EBook Gratis

APRENDIZAJE Docker

Free unaffiliated eBook created from
Stack Overflow contributors.

#docker

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: docker

It is an unofficial and free Docker ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Docker.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Docker

Observaciones

Docker es un proyecto de código abierto que automatiza la implementación de aplicaciones dentro de los contenedores de software. Estos contenedores de aplicaciones son similares a las máquinas virtuales ligeras, ya que se pueden ejecutar de forma aislada entre sí y con el host en ejecución.

Docker requiere que las funciones presentes en los kernels de linux recientes funcionen correctamente, por lo tanto, en Mac OSX y Windows host se requiere una máquina virtual que ejecute linux para que la ventana acoplable funcione correctamente. Actualmente, el método principal de instalación y configuración de esta máquina virtual es a través de Docker Toolbox que usa VirtualBox internamente, pero hay planes para integrar esta funcionalidad en la propia ventana acoplable, utilizando las características de virtualización nativas del sistema operativo. En los sistemas Linux, la ventana acoplable se ejecuta de forma nativa en el propio host.

Versiones

Versión	Fecha de lanzamiento
17.05.0	2017-05-04
17.04.0	2017-04-05
17.03.0	2017-03-01
1.13.1	2016-02-08
1.12.0	2016-07-28
1.11.2	2016-04-13
1.10.3	2016-02-04
1.9.1	2015-11-03
1.8.3	2015-08-11
1.7.1	2015-06-16
1.6.2	2015-04-07
1.5.0	2015-02-10

Examples

Instalación de Docker en Mac OS X

Requisitos: OS X 10.8 "Mountain Lion" o más reciente requerido para ejecutar Docker.

Si bien el binario de la ventana acoplable puede ejecutarse de forma nativa en Mac OS X, para construir y alojar contenedores, necesita ejecutar una máquina virtual Linux en la caja.

1.12.0

Desde la versión 1.12, no es necesario tener una VM separada para instalarla, ya que Docker puede usar la funcionalidad `Hypervisor.framework` nativa de OSX para iniciar una pequeña máquina Linux que funcione como backend.

Para instalar la ventana acoplable siga los siguientes pasos:

1. Ir a Docker para Mac
2. Descarga y ejecuta el instalador.
3. Continúe con el instalador con las opciones predeterminadas e ingrese las credenciales de su cuenta cuando se le solicite.

Consulte aquí para más información sobre la instalación.

1.11.2

Hasta la versión 1.11, la mejor manera de ejecutar esta máquina virtual de Linux es instalar Docker Toolbox, que instala Docker, VirtualBox y la máquina invitada de Linux.

Para instalar la caja de herramientas de la ventana acoplable, siga los siguientes pasos:

1. Ir a Docker Toolbox
2. Haga clic en el enlace para Mac y ejecute el instalador.
3. Continúe con el instalador con las opciones predeterminadas e ingrese las credenciales de su cuenta cuando se le solicite.

Esto instalará los archivos binarios de Docker en `/usr/local/bin` y actualizará cualquier instalación de Virtual Box existente. Consulte aquí para más información sobre la instalación.

Para verificar la instalación:

1.12.0

1. Inicie `Docker.app` desde la carpeta Aplicaciones y asegúrese de que se está ejecutando. Siguiente abrir Terminal.

1.11.2

1. Abra el `Docker Quickstart Terminal` , que abrirá un terminal y lo preparará para su uso para los comandos de Docker.

2. Una vez que el terminal está abierto escribe

```
$ docker run hello-world
```

3. Si todo está bien, esto debería imprimir un mensaje de bienvenida que verifique que la instalación se realizó correctamente.

Instalación de Docker en Windows

Requisitos: la versión de 64 bits de Windows 7 o superior en una máquina que admita la tecnología de virtualización de hardware, y está habilitada.

Si bien el binario de la ventana acoplable puede ejecutarse de forma nativa en Windows, para construir y alojar contenedores, necesita ejecutar una máquina virtual de Linux en la caja.

1.12.0

Desde la versión 1.12, no es necesario tener una VM separada para instalarla, ya que Docker puede usar la funcionalidad Hyper-V nativa de Windows para iniciar una pequeña máquina Linux que funcione como backend.

Para instalar la ventana acoplable siga los siguientes pasos:

1. Ir a Docker para Windows
2. Descarga y ejecuta el instalador.
3. Continúe con el instalador con las opciones predeterminadas e ingrese las credenciales de su cuenta cuando se le solicite.

Consulte aquí para más información sobre la instalación.

1.11.2

Hasta la versión 1.11, la mejor manera de ejecutar esta máquina virtual de Linux es instalar Docker Toolbox, que instala Docker, VirtualBox y la máquina invitada de Linux.

Para instalar la caja de herramientas de la ventana acoplable, siga los siguientes pasos:

1. Ir a Docker Toolbox
2. Haga clic en el enlace para Windows y ejecute el instalador.
3. Continúe con el instalador con las opciones predeterminadas e ingrese las credenciales de su cuenta cuando se le solicite.

Esto instalará los binarios de Docker en Archivos de programa y actualizará cualquier instalación de Virtual Box existente. Consulte aquí para más información sobre la instalación.

Para verificar la instalación:

1.12.0

1. Inicie `Docker` desde el menú de inicio si aún no se ha iniciado, y asegúrese de que se está

ejecutando. A continuación, actualice cualquier terminal (ya sea `cmd` o PowerShell)

1.11.2

1. En tu escritorio, encuentra el ícono de Docker Toolbox. Haga clic en el icono para iniciar un terminal Docker Toolbox.

2. Una vez que el terminal está abierto escribe

```
docker run hello-world
```

3. Si todo está bien, esto debería imprimir un mensaje de bienvenida que verifique que la instalación se realizó correctamente.

Instalación de docker en Ubuntu Linux

Docker es compatible con las siguientes versiones de *64 bits* de Ubuntu Linux:

- Ubuntu Xenial 16.04 (LTS)
- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

Un par de notas:

> Las siguientes instrucciones implican la instalación utilizando solo paquetes de **Docker** , y esto garantiza la obtención de la última versión oficial de **Docker** . Si necesita instalar solo utilizando paquetes `Ubuntu-managed` , consulte la documentación de Ubuntu (No se recomienda de otra manera por razones obvias).
>
> Ubuntu Utopic 14.10 y 15.04 existen en el repositorio APT de Docker, pero ya no se admiten oficialmente debido a problemas de seguridad conocidos.

Prerrequisitos

- Docker solo funciona en una instalación de Linux de 64 bits.
- Docker requiere la versión 3.10 o superior del kernel de Linux (excepto para `Ubuntu Precise 12.04` , que requiere la versión 3.13 o superior). Los núcleos anteriores a 3.10 carecen de algunas de las funciones necesarias para ejecutar los contenedores de Docker y contienen errores conocidos que causan la pérdida de datos y con frecuencia entran en pánico bajo ciertas condiciones. Verifique la versión actual del kernel con el comando `uname -r` . Verifique esta publicación si necesita actualizar su kernel de `Ubuntu Precise (12.04 LTS)` desplazándose hacia abajo. Consulte esta publicación de WikiHow para obtener la última versión de otras instalaciones de Ubuntu.

Actualizar las fuentes de APT

Esto debe hacerse para acceder a los paquetes desde el repositorio de Docker.

1. Inicie sesión en su máquina como usuario con privilegios de `sudo` o `root` .
2. Abra una ventana de terminal.
3. Actualice la información del paquete, asegúrese de que APT funcione con el método https y que los certificados de CA estén instalados.

```
$ sudo apt-get update
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
```

4. Agregue la llave GPG oficial de Docker:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Verifique que la huella dactilar de la clave sea **9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88** .

```
$ sudo apt-key fingerprint 0EBFCD88
```

```
pub   4096R/0EBFCD88 2017-02-22
      Key fingerprint = 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid                  Docker Release (CE deb) <docker@docker.com>
sub   4096R/F273FCD8 2017-02-22
```

5. Encuentre la entrada en la tabla a continuación que corresponda a su versión de Ubuntu. Esto determina dónde APT buscará los paquetes de Docker. Cuando sea posible, ejecute una edición de soporte a largo plazo (LTS) de Ubuntu.

Versión de Ubuntu	Repositorio
Preciso 12.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-precise main`
Trusty 14.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-trusty main`
Astuto 15.10	`deb https://apt.dockerproject.org/repo ubuntu-wily main`
Xenial 16.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-xenial main`

Nota: Docker no proporciona paquetes para todas las arquitecturas. Los artefactos binarios se crean todas las noches y puede descargarlos desde `https://master.dockerproject.org` . Para instalar la ventana acoplable en un sistema de arquitectura múltiple, agregue una cláusula `[arch=...]` a la entrada. Consulte la wiki de Debian Multiarch para más detalles.

6. Ejecute el siguiente comando, sustituyendo la entrada de su sistema operativo por el marcador de posición `<REPO>` .

```
$ echo "" | sudo tee /etc/apt/sources.list.d/docker.list
```

7. Actualice el índice del paquete `APT` ejecutando `sudo apt-get update` .

8. Verifique que `APT` está extrayendo del repositorio correcto.

Cuando ejecuta el siguiente comando, se devuelve una entrada para cada versión de Docker que está disponible para su instalación. Cada entrada debe tener la URL `https://apt.dockerproject.org/repo/` . La versión actualmente instalada está marcada con `***` Vea la salida del ejemplo a continuación.

```
$ apt-cache policy docker-engine

  docker-engine:
    Installed: 1.12.2-0~trusty
    Candidate: 1.12.2-0~trusty
    Version table:
   *** 1.12.2-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
          100 /var/lib/dpkg/status
       1.12.1-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
       1.12.0-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
```

A partir de ahora, cuando ejecute `apt-get upgrade` , `APT` extraerá del nuevo repositorio.

Prerrequisitos por la versión de Ubuntu

Para Ubuntu Trusty (14.04), Wily (15.10) y Xenial (16.04), instale los paquetes `linux-image-extra-*` kernel, que le permiten usar el controlador de almacenamiento `aufs` .

Para instalar los paquetes `linux-image-extra-*` :

1. Abra una terminal en su host de Ubuntu.

2. Actualice su gestor de paquetes con el comando `sudo apt-get update` .

3. Instale los paquetes recomendados.

   ```
   $ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
   ```

4. Continuar con la instalación de Docker.

Para Ubuntu Precise (12.04 LTS), Docker requiere la versión del kernel 3.13. Si la versión de su kernel es anterior a la 3.13, debe actualizarla. Consulte esta tabla para ver qué paquetes son necesarios para su entorno:

Paquete	Descripción
`linux-image-generic-lts-trusty`	Imagen genérica del kernel de linux. Este núcleo tiene `AUFS` incorporado. Esto es necesario para ejecutar Docker.

Paquete	Descripción
`linux-headers-generic-lts-trusty`	Permite paquetes como `ZFS` y `VirtualBox guest additions` que dependen de ellos. Si no instaló los encabezados para su kernel existente, puede omitir estos encabezados para el kernel de `trusty` . Si no está seguro, debe incluir este paquete para su seguridad.
`xserver-xorg-lts-trusty`	Opcional en entornos no gráficos sin Unity / Xorg. **Se requiere** cuando se ejecuta Docker en la máquina con un entorno gráfico.
`libgl1-mesa-glx-lts-trusty`	Para obtener más información acerca de las razones de estos paquetes, lea las instrucciones de instalación para los núcleos con puerto inverso, específicamente la pila de habilitación LTS . Consulte la nota 5 debajo de cada versión.

Para actualizar su kernel e instalar los paquetes adicionales, haga lo siguiente:

1. Abra una terminal en su host de Ubuntu.
2. Actualice su gestor de paquetes con el comando `sudo apt-get update` .
3. Instale los paquetes requeridos y opcionales.

```
$ sudo apt-get install linux-image-generic-lts-trusty
```

4. Repita este paso para otros paquetes que necesita instalar.
5. Reinicie su host para usar el kernel actualizado usando el comando `sudo reboot` .
6. Después de reiniciar, siga adelante e instale Docker.

Instala la última versión

Asegúrese de cumplir los requisitos previos, solo luego siga los pasos a continuación.

> **Nota:** Para los sistemas de producción, se recomienda instalar una versión específica para no actualizar accidentalmente Docker. Usted debe planear las actualizaciones para los sistemas de producción con cuidado.

1. Inicie sesión en su instalación de Ubuntu como usuario con privilegios de `sudo` . (Posiblemente ejecutando `sudo -su`).
2. Actualice su índice de paquetes APT ejecutando `sudo apt-get update` .
3. Instale Docker Community Edition con el comando `sudo apt-get install docker-ce` .
4. Inicie el daemon `docker` con el comando `sudo service docker start` .
5. Verifique que la `docker` esté instalada correctamente ejecutando la imagen de hello-world.

```
$ sudo docker run hello-world
```

Este comando descarga una imagen de prueba y la ejecuta en un contenedor. Cuando el contenedor se ejecuta, imprime un mensaje informativo y sale.

Administrar Docker como un usuario no root

Si no desea utilizar `sudo` cuando usa el comando de la ventana acoplable, cree un grupo Unix llamado `docker` y agregue usuarios a él. Cuando la `docker` comienza demonio, hace que la propiedad del socket de Unix de lectura / escritura por el grupo ventana acoplable.

Para crear el grupo `docker` y agregar su usuario:

1. Inicie sesión en Ubuntu como usuario con privilegios `sudo` .
2. Crear la `docker` grupo con el comando `sudo groupadd docker` .
3. Agregue su usuario al grupo `docker` .

   ```
   $ sudo usermod -aG docker $USER
   ```

4. Cierre la sesión y vuelva a iniciarla para que su membresía de grupo se vuelva a evaluar.
5. Verifique que pueda `docker` comandos sin `sudo` .

   ```
   $ docker run hello-world
   ```

Si esto falla, verá un error:

```
Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

Compruebe si la variable de entorno `DOCKER_HOST` está establecida para su shell.

```
$ env | grep DOCKER_HOST
```

Si está configurado, el comando anterior devolverá un resultado. Si es así, desactívalo.

```
$ unset DOCKER_HOST
```

Es posible que deba editar su entorno en archivos como `~/.bashrc` o `~/.profile` para evitar que la variable `DOCKER_HOST` se configure erróneamente.

Instalando Docker en Ubuntu

Requisitos: Docker se puede instalar en cualquier Linux con un kernel de al menos la versión 3.10. Docker es compatible con las siguientes versiones de 64 bits de Ubuntu Linux:

- Ubuntu Xenial 16.04 (LTS)

- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

Fácil instalación

Nota: la instalación de Docker desde el repositorio predeterminado de Ubuntu instalará una versión anterior de Docker.

Para instalar la última versión de Docker usando el repositorio de Docker, use `curl` para agarrar y ejecutar el script de instalación proporcionado por Docker:

```
$ curl -sSL https://get.docker.com/ | sh
```

Alternativamente, `wget` puede usarse para instalar Docker:

```
$ wget -qO- https://get.docker.com/ | sh
```

Docker ahora será instalado.

Instalación manual

Sin embargo, si ejecutar el script de instalación no es una opción, se pueden usar las siguientes instrucciones para instalar manualmente la última versión de Docker desde el repositorio oficial.

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates
```

Agregue la clave GPG:

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 \
  --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

A continuación, abra el archivo `/etc/apt/sources.list.d/docker.list` en su editor favorito. Si el archivo no existe, créelo. Eliminar cualquier entrada existente. Luego, dependiendo de su versión, agregue la siguiente línea:

- Ubuntu Precise 12.04 (LTS):

  ```
  deb https://apt.dockerproject.org/repo ubuntu-precise main
  ```

- Ubuntu Trusty 14.04 (LTS)

  ```
  deb https://apt.dockerproject.org/repo ubuntu-trusty main
  ```

- Ubuntu Wily 15.10

  ```
  deb https://apt.dockerproject.org/repo ubuntu-wily main
  ```

- Ubuntu Xenial 16.04 (LTS)

```
deb https://apt.dockerproject.org/repo ubuntu-xenial main
```

Guarde el archivo y salga, luego actualice el índice de su paquete, desinstale las versiones instaladas de Docker y verifique que `apt` está obteniendo el repositorio correcto:

```
$ sudo apt-get update
$ sudo apt-get purge lxc-docker
$ sudo apt-cache policy docker-engine
```

Dependiendo de su versión de Ubuntu, algunos requisitos previos pueden ser requeridos:

- Ubuntu Xenial 16.04 (LTS), Ubuntu Wily 15.10, Ubuntu Trusty 14.04 (LTS)

  ```
  sudo apt-get update && sudo apt-get install linux-image-extra-$(uname -r)
  ```

- Ubuntu Precise 12.04 (LTS)

  Esta versión de Ubuntu requiere el kernel versión 3.13. Es posible que necesite instalar paquetes adicionales dependiendo de su entorno:

  ```
  linux-image-generic-lts-trusty
  ```

  Imagen genérica del kernel de linux. Este núcleo tiene AUFS incorporado. Esto es necesario para ejecutar Docker.

  ```
  linux-headers-generic-lts-trusty
  ```

  Permite paquetes como ZFS y adiciones de invitados de VirtualBox que dependen de ellos. Si no instaló los encabezados para su kernel existente, puede omitir estos encabezados para el kernel de `trusty` . Si no está seguro, debe incluir este paquete para su seguridad.

  ```
  xserver-xorg-lts-trusty
  ```

  ```
  libgl1-mesa-glx-lts-trusty
  ```

  Estos dos paquetes son opcionales en entornos no gráficos sin Unity / Xorg. Se requiere cuando se ejecuta Docker en la máquina con un entorno gráfico.

  Para obtener más información acerca de los motivos de estos paquetes, lea las instrucciones de instalación para los núcleos con puerto trasero, específicamente la pila de habilitación de LTS; consulte la nota 5 en cada versión.

  Instale los paquetes necesarios y luego reinicie el host:

  ```
  $ sudo apt-get install linux-image-generic-lts-trusty
  ```

  ```
  $ sudo reboot
  ```

Finalmente, actualice el índice del paquete `apt` e instale Docker:

```
$ sudo apt-get update
$ sudo apt-get install docker-engine
```

Inicia el demonio:

```
$ sudo service docker start
```

Ahora verifique que la ventana acoplable esté funcionando correctamente iniciando una imagen de prueba:

```
$ sudo docker run hello-world
```

Este comando debe imprimir un mensaje de bienvenida que verifique que la instalación se realizó correctamente.

Crear un contenedor docker en Google Cloud

Puede usar la ventana acoplable, sin usar el demonio de la ventana acoplable (motor), utilizando proveedores en la nube. En este ejemplo, debe tener un `gcloud` (Google Cloud util), que está conectado a su cuenta

```
docker-machine create --driver google --google-project `your-project-name` google-machine-type
f1-large fm02
```

Este ejemplo creará una nueva instancia, en su consola de Google Cloud. Usando el tiempo de la máquina `f1-large`

Instalar Docker en Ubuntu

Docker es compatible con las siguientes versiones de *64 bits* de Ubuntu Linux:

- Ubuntu Xenial 16.04 (LTS)
- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

Un par de notas:

> Las siguientes instrucciones implican la instalación utilizando solo paquetes de **Docker** , y esto garantiza la obtención de la última versión oficial de **Docker** . Si necesita instalar solo utilizando paquetes `Ubuntu-managed` , consulte la documentación de Ubuntu (No se recomienda de otra manera por razones obvias).
>
> Ubuntu Utopic 14.10 y 15.04 existen en el repositorio APT de Docker, pero ya no se admiten oficialmente debido a problemas de seguridad conocidos.

Prerrequisitos

- Docker solo funciona en una instalación de Linux de 64 bits.
- Docker requiere la versión 3.10 o superior del kernel de Linux (excepto para `Ubuntu Precise 12.04` , que requiere la versión 3.13 o superior). Los núcleos anteriores a 3.10 carecen de algunas de las funciones necesarias para ejecutar los contenedores de Docker y contienen errores conocidos que causan la pérdida de datos y con frecuencia entran en pánico bajo

ciertas condiciones. Verifique la versión actual del kernel con el comando `uname -r` . Verifique esta publicación si necesita actualizar su kernel de `Ubuntu Precise (12.04 LTS)` desplazándose hacia abajo. Consulte esta publicación de WikiHow para obtener la última versión de otras instalaciones de Ubuntu.

Actualizar las fuentes de APT

Esto debe hacerse para acceder a los paquetes desde el repositorio de Docker.

1. Inicie sesión en su máquina como usuario con privilegios de `sudo` o `root` .
2. Abra una ventana de terminal.
3. Actualice la información del paquete, asegúrese de que APT funcione con el método https y que los certificados de CA estén instalados.

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates
```

4. Agregue la nueva llave `GPG` . Este comando descarga la clave con el ID `58118E89F3A912897C070ADBF76221572C52609D` del `58118E89F3A912897C070ADBF76221572C52609D` de claves `hkp://ha.pool.sks-keyservers.net:80` y la agrega al `adv keychain` . Para más información, vea la salida de `man apt-key` .

```
$ sudo apt-key adv \
       --keyserver hkp://ha.pool.sks-keyservers.net:80 \
       --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

5. Encuentre la entrada en la tabla a continuación que corresponda a su versión de Ubuntu. Esto determina dónde APT buscará los paquetes de Docker. Cuando sea posible, ejecute una edición de soporte a largo plazo (LTS) de Ubuntu.

Versión de Ubuntu	Repositorio
Preciso 12.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-precise main`
Trusty 14.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-trusty main`
Astuto 15.10	`deb https://apt.dockerproject.org/repo ubuntu-wily main`
Xenial 16.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-xenial main`

Nota: Docker no proporciona paquetes para todas las arquitecturas. Los artefactos binarios se crean todas las noches y puede descargarlos desde `https://master.dockerproject.org` . Para instalar la ventana acoplable en un sistema de arquitectura múltiple, agregue una cláusula `[arch=...]` a la entrada. Consulte la wiki de Debian Multiarch para más detalles.

6. Ejecute el siguiente comando, sustituyendo la entrada de su sistema operativo por el marcador de posición `<REPO>` .

```
$ echo "" | sudo tee /etc/apt/sources.list.d/docker.list
```

7. Actualice el índice del paquete `APT` ejecutando `sudo apt-get update` .

8. Verifique que `APT` está extrayendo del repositorio correcto.

Cuando ejecuta el siguiente comando, se devuelve una entrada para cada versión de Docker que está disponible para su instalación. Cada entrada debe tener la URL `https://apt.dockerproject.org/repo/` . La versión actualmente instalada está marcada con `***` Vea la salida del ejemplo a continuación.

```
$ apt-cache policy docker-engine

  docker-engine:
    Installed: 1.12.2-0~trusty
    Candidate: 1.12.2-0~trusty
    Version table:
   *** 1.12.2-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
          100 /var/lib/dpkg/status
       1.12.1-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
       1.12.0-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
```

A partir de ahora, cuando ejecute `apt-get upgrade` , `APT` extraerá del nuevo repositorio.

Prerrequisitos por la versión de Ubuntu

Para Ubuntu Trusty (14.04), Wily (15.10) y Xenial (16.04), instale los paquetes `linux-image-extra-*` kernel, que le permiten usar el controlador de almacenamiento `aufs` .

Para instalar los paquetes `linux-image-extra-*` :

1. Abra una terminal en su host de Ubuntu.

2. Actualice su gestor de paquetes con el comando `sudo apt-get update` .

3. Instale los paquetes recomendados.

   ```
   $ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
   ```

4. Continuar con la instalación de Docker.

Para Ubuntu Precise (12.04 LTS), Docker requiere la versión del kernel 3.13. Si la versión de su kernel es anterior a la 3.13, debe actualizarla. Consulte esta tabla para ver qué paquetes son necesarios para su entorno:

Paquete	Descripción
`linux-image-generic-lts-trusty`	Imagen genérica del kernel de linux. Este núcleo tiene `AUFS` incorporado. Esto es necesario para ejecutar Docker.

Paquete	Descripción
`linux-headers-generic-lts-trusty`	Permite paquetes como `ZFS` y `VirtualBox guest additions` que dependen de ellos. Si no instaló los encabezados para su kernel existente, puede omitir estos encabezados para el kernel de `trusty` . Si no está seguro, debe incluir este paquete para su seguridad.
`xserver-xorg-lts-trusty`	Opcional en entornos no gráficos sin Unity / Xorg. **Se requiere** cuando se ejecuta Docker en la máquina con un entorno gráfico.
`libgl1-mesa-glx-lts-trusty`	Para obtener más información acerca de las razones de estos paquetes, lea las instrucciones de instalación para los núcleos con puerto inverso, específicamente la pila de habilitación LTS . Consulte la nota 5 debajo de cada versión.

Para actualizar su kernel e instalar los paquetes adicionales, haga lo siguiente:

1. Abra una terminal en su host de Ubuntu.
2. Actualice su gestor de paquetes con el comando `sudo apt-get update` .
3. Instale los paquetes requeridos y opcionales.

```
$ sudo apt-get install linux-image-generic-lts-trusty
```

4. Repita este paso para otros paquetes que necesita instalar.
5. Reinicie su host para usar el kernel actualizado usando el comando `sudo reboot` .
6. Después de reiniciar, siga adelante e instale Docker.

Instala la última versión

Asegúrese de cumplir los requisitos previos, solo luego siga los pasos a continuación.

> **Nota:** Para los sistemas de producción, se recomienda instalar una versión específica para no actualizar accidentalmente Docker. Usted debe planear las actualizaciones para los sistemas de producción con cuidado.

1. Inicie sesión en su instalación de Ubuntu como usuario con privilegios de `sudo` . (Posiblemente ejecutando `sudo -su`).
2. Actualice su índice de paquetes APT ejecutando `sudo apt-get update` .
3. Instale Docker con el comando `sudo apt-get install docker-engine` .
4. Inicie el daemon `docker` con el comando `sudo service docker start` .
5. Verifique que la `docker` esté instalada correctamente ejecutando la imagen de hello-world.

```
$ sudo docker run hello-world
```

Este comando descarga una imagen de prueba y la ejecuta en un contenedor. Cuando el contenedor se ejecuta, imprime un mensaje informativo y sale.

Administrar Docker como un usuario no root

Si no desea utilizar `sudo` cuando usa el comando de la ventana acoplable, cree un grupo Unix llamado `docker` y agregue usuarios a él. Cuando la `docker` comienza demonio, hace que la propiedad del socket de Unix de lectura / escritura por el grupo ventana acoplable.

Para crear el grupo `docker` y agregar su usuario:

1. Inicie sesión en Ubuntu como usuario con privilegios `sudo` .
2. Crear la `docker` grupo con el comando `sudo groupadd docker` .
3. Agregue su usuario al grupo `docker` .

   ```
   $ sudo usermod -aG docker $USER
   ```

4. Cierre la sesión y vuelva a iniciarla para que su membresía de grupo se vuelva a evaluar.
5. Verifique que pueda `docker` comandos sin `sudo` .

   ```
   $ docker run hello-world
   ```

Si esto falla, verá un error:

```
Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

Compruebe si la variable de entorno `DOCKER_HOST` está establecida para su shell.

```
$ env | grep DOCKER_HOST
```

Si está configurado, el comando anterior devolverá un resultado. Si es así, desactívalo.

