Overview

docker - the base command for the Docker CLI, does include a lot of subset commands. Here are some examples and how to use them.

- docker ps
  - The docker ps command only shows running containers by default.
- docker ps -a
  - This shows all active / running and stopped containers
- docker images
  - The default docker images will show all top level images, their repository and tags, and their size.
- docker search
  - By default, it searches the Docker Hub for images
- docker pull
  - Pull an image or a repository from a registry. This will only download the image and will not start any container.
- docker run
  - The docker run command first creates a writeable container layer over the specified image, and then starts it using the specified command. That is, docker run is equivalent to the API /containers/create then /containers/(id)/start.
- docker inspect
  - Docker inspect provides detailed information on constructs controlled by Docker. By default, docker inspect will render results in a JSON array.
- docker network
  - Manage networks, new client do not have the issues, but keep in mind that the client and daemon API must both be at least 1.21 to use this command.
- docker cp
  - The docker cp utility copies the contents of SRC_PATH to the DEST_PATH. You can copy from the container's file system to the local machine or the reverse, from the local filesystem to the container. If - is specified for either the SRC_PATH or DEST_PATH, you can also stream a tar archive from STDIN or to STDOUT.
    The CONTAINER can be a running or stopped container. The SRC_PATH or DEST_PATH can be a file or directory.
    The docker cp command assumes container paths are relative to the container's / (root) directory. This means supplying the initial forward slash is optional; The command sees compassionate_darwin:/tmp/foo/myfile.txt and compassionate_darwin:tmp/foo/myfile.txt as identical. Local machine paths can be an absolute or relative value. The command interprets a local machine's relative paths as relative to the current working directory where docker cp is run.
    The cp command behaves like the Unix cp -a command in that directories are copied recursively with permissions preserved if possible. Ownership is set to the user and primary group at the destination. For example, files copied to a container are created with UID:GID of the root user. Files copied to the local machine are created with the UID:GID of the user which invoked the docker cp command. However, if you specify the

-a option, docker cp sets the ownership to the user and primary group at the source. If you specify the -L option, docker cp follows any symbolic link in the SRC_PATH. docker cp does not create parent directories for DEST_PATH if they do not exist.

- docker container
  - Manage containers. This is the new command, but docker legacy is still interpreted by default. This is why may people that used docker for a while omit the container word.
- docker exec
  - The docker exec command runs a new command in a running container.
    The command started using docker exec only runs while the container's primary process (PID 1) is running, and it is not restarted if the container is restarted.
    COMMAND will run in the default directory of the container. If the underlying image has a custom directory specified with the WORKDIR directive in its Dockerfile, this will be used instead.
    COMMAND should be an executable, a chained or a quoted command will not work.
    Example: docker exec -ti mycontainer "echo a && echo b" will not work, but docker exec -ti mycontainer sh -c "echo a && echo b" will.

Lab

- We will start with the docker ps

```
docker ps
```

OUTPUT: (No running containers)

```
[root@ip-10-1-1-6 /]# docker ps
CONTAINER ID        IMAGE
```

and also check docker ps -a to see if there are any containers in our docker.

```
docker ps -a
```

OUTPUT: (No containers have been run)

```
[root@ip-10-1-1-6 /]# docker ps -a
CONTAINER ID        IMAGE
```

- Let's look at the docker image

```
docker images
```

OUTPUT: (No Images downloaded and available locally)

```
[root@ip-10-1-1-6 /]# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED
SIZE
[root@ip-10-1-1-6 /]#
```

