

# Table of contents

---

- [Table of contents](#)
  - [Docker Engine](#)
    - [Creating Docker Images](#)
      - [1. Using container](#)
      - [2.Using Dockerfile](#)
        - [Dockerfile Definitions](#)
    - [Docker Image Registry](#)
      - [Amazon ECR Terminology](#)
    - [Docker Hub](#)
      - [Official and Unofficial Docker Images](#)
  - [Docker ECR Push Shell Script](#)
    - [Set environment variables for docker containers](#)
  - [Docker-Essential Commands](#)
    - [CMD vs ENTRYPOINT](#)
    - [CMD](#)
    - [ENTRYPOINT](#)

---

## Docker Engine

- Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:
  - A server with a long-running daemon process dockerd.
  - APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
  - A command line interface (CLI) client docker.

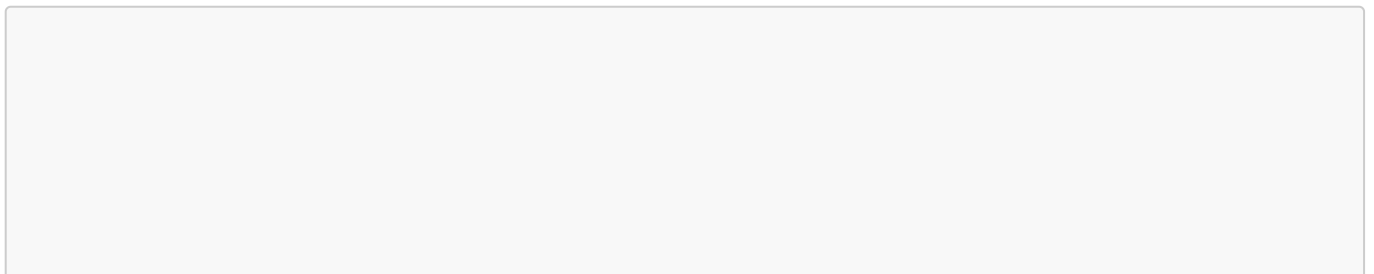
---

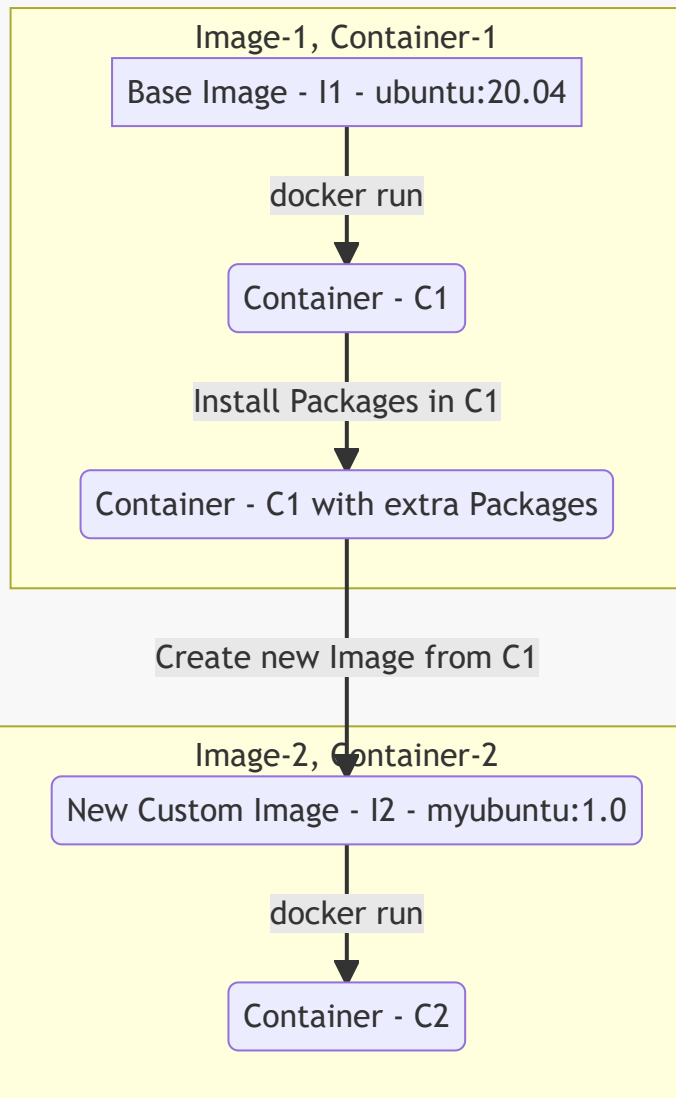
## Creating Docker Images

### 1. Using container

- **I1 - ubuntu 20.04 > C1 > install wget or any package in C1 > I2-Custom image > C2 from I2**
- Check existing images and run a container using image.
  - Use ubuntu image as base image , install wget inside the container , then create an image from running container.
  - This setup is not preferred in a deployment environment.

--





--

```
sudo docker images
sudo docker run --name=base-image-container -i -t ubuntu:20.04 /bin/bash
```

- Run below commands inside container

```
whereis wget
apt-get update && apt-get install -y wget
whereis wget
```

- From another linux shell, use below `docker diff` command to check changes made in the above container

```
sudo docker diff <CONTAINER_NAME>
```

--

- To create a new image from existing running container using **docker commit**

```
docker images
sudo docker commit base-image-container ubuntu/docker/ubuntu_wget
sudo docker images
sudo docker run -i -t ubuntu/docker/ubuntu_wget:latest /bin/bash
whereis wget
```

--

- Below is the information for the details related to **docker images** command
  - **REPOSITORY**: This is the name of the repository or image.
  - **TAG**: This is the tag associated with the image
  - **IMAGE ID**: Every image is associated with a unique ID.
  - **CREATED**: Indicates the time when the image was created.
  - **SIZE**: Highlights the virtual size of the image.

---

## 2.Using Dockerfile

- A **Dockerfile** is a script that consists of instructions to build Docker images which can then be used to deploy a Docker container.
- A Dockerfile is a text document that contains all the commands/set of instructions a user could call on the command line to create an image.
- The commands within the Dockerfile can be configured to use specific versions & dependencies.
