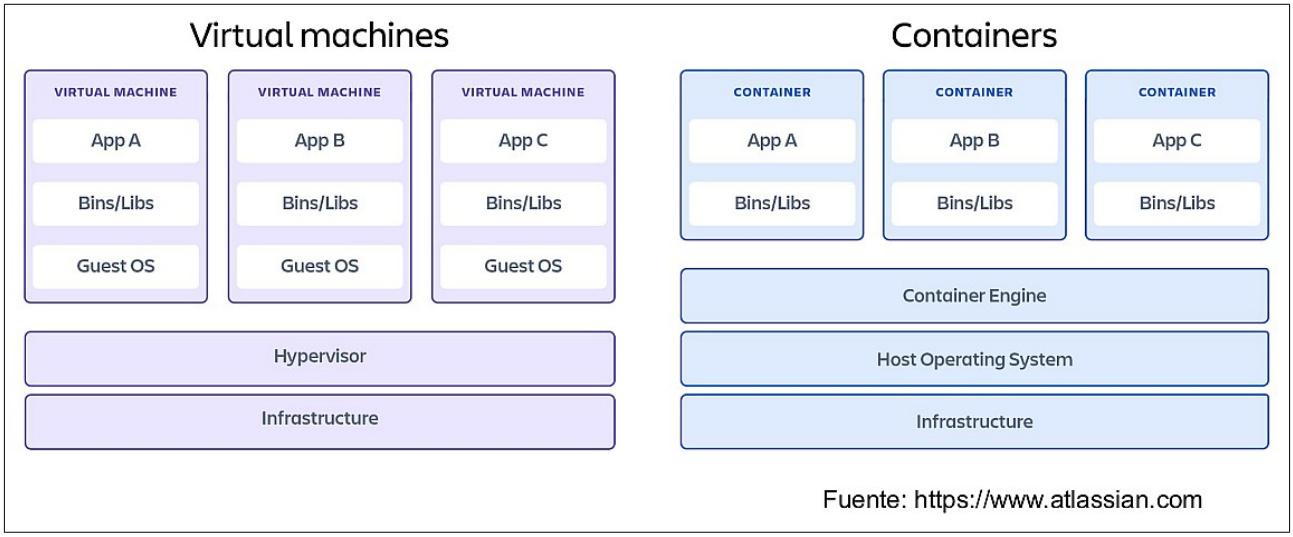
Fuente: https://raul-profesor.github.io/DEAW/introduction/

Según la Wikipedia: "Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Docker utiliza características de aislamiento de recursos del kernel Linux, tales como cgroups y espacios de nombres (namespaces) para permitir que 'contenedores' independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.".



Índice

¿A quién le puede interesar usar docker?

Docker es útil a administradores de sistemas, pero también a desarrolladores. Uno de los problemas que se presentan durante el desarrollo y despliegue de aplicaciones es encontrarnos con sistemas heterogéneos, no ya entre los desarrolladores, también entre los sistemas de pruebas, pre-producción y producción. Es decir, que los desarrolladores y los sistemas donde se ejecuta la aplicación tienen librerías y sistemas operativos diferentes. ¿Y por qué es un problema? Pues porque la aplicación puede funcionar bien en una distribución de GNU/Linux pero no bien en otra, o ejecutarse bien con la versión de un lenguaje pero no con otra. Para asegurar la calidad de desarrollo tenemos que asegurar que todo el mundo usa las mismas versiones de todas las aplicaciones y librerías necesarios.

Esto es más complicado de lo que parece, porque hay desarrolladores que prefieron una distribución concreta, o incluso sistemas privativos. Incluso los sistemas de pruebas, pre-producción y producción suelen ser distintos. Los sistemas de producción suelen ser más nuevos y potentes y los antiguos se dejan para pruebas y pre-producción.

Otro problema es que un mismo desarrollador o un mismo sistema de despliegue tenga que trabajar en más de un proyecto que requiera versiones distintas de librerías, complicándolo aún más.

Docker viene a solucionar todos estos problemas, tanto para los desarrolladores como para los administradores de sistemas. Con Docker podemos crear entornos aislados con configuraciones que serán exactamente igual siempre.

¿Docker es virtualización?

En GNU/Linux Docker no es virtualizado, no hay un hipervisor. Los procesos que corren dentro de un contenedor de docker se ejecutan con el mismo kernel que la máquina anfitrión. Linux lo que hace es aislar esos procesos del resto de procesos del sistema, ya sean los propios de la máquina anfitrión o procesos de otros contenedores. Además, es capaz de controlar los recursos que se le asignan a esos contenedores (cpu, memoria, red, etc.). Internamente, el contenedor no sabe que lo es y a todos los efectos es una distribución GNU/Linux independiente, pero sin la penalización de rendimiento que tienen los sistemas virtualizados.