```
$ unset DOCKER_HOST
```

Es posible que deba editar su entorno en archivos como `~/.bashrc` o `~/.profile` para evitar que la variable `DOCKER_HOST` se configure erróneamente.

Instalación de Docker-ce O Docker-ee en CentOS

Docker ha anunciado las siguientes ediciones:

-Docker-ee (Enterprise Edition) junto con Docker-ce (Community Edition) y Docker (Soporte comercial)

Este documento lo ayudará con los pasos de instalación de Docker-ee y Docker-ce edition en CentOS

Instalacion Docker-ce

Los siguientes son los pasos para instalar la edición docker-ce

1. Instale yum-utils, que proporciona la utilidad yum-config-manager:

```
$ sudo yum install -y yum-utils
```

2. Use el siguiente comando para configurar el repositorio estable:

```
$ sudo yum-config-manager \
 --add-repo \
 https://download.docker.com/linux/centos/docker-ce.repo
```

3. Opcional: habilitar el repositorio de borde. Este repositorio se incluye en el archivo docker.repo anterior, pero está deshabilitado de forma predeterminada. Puedes habilitarlo junto al repositorio estable.

```
 $ sudo yum-config-manager --enable docker-ce-edge
```

- Puede deshabilitar el repositorio de borde ejecutando el comando `yum-config-manager` con el indicador `--disable` . Para volver a habilitarlo, use la `--enable` flag. El siguiente comando desactiva el repositorio de borde.

```
 $ sudo yum-config-manager --disable docker-ce-edge
```

4. Actualizar el índice del paquete yum.

```
 $ sudo yum makecache fast
```

5. Instale el docker-ce usando el siguiente comando:

```
 $ sudo yum install docker-ce-17.03.0.ce
```

6. Confirmar la huella digital de Docker-ce

   `060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35`

   Si desea instalar alguna otra versión de docker-ce, puede usar el siguiente comando:

```
$ sudo yum install docker-ce-VERSION
```

   Especifique el número de `VERSION`

7. Si todo salió bien, el docker-ce ahora está instalado en su sistema, use el siguiente

comando para iniciar:

```
$ sudo systemctl start docker
```

8. Pruebe su instalación docker:

```
$ sudo docker run hello-world
```

debería recibir el siguiente mensaje:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

-Docker-ee (Enterprise Edition) Instalación

Para Enterprise Edition (EE) sería necesario registrarse, para obtener su <DOCKER-EE-URL>.

1. Para registrarse vaya a https://cloud.docker.com/ . Ingrese sus datos y confirme su ID de correo electrónico. Después de la confirmación, recibirá un <DOCKER-EE-URL>, que podrá ver en su panel de control después de hacer clic en la configuración.
2. Elimine los repositorios Docker existentes de `/etc/yum.repos.d/`
3. Almacene su URL del repositorio de Docker EE en una variable yum en `/etc/yum/vars/` . Reemplace <DOCKER-EE-URL> con la URL que anotó en el primer paso.

```
$ sudo sh -c 'echo "<DOCKER-EE-URL>" > /etc/yum/vars/dockerurl'
```

4. Instale yum-utils, que proporciona la utilidad yum-config-manager:

```
$ sudo yum install -y yum-utils
```

5. Use el siguiente comando para agregar el repositorio estable:

```
$ sudo yum-config-manager \
--add-repo \
<DOCKER-EE-URL>/docker-ee.repo
```

6. Actualizar el índice del paquete yum.

```
$ sudo yum makecache fast
```

7. Instalar docker-ee

```
sudo yum install docker-ee
```

8. Puede iniciar el docker-ee usando el siguiente comando:

```
$ sudo systemctl start docker
```

Lea Empezando con Docker en línea: https://riptutorial.com/es/docker/topic/658/empezando-con-docker

Capítulo 2: API de Docker Engine

Introducción

Una API que le permite controlar cada aspecto de Docker desde sus propias aplicaciones, crear herramientas para administrar y monitorear las aplicaciones que se ejecutan en Docker e incluso usarlas para crear aplicaciones en Docker.

Examples

Habilitar el acceso remoto a la API de Docker en Linux

Edite `/etc/init/docker.conf` y actualice la variable `DOCKER_OPTS` a lo siguiente:

```
DOCKER_OPTS='-H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock'
```

Reiniciar Docker Docker

```
service docker restart
```

Verificar si la API remota está funcionando

```
curl -X GET http://localhost:4243/images/json
```

Habilitar el acceso remoto a la API de Docker en Linux ejecutando systemd

Linux ejecutando systemd, como Ubuntu 16.04, agregar `-H tcp://0.0.0.0:2375` a `/etc/default/docker` no tiene el efecto que solía tener.

En su lugar, cree un archivo llamado `/etc/systemd/system/docker-tcp.socket` para que la ventana acoplable esté disponible en un socket TCP en el puerto 4243:

```
[Unit]
Description=Docker Socket for the API
[Socket]
ListenStream=4243
Service=docker.service
[Install]
WantedBy=sockets.target
```

Entonces habilita el nuevo socket:

```
systemctl enable docker-tcp.socket
systemctl enable docker.socket
systemctl stop docker
systemctl start docker-tcp.socket
systemctl start docker
```

Ahora, verifique si la API remota está funcionando:

```
curl -X GET http://localhost:4243/images/json
```

Habilitar el acceso remoto con TLS en Systemd

Copie el archivo de la unidad del instalador de paquetes en / etc, donde los cambios no se sobrescribirán en una actualización:

```
cp /lib/systemd/system/docker.service /etc/systemd/system/docker.service
```

Actualice /etc/systemd/system/docker.service con sus opciones en ExecStart:

```
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2376 \
  --tlsverify --tlscacert=/etc/docker/certs/ca.pem \
  --tlskey=/etc/docker/certs/key.pem \
  --tlscert=/etc/docker/certs/cert.pem
```

Tenga en cuenta que `dockerd` es el nombre del daemon 1.12, antes de que fuera el `docker daemon` . También tenga en cuenta que 2376 es el puerto TLS estándar de los dockers, 2375 es el puerto no cifrado estándar. Consulte esta página para conocer los pasos para crear su propia CA, certificado y clave autofirmados por TLS.

Después de realizar cambios en los archivos de la unidad del sistema, ejecute lo siguiente para volver a cargar la configuración del sistema:

```
systemctl daemon-reload
```

Y luego ejecuta lo siguiente para reiniciar la ventana acoplable:

```
systemctl restart docker
```

Es una mala idea omitir el cifrado TLS al exponer el puerto Docker, ya que cualquier persona que tenga acceso a la red a este puerto efectivamente tiene acceso total a la raíz en el host.

Imagen tirando con barras de progreso, escrita en Ir.

Este es un ejemplo de extracción de imágenes con la `Docker Engine API Go` and `Docker Engine API` y las mismas barras de progreso que las que se muestran cuando ejecuta la `docker pull your_image_name` en la `CLI` . A los efectos de las barras de progreso se utilizan algunos códigos ANSI .

```
package yourpackage

import (
    "context"
    "encoding/json"
    "fmt"
    "io"
```

```
    "strings"

    "github.com/docker/docker/api/types"
    "github.com/docker/docker/client"
)

// Struct representing events returned from image pulling
type pullEvent struct {
    ID             string `json:"id"`
    Status         string `json:"status"`
    Error          string `json:"error,omitempty"`
    Progress       string `json:"progress,omitempty"`
    ProgressDetail struct {
        Current int `json:"current"`
        Total   int `json:"total"`
    } `json:"progressDetail"`
}

// Actual image pulling function
func PullImage(dockerImageName string) bool {
    client, err := client.NewEnvClient()

    if err != nil {
        panic(err)
    }

    resp, err := client.ImagePull(context.Background(), dockerImageName,
types.ImagePullOptions{})

    if err != nil {
        panic(err)
    }

    cursor := Cursor{}
    layers := make([]string, 0)
    oldIndex := len(layers)

    var event *pullEvent
    decoder := json.NewDecoder(resp)

    fmt.Printf("\n")
    cursor.hide()

    for {
        if err := decoder.Decode(&event); err != nil {
            if err == io.EOF {
                break
            }

            panic(err)
        }

        imageID := event.ID

        // Check if the line is one of the final two ones
        if strings.HasPrefix(event.Status, "Digest:") || strings.HasPrefix(event.Status,
"Status:") {
            fmt.Printf("%s\n", event.Status)
            continue
        }
```

```
        // Check if ID has already passed once
        index := 0
        for i, v := range layers {
            if v == imageID {
                index = i + 1
                break
            }
        }

        // Move the cursor
        if index > 0 {
            diff := index - oldIndex

            if diff > 1 {
                down := diff - 1
                cursor.moveDown(down)
            } else if diff < 1 {
                up := diff*(-1) + 1
                cursor.moveUp(up)
            }

            oldIndex = index
        } else {
            layers = append(layers, event.ID)
            diff := len(layers) - oldIndex

            if diff > 1 {
                cursor.moveDown(diff) // Return to the last row
            }

            oldIndex = len(layers)
        }

        cursor.clearLine()

        if event.Status == "Pull complete" {
            fmt.Printf("%s: %s\n", event.ID, event.Status)
        } else {
            fmt.Printf("%s: %s %s\n", event.ID, event.Status, event.Progress)
        }

    }

    cursor.show()

    if strings.Contains(event.Status, fmt.Sprintf("Downloaded newer image for %s",
dockerImageName)) {
        return true
    }

    return false
}
```

Para una mejor legibilidad, las acciones del cursor con los códigos ANSI se mueven a una estructura separada, que se ve así:

```
package yourpackage

import "fmt"
```

```
// Cursor structure that implements some methods
// for manipulating command line's cursor
type Cursor struct{}

func (cursor *Cursor) hide() {
    fmt.Printf("\033[?25l")
}

func (cursor *Cursor) show() {
    fmt.Printf("\033[?25h")
}

func (cursor *Cursor) moveUp(rows int) {
    fmt.Printf("\033[%dF", rows)
}

func (cursor *Cursor) moveDown(rows int) {
    fmt.Printf("\033[%dE", rows)
}

func (cursor *Cursor) clearLine() {
    fmt.Printf("\033[2K")
}
```

Después de eso, en su paquete principal puede llamar a la función `PullImage` pasando el nombre de la imagen que desea extraer. Por supuesto, antes de llamarlo, debe iniciar sesión en el registro de Docker, donde se encuentra la imagen.

Haciendo una solicitud cURL con pasar alguna estructura compleja

Al usar `cURL` para algunas consultas a la `Docker API` , puede ser un poco difícil pasar algunas estructuras complejas. Digamos que obtener una lista de imágenes permite utilizar filtros como un parámetro de consulta, que debe ser una representación `JSON` de la `map[string][]string` (sobre los mapas en `Go` , puede encontrar más información aquí).
Aquí es cómo lograr esto:

```
curl --unix-socket /var/run/docker.sock \
    -XGET "http:/v1.29/images/json" \
    -G \
    --data-urlencode 'filters={"reference":{"yourpreciousregistry.com/path/to/image": true},
"dangling":{"true": true}}'
```

Aquí, el indicador `-G` se usa para especificar que los datos en el parámetro `--data-urlencode` se usarán en una solicitud `HTTP GET` lugar de la solicitud `POST` que de lo contrario se usaría. Los datos se adjuntarán a la URL con un `?` separador.

Lea API de Docker Engine en línea: https://riptutorial.com/es/docker/topic/3935/api-de-docker-engine

Capítulo 3: Cómo configurar la réplica de Mongo de tres nodos con Docker Image y aprovisionado con Chef

Introducción

Esta documentación describe cómo construir un conjunto de réplicas Mongo de tres nodos con Docker Image y el aprovisionamiento automático con Chef.

Examples

Paso de compilación

Pasos:

1. Genere un archivo de claves Base 64 para la autenticación de nodo Mongo. Pon este archivo en chef data_bags
2. Ve al chef suppermarket y descarga el libro de cocina docker. Genere un libro de cocina personalizado (por ejemplo, custom_mongo) y agregue 'docker', '~> 2.0' a los metadatos.rb de su libro de cocina
3. Crea una receta y atributos en tu libro de cocina personalizado.
4. Inicializar Mongo para formar un conjunto de grupos de representantes

Paso 1: Crear archivo de clave

crear una bolsa de datos llamada mongo-keyfile y un elemento llamado keyfile. Esto estará en el directorio data_bags en chef. El contenido del artículo será el siguiente

```
openssl rand -base64 756 > <path-to-keyfile>
```

contenido del elemento del archivo clave

```
{
  "id": "keyfile",
  "comment": "Mongo Repset keyfile",
  "key-file": "generated base 64 key above"
}
```

Paso 2: descargue el libro de cocina docker del chef supper market y luego cree el libro de cocina custom_mongo

```
knife cookbook site download docker
knife cookbook create custom_mongo
```

en metadat.rb de custom_mongo add

```
depends          'docker', '~> 2.0'
```

Paso 3: crear atributo y receta

Atributos

```
default['custom_mongo']['mongo_keyfile'] = '/data/keyfile'
default['custom_mongo']['mongo_datadir'] = '/data/db'
default['custom_mongo']['mongo_datapath'] = '/data'
default['custom_mongo']['keyfilename'] = 'mongodb-keyfile'
```

Receta

```
#
# Cookbook Name:: custom_mongo
# Recipe:: default
#
# Copyright 2017, Innocent Anigbo
#
# All rights reserved - Do Not Redistribute
#

data_path = "#{node['custom_mongo']['mongo_datapath']}"
data_dir = "#{node['custom_mongo']['mongo_datadir']}"
key_dir = "#{node['custom_mongo']['mongo_keyfile']}"
keyfile_content = data_bag_item('mongo-keyfile', 'keyfile')
keyfile_name = "#{node['custom_mongo']['keyfilename']}"

#chown of keyfile to docker user
execute 'assign-user' do
 command "chown 999 #{key_dir}/#{keyfile_name}"
 action :nothing
end

#Declaration to create Mongo data DIR and Keyfile DIR
%W[ #{data_path} #{data_dir} #{key_dir} ].each do |path|
directory path do
  mode '0755'
  end
end

#declaration to copy keyfile from data_bag to keyfile DIR on your mongo server
file "#{key_dir}/#{keyfile_name}" do
  content keyfile_content['key-file']
  group 'root'
  mode '0400'
  notifies :run, 'execute[assign-user]', :immediately
end

#Install docker
docker_service 'default' do
  action [:create, :start]
```

```
end

#Install mongo 3.4.2
docker_image 'mongo' do
  tag '3.4.2'
  action :pull
end
```

Crear un rol llamado mongo-role en el directorio de roles

```
{
  "name": "mongo-role",
  "description": "mongo DB Role",
  "run_list": [
    "recipe[custom_mongo]"
  ]
}
```

Agregue el rol anterior a la lista de ejecución de tres nodos mongo

```
knife node run_list add FQDN_of_node_01 'role[mongo-role]'
knife node run_list add FQDN_of_node_02 'role[mongo-role]'
knife node run_list add FQDN_of_node_03 'role[mongo-role]'
```

Paso 4: Inicializa el Mongo de tres nodos para formar el repset

Supongo que el rol anterior ya se ha aplicado en los tres nodos de Mongo. Solo en el nodo 01, inicie Mongo con --auth para habilitar la autenticación

```
docker run --name mongo -v /data/db:/data/db -v /data/keyfile:/opt/keyfile --hostname="mongo-
01.example.com" -p 27017:27017 -d mongo:3.4.2 --keyFile /opt/keyfile/mongodb-keyfile --auth
```

Acceda al shell interactivo de la ejecución del contenedor docker en el nodo 01 y cree un usuario administrador

```
docker exec -it mongo /bin/sh
    mongo
    use admin
    db.createUser( {
         user: "admin-user",
         pwd: "password",
         roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
       });
```

Crear usuario root

```
db.createUser( {
         user: "RootAdmin",
         pwd: "password",
         roles: [ { role: "root", db: "admin" } ]
       });
```

Detenga y elimine el contenedor Docker creado anteriormente en el nodo 01. Esto no afectará los

datos y el archivo de claves en el DIR del host. Después de eliminar, inicie Mongo de nuevo en el nodo 01, pero esta vez con el indicador de repetición

```
docker rm -fv mongo
docker run --name mongo-uat -v /data/db:/data/db -v /data/keyfile:/opt/keyfile --
hostname="mongo-01.example.com" -p 27017:27017 -d mongo:3.4.2 --keyFile /opt/keyfile/mongodb-
keyfile --replSet "rs0"
```

ahora inicie mongo en los nodos 02 y 03 con la marca de configuración de rep

```
docker run --name mongo -v /data/db:/data/db -v /data/keyfile:/opt/keyfile --hostname="mongo-
02.example.com" -p 27017:27017 -d mongo:3.4.2 --keyFile /opt/keyfile/mongodb-keyfile --replSet
"rs0"
docker run --name mongo -v /data/db:/data/db -v /data/keyfile:/opt/keyfile --hostname="mongo-
03.example.com" -p 27017:27017 -d mongo:3.4.2 --keyFile /opt/keyfile/mongodb-keyfile --replSet
"rs0"
```

Autentíquese con el usuario root en el Nodo 01 e inicie el conjunto de réplicas

```
use admin
db.auth("RootAdmin", "password");
rs.initiate()
```

En el nodo 01, agregue los Nodos 2 y 3 al conjunto de réplicas para formar el grupo repset0

```
rs.add("mongo-02.example.com")
rs.add("mongo-03.example.com")
```

Pruebas

En la ejecución principal db.printSlaveReplicationInfo () y observe la hora sincronizada y sincronizada. El último debe ser 0 segundos como abajo

Salida

```
rs0:PRIMARY> db.printSlaveReplicationInfo()
   source: mongo-02.example.com:27017
           syncedTo: Mon Mar 27 2017 15:01:04 GMT+0000 (UTC)
           0 secs (0 hrs) behind the primary
   source: mongo-03.example.com:27017
           syncedTo: Mon Mar 27 2017 15:01:04 GMT+0000 (UTC)
           0 secs (0 hrs) behind the primary
```

Espero que esto ayude a alguien

Lea Cómo configurar la réplica de Mongo de tres nodos con Docker Image y aprovisionado con Chef en línea: https://riptutorial.com/es/docker/topic/10014/como-configurar-la-replica-de-mongo-de-tres-nodos-con-docker-image-y-aprovisionado-con-chef

Capítulo 4: Cómo depurar cuando falla la compilación del docker

Introducción

Cuando una `docker build -t mytag .` falla con un mensaje como `---> Running in d9a42e53eb5a The command '/bin/sh -c returned a non-zero code: 127` (127 significa "comando no encontrado, pero 1) no es trivial para todos 2) 127 puede ser reemplazado por 6 o cualquier cosa) puede ser no trivial para encontrar el error en una línea larga

Examples

ejemplo básico

Como la última capa creada por

```
docker build -t mytag .
```

mostró

```
---> Running in d9a42e53eb5a
```

Simplemente inicie la última imagen creada con un shell e inicie el comando, y tendrá un mensaje de error más claro

```
docker run -it d9a42e53eb5a /bin/bash
```

(esto asume que / bin / bash está disponible, puede ser / bin / sh o cualquier otra cosa)

y con el indicador, ejecuta el último comando que falla y ve lo que se muestra

Lea Cómo depurar cuando falla la compilación del docker en línea:
https://riptutorial.com/es/docker/topic/8078/como-depurar-cuando-falla-la-compilacion-del-docker

Capítulo 5: Concepto de volúmenes Docker

Observaciones

La gente nueva en Docker a menudo no se da cuenta de que los sistemas de archivos de Docker son temporales por defecto. Si inicia una imagen de Docker, obtendrá un contenedor que en la superficie se comporta como una máquina virtual. Puedes crear, modificar y borrar archivos. Sin embargo, a diferencia de una máquina virtual, si detiene el contenedor y lo vuelve a iniciar, todos los cambios se perderán: todos los archivos que eliminó anteriormente volverán y los nuevos archivos o ediciones que haya realizado no estarán presentes.

Los volúmenes en los contenedores de la ventana acoplable permiten datos persistentes y compartir datos de la máquina host dentro de un contenedor.

Examples

A) Lanzar un contenedor con un volumen.

```
[root@localhost ~]# docker run -it -v  /data  --name=vol3   8251da35e7a7 /bin/bash
root@d87bf9607836:/# cd /data/
root@d87bf9607836:/data# touch abc{1..10}
root@d87bf9607836:/data# ls
```

abc1 abc10 abc2 abc3 abc4 abc5 abc6 abc8 abc8 abc9

B) Ahora presione [cont + P + Q] para salir del contenedor sin terminar el contenedor buscando el contenedor que se está ejecutando

```
[root@localhost ~]# docker ps
```

CONTENIDO ID DE IMAGEN COMANDO IMAGEN ESTADO PUERTOS NOMBRES d87bf9607836 8251da35e7a7 "/ bin / bash" Hace aproximadamente un minuto Hasta 31 segundos vol3 [root @ localhost ~] #

C) Ejecute 'docker inspect' para ver más información sobre el volumen

```
[root@localhost ~]# docker inspect  d87bf9607836
```

" ' "Modo": "", "RW": verdadero

D) Puede adjuntar un volumen de contenedores en ejecución a otros contenedores

```
[root@localhost ~]# docker run -it  --volumes-from  vol3  8251da35e7a7  /bin/bash
```

```
root@ef2f5cc545be:/# ls
```

bin boot data dev etc inicio lib lib64 media mnt opt proc raíz ejecutar sbin srv sys tmp usr var

`root@ef2f5cc545be:/# ls` / data abc1 abc10 abc2 abc4 abc5 abc6 abc7 abc8 abc9

E) También puedes montar tu directorio base dentro del contenedor

```
[root@localhost ~]# docker run -it  -v  /etc:/etc1 8251da35e7a7 /bin/bash
```

Aquí: / etc es el directorio de la máquina host y / etc1 es el destino dentro del contenedor

Lea Concepto de volúmenes Docker en línea:
https://riptutorial.com/es/docker/topic/5908/concepto-de-volumenes-docker

Capítulo 6: Construyendo imagenes

Parámetros

Parámetro	Detalles
--Halar	Asegura que la imagen base (`FROM`) esté actualizada antes de construir el resto del Dockerfile.

Examples

Construyendo una imagen desde un Dockerfile

Una vez que tenga un Dockerfile, puede construir una imagen a partir de él utilizando la `docker build` . La forma básica de este comando es:

```
docker build -t image-name path
```

Si su Dockerfile no se nombra `Dockerfile` , se puede utilizar el `-f` pabellón que suministre el nombre de la Dockerfile para construir.

```
docker build -t image-name -f Dockerfile2 .
```

Por ejemplo, para crear una imagen llamada `dockerbuild-example:1.0.0` partir de un `Dockerfile` en el directorio de trabajo actual:

```
$ ls
Dockerfile Dockerfile2

$ docker build -t dockerbuild-example:1.0.0 .

$ docker build -t dockerbuild-example-2:1.0.0 -f Dockerfile2 .
```

Consulte la documentación de uso de `docker build` la `docker build` para obtener más opciones y configuraciones.

Un error común es crear un Dockerfile en el directorio de inicio del usuario (`~`). Esta es una mala idea porque durante la `docker build -t mytag .` Este mensaje aparecerá durante mucho tiempo:

> Cargando contexto

La causa es que el demonio docker intenta copiar todos los archivos del usuario (tanto el directorio de inicio como sus subdirectorios). Evite esto especificando siempre un directorio para el Dockerfile.

Agregar un archivo `.dockerignore` al directorio de compilación es una buena práctica . Su sintaxis

es similar a los archivos `.gitignore` y se asegurará de que solo los archivos y directorios deseados se carguen como contexto de la compilación.

Un simple Dockerfile

```
FROM node:5
```

La directiva `FROM` especifica una imagen para comenzar. Se puede utilizar cualquier referencia de imagen válida.

```
WORKDIR /usr/src/app
```

La directiva `WORKDIR` establece el directorio de trabajo actual dentro del contenedor, equivalente a ejecutar `cd` dentro del contenedor. (Nota: `RUN cd` *no* cambiará el directorio de trabajo actual.)

```
RUN npm install cowsay knock-knock-jokes
```

`RUN` ejecuta el comando dado dentro del contenedor.

```
COPY cowsay-knockknock.js ./
```

`COPY` copia el archivo o directorio especificado en el primer argumento del contexto de construcción (la *`path`* pasó a `docker build` *`path`*) a la ubicación en el contenedor especificado por el segundo argumento.

```
CMD node cowsay-knockknock.js
```

`CMD` especifica un comando para ejecutar cuando la imagen se ejecuta y no se da ningún comando. Puede anularse pasando un comando a la `docker run` .

Hay muchas otras instrucciones y opciones; vea la referencia de Dockerfile para una lista completa.

Diferencia entre ENTRYPOINT y CMD

Hay dos directivas de `Dockerfile` para especificar qué comando se ejecutará de forma predeterminada en las imágenes integradas. Si solo especifica `CMD` , la `ENTRYPOINT` acoplable ejecutará ese comando usando el `ENTRYPOINT` predeterminado, que es `/bin/sh -c` . Puede anular cualquiera de los puntos de entrada y / o el comando al iniciar la imagen construida. Si especifica ambos, entonces `ENTRYPOINT` especifica el ejecutable de su proceso contenedor, y `CMD` se suministrará como los parámetros de ese ejecutable.

Por ejemplo, si su `Dockerfile` contiene

```
FROM ubuntu:16.04
CMD ["/bin/date"]
```

Entonces está utilizando la directiva predeterminada `ENTRYPOINT` de `/bin/sh -c` , y ejecutando `/bin/date` con ese punto de entrada predeterminado. El comando de su proceso contenedor será `/bin/sh -c /bin/date` . Una vez que ejecute esta imagen, por defecto imprimirá la fecha actual

```
$ docker build -t test .
$ docker run test
Tue Jul 19 10:37:43 UTC 2016
```

Puede anular `CMD` en la línea de comando, en cuyo caso ejecutará el comando que ha especificado.

```
$ docker run test /bin/hostname
bf0274ec8820
```

Si especifica una directiva `ENTRYPOINT` , Docker usará ese ejecutable, y la directiva `CMD` especifica los parámetros predeterminados del comando. Así que si tu `Dockerfile` contiene:

```
FROM ubuntu:16.04
ENTRYPOINT ["/bin/echo"]
CMD ["Hello"]
```

Entonces ejecutarlo producirá

```
$ docker build -t test .
$ docker run test
Hello
```

Puede proporcionar diferentes parámetros si lo desea, pero todos se ejecutarán `/bin/echo`

```
$ docker run test Hi
Hi
```

Si desea anular el punto de entrada listado en su Dockerfile (es decir, si desea ejecutar un comando diferente al `echo` en este contenedor), debe especificar el parámetro `--entrypoint` en la línea de comandos:

```
$ docker run --entrypoint=/bin/hostname test
b2c70e74df18
```

Por lo general, usa la directiva `ENTRYPOINT` para apuntar a la aplicación principal que desea ejecutar y `CMD` a los parámetros predeterminados.

Exponiendo un puerto en el Dockerfile

```
EXPOSE <port> [<port>...]
```

De la documentación de Docker:

> La instrucción `EXPOSE` informa a Docker que el contenedor escucha en los puertos de

red especificados en tiempo de ejecución. `EXPOSE` no hace que los puertos del contenedor sean accesibles para el host. Para hacerlo, debe usar el indicador `-p` para publicar un rango de puertos o el indicador `-P` para publicar todos los puertos expuestos. Puede exponer un número de puerto y publicarlo externamente bajo otro número.

Ejemplo:

Dentro de tu Dockerfile:

```
EXPOSE 8765
```

Para acceder a este puerto desde la máquina host, incluya este argumento en su comando de `docker run` :

```
-p 8765:8765
```

ENTRYPOINT y CMD vistos como verbo y parámetro

Supongamos que tiene un archivo Docker que termina con

```
ENTRYPOINT [ "nethogs"] CMD ["wlan0"]
```

Si construyes esta imagen con una

```
docker built -t inspector .
```

inicie la imagen creada con un archivo Docker con un comando como

```
docker run -it --net=host --rm inspector
```

, nethogs monitoreará la interfaz llamada wlan0

Ahora, si quieres monitorear la interfaz eth0 (o wlan1, o ra1 ...), harás algo como

```
docker run -it --net=host --rm inspector eth0
```

o

```
docker run -it --net=host --rm inspector wlan1
```

Empujando y tirando de una imagen a Docker Hub u otro registro

Las imágenes creadas localmente se pueden enviar a Docker Hub o cualquier otro host de repo docker, conocido como registro. Utilice el `docker login` de `docker login` para `docker login` sesión en una cuenta de concentrador de la ventana acoplable existente.

