Docker Commands

Docker version

```
1  root@ubuntu22:~# docker --version
2  Docker version 23.0.3, build 3e7cbfd
```

Docker Hub

- Docker hub hosts all Images; Official and Community.
- Docker hub can be accessible Docker API: https://registry.hub.docker.com/v2
- Docker hub can also be accessible by Docker client using `docker <arguments>` commands

docker search

Search a image on Docker hub.

```
1  docker search MySQL
2  docker search ubuntu
3  docker search httpd
```

To pull image to your computer from Docker hub

```
1  docker pull httpd
2  docker pull httpd:latest
3  docker pull httpd:2.4.57
4  docker pull httpd:2.4
5  docker pull httpd:alpine
6  docker pull httpd:alpine3.18
7  docker pull httpd:bullseye
```

To see the list of available images on our host

```
1  root@ubuntu22:~# docker images
2  REPOSITORY       TAG       IMAGE ID       CREATED       SIZE
3  website-ssl      latest    26a6015f3619   26 hours ago  291MB
4  website          1.0       2d7bde687665   27 hours ago  239MB
5  ubuntu/apache2   latest    edd92437b7eb   3 weeks ago   179MB
6  ubuntu           latest    08d22c0ceb15   5 weeks ago   77.8MB
```

Running your first container

```
1  docker run httpd
```

You will find that the container runs in Foreground and will hijack your command prompt. To come out type Ctrl+Z

Running containers in attached mode

We get the shell of the container

```
1   root@ubuntu22:~# docker run -it ubuntu
2   root@5d1274eb7a19:/#
```

You can see that we are now inside the container.

To confirm find the container os version: `cat /etc/os-release`

Come out of container

There are 2 ways:

1. `exit` will get you of the container and also exits the container

2. `Control p`, `Control q`

Verify with `docker ps`

To see active containers

```
1   root@ubuntu22:~# docker ps
```

Container States

- **Created:** A created container was prepared but never started. You can create a container in advance with docker create in preparation for a job you want to run later. It's also possible for a container to get stuck in the created state. This can happen when it needs a resource that another container already has, such as a network port.
- **Restarting:** A container is in the process of restarting naturally or afer a failure.
- **Running:** A container that's up and running. This indicates that docker start succeeded.
- **Removing:** After you stop a container, it remains available until it's removed. This state indicates that removal has started. If removal state persists, it may also mean it's a large container or there's a problem removing it.
- **Paused:** A container has been paused with docker pause command.
- **Exited:** The command that started the container has exited. This may be because of an exception, the docker stop command, or because the command completed. An exited container isn't consuming any CPU or memory.
- **Dead:** Containers in a dead state aren't operational and can only be removed.

To see all containers including running or exited

```
1   root@ubuntu22:~# docker ps -a
2   CONTAINER ID    IMAGE          COMMAND               CREATED          STATUS                      PORTS
3   aa57b8ddd861    ubuntu         "/bin/bash"           9 minutes ago    Exited (127) 5 minutes ago
4   af14a5dce9b0    website-ssl    "apache2-foreground"  18 hours ago     Up 21 seconds               0.0.0.0:80->80/t
5
```

Stop and Start a Container

```
1   docker stop af14a5dce9b0
2   docker start af14a5dce9b0
```

Removing a container

Before removing a container you ensure the container is stopped.

You can use `docker stop <containerID>`

```
1   docker rm <containerID>
2
3   docker rm -f <containerID>
4
5   #Remove all containers
6   docker rm -f $(docker ps -aq)
```

Running containers in detached mode

In detached mode we don't get the shell of the container. But the container still run in the back.

```
1   root@ubuntu22:~# docker run -itd ubuntu
2   669e377f2e1a7bb596730a2e4b7352f49d62ea3425bb35ff5cc42878e5ae4510
```

Observe the difference between

`docker run -itd ubuntu`

`docker run -d ubuntu`

Observe this

```
1   docker run -d httpd
```

Giving a name to the container

```
1   docker run --name test-container ubuntu
```

Attaching to a container

To attach to a container firstly the container should be in running state.

```
1   root@ubuntu22:~# docker start aa57b8ddd861
2   aa57b8ddd861
3   root@ubuntu22:~# docker attach aa57b8ddd861
4   root@aa57b8ddd861:/#
```

You again get the shell of the container.

Docker Volumes

- When applications run in your container they generate lot of data.
- They can be application specific data or log files or any data.
- When you delete a container the data present in the container is also deleted.

- If you want to preserve the data then you have to use docker volumes.

Create Volume

```
docker volume create
```

Before we can create a Docker volume, let us check the **list of volumes that are already available**:

```
1  root@ubuntu22:~# docker volume ls
2  DRIVER     VOLUME NAME
```

Let us create a Volume

```
1  root@ubuntu22:~# docker volume create vol1
2  vol1
```

List and see the Volumes

```
1  root@ubuntu22:~# docker volume ls
2  DRIVER     VOLUME NAME
3  local      vol1
```

Where is this volume stored?