Así que, cuando ejecutamos un contenedor, estamos ejecutando un servicio dentro de una distribución construida a partir de una "receta". Esa receta permite que el sistema que se ejecuta sea siempre el mismo, independientemente de si estamos usando Docker en Ubuntu, Fedora o, incluso, sistemas privativos compatibles con Docker. De esa manera podemos garantizar que estamos desarrollando o desplegando nuestra aplicación, siempre con la misma versión de todas las dependencias.

Obviamente, si ejecutamos contenedores GNU/Linux dentro de sistemas privativos, sí habrá virtualización.

Conceptos básicos

Antes de comenzar a instalar y usar docker es importante tener una serie de conceptos claros:

Demonio de docker (docker daemon) : Es el proceso principal de docker. Escucha peticiones a la API y maneja los objetos de docker: imágenes, contenedores, redes, volúmenes. También es capaz de comunicarse con otros demonios para controlar servicios docker.

Cliente de docker (docker client) : Es la principal herramienta que usan los administradores de sistema para interaccionar con el sistema Docker.

Registro de docker (docker registry) : Es el lugar donde se almacenan las imágenes de Docker y desde donde poder descargarlas para reutilizarlas. Docker Hub es el principal registro público de Docker y contiene ya un montón de imágenes listas para ser usadas de multitud de servicios (mysql, wordpress, etc).

Objetos de docker

Cuando usamos Docker, estamos creando y usando imágenes, contenedores, redes o volúmenes, entre otros. A todo esto se le denominan objetos. Veamos los más importantes:

- Imagen (**image**): Plantilla de solo lectura que contiene las instrucciones para crear un contenedor Docker. Pueden estar basadas en otras imagenes, lo cual es habitual.
- Contenedor (**container**): Es una instancia ejecutable de una imagen. Esta instancia puede ser creada, iniciada, detenida, movida o eliminada a través del cliente de Docker o de la API. Las instancias se pueden conectar a una o más redes, sistemas de almacenamiento, o incluso se puede crear una imagen a partir del estado de un contenedor. Se puede controlar cómo de aislado está el contenedor del sistema anfitrión y del resto de contenedores. El contenedor está definido tanto por la imagen de la que procede como de las opciones de configuración que permita. Por ejemplo, la imagen oficial de *MariaDb* permite configurar a través de opciones la contraseña del administrador, de la primera base de datos que se cree, del usuario que la maneja, etc.
- Servicios (**services**): Los servicios permiten escalar contenedor a través de múltiples demonios de Docker, los cuales trabajarán conjuntamente como un enjambre (**swarm**).

Instalación

Docker CE está disponible para los siguientes sistemas GNU/Linux: CentOS, Debian, Fedora y Ubuntu. No todas están en múltiples arquitecturas, pero sí todas soportan *x86_64/amd64*. Si tienes otra arquitectura u otro sistema es mejor que uses una máquina virtual para arrancar una distribución compatible.

Existe dos versiones de Docker, una libre y otra que no lo es. Nos ocuparemos exclusivamente de la primera: Docker CE (Community Edition).

Debido a que, dependiendo de la distribución, la forma de instalarlo difiere, es mejor consultar la documentación oficial para saber como instalar Docker en tu máquina.

- Ubuntu: https://docs.docker.com/install/linux/docker-ce/ubuntu/
- Debian: https://docs.docker.com/install/linux/docker-ce/debian/

Para Ubuntu estas son las órdenes:

```
#Instalar el gestor de claves GPG de Docker:

sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings

#Esta orden es solo una:
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
```

```
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Añade el repositorio de Doker para apt:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

#Instalamos los paquetes necesarios para Docker:
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
```

Para saber si se ha instalado correctamente ejecuta este comando:

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

El resultado es el siguiente:

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
```

En la línea 1 estamos ejecutando el cliente de Docker, y estamos indicando que queremos ejecutar un contenedor a partir de la imagen hello-world del registro público de Docker.