- We will search for the images now

```
docker search nginx
```

OUTPUT:

```
INDEX       NAME                                      DESCRIPTION
STARS     OFFICIAL   AUTOMATED
docker.io   docker.io/nginx                           Official build of Nginx.
14843     [OK]
docker.io   docker.io/jwilder/nginx-proxy             Automated Nginx reverse proxy
for docker c...   2026                [OK]
docker.io   docker.io/richarvey/nginx-php-fpm         Container running Nginx +
PHP-FPM capable ...   813                 [OK]
docker.io   docker.io/jc21/nginx-proxy-manager        Docker container for managing
Nginx proxy ...   186
docker.io   docker.io/linuxserver/nginx               An Nginx container, brought
to you by Linu...   143
docker.io   docker.io/tiangolo/nginx-rtmp            Docker image with Nginx using
the nginx-rt...   124                 [OK]
docker.io   docker.io/jlesage/nginx-proxy-manager     Docker container for Nginx
Proxy Manager        107                 [OK]
docker.io   docker.io/bitnami/nginx                   Bitnami nginx Docker Image
95                  [OK]
docker.io   docker.io/alfg/nginx-rtmp                 NGINX, nginx-rtmp-module and
FFmpeg from s...   94                   [OK]
docker.io   docker.io/nginxdemos/hello                NGINX webserver that serves a
simple page ...   68                   [OK]
docker.io   docker.io/privatebin/nginx-fpm-alpine     PrivateBin running on an
Nginx, php-fpm & ...   53                  [OK]
docker.io   docker.io/nginx/nginx-ingress             NGINX Ingress Controller for
Kubernetes         51
docker.io   docker.io/nginxinc/nginx-unprivileged     Unprivileged NGINX
Dockerfiles                 33
docker.io   docker.io/staticfloat/nginx-certbot       Opinionated setup for
automatic TLS certs ...   21                  [OK]
docker.io   docker.io/schmunk42/nginx-redirect       A very simple container to
redirect HTTP t...   19                   [OK]
docker.io   docker.io/nginx/nginx-prometheus-exporter  NGINX Prometheus Exporter for
NGINX and NG...   17
docker.io   docker.io/centos/nginx-112-centos7        Platform for running nginx
```

```
1.12 or buildin...   15
docker.io   docker.io/blacklabelops/nginx               Dockerized Nginx Reverse
Proxy Server.          13                   [OK]
docker.io   docker.io/centos/nginx-18-centos7           Platform for running nginx
1.8 or building...   13
docker.io   docker.io/bitwarden/nginx                   The Bitwarden nginx web
server acting as a...   12
docker.io   docker.io/flashspys/nginx-static            Super Lightweight Nginx Image
10                   [OK]
docker.io   docker.io/bitnami/nginx-ingress-controller   Bitnami Docker Image for
NGINX Ingress Con...   8                    [OK]
docker.io   docker.io/mailu/nginx                       Mailu nginx frontend
8                    [OK]
docker.io   docker.io/ansibleplaybookbundle/nginx-apb    An APB to deploy NGINX
2                    [OK]
docker.io   docker.io/wodby/nginx                       Generic nginx
1                    [OK]
```

- We will pull an nginx down from the list.

```
docker pull nginx
```

OUTPUT:

```
[root@ip-10-1-1-6 /]# docker pull nginx
Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
f7ec5a41d630: Pull complete
aa1efa14b3bf: Pull complete
b78b95af9b17: Pull complete
c7d6bca2b8dc: Pull complete
cf16cd8e71e0: Pull complete
0241c68333ef: Pull complete
Digest: sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412
Status: Downloaded newer image for docker.io/nginx:latest
[root@ip-10-1-1-6 /]#
```

Now let’s check out docker images

```
docker images
```

OUTPUT:

```
[root@ip-10-1-1-6 /]# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED
SIZE
docker.io/nginx               latest              62d49f9bab67        3 weeks ago
133 MB
[root@ip-10-1-1-6 /]#
```

NOTE	You can see by default we got the latest version of NGINX. You can specify a version if needed.

- Let's run a python shell and then we can use the nginx image we pull down.

Docker exec allows to explore the running container, and nginx as an example will not have less, therefore we need to use cat, tail etc to view a file.

First of all, let's check what python version we have in out system. From our terminal:

```
python --version
```

OUTPUT:

```
[root@ip-10-1-1-6 /]# python --version
Python 2.7.5
[root@ip-10-1-1-6 /]#
```

We notice that we do not have python 3 install and we need to test our code, therefore we will run a python3 container.

Run Python 3 for example:

```
docker run --rm -it python:3 python
```

OUTPUT:

```
root@ip-10-1-1-6 /]# docker run --rm -it python:3 python
Unable to find image 'python:3' locally
Trying to pull repository docker.io/library/python ...
3: Pulling from docker.io/library/python
bd8f6a7501cc: Pull complete
44718e6d535d: Pull complete
efe9738af0cb: Pull complete
f37aabde37b8: Pull complete
3923d444ed05: Pull complete
1ecef690e281: Pull complete
1c053581d9c9: Pull complete
81182ea718cf: Pull complete
83ebd4edf9af: Pull complete
Digest: sha256:0813df59b3d73a13fc581fd416d7733a2de6540d7e3f7633a1a9aabf9b201548
Status: Downloaded newer image for docker.io/python:3
```

NOTE Because we do not have it in out local inventory, we will download and run it.