Inspect the volume:

```
1   root@ubuntu22:~# docker volume inspect vol1
2   [
3       {
4           "CreatedAt": "2023-04-18T09:37:15Z",
5           "Driver": "local",
6           "Labels": null,
7           "Mountpoint": "/var/lib/docker/volumes/vol1/_data",
8           "Name": "vol1",
9           "Options": null,
10          "Scope": "local"
11      }
12  ]
```

Volumes are by default stored in `/var/lib/docker/volumes`

In this case `/var/lib/docker/volumes/vol1/_data` will act as a storage.

Attach volume to a container

```
1  docker run -it --name container1 -v vol1:/data ubuntu
```

`v flag` takes source and destination.

in my case i am mapping `vol1` to a directory in the container called `/data`

```
1  root@ubuntu22:~# docker run -it --name container1 -v vol1:/data ubuntu
2  root@4874024f5d28:/# ls /
3  bin   data  etc   lib    lib64   media  opt   root  sbin  sys  usr
4  boot  dev   home  lib32  libx32  mnt    proc  run   srv   tmp  var
```

We are inside the container.

Now let us go to `/data` folder and create a file with some content.

```
1  root@4874024f5d28:/# cd /data
2  root@4874024f5d28:/data# touch reports.txt
3  root@4874024f5d28:/data# echo "This is an Oracle report" > reports.txt
```

This folder `/data` can also be accessible from the Docker host.

Come out of the container and verify thet data is persistent:

```
1   root@4874024f5d28:/data# exit
2   exit
3   root@ubuntu22:~# docker volume ls
4   DRIVER     VOLUME NAME
5   local      vol1
6   root@ubuntu22:~# docker volume inspect vol1
7   [
8       {
9           "CreatedAt": "2023-04-18T09:37:15Z",
10          "Driver": "local",
11          "Labels": null,
12          "Mountpoint": "/var/lib/docker/volumes/vol1/_data",
13          "Name": "vol1",
14          "Options": null,
15          "Scope": "local"
16      }
17  ]
18  root@ubuntu22:~# cd /var/lib/docker/volumes/vol1/_data
19  root@ubuntu22:/var/lib/docker/volumes/vol1/_data# ls
20  reports.txt
21  root@ubuntu22:/var/lib/docker/volumes/vol1/_data# cat reports.txt
22  This is an Oracle report
23
```

This is how we can share the data from the container to the host.

So we can have a backup of the container data.

Similarly you can also share any data from the docker host to the container:

```
1  root@ubuntu22:/var/lib/docker/volumes/vol1/_data# touch logs.txt
2  root@ubuntu22:/var/lib/docker/volumes/vol1/_data# echo "This logs file is shared from the docker host" > logs.txt
```

Now get into the container and verify the file presence.

The container `"container1"` exited as we came out:

```
1  root@ubuntu22:/var/lib/docker/volumes/vol1/_data# docker ps -a
2  CONTAINER ID    IMAGE         COMMAND          CREATED          STATUS                        PORTS
3  4874024f5d28    ubuntu        "/bin/bash"      44 minutes ago   Exited (130) 7 minutes ago
```

Let us attach to the container.

Since we cannot attach to the stopped container first we need to start it.

```
1  root@ubuntu22:~# docker start container1
2  container1
3  root@ubuntu22:~# docker attach container1
4  root@4874024f5d28:/# cd /data
5  root@4874024f5d28:/data# ls
6  logs.txt  reports.txt
```

```
7   root@4874024f5d28:/data# cat logs.txt
8   This logs file is shared from the docker host
```

> **i** In these kind of volumes, even when the container is lost or deleted, the data can still be accessed.

Let us remove this container and validate that data persists.

```
1   root@4874024f5d28:/data# exit
2   exit
3   root@ubuntu22:~# docker rm container1
4   container1
5   root@ubuntu22:~# docker ps -a
6   #You can see that container1 is deleted.
7
8   #Now verify the data
9   root@ubuntu22:~# ls /var/lib/docker/volumes/vol1/_data
10  logs.txt  reports.txt
```

Attaching volume to multiple containers

Here we already have a volume `vol1`

Let us create a new container with the name container2 and attach the same volume as /data2 directory

```
1   root@ubuntu22:~# docker run -it --name container2 -v vol1:/data2 ubuntu
2   root@8f6a36cda5e4:/# ls /data2
3   logs.txt  reports.txt
```

Come out of the container but still keep the container running using `Control p`, `Control q`

