos de control v1 de manera predeterminada, no puede pausar los contenedores en la instalación predeterminada de RHEL 8.

Consulte la sección de referencias, para obtener más información sobre habilitar los grupos de control v2 en RHEL 8.

### Reinicio de contenedores

Ejecute el comando `docker restart` para reiniciar un contenedor en ejecución. Puede usar el comando para iniciar los contenedores detenidos.

El siguiente comando reinicia un contenedor llamado `nginx`.

```
[user@host ~]$ docker restart nginx
1b98...75dd
```

## Eliminación de contenedores

Use el comando `docker rm` para eliminar un contenedor detenido. El siguiente comando elimina un contenedor detenido con el ID de contenedor `c58cfd4b90df`.

```
[user@host ~]$ docker rm c58cfd4b90df
c58c...3150
```

No puede eliminar los contenedores en ejecución de forma predeterminada. Primero, debe detener el contenedor en ejecución y, luego, eliminarlo. El siguiente comando intenta eliminar un contenedor en ejecución.

```
[user@host ~]$ docker rm c58cfd4b90df
Error: cannot remove container c58c...3150 as it is running - running or paused containers
cannot be removed without force: container state improper
```

Puede agregar el indicador `--force` (o `-f`) para eliminar el contenedor de manera forzosa.

```
[user@host ~]$ docker rm c58cfd4b90df --force
c58c...3150
```

También puede agregar el indicador `--all` (o `-a`) para eliminar todos los contenedores detenidos. Este indicador no elimina los contenedores en ejecución. El siguiente comando elimina dos contenedores.

```
[user@host ~]$ docker rm --all
6b18...54ea
6c0d...a6fb
```

Puede combinar los indicadores `--force` y `--all` para eliminar todos los contenedores, incluidos los contenedores en ejecución.



## Ejercicio guiado: Gestión del ciclo de vida del contenedor

Gestionar el ciclo de vida de un contenedor que ejecuta un servidor HTTP de Apache.

### Resultados

Usted deberá ser capaz de realizar lo siguiente:

- Obtener información detallada sobre un contenedor.
- Detener contenedores.
- Reiniciar un contenedor detenido.
- Eliminar contenedores.

Con el usuario `student` en la máquina `workstation`, use el comando `lab` para preparar el sistema para este ejercicio.

### Procedimiento 2.4. Instrucciones

1. Cree un contenedor que ejecute un servidor HTTP de Apache en segundo plano.

1. Ejecute el comando `docker run` para crear el contenedor. Exponga el puerto `8080` y use la imagen `registry.ocp4.example.com:8443/ubi8/httpd-24`.

```
[student@workstation ~]$ docker run --name httpd -d -p \
8080:8080 quay.io/piracarter1/httpd-24
Trying to pull registry.ocp4.example.com:8443/ubi8/httpd-24:latest...
...output omitted...
```

2. Verifique que el contenedor esté ejecutándose.

1. Use `docker ps` para enumerar todos los contenedores en ejecución.

```
[student@workstation ~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ID	registry...	...run-http...	...ago	Up...	...8080/tcp	httpd

2. Use el comando `docker inspect` para obtener el campo `status`, que indica si el contenedor se está ejecutando.

```
[student@workstation ~]$ docker inspect --format='{{.State.Status}}' httpd
```

```
running
```

3. Verifique que el contenedor se esté ejecutando mediante el campo Running.

```
[student@workstation ~]$ docker inspect --format='{{.State.Running}}' httpd
true
```

4. En un explorador web, navegue hasta localhost:8080 para verificar que el servidor Apache se esté ejecutando.

3. Detenga el contenedor.

1. Regrese al terminal de la línea de comandos. Use el nombre del contenedor para detenerlo.

```
[student@workstation ~]$ docker stop httpd
httpd
```

2. Recupere el status del contenedor.

3. [student@workstation ~]\$ docker inspect --format='{{.State.Status}}' httpd

```
exited
```

4. Verifique que el contenedor no esté ejecutándose.

```
[student@workstation ~]$ docker inspect --format='{{.State.Running}}' httpd
false
```

5. En un explorador web, verifique que el servidor Apache ya no sea accesible en el puerto 8080.

4. Reinicie el contenedor.

1. En su terminal de línea de comandos, use el comando docker restart para reiniciar el contenedor.

```
[student@workstation ~]$ docker restart httpd
CONTAINER_ID
```

2. Verifique que el contenedor esté ejecutándose nuevamente.

```
[student@workstation ~]$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ID	registry...	...run-http...	...ago	Up...	...8080/tcp	httpd

3. En un explorador web, navegue hasta <http://localhost:8080> para verificar que el servidor Apache se esté ejecutando.

## 5. Elimine el contenedor.

1. Intente eliminar el contenedor en ejecución mediante el comando `docker rm`.

```
[student@workstation ~]$ docker rm httpd
```

Error: cannot remove container CONTAINER\_ID as it is running - running or paused containers cannot be removed without force: container state improper

Docker requiere que detenga el contenedor para poder eliminarlo. Sin embargo, puede agregar el indicador `--force` para eliminar el contenedor de manera forzosa.

2. Forzar la eliminación del contenedor.

```
[student@workstation ~]$ docker rm httpd --force
```

httpd

## 6. Verifique que el contenedor se haya eliminado.

1. Intente recuperar el estado del contenedor

```
[student@workstation ~]$ docker inspect --format='{{.State.Running}}' httpd
```

Error: error inspecting object: no such object: "httpd"

Debido a que eliminó el contenedor, ya no puede recuperar su estado.

## 7. Detenga de manera forzosa un contenedor que no responde a la señal SIGTERM.

1. Inicie un nuevo contenedor en segundo plano con la imagen `registry.ocp4.example.com:8443/redhattraining/docker-greeter-ignore-sigterm`. La aplicación ignora las señales SIGTERM.

```
[student@workstation ~]$ docker run --name greeter -d \
```

```
quay.io/piracarter1/docker-greeter-ignore-sigterm
...output omitted...
f919...e69b
```

2. Intente detener el contenedor con un período de gracia de 5 segundos.

```
[student@workstation ~]$ docker stop greeter --time=5
WARN[0005] StopSignal SIGTERM failed to stop container greeter in 5 seconds
, resorting to SIGKILL
greeter
```

Docker envía una señal SIGTERM para detener el contenedor sin problemas, pero la aplicación ignora la señal. Después de 5 segundos, Docker envía una señal SIGKILL al contenedor. El indicador `--time` indica el tiempo que Docker espera antes de enviar una señal SIGKILL para detener forzosamente el contenedor.

3. Reinicie el contenedor.

```
[student@workstation ~]$ docker restart greeter
f919...e69b
```

4. Use el comando `docker kill` para enviar directamente una señal SIGKILL y detener el contenedor por la fuerza.

```
[student@workstation ~]$ docker kill greeter
greeter
```

## Capítulo 3. Container Images

### Registros de imágenes de contenedor

#### Objetivos

- Navegar por los registros de contenedor.

#### Registros de contenedor

Una imagen de contenedor es una versión empaquetada de su aplicación, con todas las dependencias necesarias para que la aplicación se ejecute. Puede usar registros de imágenes para almacenar imágenes de contenedores para luego compartirlas de manera controlada. Algunos ejemplos de registros de imágenes incluyen Quay.io, Red Hat Registry, Docker Hub o Amazon ECR.

Por ejemplo, considere el siguiente comando de Docker.

```
[user@host ~]$ docker pull registry.redhat.io/ubi8/ubi:8.6
Trying to pull registry.redhat.io/ubi8/ubi:8.6...
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
3434...8f6b
```

La imagen `registry.redhat.io/ubi8` se almacena en el Registro de Red Hat, porque el nombre del contenedor usa el host `registry.redhat.io`.

#### Registro de Red Hat

Red Hat distribuye imágenes de contenedores mediante dos registros:

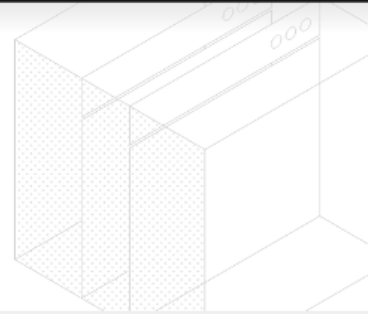
- `registry.access.redhat.com`: no requiere autenticación
- `registry.redhat.io`: requiere autenticación

Sin embargo, Red Hat proporciona la utilidad de búsqueda centralizada para ambos registros: Red Hat Ecosystem Catalog, disponible en <https://catalog.redhat.com/>. Puede usar Red Hat Ecosystem Catalog para buscar imágenes y obtener detalles técnicos sobre ellas. Navegue a <https://catalog.redhat.com/software/containers/explore> para buscar imágenes de contenedores.



[Home](#) > [Software](#) > Container images

## Container images

Discover certified container images from Red Hat and third-party providers that enable and extend your Red Hat environments.

[Search](#)

### Popular categories

 [API Management](#) [Accounting](#) [Application Delivery](#) [Application Server](#) [Automation](#) [Backup & Recovery](#) [Business Intelligence](#) [Business Process Management](#) [Capacity Management](#) [Cloud Management](#) [Collaboration / Groupware / Messaging](#) [Configuration Management](#)

La página de detalles de una imagen de contenedor le brinda información relevante sobre ella, como el Containerfile usado para crear la imagen, los paquetes instalados dentro de la imagen o un escaneo de seguridad. También puede cambiar la versión de la imagen seleccionando una *etiqueta* específica.

[Home](#) > [Software](#) > [Container images](#) > [Browse products](#) > Node.js 16

Builder image

## Node.js 16

rhel8/nodejs-16

Provided by  **Red Hat**

Architecture amd64 Tag 1-37 Repository structure: Single-stream

 latest  

[Overview](#) [Security](#) [Technical Information](#) [Packages](#) [Dockerfile](#) [Get this image](#)

### Description

Node.js 16 available as container is a base platform for building and running various Node.js 16 applications and frameworks. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

#### Usage

For this, we will assume that you are using the `rhel8/nodejs-16` image, available via `nodejs:16` imagestream tag in Openshift. Building a simple `nodejs-sample-app` application in Openshift can be achieved with the following step:

```
$ oc new-app nodejs:16-https://github.com/sclorg/s2i-nodejs-container.git --context-dir=16/test/test-app/
```

The same application can also be built using the standalone [S2I](#) application on systems that have it available:

#### Published

7 days ago

#### Release category

Generally Available 

#### Health index

#### Size

219.5 MB  
(589.0 MB uncompressed)

## Imágenes de contenedores útiles

La siguiente tabla presenta algunas imágenes de contenedores útiles.

Imagen	Proporciona	Descripción
registry.access.redhat.com/ubi9	Imagen de base universal (UBI), versión 9	Una imagen base para crear otras imágenes que se basa en RHEL 9.
registry.access.redhat.com/ubi9/python-39	Python 3.9	Una imagen basada en UBI con tiempo de ejecución Python 3.9.
registry.access.redhat.com/ubi9/nodejs-18	Node.js 18	Una imagen basada en UBI con tiempo de ejecución Node.js18.
registry.access.redhat.com/ubi9/go-toolset	Conjunto de herramientas de Go	Una imagen basada en UBI con tiempo de ejecución Go. La versión de Go depende de la etiqueta de la imagen.

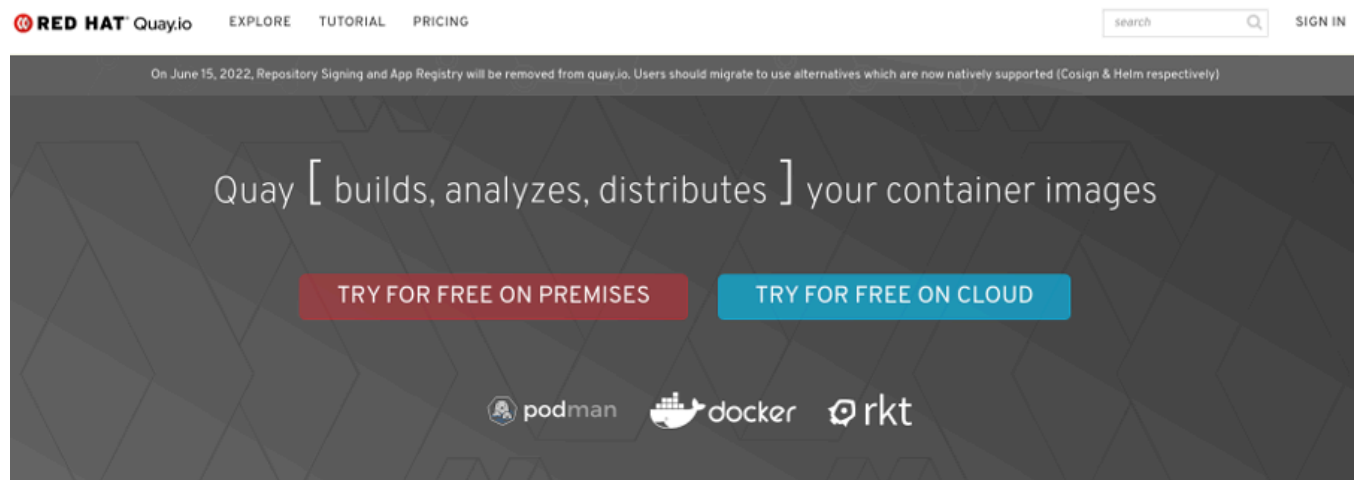
Las UBI de Red Hat son imágenes de contenedor de nivel empresarial *Open Container Initiative (OCI)* que están diseñadas para ser la capa del sistema operativo base para sus aplicaciones contenerizadas. Las UBI incluyen un subconjunto de componentes de Red Hat Enterprise Linux (RHEL). Además, las UBI pueden proporcionar un conjunto de tiempos de ejecución de lenguaje prediseñados. Las UBI se pueden distribuir libremente y se pueden usar en plataformas o registros de contenedores de Red Hat y otros, no de Red Hat.

No se requiere una suscripción a Red Hat para usar o distribuir las imágenes basadas en UBI. Sin embargo, Red Hat solo proporciona soporte completo para contenedores creados en UBI si los contenedores se implementan en una plataforma de Red Hat, como Red Hat OpenShift Container Platform (RHOCP) o RHEL.

## Quay.io

Aunque el registro de Red Hat solo almacena imágenes de Red Hat y proveedores certificados, puede usar el registro Quay.io para almacenar sus imágenes personalizadas. El almacenamiento de imágenes públicas en Quay.io es gratuito y los clientes que pagan reciben beneficios adicionales, como repositorios privados. Los desarrolladores también pueden implementar una instancia de Quay en las instalaciones, que se puede usar para configurar un registro de imágenes en su infraestructura.

Para iniciar sesión en Quay.io, puede usar su cuenta de desarrollador de Red Hat.



## Gestionar registros con Docker

Cuando extrae una imagen de contenedor, proporciona una serie de detalles. Por ejemplo, el nombre de la imagen `registry.access.redhat.com/ubi9/nodejs-18:latest` consta de la siguiente información:

- URL del registro: `registry.access.redhat.com`
- Usuario u organización: `ubi9`
- Repositorio de imágenes: `nodejs-18`



- Etiqueta de la imagen: latest

Los desarrolladores pueden especificar un nombre más corto y no calificado, que omite la URL del registro. Por ejemplo, puede acortar `registry.redhat.io/rhel7/rhel:7.9` a `rhel7/rhel:7.9`.

```
[user@host ~]$ docker pull ubi8/python-39
```

Si no proporciona la URL del registro, Docker usa el archivo `docker` para buscar otros registros de contenedores que podrían contener el nombre de la imagen

### Utilizar la CLI de Docker

Para cambiar el registro Docker predeterminado de `docker.io` a su registro privado usando la CLI de Docker, siga estos pasos:

1. Inicie sesión en su registro privado utilizando el comando de inicio de sesión de Docker. Reemplácelo `my-registry.com` con su propia URL de registro.

```
docker login my-registry.com
```

2. Etiquete la imagen de Docker que desea enviar a su registro privado con la URL del registro. Reemplácelo `my-registry.com` con su propia URL de registro y `my-image` con el nombre de su imagen de Docker.

```
docker tag my-image my-registry.com/my-image
```

3. Inserte la imagen de Docker en su registro privado. Reemplácelo `my-registry.com` con su propia URL de registro y `my-image` con el nombre de su imagen de Docker.

```
docker push my-registry.com/my-image
```

4. Verifique que la imagen de Docker se haya enviado a su registro privado extrayéndola del registro. Reemplácelo `my-registry.com` con su propia URL de registro y `my-image` con el nombre de su imagen de Docker.

```
docker pull my-registry.com/my-image
```

5. Establezca el registro Docker predeterminado en su registro privado creando o editando el archivo de configuración del demonio Docker `/etc/docker/daemon.json`. Reemplácelo `my-registry.com` con su propia URL de registro.

```
{ "registry-mirrors": [],  
  
  "insecure-registries": ["my-registry.com"],  
  
  "debug": true,  
  
  "experimental": false  
}
```

6. Reinicie el demonio Docker para que los cambios surtan efecto.

```
sudo systemctl restart docker
```

## Gestionar credenciales de registros con Docker

Algunos registros requieren que los usuarios se autenticuen, como el registro `registry.redhat.io`.

```
[user@host ~]$ docker pull registry.redhat.io/rhel8/httpd-24  
Trying to pull registry.redhat.io/rhel8/httpd-24:latest...  
Error: initializing source docker://registry.redhat.io/rhel8/httpd-24:latest: unable to retrieve auth token: invalid username/password: unauthorized: Please login to the Red Hat Registry using your Customer Portal credentials. Further instructions can be found here: https://access.redhat.com/RegistryAuthentication
```

Puede elegir una imagen diferente que no requiera autenticación, como la imagen UBI 8 del registro `registry.access.redhat.com`:

```
[user@host ~]$ docker pull registry.access.redhat.com/ubi8:latest  
Trying to pull registry.access.redhat.com/ubi8:latest...  
Getting image source signatures  
Checking if image destination supports signatures  
...output omitted...
```

Alternativamente, autentique sus llamadas ejecutando el comando `docker login`.

```
[user@host ~]$ docker login registry.redhat.io
Username: YOUR_USER
Password: YOUR_PASSWORD
Login Succeeded!
[user@host ~]$ docker pull registry.redhat.io/rhel8/httpd-24
Trying to pull registry.redhat.io/rhel8/httpd-24:latest...
Getting image source signatures
...output omitted...
```

Docker almacena las credenciales en el archivo `${HOME}/.docker/config.json`, donde `${HOME}` se refiere a un directorio específico del usuario actual. Las credenciales están codificadas en el formato base64 :

```
[user@host ~]$ cat ${HOME}/.docker/config.json
{
  "auths": {
    "quay.io": {
      "auth": "UGlyYWNhcnRlcjE6SmVzdXNBcnJpMGxh"
    },
    "registry.redhat.io": {
      "auth": "UGlyYWNhcnRlcjE6amVzdXNoYW5uYW1vcmdhbmRlcmVr"
    }
  }
}

[user@host ~]$ echo -n UGlyYWNhcnRlcjE6SmVzdXNBcnJpMGxh | base64 -d
user:hunter2
```

## NOTA

Por razones de seguridad, el comando `docker login` no muestra su contraseña en la sesión interactiva. Aunque no ve lo que está escribiendo, Docker registra cada pulsación de tecla. Presione Enter cuando haya escrito su contraseña completa en la sesión interactiva para iniciar sesión.