```
docker login

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: cjsimon
Password:
Login Succeeded
```

Se puede utilizar un registro de ventana acoplable diferente especificando un nombre de servidor. Esto también funciona para registros privados o auto alojados. Además, es posible utilizar un almacén de credenciales externo para la seguridad.

```
docker login quay.io
```

A continuación, puede etiquetar y enviar imágenes al registro en el que ha iniciado sesión. Su repositorio debe especificarse como `server/username/reponame:tag` Omitir el servidor actualmente por defecto a Docker Hub. (El registro predeterminado no se puede cambiar a otro proveedor, y no hay planes para implementar esta función).

```
docker tag mynginx quay.io/cjsimon/mynginx:latest
```

Se pueden usar diferentes etiquetas para representar diferentes versiones, o ramas, de la misma imagen. Una imagen con múltiples etiquetas diferentes mostrará cada etiqueta en el mismo repositorio.

Use las `docker images` para ver una lista de las imágenes instaladas instaladas en su máquina local, incluida la imagen recién etiquetada. Luego, presione push para cargarlo en el registro y tire para descargar la imagen.

```
docker push quay.io/cjsimon/mynginx:latest
```

Todas las etiquetas de una imagen se pueden extraer especificando la opción `-a`

```
docker pull quay.io/cjsimon/mynginx:latest
```

Construyendo usando un proxy

A menudo, al crear una imagen de Docker, el Dockerfile contiene instrucciones que ejecutan programas para obtener recursos de Internet (por ejemplo, `wget` para extraer un programa binario construido en GitHub).

Es posible instruir a Docker para que pase las variables de entorno del conjunto establecido para que dichos programas realicen esas recuperaciones a través de un proxy:

```
$ docker build --build-arg http_proxy=http://myproxy.example.com:3128 \
               --build-arg https_proxy=http://myproxy.example.com:3128 \
               --build-arg no_proxy=internal.example.com \
               -t test .
```

`build-arg` son variables de entorno que están disponibles solo en tiempo de compilación.

Lea Construyendo imagenes en línea: https://riptutorial.com/es/docker/topic/713/construyendo-imagenes

Capítulo 7: Contenedores de conexión

Parámetros

Parámetro	Detalles
`tty:true`	En docker-compose.yml, el indicador `tty: true` mantiene el comando `sh` del contenedor en espera de entrada.

Observaciones

Los controladores de red del `host` y del `bridge` pueden conectar contenedores en un solo host de docker. Para permitir que los contenedores se comuniquen más allá de una máquina, cree una red de superposición. Los pasos para crear la red dependen de cómo se administran los hosts de la ventana acoplable.

- Modo de enjambre: la `docker network create --driver overlay`
- ventana acoplable / enjambre : requiere un almacén externo de clave-valor

Examples

Red docker

Los contenedores en la misma red de ventana acoplable tienen acceso a los puertos expuestos.

```
docker network create sample
docker run --net sample --name keys consul agent -server -client=0.0.0.0 -bootstrap
```

Consul's Dockerfile expone `8500` , `8600` y varios puertos más. Para demostrar, ejecute otro contenedor en la misma red:

```
docker run --net sample -ti alpine sh
/ # wget -qO- keys:8500/v1/catalog/nodes
```

Aquí el contenedor del cónsul se resuelve a partir de las `keys` , el nombre dado en el primer comando. Docker proporciona resolución de DNS en esta red, para encontrar contenedores por su `--name` .

Docker-componer

Las redes se pueden especificar en un archivo compuesto (v2). Por defecto, todos los contenedores están en una red compartida.

Comience con este archivo: `example/docker-compose.yml` :

```
version: '2'
services:
  keys:
    image: consul
    command: agent -server -client=0.0.0.0 -bootstrap
  test:
    image: alpine
    tty: true
    command: sh
```

Al iniciar esta pila con `docker-compose up -d` se creará una red con el nombre del directorio principal, en este caso `example_default` . Consulte con la `docker network ls`

```
 > docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
719eafa8690b        example_default     bridge              local
```

Conéctese al contenedor alpino para verificar que los contenedores puedan resolverse y comunicarse:

```
 > docker exec -ti example_test_1 sh
/ # nslookup keys
...
/ # wget -qO- keys:8500/v1/kv/?recurse
...
```

Un archivo de redacción puede tener una sección de `networks:` nivel superior para especificar el nombre de la red, el controlador y otras opciones del comando de red de la ventana acoplable .

Vinculación de contenedores

La `--link` acoplable `--link` argumento de `link:` , y `link:` secciones `--link` acoplable-componer `--link` *alias* a otros contenedores.

```
docker network create sample
docker run -d --net sample --name redis redis
```

Con el enlace, ya sea el nombre original o la asignación, se resolverá el contenedor redis.

```
> docker run --net sample --link redis:cache -ti python:alpine sh -c "pip install redis &&
python"
>>> import redis
>>> r = redis.StrictRedis(host='cache')
>>> r.set('key', 'value')
True
```

Antes de `1.10.0` contenedor de la `1.10.0` acoplable `1.10.0` , también configure la conectividad de la red, el comportamiento ahora es proporcionado por la red de la ventana acoplable. Los enlaces en versiones posteriores solo proporcionan un efecto `legacy` en la red de puente predeterminada.

Lea Contenedores de conexión en línea:

https://riptutorial.com/es/docker/topic/6528/contenedores-de-conexion

Capítulo 8: Contenedores de control y restauración

Examples

Compilar ventana acoplable con punto de control y restauración habilitada (ubuntu)

Para compilar la ventana acoplable se recomienda tener al menos **2 GB de RAM** . Incluso con eso, a veces falla, así que es mejor ir a **4GB** .

1. Asegúrate de que git y make estén instalados

```
sudo apt-get install make git-core -y
```

2. instalar un nuevo kernel (al menos 4.2)

```
sudo apt-get install linux-generic-lts-xenial
```

3. reinicie la máquina para tener el nuevo kernel activo

```
sudo reboot
```

4. compile `criu` que se necesita para ejecutar el `docker checkpoint criu docker checkpoint`

```
sudo apt-get install libprotobuf-dev libprotobuf-c0-dev protobuf-c-compiler protobuf-
compiler python-protobuf libnl-3-dev libcap-dev -y
wget http://download.openvz.org/criu/criu-2.4.tar.bz2 -O - | tar -xj
cd criu-2.4
make
make install-lib
make install-criu
```

5. Compruebe si se cumplen todos los requisitos para ejecutar Criu

```
sudo criu check
```

6. compilar docker experimental (necesitamos docker para compilar docker)

```
cd ~
wget -qO- https://get.docker.com/ | sh
sudo usermod -aG docker $(whoami)
```

- **En este punto, tenemos que cerrar sesión y volver a iniciar sesión para tener un daemon docker. Después de volver a iniciar sesión continuar con el paso de**

compilación

```
git clone https://github.com/boucher/docker
cd docker
git checkout docker-checkpoint-restore
make #that will take some time - drink a coffee
DOCKER_EXPERIMENTAL=1 make binary
```

7. Ahora tenemos una ventana acoplable compilada. Vamos a mover los binarios. Asegúrese de reemplazar `<version>` con la versión instalada

```
sudo service docker stop
sudo cp $(which docker) $(which docker)_ ; sudo cp ./bundles/latest/binary-client/docker-
<version>-dev $(which docker)
sudo cp $(which docker-containerd) $(which docker-containerd)_ ; sudo cp
./bundles/latest/binary-daemon/docker-containerd $(which docker-containerd)
sudo cp $(which docker-containerd-ctr) $(which docker-containerd-ctr)_ ; sudo cp
./bundles/latest/binary-daemon/docker-containerd-ctr $(which docker-containerd-ctr)
sudo cp $(which docker-containerd-shim) $(which docker-containerd-shim)_ ; sudo cp
./bundles/latest/binary-daemon/docker-containerd-shim $(which docker-containerd-shim)
sudo cp $(which dockerd) $(which dockerd)_ ; sudo cp ./bundles/latest/binary-
daemon/dockerd $(which dockerd)
sudo cp $(which docker-runc) $(which docker-runc)_ ; sudo cp ./bundles/latest/binary-
daemon/docker-runc $(which docker-runc)
sudo service docker start
```

No te preocupes, hicimos una copia de seguridad de los viejos binarios. Todavía están allí, pero con un guion bajo agregado a sus nombres (`docker_`).

Enhorabuena, ahora tiene una ventana acoplable experimental con la capacidad de controlar un contenedor y restaurarlo.

Tenga en cuenta que las características experimentales NO están listas para la producción

Punto de control y restauración de un contenedor

```
# create docker container
export cid=$(docker run -d --security-opt seccomp:unconfined busybox /bin/sh -c 'i=0; while
true; do echo $i; i=$(expr $i + 1); sleep 1; done')

# container is started and prints a number every second
# display the output with
docker logs $cid

# checkpoint the container
docker checkpoint create $cid checkpointname

# container is not running anymore
docker np

# lets pass some time to make sure

# resume container
docker start $cid --checkpoint=checkpointname
```

```
# print logs again
docker logs $cid
```

Lea Contenedores de control y restauración en línea:
https://riptutorial.com/es/docker/topic/5291/contenedores-de-control-y-restauracion

Capítulo 9: Contenedores de correr

Sintaxis

- ventana acoplable ejecutar [OPCIONES] IMAGEN [COMANDO] [ARG ...]

Examples

Corriendo un contenedor

```
docker run hello-world
```

Esto traerá la última imagen de hello-world desde Docker Hub (si aún no la tiene), creará un nuevo contenedor y ejecutarlo. Debería ver un mensaje que indica que su instalación parece estar funcionando correctamente.

Ejecutando un comando diferente en el contenedor

```
docker run docker/whalesay cowsay 'Hello, StackExchange!'
```

Este comando le dice a Docker que cree un contenedor desde la imagen de `docker/whalesay` y ejecute el comando `cowsay 'Hello, StackExchange!'` en eso. Debería imprimir una imagen de una ballena diciendo: `Hello, StackExchange!` a tu terminal

Si el punto de entrada en la imagen es el predeterminado, puede ejecutar cualquier comando que esté disponible en la imagen:

```
docker run docker/whalesay ls /
```

Si se ha cambiado durante la creación de la imagen, debe revertirla al valor predeterminado.

```
docker run --entrypoint=/bin/bash docker/whalesay -c ls /
```

Eliminar automáticamente un contenedor después de ejecutarlo

Normalmente, un contenedor Docker persiste después de haber salido. Esto le permite ejecutar el contenedor nuevamente, inspeccionar su sistema de archivos, y así sucesivamente. Sin embargo, a veces desea ejecutar un contenedor y eliminarlo inmediatamente después de que se cierre. Por ejemplo, para ejecutar un comando o mostrar un archivo desde el sistema de archivos. Docker proporciona la opción de línea de comandos `--rm` para este propósito:

```
docker run --rm ubuntu cat /etc/hosts
```

Esto creará un contenedor a partir de la imagen "ubuntu", mostrará el contenido del **archivo / etc**

/ hosts y luego eliminará el contenedor inmediatamente después de que salga. Esto ayuda a evitar tener que limpiar los contenedores después de que haya terminado de experimentar.

> **Nota:** el indicador `--rm` no funciona junto con el `--detach -d` (`--detach`) en la ventana acoplable <1.13.0.

Cuando se `--rm` indicador `--rm` , Docker también elimina los volúmenes asociados con el contenedor cuando se elimina el contenedor. Esto es similar a la ejecución de `docker rm -v my-container` . ***Sólo se eliminan los volúmenes que se especifican sin un nombre*** .

Por ejemplo, con la `docker run -it --rm -v /etc -v logs:/var/log centos /bin/produce_some_logs` , el volumen de `/etc` se eliminará, pero el volumen de `/var/log` no se eliminará. Los volúmenes heredados a través de --volumes-from se eliminarán con la misma lógica: si el volumen original se especificó con un nombre, no se eliminará.

Especificando un nombre

De forma predeterminada, los contenedores creados con la `docker run small_roentgen docker run` reciben un nombre aleatorio como `small_roentgen` o `modest_dubinsky` . Estos nombres no son particularmente útiles para identificar el propósito de un contenedor. Es posible proporcionar un nombre para el contenedor al pasar la opción de línea de comando `--name` :

```
docker run --name my-ubuntu ubuntu:14.04
```

Los nombres deben ser únicos; Si pasa un nombre que otro contenedor ya está usando, Docker imprimirá un error y no se creará ningún contenedor nuevo.

Especificar un nombre será útil al hacer referencia al contenedor dentro de una red Docker. Esto funciona tanto para contenedores Docker de fondo como de fondo.

Los contenedores en la red puente predeterminada **deben** estar vinculados para comunicarse por nombre.

Enlace de un puerto de contenedor al host

```
docker run -p "8080:8080" myApp
docker run -p "192.168.1.12:80:80" nginx
docker run -P myApp
```

Para poder usar los puertos en el host se han expuesto en una imagen (a través de la directiva `EXPOSE` Dockerfile, o `--expose` opción de línea de comandos para la `docker run --expose docker run`), esos puertos deben vincularse al host mediante el comando `-p` o `-P` Opciones de línea. El uso de `-p` requiere que se especifique el puerto particular (y la interfaz de host opcional). El uso de la opción de línea de comando en mayúscula `-P` obligará a Docker a vincular *todos* los puertos expuestos en la imagen de un contenedor al host.

Política de reinicio del contenedor (iniciando un contenedor en el arranque)

```
docker run --restart=always -d <container>
```

De forma predeterminada, Docker no reiniciará los contenedores cuando se reinicie el demonio Docker, por ejemplo, después de reiniciar el sistema host. Docker proporciona una política de reinicio para sus contenedores al proporcionar la opción de línea de comandos `--restart` . Suministrar `--restart=always` hará que se reinicie un contenedor después de reiniciar el demonio Docker. **Sin embargo,** cuando ese contenedor se detiene manualmente (p. Ej., Con la `docker stop <container>`), la política de reinicio no se aplicará al contenedor.

Se pueden especificar varias opciones para la opción `--restart` , según el requisito (`--restart=[policy]`). Estas opciones afectan también a cómo se inicia el contenedor en el arranque.

Política	Resultado
no	El valor por **defecto** . No reiniciará el contenedor automáticamente, cuando se detiene el contenedor.
en falla [: max-reintentos]	Reinicie solo si el contenedor sale con un error (`non-zero exit status`). Para evitar reiniciarlo indefinidamente (en caso de algún problema), se puede limitar el número de reintentos de reinicio que intenta el daemon Docker.
siempre	Siempre reinicie el contenedor independientemente del estado de salida. Cuando especifique `always` , el demonio Docker intentará reiniciar el contenedor de forma indefinida. El contenedor también se iniciará siempre en el inicio del daemon, independientemente del estado actual del contenedor.
a menos que sea detenido	Siempre reinicie el contenedor independientemente de su estado de salida, pero no lo inicie en el inicio del daemon si el contenedor se ha puesto en un estado detenido anteriormente.

Ejecutar un contenedor en segundo plano

Para mantener un contenedor ejecutándose en segundo plano, suministre la opción de línea de comando `-d` durante el inicio del contenedor:

```
docker run -d busybox top
```

La opción `-d` ejecuta el contenedor en modo separado. También es equivalente a `-d=true` .

Un contenedor en modo separado no puede eliminarse automáticamente cuando se detiene, esto significa que no se puede usar la opción --rm en combinación con la opción -d.

Asignar un volumen a un contenedor

Un volumen Docker es un archivo o directorio que persiste más allá de la vida útil del contenedor.

Es posible montar un archivo o directorio de host en un contenedor como un volumen (sin pasar por el UnionFS).

Agregue un volumen con la opción de línea de comando `-v` :

```
docker run -d -v "/data" awesome/app bootstrap.sh
```

Esto creará un volumen y lo montará en la ruta `/data` dentro del contenedor.

- Nota: Puede usar la bandera `--rm` para eliminar automáticamente el volumen cuando se elimina el contenedor.

Montaje de directorios de host

Para montar un archivo o directorio host en un contenedor:

```
docker run -d -v "/home/foo/data:/data" awesome/app bootstrap.sh
```

- **Al especificar un directorio de host, se debe proporcionar una ruta absoluta.**

Esto montará el directorio host `/home/foo/data` en `/data` dentro del contenedor. Este volumen de "directorio de host montado en enlace" es lo mismo que un `mount --bind` Linux `mount --bind` y, por lo tanto, monta temporalmente el directorio de host en la ruta del contenedor especificada durante la vida útil del contenedor. Los cambios en el volumen desde el host o el contenedor se reflejan inmediatamente en el otro, porque son el mismo destino en el disco.

Ejemplo de UNIX montando una carpeta relativa

```
docker run -d -v $(pwd)/data:/data awesome/app bootstrap.sh
```

Nombrar volúmenes

Se puede nombrar un volumen al proporcionar una cadena en lugar de una ruta de directorio de host, la ventana acoplable creará un volumen con ese nombre.

```
docker run -d -v "my-volume:/data" awesome/app bootstrap.sh
```

Después de crear un volumen con nombre, el volumen se puede compartir con otros contenedores con ese nombre.

Configurando variables de entorno

```
$ docker run -e "ENV_VAR=foo" ubuntu /bin/bash
```

Tanto `-e` como `--env` se pueden usar para definir variables de entorno dentro de un contenedor. Es posible suministrar muchas variables de entorno utilizando un archivo de texto:

```
$ docker run --env-file ./env.list ubuntu /bin/bash
```

Ejemplo de archivo de variable de entorno:

```
# This is a comment
TEST_HOST=10.10.0.127
```

El `--env-file` toma un nombre de archivo como argumento y espera que cada línea tenga el formato `VARIABLE=VALUE` , imitando el argumento pasado a `--env` . Las líneas de comentario solo tienen que estar prefijadas con `#` .

Independientemente del orden de estos tres indicadores, el `--env-file` se procesa primero y luego los indicadores `-e` / `--env` . De esta manera, cualquier variable de entorno suministrada individualmente con `-e` o `--env` reemplazará las variables proporcionadas en el archivo de texto `--env-var` .

Especificando un nombre de host

De forma predeterminada, los contenedores creados con la ejecución de la ventana acoplable reciben un nombre de host aleatorio. Puede darle al contenedor un nombre de host diferente pasando la marca --hostname:

```
docker run --hostname redbox -d ubuntu:14.04
```

Ejecutar un contenedor de forma interactiva.

Para ejecutar un contenedor de forma interactiva, pase las opciones `-it` :

```
$ docker run -it ubuntu:14.04 bash
root@8ef2356d919a:/# echo hi
hi
root@8ef2356d919a:/#
```

`-i` mantiene STDIN abierto, mientras que `-t` asigna un pseudo-TTY.

Ejecutar contenedor con memoria / límites de intercambio

Establecer límite de memoria y desactivar el límite de intercambio

```
docker run -it -m 300M --memory-swap -1 ubuntu:14.04 /bin/bash
```

Establecer tanto la memoria como el límite de intercambio. En este caso, el contenedor puede usar 300M de memoria y 700M de intercambio.

```
docker run -it -m 300M --memory-swap 1G ubuntu:14.04 /bin/bash
```

Obtener un shell en un contenedor en ejecución (separado)

Inicie sesión en un contenedor en ejecución

Un usuario puede ingresar un contenedor en ejecución en un nuevo shell de bash interactivo con el comando `exec` .

Supongamos que un contenedor se llama `jovial_morse` luego puede obtener un shell bash pseudo-TTY interactivo ejecutando:

```
docker exec -it jovial_morse bash
```

Inicie sesión en un contenedor en ejecución con un usuario específico

Si desea ingresar un contenedor como usuario específico, puede configurarlo con el parámetro `-u` o `--user` . El nombre de usuario debe existir en el contenedor.

`-u, --user` usuario o UID (formato: `<name|uid>[:<group|gid>]`)

Este comando iniciará sesión en `jovial_morse` con el usuario `dockeruser`

```
docker exec -it -u dockeruser jovial_morse bash
```

Inicie sesión en un contenedor en ejecución como root

Si desea iniciar sesión como root, simplemente use el parámetro **`-u root`** . El usuario root siempre existe.

```
docker exec -it -u root jovial_morse bash
```

Iniciar sesión en una imagen

También puede iniciar sesión en una imagen con el comando de `run` , pero esto requiere un nombre de imagen en lugar de un nombre de contenedor.

```
docker run -it dockerimage bash
```

Iniciar sesión en una imagen intermedia

(depuración)

También puede iniciar sesión en una imagen intermedia, que se crea durante una compilación de Dockerfile.

Salida de `docker build .`

```
$ docker build .
Uploading context 10240 bytes
Step 1 : FROM busybox
Pulling repository busybox
 ---> e9aa60c60128MB/2.284 MB (100%) endpoint: https://cdn-registry-1.docker.io/v1/
Step 2 : RUN ls -lh /
 ---> Running in 9c9e81692ae9
total 24
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 bin
drwxr-xr-x    5 root     root        4.0K Oct 19 00:19 dev
drwxr-xr-x    2 root     root        4.0K Oct 19 00:19 etc
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 lib
lrwxrwxrwx    1 root     root           3 Mar 12  2013 lib64 -> lib
dr-xr-xr-x  116 root     root           0 Nov 15 23:34 proc
lrwxrwxrwx    1 root     root           3 Mar 12  2013 sbin -> bin
dr-xr-xr-x   13 root     root           0 Nov 15 23:34 sys
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 tmp
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 usr
 ---> b35f4035db3f
Step 3 : CMD echo Hello world
 ---> Running in 02071fceb21b
 ---> f52f38b7823e
```

Observe que `---> Running in 02071fceb21b` salida `---> Running in 02071fceb21b` , puede iniciar sesión en estas imágenes:

```
docker run -it 02071fceb21b bash
```

Pasando stdin al contenedor

En casos como la restauración de un volcado de base de datos, o el deseo de enviar información a través de una tubería desde el host, puede usar la `-i` como un argumento para la `docker run` `docker exec` o la `docker run` `docker exec` .

Por ejemplo, suponiendo que desea colocar un volcado de base de datos que tenga en el `dump.sql` en un `dump.sql` , en un archivo local `dump.sql` , puede ejecutar el siguiente comando:

```
docker exec -i mariadb bash -c 'mariadb "-p$MARIADB_PASSWORD" ' < dump.sql
```

En general,

```
docker exec -i container command < file.stdin
```

O

```
docker exec -i container command <<EOF
inline-document-from-host-shell-HEREDOC-syntax
EOF
```

Desprendimiento de un contenedor

Mientras está conectado a un contenedor en ejecución con un pty asignado (`docker run -it ...`), puede presionar `Control P` - `Control Q` para separar.

Anulando la directiva de punto de entrada de imagen

```
docker run --name="test-app" --entrypoint="/bin/bash" example-app
```

Este comando anulará la directiva `ENTRYPOINT` de la imagen de la `example-app` cuando se cree la `test-app` del contenedor. La directiva `CMD` de la imagen no se modificará a menos que se especifique lo contrario:

```
docker run --name="test-app" --entrypoint="/bin/bash" example-app /app/test.sh
```

En el ejemplo anterior, tanto el `ENTRYPOINT` como el `CMD` de la imagen se han anulado. Este proceso contenedor se convierte en `/bin/bash /app/test.sh` .

Añadir entrada de host al contenedor

```
docker run --add-host="app-backend:10.15.1.24" awesome-app
```

Este comando agrega una entrada al `/etc/hosts` del contenedor, que sigue el formato `--add-host <name>:<address>` . En este ejemplo, el nombre de la `app-backend` se resolverá a `10.15.1.24` . Esto es particularmente útil para unir programáticamente diferentes componentes de aplicaciones.

Evitar que el contenedor se detenga cuando no se ejecutan comandos

Un contenedor se detendrá si no se ejecuta ningún comando en el primer plano. El uso de la opción `-t` evitará que el contenedor se detenga, incluso cuando se desconecta con la opción `-d` .

```
docker run -t -d debian bash
```

Parando un contenedor

```
docker stop mynginx
```

Además, la identificación del contenedor también se puede usar para detener el contenedor en lugar de su nombre.

Esto detendrá un contenedor en ejecución enviando la señal SIGTERM y luego la señal SIGKILL si es necesario.

Además, el comando kill se puede usar para enviar inmediatamente un SIGKILL o cualquier otra señal especificada usando la opción `-s` .

```
docker kill mynginx
```

Señal especificada:

```
docker kill -s SIGINT mynginx
```

Detener un contenedor no lo borra. Usa la `docker ps -a` para ver tu contenedor detenido.

Ejecutar otro comando en un contenedor en ejecución

Cuando sea necesario, puede decirle a Docker que ejecute comandos adicionales en un contenedor que ya esté en ejecución usando el comando `exec` . Necesitas la identificación del contenedor que puedes obtener con `docker ps` .

```
docker exec 294fbc4c24b3 echo "Hello World"
```

Puede adjuntar un shell interactivo si usa la opción `-it` .