- Once a **Dockerfile** is written, you can use the **docker build** command to generate a Docker image based on the Dockerfile's instructions.

--

- Create a Docker image using DockerFile **touch Dockerfile**
- Copy the below content into **Dockerfile**

```
# Base Image will be as below
FROM ubuntu:20.04
ARG SDLC_ARG
ENV SDLC_ENV=${SDLC_ARG}
RUN echo "ARG value for SDLC_ARG is $SDLC_ARG"
RUN echo "ENV value for SDLC_ENV is $SDLC_ENV"
# To Set a default value
# ARG SDLC_ENV=test
ENV DEBIAN_FRONTEND=noninteractive
# Install dependencies
RUN apt-get update
RUN apt-get install -y apache2
RUN apt-get install -y apache2-utils
```

```
# Replace content of Apache Home Page
RUN echo "Docker Image created using Dockerfile for $SDLC_ENV" >
/var/www/html/index.html
# Expose Container Port
EXPOSE 80
# Execute command at container launch
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

--

- The **DEBIAN\_FRONTEND=noninteractive** instruction ensures that the subsequent RUN apt-get commands execute without requiring additional user input when building images.
- A Dockerfile typically contains of **INSTRUCTION** and **ARGUMENT**. **INSTRUCTION** like **FROM**, **RUN** and **ARGUMENT** like **ubuntu:20.04**, **apt-get install -y apache2**.

--

## Dockerfile Definitions

- **Basic Definitions**

- **FROM**: Define the base image, such as ubuntu or debian, used to start the build process. **Required** for each Dockerfile.

- **Variables**

- **ENV**: Set environment variables that persist when the container is deployed.
- **ARG**: It is only available during the build of a Docker image (**RUN** etc), not after the image is created and containers are started from it. It is used to Pass a variable during Image build. Variable set as ARG does not persist when container is deployed from the image.

--

- **Command Execution**

- **RUN**: Execute commands, such as package installation commands that runs on a new intermediate container.
- **CMD**: Execute a specific command within the container that is deployed with the image. Only one is used per Dockerfile.
- **ENTRYPOINT**: Set a default application to be used every time a container is deployed with the image. Only one is used per Dockerfile.
- **WORKDIR**: Set the container path where subsequent Dockerfile commands are executed.

--

- **Data Management**

- **ADD**: Copy files from a source to the image's filesystem at the set destination along with remote URL handling
- **COPY**: Similar to ADD but without automatic tarball and remote URL handling.

--

- **Networking**

- **EXPOSE**: Expose a specific port to enable networking between the container and the Host.

--

- Use below command to build an image locally using Dockerfile.

