First Images and Containers

# Image versions/tags

Enter the command:

**$ docker pull -a busybox**

What does it do extra compared to the normal pull command?

| It pulls/downloads all past (and current) images of busybox from the Docker registry and saves it in our system. |
| --- |

How can you check the images that are present?

| “$ docker image ls” |
| --- |

How do you run or pull a specific version of an image?

| By specifying the tag.  For example.  “$ docker run busybox:1.31.1” |
| --- |

# Run interactively and apply changes

Can you run the latest busybox so that you have access to the command-line and can enter commands in this new container? (Note: <https://docs.docker.com/engine/reference/commandline/run/> and you are running it in an “interactive” way.)

| “$ docker run -it busybox:latest” |
| --- |

Can you run the **bash** shell of a specific ubuntu version in an interactive way?

| Yes.  “$ docker container run -it ubuntu /bin/bash” |
| --- |

What is the hostname of the newly started container?

| The hostname of the newly started container is – ‘6be176b38043’. |
| --- |

## Detach / Re-attach

Now, let's use one of the *niche* features of Docker for detaching it from the

interactive container and then look at the details that Docker manages for this

container. Yes, we can detach it from our container by using the Ctrl+P and Ctrl+Q escape sequence. This escape sequence will detach the TTY from the container and

land us in the Docker host prompt $, however **the container will continue to run**.

Detach from the running container.

Run: **$ docker ps** or **$ docker container ls**

Let’s re-attach to our container:

**$ docker attach your\_container’s\_name**

Test the interactive session and quit by typing “exit”.  
Is the container still running?

| $ docker container run -it ubuntu /bin/bash   * Ctrl+P and Ctrl+Q pressed   $ docker ps   * Shows that the container is still running   $ docker attach 655   * Attached back to the container, opens up the interactive mode   $ exit  $ docker ps   * No active container |
| --- |

## Change contents of the container

Restart the container:

**$ docker run -i -t ubuntu[:TAG] /bin/bash**

Go into the home directory and create a couple of files or directories.

Exit the container and execute the following command:

**$ docker diff your\_container\_id**

* ‘C’ stands for modifications;
* ‘A’ stands for additions
* ‘D’ stands for deletions

# Run daemon in detached mode

Run the following command:

**$ docker run -d ubuntu \**

**/bin/bash -c "while true; do date; sleep 5; done"**

→ The container is launched in detached mode (as a daemon).

You can check the activity through:

**$ docker logs CONT\_ID**

Stop the running container:

**$ docker container stop CONT\_ID**

Text

Description automatically generated

# Management

## Start, stop & restart

We saw how to run containers.  
But you can also stop, start & restart containers.  
Test this by launching an interactive container in a terminal.

Open a 2nd terminal where you type the following commands:

**$ docker stop your\_containerID**

What happens in the 1st console?

| It exits the interactive terminal. |
| --- |

Verify that no containers are running with:

**$ docker ps [-a]**

**$ docker container ls [-a]**

Restart the stopped container:

**$ docker start your\_containerID**

‘docker start’ will not re-attach automatically. Use the 1st terminal to re-attach to the container to verify that it is running. Which command do you use?

| $ docker attach <container\_id>. |
| --- |

Check in the second terminal that the container is running. Use both with and without the ‘-a’ option.

Note: ‘restart’ is stop & start one after the other.

## Pause & Unpause

First, make sure no containers are running.

Let’s start an interactive container in the 1st console.

**$ docker run -i -t ubuntu:20.04 /bin/bash**

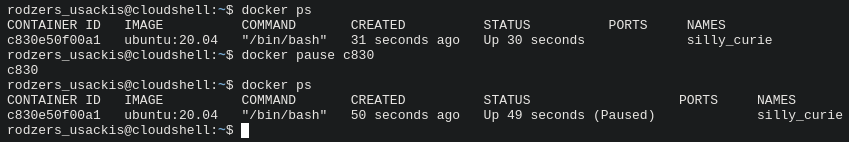
**root@CONT\_ID:/# while true; do date; sleep 5; done**

Now test the following commands in a 2nd console:

**$ docker pause CONT\_ID**

Check the status of the container with “docker ps”. What does it say?

| It says that the container is paused. |
| --- |



**$ docker unpause CONT\_ID**

Stop the container.