Once you get the python3 prompt:

```
Python 3.9.5 (default, May  4 2021, 18:15:18)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Then use the following code:

```
print("hello workshop")
```

OUTPUT:

```
>>> print('hello workshop')
hello workshop
>>>
```

Exit out from the container and you will see the container is gone. Verify it with

```
docker ps
```

also check if the container stopped

```
docker ps -a
```

This is the magic flag we used --rm which will clean up after the termination of the container.

To resume, docker run command is docker's standard tool to help you start, stop, and run your containers. The --rm flag will simply tell the Docker Daemon to clean up the container and remove the file system after the container exits. There are use cases where you may not want to do this but it helps you save disk space after running short-lived containers like this one, that we only started to print "Hello, World!".

Now let's run the nginx container. We have pull down the image and we will spin up the container based on that.

Start Nginx container

```
docker run -d --name ng -p 8800:80 nginx
```

OUTPUT:

```
[root@ip-10-1-1-6 /]# docker run -d --name ng -p 8800:80 nginx
2eba70f7b89ccc2c752fbd5e53188a5d67ce160eda0e511a2e48f34619bca160
[root@ip-10-1-1-6 /]#
```

first let's inspect the container

```
docker inspect ng
```

OUTPUT:

```
[root@ip-10-1-1-6 /]# docker inspect ng
[
    {
        "Id": "2eba70f7b89ccc2c752fbd5e53188a5d67ce160eda0e511a2e48f34619bca160",
        "Created": "2021-05-11T18:31:19.010635353Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 3699,
            "ExitCode": 0,
```

```
            "Error": "",
            "StartedAt": "2021-05-11T18:31:19.543420292Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image":
"sha256:62d49f9bab67f7c70ac3395855bf01389eb3175b374e621f6f191bf31b54cd5b",
        "ResolvConfPath":
"/var/lib/docker/containers/2eba70f7b89ccc2c752fbd5e53188a5d67ce160eda0e511a2e48f34619
bca160/resolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/2eba70f7b89ccc2c752fbd5e53188a5d67ce160eda0e511a2e48f34619
bca160/hostname",
        "HostsPath":
"/var/lib/docker/containers/2eba70f7b89ccc2c752fbd5e53188a5d67ce160eda0e511a2e48f34619
bca160/hosts",
        "LogPath": "",
        "Name": "/ng",
        "RestartCount": 0,
        "Driver": "overlay2",
        "MountLabel": "system_u:object_r:svirt_sandbox_file_t:s0:c596,c804",
        "ProcessLabel": "system_u:system_r:svirt_lxc_net_t:s0:c596,c804",
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "journald",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {
                "80/tcp": [
                    {
                        "HostIp": "",
                        "HostPort": "8800"
                    }
                ]
            },
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "CapAdd": null,
            "CapDrop": null,
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
```

```
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "",
            "Cgroup": "",
            "Links": null,
            "OomScoreAdj": 0,
            "PidMode": "",
            "Privileged": false,
            "PublishAllPorts": false,
            "ReadonlyRootfs": false,
            "SecurityOpt": null,
            "UTSMode": "",
            "UsernsMode": "",
            "ShmSize": 67108864,
            "Runtime": "docker-runc",
            "ConsoleSize": [
                0,
                0
            ],
            "Isolation": "",
            "CpuShares": 0,
            "Memory": 0,
            "NanoCpus": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": null,
            "BlkioDeviceReadBps": null,
            "BlkioDeviceWriteBps": null,
            "BlkioDeviceReadIOps": null,
            "BlkioDeviceWriteIOps": null,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DiskQuota": 0,
            "KernelMemory": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": -1,
            "OomKillDisable": false,
            "PidsLimit": 0,
            "Ulimits": null,
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0
        },
        "GraphDriver": {
```