```
docker build -t docker-apache2 . --build-arg SDLC_ARG="dev"
docker build -t docker-apache2 -f Dockerfile --build-arg SDLC_ENV="dev"
docker images
```

- `.` in the above command specifies the path of the `Dockerfile`
- This Dockerfile uses the `ubuntu:20.04` image.
- The `RUN` instructions will simply run the linux commands for that image and then write the some content to the web server's document root.

--

- Run a container with the newly built image and keep docker running in detached mode with `-d` parameter
- The `-p 80:80` option maps the exposed port 80 on the container to port 80 on the EC2 host system.
  - `-p <HOST_PORT>:<CONTAINER_PORT>`

--

```
netstat -nltp
docker run -d -p 80:80 -t docker-apache2
# to get into container bash session
docker exec -it container-id bash
docker ps
# Check for Config Env values set for this container using inspect command
docker inspect <CONTAINER_NAME>
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
86c6b6c2212e	docker-test	"apache2ctl -D FOREG..."	2 seconds ago	Up 1
second	0.0.0.0:80->80/tcp	relaxed_curie		

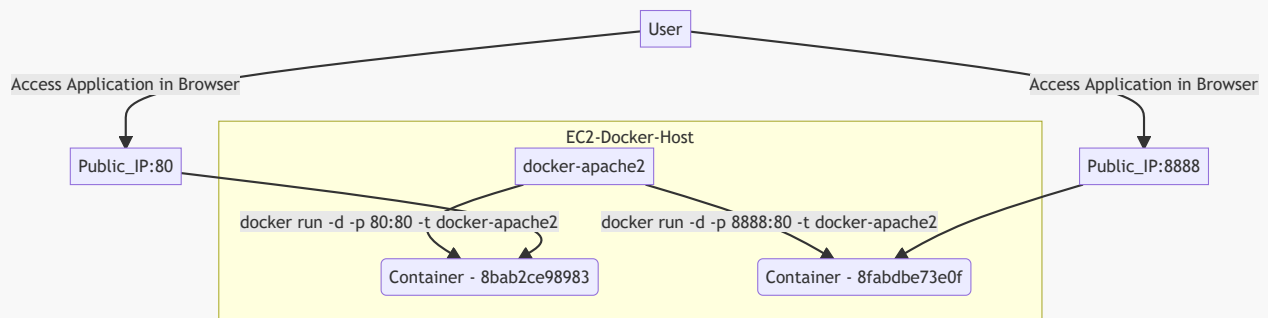
```
netstat -nltp
# Access the container application in browser
# Launch another container on another port with same image
docker run -d -p 8888:80 -t docker-apache2
docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
8bab2ce98983	docker-apache2	"apache2ctl -D FOREG..."	20 seconds ago
Up 19 seconds	0.0.0.0:8888->80/tcp	amazing_nash	
8fabdbe73e0f	docker-apache2	"apache2ctl -D FOREG..."	2 minutes ago
Up 2 minutes	0.0.0.0:80->80/tcp	loving_goodall	

```
docker run --name <container_name> -p <host_port>:<container_port> -d  
<container_image_label or ID>
```

- Test the content of the html file in the browser using public ip, make sure you have port 80 open in security group.

--



--

```
curl localhost:80  
curl localhost:8888
```

- Checking Docker Image Size

```
docker image ls  
docker image inspect <IMAGE_NAME>
```

- Docker images are stored at `/var/lib/docker/overlay2`

