Docker

**Cheat sheet:**

<https://www.docker.com/sites/default/files/d8/2019-09/docker-cheat-sheet.pdf>

<https://dev.to/ankushsinghgandhi/docker-cheat-sheet-56cc>

<https://devhints.io/docker>

<https://stackify.com/docker-tutorial/>

<https://docker-curriculum.com/>

**What is Docker?**

A platform for building,running and shipping applications.

in a consistent manner so if your application works on your development machine it can run and function the same way on other machine.

**Container**: An isolated environment for running an application.

**Container states** – A container can be in one of five states: created,running, paused, exited, restarting.

# 

# Docker Ecosystem Terminology

1. <https://docs.docker.com/glossary/>
2. <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-terminology>
3. <https://node.university/courses/130765/lectures/2789200>
4. <https://developers.redhat.com/blog/2016/01/13/a-practical-introduction-to-docker-container-terminology/>
5. <https://mindmajix.com/docker/basic-terminologies-of-docker>
6. <https://blogs.perficient.com/2021/03/01/docker-bootcamp-container-terminology-and-examples/>
7. <https://www.microfocus.com/documentation/enterprise-developer/ed40pu5/ETS-help/GUID-286C06BD-C95F-498A-9E8C-930829EFC2AA.html>

**Docker Image:** - Docker images are The blueprints of our application which form the basis of containers. We use docker pull command to download an image.

1. A standalone, executable package that can be run in a container.
2. A Docker image is a binary that includes all of the requirements for running a single Docker container, as well as metadata describing its needs and capabilities.
3. An image includes everything that is needed to run an application, including the application's executable code, any software on which the application depends, and any required configuration settings. You can build your own images (using a Dockerfile) or use images that have been built by others and then made available in a registry (such as Docker Hub).
4. To build an image from a Dockerfile you use the docker build command.
5. To run an image in a container you use the docker run command.

**Docker image -** can be compared to the architectural sketch of a house. Similarly, the container is a house that is built with this drawing. The perfect analogy could be OOP (Object Oriented Programming), where the Docker image is compared to class and the Docker container to the object. The Docker container performs all the major computing. A Docker image acts similar to templates for Docker containers.

**Containers** - Created from Docker images and run the actual application. After downloading the image We create a container using **docker run** command. A list of running containers can be seen using the **docker ps** command.

Contains all the necessary components to run one application in isolation. Limits the impact software can have on the host system. Easily removable. Created from an image. Runs the same from one environment to another.

1. A run-time instance of an image.
2. A container is usually completely isolated from the host environment, only able to access host files and ports if it has been configured to do so.
3. To run an image in a container you use the docker run command.

**Daemon** – A process that runs in the background.

**Docker Daemon** - The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system which clients talk to.

**Docker Client** - The command line tool that allows the user to interact with the daemon. More generally, there can be other forms of clients too - such as Kitematic which provide a GUI to the users.

**Docker Hub** - A registry of Docker images. You can think of the registry as a directory of all available Docker images. If required, one can host their own Docker registries and can use them for pulling images.

**Docker Store**

A cloud-based resource that is similar to Docker Hub, but the images on Docker Store have been submitted by commercial entities who have been approved or certified by Docker.

**Dockerfile:**

A text document containing the commands to build a Docker image.

To build an image from a Dockerfile you use the docker build command.

**Running a container:**

docker run hello-world does exactly what it sounds like. It runs an image named “hello-world.”