```
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest
```

Si es la primera vez que hemos ejecutado esa imagen, nos aparecerá la línea 2, que indica que la imagen no puede ser encontrada y va a proceder a buscarla, por defecto, en el registro público. Si tenemos conexión a Internet se descargará la imagen y automáticamente creará un contenedor.

Tanto si se ha descargado la imagen o ya estaba descargada, el contenedor se ejecutará, obteniendo el texto de bienvenida que se ve en el cuadro siguiente:

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

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Preparando el entorno de prácticas

Podemos configurar nuestro usuario para usar el cliente sin tener que poner *sudo* delante. Para ello añade tu usuario al grupo de docker. Después cierra y abre de la sesión para actualizar los permisos:

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

Vamos a trabajar en las prácticas con la imagen de wordpress. Puedes traerla con el comando:

```
docker pull wordpress:latest
```

También es necesario traer una herramienta llamada Docker Compose, se utiliza para definir y ejecutar aplicaciones de Docker de varios contenedores. En Compose, se usa un archivo YAML para configurar los servicios de la aplicación. Después, con un solo comando, se crean y se inician todos los servicios de la configuración. La instalamos desde apt:

```
sudo apt install docker-compose
```

Imágenes

Nuestros contenedores se iniciarán a partir de las imágenes. Una imagen es una plantilla de solo lectura que se crea incorporando los requisitos necesarios para cumplir el objetivo para el cual fue creada. Por ejemplo, si estamos creando un proyecto con PHP, incorporará el intérprete del lenguaje de PHP. Si es una página web, incorporará el servidor web (*apache*, *nginx*, etc.).

Crear una imagen desde cero supone un esfuerzo demasiado grande, así que lo normal es partir o usar una ya creada. Para ellos buscaremos en los registros, el lugar donde se almacenan. Hay un registro oficial (https://hub.docker.com), pero nada impide a otras organizaciones, o a nosotros mismo, tener un registro propio. Estos registros pueden ser privados o públicos.

Imaginemos que queremos crear una web con *WordPress*. Si buscamos en el registro encontraremos una imagen llamada *wordpress*, con la etiqueta oficial. La recomendación es que siempre busquemos imágenes oficiales, están mantenidas y bien documentadas.

En la página encontraremos las diferentes opciones que tiene esta imagen para configurarla, aunque las veremos con más detalle más adelante.

Por ahora iniciemos la imagen de wordpress con el siguiente comando:

```
docker run -p 8080:80 wordpress
```

Y comprobaremos como se inicia el contenedor:

```
WordPress not found in /var/www/html - copying now...
Complete! WordPress has been successfully copied to /var/www/html
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName'
 directive globally to suppress this message
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName'
 directive globally to suppress this message
[Tue Feb 20 08:15:06.252733 2024] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.57 (Debian) PHP/8.2.16 configured -- resumin
g normal operations
[Tue Feb 20 08:15:06.252796 2024] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
[Tue Feb 20 08:15:17.392303 2024] [mpm_prefork:notice] [pid 1] AH00170: caught SIGWINCH, shutting down gracefully
```

Vemos en la línea nueva un nuevo parámetro: `-p 8080:80`. Por defecto, un contenedor está totalmente aislado. Pero si estamos montando un blog con *WordPress* vamos a necesitar acceder a él desde el navegador.

Con el parámetro `-p`, versión corta de `--publish`, podemos indicar que estamos enlazando un puerto de la máquina anfitrión con el contenedor. En este caso estamos enlazando el puerto 8080 de la máquina anfitrión con el 80 del contenedor.

No vamos a explicar todas las opciones posibles, el tutorial sería demasiado largo. Puedes consultar la página del manual con `man docker-run` o llamando a la ayuda desde el cliente con `docker run --help`.

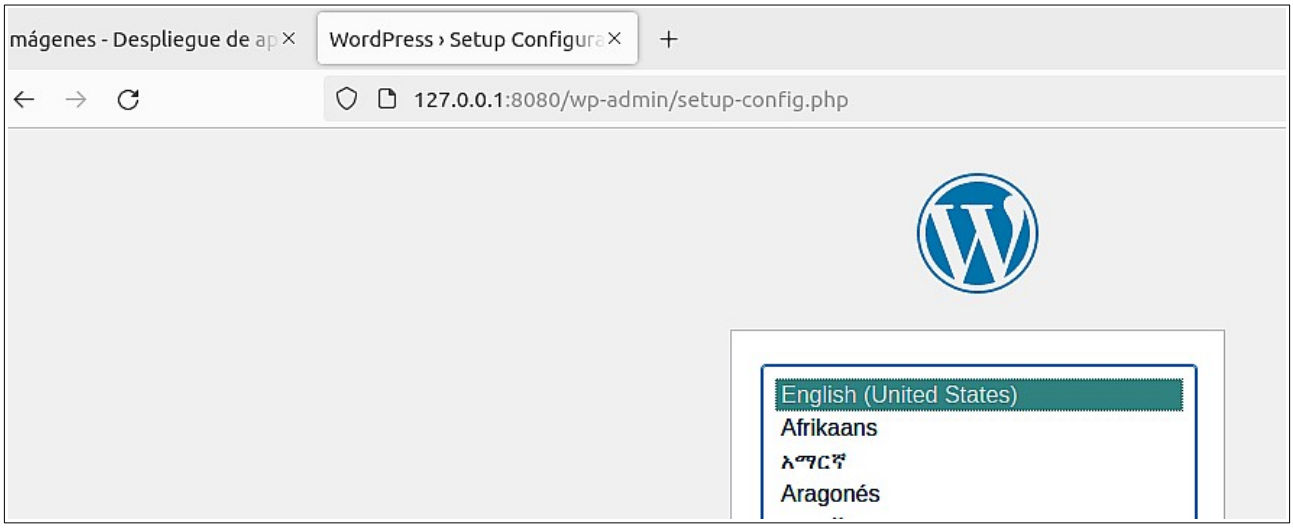
En este caso, el formato de `publish` es:

```
     -p, --publish ip:[hostPort]:containerPort | [hostPort:]containerPort
        Publish a container's port, or range of ports, to the host.

    Both hostPort and containerPort can be specified as a range.  When specifying ranges for both,
the number of ports in ranges should be equal.

    Examples: -p 1234-1236:1222-1224, -p 127.0.0.1:$HOSTPORT:$CONTAINERPORT.

    Use docker port(1) to see the actual mapping, e.g. docker port CONTAINER $CONTAINERPORT.
```

Abrimos la siguiente página web en nuestro navegador:

http://localhost:8080

La cual nos mostrará, como la siguiente imagen, el asistente de instalación de *WordPress*, el cual no vamos a instalar porque necesitamos una base de datos que aún no tenemos.



En su lugar vamos a la consola e interrumpimos la ejecución del contenedor con `Control`+C:

```
^C[DDD mmm dd hh:mm:ss.iiiiii yyyy] [mpm_prefork:notice] [pid 1] AH00169: caught SIGTERM, shutting
down
```

Gestión de imágenes

Las imagenes que nos descargamos se identifican, además de por el nombre, por una versión. De esa manera podemos tener distintas versiones de una misma imagen. En la página del registro de *WordPress* veremos una pestaña con el nombre *Tags*, con las versiones disponibles.

Para usar una en concreto se usa dos puntos seguido del nombre de la versión. Si no se indica nada, como hasta ahora, por defecto se descarga la etiquetada como *latest*. Podemos descargar imágenes con la orden docker pull.

Por ejemplo, podemos descargar una versión concreta de wordpress que utiliza php7.1 en vez de php7.2:

```
docker pull wordpress:php7.1
```

Para ver el listado de images disponibles usamos `docker images`:

```
$ docker images
REPOSITORY   TAG     IMAGE ID      CREATED      SIZE
wordpress    latest  ca0fefec932b  7 days ago   409MB
wordpress    php7.1  37664bd9863e  7 days ago   400MB
```

```
hello-world  latest  4ab4c602aa5e  2 weeks ago  1.84kB
```

Si queremos dejar de usar alguna imagen usaremos `docker rmi`:

```
$ docker rmi wordpress:php7.1
Untagged: wordpress:php7.1
Untagged: wordpress@sha256:2cc529d3d4ac538f8565d18a893bd1308d6f5522422f4696d87267695f69702c
Deleted: sha256:37664bd9863efe67a83cb2ff293f1816a9b5f918668ae19ca36b2af3d3b9f62d
Deleted: sha256:77c97f008777c89455c8e5f248a626b192b62cf07ed1993c9acdfab73be210ee
....
```

Si una imagen está en uso por algún contenedor, no nos dejará eliminarla.

```
$ docker rmi hello-world:latest
Error response from daemon: conflict: unable to remove repository reference "hello-world:latest"
(must force) - container 5ae8bbb8768d is using its referenced image 4ab4c602aa5e
```

Otros comandos para seguimiento y gestión

Se puede consultar la carga en el sistema de los procesos docker mediante estos comandos:

```
$ docker system info
```

```
$ docker info

Client:
 Version:    25.0.0
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.12.1
    Path:     /usr/local/libexec/docker/cli-plugins/docker-buildx
  compose: Docker Compose (Docker Inc.)
    Version:  v2.24.1
    Path:     /usr/local/libexec/docker/cli-plugins/docker-compose

Server:
 Containers: 14
  Running: 3
  Paused: 1
  Stopped: 10
 Images: 52
 Server Version: 25.0.0
 Storage Driver: overlayfs
  driver-type: io.containerd.snapshotter.v1
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 2
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local splunk syslog
 CDI spec directories:
```

Muestra información sobre la versión de docker, el n.º de contenedores total, los que están en ejecución, cuántas imágenes hay, qué redes se han creado, etc.