```
du -sh -m /var/lib/docker/overlay2
```

- We can use the `docker rmi` command to remove the images:

```
docker rmi <IMAGE_NAME>
du -sh -m /var/lib/docker/overlay2
```

---

## Docker Image Registry

### Amazon ECR Terminology

- **Registry:** An ECR registry is provided to each AWS account; we can create image repositories in our registry and store images in them.
- **Repository:** An ECR image repository contains our Docker images.
- **Authorization token:** The Docker client must authenticate to Amazon ECR registries as an **AWS** user before it can push and pull images. The AWS CLI **get-login** command provides us with authentication credentials to pass to Docker.
- **Image:** We can push and pull container images to our repositories.
- Attach a role to ec2 instance with ECR Permissions to create ECR Repository and push images to ECR Repository
- Create a ECR repository

```
# aws ecr-public describe-repositories --region ap-south-1
aws ecr describe-repositories --region ap-south-1
# aws ecr-public create-repository --repository-name docker-testing
aws ecr create-repository --repository-name docker-testing --region ap-south-1
```

- Use the following steps to authenticate and push an image to your repository.
  - Retrieve an authentication token and authenticate your Docker client to your registry.

```
# aws ecr-public get-login-password | docker login --username AWS --password-stdin
<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com
aws ecr get-login-password --region ap-south-1 | docker login --username AWS --
password-stdin <ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com
```

- After the docker build is completed, check for local docker images present and tag your image so you can push the image to this repository:

**docker tag** is what we use to define which repository an image will be pushed to, and **docker push** is the command that does the upload itself.

```
docker tag docker-apache2:latest <ACCOUNT_ID>.dkr.ecr.
<REGION_NAME>.amazonaws.com/docker-testing:latest
```

```
docker images
```

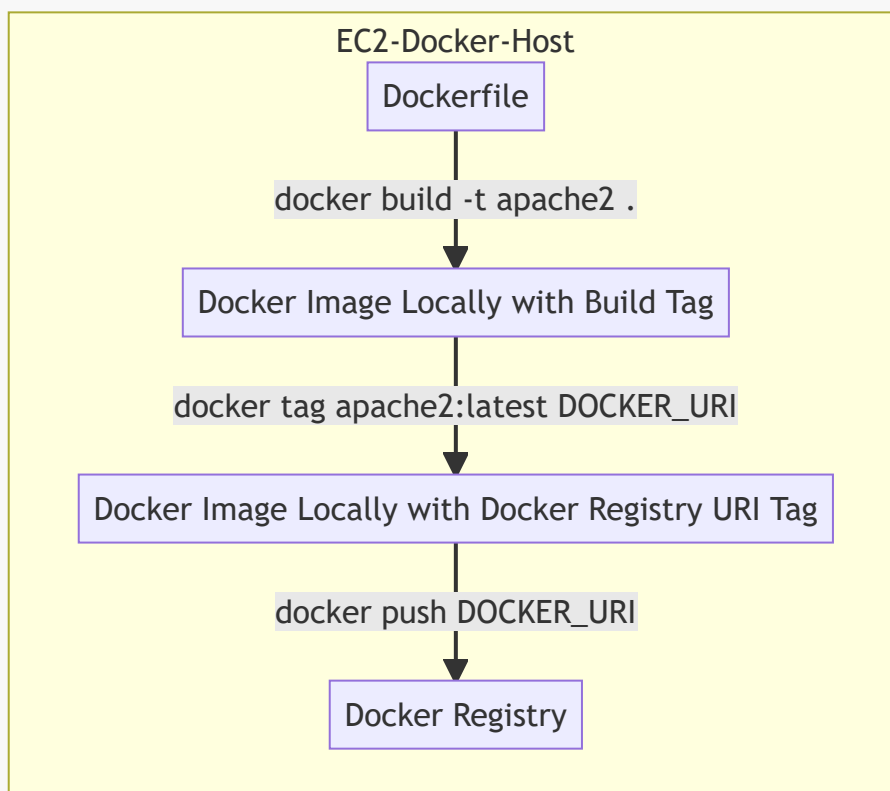
Note that the image id remains the same between the 2 versions of the image. This is ideally the same image, just with 2 references.

- Run the following command to push this image to your newly created AWS repository:

```
docker push 01234567890.dkr.ecr.ap-south-1.amazonaws.com/docker-testing:latest
```

- Check whether docker image is available in ECR using AWS Console OR CLI

```
# aws ecr-public describe-repositories --region ap-south-1
aws ecr describe-images --repository-name docker-testing --region us-east-1
```



- To Delete the ECR Repository use below command

```
# aws ecr-public delete-repository --repository-name docker-testing
aws ecr delete-repository --repository-name docker-testing --region us-east-1 --force
```



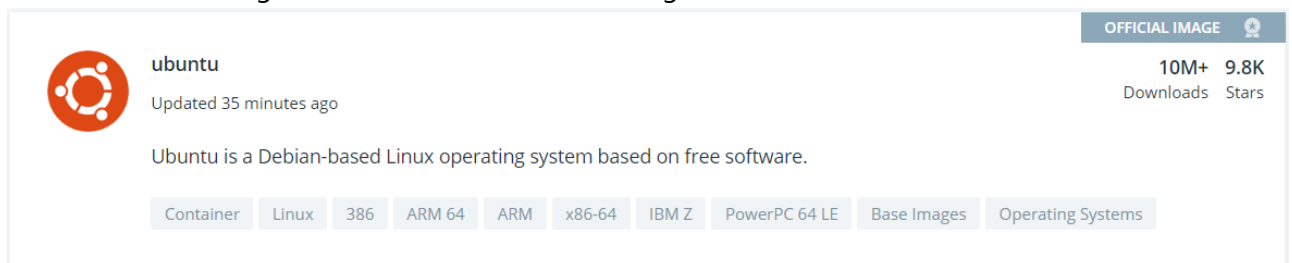
As part of the AWS Free Tier, new Amazon ECR customers get 500 MB-month of storage for one year for your private repositories. As a new or existing customer, Amazon ECR offers you 50 GB-month of always-free storage for your public repositories.

## Docker Hub

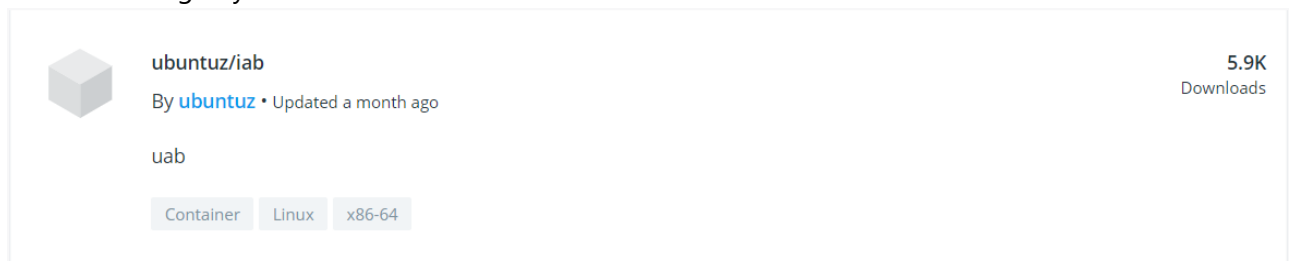
- **Docker Hub** is a service provided by Docker for hosting, finding, and sharing Docker Repositories. A Docker repository can be public or private.
- The Docker Hub and other third party repository hosting services are called **registries**.
- A **registry** has many **repositories**, while a **repository** has many **different versions** of the same image.

## Official and Unofficial Docker Images

- On **Docker Hub**, there are two kinds of images - **official** and **unofficial**.
- Official images are trusted and optimized.
  - They have clear documentation, promote best practices, and are designed for the most common use cases.
- An unofficial image is any image that is created by a user.
- Docker Hub follows some standards so that both can easily be identified. Official images contain only the `<image_name>` as its image name but unofficial images have the syntax as `/<image_name>`.
- Also, the official image has official written in the listing as shown in the below screenshot.



- Unofficial Image by User **ubuntuz**



- Sign Up to create a Docker Hub Account on <https://hub.docker.com/>
- Login to docker from CLI using **docker login** command.
  - Execute this command from a Shell or Terminal where docker is installed.