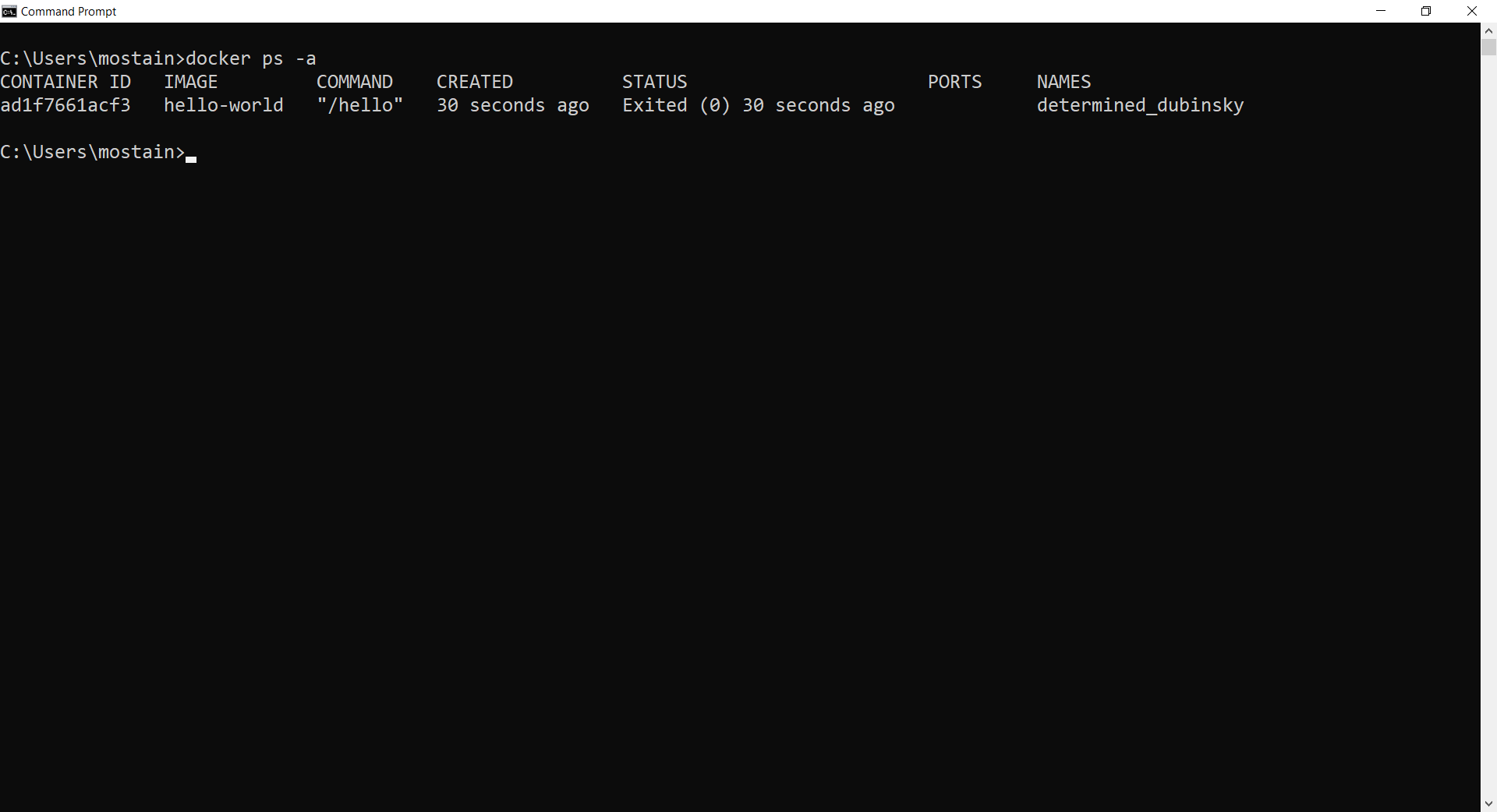
Docker looks for this image on our local system. When it can’t find the image, Docker downloads it from Docker Hub for us.

Hello-world displays a message telling us everything’s working. Then it spells out the process for us before recommending some additional steps.