```
docker exec -it 294fbc4c24b3 bash
```

Ejecutar aplicaciones GUI en un contenedor de Linux

De forma predeterminada, un contenedor Docker no podrá *ejecutar* una aplicación GUI.

Antes de eso, el socket X11 debe reenviarse primero al contenedor, de modo que se pueda usar directamente. La variable de entorno *DISPLAY también se* debe reenviar:

```
docker run -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=unix$DISPLAY <image-name>
```

Esto fallará al principio, ya que no configuramos los permisos para el host del servidor X:

```
cannot connect to X server unix:0
```

La forma más rápida (pero no la más segura) es permitir el acceso directamente con:

```
xhost +local:root
```

Después de terminar con el contenedor, podemos volver al estado original con:

```
xhost -local:root
```

Otra forma (más segura) es preparar un Dockerfile que construirá una nueva imagen que utilizará nuestras credenciales de usuario para acceder al servidor X:

```
FROM <iamge-name>
MAINTAINER <you>

# Arguments picked from the command line!
ARG user
ARG uid
ARG gid

#Add new user with our credentials
ENV USERNAME ${user}
RUN useradd -m $USERNAME && \
        echo "$USERNAME:$USERNAME" | chpasswd && \
        usermod --shell /bin/bash $USERNAME && \
        usermod  --uid ${uid} $USERNAME && \
        groupmod --gid ${gid} $USERNAME

USER ${user}

WORKDIR /home/${user}
```

Al invocar la `docker build` desde la línea de comando, debemos pasar las variables *ARG* que aparecen en el Dockerfile:

```
docker build --build-arg user=$USER --build-arg uid=$(id -u) --build-arg gid=$(id -g) -t <new-image-with-X11-enabled-name> -f <Dockerfile-for-X11> .
```

Ahora, antes de generar un nuevo contenedor, tenemos que crear un archivo xauth con permiso de acceso:

```
xauth nlist $DISPLAY | sed -e 's/^..../ffff/' | xauth -f /tmp/.docker.xauth nmerge -
```

Este archivo debe montarse en el contenedor al crearlo / ejecutarlo:

```
docker run -e DISPLAY=unix$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix -v
/tmp/.docker.xauth:/tmp/.docker.xauth:rw -e XAUTHORITY=/tmp/.docker.xauth
```

Lea Contenedores de correr en línea: https://riptutorial.com/es/docker/topic/679/contenedores-de-correr

Capítulo 10: Creando un servicio con persistencia.

Sintaxis

- docker volume create --name <volume_name> # Crea un volumen llamado <volume_name>
- docker run -v <volume_name>: <mount_point> -d crramirez / limesurvey: latest # Monte el volumen <volume_name> en el directorio <mount_point> en el contenedor

Parámetros

Parámetro	Detalles
--nombre <nombre_volumen>	Especifique el nombre del volumen que se creará
-v <nombre_volumen>: <punto_montaje>	Especifique dónde se montará el volumen nombrado en el contenedor

Observaciones

La persistencia se crea en contenedores docker utilizando volúmenes. Docker tiene muchas formas de lidiar con los volúmenes. Los volúmenes nombrados son muy convenientes por:

- Persisten incluso cuando el contenedor se elimina utilizando la opción -v.
- La única forma de eliminar un volumen con nombre es mediante una llamada explícita al volumen de la ventana acoplable rm
- Los volúmenes nombrados se pueden compartir entre el contenedor sin vincular o la opción --volumes-from.
- No tienen problemas de permisos que tienen los volúmenes montados en el host.
- Se pueden manipular usando el comando de volumen docker.

Examples

Persistencia con volúmenes nombrados.

La persistencia se crea en contenedores docker utilizando volúmenes. Vamos a crear un contenedor Limesurvey y persistir la base de datos, el contenido cargado y la configuración en un volumen con nombre:

```
docker volume create --name mysql
docker volume create --name upload

docker run -d --name limesurvey -v mysql:/var/lib/mysql -v upload:/app/upload -p 80:80
```

```
crramirez/limesurvey:latest
```

Copia de seguridad de un contenido de volumen con nombre

Necesitamos crear un contenedor para montar el volumen. Luego archívalo y descarga el archivo a nuestro anfitrión.

Vamos a crear primero un volumen de datos con algunos datos:

```
docker volume create --name=data
echo "Hello World" |  docker run -i --rm=true -v data:/data ubuntu:trusty tee /data/hello.txt
```

Vamos a hacer una copia de seguridad de los datos:

```
docker run -d --name backup -v data:/data ubuntu:trusty tar -czvf /tmp/data.tgz /data
docker cp backup:/tmp/data.tgz data.tgz
docker rm -fv backup
```

Vamos a probar:

```
tar -xzvf data.tgz
cat data/hello.txt
```

Lea Creando un servicio con persistencia. en línea:
https://riptutorial.com/es/docker/topic/7429/creando-un-servicio-con-persistencia-

Capítulo 11: Depuración de un contenedor

Sintaxis

- docker stats [OPCIONES] [CONTENEDOR ...]
- troncos acoplables [OPCIONES] CONTENEDOR
- tapa acoplable [OPCIONES] CONTENEDOR [ps OPCIONES]

Examples

Entrando en un contenedor corriendo

Para ejecutar operaciones en un contenedor, use el comando `docker exec` . A veces esto se denomina "entrar en el contenedor" ya que todos los comandos se ejecutan dentro del contenedor.

```
docker exec -it container_id bash
```

o

```
docker exec -it container_id /bin/sh
```

Y ahora tienes una cáscara en tu contenedor corriendo. Por ejemplo, liste los archivos en un directorio y luego deje el contenedor:

```
docker exec container_id ls -la
```

Puede usar la `-u flag` para ingresar al contenedor con un usuario específico, por ejemplo, `uid=1013` , `gid=1023` .

```
docker exec -it -u 1013:1023 container_id ls -la
```

El uid y el gid no tienen que existir en el contenedor, pero el comando puede dar como resultado errores. Si desea iniciar un contenedor e ingresar inmediatamente dentro para verificar algo, puede hacer

```
docker run...; docker exec -it $(docker ps -lq) bash
```

el comando `docker ps -lq` solo genera el ID del último contenedor (el l en `-lq`) iniciado. (esto supone que tiene a bash como intérprete disponible en su contenedor, puede tener sh o zsh o cualquier otro)

Monitoreo del uso de recursos

La inspección del uso de los recursos del sistema es una forma eficiente de encontrar

aplicaciones que se comportan mal. Este ejemplo es un equivalente del comando `top` tradicional para contenedores:

```
docker stats
```

Para seguir las estadísticas de contenedores específicos, enumérelos en la línea de comando:

```
docker stats 7786807d8084 7786807d8085
```

Las estadísticas de Docker muestran la siguiente información:

```
CONTAINER     CPU %   MEM USAGE / LIMIT   MEM %    NET I/O               BLOCK I/O
7786807d8084  0.65%   1.33 GB / 3.95 GB   33.67%   142.2 MB / 57.79 MB   46.32 MB / 0 B
```

Por defecto, las `docker stats` muestran la identificación de los contenedores, y esto no es muy útil, si prefiere mostrar los nombres del contenedor, simplemente haga

```
docker stats $(docker ps --format '{{.Names}}')
```

Seguimiento de procesos en un contenedor.

La inspección del uso de los recursos del sistema es una forma eficiente de reducir un problema en una aplicación en ejecución. Este ejemplo es un equivalente del comando `ps` tradicional para contenedores.

```
docker top 7786807d8084
```

Para filtrar el formato de la salida, agregue las opciones `ps` en la línea de comando:

```
docker top 7786807d8084 faux
```

O, para obtener la lista de procesos que se ejecutan como root, que es una práctica potencialmente dañina:

```
docker top 7786807d8084 -u root
```

El comando `docker top` resulta especialmente útil cuando se resuelven contenedores minimalistas sin un shell o el comando `ps` .

Adjuntar a un contenedor corriendo

'Conectarse a un contenedor' es el acto de iniciar una sesión de terminal dentro del contexto en el que se ejecuta el contenedor (y cualquier programa en él). Esto se usa principalmente para fines de depuración, pero también puede ser necesario si se deben pasar datos específicos a programas que se ejecutan dentro del contenedor.

El comando `attach` se utiliza para hacer esto. Tiene esta sintaxis:

```
docker attach <container>
```

`<container>` puede ser el ID del contenedor o el nombre del contenedor. Por ejemplo:

```
docker attach c8a9cf1a1fa8
```

O:

```
docker attach graceful_hopper
```

Es posible que necesite `sudo` los comandos anteriores, dependiendo de su usuario y cómo cargador de muelle está configurado.

> Nota: Adjuntar solo permite que una sola sesión de shell se adjunte a un contenedor a la vez.
>
> Advertencia: *todas las* entradas del teclado se reenviarán al contenedor. Golpear `Ctrl-c` *matará* tu contenedor.

Para desprenderse de un contenedor adjunto, presione sucesivamente `Ctrl-p y` luego `Ctrl-q`

Para adjuntar varias sesiones de shell a un contenedor, o simplemente como alternativa, puede usar `exec` . Usando la identificación del contenedor:

```
docker exec -i -t c8a9cf1a1fa8 /bin/bash
```

Usando el nombre del contenedor:

```
docker exec -i -t graceful_hopper /bin/bash
```

`exec` ejecutará un programa dentro de un contenedor, en este caso `/bin/bash` (un shell, probablemente uno que el contenedor tiene). `-i` indica una sesión interactiva, mientras que `-t` asigna un pseudo-TTY.

> Nota: A diferencia de *adjuntar* , `presionar Ctrl-c` solo terminará el comando *exec* 'd cuando se ejecute de forma interactiva.

Imprimiendo los logs

Seguir los registros es la forma menos intrusiva de depurar una aplicación en ejecución en vivo. Este ejemplo reproduce el comportamiento de la `tail -f some-application.log` tradicional `tail -f some-application.log` en el contenedor `7786807d8084` .

```
docker logs --follow --tail 10 7786807d8084
```

Este comando básicamente muestra la salida estándar del proceso contenedor (el proceso con pid 1).

Si sus registros no incluyen la marca de tiempo de forma nativa, puede agregar la `--timestamps` .

Es posible mirar los registros de un contenedor detenido, ya sea

- iniciar el contenedor `docker run ... ; docker logs $(docker ps -lq)` con la `docker run ... ; docker logs $(docker ps -lq)`
- Encuentra la identificación del contenedor o el nombre con

```
docker ps -a
```

y entonces

`docker logs container-id` o

`docker logs containername`

Como es posible mirar los registros de un contenedor detenido

Depuración del proceso de contenedor Docker

Docker es solo una forma elegante de ejecutar un proceso, no una máquina virtual. Por lo tanto, la depuración de un proceso "en un contenedor" también es posible "en el host" simplemente examinando el proceso contenedor en ejecución como un usuario con los permisos adecuados para inspeccionar esos procesos en el host (por ejemplo, root). Por ejemplo, es posible enumerar cada "proceso contenedor" en el host ejecutando un simple `ps` como root:

```
sudo ps aux
```

Todos los contenedores Docker que se estén ejecutando actualmente se mostrarán en la salida.

Esto puede ser útil durante el desarrollo de la aplicación para depurar un proceso que se ejecuta en un contenedor. Como usuario con los permisos adecuados, se pueden usar utilidades de depuración típicas en el proceso de contenedor, como strace, ltrace, gdb, etc.

Lea Depuración de un contenedor en línea:
https://riptutorial.com/es/docker/topic/1333/depuracion-de-un-contenedor

Capítulo 12: Docker - Modos de red (puente, zonas activas, contenedor asignado y ninguno).

Introducción

Empezando

Modo de puente Es un valor predeterminado y se adjunta al puente docker0. Coloque el contenedor en un espacio de nombres de red completamente separado.

Modo host Cuando el contenedor es solo un proceso que se ejecuta en un host, adjuntaremos el contenedor a la NIC del host.

Modo de contenedor asignado Este modo esencialmente mapea un nuevo contenedor en una pila de red de contenedores existente. También se le llama 'contenedor en modo contenedor'.

Ninguno Le dice a la ventana acoplable que coloque el contenedor en su propia pila de red sin configuración

Examples

Modo de puente, modo de host y modo de contenedor asignado

Modo Puente

```
$ docker run -d --name my_app -p 10000:80 image_name
```

Tenga en cuenta que no tuvimos que especificar **--net = bridge** porque este es el modo de trabajo predeterminado para la ventana acoplable. Esto permite ejecutar múltiples contenedores para ejecutarse en el mismo host sin ninguna asignación de puerto dinámico. Por lo tanto, el modo **BRIDGE** evita el conflicto de puertos y es seguro ya que cada contenedor está ejecutando su propio espacio de nombres de red privada.

Modo host

```
$ docker run -d --name my_app -net=host image_name
```

Como utiliza el espacio de nombres de la red del host, no es necesaria una configuración especial, pero puede provocar problemas de seguridad.

Modo de contenedor mapeado

Este modo esencialmente asigna un nuevo contenedor a una pila de red de contenedores

existente. Esto implica que los recursos de red como la dirección IP y las asignaciones de puertos del primer contenedor serán compartidos por el segundo contenedor. Esto también se denomina modo 'contenedor en contenedor'. Supongamos que tiene dos atributos como web_container_1 y web_container_2 y ejecutaremos web_container_2 en el modo de contenedor asignado. Primero descarguemos web_container_1 y lo ejecutemos en modo separado con el siguiente comando,

```
$ docker run -d --name web1 -p 80:80 USERNAME/web_container_1
```

Una vez descargado, echemos un vistazo y asegurémonos de que está funcionando. Aquí simplemente asignamos un puerto a un contenedor que se ejecuta en el modo de puente predeterminado. Ahora, vamos a ejecutar un segundo contenedor en el modo de contenedor asignado. Lo haremos con este comando.

```
$ docker run -d --name web2 --net=container:web1 USERNAME/web_container_2
```

Ahora, si simplemente obtiene la información de la interfaz en ambos contaienrs, obtendrá la misma configuración de red. En realidad, esto incluye el modo HOST que se mapea con la información exacta del host. El primer contaienr se ejecutó en el modo de puente predeterminado y el segundo contenedor se está ejecutando en el modo de contenedor asignado. Podemos obtener resultados muy similares iniciando el primer contenedor en el modo host y el segundo contenedor en el modo contenedor asignado.

Lea Docker - Modos de red (puente, zonas activas, contenedor asignado y ninguno). en línea: https://riptutorial.com/es/docker/topic/9643/docker---modos-de-red--puente--zonas-activas--contenedor-asignado-y-ninguno--

Capítulo 13: Docker en Docker

Examples

Jenkins CI Container utilizando Docker

Este capítulo describe cómo configurar un contenedor Docker con Jenkins dentro, que es capaz de enviar comandos Docker a la instalación Docker (el demonio Docker) del host. Efectivamente utilizando Docker en Docker. Para lograr esto, tenemos que construir una imagen Docker personalizada que se base en una versión arbitraria de la imagen oficial de Jenkins Docker. El archivo Docker (la instrucción de cómo construir la imagen) se ve así:

```
FROM jenkins

USER root

RUN cd /usr/local/bin && \
curl https://master.dockerproject.org/linux/amd64/docker > docker  && \
chmod +x docker  && \
groupadd -g 999 docker && \
usermod -a -G docker jenkins

USER Jenkins
```

Este Dockerfile construye una imagen que contiene los archivos binarios del cliente Docker, este cliente se utiliza para comunicarse con un demonio Docker. En este caso el demonio Docker del host. La instrucción `RUN` en este archivo también crea un grupo de usuarios UNIX con el UID 999 y le agrega el usuario Jenkins. Por qué exactamente esto es necesario se describe en el capítulo siguiente. Con esta imagen podemos ejecutar un servidor Jenkins que puede usar los comandos de Docker, pero si solo ejecutamos esta imagen, el cliente de Docker que instalamos dentro de la imagen no se puede comunicar con el demonio Docker del host. Estos dos componentes se comunican a través de un zócalo UNIX `/var/run/docker.sock` . En Unix, este es un archivo como todo lo demás, por lo que podemos montarlo fácilmente dentro del Contenedor Jenkins. Esto se hace con el comando `docker run -v /var/run/docker.sock:/var/run/docker.sock --name jenkins MY_CUSTOM_IMAGE_NAME` . Pero este archivo montado es propiedad de `docker:root` y debido a esto, Dockerfile crea este grupo con un UID conocido y agrega al usuario Jenkins. Ahora es el contenedor Jenkins realmente capaz de ejecutar y usar Docker. En producción, el comando de ejecución también debe contener `-v jenkins_home:/var/jenkins_home` para hacer una copia de seguridad del directorio Jenkins_home y, por supuesto, una asignación de puertos para acceder al servidor a través de la red.

Lea Docker en Docker en línea: https://riptutorial.com/es/docker/topic/8012/docker-en-docker

Capítulo 14: Docker Machine

Introducción

Gestión remota de múltiples hosts del motor docker.

Observaciones

`docker-machine` gestiona hosts remotos ejecutando Docker.

La herramienta de línea de comandos `docker-machine` administra el ciclo de vida completo de la máquina utilizando controladores específicos del proveedor. Se puede utilizar para seleccionar una máquina "activa". Una vez seleccionado, una máquina activa se puede usar como si fuera el motor Docker local.

Examples

Obtener información actual del entorno de Docker Machine

Todos estos son comandos de shell.

`docker-machine env` para obtener la configuración predeterminada actual de docker-machine

`eval $(docker-machine env)` para obtener la configuración actual de docker-machine y configurar el entorno de shell actual para usar esta docker-machine con.

Si su shell está configurado para usar un proxy, puede especificar la opción --no-proxy para omitir el proxy al conectarse a su docker-machine: `eval $(docker-machine env --no-proxy)`

Si tiene varias máquinas docker, puede especificar el nombre de la máquina como argumento: `eval $(docker-machine env --no-proxy machinename)`

SSH en una máquina docker

Todos estos son comandos de shell.

- Si necesita iniciar sesión en una máquina acoplable en ejecución directamente, puede hacerlo:

`docker-machine ssh` a ssh en el docker-machine predeterminado

`docker-machine ssh machinename` a ssh en un docker-machine no predeterminado

- Si solo desea ejecutar un solo comando, puede hacerlo. Para ejecutar el `uptime` de `uptime` en el docker-machine predeterminado para ver cuánto tiempo ha estado funcionando, ejecute el `docker-machine ssh default uptime`

Crear una máquina Docker

El uso de `docker-machine` es el mejor método para instalar Docker en una máquina. Se aplicará automáticamente la mejor configuración de seguridad disponible, incluida la generación de un par único de certificados SSL para la autenticación mutua y las claves SSH.

Para crear una máquina local usando Virtualbox:

```
docker-machine create --driver virtualbox docker-host-1
```

Para instalar Docker en una máquina existente, use el controlador `generic` :

```
docker-machine -D create -d generic --generic-ip-address 1.2.3.4 docker-host-2
```

La opción `--driver` le dice a la `--driver` acoplable cómo crear la máquina. Para obtener una lista de los controladores compatibles, consulte:

- apoyado oficialmente
- tercero

Lista de máquinas portuarias

El listado de máquinas acopladoras devolverá el estado, la dirección y la versión de Docker de cada una de las máquinas acopladoras.

```
 docker-machine ls
```

Imprimirá algo como:

```
NAME             ACTIVE   DRIVER    STATE     URL                         SWARM   DOCKER
ERRORS
docker-machine-1 -        ovh       Running   tcp://1.2.3.4:2376                  v1.11.2
docker-machine-2 -        generic   Running   tcp://1.2.3.5:2376                  v1.11.2
```

Para listar máquinas en funcionamiento:

```
docker-machine ls --filter state=running
```

Para listar máquinas de error:

```
docker-machine ls --filter state=
```

Para enumerar las máquinas cuyo nombre comienza con 'side-project-', use el filtro de Golang:

```
docker-machine ls --filter name="^side-project-"
```

Para obtener solo la lista de URL de la máquina:

```
docker-machine ls --format '{{ .URL }}'
```

Consulte https://docs.docker.com/machine/reference/ls/ para ver la referencia completa del comando.

Actualizar una máquina Docker

La actualización de una máquina acoplable implica un tiempo de inactividad y puede requerir un cepillado. Para actualizar una máquina docker, ejecute:

```
docker-machine upgrade docker-machine-name
```

Este comando no tiene opciones.

Obtener la dirección IP de una máquina docker

Para obtener la dirección IP de una máquina docker, puede hacerlo con este comando:

```
docker-machine ip machine-name
```

Lea Docker Machine en línea: https://riptutorial.com/es/docker/topic/1349/docker-machine

Capítulo 15: Docker registro privado / seguro con API v2

Introducción

Un registro de docker privado y seguro en lugar de un Docker Hub. Se requieren habilidades básicas de docker.

Parámetros

Mando	Explicación
sudo docker run -p 5000: 5000	Inicie un contenedor de la ventana acoplable y enlace el puerto 5000 del contenedor al puerto 5000 de la máquina física.
- registro de nombres	Nombre del contenedor (se utiliza para mejorar la legibilidad de "docker ps").
-v 'pwd' / certs: / certs	Enlace CURRENT_DIR / certs de la máquina física en / certs del contenedor (como una "carpeta compartida").
-e REGISTRY_HTTP_TLS_CERTIFICATE = / certs / server.crt	Especificamos que el registro debe usar el archivo /certs/server.crt para comenzar. (variable env)
-e REGISTRY_HTTP_TLS_KEY = / certs / server.key	Lo mismo para la clave RSA (server.key).
-v / root / images: / var / lib / registry /	Si desea guardar todas las imágenes de registro, debe hacer esto en la máquina física. Aquí guardamos todas las imágenes en / root / images en la máquina física. Si hace esto, puede detener y reiniciar el registro sin perder ninguna imagen.
registro: 2	Especificamos que nos gustaría extraer la imagen de registro desde el concentrador de la ventana acoplable (o localmente), y agregamos «2» porque queremos instalar la versión 2 del registro.

Observaciones

Cómo instalar un motor docker (llamado cliente en este tutorial)

Cómo generar un certificado autofirmado SSL

Examples

Generando certificados

Genere una clave privada RSA: `openssl genrsa -des3 -out server.key 4096`

Openssl debe solicitar una frase de contraseña en este paso. Tenga en cuenta que solo usaremos el certificado para la comunicación y la autenticación, sin contraseña. Simplemente use 123456 por ejemplo.

Genere la solicitud de firma de certificado: `openssl req -new -key server.key -out server.csr`

Este paso es importante porque se le pedirá información sobre los certificados. La información más importante es "Nombre común", que es el nombre de dominio, que se utiliza para la comunicación entre el registro de docker privado y el resto de la máquina. Ejemplo: mydomain.com

Elimine la frase de acceso de la clave privada RSA: `cp server.key server.key.org && openssl rsa -in server.key.org -out server.key`

Como he dicho, nos centraremos en el certificado sin contraseña. Así que tenga cuidado con todos los archivos de su clave (.key, .csr, .crt) y manténgalos en un lugar seguro.

Genere el certificado autofirmado: `openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt`

Ahora tiene dos archivos esenciales, *server.key* y *server.crt* , que son necesarios para la autenticación del registro privado.

Ejecutar el registro con certificado autofirmado.

Para ejecutar el registro privado (de forma segura) tiene que generar un certificado autofirmado, puede consultar el ejemplo anterior para generarlo.

Para mi ejemplo, puse *server.key* y *server.crt* en / root / certs

Antes de ejecutar el comando docker, debe colocarse (use `cd`) en el directorio que contiene la carpeta *certs* . Si no lo eres e intentas ejecutar el comando, recibirás un error como

> level = fatal msg = "abrir /certs/server.crt: no existe tal archivo o directorio"

Cuando esté (`cd /root` en mi ejemplo), básicamente puede iniciar el registro seguro / privado usando: `sudo docker run -p 5000:5000 --restart=always --name registry -v `pwd`/certs:/certs -e`

`REGISTRY_HTTP_TLS_CERTIFICATE=/certs/server.crt -e REGISTRY_HTTP_TLS_KEY=/certs/server.key -v /root/Documents:/var/lib/registry/ registry:2`

Las explicaciones sobre el comando están disponibles en la parte Parámetros.

Tire o empuje de un cliente docker

Cuando se ejecuta un registro operativo, puede extraer o insertar imágenes en él. Para eso necesita el archivo *server.crt* en una carpeta especial en su cliente docker. El certificado le permite autenticarse con el registro y luego cifrar la comunicación.

Copie *server.crt* de la máquina de registro en /etc/docker/certs.d/mydomain.com:5000/ en su máquina cliente. Y luego `mv /etc/docker/certs.d/mydomain.com:5000/server.crt /etc/docker/certs.d/mydomain.com:5000/ca-certificates.crt` a *ca-certificados.crt* : `mv /etc/docker/certs.d/mydomain.com:5000/server.crt /etc/docker/certs.d/mydomain.com:5000/ca-certificates.crt`

En este punto, puede extraer o insertar imágenes de su registro privado:
PULL: `docker pull mydomain.com:5000/nginx` o
EMPUJAR :

1. Obtenga una imagen oficial de hub.docker.com: `docker pull nginx`
2. Etiqueta esta imagen antes de `docker tag IMAGE_ID mydomain.com:5000/nginx` en el registro privado: `docker tag IMAGE_ID mydomain.com:5000/nginx` (usa `docker images` para obtener el IMAGE_ID)
3. Empuje la imagen al registro: `docker push mydomain.com:5000/nginx`

Lea Docker registro privado / seguro con API v2 en línea:
https://riptutorial.com/es/docker/topic/8707/docker-registro-privado---seguro-con-api-v2

Capítulo 16: Docker stats todos los contenedores en ejecución

Examples

Docker stats todos los contenedores en ejecución

```
sudo docker stats $(sudo docker inspect -f "{{ .Name }}" $(sudo docker ps -q))
```

Muestra el uso en vivo de la CPU de todos los contenedores en ejecución.

Lea Docker stats todos los contenedores en ejecución en línea:
https://riptutorial.com/es/docker/topic/5863/docker-stats-todos-los-contenedores-en-ejecucion

Capítulo 17: Dockerfiles

Introducción

Los Dockerfiles son archivos que se utilizan para crear imágenes Docker mediante programación. Le permiten crear de forma rápida y reproducible una imagen de Docker, por lo que son útiles para colaborar. Dockerfiles contiene instrucciones para construir una imagen de Docker. Cada instrucción se escribe en una fila y se da en la forma `<INSTRUCTION><argument(s)>` . Los Dockerfiles se utilizan para crear imágenes de Docker utilizando el comando de `docker build` ventana `docker build` .

Observaciones

Dockerfiles son de la forma:

```
# This is a comment
INSTRUCTION arguments
```

- Los comentarios comienzan con un `#`
- Las instrucciones son solo mayúsculas
- La primera instrucción de un Dockerfile debe ser `FROM` para especificar la imagen base

---

Al crear un Dockerfile, el cliente Docker enviará un "contexto de compilación" al demonio Docker. El contexto de compilación incluye todos los archivos y carpetas en el mismo directorio que Dockerfile. `COPY` operaciones `COPY` y `ADD` solo pueden usar archivos de este contexto.

Algunos archivos de Docker pueden comenzar con:

```
# escape=`
```

Esto se utiliza para indicar al analizador Docker que use `` ` `` como un carácter de escape en lugar de `\` . Esto es principalmente útil para archivos de Windows Docker.

Examples

HelloWorld Dockerfile

Un Dockerfile mínimo se ve así:

```
FROM alpine
CMD ["echo", "Hello StackOverflow!"]
```

Esto le indicará a Docker que genere una imagen basada en Alpine (`FROM`), una distribución

mínima para contenedores, y que ejecute un comando específico (`CMD`) al ejecutar la imagen resultante.

Constrúyelo y ejecútalo:

```
docker build -t hello .
docker run --rm hello
```

Esto dará como resultado:

```
Hello StackOverflow!
```

Copiando documentos

Para copiar archivos del contexto de compilación en una imagen de Docker, use la instrucción `COPY` :

```
COPY localfile.txt containerfile.txt
```

Si el nombre del archivo contiene espacios, use la sintaxis alternativa:

```
COPY ["local file", "container file"]
```

El comando `COPY` soporta comodines. Se puede usar, por ejemplo, para copiar todas las imágenes al directorio `images/` :

```
COPY *.jpg images/
```

Nota: en este ejemplo, las `images/` pueden no existir. En este caso, Docker lo creará automáticamente.

Exponiendo un puerto

Para declarar puertos expuestos desde un Dockerfile use la instrucción `EXPOSE` :

```
EXPOSE 8080 8082
```

La configuración de puertos expuestos puede anularse desde la línea de comandos de Docker, pero es una buena práctica establecerlos explícitamente en el Dockerfile ya que ayuda a entender lo que hace una aplicación.

Dockerfiles mejores practicas

Operaciones comunes de grupo

Docker construye imágenes como una colección de capas. Cada capa solo puede agregar datos, incluso si estos datos indican que un archivo se ha eliminado. Cada instrucción crea una nueva

capa. Por ejemplo:

```
RUN apt-get -qq update
RUN apt-get -qq install some-package
```

Tiene un par de inconvenientes:

- Creará dos capas, produciendo una imagen más grande.
- El uso de `apt-get update` solo en una instrucción `RUN` causa problemas de almacenamiento en caché y, posteriormente `apt-get install` instrucciones de `apt-get install` pueden **fallar** . Supongamos que luego modifica `apt-get install` agregando paquetes adicionales, la ventana acoplable interpreta las instrucciones iniciales y modificadas como idénticas y reutiliza el caché de los pasos anteriores. Como resultado, el comando `apt-get update` **no se** ejecuta porque su versión en caché se usa durante la compilación.

En su lugar, utilice:

```
RUN apt-get -qq update && \
    apt-get -qq install some-package
```

Como esto solo produce una capa.

Mencionar al mantenedor

Esta suele ser la segunda línea del archivo Docker. Indica quién está a cargo y podrá ayudar.