## Ejercicio guiado: Registros de imágenes de contenedor

Interactuar con imágenes de contenedores en múltiples registros de contenedores.

### Resultados

Usted deberá ser capaz de realizar lo siguiente:

- Iniciar sesión en registros de imágenes de contenedor.
- Mover imágenes de contenedor entre registros.
- Probar imágenes de Docker.

### Procedimiento 3.1. Instrucciones

En este ejercicio, su equipo debe asegurarse de que los pipelines internos Dev y Test usen la misma imagen de contenedor Python que la que se encuentra en Red Hat OpenShift Container Platform (RHOCP).

Para hacerlo, copie la imagen de Python del registro QUAY al registro DOCKER:

- Origen: quay.io/piracarter1/ubi8-python-39
  - Destino: docker.io/\$MY\_USER/python:3.9-ubi8
1. Inicie sesión en los registros de imágenes de contenedor.
    1. Inicie sesión en el registro Docker.io con Docker.

```
[student@workstation ~]$ docker login docker.io

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: XXXXXX
Password:

WARNING! Your password will be stored unencrypted in /home/student/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

2. Use docker para copiar la imagen:

```
[student@workstation ~]$ docker pull quay.io/piracarter1/ubi8-python-39
```

```
quay.io/piracarter1/ubi8-python-39:latest
```

Etiquetar la imagen (Cambiar el nombre de usuario que esta en amarillo por el su user de docker.io):

```
docker tag quay.io/piracarter1/ubi8-python-39 docker.io/piracarter1/python:3.9-ubi8
```

Subir la imagen:

```
docker push docker.io/piracarter1/python:3.9-ubi8
```

### 3. Pruebe la imagen como un usuario no autenticado.

1. En un terminal, cierre sesión en todos los registros de imágenes de contenedores.

```
[student@workstation ~]$ docker run --rm \
docker.io/piracarter1/python:3.9-ubi8 python3 --version
Python 3.9.16
```

## Administración de imágenes

### Objetivos

- Extraer y gestionar imágenes de contenedores.

### Gestión de imágenes

La gestión de imágenes incluye diferentes operaciones:

- Etiquetar versiones de imágenes para que se asignen a versiones y actualizaciones de productos.
- Extraer imágenes en su sistema.
- Crear imágenes.
- Enviar imágenes a un repositorio de imágenes.
- Inspección de imágenes para obtener metadatos.
- Eliminación de imágenes para recuperar espacio de almacenamiento.

Para obtener la lista completa de comandos relacionados con la imagen, ejecute `docker image --help` en una ventana de terminal.

#### *Control de versiones de imágenes y etiquetas*

Debido a que las imágenes de contenedor empaquetan software, los desarrolladores a menudo consideran las imágenes como artefactos de implementación. Para mantener las imágenes actualizadas, los desarrolladores suelen asignar las versiones de las imágenes a las versiones del software empaquetado. Mantener las imágenes actualizadas también significa actualizar las librerías del sistema operativo dentro de la imagen para recibir mejoras y correcciones de seguridad.

Una forma de versionar imágenes relacionadas con su producto de software empaquetado es usar el control de versiones semántico. Los números de versión semántica forman una cadena con el formato `MAJOR.MINOR.PATCH` que significa:

- MAJOR: cambios incompatibles con versiones anteriores
- MINOR: cambios compatibles con versiones anteriores
- PATCH: correcciones de errores

Debido a que el control de versiones no tiene una estructura impuesta, los encargados de mantener la imagen deben seguir las buenas prácticas de control de versiones. Esta es una de las razones por las que debe usar registros y repositorios de imágenes confiables.

Las versiones de la imagen se pueden usar en el nombre de la imagen o en la etiqueta de la imagen. Una etiqueta de imagen es una cadena que se especifica después del nombre de la imagen. Además, la misma imagen puede tener varias etiquetas.

```
[<image repository>/<namespace>/]<image name>[:<tag>]
```

El uso de una etiqueta en Docker es opcional. Cuando no especifica una etiqueta en un comando de Docker, Docker usa la etiqueta `latest` de forma predeterminada.

```
[user@host ~]$ docker pull quay.io/argoproj/argocd
Trying to pull quay.io/argoproj/argocd:latest...
...output omitted...
```

El uso de la etiqueta `latest` se considera una mala práctica. Debido a que la etiqueta `latest` también representa la versión más reciente de la imagen, puede incluir cambios incompatibles con versiones anteriores y hacer que los contenedores que usan la imagen se rompan.

Para crear etiquetas adicionales para imágenes locales, use el comando `docker image tag`.

```
[user@host ~]$ docker image tag LOCAL_IMAGE:TAG LOCAL_IMAGE:NEW_TAG
```

### Extracción de imágenes

Para buscar imágenes en diferentes registros de imágenes, use un explorador web para navegar a la URL del registro y use la interfaz de usuario web.

Alternativamente, use el comando `docker search` para buscar imágenes:

```
[user@host ~]$ docker search nginx

nginx                                     Official build of Nginx.                                189
48      [OK]

unit                                     Official build of NGINX Unit: Universal Web ...         10
[OK]

nginxinc/nginx-unprivileged              Unprivileged NGINX Dockerfiles                          114
nginx/nginx-ingress                     NGINX and NGINX Plus Ingress Controllers fo...         76
nginx/nginx-prometheus-exporter          NGINX Prometheus Exporter for NGINX and NGIN...         33
nginx/unit                               NGINX Unit is a dynamic web and application ...         64
...output omitted...
```

Para recuperar una imagen, ejecute `docker image pull IMAGE_NAME`, que descarga la imagen de un registro. Alternativamente, `docker pull IMAGE_NAME` proporciona la misma funcionalidad.

```
[user@host ~]$ docker pull registry.redhat.io/rhel8/mariadb-103:1
```

Cuando extrae una imagen como un usuario no root, Docker almacena imágenes de contenedor en el directorio `~/.local/share/containers`.

Para encontrar las imágenes que su usuario tiene disponibles localmente, use el comando `docker image ls` o `docker images`.

```
[user@host ~]$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
registry.redhat.io/rhel9/python-39	1-52	d336a3191d35	3 weeks ago	995 MB
registry.access.redhat.com/ubi8/python-39	latest	6b7a42c9d513	4 weeks ago	894 MB

...output omitted...

Si el usuario root extrae una imagen, se almacena en el directorio `/var/lib/containers`. Esta imagen solo se muestra cuando `docker image ls` se ejecuta como root.

### Creación de imágenes

También puede crear una imagen a partir de un Containerfile, que describe los pasos que se usan para crear una imagen. Ejecute `docker build --file CONTAINERFILE --tag IMAGE_REFERENCE` para compilar la imagen de contenedor.

Por ejemplo, para compilar una imagen que luego pueda enviar a Red Hat Quay.io, ejecute el siguiente comando:

```
[user@host ~]$ docker build --file Dockerfile \  
--tag quay.io/YOUR_QUAY_USER/IMAGE_NAME:TAG
```

### Envío de imágenes

Después de crear una imagen, envíela a un registro remoto para compartirla. Para enviar una imagen, debe haber iniciado sesión en el registro. Ejecute `docker login REGISTRY` para iniciar sesión en el registro especificado. Luego, puede usar el comando `docker push IMAGE` para enviar una imagen local al registro remoto.

Por ejemplo, para enviar una imagen a Quay.io, ejecute el siguiente comando:

```
[user@host ~]$ docker push quay.io/YOUR_QUAY_USER/IMAGE_NAME:TAG
```

Getting image source signatures



```
Copying blob fb3154998920 done
...output omitted...
Writing manifest to image destination
Storing signatures
```

### *Inspección de imágenes*

El comando `docker image inspect` proporciona información útil sobre una imagen disponible localmente en su sistema.

El siguiente ejemplo de uso muestra información sobre una imagen `mariadb`.

```
[user@host ~]$ docker image inspect registry.redhat.io/rhel8/mariadb-103:1
[
  {
    "Id": "6683...98ea",
    ...output omitted...
    "Config": {

      "User": "27",

      "ExposedPorts": {
        "3306/tcp": {}
      },

      "Env": [
        "PATH=/opt/app-root/src/bin:/opt/app-root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        ...output omitted...
      ],

      "Entrypoint": [
        "container-entrypoint"
      ],

      "Cmd": [
        "run-mysqld"
      ],
```

```
"WorkingDir": "/opt/app-root/src",

"Labels": {
  ...output omitted...
  "release": "177.1654147959",
  "summary": "MariaDB 10.3 SQL database server",
  "url": "https://access.redhat.com/containers/#/registry.access.redhat.com/rhel8/mariadb-103/images/1-177.1654147959",
  ...output omitted...

"Architecture": "amd64",
"Os": "linux",
"Size": 573593952,
...output omitted...
```

El usuario predeterminado de la imagen.

El puerto que expone la aplicación.

Las variables del entorno usadas por la imagen.

El punto de entrada, un comando que se ejecuta cuando se inicia el contenedor.

El comando que ejecuta el script container-entrypoint.

El directorio de trabajo para los comandos en la imagen.

Etiquetas que proporcionan metadatos adicionales.

La arquitectura donde se puede usar esta imagen.

La salida de `docker inspect` es detallada, lo que dificulta la búsqueda de información. Para seleccionar una parte específica de la salida, use la característica de plantillas Go en Docker al proporcionar la opción `--format`. Use las teclas de salida `docker inspect` precedidas por puntos y rodeadas por llaves dobles.

```
--format="{{.Key.Subkey}}"
```

Por ejemplo, el siguiente comando extrae la instrucción CMD de la imagen `rhel8/mariadb-103`:

```
[user@host ~]$ docker image \
inspect registry.redhat.io/rhel8/mariadb-103:1 \
--format="{{.Config.Cmd}}"
[run-mysqld]
```

Para inspeccionar una imagen remota, puede usar la herramienta Skopeo.

### *Eliminación de imágenes*

Puede eliminar las imágenes locales que ya no usa ningún contenedor. Ejecute el comando `docker image rm` o `docker rmi` para eliminar una imagen de contenedor.

```
[user@host ~]$ docker image rm REGISTRY/NAMESPACE/IMAGE_NAME:TAG
```

Si un contenedor está usando la imagen, Docker no puede eliminarla. Primero debe eliminar cualquier contenedor que use la imagen ejecutando `docker stop container-name`. Alternativamente, obligue a Docker a eliminar la imagen proporcionando la opción `-f`. Esto detiene y elimina automáticamente cualquier contenedor que use la imagen y luego elimina la imagen.

```
[user@host ~]$ docker image rm -f REGISTRY/NAMESPACE/IMAGE_NAME:TAG
```

Con la opción `--all`, puede eliminar todas las imágenes en el almacenamiento local.

```
[user@host ~]$ docker rmi --all
```

```
[user@host ~]$ docker image rm --all
```

Las imágenes sin etiquetas y a las que no hacen referencia otras imágenes se consideran imágenes pendientes. Use el comando `docker image prune` para eliminar las imágenes pendientes de su almacenamiento local. Al ejecutar el comando `docker image prune`, Docker muestra un mensaje interactivo para confirmar la eliminación de la imagen.

```
[user@host ~]$ docker image prune
WARNING! This command removes all dangling images.
Are you sure you want to continue? [y/N]
```

Para eliminar las imágenes pendientes y no usadas, proporcione la opción `--all` o `-a`.

```
[user@host ~]$ docker image prune -a
```

```
WARNING! This command removes all images without at least one container associated with the m.
```

```
Are you sure you want to continue? [y/N]
```

Puede incluir la opción `-f` para forzar la eliminación y evitar el mensaje interactivo.

```
[user@host ~]$ docker image prune -af
```

## Ejercicio guiado: Administración de imágenes

Aprenda a gestionar imágenes de contenedores.

### Resultados

Debería poder administrar imágenes mediante la realización de operaciones comunes como:

- Creación de imágenes
- Enumeración de imágenes.
- Inspección de imágenes
- Etiquetado de imágenes
- Eliminación de imágenes
- Búsqueda de imágenes
- Extracción de imágenes

### Procedimiento 3.2. Instrucciones

1. Verifique que la imagen `simple-server` no esté presente en su máquina.

```
[student@workstation ~]$ docker image ls --format "{{.Repository}}"  
no output expected
```

2. Compile y ejecute la imagen `simple-server` utilizando el archivo `~/cdiputadosdocker/images-managing/Containerfile`.

1. Cambie al directorio de ejercicios y examine su contenido.

```
[student@workstation ~] cd ~/cdiputadosdocker/images-managing  
no output expected
```

renombre el archivo `Containerfile` como `Dockerfile`.

2. Use el `Containerfile` del ejercicio y el comando `docker build` para crear la imagen. Use la opción `-t` para localizar la imagen `simple-server`.

```
[student@workstation images-managing]$ docker build -f Dockerfile -t \simple-server .  
=> [1/2] FROM quay.io/piracarter1/ubi7-python-38:1-104  
0.0s  
=> CACHED [2/2] RUN echo "Hello from the container" > hello.html  
0.0s
```

```
=> exporting to image
0.0s

=> => exporting layers
0.0s

=> => writing image sha256:a6c5c584985b1f3d961be56ea9251618068974610130ff7
4c102e03fb14fd907
0.0s

=> => naming to docker.io/library/simple-server
```

3. Use el comando `docker image ls` para verificar que la imagen `simple-server` esté presente.

```
[student@workstation images-managing]$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/simple-server	latest	87dc46f07c8c	5 minutes ago	892 MB

Debido a que no proporcionó un nombre de servidor y una etiqueta para el nombre de la imagen `simple-server`, la imagen usa el servidor de registro `localhost` y la etiqueta `latest` de forma predeterminada.

4. Cree un contenedor a partir de la imagen `simple-server` y llámelo `http-server`. Agregue la opción `-d` para ejecutarla en segundo plano y agregue la opción `-p` para asignar el puerto 8080 en el host local al puerto 8000 en el contenedor.

```
[student@workstation images-managing]$ docker run -d \
-p 8080:8000 \
--name http-server \
simple-server
2b81..0207
```

5. Use una herramienta como `curl` o un explorador web para enviar una solicitud a `http://localhost:8080/hello.html`.

```
[student@workstation images-managing]$ curl http://localhost:8080/hello.html
Hello from the container
```

3. Inspeccione la imagen `simple-server` para ver qué comando se ejecuta de forma predeterminada.

Verifique que `python -m http.server` sea el comando predeterminado para esta imagen. Este comando de Python proviene de la instrucción CMD en el ejercicio Containerfile.

```
[student@workstation images-managing]$ docker image inspect simple-server \
  --format="{{.Config.Cmd}}"
[/bin/sh -c python -m http.server]
```

4. Etiquete la imagen `simple-server` con la etiqueta `0.1` y genere una lista de las imágenes para verificar el resultado.

```
[student@workstation images-managing]$ docker image tag simple-server \
simple-server:0.1
no output expected
```

Cuando enumera las imágenes en su máquina, puede ver que `simple-server` se muestra para la etiqueta `latest`, y también para la etiqueta `0.1`. Verifique que `IMAGE ID` tenga el mismo valor porque ambas etiquetas apuntan a la misma imagen.

```
[student@workstation images-managing]$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
localhost/simple-server	latest	87dc46f07c8c	10 minutes ago	892 MB
localhost/simple-server	0.1	87dc46f07c8c	10 minutes ago	892 MB

5. Elimine la imagen `simple-server` completamente de su sistema.

1. Use el comando `docker image rm` para eliminar `simple-server`.

```
[student@workstation images-managing]$ docker image rm simple-server
Untagged: localhost/simple-server:latest
```

El comando anterior solo eliminó la etiqueta `latest` porque no especificó la etiqueta en el comando y tiene dos etiquetas que apuntan a la misma imagen.

2. Intente eliminar la etiqueta `simple-server:0.1` para eliminar la imagen `simple-server` por completo.

```
[student@workstation images-managing]$ docker image rm simple-server:0.1
```

```
Error: Image used by 1bd7...250d: image is in use by a container
```

El mensaje de error indica que no puede eliminar una imagen que está siendo usada actualmente por un contenedor. Antes de eliminar la imagen debe detenerse y retirar el contenedor.

3. Fuerce la eliminación del contenedor y de la imagen del contenedor.

```
[student@workstation images-managing]$ docker image rm -f simple-server:0.1
WARN[0010] StopSignal SIGTERM failed to stop container http-server in 10 se
conds, resorting to SIGKILL
Untagged: localhost/simple-server:0.1
Deleted: 87dc...fc02
Deleted: 2318...7bbb
```



## Capítulo 4. Imágenes de contenedores personalizadas

### Crear imágenes con Containerfiles

#### Objetivos

- Crear un Containerfile con comandos básicos.

#### Creación de imágenes con Containerfiles

Un *Containerfile* enumera un conjunto de instrucciones que usa el tiempo de ejecución de un contenedor para crear una imagen de contenedor. Puede usar la imagen para crear cualquier cantidad de contenedores.

Cada instrucción provoca un cambio que se captura en una capa de imagen resultante. Estas capas se apilan juntas para formar la imagen de contenedor resultante.

#### NOTA

Es posible que haya visto o usado Dockerfiles en lugar de Containerfiles. Estos archivos son en gran parte iguales y se diferencian principalmente en el contexto histórico.

Este curso usa Containerfiles porque no están asociados con ningún motor de tiempo de ejecución de contenedor específico. Docker soporta tanto Dockerfiles como Containerfiles.

#### Elección de una imagen base

Cuando crea una imagen, Docker ejecuta las instrucciones en el Containerfile y aplica los cambios en la parte superior de una *imagen base* de contenedor. Una imagen base es la imagen a partir de la cual se compila su Containerfile y su imagen resultante.

La imagen base que elija determina la distribución de Linux y sus componentes, como:

- Gestión de paquetes
- Sistema de inicio
- Diseño del sistema de archivos
- Dependencias y tiempos de ejecución preinstalados

La imagen base también puede influir en otros factores, como el tamaño de la imagen, el soporte del proveedor y la compatibilidad del procesador.

Red Hat proporciona imágenes de contenedor diseñadas como un punto de partida común para los contenedores conocidos como *imágenes base universales (UBI)*. Estas UBI vienen en cuatro variantes: `standard`, `init`, `minimal` y `micro`. Además, Red Hat también

proporciona imágenes basadas en UBI que incluyen tiempos de ejecución populares, como Python y Node.js.

Todas estas UBI usan Red Hat Enterprise Linux (RHEL) en su núcleo y están disponibles en Red Hat Container Catalog. Las principales diferencias son las siguientes:

### **Estándar**

Esta es la UBI principal, que incluye DNF, systemd y utilidades como gzip y tar.

### **Init**

Simplifica la ejecución de varias aplicaciones dentro de un solo contenedor al gestionarlas con systemd.