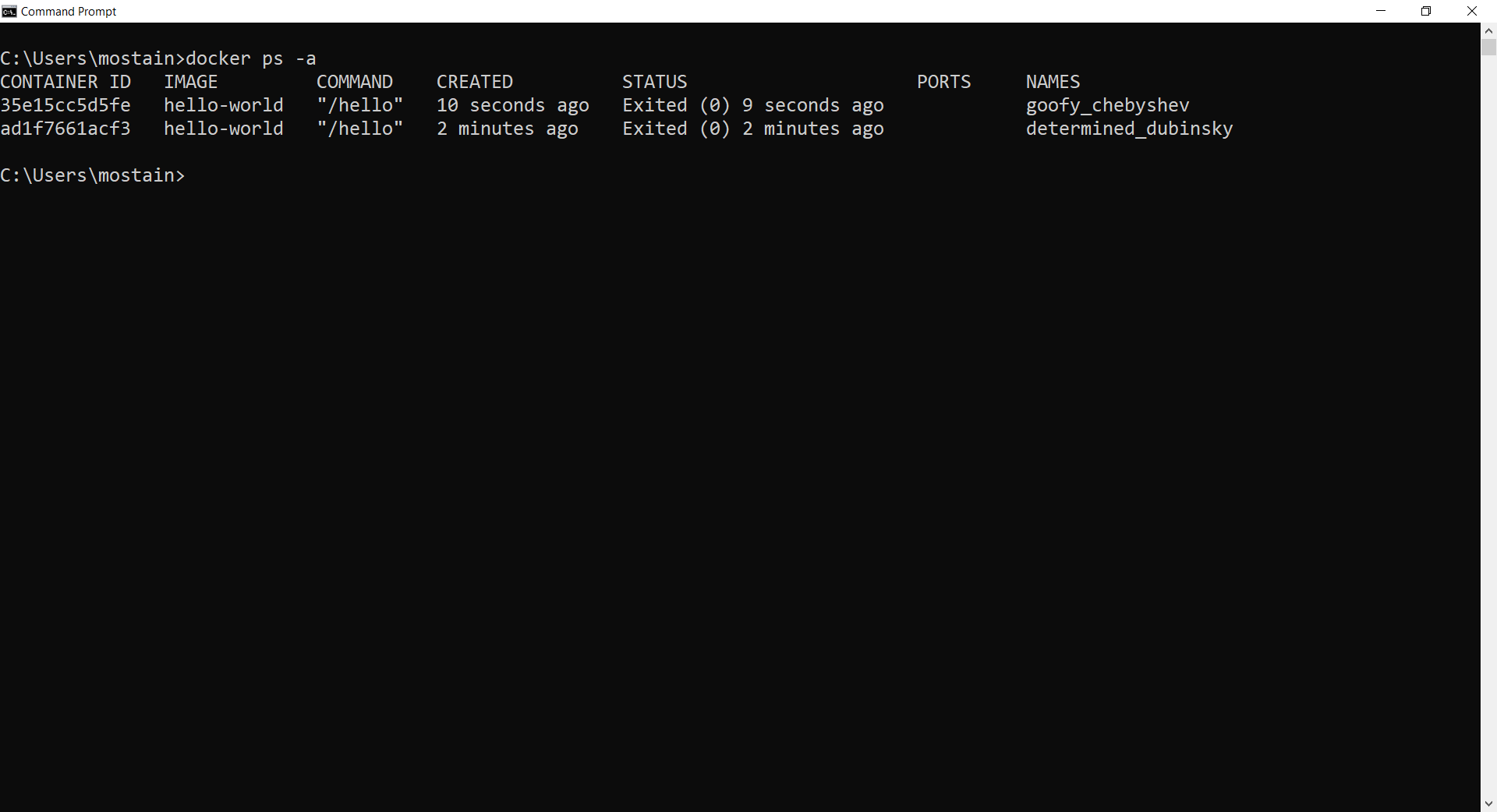
**Under the covers:**

Let’s take a look at a few more Docker commands that tell us more about the environment.