## Housekeeping: removing containers

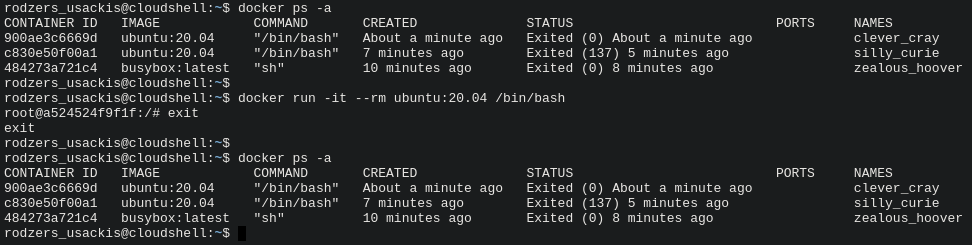
**$ docker ps -a**

This command displays how many stopped containers are available for immediate reuse, but are taking up disk space.

By adding -rm to the docker run command we can “cleanup” the container after it has been stopped. Test this by starting a container with the next command.

**$ docker run -i -t –-rm ubuntu:20.04 /bin/bash**

Stop the running container and check if the container is there or not with ‘docker ps -a’.

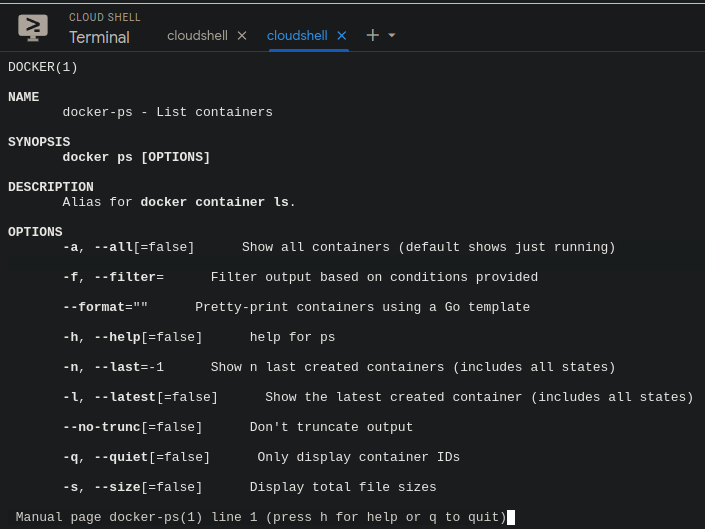


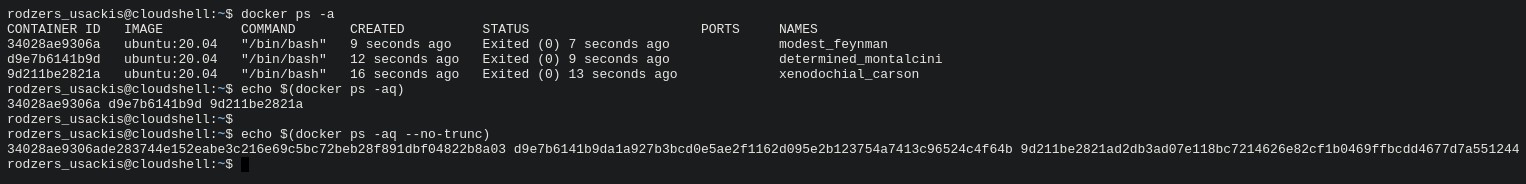
The ‘rm’ command can also be used to clean up stopped containers. (Running containers can’t be removed.)