```
            "Name": "overlay2",
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/9736752d591b4a3ebe6ea1c788d8d75eb8619a31160168285e0bf0d2e5ee
ae87-
init/diff:/var/lib/docker/overlay2/54263919b0b59cfcc9c17735dba55143ccc97b627a8da5a70e1
6a61375cf936a/diff:/var/lib/docker/overlay2/f673172af45df4d1eeeea967c7826be048ba51e124
52bc3a72e4ecb78a81eab4/diff:/var/lib/docker/overlay2/a766212d7352dc904b7096961156153cf
23623664789233df97b022b7b1d8bde/diff:/var/lib/docker/overlay2/4591b1f7e99e1567b44c4459
983b04688fe544eb705304021d1aba197eab18bb/diff:/var/lib/docker/overlay2/b347a4cc6e9e147
9a92442972695247c529be3a2a2aae9d27e318f3ee3c395c4/diff:/var/lib/docker/overlay2/b31630
ec15fb9609c0de0f27d213bc828c851da5e11444107934150a3832b315/diff",
                "MergedDir":
"/var/lib/docker/overlay2/9736752d591b4a3ebe6ea1c788d8d75eb8619a31160168285e0bf0d2e5ee
ae87/merged",
                "UpperDir":
"/var/lib/docker/overlay2/9736752d591b4a3ebe6ea1c788d8d75eb8619a31160168285e0bf0d2e5ee
ae87/diff",
                "WorkDir":
"/var/lib/docker/overlay2/9736752d591b4a3ebe6ea1c788d8d75eb8619a31160168285e0bf0d2e5ee
ae87/work"
            }
        },
        "Mounts": [],
        "Config": {
            "Hostname": "2eba70f7b89c",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.19.10",
                "NJS_VERSION=0.5.3",
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "nginx",
                "-g",
                "daemon off;"
            ],
            "Image": "nginx",
            "Volumes": null,
            "WorkingDir": "",
```

```
            "Entrypoint": [
                "/docker-entrypoint.sh"
            ],
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers <docker-maint@nginx.com>"
            },
            "StopSignal": "SIGQUIT"
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"dbd0a7e4a3ec0a6f04ee71e576a7fbdfde83d34f137226ee597e0de78f166cec",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "80/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "8800"
                    }
                ]
            },
            "SandboxKey": "/var/run/docker/netns/dbd0a7e4a3ec",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"017212583f80d38f07ec68e159e3e6ec74a513aedb1b7be437fa99037f2d506b",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"c50eb3ab0250af4ba10ee1c2570b405a1c1d55dcc0ed358050a03a4e3ee9213a",
                    "EndpointID":
"017212583f80d38f07ec68e159e3e6ec74a513aedb1b7be437fa99037f2d506b",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
```

```
                    "MacAddress": "02:42:ac:11:00:02"
                }
            }
        }
    }
]
[root@ip-10-1-1-6 /]#
```

By default the output is in JSON. We want to capture the ip address, there are several ways but here we use the docker inspect -f flag to get that

```
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' ng
```

OUTPUT:

```
[root@ip-10-1-1-6 /]# docker inspect -f
'{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' ng
172.17.0.2
[root@ip-10-1-1-6 /]#
```

Now let's try to see the Host Port.

NOTE	The answer can be found at Docker Inspect Official Documentation

Or you can use the old fashion grep

```
[root@ip-10-1-1-6 /]# docker inspect ng |grep -G5 HostPort
            "NetworkMode": "default",
            "PortBindings": {
                "80/tcp": [
                    {
                        "HostIp": "",
                        "HostPort": "8800"
                    }
                ]
            },
            "RestartPolicy": {
                "Name": "no",
--
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "80/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "8800"
                    }
                ]
            },
            "SandboxKey": "/var/run/docker/netns/dbd0a7e4a3ec",
            "SecondaryIPAddresses": null,
[root@ip-10-1-1-6 /]#
```

- Docker Networks
  In order to establish a communication between the host and the container, docker networking is leveraged.

Let's have some short introduction on all of them.

Bridge network : When you start Docker, a default bridge network is created automatically. A newly-started containers will connect automatically to it. You can also create user-defined custom bridge networks. User-defined bridge networks are superior to the default bridge network. Host network : It remove network isolation between the container and the Docker host, and use the host's networking directly. If you run a container which binds to port 80 and you use host networking, the container's application is available on port 80 on the host's IP address. Means you will not be able to run multiple web containers on the same host, on the same port as the port is now common to all containers in the host network.

None network : In this kind of network, containers are not attached to any network and do not have any access to the external network or other containers. So, this network is used when you want to completely disable the networking stack on a container.

Overlay network : Creates an internal private network that spans across all the nodes participating in the swarm cluster. So, Overlay networks facilitate communication between a docker swarm service and a standalone container, or between two standalone containers on different Docker Daemons.

Macvlan network : Some applications, especially legacy applications or applications which monitor network traffic, expect to be directly connected to the physical network. In this type of situation, you can use the Macvlan network driver to assign a MAC address to each container's virtual network interface, making it appear to be a physical network interface directly connected to the physical network.

Let's take a look at what networks we have