**docker ps -a** lists the containers on our system:



Let’s run this image again with docker run hello-world. The output is almost the same,The output is almost the same…But what does **docker ps -a** show us now?



**docker ps -a** lists all the containers in our system:

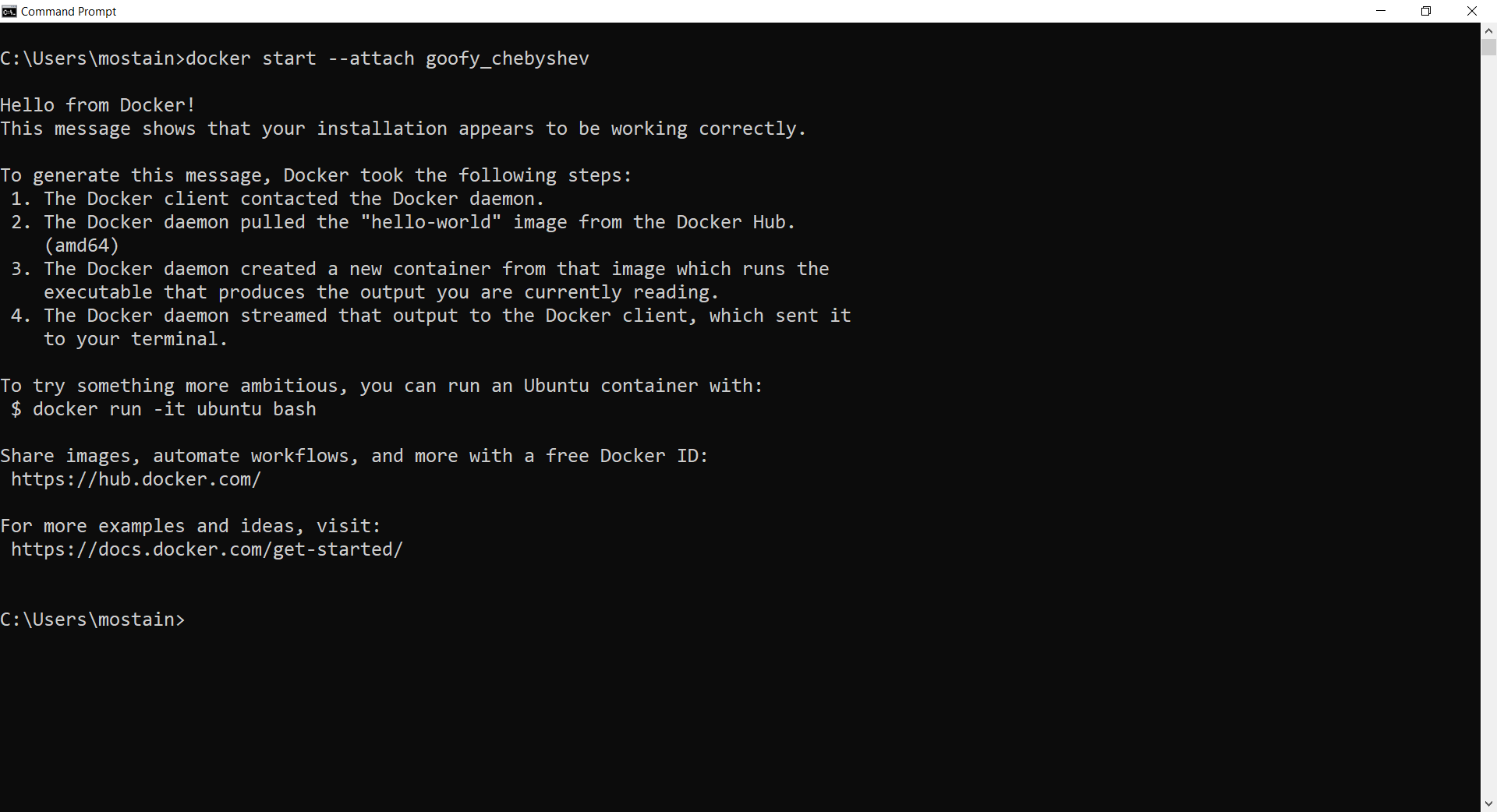
We see two stopped instances of hello-world, with two different names. Docker created an additional container. It didn’t reuse the first. *When we told Docker to run an image named hello-world, it did exactly that; it ran a new instance of the image*. If we want to reuse a container, we refer to it by name.