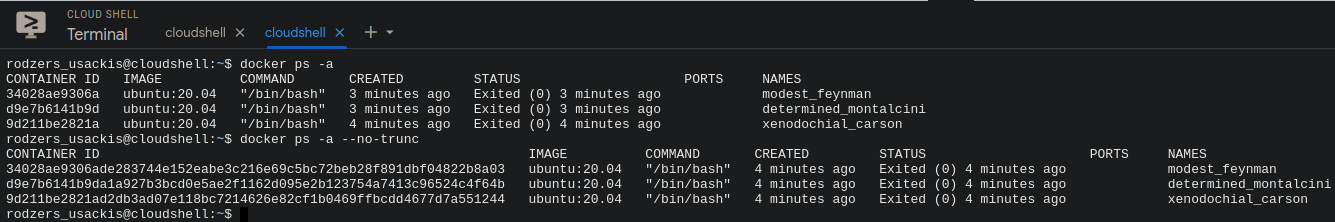
This command removes all non-running containers by first listing them and then running the ‘rm’ command on them = “command expansion”

**$ docker rm $(sudo docker ps -aq --no-trunc)**

What is the meaning of the options ‘-aq’ and ‘--no-trunc’? Why would you add a ‘--no-trunc’?







| “-a” displays all containers  “-q” limits/filters it to display only the container IDs  “--no-trunc” shows the full length of the IDs |
| --- |

## Building our first image

So far, we have crafted a handful of containers by using the standard base images

busybox and ubuntu. In this section, let us see how we can add more software to

our base image on a running container and then convert that container into an

image for future use.

Let's use ubuntu:20.04 as our base image, install the wget application, and then

convert the running container to an image by performing the following steps:

1. Launch an interactive ubuntu:20.04 container by using the docker run subcommand, shown here:

**$ docker run -i -t ubuntu:20.04 /bin/bash**

1. Having launched the container, let's quickly verify if wget is available in our image or not. We have used the which command with wget as an argument for this purpose and, in our case, it returns empty, which essentially means that it could not find any wget installation in this container. This command is run as follows:

**root@472c96295678:/# which wget**

**root@472c96295678:/#**

1. Now let's move on to the next step which involves the wget installation. Since it is a brand new ubuntu container, before installing wget, we must synchronize with the ubuntu package repository, as shown here:

**root@472c96295678:/# apt update**

1. Once the ubuntu package repository synchronization is done, we can continue with installing wget, as shown here:  
     
   **root@472c96295678:/# apt install -y wget**
2. Having completed the wget installation, let's confirm our installation of wget:

**root@472c96295678:/#which wget**

**/usr/bin/wget**

**root@472c96295678:/#**

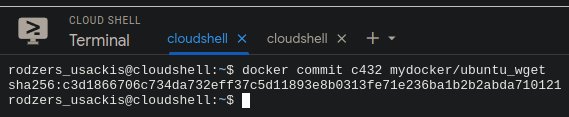
1. Installation of any software would alter the base image composition, which we can trace by using the docker diff subcommand.  
   From a second terminal or screen, issue the docker diff subcommand, as follows:

**$ docker diff 472c96295678**  
  
The preceding command would show a few hundred lines of modification to the ubuntu image. This modification includes the update on package repository, wget binary, and the support files for wget.

1. Finally, let's move to the most important step of committing the image. The docker commit subcommand can be performed on a running or a stopped container. When a commit is performed on a running container, the Docker engine will pause the container during the commit operation in order to avoid any data inconsistency. It is strongly recommended to perform the commit operation on a stopped container. We can commit a container to an image with the docker commit subcommand, as shown here:

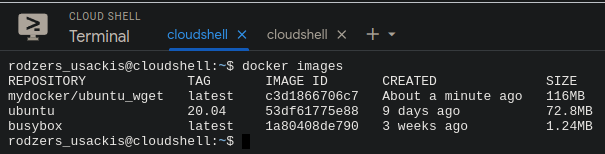
**$ docker commit 472c96295678 mydocker/ubuntu\_wget**

**a530f0a0238654fa741813fac39bba2cc14457aee079a7ae1fe1c64dc7e1ac25**

****We have committed our image by using the name mydocker/ubuntu\_wget. Step by step, we saw how to create an image from a container. Display the list of images of your Docker host and see if this newly created image is a part of the image list.

**$ docker images**

Is the image present?



| Yes, the image is present. |
| --- |