```
$ docker system df
```

Muestra un resumen del espacio utilizado:

```
$ docker system df -v

Images space usage:

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
my-curl             latest              b2789dd875bf        6 minutes ago       11 MB
my-jq               latest              ae67841be6d0        6 minutes ago       9.623 MB
<none>              <none>              a0971c4015c1        6 minutes ago       11 MB
alpine              latest              4e38e38c8ce0        9 weeks ago         4.799 MB
alpine              3.3                 47cf20d8c26c        9 weeks ago         4.797 MB

Containers space usage:

CONTAINER ID        IMAGE               COMMAND             LOCAL VOLUMES       SIZE
4a7f7eebae0f        alpine:latest       "sh"                1                   0 B
f98f9c2aa1ea        alpine:3.3          "sh"                1                   212 B

Local Volumes space usage:

NAME                                                               LINKS               SIZE
07c7bdf3e34ab76d921894c2b834f073721fccfbbcba792aa7648e3a7a664c2e   2                   36 B
my-named-vol                                                       0                   0 B
```

```
$ docker system prune
```

Se utiliza para limpiar los contenedores, imágenes y redes no utilizadas.

```
$ docker system prune

WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all dangling images
        - unused build cache
Are you sure you want to continue? [y/N] y

Deleted Containers:
f44f9b81948b3919590d5f79a680d8378f1139b41952e219830a33027c80c867
792776e68ac9d75bce4092bc1b5cc17b779bc926ab04f4185aec9bf1c0d4641f

Deleted Networks:
network1
network2

Deleted Images:
untagged: hello-world@sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74
deleted: sha256:1815c82652c03bfd8644afda26fb184f2ed891d921b20a0703b46768f9755c57
deleted: sha256:45761469c965421a92a69cc50e92c01e0cfa94fe026cdd1233445ea00e96289a
```

Con el parámetro -a --volumes también elimina los volúmenes sin uso.

Para limpiar las redes y conexiones que no se utilizan:

```
$ docker network prune
```

Para seguir los ejecutables lanzados, parar un contenedor y/o limpiar docker de los contenedores no empleados se utilizan los siguientes comandos:

```
$ docker ps
```

```
$ docker ps --size

CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS       PORTS   NAMES
SIZE
e90b8831a4b8   nginx          "/bin/bash -c 'mkdir "   11 weeks ago   Up 4 hours           my_nginx
35.58 kB (virtual 109.2 MB)
00c6131c5e30   telegraf:1.5   "/entrypoint.sh"         11 weeks ago   Up 11 weeks
my_telegraf  0 B (virtual 209.5 MB)
```

Con el parámetro –size muestra el espacio ocupado en disco por cada contenedor.

```
$ docker stop CONTAINER_ID
```

Para eliminar los contenedores no en uso:

```
$ docker container prune
```

```
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
4a7f7eebae0f63178aff7eb0aa39cd3f0627a203ab2df258c1a00b456cf20063
f98f9c2aa1eaf727e4ec9c0283bc7d4aa4762fbdba7f26191f26c97f64090360

Total reclaimed space: 212 B
```

Se pueden borrar contenedores con filtros de fecha. En la imagen se borran los contenedores creados hasta el 4 de enero de 2024 :

```
$ docker ps -a --format 'table {{.ID}}\t{{.Image}}\t{{.Command}}\t{{.CreatedAt}}\t{{.Status}}'

CONTAINER ID        IMAGE               COMMAND             CREATED AT                      STATUS
53a9bc23a516        busybox             "sh"                2024-01-04 13:11:59 -0800 PST   Exited
(0) 7 minutes ago
4a75091a6d61        busybox             "sh"                2024-01-04 13:09:53 -0800 PST   Exited
(0) 9 minutes ago

$ docker container prune --force --filter "until=2024-01-04T13:10:00"

Deleted Containers:
4a75091a6d618526fcd8b33ccd6e5928ca2a64415466f768a6180004b0c72c6c

Total reclaimed space: 27 B

$ docker ps -a --format 'table {{.ID}}\t{{.Image}}\t{{.Command}}\t{{.CreatedAt}}\t{{.Status}}'

CONTAINER ID        IMAGE               COMMAND             CREATED AT                      STATUS
53a9bc23a516        busybox             "sh"                2024-01-04 13:11:59 -0800 PST   Exited
(0) 9 minutes ago
```