**Running a container:**

Let’s try starting one of the stopped containers:

**Syntax:** **docker start –attach <containerName or ID>**

**$ docker start --attach goofy\_chebyshev**

****

We used **docker start --attach goofy\_chebyshev** instead of **docker run**. We use the **start** command, and rather than naming the image, we specify the name of a container that’s already loaded. The **–attach** tells Docker to connect to the container output so we can see the results.

We stop containers with **docker stop <container name>** and remove them with

**docker rm <container name>**

**How to delete/remove containers:**

Delete all the process which are exited

docker ps -q -a -f status=exited

docker ps -qa --no-trunc --filter status=exited

docker ps -qa --no-trunc --filter "status=exited"

<https://support.novastor.com/hc/en-us/articles/360020676033-How-to-Enable-Hyper-V>

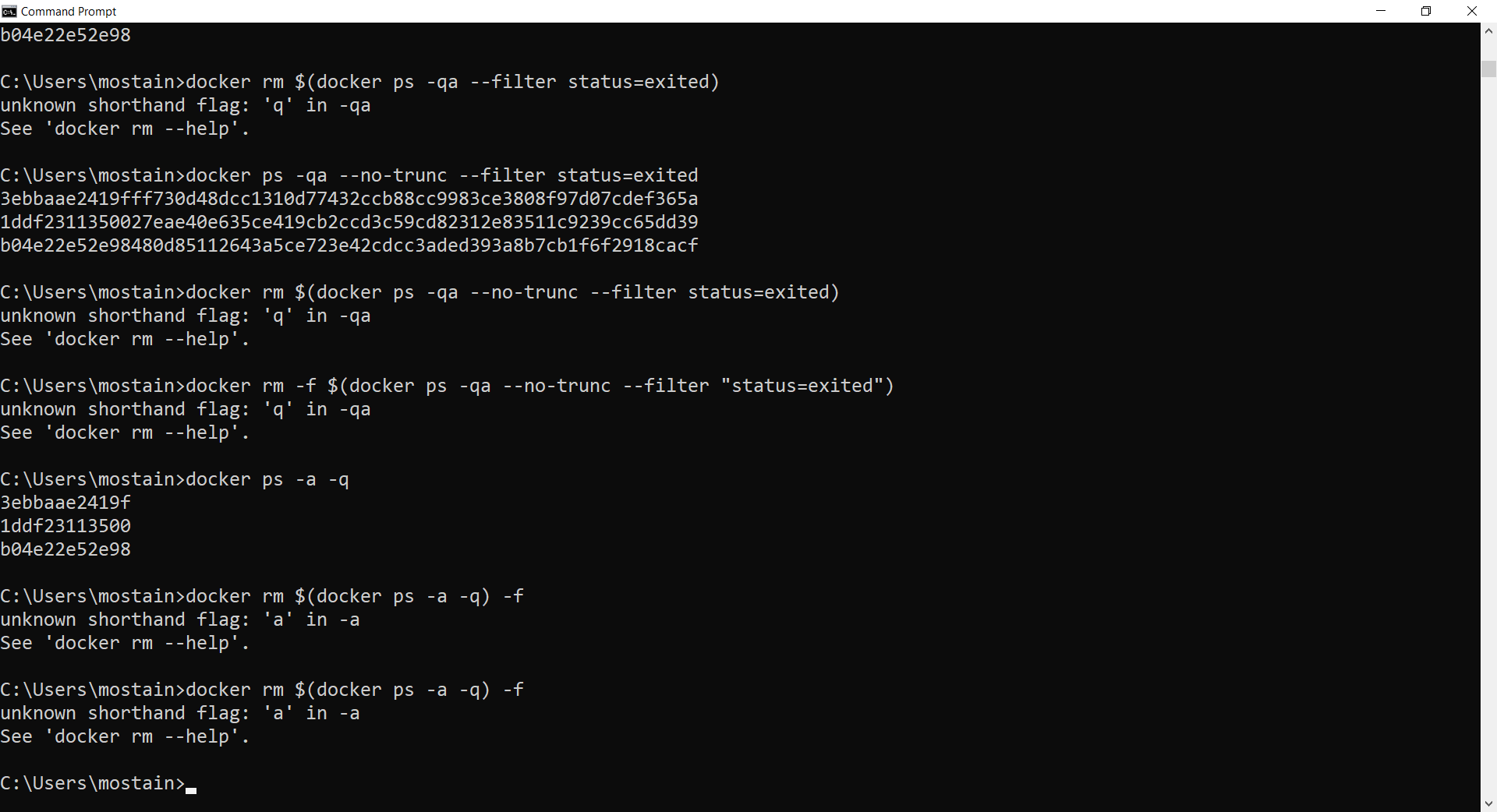
This docker command will remove all stopped containers.

