Throw your container away

As containers are just a thin base layer on top of the host kernel, it is really fast to spin up a new instance if you crashed your old one.

Let's try to run an alpine container and delete the file system.

Spin up the container with `docker container run -ti alpine`

list all the folders on the root level to see the whole distribution:

```
# ls /

bin    etc    lib    mnt    root   sbin   sys    usr
dev    home   media  proc   run    srv    tmp    var
```

Now, delete the whole file system with `rm -rf /`

> **Warning:** Do never try to run this command as a super user in your own environment, as it will delete *everything* on your computer.

Try to navigate around to see how much of the OS is gone

```
# ls
/bin/sh: ls: not found

# whoami
sh: whoami: not found

# date
/bin/sh: date: not found
```

Exit out by `Ctrl+D` and create a new instance of the Alpine image and look a bit around:

```
$ docker container run -ti alpine
# ls /
bin    etc    lib    mnt    root   sbin   sys    usr
dev    home   media  proc   run    srv    tmp    var
```

Try to perform the same tasks as displayed above to see that you have a fresh new instance ready to go.

Auto-remove a container after use

Every time you create a new container, it will take up some space, even though it usually is minimal.

To see what your containers are taking up of space try to run the `docker container ls -as` command.

```
CONTAINER ID        IMAGE                     COMMAND                  CREATED
STATUS                      PORTS
NAMES               SIZE
4b09b2fe1d8c        alpine                    "/bin/sh"                7 seconds
ago       Exited (1) 1 second ago
silly_jones         0B (virtual 3.97MB)
```

Here you can see that the alpine image itself takes 3.97MB, and the container itself takes 0B. When you begin to manipulate files in your container, the size of the container will rise.

If you are creating a lot of new containers eg. to test something, you can tell the Docker daemon to remove the container once stopped with the `--rm` option: `docker container run --rm -it alpine`

This will remove the container immediately after it is stopped.

Cleaning up containers you do not use anymore

Containers are still persisted, even though they are stopped. If you want to delete them from your server you need to use the `docker container rm` command. `docker container rm` can take either the `CONTAINER ID` or `NAME` as seen above. Try to remove the `hello-world` container:

```
sofus@Praq-Sof:~/git/docker-exercises$ docker container ls -a
CONTAINER ID        IMAGE                     COMMAND                  CREATED
STATUS                      PORTS
NAMES
6a9246ff53cb        hello-world               "/hello"                 18 seconds
ago      Exited (0) 16 seconds ago
ecstatic_cray

sofus@Praq-Sof:~/git/docker-exercises$ docker container rm ecstatic_cray
ecstatic_cray
```

The container is now gone when you execute a `ls -a` command.

> **Tip:** As with Git, you can use any unique part of the container ID to refer to it.

Deleting images

You deleted the container instance above, but not the image of hello-world itself. And as you are now on the verge to become a docker expert, you do not need the hello-world image anymore so let us delete it.

First off, list all the images you have downloaded to your computer:

```
sofus@praq-sal:~$ docker image ls
REPOSITORY                          TAG                 IMAGE ID
CREATED             SIZE
alpine                              latest              053cde6e8953        9
days ago          3.97MB
hello-world                         latest              48b5124b2768
10 months ago       1.84kB
```

Here you can see the images downloaded as well as their size. To remove the hello-world image use the `docker image rm` command together with the id of the docker image.

```
sofus@praq-sal:~$ docker image rm 48b5124b2768
Untagged: hello-world:latest
Untagged:
hello-world@sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Deleted: sha256:48b5124b2768d2b917edcb640435044a97967015485e812545546cbed5cf0233
Deleted: sha256:98c944e98de8d35097100ff70a31083ec57704be0991a92c51700465e4544d08
```

What docker did here was to `untag` the image removing the references to the sha of the image. After the image has no references, it deletes the two layers the image itself is comprised of.

Cleaning up

When building, running and rebuilding images, you download and store a lot of layers. These layers will not be deleted, as docker takes a very conservative approach to clean up.

Docker provides a `prune` command, taking all dangling containers/images/networks/volumes.

> You need version `1.13` or newer to have access to pruning.

- `docker container prune`
- `docker image prune`
- `docker network prune`
- `docker volume prune`

The docker image prune command allows you to clean up unused images. By default, docker image prune only cleans up dangling images. A dangling image is one that is not tagged and is not referenced by any container. To remove all *unused* resources, resources that are not directly used by any existing containers, use the `-a` command as well.

If you want a general cleanup, then `docker system prune` is your friend.

Summary

You have now seen the swiftness of creating a new container from an image, trash it, and create a new one on top of it. You have learned to use `container rm` for deleting containers, `image rm` for

images, `image ls` for listing the images and `container ls -a` to look at all the containers on your host.