```
LABEL maintainer John Doe <john.doe@example.com>
```

Si lo saltas, no romperá tu imagen. Pero tampoco ayudará a tus usuarios.

Sé conciso

Mantenga su Dockerfile corto. Si es necesaria una configuración compleja, considere usar un script dedicado o configurar imágenes base.

Instrucción de usuario

```
USER daemon
```

La instrucción `USER` establece el nombre de usuario o UID que se usará cuando se ejecute la imagen y para cualquier instrucción `RUN` , `CMD` y `ENTRYPOINT` que la siga en el `Dockerfile` .

Instrucciones de trabajo

```
WORKDIR /path/to/workdir
```

La instrucción `WORKDIR` establece el directorio de trabajo para las instrucciones `RUN` , `CMD` , `ENTRYPOINT` , `COPY` y `ADD` que lo siguen en el Dockerfile. Si el `WORKDIR` no existe, se creará incluso si no se utiliza

en ninguna instrucción de `Dockerfile` posterior.

Se puede usar varias veces en un `Dockerfile` . Si se proporciona una ruta relativa, será relativa a la ruta de la instrucción anterior `WORKDIR` . Por ejemplo:

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

La salida del comando `pwd` final en este `Dockerfile` sería `/a/b/c` .

La instrucción `WORKDIR` puede resolver las variables de entorno previamente establecidas utilizando `ENV` . Solo puede usar variables de entorno establecidas explícitamente en el `Dockerfile` . Por ejemplo:

```
ENV DIRPATH /path
WORKDIR $DIRPATH/$DIRNAME
RUN pwd
```

La salida del comando `pwd` final en este Dockerfile sería `/path/$DIRNAME`

Instrucción de volumen

```
VOLUME ["/data"]
```

La instrucción `VOLUME` crea un punto de montaje con el nombre especificado y lo marca como que contiene volúmenes montados externamente desde el host nativo u otros contenedores. El valor puede ser una matriz JSON, `VOLUME ["/var/log/"]` , o una cadena simple con múltiples argumentos, como `VOLUME /var/log` o `VOLUME /var/log /var/db` . Para obtener más información / ejemplos e instrucciones de montaje a través del cliente Docker, consulte Compartir directorios a través de la documentación de Volumes.

El comando de `docker run` inicializa el volumen recién creado con cualquier dato que exista en la ubicación especificada dentro de la imagen base. Por ejemplo, considere el siguiente fragmento de Dockerfile:

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

Este Dockerfile da como resultado una imagen que causa la ejecución de la ventana acoplable, para crear un nuevo punto de montaje en / myvol y copiar el archivo de saludo en el volumen recién creado.

Nota: Si algún paso de compilación cambia los datos dentro del volumen después de que se haya declarado, esos cambios se descartarán.

Nota: la lista se analiza como una matriz JSON, lo que significa que debe usar comillas dobles (“) alrededor de palabras no comillas simples (').

Instrucciones de copia

`COPY` tiene dos formas:

```
COPY <src>... <dest>
COPY ["<src>",... "<dest>"] (this form is required for paths containing whitespace)
```

La instrucción `COPY` copia los archivos o directorios nuevos desde `<src>` y los agrega al sistema de archivos del contenedor en la ruta `<dest>` .

Se pueden especificar varios recursos `<src>` pero deben ser relativos al directorio de origen que se está construyendo (el contexto de la compilación).

Cada `<src>` puede contener comodines y la coincidencia se realizará utilizando las reglas de la `filepath.Match` de `filepath.Match` de Go `filepath.Match` . Por ejemplo:

```
COPY hom* /mydir/        # adds all files starting with "hom"
COPY hom?.txt /mydir/    # ? is replaced with any single character, e.g., "home.txt"
```

El `<dest>` es una ruta absoluta, o una ruta relativa a `WORKDIR` , en la que se copiará la fuente dentro del contenedor de destino.

```
COPY test relativeDir/   # adds "test" to `WORKDIR`/relativeDir/
COPY test /absoluteDir/  # adds "test" to /absoluteDir/
```

Todos los archivos y directorios nuevos se crean con un UID y GID de 0.

Nota: Si `docker build - < somefile` utilizando stdin (`docker build - < somefile`), no hay un contexto de compilación, por lo que no se puede usar `COPY` .

`COPY` obedece las siguientes reglas:

- La ruta `<src>` debe estar dentro del contexto de la compilación; no puede `COPY` ../algo / algo, porque el primer paso de una construcción de ventana acoplable es enviar el directorio de contexto (y los subdirectorios) al demonio de la ventana acoplable.
- Si `<src>` es un directorio, se copia todo el contenido del directorio, incluidos los metadatos del sistema de archivos. Nota: el directorio en sí no se copia, solo su contenido.
- Si `<src>` es cualquier otro tipo de archivo, se copia individualmente junto con sus metadatos. En este caso, si `<dest>` finaliza con una barra inclinada /, se considerará un directorio y el contenido de `<src>` se escribirá en `<dest>/base(<src>)` .
- Si se especifican múltiples recursos `<src>` , ya sea directamente o debido al uso de un comodín, entonces `<dest>` debe ser un directorio, y debe terminar con una barra oblicua `/` .

- Si `<dest>` no finaliza con una barra diagonal, se considerará un archivo normal y el contenido de `<src>` se escribirá en `<dest>` .
- Si `<dest>` no existe, se crea junto con todos los directorios faltantes en su ruta.

Las instrucciones ENV y ARG

ENV

```
ENV <key> <value>
ENV <key>=<value> ...
```

La instrucción `ENV` establece la variable de entorno `<key>` al valor. Este valor estará en el entorno de todos los comandos de Dockerfile "descendientes" y se puede reemplazar en línea en muchos también.

La instrucción `ENV` tiene dos formas. La primera forma, `ENV <key> <value>` , establecerá una única variable en un valor. La cadena completa después del primer espacio se tratará como el `<value>` , incluidos caracteres como espacios y comillas.

La segunda forma, `ENV <key>=<value> ...` , permite configurar múltiples variables a la vez. Observe que la segunda forma usa el signo igual (=) en la sintaxis, mientras que la primera forma no lo hace. Al igual que el análisis de líneas de comando, se pueden usar comillas y barras invertidas para incluir espacios dentro de los valores.

Por ejemplo:

```
ENV myName="John Doe" myDog=Rex\ The\ Dog \
    myCat=fluffy
```

y

```
ENV myName John Doe
ENV myDog Rex The Dog
ENV myCat fluffy
```

arrojará los mismos resultados netos en el contenedor final, pero se prefiere la primera forma porque produce una sola capa de caché.

Las variables de entorno establecidas con `ENV` persistirán cuando se ejecute un contenedor desde la imagen resultante. Puede ver los valores usando la ventana acoplable inspeccionar, y cambiarlos usando la `docker run --env <key>=<value>` .

ARG

Si no desea mantener la configuración, use `ARG` lugar. `ARG` establecerá entornos solo durante la construcción. Por ejemplo, la configuración

```
ENV DEBIAN_FRONTEND noninteractive
```

puede confundir a `apt-get` usuarios de `apt-get` en una imagen basada en Debian cuando entran al contenedor en un contexto interactivo a través de la `docker exec -it the-container bash` .

En su lugar, utilice:

```
ARG DEBIAN_FRONTEND noninteractive
```

Alternativamente, también puede establecer un valor para un solo comando utilizando:

```
RUN <key>=<value> <command>
```

Exponer instrucción

```
EXPOSE <port> [<port>...]
```

La instrucción `EXPOSE` informa a Docker que el contenedor escucha en los puertos de red especificados en tiempo de ejecución. `EXPOSE` NO hace que los puertos del contenedor sean accesibles para el host. Para hacerlo, debe usar el indicador `-p` para publicar un rango de puertos o el indicador `-P` para publicar todos los puertos expuestos. Estos indicadores se utilizan en la `docker run [OPTIONS] IMAGE [COMMAND][ARG...]` para exponer el puerto al host. Puede exponer un número de puerto y publicarlo externamente bajo otro número.

```
docker run -p 2500:80 <image name>
```

Este comando creará un contenedor con el nombre <image> y vinculará el puerto 80 del contenedor al puerto 2500 de la máquina host.

Para configurar la redirección de puertos en el sistema host, consulte el uso del indicador `-P` . La función de red de Docker admite la creación de redes sin la necesidad de exponer puertos dentro de la red; para obtener información detallada, consulte la descripción general de esta función).

Etiqueta de instrucciones

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

La instrucción `LABEL` agrega metadatos a una imagen. Una `LABEL` es un par clave-valor. Para incluir espacios dentro de un valor de `LABEL` , use comillas y barras invertidas como lo haría en el análisis de línea de comandos. Algunos ejemplos de uso:

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
LABEL version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines."
```

Una imagen puede tener más de una etiqueta. Para especificar múltiples etiquetas, Docker recomienda combinar etiquetas en una sola instrucción de `LABEL` siempre que sea posible. Cada instrucción de `LABEL` produce una nueva capa que puede resultar en una imagen ineficiente si usa muchas etiquetas. Este ejemplo da como resultado una capa de imagen única.

```
LABEL multi.label1="value1" multi.label2="value2" other="value3"
```

Lo anterior también se puede escribir como:

```
LABEL multi.label1="value1" \
      multi.label2="value2" \
      other="value3"
```

Las etiquetas son aditivas, incluidas las `LABEL` en las imágenes `FROM` . Si Docker encuentra una etiqueta / clave que ya existe, el nuevo valor anula cualquier etiqueta anterior con claves idénticas.

Para ver las etiquetas de una imagen, use el comando de inspección de la ventana acoplable.

```
"Labels": {
    "com.example.vendor": "ACME Incorporated"
    "com.example.label-with-value": "foo",
    "version": "1.0",
    "description": "This text illustrates that label-values can span multiple lines.",
    "multi.label1": "value1",
    "multi.label2": "value2",
    "other": "value3"
},
```

Instrucción CMD

La instrucción `CMD` tiene tres formas:

```
CMD ["executable","param1","param2"] (exec form, this is the preferred form)
CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
CMD command param1 param2 (shell form)
```

Solo puede haber una instrucción `CMD` en un `Dockerfile` . Si enumera más de un `CMD` , solo el último `CMD` tendrá efecto.

El propósito principal de un `CMD` es proporcionar valores predeterminados para un contenedor en ejecución. Estos valores predeterminados pueden incluir un ejecutable, o pueden omitir el ejecutable, en cuyo caso debe especificar también una instrucción `ENTRYPOINT` .

Nota: Si se utiliza `CMD` para proporcionar argumentos predeterminados para la instrucción `ENTRYPOINT` , tanto las instrucciones `CMD` como `ENTRYPOINT` deben especificarse con el formato de matriz JSON.

Nota: la forma ejecutiva se analiza como una matriz JSON, lo que significa que debe usar comillas dobles (“) alrededor de palabras, no comillas simples (').

Nota: A diferencia del formulario de shell, el formulario exec no invoca un shell de comando. Esto significa que el procesamiento de shell normal no ocurre. Por ejemplo, `CMD [ "echo", "$HOME" ]` no realizará la sustitución de variables en `$HOME` . Si desea el procesamiento de shell, use el formulario de shell o ejecute un shell directamente, por ejemplo: `CMD [ "sh", "-c", "echo $HOME" ]` .

Cuando se utiliza en los formatos de shell o exec, la instrucción `CMD` establece el comando que se ejecutará al ejecutar la imagen.

Si usa el formulario de shell de la `CMD` , el comando se ejecutará en `/bin/sh -c` :

```
FROM ubuntu
CMD echo "This is a test." | wc -
```

Si desea ejecutar su comando sin un shell, debe expresar el comando como una matriz JSON y dar la ruta completa al ejecutable. Esta forma de matriz es el formato preferido de `CMD` . Cualquier parámetro adicional debe expresarse individualmente como cadenas en la matriz:

```
FROM ubuntu
CMD ["/usr/bin/wc","--help"]
```

Si desea que su contenedor ejecute el mismo ejecutable cada vez, debe considerar utilizar `ENTRYPOINT` en combinación con `CMD` . Ver `ENTRYPOINT` .

Si el usuario especifica argumentos para la ejecución de la ventana acoplable, anulará el valor predeterminado especificado en `CMD` .

Nota: no confundas `RUN` con `CMD` . `RUN` realmente ejecuta un comando en el momento de crear la imagen y confirma el resultado; `CMD` no ejecuta nada en el momento de la compilación, pero especifica el comando deseado para la imagen.

MAINTAINER Instrucción

```
MAINTAINER <name>
```

La instrucción `MAINTAINER` le permite configurar el campo Autor de las imágenes generadas.

NO UTILICE LA DIRECTIVA DE MANTENIMIENTO

Según la documentación oficial de Docker, la instrucción `MAINTAINER` está en desuso. En su lugar, se debe usar la instrucción `LABEL` para definir el autor de las imágenes generadas. La instrucción `LABEL` es más flexible, permite configurar metadatos y puede verse fácilmente con el comando `docker inspect` .

```
LABEL maintainer="someone@something.com"
```

De instrucción

```
FROM <image>
```

O

```
FROM <image>:<tag>
```

O

```
FROM <image>@<digest>
```

La instrucción `FROM` establece la imagen base para instrucciones posteriores. Como tal, un archivo Docker válido debe tener `FROM` como primera instrucción. La imagen puede ser cualquier imagen válida; es especialmente fácil comenzar por extraer una imagen de los repositorios públicos.

`FROM` debe ser la primera instrucción sin comentarios en el Dockerfile.

`FROM` puede aparecer varias veces dentro de un único archivo Docker para crear múltiples imágenes. Simplemente tome nota de la salida de la última ID de imagen por la confirmación antes de cada nuevo comando `FROM` .

Los valores de etiqueta o compendio son opcionales. Si omite alguno de ellos, el constructor asume un último por defecto. El constructor devuelve un error si no puede coincidir con el valor de la etiqueta.

RUN Instrucción

`RUN` tiene 2 formas:

```
RUN <command> (shell form, the command is run in a shell, which by default is /bin/sh -c on
Linux or cmd /S /C on Windows)
RUN ["executable", "param1", "param2"] (exec form)
```

La instrucción `RUN` ejecutará cualquier comando en una nueva capa sobre la imagen actual y confirmará los resultados. La imagen confirmada resultante se utilizará para el siguiente paso en el `Dockerfile` .

Las instrucciones `RUN` capas y los compromisos de generación se ajustan a los conceptos centrales de Docker, donde los compromisos son baratos y se pueden crear contenedores desde cualquier punto en el historial de una imagen, al igual que el control de código fuente.

La forma exec hace que sea posible evitar munging cáscara de la secuencia, y para `RUN` comandos usando una imagen de base que no contiene el ejecutable shell especificado.

El shell predeterminado para el formulario de shell se puede cambiar usando el comando `SHELL` .

En el formulario de shell, puede usar una `\` (barra invertida) para continuar una instrucción `RUN` en la siguiente línea. Por ejemplo, considera estas dos líneas:

```
RUN /bin/bash -c 'source $HOME/.bashrc ;\
echo $HOME'
```

Juntos son equivalentes a esta sola línea:

```
RUN /bin/bash -c 'source $HOME/.bashrc ; echo $HOME'
```

Nota: Para usar un shell diferente, que no sea '/ bin / sh', use el formulario exec que pasa en el shell deseado. Por ejemplo, `RUN ["/bin/bash", "-c", "echo hello"]`

Nota: la forma ejecutiva se analiza como una matriz JSON, lo que significa que debe usar comillas dobles (") alrededor de palabras, no comillas simples (').

Nota: A diferencia del formulario de shell, el formulario exec no invoca un shell de comando. Esto significa que el procesamiento de shell normal no ocurre. Por ejemplo, `RUN [ "echo", "$HOME" ]` no realizará la sustitución de variables en `$HOME` . Si desea el procesamiento de shell, use el formulario de shell o ejecute un shell directamente, por ejemplo: `RUN [ "sh", "-c", "echo $HOME" ]` .

Nota: En el formulario JSON, es necesario escapar de barras invertidas. Esto es particularmente relevante en Windows donde la barra diagonal inversa es el separador de ruta. De lo contrario, la siguiente línea se trataría como un formulario de shell debido a que no es JSON válido y fallará de una manera inesperada: `RUN ["c:\windows\system32\tasklist.exe"]`

La sintaxis correcta para este ejemplo es: `RUN ["c:\\windows\\system32\\tasklist.exe"]`

La memoria caché para las instrucciones `RUN` no se invalida automáticamente durante la siguiente compilación. La memoria caché para una instrucción como `RUN apt-get dist-upgrade -y` se reutilizará durante la siguiente compilación. La memoria caché para las instrucciones `RUN` se puede invalidar mediante el uso del indicador --no-cache, por ejemplo, la compilación de docker --no-cache.

Consulte la guía de mejores prácticas de Dockerfile para obtener más información.

El caché para las instrucciones `RUN` puede ser invalidado por las instrucciones `ADD` . Vea a continuación para más detalles.

Instrucción ONBUILD

```
ONBUILD [INSTRUCTION]
```

La instrucción `ONBUILD` agrega a la imagen una instrucción de activación que se ejecutará más adelante, cuando la imagen se use como base para otra construcción. El disparador se ejecutará en el contexto de la compilación descendente, como si se hubiera insertado inmediatamente después de la instrucción `FROM` en el Dockerfile descendente.

Cualquier instrucción de construcción puede ser registrada como un disparador.

Esto es útil si está compilando una imagen que se usará como base para compilar otras imágenes, por ejemplo, un entorno de compilación de aplicaciones o un demonio que se puede personalizar con la configuración específica del usuario.

Por ejemplo, si su imagen es un generador de aplicaciones Python reutilizable, requerirá que se

agregue el código fuente de la aplicación en un directorio en particular, y podría requerir que se llame un script de compilación después de eso. No puede simplemente llamar a `ADD` y `RUN` ahora, porque todavía no tiene acceso al código fuente de la aplicación, y será diferente para cada compilación de la aplicación. Simplemente puede proporcionar a los desarrolladores de aplicaciones un Dockerfile para copiar y pegar en su aplicación, pero eso es ineficiente, propenso a errores y difícil de actualizar porque se mezcla con el código específico de la aplicación.

La solución es usar `ONBUILD` para registrar instrucciones avanzadas para ejecutar más tarde, durante la siguiente etapa de compilación.

Así es como funciona:

Cuando encuentra una instrucción `ONBUILD` , el constructor agrega un disparador a los metadatos de la imagen que se está construyendo. La instrucción no afecta de otra manera la construcción actual.

Al final de la compilación, se almacena una lista de todos los desencadenantes en el manifiesto de la imagen, bajo la clave OnBuild. Se pueden inspeccionar con el comando de `docker inspect` la `docker inspect` . Más tarde, la imagen se puede usar como base para una nueva construcción, utilizando la instrucción `FROM` . Como parte del procesamiento de la instrucción `FROM` , el constructor descendente busca los activadores `ONBUILD` y los ejecuta en el mismo orden en que se registraron. Si alguno de los disparadores falla, la instrucción `FROM` se cancela, lo que a su vez hace que la compilación falle. Si todos los activadores tienen éxito, la instrucción `FROM` completa y la compilación continúa como siempre.

Los disparadores se eliminan de la imagen final después de ejecutarse. En otras palabras, no son heredados por construcciones de "nietos".

Por ejemplo, podría agregar algo como esto:

```
[...]
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
[...]
```

Advertencia: no se permite el encadenamiento de instrucciones `ONBUILD` utilizando `ONBUILD ONBUILD` .

Advertencia: la instrucción `ONBUILD` no puede activar las instrucciones `FROM` o `MAINTAINER` .

Instrucción de STOPSIGNAL

```
STOPSIGNAL signal
```

La instrucción `STOPSIGNAL` establece la señal de llamada del sistema que se enviará al contenedor para salir. Esta señal puede ser un número sin signo válido que coincida con una posición en la tabla syscall del kernel, por ejemplo 9, o un nombre de señal en el formato SIGNAME, por ejemplo SIGKILL.

Instrucción de SALUD

La instrucción `HEALTHCHECK` tiene dos formas:

```
HEALTHCHECK [OPTIONS] CMD command (check container health by running a command inside the
container)
HEALTHCHECK NONE (disable any healthcheck inherited from the base image)
```

La instrucción `HEALTHCHECK` le dice a Docker cómo probar un contenedor para verificar que aún funciona. Esto puede detectar casos como un servidor web que está atascado en un bucle infinito y no puede manejar nuevas conexiones, incluso aunque el proceso del servidor todavía se esté ejecutando.

Cuando un contenedor tiene un chequeo de salud especificado, tiene un estado de salud además de su estado normal. Este estado se está iniciando inicialmente. Cada vez que pasa un chequeo de salud, se vuelve saludable (sea cual sea el estado en el que se encontraba anteriormente). Después de un cierto número de fallas consecutivas, se vuelve insalubre.

Las opciones que pueden aparecer antes de `CMD` son:

```
--interval=DURATION (default: 30s)
--timeout=DURATION (default: 30s)
--retries=N (default: 3)
```

La comprobación de estado se ejecutará primero en intervalos de segundos después de que se inicie el contenedor, y luego nuevamente en intervalos de segundos después de que se complete cada verificación anterior.

Si una sola ejecución de la verificación lleva más tiempo que el tiempo de espera, se considera que la verificación ha fallado.

Es necesario volver a intentar las fallas consecutivas de la comprobación de estado para que el contenedor se considere insalubre.

Solo puede haber una instrucción `HEALTHCHECK` en un `Dockerfile` . Si enumera más de uno, solo tendrá efecto el último `HEALTHCHECK` .

El comando después de la palabra clave `CMD` puede ser un comando de shell (por ejemplo, `HEALTHCHECK CMD /bin/check-running`) o una matriz exec (como con otros comandos de Dockerfile; consulte, por ejemplo, `ENTRYPOINT` para obtener más información).

El estado de salida del comando indica el estado de salud del contenedor. Los valores posibles son:

- `0: success` - el envase está sano y listo para usar
- `1: unhealthy` - el contenedor no funciona correctamente
- `2: starting` : el contenedor aún no está listo para su uso, pero funciona correctamente

Si la sonda devuelve 2 ("inicio") cuando el contenedor ya se ha movido fuera del estado de

"inicio", entonces se trata como "no saludable" en su lugar.

Por ejemplo, para comprobar cada cinco minutos aproximadamente, un servidor web puede servir la página principal del sitio en tres segundos:

```
HEALTHCHECK --interval=5m --timeout=3s \
  CMD curl -f http://localhost/ || exit 1
```

Para ayudar a depurar las sondas que fallan, cualquier texto de salida (codificado en UTF-8) que el comando escribe en stdout o stderr se almacenará en el estado de salud y se puede consultar con `docker inspect` . Dicha salida debe mantenerse corta (solo se almacenan actualmente los primeros 4096 bytes).

Cuando cambia el estado de `health_status` de un contenedor, se genera un evento `health_status` con el nuevo estado.

La característica `HEALTHCHECK` se agregó en Docker 1.12.

Instrucción SHELL

```
SHELL ["executable", "parameters"]
```

La instrucción `SHELL` permite anular el shell predeterminado utilizado para la forma de shell de los comandos. El shell predeterminado en Linux es `["/bin/sh", "-c"]` , y en Windows es `["cmd", "/S", "/C"]` . La instrucción `SHELL` debe estar escrita en forma JSON en un Dockerfile.

La instrucción `SHELL` es particularmente útil en Windows donde hay dos shells nativos muy diferentes y de uso común: cmd y powershell, así como shells alternativos disponibles, incluyendo sh.

La instrucción `SHELL` puede aparecer varias veces. Cada instrucción `SHELL` anula todas las instrucciones anteriores de `SHELL` y afecta a todas las instrucciones posteriores. Por ejemplo:

```
FROM windowsservercore

# Executed as cmd /S /C echo default
RUN echo default

# Executed as cmd /S /C powershell -command Write-Host default
RUN powershell -command Write-Host default

# Executed as powershell -command Write-Host hello
SHELL ["powershell", "-command"]
RUN Write-Host hello

# Executed as cmd /S /C echo hello
SHELL ["cmd", "/S"", "/C"]
RUN echo hello
```

Las siguientes instrucciones pueden verse afectadas por la instrucción `SHELL` cuando su forma de shell se usa en un Dockerfile: `RUN` , `CMD` y `ENTRYPOINT` .

El siguiente ejemplo es un patrón común que se encuentra en Windows que puede optimizarse utilizando la instrucción `SHELL` :

```
...
RUN powershell -command Execute-MyCmdlet -param1 "c:\foo.txt"
...
```

El comando invocado por la ventana acoplable será:

```
cmd /S /C powershell -command Execute-MyCmdlet -param1 "c:\foo.txt"
```

Esto es ineficiente por dos razones. Primero, se invoca un procesador de comando cmd.exe innecesario (también conocido como shell). En segundo lugar, cada instrucción `RUN` en el formulario de shell requiere un comando de PowerShell adicional que prefija el comando.

Para hacer esto más eficiente, uno de los dos mecanismos puede ser empleado. Una es usar la forma JSON del comando `RUN` tal como:

```
...
RUN ["powershell", "-command", "Execute-MyCmdlet", "-param1 \"c:\\foo.txt\""]
...
```

Si bien el formulario JSON es inequívoco y no utiliza el cmd.exe innecesario, requiere más verbosidad a través de la doble cita y el escape. El mecanismo alternativo es usar la instrucción `SHELL` y el formulario de shell, lo que crea una sintaxis más natural para los usuarios de Windows, especialmente cuando se combina con la directiva del analizador de escape:

```
# escape=`

FROM windowsservercore
SHELL ["powershell","-command"]
RUN New-Item -ItemType Directory C:\Example
ADD Execute-MyCmdlet.ps1 c:\example\
RUN c:\example\Execute-MyCmdlet -sample 'hello world'
```

Resultando en:

```
PS E:\docker\build\shell> docker build -t shell .
Sending build context to Docker daemon 3.584 kB
Step 1 : FROM windowsservercore
 ---> 5bc36a335344
Step 2 : SHELL powershell -command
 ---> Running in 87d7a64c9751
 ---> 4327358436c1
Removing intermediate container 87d7a64c9751
Step 3 : RUN New-Item -ItemType Directory C:\Example
 ---> Running in 3e6ba16b8df9


Directory: C:\


Mode                LastWriteTime         Length Name
```

```
----                 -------------         ------ ----
d-----         6/2/2016   2:59 PM                Example


 ---> 1f1dfdcec085
Removing intermediate container 3e6ba16b8df9
Step 4 : ADD Execute-MyCmdlet.ps1 c:\example\
 ---> 6770b4c17f29
Removing intermediate container b139e34291dc
Step 5 : RUN c:\example\Execute-MyCmdlet -sample 'hello world'
 ---> Running in abdcf50dfd1f
Hello from Execute-MyCmdlet.ps1 - passed hello world
 ---> ba0e25255fda
Removing intermediate container abdcf50dfd1f
Successfully built ba0e25255fda
PS E:\docker\build\shell>
```

La instrucción `SHELL` también podría usarse para modificar la forma en que funciona un shell. Por ejemplo, al usar `SHELL cmd /S /C /V:ON|OFF` en Windows, la semántica de expansión de la variable de entorno diferida podría modificarse.

La instrucción `SHELL` también se puede usar en Linux en caso de que se requiera un shell alternativo como zsh, csh, tcsh y otros.

La característica `SHELL` se agregó en Docker 1.12.