**$ docker rm $(docker ps -a -q)**

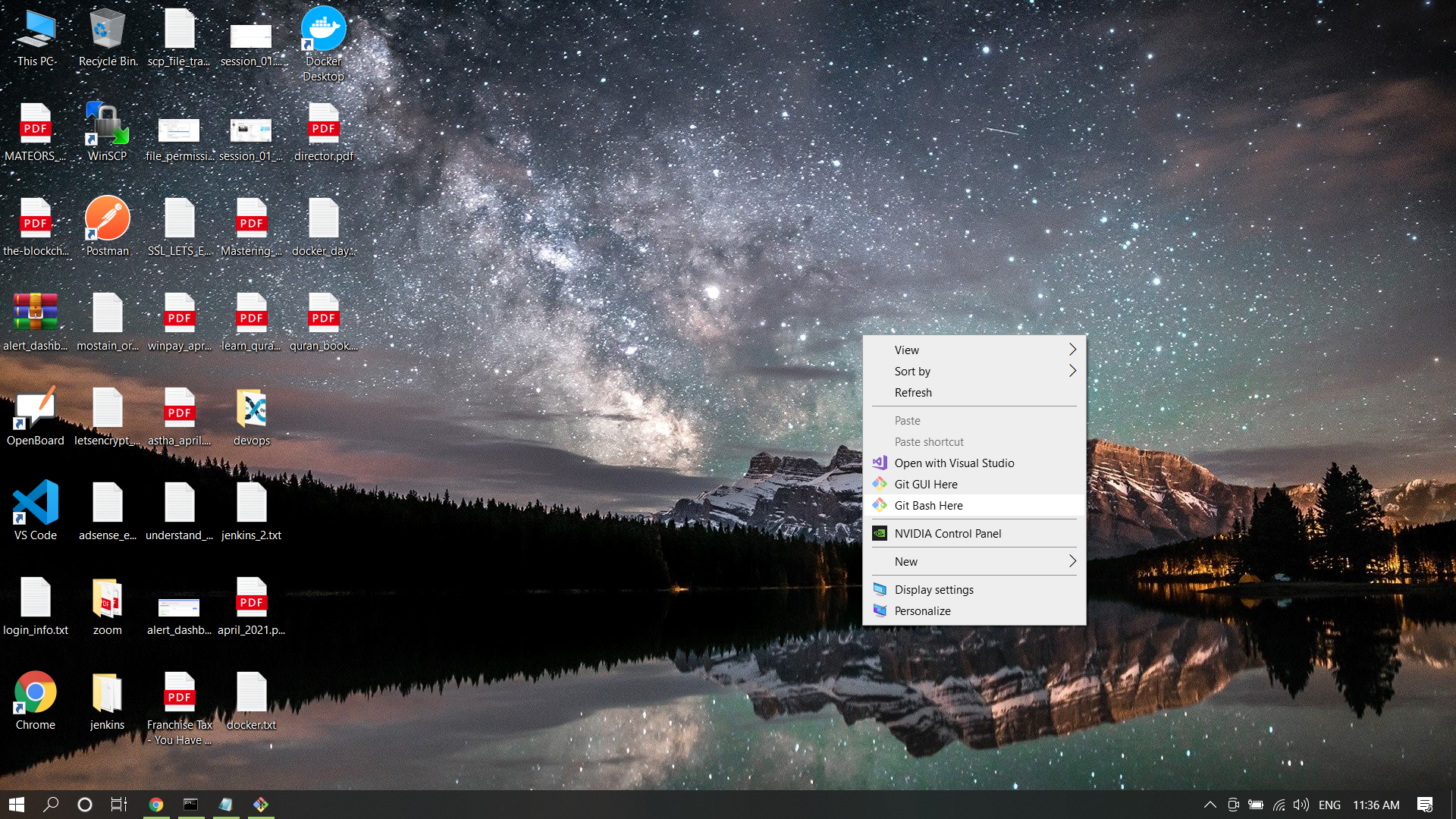
This docker command will remove all containers, even if they are running.