### **Mínima**

Esta imagen es más pequeña que la imagen `init` y aún ofrece características deseables. Esta imagen usa el administrador de paquetes mínimo `microdnf` en lugar de la versión de tamaño completo de DNF.

### **Micro**

Esta es la UBI más pequeña disponible porque solo incluye la cantidad mínima de paquetes. Por ejemplo, esta imagen no incluye un administrador de paquetes.

## **Instrucciones de Containerfile**

Containerfiles usan un pequeño lenguaje específico de dominio (DSL) que consta de instrucciones básicas para crear imágenes de contenedor. Las siguientes son las instrucciones más comunes.

#### **FROM**

Establece la imagen base para la imagen de contenedor resultante. Toma el nombre de la imagen base como argumento.

#### **WORKDIR**

Establece el directorio de trabajo actual dentro del contenedor. Las instrucciones que siguen a la instrucción `WORKDIR` se ejecutan dentro de este directorio.

#### **COPY y ADD**

Copia archivos del host de compilación en el sistema de archivos de la imagen de contenedor resultante. Las rutas relativas usan el directorio de trabajo actual del host, conocido como contexto de compilación. Ambas instrucciones usan el directorio de trabajo dentro del contenedor según lo definido por la instrucción `WORKDIR`.

La instrucción `ADD` agrega la siguiente funcionalidad:

- Copia de archivos desde URL.

- Desempaquetado de archivos tar en la imagen de destino.

Debido a que la instrucción ADD agrega funcionalidad que puede no ser obvia, los desarrolladores tienden a preferir la instrucción COPY para copiar archivos locales en la imagen del contenedor.

#### RUN

Ejecuta un comando en el contenedor y confirma el estado resultante del contenedor en una nueva capa dentro de la imagen.

#### ENTRYPOINT

Establece el ejecutable para que se ejecute cuando se inicia el contenedor.

#### CMD

Ejecuta un comando cuando se inicia el contenedor. Este comando se pasa al ejecutable definido por ENTRYPOINT. Las imágenes base definen un ENTRYPOINT predeterminado, que suele ser un ejecutable de shell, como Bash.

### NOTA

Ni ENTRYPOINT ni CMD se ejecutan al crear una imagen de contenedor. Docker los ejecuta cuando inicia un contenedor desde la imagen.

#### USER

Cambia el usuario activo dentro del contenedor. Las instrucciones posteriores a la instrucción USER se ejecutan como este usuario, incluida la instrucción CMD. La práctica adecuada es definir un usuario diferente a root por motivos de seguridad.

#### LABEL

Agrega un par clave-valor a los metadatos de la imagen para la organización y la selección de imágenes.

#### EXPOSE

Agrega un puerto a los metadatos de la imagen que indica que una aplicación dentro del contenedor se vincula a este puerto. Esta instrucción no vincula el puerto en el host y es para fines de documentación.

#### ENV

Define las variables de entorno que están disponibles en el contenedor. Puede declarar varias instrucciones ENV dentro del Containerfile. Puede usar el comando env dentro del contenedor para ver cada una de las variables de entorno.

#### ARG

Define variables de tiempo de compilación, generalmente para hacer una compilación de contenedor personalizable. Los desarrolladores suelen configurar las instrucciones ENV mediante la instrucción ARG. Esto es útil para conservar las variables de tiempo de compilación para el tiempo de ejecución.

## VOLUME

Define dónde almacenar los datos fuera del contenedor. El valor configura la ruta donde Docker monta el volumen persistente dentro del contenedor. Puede definir más de una ruta para crear varios volúmenes.

Cada instrucción de Containerfile se ejecuta en un contenedor independiente usando una imagen intermedia compilada de todos los comandos anteriores. Esto significa que cada instrucción es independiente de otras instrucciones en el Containerfile. A continuación, se incluye un ejemplo de Containerfile para compilar un contenedor de servidores web simple de Apache:

```
# This is a comment line
FROM      registry.redhat.io/ubi8/ubi:8.6
LABEL     description="This is a custom httpd container image"
RUN       yum install -y httpd
EXPOSE    80
ENV       LogLevel "info"
ADD       http://someserver.com/filename.pdf /var/www/html
COPY      ./src/    /var/www/html/
USER      apache
ENTRYPOINT ["/usr/sbin/httpd"]
CMD       [ "-D", "FOREGROUND" ]
```

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

La instrucción FROM declara que la nueva imagen de contenedor se extiende a la imagen base del contenedor registry.redhat.io/ubi8/ubi:8.6.

La instrucción LABEL agrega metadatos a la imagen.

RUN ejecuta comandos en una nueva capa sobre la imagen actual. La shell que se usa para ejecutar comandos es /bin/sh.

EXPOSE configura metadatos que indican que el contenedor escucha en el puerto de red especificado en tiempo de ejecución.

ENV es responsable de definir las variables de entorno que están disponibles en el contenedor.

La instrucción ADD copia archivos o directorios desde una fuente local o remota y los agrega al sistema de archivos del contenedor.

COPY copia archivos desde la ruta relativa al directorio de trabajo y los agrega al sistema de archivos del contenedor.

USER especifica el nombre de usuario o el UID que se usará cuando se ejecute la imagen de contenedor para las instrucciones RUN, CMD y ENTRYPOINT.

ENTRYPOINT especifica el comando predeterminado para ejecutarse cuando los usuarios crean una instancia de la imagen como un contenedor.

CMD proporciona los argumentos predeterminados para la instrucción ENTRYPOINT.

## Etiquetas de imágenes de contenedores

Al crear una imagen de contenedor, puede especificar un nombre de imagen para identificarla posteriormente. El nombre de la imagen es una cadena compuesta por letras, números y algunos caracteres especiales.

Una *etiqueta de imagen* viene después del nombre de la imagen y está delimitada por dos puntos (:). Cuando omite una etiqueta de imagen, Docker usa la etiqueta predeterminada latest.

### **NOTA**

Es una buena práctica especificar etiquetas además de la etiqueta latest. Debido a que latest es la etiqueta predeterminada que se aplica a las imágenes nuevas, las referencias que usan la etiqueta latest pueden cambiar involuntariamente.

El nombre completo de la imagen incluye un nombre y una etiqueta de imagen opcional.

Por ejemplo, en el nombre completo de la imagen my-app:1.0, el nombre es my-app y la etiqueta es 1.0.

# Ejercicio guiado: Crear imágenes con Containerfiles

Crear un Dockerfile básico para una aplicación Node.js de ejemplo. Durante el ejercicio, mejorará gradualmente el Dockerfile y la imagen del contenedor resultante.

## Resultados

Usted deberá ser capaz de realizar lo siguiente:

- Cree un Dockerfile con una imagen base adecuada.
- Use instrucciones comunes de Dockerfile, como FROM, COPY, RUN y CMD.
- Optimice las instrucciones para reducir el tamaño de la imagen y la cantidad de capas.

## Procedimiento 4.1. Instrucciones

1. Cree un nuevo Containerfile para la aplicación Node.js hello-server.

### IMPORTANTE

El Containerfile que crea en este paso inicial no sigue la guía de Red Hat para crear imágenes de contenedor. Mejorará este Containerfile a lo largo del ejercicio.

1. Cambie al directorio ~/D0188/labs/custom-containerfiles/hello-server.

```
[student@workstation ~]$ cd ~/cdiputadosdocker/custom-containerfiles/hello-server/  
  
no output expected
```

2. Cree un nuevo archivo denominado Dockerfile con el siguiente contenido:

```
FROM quay.io/piracarter1/ubi8:8.3  
COPY . /tmp/hello-server  
RUN dnf module enable -y nodejs:16  
RUN dnf install -y nodejs  
RUN cd /tmp/hello-server && npm install  
CMD cd /tmp/hello-server && npm start
```

3. Cree una imagen de contenedor con la etiqueta hello-server:bad usando el Dockerfile que creó.

```
[student@workstation hello-server]$ docker build -t hello-server:bad .  
...output omitted...  
Successfully tagged localhost/hello-server:bad  
...output omitted...
```

4. Cree un contenedor denominado hello-bad que use la imagen hello-server:bad y vincule el puerto 3000 del contenedor al puerto 3000 del host.

```
[student@workstation hello-server]$ docker run -d --rm --name hello-bad \  
-p 3000:3000 hello-server:bad  
ecb8...b5af
```

El comando imprime el ID del contenedor en ejecución, que será diferente en su máquina.

5. Realice una solicitud al contenedor en ejecución mediante el comando curl.

```
[student@workstation hello-server]$ curl http://localhost:3000/greet ;echo  
{"hello":"world"}
```

6. Detenga el contenedor hello-bad.

```
[student@workstation hello-server]$ docker stop hello-bad  
hello-bad
```

2. Actualice el Containerfile para reducir la cantidad de capas de imágenes que produce y simplifique el uso del directorio.

1. Inspeccione el número de capas de imagen en la imagen hello-server:bad.

```
[student@workstation hello-server]$ docker history hello-server:bad
```

IMAGE SIZE	CREATED COMMENT	CREATED BY
17df86aab957 ... 0B	3 minutes ago buildkit.dockerfile.v0	CMD ["/bin/sh" "-c" "cd /tmp/hello-server && buildkit.dockerfile.v0
<missing> ... 107MB	3 minutes ago buildkit.dockerfile.v0	RUN /bin/sh -c cd /tmp/hello-server && npm i buildkit.dockerfile.v0

```

<missing>      3 minutes ago  RUN /bin/sh -c dnf install -y nodejs # build
...    182MB      buildkit.dockerfile.v0

<missing>      3 minutes ago  RUN /bin/sh -c dnf module enable -y nodejs:1
...    10.4MB     buildkit.dockerfile.v0

<missing>      4 minutes ago  COPY . /tmp/hello-server # buildkit
920B      buildkit.dockerfile.v0

<missing>      2 years ago
4.76kB

<missing>      2 years ago
205MB      Imported from -

```

Los ID de capa y los tamaños en su entorno difieren del ejemplo anterior.

Observe que el Dockerfile crea varias capas sobre la imagen base UBI de Red Hat.

2. Abra el Containerfile que creó y refactorice el archivo para usar una instrucción `WORKDIR`. Actualice las rutas en el resto del archivo para que coincidan.

```

FROM quay.io/piracarter1/ubi8:8.3
WORKDIR /tmp/hello-server
COPY . .
RUN dnf module enable -y nodejs:16
RUN dnf install -y nodejs
RUN npm install
CMD npm start

```

Al usar `WORKDIR`, las otras instrucciones ya no necesitan cambiar de directorio. Esto facilita la lectura del Containerfile y reduce los posibles errores en las rutas de los archivos.

La instrucción `COPY . .` copia el contenido del directorio actual (`~/D0188/labs/custom-containerfiles/hello-server`) en el directorio actual en el contenedor (`/tmp/hello-server`).

3. Actualice el Containerfile fusionando las instrucciones `RUN` en una sola instrucción. El Containerfile final contiene las siguientes instrucciones:

```

FROM quay.io/piracarter1/ubi8:8.3
WORKDIR /tmp/hello-server

```



```

COPY . .
RUN dnf module enable -y nodejs:16 && \
    dnf install -y nodejs && \
    npm install
CMD npm start

```

4. Cree la imagen de contenedor con el Containerfile actualizado. Utilice la etiqueta de imagen hello-server:better.

```

[student@workstation hello-server]$ docker build -t hello-server:better .
...output omitted...
Successfully tagged localhost/hello-server:better
...output omitted...

```

5. Inspeccione el número de capas de imagen en la imagen hello-server:better.

```

[student@workstation hello-server]$ docker history hello-server:better

```

IMAGE SIZE	CREATED COMMENT	CREATED BY
bbd38cb14b8d 0B	29 seconds ago buildkit.dockerfile.v0	CMD ["/bin/sh" "-c" "npm start"]
<missing> ejs:1... 297MB	29 seconds ago buildkit.dockerfile.v0	RUN /bin/sh -c dnf module enable -y nodejs:16 && dnf install -y nodejs && npm install
<missing> 887B	About a minute ago buildkit.dockerfile.v0	COPY . . # buildkit
<missing> 0B	About a minute ago buildkit.dockerfile.v0	WORKDIR /tmp/hello-server
<missing> 4.76kB	2 years ago	
<missing> 205MB	2 years ago Imported from -	

Observe que el Dockerfile crea dos capas sobre la imagen ubi8.3. Reducir el número de instrucciones RUN también reduce el número de capas de imagen resultantes.

6. Ejecute y pruebe el contenedor para verificar que funciona.

```
[student@workstation hello-server]$ docker run -d --rm --name hello-better \
-p 3000:3000 hello-server:better
d6c3...9f21
[student@workstation hello-server]$ curl http://localhost:3000/greet ;echo
{"hello":"world"}
[student@workstation hello-server]$ docker stop hello-better
hello-better
```

3. Actualice el Containerfile para usar una imagen base mejor y use variables de entorno para ajustar la configuración del servidor.

1. Abra el Containerfile y cambie la instrucción FROM para usar la imagen de tiempo de ejecución de Node.js proporcionada por Red Hat. Esta imagen se basa en la imagen base universal (UBI) de Red Hat e incluye el tiempo de ejecución de Node.js.

```
FROM quay.io/piracarter1/ubi8-nodejs-16:1
```

### NOTA

El uso de la imagen `ubi8/nodejs-16:1` reduce el tiempo de creación y la complejidad de la imagen del contenedor. Para optimizar el tamaño general de la imagen, puede usar la imagen `ubi8/nodejs-16-minimal` o `ubi8/ubi-minimal` en su lugar.

La elección de una imagen base es un tema complejo que implica inquietudes contrapuestas. Las organizaciones eligen una imagen base que está optimizada para su principal preocupación, como el tamaño de la imagen, la complejidad o la velocidad para eliminar las vulnerabilidades del tiempo de ejecución.

2. Debido a que la imagen base incluye el tiempo de ejecución de Node.js, elimine la parte `dnf module enable -y nodejs:16` y `dnf install -y nodejs` de la instrucción RUN.

```
COPY . .
RUN npm install
CMD npm start
```

Esta actualización no cambia el número de capas de imagen porque el número de instrucciones RUN no cambió.

3. Agregue instrucciones ENV para definir variables de entorno dentro del contenedor que usa la aplicación.

```
FROM registry.ocp4.example.com:8443/ubi8/nodejs-16:1
```

```
ENV SERVER_PORT=3000
```

```
ENV NODE_ENV="production"
```

```
WORKDIR /tmp/hello-server
```

### NOTA

Al establecer la variable de entorno `NODE_ENV` en `production`, se indica a NPM que ignore las dependencias dentro de la sección `devDependencies` del archivo `package.json`.

Debido a que los paquetes `devDependencies` no son necesarios para ejecutar la aplicación, establecer la variable `NODE_ENV` en `production` reduce el tamaño de la imagen.

La aplicación usa la variable de entorno `SERVER_PORT` para determinar el puerto al que se une.

4. Actualice el directorio de trabajo a `/opt/app-root/src`, que es una ubicación mejor que `/tmp/hello-server`.

```
WORKDIR /opt/app-root/src
```

4. Aplique los metadatos adecuados a la imagen.

1. Agregue una instrucción LABEL para indicar quién es responsable de mantener la imagen.

```
FROM registry.ocp4.example.com:8443/ubi8/nodejs-16:1
```

```
LABEL org.opencontainers.image.authors="Your Name"
```

```
ENV SERVER_PORT=3000
```

2. Agregue más instrucciones LABEL para proporcionar sugerencias sobre la versión y el uso previsto de la imagen del contenedor.

```
LABEL org.opencontainers.image.authors="Your Name"
LABEL com.example.environment="production"
LABEL com.example.version="0.0.1"
ENV SERVER_PORT=3000
```

3. Agregue una instrucción EXPOSE para indicar que la aplicación dentro del contenedor se une al puerto definido en la variable de entorno SERVER\_PORT.

```
ENV SERVER_PORT=3000
ENV NODE_ENV="production"

EXPOSE $SERVER_PORT
WORKDIR /opt/app-root/src
```

La instrucción EXPOSE sirve para fines de documentación. No vincula el puerto en el host que ejecuta el contenedor.

4. El Dockerfile final contiene las siguientes instrucciones:

```
FROM quay.io/piracarter1/ubi8-nodejs-16:1
LABEL org.opencontainers.image.authors="Jesus Arriola"
LABEL com.example.environment="production"
LABEL com.example.version="0.0.1"
ENV SERVER_PORT=3000
ENV NODE_ENV="production"
EXPOSE $SERVER_PORT
WORKDIR /opt/app-root/src
COPY . .
RUN npm install
CMD npm start
```

5. Cree la imagen de contenedor con el Dockerfile actualizado. Utilice la etiqueta de imagen hello-server:best.

```
[student@workstation hello-server]$ docker build -t hello-server:best .  
...output omitted...  
Successfully tagged localhost/hello-server:best  
...output omitted...
```

6. Inspeccione la imagen del contenedor para observar los metadatos agregados.

```
[student@workstation hello-server]$ docker inspect hello-server:best \  
-f '{{.Config.Env}}'  
...output omitted... SERVER_PORT=3000 NODE_ENV=production]
```

7. Verifique que la aplicación funcione.

```
[student@workstation hello-server]$ docker run -d --rm --name hello-best \  
-p 3000:3000 hello-server:best  
d6c3...9f21  
[student@workstation hello-server]$ curl http://localhost:3000/greet ;echo  
{"hello":"world"}  
[student@workstation hello-server]$ docker stop hello-best  
hello-better
```

8. Pase al directorio de inicio.

```
[student@workstation hello-server]$ cd  
no output expected
```

# Compilación de imágenes con instrucciones avanzadas de Containerfile

## Objetivos

- Crear un Dockerfile que use las mejores prácticas.

## Instrucciones avanzadas de Containerfile

Puede empaquetar su aplicación en un contenedor con las dependencias del tiempo de ejecución de la aplicación. Sin embargo, los desarrolladores suelen crear contenedores que tienen varias desventajas, por ejemplo:

- El contenedor está vinculado a un entorno específico y requiere una reconstrucción antes de implementarlo en un entorno de producción.
- El contenedor contiene herramientas para desarrolladores, como un depurador, editores de texto o compiladores.
- El contenedor contiene dependencias en tiempo de compilación que no son necesarias en tiempo de ejecución.
- El contenedor genera un gran volumen de archivos almacenados en el sistema de archivos de copia en escritura, lo que limita el rendimiento de la aplicación.

Esta sección se centra en los patrones de contenedores avanzados que lo ayudan a limitar los problemas mencionados anteriormente, como:

- Reducción del tamaño de almacenamiento de imágenes mediante el uso del patrón de compilación de contenedores de varias etapas.
- Personalización del tiempo de ejecución del contenedor con variables de entorno.
- Uso de volúmenes para disminuir el tamaño del contenedor y aumentar el rendimiento de la escritura de archivos.

## La instrucción ENV

La instrucción ENV le permite especificar la configuración dependiente del entorno, por ejemplo, nombres de host, puertos o nombres de usuario. La aplicación contenerizada puede usar las variables de entorno en tiempo de ejecución.