Instalación de paquetes Debian / Ubuntu

Ejecute la instalación en un solo comando de ejecución para fusionar la actualización e instalar. Si agrega más paquetes más tarde, esto ejecutará la actualización nuevamente e instalará todos los paquetes necesarios. Si la actualización se ejecuta por separado, se almacenará en caché y las instalaciones de paquetes pueden fallar. Configurar la interfaz para que no sea interactivo y pasar la opción -y para instalar es necesario para las instalaciones con script. La limpieza y purga al final de la instalación minimiza el tamaño de la capa.

```
FROM debian

RUN apt-get update \
 && DEBIAN_FRONTEND=noninteractive apt-get install -y \
    git \
    openssh-client \
    sudo \
    vim \
    wget \
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*
```

Lea Dockerfiles en línea: https://riptutorial.com/es/docker/topic/3161/dockerfiles

Capítulo 18: Ejecutando la aplicación Node.js simple

Examples

Ejecutar una aplicación Node.js básica dentro de un contenedor

El ejemplo que voy a discutir asume que tienes una instalación de Docker que funciona en tu sistema y una comprensión básica de cómo trabajar con Node.js. Si sabe cómo debe trabajar con Docker, debería ser evidente que el marco Node.js no necesita estar instalado en su sistema, sino que estaríamos usando la `latest` versión de la imagen de `node` disponible en Docker. Por lo tanto, si es necesario, puede descargar la imagen de antemano con el comando `docker pull node` . (El comando `pulls` automáticamente la última versión de la imagen del `node` desde la ventana acoplable).

1. Proceda a hacer un directorio donde residan todos sus archivos de aplicación de trabajo. Cree un archivo `package.json` en este directorio que describa su aplicación, así como las dependencias. El archivo `package.json` debería tener este aspecto:

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.13.3"
  }
}
```

2. Si necesitamos trabajar con Node.js, generalmente creamos un archivo de `server` que define una aplicación web. En este caso, utilizamos el framework `Express.js` (versión `4.13.3` adelante). Un archivo `server.js` básico se vería así:

```
var express = require('express');
var PORT = 8080;
var app = express();
app.get('/', function (req, res) {
  res.send('Hello world\n');
});

app.listen(PORT);
console.log('Running on http://localhost:' + PORT);
```

3. Para aquellos familiarizados con Docker, `Dockerfile` encontrado un `Dockerfile` . Un `Dockerfile` es un archivo de texto que contiene todos los comandos necesarios para crear

una imagen personalizada que se adapte a su aplicación.

Cree un archivo de texto vacío llamado `Dockerfile` en el directorio actual. El método para crear uno es sencillo en Windows. En Linux, es posible que desee ejecutar `touch Dockerfile` en el directorio que contiene todos los archivos necesarios para su aplicación. Abra el Dockerfile con cualquier editor de texto y agregue las siguientes líneas:

```
FROM node:latest
RUN mkdir -p /usr/src/my_first_app
WORKDIR /usr/src/my_first_app
COPY package.json /usr/src/my_first_app/
RUN npm install
COPY . /usr/src/my_first_app
EXPOSE 8080
```

- `FROM node:latest` instruye al demonio Docker desde qué imagen queremos construir. En este caso, utilizamos la `latest` versión del `node` imagen oficial de Docker disponible en Docker Hub .
- Dentro de esta imagen, procedemos a crear un directorio de trabajo que contiene todos los archivos necesarios y le indicamos al demonio que establezca este directorio como el directorio de trabajo deseado para nuestra aplicación. Para esto sumamos

  ```
  RUN mkdir -p /usr/src/my_first_app
  WORKDIR /usr/src/my_first_app
  ```

- Luego procedemos a instalar las dependencias de la aplicación moviendo primero el archivo `package.json` (que especifica la información de la aplicación, incluidas las dependencias) al directorio de trabajo `/usr/src/my_first_app` en la imagen. Hacemos esto por

  ```
  COPY package.json /usr/src/my_first_app/
  RUN npm install
  ```

- Luego `COPY . /usr/src/my_first_app` para agregar todos los archivos de la aplicación y el código fuente al directorio de trabajo en la imagen.
- Luego, usamos la directiva `EXPOSE` para indicar al demonio que haga visible el puerto `8080` del contenedor resultante (a través de una asignación de contenedor a host), ya que la aplicación se enlaza con el puerto `8080` .
- En el último paso, le indicamos al demonio que ejecute el comando `node server.js` dentro de la imagen ejecutando el comando de `npm start` básico. Usamos la directiva `CMD` para esto, que toma los comandos como argumentos.

  ```
  CMD [ "npm", "start" ]
  ```

4. Luego, creamos un archivo `.dockerignore` en el mismo directorio que `Dockerfile` para evitar que nuestra copia de `node_modules` y registros utilizados por nuestra instalación del sistema Node.js se copie en la imagen de Docker. El archivo `.dockerignore` debe tener el siguiente

contenido:

```
node_modules
npm-debug.log
```

5. # Construye tu imagen

Navegue hasta el directorio que contiene el `Dockerfile` y ejecute el siguiente comando para crear la imagen de Docker. La marca `-t` te permite etiquetar tu imagen para que sea más fácil encontrarla más tarde con el comando de imágenes acoplables:

```
    $ docker build -t <your username>/node-web-app .
```

Su imagen ahora será listada por Docker. Ver imágenes usando el siguiente comando:

```
$ docker images

REPOSITORY                        TAG         ID              CREATED
node                              latest      539c0211cd76    10 minutes ago
<your username>/node-web-app      latest      d64d3505b0d2    1 minute ago
```

6. # Corriendo la imagen

Ahora podemos ejecutar la imagen que acabamos de crear utilizando el contenido de la aplicación, la imagen base del `node` y el `Dockerfile` . Ahora procedemos a ejecutar nuestra imagen recién creada `<your username>/node-web-app` . Al proporcionar el interruptor `-d` al comando de `docker run` la `docker run` ejecuta el contenedor en modo separado, de modo que el contenedor se ejecuta en segundo plano. La bandera `-p` redirige un puerto público a un puerto privado dentro del contenedor. Ejecuta la imagen que creaste previamente usando este comando:

```
$ docker run -p 49160:8080 -d <your username>/node-web-app
```

7. Imprima la salida de su aplicación ejecutando `docker ps` en su terminal. La salida debe verse algo como esto.

```
 CONTAINER ID        IMAGE                          COMMAND             CREATED
STATUS              PORTS                      NAMES
 7b701693b294      <your username>/node-web-app   "npm start"         20 minutes ago
Up 48 seconds       0.0.0.0:49160->8080/tcp    loving_goldstine
```

Obtenga la salida de la aplicación ingresando los `docker logs <CONTAINER ID>` En este caso se trata de `docker logs 7b701693b294` .

Salida: `Running on http://localhost:8080`

8. Desde la salida de la `0.0.0.0:49160->8080/tcp` `docker ps` , la asignación de puertos obtenida

es `0.0.0.0:49160->8080/tcp` . Por lo tanto, Docker asignó el puerto `8080` dentro del contenedor al puerto 49160 en la máquina host. En el navegador ahora podemos ingresar `localhost:49160` .

También podemos llamar a nuestra aplicación usando `curl` :

```
$ curl -i localhost:49160

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
Date: Sun, 08 Jan 2017 14:00:12 GMT
Connection: keep-alive

Hello world
```

Lea Ejecutando la aplicación Node.js simple en línea:
https://riptutorial.com/es/docker/topic/8754/ejecutando-la-aplicacion-node-js-simple

Capítulo 19: ejecutar cónsul en docker 1.12 swarm

Examples

Ejecutar cónsul en un enjambre 1.12 enjambre

Esto se basa en la imagen oficial de la ventana acoplable del cónsul para ejecutar al cónsul en modo agrupado en un enjambre acoplable con el nuevo modo de enjambre en Docker 1.12. Este ejemplo se basa en http://qnib.org/2016/08/11/consul-service/ . Brevemente, la idea es utilizar dos servicios de enjambre de ventana acoplable que se comuniquen entre sí. Esto resuelve el problema de que no puede conocer los ips de los contenedores de los cónsules individuales por adelantado y le permite confiar en los dns de enjambres de docker.

Esto supone que ya tiene un clúster de enjambres de docking 1.12 en ejecución con al menos tres nodos.

Es posible que desee configurar un controlador de registro en sus demonios docker para que pueda ver lo que está sucediendo. `--log-driver=syslog` el controlador de syslog para esto: configuré la `--log-driver=syslog` en dockerd.

Primero crea una red superpuesta para cónsul:

```
docker network create consul-net -d overlay
```

Ahora reinicie el cluster con solo 1 nodo (el valor predeterminado --replicas es 1):

```
docker service create --name consul-seed \
  -p 8301:8300 \
  --network consul-net \
  -e 'CONSUL_BIND_INTERFACE=eth0' \
  consul agent -server -bootstrap-expect=3  -retry-join=consul-seed:8301 -retry-join=consul-
cluster:8300
```

Ahora debería tener un clúster de 1 nodo. Ahora trae el segundo servicio:

```
docker service create --name consul-cluster \
  -p 8300:8300 \
  --network consul-net \
  --replicas 3 \
  -e 'CONSUL_BIND_INTERFACE=eth0' \
  consul agent -server -retry-join=consul-seed:8301 -retry-join=consul-cluster:8300
```

Ahora debería tener un clúster de cuatro nodos. Puede verificar esto ejecutando en cualquiera de los contenedores de la ventana acoplable:

```
docker exec <containerid> consul members
```

Lea ejecutar cónsul en docker 1.12 swarm en línea:
https://riptutorial.com/es/docker/topic/6437/ejecutar-consul-en-docker-1-12-swarm

Capítulo 20: Eventos docker

Examples

Lanzar un contenedor y ser notificado de eventos relacionados.

La documentación para los `docker events` proporciona detalles, pero cuando la depuración puede ser útil para iniciar un contenedor y recibir una notificación inmediata de cualquier evento relacionado:

```
docker run... & docker events --filter 'container=$(docker ps -lq)'
```

En `docker ps -lq` , la `l` representa el `last` y la `q` para el `quiet` . Esto elimina la `id` del último contenedor lanzado y crea una notificación inmediatamente si el contenedor muere o si ocurre otro evento.

Lea Eventos docker en línea: https://riptutorial.com/es/docker/topic/6200/eventos-docker

Capítulo 21: Explotación florestal

Examples

Configurando un controlador de registro en el servicio systemd

```
[Service]

# empty exec prevents error "docker.service has more than one ExecStart= setting, which is
only allowed for Type=oneshot services. Refusing."
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// --log-driver=syslog
```

Esto habilita el registro de syslog para el demonio docker. El archivo debe crearse en el directorio apropiado con la raíz del propietario, que normalmente sería `/etc/systemd/system/docker.service.d` en, por ejemplo, Ubuntu 16.04.

Visión general

El enfoque de Docker para el registro es que usted construya sus contenedores de tal manera, de modo que los registros se escriban en la salida estándar (consola / terminal).

Si ya tiene un contenedor que escribe registros en un archivo, puede redirigirlo creando un enlace simbólico:

```
ln -sf /dev/stdout /var/log/nginx/access.log
ln -sf /dev/stderr /var/log/nginx/error.log
```

Una vez hecho esto, puede usar varios controladores de registro para colocar sus registros donde los necesite.

Lea Explotación florestal en línea: https://riptutorial.com/es/docker/topic/7378/explotacion-florestal

Capítulo 22: Gestion de contenedores

Sintaxis

- docker rm [OPCIONES] CONTENEDOR [CONTENEDOR ...]
- acoplar acoplador [OPCIONES] CONTENEDOR
- docker exec [OPCIONES] COMANDO DE CONTENEDORES [ARG ...]
- docker ps [OPCIONES]
- troncos acoplables [OPCIONES] CONTENEDOR
- Docker inspeccionar [OPCIONES] CONTENEDOR | IMAGEN [CONTENEDOR | IMAGEN ...]

Observaciones

- En los ejemplos anteriores, siempre que el contenedor es un parámetro del comando de ventana acoplable, se menciona como `<container>` o el `container id` o `<CONTAINER_NAME>` . En todos estos lugares puede pasar un nombre de contenedor o un ID de contenedor para especificar un contenedor.

Examples

Listado de contenedores

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                     NAMES
2bc9b1988080        redis               "docker-entrypoint.sh"   2 weeks ago         Up 2
hours          0.0.0.0:6379->6379/tcp    elephant-redis
817879be2230        postgres            "/docker-entrypoint.s"   2 weeks ago         Up 2
hours          0.0.0.0:65432->5432/tcp   pt-postgres
```

`docker ps` solo imprime contenedores actualmente en ejecución. Para ver todos los contenedores (incluidos los detenidos), use la bandera `-a` :

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                     NAMES
9cc69f11a0f7        docker/whalesay     "ls /"                   26 hours ago        Exited
(0) 26 hours ago                            berserk_wozniak
2bc9b1988080        redis               "docker-entrypoint.sh"   2 weeks ago         Up 2
hours                0.0.0.0:6379->6379/tcp    elephant-redis
817879be2230        postgres            "/docker-entrypoint.s"   2 weeks ago         Up 2
hours                0.0.0.0:65432->5432/tcp   pt-postgres
```

Para listar contenedores con un estado específico, use la opción de línea de comando `-f` para filtrar los resultados. Aquí hay un ejemplo de listado de todos los contenedores que han salido:

```
$ docker ps -a -f status=exited
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                     NAMES
9cc69f11a0f7        docker/whalesay     "ls /"                   26 hours ago        Exited
(0) 26 hours ago
```

También es posible enumerar solo las ID de contenedor con el conmutador `-q` . Esto hace que sea muy fácil operar el resultado con otras utilidades Unix (como `grep` y `awk`):

```
$ docker ps -aq
9cc69f11a0f7
2bc9b1988080
817879be2230
```

Al iniciar un contenedor con la aplicación `docker run --name mycontainer1` le da un nombre específico y no un nombre aleatorio (en la forma mood_famous, como nostalgic_stallman), y puede ser fácil encontrarlos con un comando de este tipo

```
docker ps -f name=mycontainer1
```

Contenedores de referencia

Los comandos de Docker que toman el nombre de un contenedor aceptan tres formas diferentes:

Tipo	Ejemplo
UUID completo	`9cc69f11a0f76073e87f25cb6eaf0e079fbfbd1bc47c063bcd25ed3722a8cc4a`
UUID corto	`9cc69f11a0f7`
Nombre	`berserk_wozniak`

Use `docker ps` para ver estos valores para los contenedores en su sistema.

El UUID es generado por Docker y no puede ser modificado. Puede proporcionar un nombre al contenedor cuando lo inicie `docker run --name <given name> <image>` . Docker generará un nombre aleatorio para el contenedor si no especifica uno al momento de iniciar el contenedor.

***NOTA** : El valor del UUID (o un UUID 'corto') puede ser de cualquier longitud siempre que el valor dado sea único para un contenedor*

Arranque y parada de contenedores.

Para detener un contenedor en ejecución:

```
docker stop <container> [<container>...]
```

Esto enviará al proceso principal en el contenedor un SIGTERM, seguido de un SIGKILL si no se detiene dentro del período de gracia. El nombre de cada contenedor se imprime cuando se

detiene.

Para iniciar un contenedor que está detenido:

```
docker start <container> [<container>...]
```

Esto iniciará cada contenedor pasado en el fondo; El nombre de cada contenedor se imprime a medida que comienza. Para iniciar el contenedor en primer plano, pase la `--attach -a` (`--attach`).

Listar contenedores con formato personalizado.

```
docker ps --format 'table {{.ID}}\t{{.Names}}\t{{.Status}}'
```

Encontrar un contenedor específico

```
docker ps --filter name=myapp_1
```

Encontrar contenedor IP

Para averiguar la dirección IP de su contenedor, use:

```
docker inspect <container id> | grep IPAddress
```

o usar el docker inspeccionar

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' ${CID}
```

Reiniciando contenedor contenedor

```
docker restart <container> [<container>...]
```

Opción - **tiempo** : segundos para esperar antes de detener el contenedor (valor predeterminado 10)

```
docker restart <container> --time 10
```

Eliminar, eliminar y limpiar contenedores.

`docker rm` se puede utilizar para eliminar contenedores específicos como este:

```
docker rm <container name or id>
```

Para eliminar todos los contenedores puedes usar esta expresión:

```
docker rm $(docker ps -qa)
```

Por defecto, la ventana acoplable no eliminará un contenedor que se esté ejecutando. Cualquier contenedor que se esté ejecutando producirá un mensaje de advertencia y no se eliminará. Todos los demás contenedores serán eliminados.

Alternativamente puedes usar `xargs` :

```
docker ps -aq -f status=exited | xargs -r docker rm
```

Donde el `docker ps -aq -f status=exited` devolverá una lista de ID de contenedor de contenedores que tienen el estado "Exited".

> Advertencia: Todos los ejemplos anteriores solo eliminarán los contenedores 'detenidos'.

Para eliminar un contenedor, independientemente de si está detenido o no, puede utilizar el indicador de fuerza `-f` :

```
docker rm -f <container name or id>
```

Para eliminar todos los contenedores, independientemente del estado:

```
docker rm -f $(docker ps -qa)
```

Si desea eliminar solo los contenedores con un estado `dead` :

```
docker rm $(docker ps --all -q -f status=dead)
```

Si desea eliminar solo los contenedores con un estado de `exited` :

```
docker rm $(docker ps --all -q -f status=exited)
```

Estas son todas las permutaciones de los filtros utilizados al enumerar contenedores .

Para eliminar tanto los contenedores no deseados como las imágenes colgantes que usan espacio después de la versión 1.3 , use lo siguiente (similar a la herramienta `df` Unix):

```
$ docker system df
```

Para eliminar todos los datos no utilizados:

```
$ docker system prune
```

Ejecutar comando en un contenedor de ventana acoplable ya existente

```
docker exec -it <container id> /bin/bash
```

Es común iniciar sesión en un contenedor que ya se está ejecutando para realizar algunas pruebas rápidas o ver qué está haciendo la aplicación. A menudo denota malas prácticas de uso

de contenedores debido a los registros y los archivos modificados deben colocarse en volúmenes. Este ejemplo nos permite iniciar sesión en el contenedor. Esto supone que / bin / bash está disponible en el contenedor, puede ser / bin / sh o cualquier otra cosa.

```
docker exec <container id> tar -czvf /tmp/backup.tgz /data
docker cp <container id>:/tmp/backup.tgz .
```

Este ejemplo archiva el contenido del directorio de datos en un tar. Luego con el `docker cp` puedes recuperarlo.

Registros de contenedores

```
Usage:  docker logs [OPTIONS] CONTAINER

Fetch the logs of a container

  -f, --follow=false        Follow log output
  --help=false              Print usage
  --since=                  Show logs since timestamp
  -t, --timestamps=false    Show timestamps
  --tail=all                Number of lines to show from the end of the logs
```

Por ejemplo:

```
$ docker ps
CONTAINER ID    IMAGE    COMMAND                  CREATED     STATUS       PORTS
ff9716dda6cb    nginx    "nginx -g 'daemon off"  8 days ago  Up 22 hours  443/tcp,
0.0.0.0:8080->80/tcp

$ docker logs ff9716dda6cb
xx.xx.xx.xx - - [15/Jul/2016:14:03:44 +0000] "GET /index.html HTTP/1.1" 200 511
"https://google.com" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/50.0.2661.75 Safari/537.36"
xx.xx.xx.xx - - [15/Jul/2016:14:03:44 +0000] "GET /index.html HTTP/1.1" 200 511
"https://google.com" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/50.0.2661.75 Safari/537.36"
```

Conectar a una instancia que se ejecuta como demonio

Hay dos formas de lograrlo, la primera y la más conocida es la siguiente:

```
docker attach --sig-proxy=false <container>
```

Este, literalmente, adjunta su bash al contenedor bash, lo que significa que si tiene un script en ejecución, verá el resultado.

Para separar, simplemente escriba: `Ctl-P Ctl-Q`

Pero si necesita una forma más amigable y poder crear nuevas instancias de bash, simplemente ejecute el siguiente comando:

```
docker exec -it <container> bash
```

Copiando archivo desde / a contenedores

del contenedor al host

```
docker cp CONTAINER_NAME:PATH_IN_CONTAINER PATH_IN_HOST
```

de host a contenedor

```
docker cp PATH_IN_HOST CONTAINER_NAME:PATH_IN_CONTAINER
```

Si utilizo jess / transmisión desde

https://hub.docker.com/r/jess/transmission/builds/bsn7eqxrkzrhxazcuytbmzp/

, los archivos en el contenedor están en / transmisión / descarga

y mi directorio actual en el host es / home / $ USER / abc, después de

`docker cp transmission_id_or_name:/transmission/download .`

Tendré los archivos copiados a

`/home/$USER/abc/transmission/download`

no puede, utilizando `docker cp` copiar solo un archivo, copie el árbol de directorios y los archivos

Eliminar, eliminar y limpiar los volúmenes de la ventana acoplable.

Los volúmenes de Docker no se eliminan automáticamente cuando se detiene un contenedor. Para eliminar volúmenes asociados al detener un contenedor:

```
docker rm -v <container id or name>
```

Si no se especifica el indicador `-v` , el volumen permanece en el disco como un 'volumen colgante'. Para borrar todos los volúmenes colgantes:

```
docker volume rm $(docker volume ls -qf dangling=true)
```

El `docker volume ls -qf dangling=true` filter devolverá una lista de nombres de volúmenes de la `docker volume ls -qf dangling=true` acoplable, incluidos los no etiquetados, que no están adjuntos a un contenedor.

Alternativamente, puedes usar `xargs` :

```
docker volume ls -f dangling=true -q | xargs --no-run-if-empty docker volume rm
```

Exportar e importar sistemas de archivos contenedor Docker

Es posible guardar el contenido del sistema de archivos de un contenedor Docker en un archivo de archivo tarball. Esto es útil en una pizca para mover sistemas de archivos de contenedores a diferentes hosts, por ejemplo, si un contenedor de base de datos tiene cambios importantes y de lo contrario no es posible replicar esos cambios en otros lugares. **Tenga en cuenta** que es preferible crear un contenedor completamente nuevo a partir de una imagen actualizada utilizando un comando de `docker run docker-compose.yml` o `docker-compose.yml` archivo `docker-compose.yml` , en lugar de exportar y mover el sistema de archivos de un contenedor. Parte del poder de Docker es la capacidad de auditoría y la responsabilidad de su estilo declarativo de creación de imágenes y contenedores. Al utilizar la `docker export` y la `docker import` , esta potencia se ve reducida debido a la ofuscación de los cambios realizados dentro del sistema de archivos de un contenedor desde su estado original.

```
docker export -o redis.tar redis
```

El comando anterior creará una imagen vacía y luego exportará el sistema de archivos del contenedor `redis` a esta imagen vacía. Para importar desde un archivo tarball, use:

```
docker import ./redis.tar redis-imported:3.0.7
```

Este comando creará la imagen `redis-imported:3.0.7` , a partir de la cual se pueden crear los contenedores. También es posible crear cambios en la importación, así como establecer un mensaje de confirmación:

```
docker import -c="ENV DEBUG true" -m="enable debug mode" ./redis.tar redis-changed
```

Las directivas Dockerfile disponibles para uso con la opción de línea de comando `-c` son `CMD` , `ENTRYPOINT` , `ENV` , `EXPOSE` , `ONBUILD` , `USER` , `VOLUME` , `WORKDIR` .

Lea Gestion de contenedores en línea: https://riptutorial.com/es/docker/topic/689/gestion-de-contenedores

Capítulo 23: Gestionando imagenes

Sintaxis

- Imágenes de la ventana acoplable [OPCIONES] [REPOSITORIO [: TAG]]
- Docker inspeccionar [OPCIONES] CONTENEDOR | IMAGEN [CONTENEDOR | IMAGEN ...]
- docker pull [OPCIONES] NOMBRE [: TAG | @DIGEST]
- docker rmi [OPCIONES] IMAGEN [IMAGEN ...]
- etiqueta acoplable [OPCIONES] IMAGEN [: ETIQUETA] [REGISTRYHOST /] [NOMBRE DE USUARIO /] NOMBRE [: ETIQUETA]

Examples

Recuperando una imagen de Docker Hub

Normalmente, las imágenes se extraen automáticamente de Docker Hub . Docker intentará extraer cualquier imagen de Docker Hub que aún no exista en el host de Docker. Por ejemplo, el uso de `docker run ubuntu` cuando la imagen de `ubuntu` no está en el host de Docker hará que Docker inicie una extracción de la última imagen de `ubuntu` . Es posible extraer una imagen por separado utilizando la función `docker pull` para recuperar o actualizar manualmente una imagen desde Docker Hub.

```
docker pull ubuntu
docker pull ubuntu:14.04
```

Existen opciones adicionales para extraer de un registro de imagen diferente o extraer una versión específica de una imagen. La indicación de un registro alternativo se realiza utilizando el nombre completo de la imagen y la versión opcional. Por ejemplo, el siguiente comando intentará extraer la imagen de `ubuntu:14.04` del `registry.example.com` registry.example.com:

```
docker pull registry.example.com/username/ubuntu:14.04
```

Listado de imágenes descargadas localmente

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              693bce725149        6 days ago          967 B
postgres            9.5                 0f3af79d8673        10 weeks ago        265.7 MB
postgres            latest              0f3af79d8673        10 weeks ago        265.7 MB
```

Imágenes de referencia

Los comandos de Docker que toman el nombre de una imagen aceptan cuatro formas diferentes:

Tipo	Ejemplo
ID corta	`693bce725149`
Nombre	`hello-world` *(predeterminado `:latest` etiqueta)*
Nombre + etiqueta	`hello-world:latest`
Digerir	`hello-world@sha256:e52be8ffeeb1f374f440893189cd32f44cb166650e7ab185fa7735b7dc48d619`

Nota: Solo puede hacer referencia a una imagen por su resumen si la imagen se extrajo originalmente con ese resumen. Para ver el resumen de una imagen (si hay una disponible), ejecute las `docker images --digests` .

Eliminando imagenes

El comando `docker rmi` se utiliza para eliminar imágenes:

```
docker rmi <image name>
```

El nombre completo de la imagen debe usarse para eliminar una imagen. A menos que la imagen se haya etiquetado para eliminar el nombre de registro, debe especificarse. Por ejemplo:

```
docker rmi registry.example.com/username/myAppImage:1.3.5
```

También es posible eliminar imágenes por su ID en su lugar:

```
docker rmi 693bce725149
```

Como conveniencia, es posible eliminar imágenes por su ID de imagen especificando solo los primeros caracteres de la ID de imagen, siempre que la subcadena especificada sea inequívoca:

```
docker rmi 693
```

> **Nota: las** imágenes se pueden eliminar incluso si hay contenedores existentes que usan esa imagen; docker rmi simplemente "desata" la imagen.

Si ningún contenedor está utilizando una imagen, se recolecta en la basura. Si un contenedor usa una imagen, la imagen se recolectará en la basura una vez que se eliminen todos los contenedores que la usan. Por ejemplo:

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED                  STATUS
PORTS               NAMES
5483657ee07b        hello-world         "/hello"            Less than a second ago   Exited
(0) 2 seconds ago                       small_elion

$ docker rmi hello-world
```

```
Untagged: hello-world:latest

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED                   STATUS
PORTS               NAMES
5483657ee07b        693bce725149        "/hello"            Less than a second ago    Exited
(0) 12 seconds ago                       small_elion
```

Eliminar todas las imágenes sin contenedores iniciados

Para eliminar todas las imágenes locales que no tienen contenedores iniciados, puede proporcionar una lista de las imágenes como un parámetro:

```
docker rmi $(docker images -qa)
```

Eliminar todas las imágenes

Si desea eliminar imágenes, independientemente de si tienen o no un contenedor iniciado, use el indicador de fuerza (`-f`):

```
docker rmi -f $(docker images -qa)
```

Eliminar imágenes colgando

Si una imagen no está etiquetada y no está siendo utilizada por ningún contenedor, está "colgando" y puede eliminarse así:

```
docker images -q --no-trunc -f dangling=true | xargs -r docker rmi
```

Busca en el Docker Hub imágenes

Puede buscar imágenes en Docker Hub usando el comando de búsqueda :

```
docker search <term>
```

Por ejemplo:

```
$ docker search nginx
NAME                     DESCRIPTION                                     STARS     OFFICIAL
AUTOMATED
nginx                    Official build of Nginx.                        3565      [OK]
jwilder/nginx-proxy      Automated Nginx reverse proxy for docker c...   717
[OK]
richarvey/nginx-php-fpm  Container running Nginx + PHP-FPM capable ...   232
[OK]
...
```

Inspeccionando imagenes

```
docker inspect <image>
```

La salida está en formato JSON. Puede usar la utilidad de línea de comandos `jq` para analizar e imprimir solo las teclas deseadas.

```
docker inspect <image> | jq -r '.[0].Author'
```

El comando anterior mostrará el nombre del autor de las imágenes.