> **i** You can attach the volume to any number of containers

Let us now create another container container3 that simultaneously runs and attach the volume as /data3 directory

```
1   root@ubuntu22:~# docker run -it --name container3 -v vol1:/data3 ubuntu
2   root@9bc518dd5f8e:/# ls /data3
3   logs.txt  reports.txt
```

Come out of the container but still keep the container running using `Control p`, `Control q`

Verify that both containers are running:

```
1   root@ubuntu22:~# docker ps
2   CONTAINER ID    IMAGE     COMMAND       CREATED         STATUS        PORTS      NAMES
3   9bc518dd5f8e    ubuntu    "/bin/bash"   3 minutes ago   Up 3 minutes             container3
4   8f6a36cda5e4    ubuntu    "/bin/bash"   5 minutes ago   Up 5 minutes             container2
```

Running Non-interactive Commands in a Docker Container:

We have mounted the same volume in both containers.

Let us verify that both are mounted and working.

```
1   root@ubuntu22:~# docker exec container2 ls /data2
2   logs.txt
3   reports.txt
```

```
4
5  root@ubuntu22:~# docker exec container3 ls /data3
6  logs.txt
7  reports.txt
```

Another example: Starting a Test Container:

```
1  docker run -d --name container4 alpine watch "date >> /var/log/date.log"
```

This command creates a new Docker container from the official `alpine` image. This is a popular Linux container image that uses Alpine Linux, a lightweight, minimal Linux distribution.

We use the `-d` flag to detach the container from our terminal and run it in the background.

`--name container4` will name the container `container4`.

And finally we have `watch "date >> /var/log/date.log"`. This is the command we want to run in the container.

`watch` will repeatedly run the command you give it, every two seconds by default.

It looks like this:

```
1   root@ubuntu22:~# docker run -d --name container4 alpine watch "date >> /var/log/date.log"
2   Unable to find image 'alpine:latest' locally
3   latest: Pulling from library/alpine
4   f56be85fc22e: Pull complete
5   Digest: sha256:124c7d2707904eea7431fffe91522a01e5a861a624ee31d03372cc1d138a3126
6   Status: Downloaded newer image for alpine:latest
7   50cac9ec4898595a1b4cfa30fe76ba131bbea7f1f71db19682d73c8d49c2778d
8
9   root@ubuntu22:~# docker ps
10  CONTAINER ID    IMAGE     COMMAND               CREATED         STATUS          PORTS      NAMES
11  50cac9ec4898    alpine    "watch 'date >> /var…"  5 seconds ago   Up 4 seconds               container4
12  9bc518dd5f8e    ubuntu    "/bin/bash"           11 minutes ago  Up 11 minutes              container3
13  8f6a36cda5e4    ubuntu    "/bin/bash"           14 minutes ago  Up 14 minutes              container2
14
```

If you need to run a command inside a running Docker container, but don't need any interactivity, use the `docker exec` command without any flags:

```
1  docker exec container4 tail /var/log/date.log
```

It looks like this:

```
1   root@ubuntu22:~# docker exec container4 tail /var/log/date.log
2   Tue Apr 18 10:59:53 UTC 2023
3   Tue Apr 18 10:59:55 UTC 2023
4   Tue Apr 18 10:59:57 UTC 2023
5   Tue Apr 18 10:59:59 UTC 2023
6   Tue Apr 18 11:00:01 UTC 2023
7   Tue Apr 18 11:00:03 UTC 2023
8   Tue Apr 18 11:00:05 UTC 2023
9   Tue Apr 18 11:00:07 UTC 2023
10  Tue Apr 18 11:00:09 UTC 2023
11  Tue Apr 18 11:00:11 UTC 2023
```

Running Commands in an Alternate Directory in a Docker Container

To run a command in a certain directory of your container, use the `--workdir` flag to specify the directory:

```
1   root@ubuntu22:~# docker exec --workdir /tmp container3 pwd
2   /tmp
```

Running Commands as a Different User in a Docker Container

To run a command as a different user inside your container, add the `--user` flag:

Try with the Alpine Linux container we created earlier.

```
1   root@ubuntu22:~# docker exec --user guest container4 whoami
2   guest
```

Passing Environment Variables into a Docker Container

Sometimes you need to pass environment variables into a container along with the command to run.

The `-e` flag lets you specify an environment variable.

Here let us set the Time Zone as Environment variable

```
1   docker exec -e TZ='Asia/Kolkata' container4 env
```

Another example:

```
1   docker exec -e TEST_VARIABLE=dummy_value container4 env
```

Another example:

To set multiple variables, repeat the `-e` flag for each one:

```
1   docker exec -e OWNER=sree@cloudiq.in -e ENVIRONMENT=prod container4 env
```

If you'd like to **pass in a file full of environment variables** you can do that with the `--env-file` flag.