Para incluir una variable de entorno, use el formato `key=value`. El siguiente ejemplo declara una variable `DB_HOST` con el nombre de host de la base de datos.

```
ENV DB_HOST="database.example.com"
```

Luego, puede recuperar la variable de entorno en su aplicación. El siguiente ejemplo es un script de Python que recupera la variable de entorno `DB_HOST`.

```
from os import environ

DB_HOST = environ.get('DB_HOST')

# Connect to the database at DB_HOST...
```

## La instrucción ARG

Use la instrucción ARG para definir variables de tiempo de compilación, generalmente para hacer una compilación de contenedor personalizable.

Puede configurar opcionalmente un valor de variable de tiempo de compilación predeterminado si el desarrollador no lo proporciona en el momento de compilación. Use la siguiente sintaxis para definir una variable de tiempo de compilación:

```
ARG key[=default value]
```

Considere el siguiente ejemplo de Containerfile:

```
ARG VERSION="1.16.8" \
    BIN_DIR=/usr/local/bin/

RUN curl "https://dl.example.io/${VERSION}/example-linux-amd64" \
    -o ${BIN_DIR}/example
```

Los desarrolladores suelen configurar las instrucciones ENV mediante la instrucción ARG. Esto es útil para conservar las variables de tiempo de compilación para el tiempo de ejecución.

Considere el siguiente ejemplo de Dockerfile:

```
ARG VERSION="1.16.8" \
    BIN_DIR=/usr/local/bin/

ENV VERSION=${VERSION} \
    BIN_DIR=${BIN_DIR}

RUN curl "https://dl.example.io/${VERSION}/example-linux-amd64" \
```

```
-o ${BIN_DIR}/example
```

En el ejemplo anterior, la instrucción ENV usa el valor de la instrucción ARG para configurar las variables de entorno de tiempo de ejecución.

Si no configura los valores predeterminados ARG pero debe configurar los valores predeterminados ENV, use la sintaxis `${VARIABLE:-DEFAULT_VALUE}`, como:

```
ARG VERSION \
    BIN_DIR

ENV VERSION=${VERSION:-1.16.8} \
    BIN_DIR=${BIN_DIR:-/usr/local/bin/}

RUN curl "https://dl.example.io/${VERSION}/example-linux-amd64" \
    -o ${BIN_DIR}/example
```

## La instrucción VOLUME

Utilice la instrucción VOLUME para almacenar datos de forma persistente. El valor es la ruta donde Docker monta el volumen persistente dentro del contenedor. La instrucción VOLUME acepta más de una ruta para crear varios volúmenes.

Por ejemplo, el siguiente Dockerfile crea un volumen para almacenar datos de PostgreSQL.

```
FROM registry.redhat.io/rhel9/postgresql-13:1

VOLUME /var/lib/pgsql/data
```

En el Dockerfile anterior, si agrega instrucciones que actualizan `/var/lib/pgsql/data` después de la instrucción VOLUME, esas instrucciones se ignoran.

Para recuperar el directorio local usado por un volumen, puede usar el comando `docker inspect VOLUME_ID`.

```
[user@host ~]$ docker inspect VOLUME_ID
[
  {
    "Name": "VOLUME_ID",
    "Driver": "local",
```



```

    "Mountpoint": "/home/your-name/.local/share/containers/storage/volumes/VOLUME_ID/_data",
    ...output omitted...
    "Anonymous": true
  }
]

```

El campo `Mountpoint` le da la ruta absoluta al directorio donde existe el volumen en su sistema de archivos host. Los volúmenes creados a partir de la instrucción `VOLUME` tienen un ID aleatorio en el campo `Name` y se consideran volúmenes `Anonymous`.

Para eliminar volúmenes no utilizados, use el comando `docker volume prune`.

```

[user@host ~]$ docker volume prune
WARNING! This will remove all volumes not used by at least one container. The following volumes will be removed:
8c0c...bcb8c
Are you sure you want to continue? [y/N] y
8c0c...bcb8c

```

Puede crear un volumen *con nombre* con el comando `docker volume create`.

```

[user@host ~]$ docker volume create VOLUME_NAME
VOLUME_NAME

```

También puede formatear el comando `docker volume ls` para incluir el campo `Mountpoint` de cada volumen. El almacenamiento de datos persistentes con volúmenes se trata en profundidad más adelante en el curso.

```

[user@host ~]$ docker volume ls \
  --format="{{.Name}}\t{{.Mountpoint}}"
0a8c...82c2  /home/your-name/.local/share/containers/storage/volumes/0a8c...82c2/_data
252d...b2ed  /home/your-name/.local/share/containers/storage/volumes/252d...b2ed/_data

```

## Las instrucciones `ENTRYPOINT` y `CMD`

Las instrucciones `ENTRYPOINT` y `CMD` especifican el comando que se ejecutará cuando se inicie el contenedor. Un `Containerfile` válido debe tener al menos una de estas instrucciones.

La instrucción `ENTRYPOINT` define un ejecutable, o comando, que siempre es parte de la ejecución del contenedor. Esto significa que se pasan argumentos adicionales al comando proporcionado.

Considere el siguiente ejemplo:

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.5
ENTRYPOINT ["echo", "Hello"]
```

La ejecución de un contenedor desde el Containerfile anterior sin argumentos imprime "Hello".

```
[user@host ~]$ docker run my-image
Hello
```

Si proporciona al contenedor los argumentos "Hola" y "Mundo", este imprime "Hello Hola Mundo".

```
[user@host ~]$ docker run my-image Hola Mundo
Hello Hola Mundo
```

Si solo usa la instrucción `CMD`, el paso de argumentos al contenedor anula el comando provisto en la instrucción `CMD`.

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.5
CMD ["echo", "Hello", "World"]
```

La ejecución de un contenedor desde el Containerfile anterior sin argumentos imprime Hello Red Hat.

```
[user@host ~]$ docker run my-image
Hello World
```

La ejecución de un contenedor con el argumento `whoami` anula el comando `echo`.

```
[user@host ~]$ docker run my-image whoami
root
```

Cuando un Containerfile especifica tanto `ENTRYPOINT` como `CMD`, `CMD` cambia su comportamiento. En este caso, los valores proporcionados a `CMD` se pasan como argumentos predeterminados a `ENTRYPOINT`.

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.5
ENTRYPOINT ["echo", "Hello"]
CMD ["Red", "Hat"]
```

El ejemplo anterior imprime Hello Red Hat cuando ejecuta el contenedor.

```
[user@host ~]$ docker run my-image
Hello Red Hat
```

Si proporciona el argumento Docker al contenedor, imprime Hello Docker.

```
[user@host ~]$ docker run my-image Peter
Hello Peter
```

Las instrucciones ENTRYPOINT y CMD tienen dos formatos para ejecutar comandos:

### Matriz de texto

El ejecutable toma la forma de una matriz de texto, como:

```
ENTRYPOINT ["executable", "param1", ... "paramN"]
```

En este formulario, debe proporcionar la ruta completa al ejecutable.

### Forma de cadena

El comando y los parámetros se escriben en forma de texto, como:

```
CMD executable param1 ... paramN
```

La cadena envuelve el ejecutable en un comando de shell como `sh -c "executable param1 ... paramN"`. Esto es útil cuando necesita procesamiento de shell, por ejemplo, para la sustitución de variables.

### Compilaciones de varias etapas

Una compilación de varias etapas usa varias instrucciones FROM para crear varios procesos de compilación de contenedores independientes, también llamados *etapas*. Cada etapa puede usar una imagen base diferente y puede copiar archivos entre etapas. La imagen de contenedor resultante consta de la última etapa.

Las compilaciones de varias etapas pueden reducir el tamaño de la imagen manteniendo solo las dependencias de tiempo de ejecución necesarias. Por ejemplo, considere el siguiente ejemplo, donde una aplicación Node está containerizada.

```
FROM registry.redhat.io/ubi8/nodejs-14:1
```

```
WORKDIR /app
```

```
COPY . .
```

```
RUN npm install
```

```
RUN npm run build
```

```
RUN serve build
```

El comando `npm install` instala los paquetes NPM requeridos, que incluyen paquetes que solo se necesitan en el momento de la compilación. El comando `npm run build` usa los paquetes para crear una compilación de aplicaciones optimizada y lista para la producción. Luego, el contenedor usa un servidor HTTP para exponer la aplicación con el comando `serve`.

Si crea una imagen a partir del Dockerfile anterior, esta va a tener las dependencias de tiempo de compilación y las de tiempo de ejecución, lo que aumentará el tamaño de la imagen. La imagen resultante también contiene el tiempo de ejecución de Node.js, que no se usa en el tiempo de ejecución del contenedor, pero podría aumentar la superficie de ataque del contenedor.

Para evitar este problema, puede definir dos etapas:

- **Primera etapa:** compile la aplicación.
- **Segunda etapa:** copie y entregue los archivos estáticos mediante un servidor HTTP, como NGINX o Apache Server.

El siguiente ejemplo implementa el proceso de compilación de dos etapas.

```
# First stage
```

```
FROM registry.access.redhat.com/ubi8/nodejs-14:1 as builder
```

```
COPY ./ /opt/app-root/src/
```

```
RUN npm install
```

```
RUN npm run build
```

```
# Second stage
```

```
FROM registry.access.redhat.com/ubi8/nginx-120
```

```
COPY --from=builder /opt/app-root/src/ /usr/share/nginx/html
```

Defina la primera etapa con un alias. La segunda etapa usa el alias `builder` para hacer referencia a esta etapa.

Cree la aplicación.

Defina la segunda etapa sin un alias. Usa una imagen base `ubi8/nginx-120` para entregar la versión de la aplicación lista para producción.

Copie los archivos de la aplicación en un directorio en la imagen final. El indicador `--from` muestra que Docker copia los archivos desde la etapa `builder`.

## Examinar las capas de datos del contenedor

Las imágenes de contenedor usan un sistema de archivos en capas de copia en escritura (COW). Cuando crea un Dockerfile, las instrucciones `RUN`, `COPY` y `ADD` crean *capas* (a veces denominadas *blobs*).

El sistema de archivos en capas COW garantiza que un contenedor permanezca inmutable. Cuando inicia un contenedor, docker crea y monta una capa delgada, efímera y de escritura sobre las capas de la imagen del contenedor. Cuando elimina el contenedor, Docker elimina la capa fina de escritura. Esto significa que todas las capas del contenedor permanecen idénticas, excepto la capa fina que se puede escribir.

### *Capas de imágenes en caché*

Dado que todas las capas de contenedores son idénticas, varios contenedores pueden compartir las capas. Esto significa que Docker puede almacenar capas en caché y crear solo aquellas capas que se modifican o no se almacenan en caché.

Puede usar el almacenamiento en caché para reducir el tiempo de compilación. Por ejemplo, considere una aplicación Dockerfile de Node.js:

```
...content omitted...  
COPY . /app/  
...content omitted...
```

La instrucción anterior copia todos los archivos y directorios del directorio Containerfile en el directorio `/app` del contenedor. Esto significa que cualquier cambio en la aplicación dará como resultado la reconstrucción de todas las capas después de la capa `COPY`.

Puede cambiar las instrucciones de la siguiente manera:

```
...content omitted...  
COPY package.json /app/  
RUN npm ci --production  
COPY src ./src  
...content omitted...
```

Esto significa que si cambia el código fuente de su aplicación en el directorio `src` y reconstruye su imagen de contenedor, la capa de dependencia se almacena en caché y se omite, lo que reduce el tiempo de compilación. Docker reconstruye la capa de dependencia solo cuando cambia el archivo `package.json`.

### *Reducir capas de imágenes*

Puede reducir la cantidad de capas de imágenes de contenedores, por ejemplo, encadenando instrucciones. Tenga en cuenta las siguientes instrucciones RUN:

```
RUN mkdir /etc/gitea  
RUN chown root:gitea /etc/gitea  
RUN chmod 770 /etc/gitea
```

Puede reducir la cantidad de capas a una encadenando los comandos. En Linux, puede encadenar comandos usando el doble ampersand (`&&`). También puede usar el carácter de barra invertida (`\`) para dividir un comando largo en varias líneas.

En consecuencia, el comando RUN resultante tiene el siguiente aspecto:

```
RUN mkdir /etc/gitea && \  
    chown root:gitea /etc/gitea && \  
    chmod 770 /etc/gitea
```

La ventaja de encadenar comandos es que crea menos capas de imágenes de contenedor, lo que generalmente da como resultado imágenes más pequeñas.

Sin embargo, los comandos encadenados son más difíciles de depurar y almacenar en caché.

Los desarrolladores suelen reducir la cantidad de capas mediante el uso de compilaciones de varias etapas en combinación con el encadenamiento de algunos comandos.

## Ejercicio guiado: Compilación de imágenes con instrucciones avanzadas de Containerfile

Usar una aplicación Python simple para crear un Dockerfile con instrucciones avanzadas.

### Resultados

Debería poder trabajar con:

- Compilaciones de varias etapas.
- La instrucción USER.
- La instrucción WORKDIR.
- La instrucción ENV.
- La instrucción VOLUME.

### Procedimiento 4.2. Instrucciones

1. Examine la aplicación de ejercicios.

1. Diríjase al directorio ~/D0188/labs/custom-advanced.

```
[student@workstation ~]$ cd ./cdiputadosdocker/custom-advanced  
no output expected
```

2. Examine el archivo main.py, que contiene una aplicación básica de Python. La aplicación lee el archivo numbers.txt, imprime su contenido y agrega otro número al archivo. La aplicación usa una variable de entorno para ubicar el archivo numbers.txt.

```
...output omitted...
```

```
FILE = environ.get('FILE')
```

```
...output omitted...
```

3. Examine el archivo Dockerfile incompleto.
2. Use una compilación de varias etapas para completar el archivo Dockerfile.

Agregue una etapa que use la imagen

base registry.ocp4.example.com:8443/redhattraining/docker-random-numbers.

La imagen base contiene el script random\_generator.py. Use el script para generar un archivo numbers.txt y copie el archivo a la etapa final.

1. Cree una etapa agregando una instrucción FROM.

```
FROM quay.io/piracarter1/docker-random-numbers as generator
```

```
FROM quay.io/piracarter1/ubi7-python-38:1-104  
...output omitted...
```

2. Genere el archivo numbers.txt.

```
FROM quay.io/piracarter1/docker-random-numbers as generator
```

```
RUN python3 random_generator.py
```

```
FROM quay.io/piracarter1/ubi7-python-38:1-104  
...output omitted...
```

3. En la segunda etapa de compilación, copie el archivo numbers.txt mediante la opción --from=generator de la instrucción COPY. Use la opción --chown para hacer que el usuario default sea el nuevo propietario.

```
...output omitted...
```

```
WORKDIR /redhat
```

```
COPY --from=generator --chown=default /app/numbers.txt materials/numbers.txt
```

```
COPY main.py .
```

```
...output omitted...
```

3. Agregue la variable de entorno FILE para establecer una ruta al archivo numbers.txt.

```
FROM quay.io/piracarter1/docker-random-numbers as generator
```

```
RUN python3 random_generator.py
```

```
FROM quay.io/piracarter1/ubi7-python-38:1-104
```

```
ENV FILE="/redhat/materials/numbers.txt"
```



```
USER default
WORKDIR /redhat

COPY --from=generator --chown=default /app/numbers.txt materials/numbers.txt
COPY main.py .
CMD python3 main.py
```

#### 4. Compile y pruebe la imagen.

1. En una terminal de línea de comandos, compile la imagen a partir del Containerfile.

```
[student@workstation custom-advanced]$ docker build -f Containerfile -t redhat-local/custom-advanced .
...output omitted...
Successfully tagged localhost/redhat-local/custom-advanced:latest
3360...cc21
```

2. Ejecute la imagen.

```
[student@workstation custom-advanced]$ docker run --rm \
--name=custom-advanced redhat-local/custom-advanced
Current content: ['17 72 97 8 32 15 63 97 57']
Writing another number...
Current content: ['17 72 97 8 32 15 63 97 57 4']
```

La aplicación usa la variable de entorno para leer el archivo y escribir contenido en el archivo. Los números que obtenga pueden diferir porque se generan aleatoriamente, sin embargo, siempre se agrega 4 al final.

5. Agregue un punto de montaje al directorio /redhat/materials.

1. En el Containerfile, agregue una instrucción VOLUME.

```
...output omitted...
COPY main.py .

VOLUME /redhat/materials
```

```
CMD python3 main.py
```

## 6. Compile y pruebe la imagen.

1. En su terminal de línea de comandos, compile la imagen a partir del Containerfile.

```
[student@workstation custom-advanced]$ docker build -f Containerfile -t redhat-local/custom-advanced .  
  
...output omitted...  
  
Successfully tagged localhost/redhat-local/custom-advanced:latest  
4872...0ee0
```

2. Cree un contenedor a partir de la imagen.

```
[student@workstation custom-advanced]$ docker run --name=custom-advanced \redhat-local/custom-advanced  
  
Current content: ['46 37 98 39 70 53 81 44 59']  
  
Writing another number...  
  
Current content: ['46 37 98 39 70 53 81 44 59 4']
```

### NOTA

El comando anterior no tiene la opción `--rm`. Si usa la opción `--rm`, Docker elimina los volúmenes anónimos que usa el contenedor.

## 7. Lea el archivo `numbers.txt` en su sistema de archivos local.

1. Inspeccione el contenedor para identificar la ruta del volumen anónimo en el sistema de archivos.

```
[student@workstation custom-advanced]$ docker inspect custom-advanced  
  
...output omitted...  
  
    "Mounts": [  
      {  
        "Type": "volume",  
        "Name": "6e5ac30f1eba0a65e16e30b0a14d7b688a2ebed130c0be6532c07c1ae6cc8402",  
        "Source": "/var/lib/docker/volumes/6e5ac30f1eba0a65e16e30b0a14d7b688a2ebed130c0be6532c07c1ae6cc8402/_data",  
        "Destination": "/redhat/materials",
```

```
"Driver": "local",...output omitted...
```

Puede limitar la salida del comando `docker inspect` mediante la opción `--format`.

```
[student@workstation custom-advanced]$ docker inspect custom-advanced \
--format="{{ (index .Mounts 0).Source}}"
/home/student/.local/share/containers/storage/volumes/40ed...96bf/_data
```

2. Use la ruta `Source` para leer el archivo `numbers.txt` desde su sistema de archivos local.