Etiquetando imagenes

Etiquetar una imagen es útil para realizar un seguimiento de diferentes versiones de imágenes:

```
docker tag ubuntu:latest registry.example.com/username/ubuntu:latest
```

Otro ejemplo de etiquetado:

```
docker tag myApp:1.4.2 myApp:latest
docker tag myApp:1.4.2 registry.example.com/company/myApp:1.4.2
```

Guardando y cargando imágenes de Docker

```
docker save -o ubuntu.latest.tar ubuntu:latest
```

Este comando guardará la imagen de `ubuntu:latest` como un archivo comprimido en el directorio actual con el nombre `ubuntu.latest.tar` . Este archivo tarball se puede mover a otro host, por ejemplo, mediante `rsync` , o archivado en el almacenamiento.

Una vez que se haya movido el tarball, el siguiente comando creará una imagen del archivo:

```
docker load -i /tmp/ubuntu.latest.tar
```

Ahora es posible crear contenedores desde `ubuntu:latest` imagen como de costumbre.

Lea Gestionando imagenes en línea: https://riptutorial.com/es/docker/topic/690/gestionando-imagenes

Capítulo 24: Inspeccionando un contenedor corriendo

Sintaxis

- Docker inspeccionar [OPCIONES] CONTENEDOR | IMAGEN [CONTENEDOR | IMAGEN ...]

Examples

Obtener información del contenedor

Para obtener toda la información de un contenedor puede ejecutar:

```
docker inspect <container>
```

Obtener información específica de un contenedor

Puede obtener una información específica de un contenedor ejecutando:

```
docker inspect -f '<format>' <container>
```

Por ejemplo, puede obtener la configuración de red ejecutando:

```
docker inspect -f '{{ .NetworkSettings }}' <container>
```

También puede obtener solo la dirección IP:

```
docker inspect -f '{{ .NetworkSettings.IPAddress }}' <container>
```

El parámetro -f significa formato y recibirá una plantilla Go como entrada para formatear lo que se espera, pero esto no traerá un hermoso retorno, así que intente:

```
docker inspect -f '{{ json .NetworkSettings }}' {{containerIdOrName}}
```

la palabra clave json traerá la devolución como JSON.

Para terminar, un pequeño consejo es usar python para formatear la salida JSON:

```
docker inspect -f '{{ json .NetworkSettings }}' <container> | python -mjson.tool
```

Y listo, puedes consultar cualquier cosa en la ventana acoplable inspeccionar y hacer que se vea bonita en tu terminal.

También es posible usar una utilidad llamada " jq " para ayudar a procesar la salida de comandos de la `docker inspect` .

```
docker inspect -f '{{ json .NetworkSettings }}' aa1 | jq [.Gateway]
```

El comando anterior devolverá la siguiente salida:

```
[
  "172.17.0.1"
]
```

Esta salida es en realidad una lista que contiene un elemento. A veces, la `docker inspect` muestra una lista de varios elementos y es posible que desee consultar un elemento específico. Por ejemplo, si `Config.Env` contiene varios elementos, puede referirse al primer elemento de esta lista usando el `index` :

```
docker inspect --format '{{ index (index .Config.Env) 0 }}' <container>
```

El primer elemento se indexa en cero, lo que significa que el segundo elemento de esta lista está en el índice `1` :

```
docker inspect --format '{{ index (index .Config.Env) 1 }}' <container>
```

Usando `len` es posible obtener el número de elementos de la lista:

```
docker inspect --format `{{ len .Config.Env }}` <container>
```

Y usando números negativos, es posible referirse al último elemento de la lista:

```
docker inspect -format "{{ index .Config.Cmd $[$(docker inspect -format `{{ len .Config.Cmd
}}` <container>)-1]}}" <container>
```

Algunos datos de `docker inspect` vienen como un diccionario de clave: valor, aquí hay un extracto de una `docker inspect` de un contenedor de jess / spotify en ejecución

```
"Config": { "Hostname": "8255f4804dde", "Domainname": "", "User": "spotify", "AttachStdin":
false, "AttachStdout": false, "AttachStderr": false, "Tty": false, "OpenStdin": false,
"StdinOnce": false, "Env": [ "DISPLAY=unix:0",
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", "HOME=/home/spotify" ],
"Cmd": [ "-stylesheet=/home/spotify/spotify-override.css" ], "Image": "jess/spotify", "Volumes":
null, "WorkingDir": "/home/spotify", "Entrypoint": [ "spotify" ], "OnBuild": null, "Labels": {}
},
```

Así que obtengo los valores de toda la sección Config.

```
docker inspect -f '{{.Config}}' 825

{8255f4804dde spotify false false false map[] false false false [DISPLAY=unix:0
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin HOME=/home/spotify] [-
stylesheet=/home/spotify/spotify-override.css] false jess/spotify map[] /home/spotify [spotify]
false [] map[] }
```

sino también un solo campo, como el valor de Config.Image

```
docker inspect -f '{{index (.Config) "Image" }}' 825

jess/spotify
```

o Config.Cmd

```
docker inspect -f '{{.Config.Cmd}}' 825

[-stylesheet=/home/spotify/spotify-override.css]
```

Inspeccionar una imagen

Para inspeccionar una imagen, puede usar la ID de la imagen o el nombre de la imagen, que consiste en el repositorio y la etiqueta. Digamos que tienes la imagen base de CentOS 6:

```
➜ ~ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              centos6             cf2c3ece5e41        2 weeks ago         194.6 MB
```

En este caso, puede ejecutar cualquiera de los siguientes:

- ➜ ~ docker inspect cf2c3ece5e41
- ➜ ~ docker inspect centos:centos6

Ambos comandos le darán toda la información disponible en una matriz JSON:

```
[
    {
        "Id": "sha256:cf2c3ece5e418fd063bfad5e7e8d083182195152f90aac3a5ca4dbfbf6a1fc2a",
        "RepoTags": [
            "centos:centos6"
        ],
        "RepoDigests": [],
        "Parent": "",
        "Comment": "",
        "Created": "2016-07-01T22:34:39.970264448Z",
        "Container": "b355fe9a01a8f95072e4406763138c5ad9ca0a50dbb0ce07387ba905817d6702",
        "ContainerConfig": {
            "Hostname": "68a1f3cfce80",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) CMD [\"/bin/bash\"]"
            ],
```

```
            "Image":
"sha256:cdbcc7980b002dc19b4d5b6ac450993c478927f673339b4e6893647fe2158fa7",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "build-date": "20160701",
                "license": "GPLv2",
                "name": "CentOS Base Image",
                "vendor": "CentOS"
            }
        },
        "DockerVersion": "1.10.3",
        "Author": "https://github.com/CentOS/sig-cloud-instance-images",
        "Config": {
            "Hostname": "68a1f3cfce80",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/bash"
            ],
            "Image":
"sha256:cdbcc7980b002dc19b4d5b6ac450993c478927f673339b4e6893647fe2158fa7",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "build-date": "20160701",
                "license": "GPLv2",
                "name": "CentOS Base Image",
                "vendor": "CentOS"
            }
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 194606575,
        "VirtualSize": 194606575,
        "GraphDriver": {
            "Name": "aufs",
            "Data": null
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:2714f4a6cdee9d4c987fef019608a4f61f1cda7ccf423aeb8d7d89f745c58b18"
            ]
        }
    }
]
```

Impresión de informaciones específicas.

docker `inspect` admite las plantillas Go a través de la opción `--format` . Esto permite una mejor integración en los scripts, sin tener que recurrir a las herramientas tradicionales pipe / sed / grep.

Imprima una IP interna del contenedor :

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' 7786807d8084
```

Esto es útil para el acceso directo a la red de la configuración automática de los equilibradores de carga.

Imprimir un contenedor *init* PID :

```
docker inspect --format '{{ .State.Pid }}' 7786807d8084
```

Esto es útil para una inspección más profunda a través de `/proc` o herramientas como `strace` .

Formateo avanzado :

```
docker inspect --format 'Container {{ .Name }} listens on {{ .NetworkSettings.IPAddress }}:{{
range $index, $elem := .Config.ExposedPorts }}{{ $index }}{{ end }}' 5765847de886 7786807d8084
```

Saldrá:

```
Container /redis listens on 172.17.0.3:6379/tcp
Container /api listens on 172.17.0.2:4000/tcp
```

Depuración de los registros de contenedores utilizando la ventana acoplable inspeccionar

`docker inspect` comando `docker inspect` se puede usar para depurar los registros del contenedor.

El stdout y el stderr del contenedor se pueden verificar para depurar el contenedor, cuya ubicación se puede obtener mediante la `docker inspect` .

Comando: `docker inspect <container-id> | grep Source`

Da la ubicación de los contenedores stdout y stderr.

Examinar stdout / stderr de un contenedor en ejecución

```
docker logs --follow <containerid>
```

Esto sigue la salida del contenedor en ejecución. Esto es útil si no configuró un controlador de registro en el daemon docker.

Lea Inspeccionando un contenedor corriendo en línea:

https://riptutorial.com/es/docker/topic/1336/inspeccionando-un-contenedor-corriendo

Capítulo 25: Iptables con Docker

Introducción

Este tema trata sobre cómo limitar el acceso a sus contenedores de la ventana acoplable desde el mundo exterior utilizando iptables.

Para las personas impacientes, puedes consultar los ejemplos. Para los demás, lea la sección de comentarios para comprender cómo crear nuevas reglas.

Sintaxis

- iptables -I DOCKER [REGLA ...] [ACCEPT | DROP] // Para agregar una regla a la parte superior de la tabla DOCKER
- iptables -D DOCKER [REGLA ...] [ACCEPT | DROP] // Para eliminar una regla de la tabla DOCKER
- ipset restore </etc/ipfriends.conf // Para reconfigurar los ipset *ipfriends*

Parámetros

Parámetros	Detalles
ext_if	Su interfaz externa en el host Docker.
XXX.XXX.XXX.XXX	Se debe dar una IP particular a la cual los contenedores de Docker acceden.
YYY.YYY.YYY.YYY	Se debe proporcionar otra IP a la que accedan los contenedores Docker.
ipfriends	El nombre del conjunto de ipsets que define las direcciones IP permitidas para acceder a sus contenedores de Docker.

Observaciones

El problema

Configurar las reglas de iptables para los contenedores Docker es un poco complicado. Al principio, usted pensaría que las reglas de firewall "clásicas" deberían hacer el truco.

Por ejemplo, supongamos que ha configurado un contenedor nginx-proxy + varios contenedores de servicio para exponer a través de HTTPS algunos servicios web personales. Entonces, una regla como esta debería dar acceso a sus servicios web solo para IP XXX.XXX.XXX.XXX.

```
$ iptables -A INPUT -i eth0 -p tcp -s XXX.XXX.XXX.XXX -j ACCEPT
$ iptables -P INPUT DROP
```

No funcionará, sus contenedores siguen siendo accesibles para todos.

De hecho, los contenedores Docker no son servicios de host. Se basan en una red virtual en su host, y el host actúa como una puerta de enlace para esta red. Y con respecto a las puertas de enlace, el tráfico enrutado no se maneja con la tabla INPUT, sino con la tabla FORWARD, que hace que la regla publicada antes sea inefectiva.

Pero no es todo. De hecho, el demonio Docker crea muchas reglas de iptables cuando comienza a hacer su magia con respecto a la conectividad de la red de contenedores. En particular, se crea una tabla DOCKER para manejar las reglas relativas a los contenedores al reenviar el tráfico de la tabla FORWARD a esta nueva tabla.

```
$ iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
DOCKER-ISOLATION  all  --  anywhere             anywhere
DOCKER     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
DOCKER     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain DOCKER (2 references)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             172.18.0.4           tcp dpt:https
ACCEPT     tcp  --  anywhere             172.18.0.4           tcp dpt:http

Chain DOCKER-ISOLATION (1 references)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere
DROP       all  --  anywhere             anywhere
RETURN     all  --  anywhere             anywhere
```

La solución

Si verifica la documentación oficial (https://docs.docker.com/v1.5/articles/networking/) , se ofrece una primera solución para limitar el acceso del contenedor Docker a una IP particular.

```
$ iptables -I DOCKER -i ext_if ! -s 8.8.8.8 -j DROP
```

De hecho, agregar una regla en la parte superior de la tabla DOCKER es una buena idea. No

interfiere con las reglas configuradas automáticamente por Docker, y es simple. Pero faltan dos grandes:

- Primero, ¿qué sucede si necesita acceder desde dos IP en lugar de una? Aquí solo se puede aceptar un src IP, el otro se eliminará sin ninguna forma de evitarlo.
- En segundo lugar, ¿qué pasa si su docker necesita acceso a Internet? Prácticamente ninguna solicitud tendrá éxito, ya que solo el servidor 8.8.8.8 podría responder a ellos.
- Por último, ¿y si quieres añadir otras lógicas? Por ejemplo, dé acceso a cualquier usuario a su servidor web que sirve en el protocolo HTTP, pero limite todo lo demás a una IP particular.

Para la primera observación, podemos usar *ipset* . En lugar de permitir una IP en la regla anterior, permitimos todas las IP desde el conjunto de datos predefinido. Como beneficio adicional, el ipset se puede actualizar sin la necesidad de redefinir la regla de iptable.

```
$ iptables -I DOCKER -i ext_if -m set ! --match-set my-ipset src -j DROP
```

Para la segunda observación, este es un problema canónico para los firewalls: si se le permite comunicarse con un servidor a través de un firewall, entonces el firewall debe autorizar al servidor a responder a su solicitud. Esto se puede hacer mediante la autorización de paquetes que están relacionados con una conexión establecida. Para la lógica docker, da:

```
$ iptables -I DOCKER -i ext_if -m state --state ESTABLISHED,RELATED -j ACCEPT
```

La última observación se centra en un punto: las reglas de iptables son esenciales. De hecho, la lógica adicional para ACEPTAR algunas conexiones (incluida la que concierne a las conexiones ESTABLECIDAS) debe colocarse en la parte superior de la tabla DOCKER, antes de la regla DROP que niega todas las conexiones restantes que no coincidan con el conjunto de cambios.

Como usamos la opción -I de iptable, que inserta reglas en la parte superior de la tabla, las reglas de iptables anteriores deben insertarse en orden inverso:

```
// Drop rule for non matching IPs
$ iptables -I DOCKER -i ext_if -m set ! --match-set my-ipset src -j DROP
// Then Accept rules for established connections
$ iptables -I DOCKER -i ext_if -m state --state ESTABLISHED,RELATED -j ACCEPT
$ iptables -I DOCKER -i ext_if ... ACCEPT // Then 3rd custom accept rule
$ iptables -I DOCKER -i ext_if ... ACCEPT // Then 2nd custom accept rule
$ iptables -I DOCKER -i ext_if ... ACCEPT // Then 1st custom accept rule
```

Con todo esto en mente, ahora puede consultar los ejemplos que ilustran esta configuración.

Examples

Limite el acceso en los contenedores Docker a un conjunto de IPs

Primero, instale el *ipset* si es necesario. Por favor refiérase a su distribución para saber cómo hacerlo. Como ejemplo, aquí está el comando para distribuciones similares a Debian.

```
$ apt-get update
$ apt-get install ipset
```

Luego cree un archivo de configuración para definir un ipset que contenga las IP para las que desea abrir el acceso a sus contenedores de Docker.

```
$ vi /etc/ipfriends.conf
# Recreate the ipset if needed, and flush all entries
create -exist ipfriends hash:ip family inet hashsize 1024 maxelem 65536
flush
# Give access to specific ips
add ipfriends XXX.XXX.XXX.XXX
add ipfriends YYY.YYY.YYY.YYY
```

Cargue este ipset.

```
$ ipset restore < /etc/ipfriends.conf
```

Asegúrese de que su demonio Docker se esté ejecutando: no se debe mostrar ningún error después de ingresar el siguiente comando.

```
$ docker ps
```

Estás listo para insertar tus reglas de iptables. **Debe** respetar el orden.

```
// All requests of src ips not matching the ones from ipset ipfriends will be dropped.
$ iptables -I DOCKER -i ext_if -m set ! --match-set ipfriends src -j DROP
// Except for requests coming from a connection already established.
$ iptables -I DOCKER -i ext_if -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Si desea crear nuevas reglas, deberá eliminar todas las reglas personalizadas que haya agregado antes de insertar las nuevas.

```
$ iptables -D DOCKER -i ext_if -m set ! --match-set ipfriends src -j DROP
$ iptables -D DOCKER -i ext_if -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Configurar acceso de restricción cuando se inicia el demonio Docker

Trabajo en progreso

Algunas reglas personalizadas de iptables

Trabajo en progreso

Lea Iptables con Docker en línea: https://riptutorial.com/es/docker/topic/9201/iptables-con-docker

Capítulo 26: Modo de enjambre Docker

Introducción

Un enjambre es un número de Docker Engines (o *nodos*) que despliegan *servicios* colectivamente. Swarm se utiliza para distribuir el procesamiento en muchas máquinas físicas, virtuales o en la nube.

Sintaxis

- Inicializar un enjambre : docker swarm init [OPCIONES]
- Únase a un enjambre como nodo y / o administrador : enjambre de docker únase [OPCIONES] HOST: PUERTO
- Cree un nuevo servicio : el servicio de la ventana acoplable crea [OPCIONES] IMAGEN [COMANDO] [ARG ...]
- Visualice información detallada sobre uno o más servicios : servicio de la ventana acoplable inspeccionar [OPCIONES] SERVICIO [SERVICIO ...]
- Lista de servicios : servicio docker ls [OPCIONES]
- Elimine uno o más servicios : Servicio de la ventana acoplable SERVICIO [SERVICIO ...]
- Escala uno o varios servicios replicados : escala de servicio de ventana acoplable SERVICE = REPLICAS [SERVICE = REPLICAS ...]
- Enumere las tareas de uno o más servicios : servicio acoplable ps [OPCIONES] SERVICIO [SERVICIO ...]
- Actualizar un servicio : actualización de servicio de docker [OPCIONES] SERVICIO

Observaciones

El modo enjambre implementa las siguientes características:

- Gestión de clústeres integrada con Docker Engine.
- Diseño descentralizado
- Modelo de servicio declarativo
- Escalada
- Reconciliación del estado deseado
- Redes multi-host
- Descubrimiento de servicio
- Balanceo de carga
- Diseño seguro por defecto

- Actualizaciones de rodadura

Para obtener más documentación oficial de Docker con respecto a Swarm visit: descripción general del modo Swarm

Comandos CLI en modo enjambre

Haga clic en la descripción de comandos para la documentación

Inicializar un enjambre

```
docker swarm init [OPTIONS]
```

Unirse a un enjambre como nodo y / o administrador.

```
docker swarm join [OPTIONS] HOST:PORT
```

Crear un nuevo servicio

```
docker service create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Mostrar información detallada sobre uno o más servicios

```
docker service inspect [OPTIONS] SERVICE [SERVICE...]
```

Servicios de lista

```
docker service ls [OPTIONS]
```

Eliminar uno o más servicios

```
docker service rm SERVICE [SERVICE...]
```

Escala uno o múltiples servicios replicados

```
docker service scale SERVICE=REPLICAS [SERVICE=REPLICAS...]
```

Listar las tareas de uno o más servicios.

```
docker service ps [OPTIONS] SERVICE [SERVICE...]
```

Actualizar un servicio

```
docker service update [OPTIONS] SERVICE
```

Examples

Crea un enjambre en Linux usando docker-machine y VirtualBox

```
# Create the nodes
# In a real world scenario we would use at least 3 managers to cover the fail of one manager.
docker-machine create -d virtualbox manager
docker-machine create -d virtualbox worker1

# Create the swarm
# It is possible to define a port for the *advertise-addr* and *listen-addr*, if none is
defined the default port 2377 will be used.
docker-machine ssh manager \
    docker swarm init \
    --advertise-addr $(docker-machine ip manager)
    --listen-addr $(docker-machine ip manager)

# Extract the Tokens for joining the Swarm
# There are 2 different Tokens for joining the swarm.
MANAGER_TOKEN=$(docker-machine ssh manager docker swarm join-token manager --quiet)
WORKER_TOKEN=$(docker-machine ssh manager docker swarm join-token worker --quiet)


# Join a worker node with the worker token
docker-machine ssh worker1 \
    docker swarm join \
    --token $WORKER_TOKEN \
    --listen-addr $(docker-machine ip worker1) \
    $(docker-machine ip manager):2377
```

Averiguar trabajador y gerente unirse token

Al automatizar el aprovisionamiento de nuevos nodos en un enjambre, necesita saber cuál es el token de unión correcto para el enjambre, así como la dirección anunciada del administrador. Puede averiguarlo ejecutando los siguientes comandos en cualquiera de los nodos de administrador existentes:

```
# grab the ipaddress:port of the manager (second last line minus the whitespace)
export MANAGER_ADDRESS=$(docker swarm join-token worker | tail -n 2 |  tr -d '[[:space:]]')

# grab the manager and worker token
export MANAGER_TOKEN=$(docker swarm join-token manager -q)
export WORKER_TOKEN=$(docker swarm join-token worker -q)
```

La opción -q genera solo el token. Sin esta opción, obtienes el comando completo para registrarte en un enjambre.

Luego, en los nodos recién aprovisionados, puede unirse al enjambre utilizando.

```
docker swarm join --token $WORKER_TOKEN $MANAGER_ADDRESS
```

Hola aplicación mundial

Por lo general, desearía crear una pila de servicios para formar una aplicación replicada y orquestada.

Una aplicación web moderna típica consiste en una base de datos, api, frontend y proxy inverso.

Persistencia

La base de datos necesita persistencia, por lo que necesitamos un sistema de archivos que se comparta entre todos los nodos de un enjambre. Puede ser NAS, servidor NFS, GFS2 o cualquier otra cosa. Configurarlo está fuera de alcance aquí. Actualmente Docker no contiene y no administra la persistencia en un enjambre. Este ejemplo asume que hay `/nfs/` ubicación compartida montada en todos los nodos.

Red

Para poder comunicarse entre sí, los servicios en un enjambre deben estar en la misma red.

Elija un rango de IP (aquí `10.0.9.0/24`) y el nombre de la red (`hello-network`) y ejecute un comando:

```
docker network create \
  --driver overlay \
  --subnet 10.0.9.0/24 \
  --opt encrypted \
  hello-network
```

Base de datos

El primer servicio que necesitamos es una base de datos. Usemos postgresql como ejemplo. Cree una carpeta para una base de datos en `nfs/postgres` y ejecute esto:

```
docker service create --replicas 1 --name hello-db \
       --network hello-network -e PGDATA=/var/lib/postgresql/data \
       --mount type=bind,src=/nfs/postgres,dst=/var/lib/postgresql/data \
       kiasaki/alpine-postgres:9.5
```

Tenga en cuenta que hemos utilizado las `--network hello-network` y `--mount` .

API

La creación de API está fuera del alcance de este ejemplo, así que supongamos que tiene una imagen de API bajo `username/hello-api` .

```
docker service create --replicas 1 --name hello-api \
       --network hello-network \
       -e NODE_ENV=production -e PORT=80 -e POSTGRESQL_HOST=hello-db \
       username/hello-api
```

Tenga en cuenta que hemos pasado un nombre de nuestro servicio de base de datos. El enjambre Docker tiene un servidor DNS de round-robin integrado, por lo que la API podrá conectarse a la base de datos utilizando su nombre DNS.

Proxy inverso

Creemos el servicio nginx para servir nuestra API a un mundo exterior. Cree archivos de configuración nginx en una ubicación compartida y ejecute esto:

```
docker service create --replicas 1 --name hello-load-balancer \
       --network hello-network \
       --mount type=bind,src=/nfs/nginx/nginx.conf,dst=/etc/nginx/nginx.conf \
       -p 80:80 \
       nginx:1.10-alpine
```

Note que hemos usado la opción `-p` para publicar un puerto. Este puerto estaría disponible para cualquier nodo en un enjambre.

Disponibilidad de nodos

Disponibilidad del nodo de modo de enjambre:

- Activo significa que el programador puede asignar tareas a un nodo.
- Pausa significa que el programador no asigna nuevas tareas al nodo, pero las tareas existentes siguen ejecutándose.
- Drenar significa que el programador no asigna nuevas tareas al nodo. El programador cierra todas las tareas existentes y las programa en un nodo disponible.

Para cambiar la disponibilidad del modo:

```
#Following commands can be used on swarm manager(s)
docker node update --availability drain node-1
#to verify:
docker node ls
```

Promover o degradar nodos enjambre

Para promocionar un nodo o un conjunto de nodos, ejecute el programa de la `docker node promote` desde un nodo administrador:

```
docker node promote node-3 node-2

Node node-3 promoted to a manager in the swarm.
Node node-2 promoted to a manager in the swarm.
```

Para degradar un nodo o conjunto de nodos, ejecute `docker node demote` desde un nodo administrador:

```
docker node demote node-3 node-2
```

```
Manager node-3 demoted in the swarm.
Manager node-2 demoted in the swarm.
```

Dejando el enjambre

Nodo trabajador:

```
#Run the following on the worker node to leave the swarm.

docker swarm leave
Node left the swarm.
```

Si el nodo tiene el rol de *administrador* , recibirá una advertencia sobre el mantenimiento del quórum de administradores. Puede usar --force para salir en el nodo del administrador:

```
#Manager Node

docker swarm leave --force
Node left the swarm.
```

Los nodos que dejaron el Swarm seguirán apareciendo en la salida `docker node ls` .

Para eliminar nodos de la lista:

```
docker node rm node-2

node-2
```

Lea Modo de enjambre Docker en línea: https://riptutorial.com/es/docker/topic/749/modo-de-enjambre-docker

Capítulo 27: Múltiples procesos en una instancia de contenedor

Observaciones

Por lo general, cada contenedor debe albergar un proceso. En caso de que necesite varios procesos en un contenedor (por ejemplo, un servidor SSH para iniciar sesión en su instancia de contenedor en ejecución), podría tener la idea de escribir su propio script de shell que inicie esos procesos. En ese caso, tuvo que cuidar el manejo de la `SIGNAL` (p. Ej., Redirigir un `SIGINT` capturado a los procesos secundarios de su script). Eso no es realmente lo que quieres. Una solución simple es utilizar `supervisord` como el proceso raíz de los contenedores que se ocupa del manejo de `SIGNAL` y de la vida útil de sus procesos secundarios.

Pero tenga en cuenta que esta no es la "forma de ventana acoplable". Para lograr este ejemplo de la forma en la ventana acoplable, debe iniciar sesión en el `docker host` la `docker host` (la máquina en la que se ejecuta el contenedor) y ejecutar la `docker exec -it container_name /bin/bahs` . Este comando le abre un shell dentro del contenedor como lo haría ssh.