**$ docker rm $(docker ps -a -q) -f**

If you run this command in a regular Windows Command prompt you will get the following output:



The solution is to run this command in a PowerShell terminal or the Docker QuickStart Terminal.Or use Git Bash.



**Foreground:**

docker run -a stdin -a stdout -i -t ubuntu /bin/bash

**How To Remove Docker Images, Containers, and Volumes:**

<https://www.digitalocean.com/community/tutorials/how-to-remove-docker-images-containers-and-volumes>

$ docker system prune

$ docker system prune -a

**The Docker Handbook – 2021 Edition**

https://www.freecodecamp.org/news/the-docker-handbook/#how-to-install-docker-on-windows

* <https://12factor.net/>
* [https://docs.docker.com/develop/develop-images/dockerfile\_best-practices](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/)
* <https://docs.docker.com/engine/reference/commandline/build/>

docker build = Build an image from a Dockerfile

**Usage**: docker build [OPTIONS] PATH | URL | -

**Here is the format of the Dockerfile:**

**#comment**

**INSTRUCTION arguments**

The instruction is not case-sensitive. However, **convention is for them to be UPPERCASE to distinguish them from arguments** more easily.

Docker runs instructions in a Dockerfile in order. **A Dockerfile must begin with a FROM instruction**.

**Parser directives** are optional, and affect the way in which subsequent lines in a Dockerfile are handled. Parser directives do not add layers to the build, and will not be shown as a build step. Parser directives are written as a special type of comment in the form # directive=value. A single directive may only be used once. All parser directives must be at the very top of a Dockerfile

**Dockerfile** -- a text file that contains all commands, in order, needed to build a given image

**FROM =**

The FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction. The image can be any valid image – it is especially easy to start by pulling an image from the Public Repositories.

ARG is the only instruction that may precede FROM in the Dockerfile

FROM can appear multiple times within a single Dockerfile to create multiple images or use one build stage as a dependency for another.

**LABEL** = The LABEL instruction adds metadata to an image

**RUN** = can appear multiple times within a single Dockerfile

**RUN has 2 forms:**

1. RUN <command> (shell form)
2. RUN ["executable", "param1", "param2"] (exec form)

The exec form is parsed as a JSON array, which means that you must use double-quotes (“) around words, not single-quotes (‘).

**CMD** = **There can only be one CMD instruction in a Dockerfile**. If you list more than one CMD then only the last CMD will take effect.