```
[student@workstation custom-advanced]$ cat SOURCE/numbers.txt
46 37 98 39 70 53 81 44 59 4
```

## Capítulo 5. Persistencia de datos

### Montaje de volúmenes

### Objetivos

- Describir el proceso para montar volúmenes y casos de uso comunes.

#### Sistema de archivos de copia en escritura (COW)

Al crear una imagen de contenedor, cada instrucción que modifica el sistema de archivos del contenedor crea una nueva capa de datos de solo lectura. Debido a que no puede modificar los datos en una capa existente, cada capa contiene un conjunto de cambios, o *diferencias*, de la capa anterior.

Considere el siguiente Containerfile:

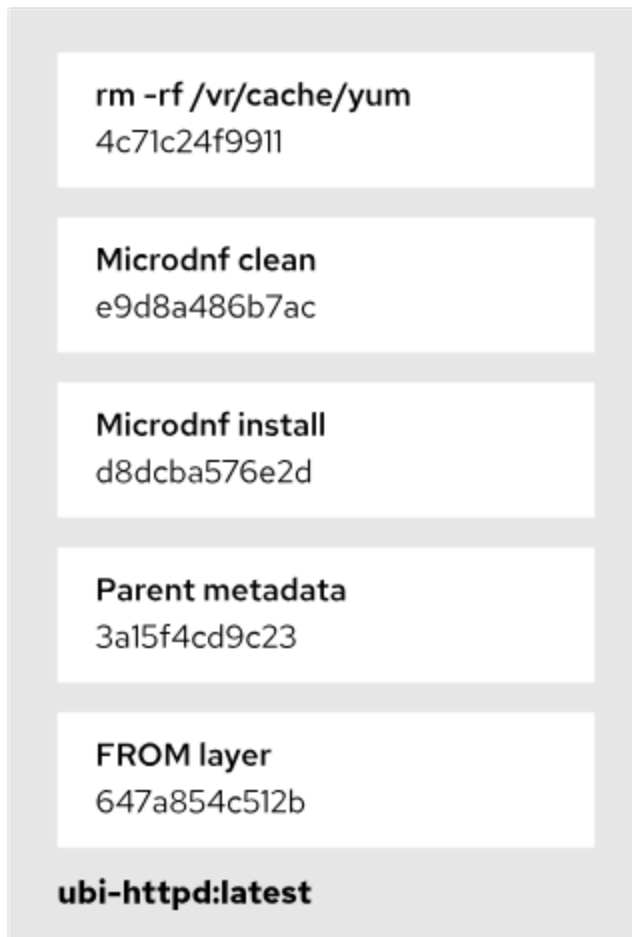
```
FROM registry.access.redhat.com/ubi8/ubi-minimal

RUN microdnf install httpd
RUN microdnf clean all
RUN rm -rf /var/cache/yum

CMD httpd -DFOREGROUND
```

Puede inspeccionar las capas de la imagen con el comando `docker-image-tree`:

```
[user@host ~]$ docker history ubi-httpd
Image ID: bdd298c12db7
Tags:      [localhost/ubi-httpd:latest]
Size:      166.8MB
Image Layers
├─ ID: 647a854c512b Size: 94.77MB
├─ ID: 3a15f4cd9c23 Size: 20.48kB Top Layer of: [registry.access.redhat.com/ubi8/ubi-minimal:latest]
├─ ID: d8dcba576e2d Size: 68.45MB
├─ ID: e9d8a486b7ac Size: 3.581MB
└─ ID: 4c71c24f9911 Size: 4.608kB Top Layer of: [localhost/ubi-httpd:latest]
```



Tenga en cuenta que debido a que el comando `microdnf clean all` crea una nueva capa, la imagen de contenedor resultante aún contiene los datos que eliminó el comando `microdnf clean all`. Aunque no se puede acceder a ellos en tiempo de ejecución, los datos están contenidos en la capa anterior, la capa `d8dcba576e2d`, y contribuyen al tamaño general del contenedor.

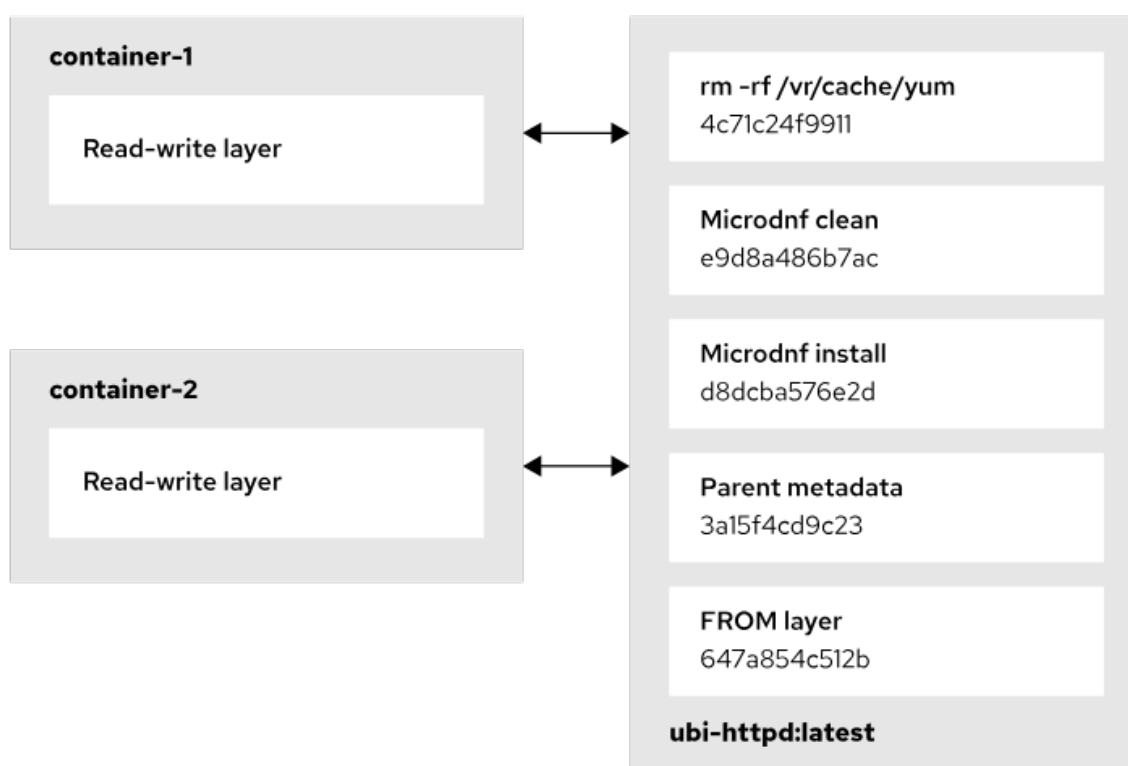
Para eliminar datos innecesarios del comando `microdnf install`, use los comandos `microdnf clean all` y `rm -rf /var/cache/yum` en la misma capa contenedora. Por ejemplo, encadene los comandos en una instrucción `RUN`:

```
FROM registry.access.redhat.com/ubi8/ubi-minimal
RUN microdnf install httpd && \
    microdnf clean all && \
```

```
rm -rf /var/cache/yum
```

```
CMD httpd -DFOREGROUND
```

Cuando crea una instancia de la imagen del contenedor, por ejemplo, mediante el comando `docker run`, Docker crea una capa *delgada* de contenedor de lectura y escritura sobre las capas anteriores. Esto significa que varios contenedores que usan una imagen de contenedor comparten capas de solo lectura y difieren en la capa de lectura y escritura en tiempo de ejecución. Al eliminar un contenedor, Docker destruye la capa de lectura y escritura. Esto significa que los datos en tiempo de ejecución del contenedor son efímeros.



Consulte la sección de referencias para obtener más información sobre la implementación de Docker del sistema de archivos de unión de superposición.

### *Implicancias de un sistema de archivos COW*

Los archivos que se encuentran en las capas de imágenes del contenedor están disponibles para la capa de datos de lectura y escritura mediante el uso de un sistema de archivos de unión. En tiempo de ejecución, el sistema de archivos del contenedor consta de una unión de archivos en las capas definidas.

Es posible modificar un archivo o sus atributos en tiempo de ejecución en varios pasos:

- Docker ubica el archivo solicitado en la capa más cercana (más alta) y lo copia en la capa de tiempo de ejecución.
- En la capa de tiempo de ejecución, cuando el archivo está disponible para operaciones de lectura y escritura, un proceso contenerizado puede modificar el archivo.

La implementación de cómo se comportan los archivos cuando un proceso intenta leer o escribir un archivo depende del controlador de almacenamiento.

Usar sistemas de archivos de unión proporciona operaciones de lectura eficientes. La arquitectura de datos COW también promueve el intercambio y la reutilización de capas de datos entre contenedores separados, porque las capas son inmutables.

Sin embargo, los sistemas de archivos de unión presentan un cuello de botella en el rendimiento para los procesos contenerizados de escritura intensiva.

### **Almacenar datos en la máquina host**

Los desarrolladores suelen escribir aplicaciones que requieren el almacenamiento de datos de forma persistente. Para tales casos, Docker implementa montajes externos mediante el uso de *volúmenes* y *montajes de enlace*.

Esto es útil por las siguientes razones:

#### **Persistencia**

Los datos montados son persistentes en las eliminaciones de contenedores. Debido a que los contenedores son efímeros, no se puede acceder a los datos de la capa de lectura y escritura en tiempo de ejecución después de la eliminación del contenedor.

#### **Uso del sistema de archivos del host**

Los datos montados generalmente no implementan el sistema de archivos COW. En consecuencia, los procesos contenerizados con mucha escritura pueden escribir datos en un montaje sin las limitaciones de los sistemas de archivos COW para las operaciones de escritura.

#### **Facilidad para compartir**

Los datos montados se pueden compartir entre varios contenedores al mismo tiempo. Esto significa que un contenedor puede escribir en un montaje y otro contenedor puede leer los mismos datos.

Además, los montajes no se limitan a la máquina host del contenedor. Puede alojar montajes de datos en la red, por ejemplo, mediante el protocolo NFS.

Los volúmenes son montajes de datos administrados por Docker. Los montajes de enlace son montajes de datos gestionados por el usuario.

Tanto los volúmenes como los montajes de enlace pueden usar el parámetro `--volume` (o `-v`).

```
--volume /path/on/host:/path/in/container:OPTIONS
```

En la sintaxis anterior, la parte `:OPTIONS` es opcional. Tenga en cuenta que puede especificar rutas de host mediante rutas absolutas, como `/home/user/www`, o rutas relativas, como `./www`, que se refiere al directorio `www` en el directorio de trabajo actual.

Use `-v volume_name:/path/in/container` para consultar un volumen.

Alternativamente, puede usar el parámetro `--mount` con la siguiente sintaxis:

```
--mount type=TYPE,source=/path/on/host,destination=/path/in/container
```

El parámetro `--mount` especifica explícitamente el tipo de volumen, como:

- `bind` para los montajes de enlace.
- `volume` para los montajes de volúmenes.
- `tmpfs` para crear montajes efímeros de solo memoria.

El parámetro `--mount` es la forma preferida de montar directorios en un contenedor. Sin embargo, debido a que el parámetro `-v` todavía se usa ampliamente, este curso usa ambos estilos.

Los desarrolladores suelen usar montajes de enlace para realizar pruebas o para montar archivos específicos del entorno, como archivos de propiedades. Debido a que Docker gestiona el sistema de archivos de volumen, use volúmenes para garantizar un comportamiento de montaje uniforme en todos los sistemas. Además, use volúmenes para usos avanzados, como montar un volumen remoto en NFS.

## Almacenamiento de datos con montajes de enlace

Los montajes de enlace pueden existir en cualquier lugar del sistema host.

Por ejemplo, para montar el directorio `/www` de su máquina host en el directorio `/var/www/html` dentro del contenedor con la opción de solo lectura, use el siguiente comando `docker-run`:

```
[user@host ~]$ docker run -p 8080:8080 --volume /www:/var/www/html:ro \
registry.access.redhat.com/ubi8/httpd-24:latest
```



## Almacenamiento de datos con volúmenes

Los volúmenes permiten que Docker gestione los montajes de datos. Puede gestionar volúmenes mediante el comando `docker volume`.

Para crear un volumen llamado `http-data`, use el siguiente comando:

```
[user@host ~]$ docker volume create http-data
d721d941960a2552459637da86c3074bbba12600079f5d58e62a11caf6a591b5
```

Puede inspeccionar el volumen con el comando `docker volume inspect`:

```
[user@host ~]$ docker volume inspect http-data
[
  {
    "CreatedAt": "2023-08-31T05:26:36-05:00",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/http-data/_data",
    "Name": "http-data",
    "Options": null,
    "Scope": "local"
  }
]
```

*Debido a que Docker gestiona el volumen, no es necesario configurar los permisos de SELinux.*

## Almacenamiento de datos con un montaje tmpfs

Algunas aplicaciones no pueden usar el sistema de archivos COW predeterminado en un directorio específico por razones de rendimiento, pero no use la persistencia o el uso compartido de datos para ese directorio.

Para tales casos, puede usar el tipo de montaje `tmpfs`, lo que significa que los datos en un montaje son efímeros, pero no usan el sistema de archivos COW:

```
[user@host ~]$ docker run -e POSTGRESQL_ADMIN_PASSWORD=redhat --network lab-net \
--mount type=tmpfs,tmpfs-size=512M,destination=/var/lib/pgsql/data \
registry.redhat.io/rhel9/postgresql-13:1
```

# Ejercicio guiado: Montaje de volúmenes

Insertar archivos en un contenedor mediante un montaje de enlace y un volumen.

## Resultados

Usted deberá ser capaz de realizar lo siguiente:

- Use montajes de enlace con sus contenedores.
- Use `docker unshare` para solucionar problemas de permisos con montajes de enlace.
- Cree volúmenes con nombre.
- Importe archivos en volúmenes con nombre.

## Procedimiento 5.1. Instrucciones

1. Examine el archivo `~/cdiputadosdocker/persisting-mounting/docker-python-server/Dockerfile`. Esta imagen usa el directorio `/server` como directorio web raíz para el servidor HTTP de Python.

La imagen del contenedor `quay.io/piracarter1/docker-python-server` se basa en este Dockerfile.

2. Copie el archivo `index.html` en el directorio `~/www`.

El directorio `~/www` funciona como un montaje de enlace que contiene el HTML para el contenedor.

```
[student@workstation ~]$ cd
[student@workstation ~]$ mkdir www
[student@workstation ~]$ cp cdiputadosdocker/persisting-mounting/index.html www/
no output expected
```

3. Pruebe el contenedor `docker-python-server` con el directorio `~/www` montado como un montaje de enlace.

1. Inicie un contenedor con los siguientes parámetros:

- Vincule el directorio `~/www` en el sistema de host al directorio `/server` del contenedor.
  - Use la opción `:z` para establecer la etiqueta SELinux correcta en el montaje de enlace.

- Designarle un nombre al contenedor `docker-server`.
- Use la opción `--rm`.
- Use las opciones `-ti` para mostrar la salida del contenedor.
- Use la imagen `registry.ocp4.example.com:8443/redhattraining/docker-python-server`.
- Vincule el puerto `8000` en la máquina local al puerto `8000` en el contenedor.

```
[student@workstation ~]$ docker run -ti --rm --name docker-server \
  --volume ~/www:/server:Z -p 8000:8000 \
  quay.io/piracarter1/docker-python-server
...output omitted...
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/)
```

2. En un explorador web, diríjase a `localhost:8000`.
4. Cree un volumen con nombre con la página `index.html`.
1. Cree un volumen denominado `html-vol`.

```
[student@workstation ~]$ docker volume create html-vol
html-vol
```

2. Cambie al directorio de laboratorio `persisting-mounting`.

```
[studentp@workstation ~]$ cd cdiputadosdocker/persisting-mounting/
no output expected
```

3. Importe el archivo de almacenamiento `index.tar.gz`, que contiene `index.html`, en el volumen `html-vol`.

```
[student@workstation persisting-mounting]$ docker volume inspect html-vol
# encuentre la línea "Mountpoint" y guarde la ruta mostrada.
[student@workstation persisting-mounting]$ sudo tar -xzf index.tar.gz -C /var/lib/docker/volumes/html-vol/_data
no output expected
```

5. Inicie un nuevo contenedor que use la imagen `docker-server`. Use un montaje de volumen en lugar del montaje de enlace.

1. Inicie el contenedor `docker-server`.

Vincule el volumen `html-vol` como un directorio de solo lectura `/server` dentro del contenedor. El resto de los parámetros siguen siendo los mismos.

```
[student@workstation ~]$ docker run -ti --rm --name docker-server -p 8000:8000 \
    --mount 'type=volume,source=html-vol,destination=/server,ro' \
    quay.io/piracarter1/docker-python-server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/)
```

2. En un explorador web, acceda a `localhost:8000`. Se le presenta la página `index.html`.
3. Presione **Ctrl+c** para detener el contenedor.

## Trabajo con bases de datos

### Objetivos

- Construir bases de datos contenerizadas.

#### Contenedores de bases de datos con estado

Los contenedores de la base de datos suelen mantener el estado porque deben conservar los datos. Por esta razón, se denominan *contenedores con estado*. Estos contenedores difieren de otros tipos de software contenerizados, como las API web o los proxies, que no necesitan mantener el estado y, por lo tanto, se denominan contenedores sin estado.

#### NOTA

Los contenedores de bases de datos pueden ser sin estado si no es necesario conservar los datos. Las cachés basadas en la memoria o las bases de datos de prueba efímeras son ejemplos de contenedores de bases de datos sin estado.

En comparación con los contenedores sin estado, los contenedores con estado presentan los siguientes desafíos:

#### Portabilidad

La capacidad de mover el contenedor entre diferentes entornos. Crear un contenedor en el nuevo entorno ya no es suficiente y debe proporcionar al contenedor datos actualizados.

#### Escalabilidad

La capacidad de mejorar el rendimiento aumentando la cantidad de réplicas de un contenedor. Muchas tecnologías de bases de datos no están diseñadas para que los datos sean operados por más de un proceso de base de datos a la vez.

#### Disponibilidad

La capacidad de mantener el sistema en funcionamiento en caso de que un contenedor se bloquee siempre que ese contenedor tenga otra réplica en ejecución. Aunque los sistemas de bases de datos incluyen configuraciones para alta disponibilidad, Docker por sí solo no proporciona las características necesarias para este tipo de configuración.

Para lograr la portabilidad con contenedores con estado, debe proporcionarles un estado válido en diferentes entornos. Por ejemplo, es posible que sus contenedores necesiten datos de muestra para entornos de desarrollo y prueba. Sin embargo, en producción, los contenedores suelen requerir datos reales y actualizados.

La escalabilidad y la disponibilidad no son requisitos para los entornos de desarrollo o prueba. Se abordan en entornos de producción mediante el uso de características que

ofrecen los sistemas de orquestación, como Red Hat OpenShift, junto con operadores de bases de datos especializados.

### **Buenas prácticas para contenedores de bases de datos**

Las siguientes prácticas proporcionan beneficios al contener los procesos de la base de datos:

#### **Use la instrucción VOLUME en Containerfiles**

Cuando cree una imagen de contenedor de base de datos personalizada, use la instrucción VOLUME para crear puntos de montaje en los directorios de almacenamiento de la base de datos. Los puntos de montaje evitan el uso del sistema de archivos de copia en escritura (COW), que no funciona bien con datos con estado.

Cuando crea un contenedor a partir de una imagen que usa la instrucción VOLUME, Docker crea un volumen anónimo y lo conecta al contenedor.

#### **Monte el directorio de datos en un volumen con nombre**

Use un volumen con nombre para montar el directorio de datos de su base de datos para agregar persistencia de datos en las recreaciones de contenedores. Los volúmenes también son más portátiles que los montajes de enlace porque no es necesario que el directorio de origen exista en la máquina host.

#### **Crear una red de base de datos**

Si solo las aplicaciones contenerizadas acceden a su base de datos, no es necesario exponer su base de datos a la red del host. Puede omitir exponer el puerto de la base de datos y usar una red Docker para acceder a la base de datos.

Esto proporciona un mejor aislamiento para la base de datos, ya que solo las aplicaciones que comparten la red tienen acceso a la base de datos.

#### **Importar datos de la base de datos**

La configuración del entorno de desarrollo a veces requiere completar la base de datos con datos de muestra para fines de desarrollo. Los entornos de prueba pueden requerir conjuntos de datos más grandes que los entornos de desarrollo, o incluso datos similares a los de producción.

Dependiendo del contenedor de la base de datos que elija, puede haber diferentes enfoques para cargar la base de datos con datos.

#### *Contenedores de bases de datos con características de carga de datos*

Las imágenes del contenedor de la base de datos generalmente configuran un directorio donde puede colocar scripts para iniciar la base de datos. Puede montar los scripts de su

base de datos en ese directorio. El contenedor de la base de datos ejecuta los scripts en la creación del contenedor o al inicio del contenedor.

También puede migrar los datos desde una base de datos en ejecución. Algunos contenedores incluyen una característica de exportación para extraer los datos del contenedor mientras se está ejecutando la base de datos.

### *Cargar datos con un cliente de base de datos*

También puede cargar los datos usando un cliente de base de datos que sea compatible con su servidor de base de datos. Proporcione al cliente un archivo que contenga los datos para cargar y la configuración para conectarse al servidor de la base de datos.

Es posible que el contenedor que ejecuta la base de datos ya incluya ese cliente. En ese caso, debe proporcionar los scripts de la base de datos al contenedor. Puede copiar los scripts en el contenedor con el comando `docker cp`.