Examples

Dockerfile + supervisord.conf

Para ejecutar múltiples procesos, por ejemplo, un servidor web Apache junto con un demonio SSH dentro del mismo contenedor, puede usar `supervisord` .

Cree su archivo de configuración `supervisord.conf` como:

```
[supervisord]
nodaemon=true

[program:sshd]
command=/usr/sbin/sshd -D

[program:apache2]
command=/bin/bash -c "source /etc/apache2/envvars && exec /usr/sbin/apache2 -DFOREGROUND"
```

Luego crea un `Dockerfile` como:

```
FROM ubuntu:16.04
RUN apt-get install -y openssh-server apache2 supervisor
RUN mkdir -p /var/lock/apache2 /var/run/apache2 /var/run/sshd /var/log/supervisor
COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
CMD ["/usr/bin/supervisord"]
```

Entonces puedes construir tu imagen:

```
docker build -t supervisord-test .
```

Después puedes ejecutarlo:

```
$ docker run -p 22 -p 80 -t -i supervisord-test
2016-07-26 13:15:21,101 CRIT Supervisor running as root (no user in config file)
2016-07-26 13:15:21,101 WARN Included extra file     "/etc/supervisor/conf.d/supervisord.conf"
during parsing
2016-07-26 13:15:21,112 INFO supervisord started with pid 1
2016-07-26 13:15:21,113 INFO spawned: 'sshd' with pid 6
2016-07-26 13:15:21,115 INFO spawned: 'apache2' with pid 7
...
```

Lea Múltiples procesos en una instancia de contenedor en línea:
https://riptutorial.com/es/docker/topic/4053/multiples-procesos-en-una-instancia-de-contenedor

Capítulo 28: Ordenamiento de contenidos de Dockerfile

Observaciones

1. Declaración de la imagen base (`FROM`)
2. Metadatos (eg `MAINTAINER` , `LABEL`)
3. Instalar dependencias del sistema (por ejemplo, `apt-get install` , `apk add`)
4. Copiar archivos dependencias de aplicaciones (por ejemplo `bower.json` , `package.json` , `build.gradle` , `requirements.txt`)
5. Instalar dependencias de aplicaciones (por ejemplo, `npm install` , `pip install`)
6. Copiando todo el código base
7. Configuración de configuraciones de tiempo de ejecución predeterminadas (por ejemplo, `CMD` , `ENTRYPOINT` , `ENV` , `EXPOSE`)

Estos pedidos se realizan para optimizar el tiempo de construcción utilizando el mecanismo de almacenamiento en caché incorporado de Docker.

Regla de los pulgares:

> Las partes que cambian a menudo (por ejemplo, código base) deben colocarse cerca del fondo del archivo Docker, y viceversa. Las partes que rara vez cambian (por ejemplo, las dependencias) deben colocarse en la parte superior.

Examples

Dockerfile simple

```
# Base image
FROM python:2.7-alpine

# Metadata
MAINTAINER John Doe <johndoe@example.com>

# System-level dependencies
RUN apk add --update \
    ca-certificates \
    && update-ca-certificates \
    && rm -rf /var/cache/apk/*

# App dependencies
COPY requirements.txt /requirements.txt
RUN pip install -r /requirements.txt

# App codebase
WORKDIR /app
COPY . ./
```

```
# Configs
ENV DEBUG true
EXPOSE 5000
CMD ["python", "app.py"]
```

MAINTAINER quedará en desuso en Docker 1.13, y deberá reemplazarse utilizando LABEL. (Fuente)

Ejemplo: LABEL Maintainer = "John Doe johndoe@example.com"

Lea Ordenamiento de contenidos de Dockerfile en línea: https://riptutorial.com/es/docker/topic/6448/ordenamiento-de-contenidos-de-dockerfile

Capítulo 29: pasar datos secretos a un contenedor en ejecución

Examples

Maneras de pasar secretos en un contenedor.

La forma no muy segura (ya que la `docker inspect` lo mostrará) es pasar una variable de entorno a

```
docker run
```

como

```
docker run -e password=abc
```

o en un archivo

```
docker run --env-file myfile
```

donde myfile puede contener

```
password1=abc password2=def
```

También es posible ponerlos en un volumen.

```
docker run -v $(pwd)/my-secret-file:/secret-file
```

algunas formas mejores, usar

keywhiz https://square.github.io/keywhiz/

vault https://www.hashicorp.com/blog/vault.html

etcd con criptografía https://xordataexchange.github.io/crypt/

Lea pasar datos secretos a un contenedor en ejecución en línea: https://riptutorial.com/es/docker/topic/6481/pasar-datos-secretos-a-un-contenedor-en-ejecucion

Capítulo 30: Red docker

Examples

Cómo encontrar la ip del host del contenedor

Debe averiguar la dirección IP del contenedor que se ejecuta en el host para poder, por ejemplo, conectarse al servidor web que se ejecuta en él.

`docker-machine` es lo que se usa en MacOSX y Windows.

En primer lugar, liste sus máquinas:

```
$ docker-machine ls

NAME      ACTIVE   DRIVER       STATE     URL                         SWARM
default   *        virtualbox   Running   tcp://192.168.99.100:2376
```

Luego seleccione una de las máquinas (la predeterminada se llama predeterminada) y:

```
$ docker-machine ip default

192.168.99.100
```

Creando una red Docker

```
docker network create app-backend
```

Este comando creará una red puente simple llamada `appBackend` . No hay contenedores conectados a esta red por defecto.

Listado de Redes

```
docker network ls
```

Este comando enumera todas las redes que se han creado en el host local de Docker. Incluye la red `bridge` puente predeterminada, la red del host `host` y la red nula `null` . Todos los contenedores se adjuntan por defecto a la red `bridge` puente predeterminada.

Agregar contenedor a la red

```
docker network connect app-backend myAwesomeApp-1
```

Este comando adjunta el `myAwesomeApp-1` a la red de `app-backend` . Cuando agrega un contenedor a una red definida por el usuario, el sistema de resolución de DNS incorporado (que no es un

servidor DNS con todas las funciones y no es exportable) permite que cada contenedor en la red resuelva cada otro contenedor en la misma red. Este simple sistema de resolución de DNS no está disponible en la red `bridge` puente predeterminada.

Separar el contenedor de la red

```
docker network disconnect app-backend myAwesomeApp-1
```

Este comando separa el `myAwesomeApp-1` de la red de `app-backend` . El contenedor ya no podrá comunicarse con otros contenedores en la red de la que se ha desconectado, ni utilizar el sistema de resolución de DNS incorporado para buscar otros contenedores en la red de la que se ha desconectado.

Eliminar una red Docker

```
docker network rm app-backend
```

Este comando elimina la red de `app-backend` definida por el usuario del host Docker. Todos los contenedores en la red que no estén conectados a través de otra red perderán la comunicación con otros contenedores. No es posible eliminar la red de `bridge` puente predeterminada, la red del `host` o la red `null` nula.

Inspeccionar una red Docker

```
docker network inspect app-backend
```

Este comando dará detalles sobre la red de `app-backend` .

La salida de este comando debe ser similar a:

```
[
    {
        "Name": "foo",
        "Id": "a0349d78c8fd7c16f5940bdbaf1adec8d8399b8309b2e8a969bd4e3226a6fc58",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Internal": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
```

```
    }
]
```

Lea Red docker en línea: https://riptutorial.com/es/docker/topic/3221/red-docker

Capítulo 31: Registro Docker

Examples

Ejecutando el registro

No use `registry:latest` ! Esta imagen apunta al antiguo registro v1. Ese proyecto Python ya no se está desarrollando. El nuevo registro v2 está escrito en Go y se mantiene activamente. Cuando las personas se refieren a un "registro privado" se refieren al registro v2, ¡ *no* al registro v1!

```
docker run -d -p 5000:5000 --name="registry" registry:2
```

El comando anterior ejecuta la versión más reciente del registro, que se puede encontrar en el proyecto Docker Distribution .

Para obtener más ejemplos de funciones de administración de imágenes, como etiquetar, tirar o empujar, consulte la sección sobre administración de imágenes.

Configure el registro con el servidor de almacenamiento AWS S3.

Configurar un registro privado para usar un backend de AWS S3 es fácil. El registro puede hacer esto automáticamente con la configuración correcta. Aquí hay un ejemplo de lo que debería estar en su archivo `config.yml` :

```
storage:
    s3:
        accesskey: AKAAAAAACCCCCCCBBBDA
        secretkey: rn9rjnNuX44iK+26qpM4cDEoOnonbBW98FYaiDtS
        region: us-east-1
        bucket: registry.example.com
        encrypt: false
        secure: true
        v4auth: true
        chunksize: 5242880
        rootdirectory: /registry
```

Los campos `secretkey` `accesskey` y `secretkey` son credenciales IAM con permisos S3 específicos (consulte la documentación para obtener más información). Puede usar las credenciales con la política de `AmazonS3FullAccess` adjunta. La `region` es la región de su cubo S3. El `bucket` es el nombre del cubo. Puede optar por almacenar sus imágenes cifradas con `encrypt` . El campo `secure` es indicar el uso de HTTPS. Por lo general, debería establecer `v4auth` en verdadero, aunque su valor predeterminado sea falso. El campo `chunksize` permite cumplir con el requisito de la API S3 de que las cargas fragmentadas tienen un tamaño de al menos cinco megabytes. Finalmente, `rootdirectory` especifica un directorio debajo de su cubo S3 para usar.

Hay otros backends de almacenamiento que pueden configurarse con la misma facilidad.

Lea Registro Docker en línea: https://riptutorial.com/es/docker/topic/4173/registro-docker

Capítulo 32: Restricción del acceso a la red de contenedores

Observaciones

Ejemplo de redes docker que bloquean el tráfico. Utilícelo como la red cuando inicie el contenedor con `--net` o con la `docker network connect --net` .

Examples

Bloquear el acceso a LAN y salir

```
docker network create -o "com.docker.network.bridge.enable_ip_masquerade"="false" lan-restricted
```

- Bloques
  - LAN local
  - Internet
- No bloquea
  - Host que ejecuta el daemon docker (ejemplo de acceso a `10.0.1.10:22`)

Bloquear el acceso a otros contenedores.

```
docker network create -o "com.docker.network.bridge.enable_icc"="false" icc-restricted
```

- Bloques
  - Contenedores que acceden a otros contenedores en la misma red `icc-restricted` .
- No bloquea
  - Acceso al host ejecutando el demonio docker.
  - LAN local
  - Internet

Bloquee el acceso de los contenedores al host local que ejecuta el demonio docker

```
iptables -I INPUT -i docker0 -m addrtype --dst-type LOCAL -j DROP
```

- Bloques
  - Acceso al host ejecutando el demonio docker.
- No bloquea
  - Tráfico de contenedor a contenedor
  - LAN local
  - Internet

- Redes docker personalizadas que no usan `docker0`

Bloquee el acceso de los contenedores al host local que ejecuta el daemon docker (red personalizada)

```
docker network create --subnet=192.168.0.0/24 --gateway=192.168.0.1 --ip-range=192.168.0.0/25
local-host-restricted
iptables -I INPUT -s 192.168.0.0/24 -m addrtype --dst-type LOCAL -j DROP
```

Crea una red llamada `local-host-restricted` que:

- Bloques
  - Acceso al host ejecutando el demonio docker.
- No bloquea
  - Tráfico de contenedor a contenedor
  - LAN local
  - Internet
  - Acceso originado desde otras redes docker.

Las redes personalizadas tienen nombres de puente como `br-15bbe9bb5bf5` , así que en su lugar usamos subred.

Lea Restricción del acceso a la red de contenedores en línea:
https://riptutorial.com/es/docker/topic/6331/restriccion-del-acceso-a-la-red-de-contenedores

Capítulo 33: seguridad

Introducción

Con el fin de mantener nuestras imágenes actualizadas para los parches de seguridad, necesitamos saber de qué imagen base dependemos.

Examples

Cómo encontrar de qué imagen proviene nuestra imagen.

Como ejemplo, veamos un contenedor de Wordpress.

El archivo Docker comienza con FROM php: 5.6-apache

así que vamos al archivo Docker mencionado anteriormente https://github.com/docker-library/php/blob/master/5.6/apache/Dockerfile

y encontramos DE debian: jessie. Así que esto significa que, para Debian jessie, aparece un parche de seguridad, necesitamos reconstruir nuestra imagen.

Lea seguridad en línea: https://riptutorial.com/es/docker/topic/8077/seguridad

Capítulo 34: Servicios en uso

Examples

Creando un servicio más avanzado.

En el siguiente ejemplo crearemos un servicio con el *visualizador de* nombres. Especificaremos una etiqueta personalizada y volveremos a asignar el puerto interno del servicio de 8080 a 9090. Además, haremos un enlace para montar un directorio externo del host en el servicio.

```
docker service create \
        --name=visualizer \
        --label com.my.custom.label=visualizer \
        --publish=9090:8080 \
        --mount type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock \
        manomarks/visualizer:latest
```

Creando un servicio sencillo.

Este sencillo ejemplo creará un servicio web de hello world que escuchará en el puerto 80.

```
docker service create \
    --publish 80:80 \
    tutum/hello-world
```

Quitando un servicio

Este ejemplo simple eliminará el servicio con el nombre "visualizador":

```
docker service rm visualizer
```

Escalando un servicio

Este ejemplo escalará el servicio a 4 instancias:

```
docker service scale visualizer=4
```

En el modo Docker Swarm no paramos un servicio. Lo reducimos a cero:

```
docker service scale visualizer=0
```

Lea Servicios en uso en línea: https://riptutorial.com/es/docker/topic/8802/servicios-en-uso

Capítulo 35: ventana acoplable inspeccionar diversos campos para clave: valor y elementos de la lista

Examples

varios ejemplos de inspeccionar ventana acoplable

Encuentro que los ejemplos en la `docker inspect` la `docker inspect` parecen mágicos, pero no explican mucho.

La inspección de Docker es importante porque es la forma limpia de extraer información de una `docker inspect -f ... container_id` contenedores en ejecución `docker inspect -f ... container_id`

(o todo el contenedor corriendo)

```
docker inspect -f ... $(docker ps -q)
```

evitando algunos poco fiables

```
docker command | grep or awk | tr or cut
```

Cuando `docker inspect` una `docker inspect` , puede obtener los valores del "nivel superior" fácilmente, con una sintaxis básica como, para un contenedor que ejecuta htop (desde https://hub.docker.com/r/jess/htop/) con un pid ae1

```
docker inspect -f '{{.Created}}' ae1
```

Puede mostrar

```
2016-07-14T17:44:14.159094456Z
```

o

```
docker inspect -f '{{.Path}}' ae1
```

Puede mostrar

```
htop
```

Ahora si extraigo una parte de mi `docker inspect`

Veo

```
"State": { "Status": "running", "Running": true, "Paused": false, "Restarting": false, "OOMKilled": false, "Dead": false, "Pid": 4525, "ExitCode": 0, "Error": "", "StartedAt": "2016-07-14T17:44:14.406286293Z", "FinishedAt": "0001-01-01T00:00:00Z"
```

Así que obtengo un diccionario, Como tiene `{ ...}` y muchas claves: valores.

Entonces el comando

```
docker inspect -f '{{.State}}' ae1
```

devolverá una lista, como

```
{running true false false false false 4525 0 2016-07-14T17:44:14.406286293Z 0001-01-
01T00:00:00Z}
```

Puedo obtener el valor de State.Pid fácilmente

```
docker inspect -f '{{ .State.Pid }}' ae1
```

yo obtengo

```
4525
```

A veces, la ventana acoplable inspecciona da una lista ya que comienza con `[` y termina con `]`

Otro ejemplo, con otro contenedor.

```
docker inspect -f '{{ .Config.Env }}' 7a7
```

da

```
[DISPLAY=:0 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin LANG=fr_FR.UTF-8
LANGUAGE=fr_FR:en LC_ALL=fr_FR.UTF-8 DEBIAN_FRONTEND=noninteractive HOME=/home/gg WINEARCH=win32
WINEPREFIX=/home/gg/.wine_captvty]
```

Para obtener el primer elemento de la lista, agregamos el índice antes del campo requerido y 0 (como primer elemento) después, por lo que

```
docker inspect -f '{{ index ( .Config.Env) 0 }}' 7a7
```

da

```
DISPLAY=:0
```

Obtenemos el siguiente elemento con 1 en lugar de 0 utilizando la misma sintaxis

```
docker inspect -f '{{ index ( .Config.Env) 1 }}' 7a7
```

da

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Podemos obtener el número de elementos de esta lista.

```
docker inspect -f '{{ len .Config.Env }}' 7a7
```

da

```
9
```

Y podemos obtener el último elemento de la lista, la sintaxis no es fácil.

```
docker inspect -f "{{ index .Config.Cmd $[$(docker inspect -format '{{ len .Config.Cmd }}'
$CID)-1]}}" 7a7
```

Lea ventana acoplable inspeccionar diversos campos para clave: valor y elementos de la lista en línea: https://riptutorial.com/es/docker/topic/6470/ventana-acoplable-inspeccionar-diversos-campos-para-clave--valor-y-elementos-de-la-lista

Capítulo 36: Volúmenes de datos Docker

Introducción

Los volúmenes de datos de Docker proporcionan una forma de conservar los datos independientemente del ciclo de vida de un contenedor. Los volúmenes presentan una serie de características útiles, tales como:

Montar un directorio de host dentro del contenedor, compartir datos entre contenedores utilizando el sistema de archivos y preservar los datos si se elimina un contenedor

Sintaxis

- volumen de la ventana acoplable [OPCIONES] [COMANDO]

Examples

Montar un directorio del host local en un contenedor

Es posible montar un directorio de host en una ruta específica en su contenedor usando la opción de línea de comandos `-v` o `--volume` . El siguiente ejemplo montará `/etc` en el host a `/mnt/etc` en el contenedor:

```
(on linux) docker run -v "/etc:/mnt/etc" alpine cat /mnt/etc/passwd
(on windows)  docker run -v "/c/etc:/mnt/etc" alpine cat /mnt/etc/passwd
```

El acceso predeterminado al volumen dentro del contenedor es de lectura y escritura. Para montar un volumen de solo lectura dentro de un contenedor, use el sufijo `:ro` :

```
docker run -v "/etc:/mnt/etc:ro" alpine touch /mnt/etc/passwd
```

Creando un volumen nombrado

```
docker volume create --name="myAwesomeApp"
```

El uso de un volumen con nombre hace que la administración de volúmenes sea mucho más legible para los usuarios. Es posible crear un volumen con nombre usando el comando especificado anteriormente, pero también es posible crear un volumen con nombre dentro de un comando de `docker run` la `--volume docker run` usando la opción de línea de comandos `-v` o `--volume` :

```
docker run -d --name="myApp-1" -v="myAwesomeApp:/data/app" myApp:1.5.3
```

Tenga en cuenta que crear un volumen con nombre en este formulario es similar a montar un

archivo / directorio host como un volumen, excepto que en lugar de una ruta válida, se especifica el nombre del volumen. Una vez creados, los volúmenes nombrados se pueden compartir con otros contenedores:

```
docker run -d --name="myApp-2" --volumes-from "myApp-1" myApp:1.5.3
```

Después de ejecutar el comando anterior, un nuevo envase ha sido creada con el nombre `myApp-2` de la `myApp:1.5.3` imagen, que comparte la `myAwesomeApp` volumen denominado con `myApp-1` . El volumen llamado `myAwesomeApp` se monta en `/data/app` en el `myApp-2` , tal como se monta en `/data/app` en el `myApp-1` .

Lea Volúmenes de datos Docker en línea: https://riptutorial.com/es/docker/topic/1318/volumenes-de-datos-docker

Capítulo 37: Volúmenes de datos y contenedores de datos

Examples

Contenedores de solo datos

¡Los contenedores de solo datos están obsoletos y ahora se consideran un anti-patrón!

En los días de antaño, antes del subcomando de `volume` de Docker, y antes de que fuera posible crear volúmenes con nombre, Docker eliminaba los volúmenes cuando no había más referencias a ellos en ningún contenedor. Los contenedores de solo datos están obsoletos porque Docker ahora ofrece la capacidad de crear volúmenes con nombre, así como muchas más utilidades a través de varios subcomandos de `docker volume` ventana `docker volume` . Los contenedores de solo datos ahora se consideran un anti-patrón por esta razón.

Muchos recursos en la web de los últimos años mencionan el uso de un patrón llamado "contenedor de datos solamente", que es simplemente un contenedor de Docker que existe solo para mantener una referencia a un volumen de datos.

Recuerde que en este contexto, un "volumen de datos" es un volumen de Docker que no se monta desde el host. Para aclarar, un "volumen de datos" es un volumen que se crea con la directiva `VOLUME` Dockerfile, o usando el interruptor `-v` en la línea de comandos en un comando de `docker run` , específicamente con el formato `-v /path/on/container` . Por lo tanto, un "contenedor de solo datos" es un contenedor cuyo único propósito es tener un volumen de datos adjunto, que es usado por el `--volumes-from` en un comando de `docker run` de la `--volumes-from` `docker run` . Por ejemplo:

```
docker run -d --name "mysql-data" -v "/var/lib/mysql" alpine /bin/true
```

Cuando se ejecuta el comando anterior, se crea un "contenedor de solo datos". Es simplemente un contenedor vacío que tiene un volumen de datos adjunto. Entonces fue posible usar este volumen en otro contenedor así:

```
docker run -d --name="mysql" --volumes-from="mysql-data" mysql
```

El contenedor `mysql` ahora tiene el mismo volumen que también está en `mysql-data` .

Dado que Docker ahora proporciona el subcomando de `volume` y los volúmenes con nombre, este patrón ahora está obsoleto y no se recomienda.

Para comenzar con el subcomando de `volume` y los volúmenes con nombre, consulte Creación de un volumen con nombre

Creando un volumen de datos

```
docker run -d --name "mysql-1" -v "/var/lib/mysql" mysql
```

Este comando crea un nuevo contenedor desde la imagen `mysql` . También crea un nuevo volumen de datos, que luego se monta en el contenedor en `/var/lib/mysql` . Este volumen ayuda a cualquier dato dentro de él a persistir más allá de la vida útil del contenedor. Es decir, cuando se elimina un contenedor, también se eliminan los cambios del sistema de archivos. Si una base de datos almacenaba datos en el contenedor y el contenedor se elimina, todos esos datos también se eliminan. Los volúmenes persistirán en una ubicación particular incluso más allá cuando se retire su contenedor.

Es posible usar el mismo volumen en varios contenedores con la `--volumes-from` línea de comandos `--volumes-from` :

```
docker run -d --name="mysql-2" --volumes-from="mysql-1" mysql
```

El contenedor `mysql-2` ahora tiene adjunto el volumen de datos de `mysql-1` , también utilizando la ruta `/var/lib/mysql` .

Lea Volúmenes de datos y contenedores de datos en línea:
https://riptutorial.com/es/docker/topic/3224/volumenes-de-datos-y-contenedores-de-datos

Creditos

S. No	Capítulos	Contributors
1	Empezando con Docker	abaracedo, Aminadav, Braiam, Carlos Rafael Ramirez, Community, ganesshkumar, HankCa, Josha Inglis, L0j1k, mohan08p, Nathaniel Ford, schumacherj, Siddharth Srinivasan, SztupY, Vishrant
2	API de Docker Engine	Ashish Bista, atv, BMitch, L0j1k, Radoslav Stoyanov, SztupY
3	Cómo configurar la réplica de Mongo de tres nodos con Docker Image y aprovisionado con Chef	Innocent Anigbo
4	Cómo depurar cuando falla la compilación del docker	user2915097
5	Concepto de volúmenes Docker	Amit Poonia, Rob Bednark, serieznyi
6	Construyendo imagenes	cjsimon, ETL, Ken Cochrane, L0j1k, Nathan Arthur, Nathaniel Ford, Nour Chawich, SztupY, user2915097, Wolfgang
7	Contenedores de conexión	Jett Jones
8	Contenedores de control y restauración	Bastian, Fuzzyma
9	Contenedores de correr	abaracedo, Adri C.S., AlcaDotS, atv, Binary Nerd, BMitch, Camilo Silva, Carlos Rafael Ramirez, cizixs, cjsimon, Claudiu, ElMesa, Emil Burzo, enderland, Felipe Plets, ganesshkumar, Gergely Fehérvári, ISanych, L0j1k, Nathan Arthur, Patrick Auld, RoyB, ssice, SztupY, Thomasleveil, tommyyards, VanagaS, Wolfgang, zinking
10	Creando un servicio con persistencia.	Carlos Rafael Ramirez, Vanuan

11	Depuración de un contenedor	allprog, Binary Nerd, foraidt, L0j1k, Nathaniel Ford, user2915097, yadutaf
12	Docker - Modos de red (puente, zonas activas, contenedor asignado y ninguno).	mohan08p
13	Docker en Docker	Ohmen
14	Docker Machine	Amine24h, kubanczyk, Nik Rahmel, user2915097, yadutaf
15	Docker registro privado / seguro con API v2	bastien enjalbert, kubanczyk
16	Docker stats todos los contenedores en ejecución	Kostiantyn Rybnikov
17	Dockerfiles	BMitch, foraidt, k0pernikus, kubanczyk, L0j1k, ob1, Ohmen, rosysnake, satsumas, Stephen Leppik, Thiago Almeida, Wassim Dhif, yadutaf
18	Ejecutando la aplicación Node.js simple	Siddharth Srinivasan
19	ejecutar cónsul en docker 1.12 swarm	Jilles van Gurp
20	Eventos docker	Nathaniel Ford, user2915097
21	Explotación florestal	Jilles van Gurp, Vanuan
22	Gestion de contenedores	akhyar, atv, Binary Nerd, BrunoLM, Carlos Rafael Ramirez, Emil Burzo, Felipe Plets, ganesshkumar, L0j1k, Matt, Nathaniel Ford, Rafal Wiliński, Sachin Malhotra, serieznyi, sk8terboi87 ツ, tommyyards, user2915097, Victor Oliveira Antonino, Wolfgang, Xavier Nicollet, zygimantus
23	Gestionando imagenes	akhyar, Björn Enochsson, dsw88, L0j1k, Nathan Arthur, Nathaniel Ford, Szymon Biliński, user2915097, Wolfgang, zygimantus
24	Inspeccionando un contenedor corriendo	AlcaDotS, devopskata, Felipe Plets, h3nrik, Jilles van Gurp, L0j1k, Milind Chawre, Nik Rahmel, Stephen Leppik, user2915097, yadutaf
25	Iptables con Docker	Adrien Ferrand

26	Modo de enjambre Docker	abronan, Christian, Farhad Farahi, Jilles van Gurp, kstromeiraos , kubanczyk, ob1, Philip, Vanuan
27	Múltiples procesos en una instancia de contenedor	h3nrik, Ohmen, Xavier Nicollet
28	Ordenamiento de contenidos de Dockerfile	akhyar, Philip
29	pasar datos secretos a un contenedor en ejecución	user2915097
30	Red docker	HankCa, L0j1k, Nathaniel Ford
31	Registro Docker	Ashish Bista, L0j1k
32	Restricción del acceso a la red de contenedores	xeor
33	seguridad	user2915097
34	Servicios en uso	Mateusz Mrozewski, Philip
35	ventana acoplable inspeccionar diversos campos para clave: valor y elementos de la lista	user2915097
36	Volúmenes de datos Docker	James Hewitt, L0j1k, NRKirby, Nuno Curado, Scott Coates, t3h2mas
37	Volúmenes de datos y contenedores de datos	GameScripting, L0j1k, melihovv