First create the file:

```
1   nano .env
2
3   OWNER=sree@cloudiq.online
4   BILLING=Project_Boeing
5   SUPPORT=Priority
6   ENVIRONMENT=Production
7
```

```
1   docker exec --env-file .env container4 env
```

Running an Interactive Shell in a running Docker Container

If you need to start an interactive shell inside a Docker Container, perhaps to explore the container or install something or debug running processes, use `docker exec` with the `-i` and `-t` flags.

The `-i` flag keeps input open to the container, and the `-t` flag creates a pseudo-terminal that the shell can attach to.

Working with the Alpine Linux Container we created earlier: `container4`

```
1  root@ubuntu22:~# docker exec -i -t container4 /bin/sh
2  / #
3  / # cat /etc/os-release
4  NAME="Alpine Linux"
5  ID=alpine
6  VERSION_ID=3.17.3
7  PRETTY_NAME="Alpine Linux v3.17"
8  HOME_URL="https://alpinelinux.org/"
9  BUG_REPORT_URL="https://gitlab.alpinelinux.org/alpine/aports/-/issues"
```

Working with the Ubuntu Container we created earlier: `container3`

```
1  root@ubuntu22:~# docker exec -i -t container3 /bin/bash
2
3  root@9bc518dd5f8e:/# cat /etc/os-release
4  PRETTY_NAME="Ubuntu 22.04.2 LTS"
5  NAME="Ubuntu"
6  VERSION_ID="22.04"
7  VERSION="22.04.2 LTS (Jammy Jellyfish)"
8  VERSION_CODENAME=jammy
9  ID=ubuntu
10 ID_LIKE=debian
11 HOME_URL="https://www.ubuntu.com/"
12 SUPPORT_URL="https://help.ubuntu.com/"
13 BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
14 PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
15 UBUNTU_CODENAME=jammy
16
```

New Container Hostname

When creating a container using the docker run command, the `-h` or `--hostname` option can be used to define the hostname of the container.

```
1  root@ubuntu22:~# docker run -it --hostname www.cloudiq.online --name cloudiq alpine sh
2  / #
3  / # hostname
4  www.cloudiq.online
```

Another way

```
1  root@ubuntu22:~# docker run -itd --hostname www.cloudiq.online --name cloudiq alpine
2  9d444de57db0cbb0b2d814cd94c286c025232321eb579254a8378f79f6d7d358
3  root@ubuntu22:~# docker exec cloudiq hostname
4  www.cloudiq.online
```

Existing Container Hostname

Create a docker container

```
1   root@ubuntu22:~# docker run -itd ubuntu
2   7629d1030fb9fe8da2f694eef0705539f35738137695aec29f43c8c87294e957
```

The string `7629d1030fb9fe8da2f694eef0705539f35738137695aec29f43c8c87294e957` is the Container Full ID.

The first 12 charcters in string is the usual container ID we see in `docker ps -a` command.

Find the container

```
1   docker ps -a
```

Find the current hostname

```
1   docker exec 7629d1030fb9 hostname
2   7629d1030fb9
```

Stop the container

```
1   docker stop 7629d1030fb9
```

Stop the Docker host

```
1   service docker stop
```

Confirm

```
1   root@ubuntu22:~# systemctl is-active docker
2   inactive
```

Get the Full ID of the container

```
1   root@ubuntu22:~# docker inspect --format="{{.Id}}" wizardly_keller
2   68ed0fe2cc4333e0e63f9b1a90360c116941d1b2bd66eb8865ae82be30ef7660
```

Edit config file (JSON)

```
 1   cd /var/lib/docker/containers
 2   cd FULL CONTAINER_ID
 3   cd 68ed0fe2cc4333e0e63f9b1a90360c116941d1b2bd66eb8865ae82be30ef7660
 4   #Here modify 2 files: config.v2.json, hostname
 5   nano config.v2.json
 6   Replace
 7     "Hostname":"WHATEVER"
 8   nano hostname
 9   Replace
10     "Hostname":"WHATEVER"
```

Start the Docker host service and Container

```
1   service docker start
2   docker start 68ed0fe2cc43
```

Find the hostname

```
1   dockee exec 68ed0fe2cc43 hostname
2   juniperbox
```

Rename a container

```
root@ubuntu22:~# docker ps
CONTAINER ID    IMAGE     COMMAND       CREATED             STATUS              PORTS        NAMES
9d444de57db0    alpine    "/bin/sh"     About a minute ago  Up About a minute                cloudiq

root@ubuntu22:~# docker container rename cloudiq newcloudiq

root@ubuntu22:~# docker ps
CONTAINER ID    IMAGE     COMMAND       CREATED             STATUS              PORTS        NAMES
9d444de57db0    alpine    "/bin/sh"     About a minute ago  Up About a minute                newcloudiq
```