CMD should almost always be used in the form of CMD ["executable", "param1", "param2"…]

**The CMD instruction has three forms:**

1. CMD ["executable","param1","param2"] (exec form, this is the preferred form)
2. CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
3. CMD command param1 param2 **(shell form)**

**EXPOSE** = The EXPOSE instruction indicates the ports on which a container listens for connections

The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

**The EXPOSE instruction does not actually publish the port**. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published. **To actually publish the port when running the container, use the -p flag on docker run to publish and map one or more ports**, or the -P flag to publish all exposed ports and map them to high-order ports.

By default, EXPOSE assumes TCP. You can also specify UDP

To expose on both TCP and UDP, include two lines:

**EXPOSE 80/tcp**

**EXPOSE 80/udp**

**ENV** = The ENV instruction sets the environment variable <key> to the value <value>

To make new software easier to run, you can use ENV to update the PATH environment variable for the software your container installs. For example, ENV PATH=/usr/local/nginx/bin:$PATH ensures that CMD ["nginx"] just works.

**ADD =** The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

ADD has two forms:

ADD [--chown=<user>:<group>] <src>... <dest>

ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]

**COPY** = The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.

COPY has two forms:

COPY [--chown=<user>:<group>] <src>... <dest>

COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]

**ENTRYPOINT** = An ENTRYPOINT allows you to configure a container that will run as an executable.

For example, the following starts nginx with its default content, listening on port 80:

$ docker run -i -t --rm -p 80:80 nginx

Only the last ENTRYPOINT instruction in the Dockerfile will have an effect.

**Dockerfile should specify at least one of CMD or ENTRYPOINT commands.**

**VOLUME** = The VOLUME instruction should be used to expose any database storage area, configuration storage, or files/folders created by your docker container. You are strongly encouraged to use VOLUME for any mutable and/or user-serviceable parts of your image.

**USER** =

**WORKDIR** = For clarity and reliability, you should always use absolute paths for your WORKDIR

**ONBUILD** = An ONBUILD command executes after the current Dockerfile build completes.

**SHELL** =

Note:

The exec form is parsed as a JSON array, which means that you must use double-quotes (“) around words not single-quotes (‘).

Do not confuse RUN with CMD. RUN actually runs a command and commits the result; CMD does not execute anything at build time, but specifies the intended command for the image.

**Hands on Training**

docker run ubuntu

docker run ubuntu sleep 5

docker exec <containerName> cat /etc/hosts

docker attach <containerID>

kodekloud.com/p/docker-labs

**#tag**

docker run redis:4.0

docker run -i <imgName>

docker run -it <imgName>

docker run master/webapp

docker run -p 8080:80 master/webapp

docker run -p 3306:3306 mysql

docker run mysql

docker stop mysql

docker rm mysql

docker run -v /opt/datadir:/var/lib/mysql mysql

docker inspect <containerName>

**#container logs**

docker logs <containerName>

#ENV variables in Docker

docker run -e APP\_COLOR=red myweb-app

#how to create my own image?

FROM ubuntu

#check the validity of Dockerfile using command

$ **docker-compose config**

<https://platformengineer.com/install-vi-bash-curl-on-alpine-linux/>

docker start -a mostain\_alpine -i

cat /etc/\*-release

apk -U add alpine-sdk

apk add go

nano ~/.profile

export GOPATH=/usr/local/bin

source ~/.profile

**ALPINE BASIC COMMANDS**

<https://www.cyberciti.biz/faq/10-alpine-linux-apk-command-examples/>

All installed package:

$ apk info

# Package stats

$ apk stats

#search a package available or not

$ apk search nginx

#install or add a package

$ apk add nginx

**#firewall install**

# apk update && apk upgrade

## Install both IPv4 and IPv6 version of IPtables ##

# apk add ip6tables iptables

## Install awall ##

# apk add -u awall

## Verify it ##

# apk version awall

**$ docker container exec -it <imgName> bash**