```
[user@host ~] docker cp SQL_FILE TARGET_DB_CONTAINER:CONTAINER_PATH
```

El comando anterior copia `SQL_FILE`, el archivo que contiene los datos, del host al contenedor.

Una vez que haya copiado los scripts en el contenedor, ejecute el comando del cliente de la base de datos para cargar los datos. Por ejemplo, para una base de datos PostgreSQL, el siguiente comando ejecuta `SQL_FILE` para crear y completar la base de datos.

```
[user@host ~] docker exec -it DATABASE_CONTAINER \  
psql -U DATABASE_USER -d DATABASE_NAME \  
-f CONTAINER_PATH/SQL_FILE
```

Si una imagen de base de datos no incluye esta característica, puede crear un contenedor que incluya un cliente de base de datos y usar este contenedor para cargar los datos.

Por ejemplo, el siguiente comando crea un contenedor efímero para cargar datos en una base de datos de PostgreSQL:

```
[user@host ~] docker run -it --rm \  
-e PGPASSWORD=DATABASE_PASSWORD \  
-v ./SQL_FILE:/tmp/SQL_FILE:Z \  
--network DATABASE_NETWORK \  
registry.redhat.io/rhel8/postgresql-12:1-113 \  
psql -U DATABASE_USER -h DATABASE_CONTAINER
```

```
-d DATABASE_NAME -f /tmp/SQL_FILE
```

Este comando usa la opción `z` de SELinux para configurar el contexto SELinux y que el contenedor tenga acceso al host `SQL_FILE`.

El comando también usa la red `DATABASE_NETWORK`, que permite que el cliente `psql` en este contenedor use el nombre `DATABASE_CONTAINER` como el nombre de host para el contenedor de la base de datos. Para que funcione el DNS, la red `DATABASE_NETWORK` debe tener el DNS habilitado.

### Exportar datos de la base de datos

Para exportar los datos de la base de datos, puede usar sus comandos de copia de seguridad (backup) presentes en la imagen de contenedor de esta. Por ejemplo, MySQL proporciona el comando `mysqldump` y PostgreSQL proporciona el comando `pg_dump`.

Puede ejecutar el siguiente comando en un contenedor PostgreSQL para exportar la base de datos denominada `DATABASE` a un archivo `BACKUP_DUMP`.

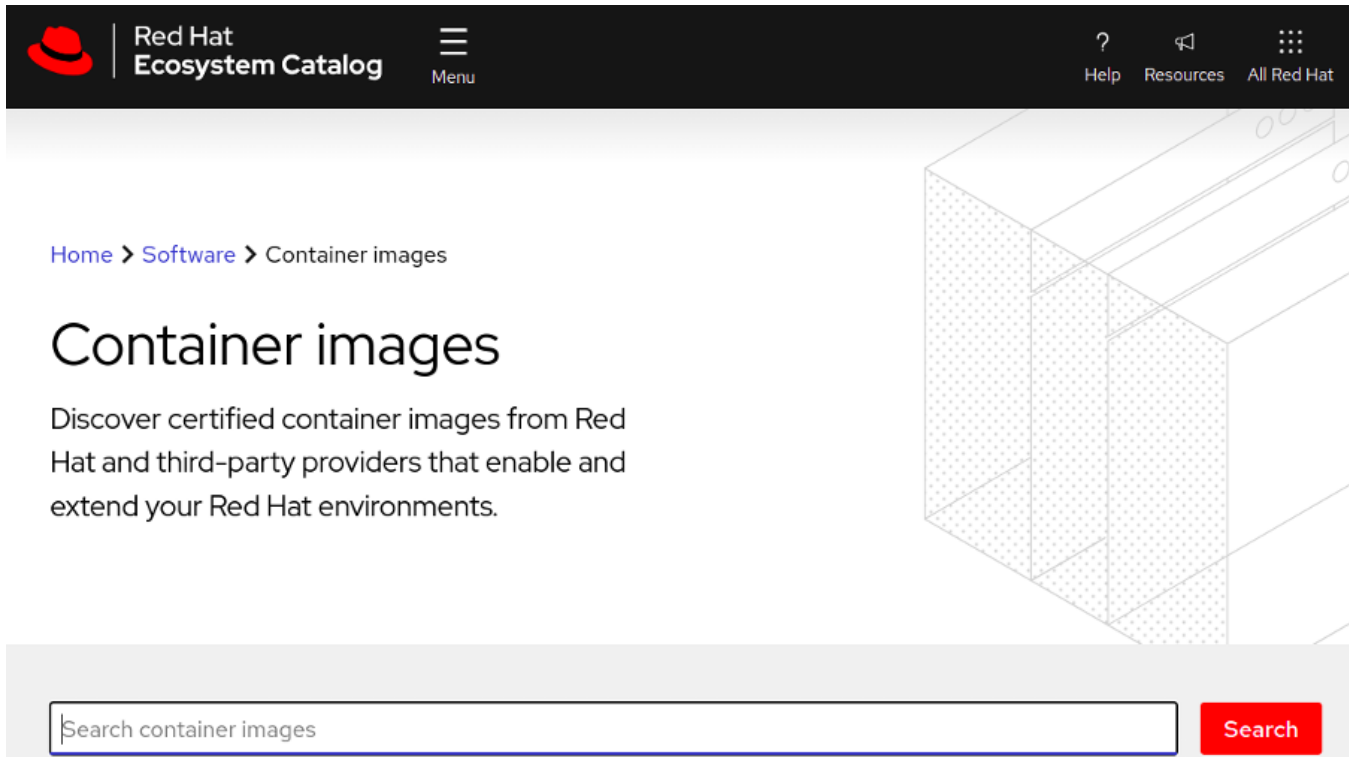
```
[student@workstation ~]$ docker exec POSTGRESQL_CONTAINER \  
    pg_dump -Fc DATABASE -f BACKUP_DUMP  
no output expected
```

Luego, puede copiar el archivo de volcado fuera del contenedor de la base de datos para importar los datos en un nuevo contenedor.

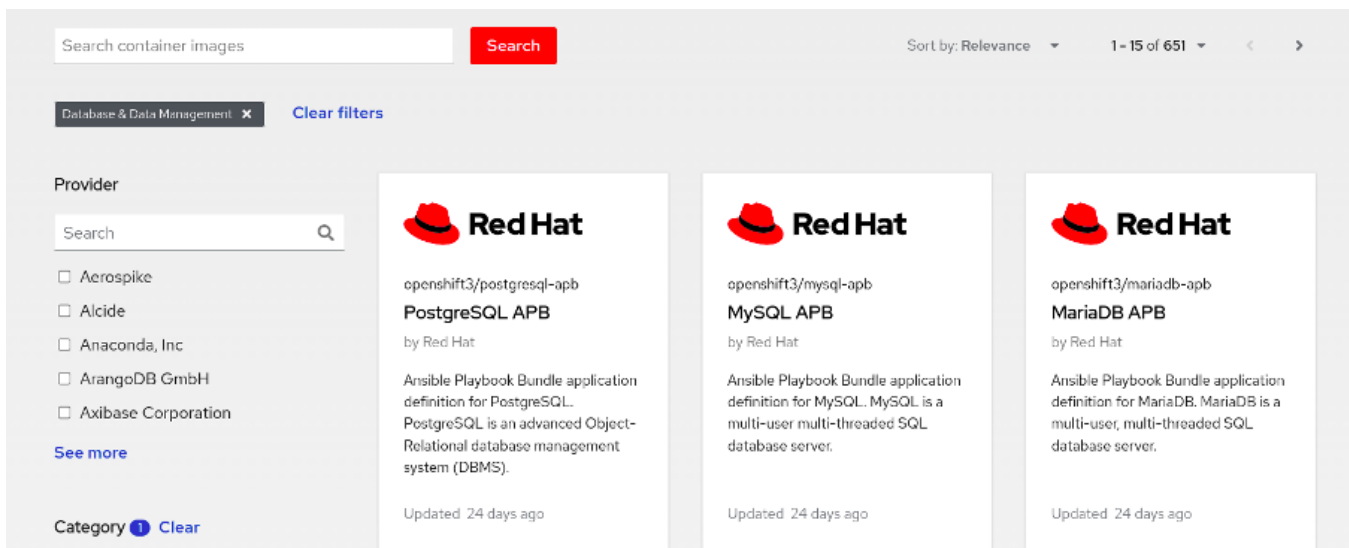
### Contenedores de bases de datos de Red Hat

Los registros en Red Hat Ecosystem Catalog contienen una lista de imágenes de contenedores de bases de datos compatibles con Red Hat.





Busque los productos de la base de datos contenerizados por Red Hat y proveedores externos seleccionando la categoría Database & Data Management.



Después de elegir el contenedor de la base de datos, extraiga la imagen del contenedor y siga las instrucciones de uso.

Por ejemplo, si selecciona la imagen `rhel8/mariadb-105`, puede extraer la imagen navegando a la pestaña `Get this image` y siguiendo las instrucciones en la sección `Using docker login`. Las instrucciones de uso están disponibles en la pestaña `Overview`

[Home](#) > [Software](#) > [Container images](#) > [Browse products](#) > Mariadb 10.5

Standalone image

# Mariadb 10.5

rhel8/mariadb-105

Provided  
by



Architecture amd64

Tag 1-63

Repository structure: Single-stream

♥ latest ♥ 1 ♥ 1-63

Overview

Security

Technical Information

Packages

Dockerfile

Get this image

## Description

This container image provides a containerized packaging of the MariaDB mysqld daemon and client application. The mysqld server daemon accepts connections from clients and provides access to content from MySQL databases on behalf of the clients. You can find more information on the MariaDB project from the project Web site (<https://mariadb.org/>).

### Published

17 days ago

### Release category

Generally Available ⓘ

## Ejercicio guiado: Trabajo con bases de datos

Configurar una base de datos PostgreSQL contenerizada para entornos de desarrollo y prueba.

### Resultados

Usted deberá ser capaz de realizar lo siguiente:

- Cree una base de datos contenerizada que contenga datos efímeros.
- Cargue el esquema de la base de datos y los datos sobre la creación del contenedor.
- Use el cliente de PostgreSQL en el contenedor de PostgreSQL para interactuar con la base de datos.
- Cree una base de datos persistente contenerizada para un entorno de prueba.
- Migre los datos del entorno de prueba a un contenedor que ejecute una versión más reciente de PostgreSQL.

Con el usuario `student` en la máquina `workstation`, use el comando `lab` para preparar el sistema para este ejercicio.

Este comando copia los scripts de inicialización de la base de datos de una tienda Raspberry Pi en el directorio de laboratorio de su sistema.

### Procedimiento 5.2. Instrucciones

1. Cree un contenedor de PostgreSQL 12 y llámelo `persisting-pg12`. Use el script `load_db.sh` en el directorio de laboratorio para cargar los datos.
  1. Examine el script `~/cdiputadosdocker/persisting-databases/load_db.sh`, que carga las definiciones del esquema de la base de datos y los datos en las tablas `model` y `inventory`.
  2. Cree un contenedor de PostgreSQL 12 que se ejecute de manera interactiva y llámelo `persisting-pg12`.

Use la imagen `quay.io/piracarter1/rhel8-postgresql-12:1-113`

Monte el directorio de host `~/cdiputadosdocker/persisting-databases/en` en el directorio del contenedor `/opt/app-root/src/postgresql-start` como un montaje de enlace con la opción `z` para SELinux.

Proporcione las siguientes variables de entorno:

- `POSTGRES_USER=backend`

- POSTGRESQL\_PASSWORD=secret\_pass
- POSTGRESQL\_DATABASE=rpi-store

```
[student@workstation ~]$ docker run -it --rm \
  --name persisting-pg12 \
  -e POSTGRESQL_USER=backend \
  -e POSTGRESQL_PASSWORD=secret_pass \
  -e POSTGRESQL_DATABASE=rpi-store \
  -v ~/cdiputadosdocker/persisting-databases:/opt/app-root/src/postgresql-start:Z \
  quay.io/piracarter1/rhel8-postgresql-12:1-113
...output omitted...
server started
/var/run/postgresql:5432 - accepting connections
=> sourcing /opt/app-root/src/postgresql-start/load_db.sh ...
...output omitted...
```

Al inicio, los contenedores basados en la imagen `rhel8/postgresql-12` ejecutan cualquier script que termine en `.sh`, que se encuentra en el directorio `/opt/app-root/src/postgresql-start`.

3. Abra un nuevo terminal y ejecute el cliente PostgreSQL `psql` en el contenedor para consultar la tabla `model`.

```
[student@workstation ~]$ docker exec -it persisting-pg12 \
  psql -d rpi-store -c "select * from model"
```

id	name	model	soc	memory_mb	ethernet	release_date
1	Raspberry Pi	B	BCM2835	256	t	2012
2	Raspberry Pi Zero	Zero	BCM2835	512	f	2015
3	Raspberry Pi Zero	2W	BCM2710A1	512	f	2021
4	Raspberry Pi 4	B	BCM2711	4096	t	2019
6	Raspberry Pi 4	400	BCM2711	4096	t	2020

(5 rows)

2. Vuelva a crear el contenedor anterior para verificar que los datos cargados en él no sean persistentes.

1. Presione **Ctrl+C** para salir del contenedor `persisting-pg12`. El contenedor se elimina automáticamente debido a la opción `--rm` proporcionada en la creación del contenedor.
2. Vuelva a crear el contenedor `persisting-pg12` sin cargar los scripts de la base de datos.

```
[student@workstation ~]$ docker run -it --rm \
--name persisting-pg12 \
-e POSTGRES_USER=backend \
-e POSTGRES_PASSWORD=secret_pass \
-e POSTGRES_DATABASE=rpi-store \
quay.io/piracarter1/rhel8-postgresql-12:1-113
```

*...output omitted...*

3. En la ventana de terminal anterior, verifique que los datos que cargó en el paso anterior no estén presentes consultando la tabla `model`.

```
[student@workstation ~]$ docker exec -it persisting-pg12 \
psql -d rpi-store -c "select * from model"
ERROR:  relation "model" does not exist
LINE 1: select * from model
```

3. Cree una base de datos de PostgreSQL contenerizada para un entorno de prueba. Use un volumen para evitar perder datos si se vuelve a crear el contenedor.

1. Presione **Ctrl+C** para salir del contenedor `persisting-pg12`. El contenedor se elimina automáticamente debido a la opción `--rm`.
2. Cree un volumen llamado `rpi-store-data` para almacenar los archivos de la base de datos.

```
[student@workstation ~]$ docker volume create rpi-store-data
rpi-store-data
```

3. Cree el contenedor `persisting-pg12` en modo separado y asigne el volumen `rpi-store-data` al directorio `/var/lib/pgsql/data` en el contenedor. Vincule el montaje del directorio `postgresql-start` como lo hizo anteriormente.

```
[student@workstation ~]$ docker run -d \
  --name persisting-pg12 \
  -e POSTGRESQL_USER=backend \
  -e POSTGRESQL_PASSWORD=secret_pass \
  -e POSTGRESQL_DATABASE=rpi-store \
  -v rpi-store-data:/var/lib/pgsql/data \
  -v ~/cdiputadosdocker/persisting-databases:/opt/app-root/src/postgresql-start
:Z \
  quay.io/piracarter1/rhel8-postgresql-12:1-113
c99b...7b7c
```

`/var/lib/pgsql/data` es el directorio donde se configura la imagen `rhel8/postgresql-12` para almacenar los archivos de la base de datos.

4. Verifique que la base de datos `rpi-store` contenga datos al consultar la tabla `model`:

```
[student@workstation ~]$ docker exec -it persisting-pg12 \
  psql -d rpi-store -c "select * from model"
...output omitted...
```

4. Vuelva a crear el contenedor para probar que los datos insertados en la base de datos `rpi-store` persisten después de la recreación del contenedor.

1. Elimine el contenedor denominado `persisting-pg12`. Proporcione la opción `-f` para detener y eliminar el contenedor con el mismo comando.

```
[student@workstation ~]$ docker rm -f persisting-pg12
c99b...7b7c
```

2. Vuelva a crear el contenedor `persisting-pg12` y conecte el volumen `rpi-store-data` sin vincular los scripts de carga de datos.

```
[student@workstation ~]$ docker run -d \
```

```
--name persisting-pg12 \  
-e POSTGRESQL_USER=backend \  
-e POSTGRESQL_PASSWORD=secret_pass \  
-e POSTGRESQL_DATABASE=rpi-store \  
-v rpi-store-data:/var/lib/pgsql/data \  
quay.io/piracarter1/rhel8-postgresql-12:1-113  
e37a...d0cb
```

3. Consulte la tabla `model` para verificar que se conservan los datos.

```
[student@workstation ~]$ docker exec -it persisting-pg12 \  
psql -d rpi-store -c "select * from model"  
...output omitted...  
(5 rows)
```

5. Inicie una interfaz pgAdmin contenerizada para gestionar la base de datos mediante una interfaz de usuario web.

1. Cree una red para permitir la resolución de nombres DNS entre pgAdmin y los contenedores de la base de datos.

```
[student@workstation ~]$ docker network create persisting-network  
persisting-network
```

2. Elimine el contenedor `persisting-pg12` existente.

```
[student@workstation ~]$ docker rm -f persisting-pg12  
e37a...d0cb
```

3. Vuelva a crear el contenedor `persisting-pg12`. El contenedor debe usar la red denominada `persisting-network`.