```
docker network ls
```

In my environment I have two extra.

OUTPUT:

```
[root@ip-10-1-1-6 /]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c50eb3ab0250        bridge              bridge              local
987b21d892fc        host                host                local
a7ca4fd5b5ec        kind                bridge              local
7c9275b84237        minikube            bridge              local
d835f485dcef        none                null                local
[root@ip-10-1-1-6 /]#
```

Now let's copy a file over the nginx container we have running.

First let's create a file

```
echo "this is my file" >>workshop
```

Now let's copy it over the container

```
docker cp workshop ng:/workshop
```

Verify the file has been uploaded. A few different ways are possible. Let's evaluate a couple of them.

List / on the container

```
docker exec ng ls /
```

OUTPUT:

```
bin
boot
dev
docker-entrypoint.d
docker-entrypoint.sh
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
workshop
```

We do see the file in the list. Now let's read the file

```
docker exec ng cat /workshop
```

OUTPUT:

```
docker exec ng cat /workshop
this is my file
[root@ip-10-1-1-6 /]#
```

We do see the text we input in our original file. You can copy any scripts, config files etc. There are other solutions as well where you mount a folder but that is not covered at this time.

Build a Container

Lastly, let 's create our custom container

Create a new folder / dir

```
mkdir myfolder
```

Cd into the folder you created, myfolder and create a new file called index.html

```
cd myfolder
touch index.html
```

Edit the file and paste the following:

```
<center><h1>Ciao</h1></center>
```

Now let's create a new file called Dockerfile. Make sure you have the uppercase D, and add the following:

```
FROM nginx:alpine
WORKDIR /usr/share/nginx/html
COPY . .
```

Now let's build the container, do not forget the dot at the end!

```
docker build -t mycontainer .
```

OUTPUT:

```
[root@ip-10-1-1-6 myfolder]# docker build -t mycontainer .
Sending build context to Docker daemon 3.072 kB
Step 1/3 : FROM nginx:alpine
Trying to pull repository docker.io/library/nginx ...
alpine: Pulling from docker.io/library/nginx
540db60ca938: Pull complete
197dc8475a23: Pull complete
39ea657007e5: Pull complete
37afbf7d4c3d: Pull complete
0c01f42c3df7: Pull complete
d590d87c9181: Pull complete
Digest: sha256:07ab71a2c8e4ecb19a5a5abcfb3a4f175946c001c8af288b1aa766d67b0d05d2
Status: Downloaded newer image for docker.io/nginx:alpine
 ---> a64a6e03b055
Step 2/3 : WORKDIR /usr/share/nginx/html
 ---> e36d532eda0a
Removing intermediate container 72bfe9e532af
Step 3/3 : COPY . .
 ---> 85d1c7704e7c
Removing intermediate container ab3279446125
Successfully built 85d1c7704e7c
[root@ip-10-1-1-6 myfolder]#
```

Now let's check the images you have available using our command we used above.

```
docker images
```

OUTPUT:

```
[root@ip-10-1-1-6 myfolder]# docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED
SIZE
mycontainer                   latest              85d1c7704e7c        50 seconds ago
22.6 MB
[root@ip-10-1-1-6 myfolder]#
```

You also will notice that we do not have any container running with the name of mycontainer. To verify that use the

```
docker ps
```

At this point let's run the new container

```
docker run -d --name ciao -p 8200:80 mycontainer
```

OUTPUT:

```
[root@ip-10-1-1-6 myfolder]# docker run -d --name ciao -p 8200:80 mycontainer
68486f2d24c99850c9b7a63ba16cbe4a4b70e8990d3cdc1228793d9bad0a1d08
```

Open Firefox and go to localhost:8200

This will open firefox from the cli and place it in the background.

```
firefox &
```

We will now stop the container

```
docker stop ciao
```

verify the container is not running any longer, by reloading the page in firefox and also using docker ps.

Now let's remove our container. You can use the ID or the name. By default if you do not use the --name flag, the name will be self generated.

```
docker stop mycontainer
```

verify, and you will notice the container is no longer active.

```
docker ps
```

Remove the running **nginx** container now.

1. stop the container
2. remove the container

OLD way

```
docker stop ng
docker rm ng
```

New way

```
docker container stop ng
docker container rm ng
```

OUTPUT:

```
[root@ip-10-1-1-6 /]# docker container stop ng
ng
[root@ip-10-1-1-6 /]# docker container rm ng
ng
[root@ip-10-1-1-6 /]#
```