```
[student@workstation ~]$ docker run -d \  
--name persisting-pg12 \  
-e POSTGRESQL_USER=backend \  
-e POSTGRESQL_PASSWORD=secret_pass \  
-e POSTGRESQL_DATABASE=rpi-store \  
-v rpi-store-data:/var/lib/pgsql/data \  
--network persisting-network \  
quay.io/piracarter1/rhel8-postgresql-12:1-113
```

```
quay.io/piracarter1/rhel8-postgresql-12:1-113
```

```
ab8f...ce8c
```

4. Cree el contenedor pgAdmin y denomínelo `persisting-pgadmin`. Conecte el contenedor a `persisting-network` para resolver el contenedor de la base de datos con el nombre `persisting-pg12`.

Use la imagen `registry.ocp4.example.com:8443/crunchydata/crunchy-pgadmin4:ubi8-4.30-1`.

Asigne el puerto del contenedor `5050` al mismo puerto del host.

Use las siguientes variables de entorno para el contenedor:

- `PGADMIN_SETUP_EMAIL=gl@example.com`
- `PGADMIN_SETUP_PASSWORD=pga_secret_pass`

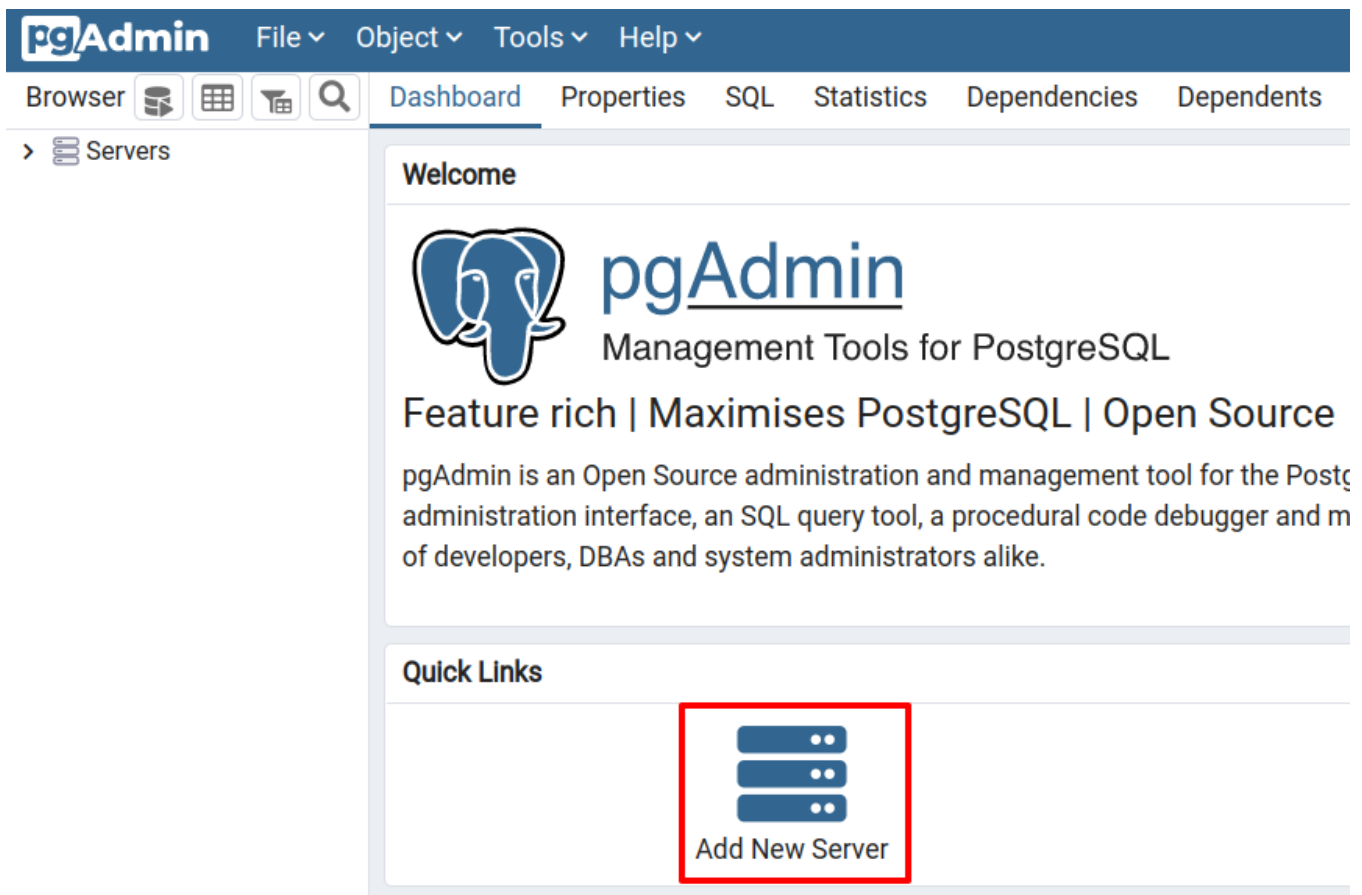
```
[student@workstation ~]$ docker run -d \
  --name persisting-pgadmin \
  -e PGADMIN_SETUP_EMAIL=gl@example.com \
  -e PGADMIN_SETUP_PASSWORD=pga_secret_pass \
  -p 5050:5050 \
  --network persisting-network \
  quay.io/piracarter1/crunchy-pgadmin4:ubi8-4.30-1
...output omitted...
cd9g...aa8f
```

## NOTA

Después de ejecutar el comando anterior, es posible que vea advertencias de SELinux. Es seguro ignorar esos errores.

5. Inicie sesión en pgAdmin mediante un explorador web y diríjase a `http://localhost:5050`. Inicie sesión con las credenciales que usó para iniciar el contenedor:
  - Dirección de correo electrónico/nombre de usuario: `gl@example.com`
  - Contraseña: `pga_secret_pass`
6. Haga clic en **Add New Server** (Agregar nuevo servidor) para conectarse al contenedor de la base de datos `persisting-pg12`.





En la pestaña General, defina rpi-store como el nombre.

Cambie a la pestaña Connection (Conexión). Complete el formulario con los siguientes datos y deje el resto de los campos con sus valores predeterminados.

Campo	Valor
Nombre de host/dirección	persisting-pg12
Nombre de usuario	backend
Contraseña	secret_pass

Haga clic en **Save** (Siguiente). La aplicación verifica la conexión antes de salir del formulario.

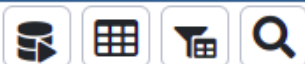
### NOTA

Debido a que la base de datos está escuchando en la máquina host, puede suponer que el valor localhost funciona para el campo Host name. Esto no funciona porque pgAdmin se ejecuta en un contenedor y localhost se refiere al propio contenedor pgAdmin.

Vea los datos en la tabla `model` para verificar el acceso de pgAdmin a la base de datos `rpi-store` en el contenedor `persisting-pg12`.

Seleccione la tabla `model` haciendo clic en **Servers > rpi-store > Databases > rpi-store > Schemas > public > Tables > model**

Browser



- ▼ Servers (1)
  - ▼ rpi-store
    - ▼ Databases (2)
      - > postgres
      - ▼ rpi-store
        - > Casts
        - > Catalogs (2)
        - > Event Triggers
        - > Extensions
        - > Foreign Data Wrappers
        - > Languages
        - ▼ Schemas (1)
          - ▼ public
            - > Collations
            - > Domains
            - > FTS Configurations
            - > FTS Dictionaries
            - > FTS Parsers
            - > FTS Templates
            - > Foreign Tables
            - > Functions
            - > Materialized Views
            - > Procedures
            - > 1.3 Sequences
            - ▼ Tables (2)
              - > inventory
              - > model

The screenshot shows a database management tool with a menu bar (Object, Tools, Help) and a toolbar. The main window is divided into three panes: Query Editor, Query History, and Scratch Pad. The Query Editor contains the following SQL query:

```
1 SELECT * FROM public.model
2 ORDER BY id ASC
```

Below the query editor is a table titled "Data Output" showing the results of the query. The table has the following columns: id [PK] Integer, name character (20), model character (20), soc character (20), memory\_mb Integer, ethernet boolean, and release\_year Integer. The data is as follows:

id	name	model	soc	memory_mb	ethernet	release_year
1	Raspberry Pi	B	BCM2835	256	true	2012
2	Raspberry Pi Zero	Zero	BCM2835	512	false	2015
3	Raspberry Pi Zero	2W	BCM2710A1	512	false	2021
4	Raspberry Pi 4	B	BCM2711	4096	true	2019
5	Raspberry Pi 4	400	BCM2711	4096	true	2020

6. Migre los datos del entorno de prueba a un contenedor que ejecute PostgreSQL 13, una versión más reciente de la base de datos PostgreSQL. Use un nuevo volumen de datos para evitar incompatibilidades entre las versiones de PostgreSQL.

1. Ejecute el comando `pg_dump` en el contenedor de la base de datos en ejecución para hacer una copia de seguridad de `rpi-store`. Un archivo llamado `/tmp/db_dump` almacena la copia de seguridad (backup) en el contenedor `persisting-pg12`. Use la opción `-Fc` para usar el formato custom que comprime el archivo de copia de seguridad (backup).

```
[student@workstation ~]$ docker exec persisting-pg12 \
    pg_dump -Fc rpi-store -f /tmp/db_dump
no output expected
```

2. Use el comando `docker cp` para copiar el archivo de volcado en la máquina host.

```
[student@workstation ~]$ docker cp persisting-pg12:/tmp/db_dump /tmp/db_dump
no output expected
```

3. Detenga el contenedor `persisting-pg12`.

```
[student@workstation ~]$ docker stop persisting-pg12
```

```
persisting-pg12
```

### NOTA

Es posible que vea una advertencia `stopSignal` cuando detenga el contenedor. Puede ignorar esta advertencia con seguridad.

4. Cree un volumen llamado `rpi-store-data-pg13` para almacenar los archivos de la base de datos.

```
[student@workstation ~]$ docker volume create rpi-store-data-pg13  
rpi-store-data-pg13
```

5. Cree un contenedor PostgreSQL 13 que use el volumen `rpi-store-data-pg13`.

```
[student@workstation ~]$ docker run -d \  
  --name persisting-pg13 \  
  -e POSTGRES_USER=backend \  
  -e POSTGRES_PASSWORD=secret_pass \  
  -e POSTGRES_DATABASE=rpi-store \  
  -v rpi-store-data-pg13:/var/lib/pgsql/data \  
  quay.io/piracarter1/rhel9-postgresql-13:1  
00e9...36a8
```

6. Copie el archivo de volcado en el nuevo contenedor con el comando `docker cp`.

```
[student@workstation ~]$ docker cp /tmp/db_dump persisting-pg13:/tmp/db_dump  
p  
no output expected
```

7. Ejecute `pg_restore` para restaurar la base de datos `rpi-store` en el contenedor `persisting-pg13`. Establezca `rpi-store` como base de datos de destino mediante la opción `-d`.

```
[student@workstation ~]$ docker exec persisting-pg13 \  
  pg_restore -d rpi-store /tmp/db_dump  
no output expected
```

8. Consulte la tabla `model` en el contenedor `persisting-pg13` para comprobar que la migración funcionó.

```
[student@workstation ~]$ docker exec -it persisting-pg13 \
    psql -d rpi-store -c "select * from model"
...output omitted...
(5 rows)
```

9. Elimine el contenedor con la versión anterior de PostgreSQL y su volumen asociado para eliminar los recursos no usados.

```
[student@workstation ~]$ docker rm persisting-pg12
ab8f...ce8c
```

Debido a que los volúmenes con nombre persisten después de eliminar el contenedor, debe eliminar el volumen `rpi-store-data` manualmente.

```
[student@workstation ~]$ docker volume rm rpi-store-data
rpi-store-data
```

## Capítulo 6. Aplicaciones con múltiples contenedores con Compose

### Descripción general y casos de uso de Compose

#### Objetivos

- Describir Compose y sus casos de uso comunes.

#### Orqueste contenedores con Docker Compose

En la computación actual, muchas aplicaciones se desarrollan como microservicios. En el contexto de los contenedores, cada microservicio es una imagen de contenedor, que los desarrolladores gestionan como un contenedor independiente. Debido a que una aplicación se compone de varios contenedores, puede resultar difícil gestionar los contenedores como un grupo.

Por ejemplo, para desarrollar un nuevo microservicio, los desarrolladores suelen implementar una de las siguientes estrategias:

- Cree una *simulación* del resto de la aplicación, como los extremos de la API que proporcionan otros microservicios.
- Inicie las partes relevantes de la aplicación en una máquina local.

Es ideal para ejecutar la aplicación completa en la máquina local. Cuando no es posible hacerlo, los desarrolladores crean contenedores de servicios simulados que se aproximan al entorno de producción.

*Docker Compose* es una herramienta de código abierto que puede usar para ejecutar *archivos Compose*. Un archivo Compose es un archivo YAML que especifica los contenedores que se deben gestionar, así como las dependencias entre ellos.

Por ejemplo, considere el siguiente archivo Compose:

```
services:
  orders:
    image: quay.io/user/python-app
    ports:
      - 3030:8080
    environment:
      ACCOUNTS_SERVICE: http://accounts
```

Declare el contenedor `orders`.

Use la imagen de contenedor `python-app`.

Vincule el puerto 3030 en su máquina host al puerto 8080 en el contenedor.

Pase la variable de entorno `ACCOUNTS_SERVICE` a la aplicación.

Docker Compose cumple con la *especificación Compose*, que define un esquema basado en YAML para gestionar aplicaciones de varios contenedores para tiempos de ejecución, como Docker.

Aunque los archivos de composición deberían funcionar con cualquier tiempo de ejecución de contenedor, algunas características pueden depender de implementaciones específicas del tiempo de ejecución. Puede encontrar archivos de Compose denominados `docker-compose.yaml`. Docker introdujo la convención de nombres `docker-compose.yaml` cuando inició el proyecto Compose. Sin embargo, Docker aplicó el código abierto de la especificación Compose, que también especifica la convención de nombres de archivos Compose. Según la especificación de Compose, el nombre preferido es `compose.yaml`.

Docker Compose se hizo popular para los siguientes usos:

### **Entornos de desarrollo**

Puede aproximar un entorno de producción en su máquina local al proporcionar aplicaciones, bases de datos o configuración de sistemas de caché en un solo archivo.

En consecuencia, los desarrolladores pueden automatizar implementaciones complejas de varios contenedores con un solo archivo.

### **Pruebas automatizadas**

Puede definir varios conjuntos de pruebas, junto con sus configuraciones y variables de entorno, dentro del mismo archivo. Además, puede aproximar un entorno de tubería (pipeline) automatizado en una sola máquina.

En consecuencia, los desarrolladores pueden ejecutar sus aplicaciones con múltiples clientes y entornos de bases de datos aislados. Los desarrolladores también pueden depurar problemas complejos en su entorno local, lo que aumenta la velocidad de desarrollo.

No se recomienda usar Docker Compose en un entorno de producción. Docker Compose no soporta funciones avanzadas que pueda necesitar en un entorno de producción, como el balanceo de carga, la distribución de contenedores a varios nodos, la gestión de contenedores en diferentes nodos y otros.



Si necesita una solución de orquestación de contenedores de producción, Kubernetes o Red Hat OpenShift es una mejor opción. Red Hat OpenShift puede ejecutar aplicaciones de varios contenedores en diferentes máquinas host o *nodos*.

## El archivo Compose

El archivo Compose es un archivo YAML que contiene las siguientes secciones:

- `version` (en desuso): especifica la versión de Compose usada.
- `services`: define los contenedores usados.
- `networks`: define las redes usadas por los contenedores.
- `volumes`: especifica los volúmenes usados por los contenedores.
- `configs`: especifica las configuraciones usadas por los contenedores.
- `secrets`: define los secretos usados por los contenedores.

Los objetos `secrets` y `configs` se montan como un archivo en los contenedores.

### AVISO

En los archivos YAML, la cantidad de espacios tiene un significado.

Por ejemplo, considere el siguiente fragmento YAML:

```
version: "3.9"
services:
backend: {}
```

El fragmento de código YAML anterior tiene un significado diferente del siguiente fragmento:

```
version: "3.9"
services:
  backend: {}
```

La palabra clave `backend` está precedida por dos espacios; por lo tanto, el objeto `backend` es un atributo del objeto `services`.

El siguiente archivo Compose define los servicios `backend` y `db`. También anula el comando predeterminado para la imagen de back end al especificar la propiedad `command`. Por último, el archivo Compose configura las variables de entorno necesarias para iniciar el contenedor de la base de datos.

## NOTA

Esta sección usa los términos *servicio* y *contenedor* indistintamente.

```
services:
  backend:
    image: quay.io/example/backend
    ports:
      - "8081:8080"
    command: sh -c "COMMAND"
  db:
    image: registry.redhat.io/rhel8/postgresql-13
    environment:
      POSTGRESQL_ADMIN_PASSWORD: redhat
```

### **NOTA:** *INSTALACION DOCKER COMPOSE*

[https://docs.docker.com/compose/install/linux/#~:text=Install%20the%20plugin%20manually%20link%201%20To%20download,%2Bx%20%24DOCKER\\_CONFIG%2Fcli-plugins%2Fdocker-compose%20...%203%20Test%20the%20installation.%20](https://docs.docker.com/compose/install/linux/#~:text=Install%20the%20plugin%20manually%20link%201%20To%20download,%2Bx%20%24DOCKER_CONFIG%2Fcli-plugins%2Fdocker-compose%20...%203%20Test%20the%20installation.%20)

### *Iniciar y detener contenedores con Docker Compose*

Puede ejecutar un archivo Compose mediante el comando `docker-compose up`, que crea los objetos definidos, como volúmenes o redes, e inicia los contenedores definidos.

```
[user@host ~]$ docker-compose up
['docker', '--version', '']
using docker version: 4.2.0
** excluding: set()

['docker', 'network', 'exists', 'user_default']
['docker', 'network', 'create', '--label', 'io.docker.compose.project=user', '--label', 'com.docker.compose.project=user', 'user_default']
...output omitted...
```

Compruebe si existe la red `user_default`.

Cree la red Docker `user_default`.

Puede enumerar los objetos que están definidos en el archivo Compose usando el comando `docker`, como `docker network ls`, para enumerar las redes Docker.

```
[user@host ~]$ docker network ls
```

NETWORK ID	NAME	VERSION	PLUGINS
cd32c76f42f9	<b>user_default</b>	0.4.0	bridge,portmap,firewall,tuning,dnsname

Docker genera nombres predecibles para los objetos que no se nombran explícitamente. En el ejemplo anterior, Docker crea una red predeterminada para la aplicación.

La convención de nomenclatura de red predeterminada usa el nombre del directorio actual y el sufijo `_default`. El ejemplo anterior contiene la red `user_default` porque muestra un archivo Compose ejecutado en el directorio de inicio `/home/user`.

La convención de nomenclatura de contenedores predeterminada usa el formato `DIRECTORY_SERVICE_NUMBER`. Dado que el ejemplo anterior no configura el nombre del contenedor, crea el contenedor `user_db_1`.

Puede anular las convenciones de nomenclatura predeterminadas al configurar nombres de objetos explícitamente.

El comando `docker-compose stop` detiene la ejecución de contenedores definidos como servicios. Ejecute `docker-compose down` para detener y eliminar contenedores que se definen como servicios.

```
[user@host ~]$ docker-compose down
['docker', '--version', '']
using docker version: 4.2.0
** excluding: set()
docker stop -t 10 compose_db_1
compose_db_1
exit code: 0
docker rm compose_db_1
aab3...b412
exit code: 0
...output omitted...
```

Docker conserva objetos que no sean contenedores, como redes y volúmenes, para que los contenedores no pierdan su estado local cuando se reinician.

## Redes

Use la palabra clave `networks` en el mismo nivel de sangría que la palabra clave `services` para crear y usar redes Docker. Si la palabra clave `networks` no está definida en el archivo Docker Compose, Docker Compose crea una red predeterminada con DNS habilitado.

Considere el siguiente archivo Compose que declara tres contenedores: una aplicación front end, una aplicación back end y una base de datos.

```
services:
  frontend:
    image: quay.io/example/frontend

    networks:
      - app-net
    ports:
      - "8082:8080"
  backend:
    image: quay.io/example/backend

    networks:
      - app-net
      - db-net
  db:
    image: registry.redhat.io/rhel8/postgresql-13
    environment:
      POSTGRESQL_ADMIN_PASSWORD: redhat

    networks:
      - db-net

networks:
  app-net: {}
```

```
db-net: {}
```

El servicio frontend es parte de la red app-net.

El servicio backend es parte de las redes app-net y db-net.

El servicio db es parte de db-net.

Definición de las redes.

### NOTA

Las llaves abiertas y cerradas ({} ) marcan un objeto vacío. Por ejemplo, las definiciones `app-net: {}` y `app-net:` son idénticas. Ambas definiciones señalan la creación de la red app-net que usa la configuración predeterminada.

Los servicios solo pueden interactuar si comparten al menos una red. En consecuencia, el servicio frontend no puede comunicarse directamente con el servicio db. El aislamiento del tráfico de red proporciona ventajas de seguridad, ya que puede evitar que contenedores específicos accedan a datos protegidos.

### Volúmenes

Puede declarar volúmenes con la palabra clave `volumes`. Monte los volúmenes en contenedores con la sintaxis `VOLUME_NAME:CONTAINER_DIRECTORY:OPTIONS`.

```
services:
  db:
    image: registry.redhat.io/rhel8/postgresql-13
    environment:
      POSTGRESQL_ADMIN_PASSWORD: redhat
    ports:
      - "5432:5432"

    volumes:
      - db-vol:/var/lib/postgresql/data

volumes:
  db-vol: {}
```

El servicio db asigna el volumen db-vol al directorio /var/lib/postgresql/data dentro del contenedor.

Declaración de volúmenes.

Si creó un volumen que no está administrado por Docker Compose, por ejemplo, porque usó el comando `docker volume create`, puede especificarlo en la definición de volúmenes.

```
services:
  db:
    image: registry.redhat.io/rhel8/postgresql-13
    environment:
      POSTGRESQL_ADMIN_PASSWORD: redhat
    ports:
      - "5432:5432"
    volumes:
      - my-volume:/var/lib/postgresql/data

volumes:
  my-volume:
    external: true
```

También puede definir montajes de enlace proporcionando una ruta relativa o absoluta en su máquina host. En el siguiente ejemplo, Docker monta el directorio `./local/redhat` en la máquina host y el directorio `/var/lib/postgresql/data` en el contenedor.

```
services:
  db:
    image: registry.redhat.io/rhel8/postgresql-13
    environment:
      POSTGRESQL_ADMIN_PASSWORD: redhat
    ports:
      - "5432:5432"
    volumes:
      - ./local/redhat:/var/lib/postgresql/data:Z
```



## Crear entornos de desarrollador con Compose

### Objetivos

- Configurar un entorno de desarrollador repetible con Compose.

### Docker Compose y Docker

Docker Compose se comunica directamente con el daemon de Docker mediante su API REST.

### Entornos de desarrollador de varios contenedores con Compose

Con Docker Compose puede configurar de forma declarativa los servicios que necesita para desarrollar su aplicación. Puede colocar los servicios requeridos en el archivo Compose dentro de su repositorio de aplicaciones para que los desarrolladores puedan iniciar estos servicios emitiendo un solo comando `docker-compose up`. Ejecutar las dependencias de su entorno de desarrollo de esta manera aísla al desarrollador de los entornos de desarrollador compartidos. Además, esta configuración predefinida evita que los desarrolladores tengan que configurar localmente servicios como bases de datos o dependencias externas.

Por ejemplo, para desarrollar una aplicación de back end, puede usar Docker Compose y un solo archivo Compose para implementar un entorno de desarrollo que incluye un contenedor de base de datos PostgreSQL y un contenedor de interfaz pgAdmin para gestionar la base de datos. Puede montar scripts de carga de datos para la base de datos para proporcionar a los desarrolladores los datos de desarrollo necesarios. Puede especificar las dependencias del servicio usando la propiedad `depends_on` seguida de una lista de servicios que deben iniciarse primero.

```
services:
  database:
    image: "registry.redhat.io/rhel9/postgresql-13"
    container_name: "appdev-postgresql"
    volumes:
      - ./scripts/database/initial-data:/opt/app-root/src/postgresql-start:Z
  ...output omitted...
  database-admin:
    image: "registry.connect.redhat.com/crunchydata/crunchy-pgadmin4:ubi8-4.30-1"
    container_name: "appdev-pgadmin"
    depends_on:
      - database
```



```
...output omitted...
```

El ejemplo anterior demuestra que el contenedor `database-admin` debe comenzar después del contenedor `database`.

Si su aplicación depende de servicios externos, puede agregar un servidor simulado a su archivo `Compose`. Puede proporcionar los extremos del servidor simulado y las respuestas fijas como archivos de configuración en su repositorio. Luego, puede enlazar el montaje de estos archivos con rutas relativas al directorio de su proyecto dentro de su archivo `Compose`.

```
services:
...output omitted...
  mock_service:
    image: "MOCK_SERVICE_IMAGE"
    volumes:
      - ./mocks/SERVICE/ENDPOINTS:TARGET_DIRECTORY:Z
      - ./mocks/SERVICE/FIXED_RESPONSES:TARGET_DIRECTORY:Z
...output omitted...
```

## Ejercicio guiado: Crear entornos de desarrollador con Compose

Configure un entorno de desarrollador repetible con Docker Compose.

### Resultados

Usted deberá ser capaz de realizar lo siguiente:

- Cree un archivo Compose que contenga la definición de un servidor PostgreSQL y una interfaz pgAdmin.
- Cree un archivo Compose que contenga la definición de una interfaz pgAdmin.
- Inicie y ejecute el entorno de desarrollador.
- Acceda a la interfaz pgAdmin desde un explorador web para recuperar los datos de las tablas.

Con el usuario `student` en la máquina `workstation`, use el comando `lab` para preparar el sistema para este ejercicio.

### Procedimiento 7.1. Instrucciones

6. Cree un archivo Compose que contenga la definición de un servidor PostgreSQL.

1. Vaya al directorio `cdiputadosdocker/compose-environments/` y abra el archivo `compose.yml`.

```
[student@workstation compose-environments]$ gedit compose.yml
```

2. Defina un contenedor de base de datos que use la imagen `quay.io/piracarter1/rhel9-postgresql-13:1`. Reenvíe el puerto 5432 del localhost al mismo puerto dentro del contenedor.

```
services:
  db:
    image: "quay.io/piracarter1/rhel9-postgresql-13:1"
    ports:
      - "5432:5432"
```

3. Defina las siguientes variables de entorno:

Campo	Valor
POSTGRESQL_USER	backend
POSTGRESQL_DATABASE	rpi-store

Campo	Valor
POSTGRESQL_PASSWORD	redhat

```
services:
  db:
    image: "registry.ocp4.example.com:8443/rhel9/postgresql-13:1"
    environment:
      POSTGRESQL_USER: backend
      POSTGRESQL_DATABASE: rpi-store
      POSTGRESQL_PASSWORD: redhat
    ports:
      - "5432:5432"
```

4. Realice un montaje de enlace del directorio ~/D0188/labs/compose-environments/database\_scripts al directorio /opt/app-root/src/postgresql-start con la opción z para SELinux. Puede usar la ruta relativa al archivo compose.yml para el directorio database\_scripts como montaje de enlace.

```
services:
  db:
    image: "registry.ocp4.example.com:8443/rhel9/postgresql-13:1"
    environment:
      POSTGRESQL_USER: backend
      POSTGRESQL_DATABASE: rpi-store
      POSTGRESQL_PASSWORD: redhat
    ports:
      - "5432:5432"
volumes:
  ./database_scripts:/opt/app-root/src/postgresql-start:Z
```

5. Defina un volumen persistente denominado rpi para el contenedor. Realice un montaje de enlace del volumen rpi al directorio /var/lib/pgsql/data en el contenedor.

```
services:
  db:
```

```

image: "registry.ocp4.example.com:8443/rhel9/postgresql-13:1"
environment:
  POSTGRESQL_USER: backend
  POSTGRESQL_DATABASE: rpi-store
  POSTGRESQL_PASSWORD: redhat
ports:
  - "5432:5432"
volumes:
  - ./database_scripts:/opt/app-root/src/postgresql-start:Z
  - rpi:/var/lib/pgsql/data

volumes:
  rpi: {}

```

6. Llame al contenedor `compose_environments_postgresql` y guarde el archivo.

```

services:
  db:
    image: "registry.ocp4.example.com:8443/rhel9/postgresql-13:1"
    container_name: "compose_environments_postgresql"
    environment:
      POSTGRESQL_USER: backend
      POSTGRESQL_DATABASE: rpi-store
      POSTGRESQL_PASSWORD: redhat
    ports:
      - "5432:5432"
    volumes:
      - ./database_scripts:/opt/app-root/src/postgresql-start:Z
      - rpi:/var/lib/pgsql/data
volumes:

```

```

rpi: {}

```

7. Defina un servidor pgAdmin en el archivo `compose.yml`.

1. Defina un contenedor de interfaz de administración de base de datos que use la imagen `registry.ocp4.example.com:8443/crunchydata/crunchy-pgadmin4:ubi8-4.30-1`. Asigne el puerto `5050` desde el contenedor al puerto `5050` en el host.

```
services:
  db-admin:
    image: "quay.io/piracarter1/crunchy-pgadmin4:ubi8-4.30-1"
    ports:
      - "5050:5050"
  db:
    image: "registry.ocp4.example.com:8443/rhel9/postgresql-13:1"
    container_name: "compose_environments_postgresql"
    environment:
      POSTGRESQL_USER: backend
      POSTGRESQL_DATABASE: rpi-store
      POSTGRESQL_PASSWORD: redhat
    ports:
      - "5432:5432"
    volumes:
      - ./database_scripts:/opt/app-root/src/postgresql-start:Z
      - rpi:/var/lib/pgsql/data

volumes:
```

```
rpi: {}
```

2. Defina las siguientes variables de entorno:

Campo	Valor
PGADMIN_SETUP_EMAIL	user@redhat.com
PGADMIN_SETUP_PASSWORD	redhat

```
services:
  db-admin:
```

```

    image: "registry.ocp4.example.com:8443/crunchydata/crunchy-pgadmin4:ubi8-4.30-1"

    environment:
        PGADMIN_SETUP_EMAIL: user@redhat.com
        PGADMIN_SETUP_PASSWORD: redhat

    ports:
        - "5050:5050"

    db:
        image: "registry.ocp4.example.com:8443/rhel9/postgresql-13:1"
        container_name: "compose_environments_postgresql"
        environment:
            POSTGRES_USER: backend
            POSTGRES_DATABASE: rpi-store
            POSTGRES_PASSWORD: redhat
        ports:
            - "5432:5432"
        volumes:
            - ./database_scripts:/opt/app-root/src/postgresql-start:Z
            - rpi:/var/lib/pgsql/data

    volumes:
        rpi: {}

```

3. Designarle un nombre al contenedor `compose_environments_pgadmin`. Guarde y cierre el archivo.

```

services:
    db-admin:
        image: "registry.ocp4.example.com:8443/crunchydata/crunchy-pgadmin4:ubi8-4.30-1"
        container_name: "compose_environments_pgadmin"
        environment:
            PGADMIN_SETUP_EMAIL: user@redhat.com
            PGADMIN_SETUP_PASSWORD: redhat
        ports:
            - "5050:5050"

```

```

db:
  image: "registry.ocp4.example.com:8443/rhel9/postgresql-13:1"
  container_name: "compose_environments_postgresql"
  environment:
    POSTGRESQL_USER: backend
    POSTGRESQL_DATABASE: rpi-store
    POSTGRESQL_PASSWORD: redhat
  ports:
    - "5432:5432"
  volumes:
    - ./database_scripts:/opt/app-root/src/postgresql-start:Z
    - rpi:/var/lib/pgsql/data

volumes:
  rpi: {}

```

## NOTA

Puede consultar el archivo `compose.yml` completo en el directorio `~/D0188/solutions/compose-environments`.

### 8. Ejecute el entorno de desarrollador.

- Desde el directorio, use el archivo `compose.yml` para iniciar el entorno de desarrollo contenerizado. Use la opción `-d` para ejecutar los contenedores en segundo plano.

```

[student@workstation compose-environments]$ /usr/libexec/docker/cli-plugins/docker-compose up -d
[+] Running 3/3
 ✓ Network compose-environments_default      Created
0.3s
 ✓ Container compose_environments_postgresql Started
1.2s
 ✓ Container compose_environments_pgadmin    Started

```

Confirme que se estén ejecutando los dos contenedores.

```

[student@workstation compose-environments]$ /usr/libexec/docker/cli-plugins/docker-compose ps

```

NAME COMMAND PORTS	SERVICE	IMAGE	CREATED	STATUS
compose_environments_pgadmin 0-1 "opt/crunchy/bin/uid..." conds 0.0.0.0:5050->5050/tcp, :::5050->5050/tcp	db-admin	quay.io/piracarter1/crunchy-pgadmin4:ubi8-4.3	57 seconds ago	Up 56 se
compose_environments_postgresql "container-entrypoin..." 0.0.0.0:5432->5432/tcp, :::5432->5432/tcp	db	quay.io/piracarter1/rhel9-postgresql-13:1	57 seconds ago	Up 56 seconds

2. Confirme que existen los volúmenes persistentes.

```
[student@workstation compose-environments]$ docker volume list
```

DRIVER	VOLUME NAME
local	html-vol
local	http-data
local	rpi-store-data-pg13 local f056...7eb0

### NOTA

El comando puede mostrar volúmenes adicionales de ejercicios anteriores.

3. Recupere los registros de ambos contenedores y confirme que no se informan errores en los registros. Presione **Ctrl+C** para salir de los registros.

```
[student@workstation compose-environments]$ /usr/libexec/docker/cli-plugins/docker-compose logs -n -f
```

```
...output omitted...

compose_environments_postgresql Starting server...

compose_environments_postgresql 2022-08-11 14:42:02.237 UTC [1] LOG:  redirecti
ng log output to logging collector process

compose_environments_postgresql 2022-08-11 14:42:02.237 UTC [1] HINT:  Future l
og output will appear in directory "log".

...output omitted...

compose_environments_pgadmin Thu Aug 11 14:41:57 UTC 2022 INFO: Setting up pgAd
min4 database..

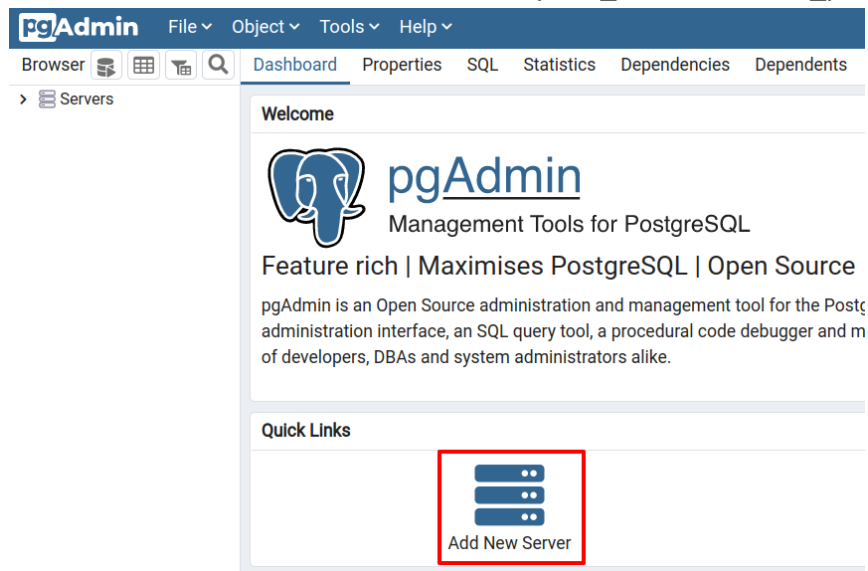
compose_environments_pgadmin Thu Aug 11 14:42:05 UTC 2022 INFO: Starting Apache
web server..
```



9. Acceda a la interfaz pgAdmin desde un explorador web. Recupere y modifique datos de la base de datos.
  1. Abra un explorador web y diríjase a <http://localhost:5050>. Acceda a la interfaz pgAdmin con el usuario `user@redhat.com` con la contraseña `redhat`.



2. Haga clic en **Add New Server** (Agregar nuevo servidor) para conectarse al contenedor de la base de datos `compose_environments_postgresql`.



3. En la pestaña General, defina `rpi-store` como el nombre.
4. Cambie a la pestaña Connection (Conexión). Complete el formulario con los siguientes datos y deje el resto de los campos con sus valores predeterminados.

Campo	Valor
Nombre/dirección de host	db
Nombre de usuario	backend
Contraseña	redhat

- Haga clic en **Save** (Siguiente). La aplicación verifica la conexión antes de salir del formulario.
- Navegue hasta **Servers** → **rpi-store** → **Databases** → **rpi-store** y, luego, seleccione **Tools** → **Query Tool** del menú. En **Query Editor** (Editor de consultas), ingrese la siguiente consulta.

```
select * from inventory
```

- Presione **F5** para ejecutar la consulta y recuperar datos de la tabla inventory.

The screenshot shows the pgAdmin interface. On the left, the 'Servers' tree is expanded to 'rpi-store' > 'Databases' > 'rpi-store'. The 'Query Editor' is active, showing the query 'select \* from inventory'. Below the editor, the 'Data Output' tab is selected, displaying a table with 4 rows and 5 columns: id [PK] integer, model\_id integer, quantity integer, and two unnamed columns. The data is as follows:

id [PK] integer	model_id integer	quantity integer		
1	1	1	0	
2	2	2	20	
3	3	3	300	
4	4	4	40	

Desde su terminal, detenga el entorno de desarrollo.

```
[student@workstation compose-environments]$ /usr/libexec/docker/cli-plugins/docker-compose
down

[+] Running 3/3
 ✓ Container compose_environments_pgadmin    Removed
0.5s
 ✓ Container compose_environments_postgresql  Removed
0.6s
```

✓ Network compose-environments_default 0.2s	Removed
--	---------