```
docker login --username <DOCKER_USERNAME>
```

#Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com> to create one.

# Username:

# Password:

# WARNING! Your password will be stored unencrypted in `/root/.docker/config.json`

- Once Docker Login is successful in Local Shell/Terminal, Tag the local image with a name matching to your profile name with command.

```
docker tag docker-apache2:latest <Dockerhub_Username>/ docker-apache2:latest
```

- Check the local docker images again to verify the tag.

```
docker images
```

- Push the docker image to docker hub using `docker push <username>/ image:tag`

```
docker push <Dockerhub_Username>/docker-apache2:latest
```

- Validate whether Image is available in Docker Hub Account from browser.

## Docker ECR Push Shell Script

- Below is the shell script that will create docker image using Dockerfile and push the Docker Image in ECR Repo.
- Create a shell script file `docker_ecr_push.sh` with code.

```
#!/bin/bash
```

```
# set -e
```

```
# This script shows how to build the Docker image and push it to ECR
```

```
# The argument to this script is the image name. This will be used as the image on the local machine and combined with the account and region to form the repository name for ECR.
```

```
SDL_C_ENVIRONMENT=$1
```

```
image=$2
```

```
region=$3
```

```
echo "value of image is $image"
```

```
if [ "$image" == "" ]
```

```
then
```

```
    echo "Usage: $0 <image-name> not specified"
```

```

        exit 1
    fi

    # Get the account number associated with the current IAM credentials
    account=$(aws sts get-caller-identity --query Account --output text)

    if [ $? -ne 0 ]
    then
        exit 255
    fi

    # region="ap-south-1"
    ecr_repo_name=$image"-ecr-repo"
    image_name=$SDLC_ENVIRONMENT-$image
    # If the repository doesn't exist in ECR, create it.
    echo "Checking ECR Repo with name $ecr_repo_name"
    # || means if the first command succeed the second will never be executed
    aws ecr describe-repositories --repository-names ${ecr_repo_name} --region $region
    || aws ecr create-repository --repository-name ${ecr_repo_name} --region $region

    # Get the login command from ECR and execute docker login
    aws ecr get-login-password --region $region | docker login --username AWS --
    password-stdin ${account}.dkr.ecr.${region}.amazonaws.com

    # Build the docker image locally with the image name and then push it to ECR with
    the full name.

    docker build -t ${image_name} .
    fullname="${account}.dkr.ecr.${region}.amazonaws.com/${ecr_repo_name}:${image_name}"
    echo "fullname is $fullname"

    # Tag the locally created docker image with the ECR Repo URI
    docker tag ${image_name} ${fullname}
    # docker images
    docker push ${fullname}

    if [ $? -eq 0 ]
    then
        echo "Docker Push Event is successfull with ${fullname}"
    else
        echo "Docker Push Event failed."
    fi

```

- As this shell script accepts below Command Line Arguments as:
  - SDLC\_ENVIRONMENT
  - image
  - region
- Use below command to run the shell script:

```
bash docker_ecr_push.sh dev testimage ap-south-1
```

## Set environment variables for docker containers

- Use the `-e`, `--env`, and `--env-file` flags to set environment variables in the container you're running, or overwrite variables that are defined in the Dockerfile of the image you're running.
- You can define the variable and its value when running the container:

```
# create env.list file locally
docker run -e MYVAR1 --env MYVAR2=foo --env-file ./env.list ubuntu bash
```

- You can define the variable and its value when running the container:

```
docker run --env VAR1=value1 --env VAR2=value2 ubuntu env | grep VAR
```

- You can also use variables that you've exported to your local environment:

```
export VAR1=value1
export VAR2=value2
docker run --env VAR1 --env VAR2 ubuntu env | grep VAR
```

- When running the command, the Docker CLI client checks the value the variable has in your local environment and passes it to the container. If no `=` is provided and that variable is not exported in your local environment, the variable won't be set in the container.
- You can also load the environment variables from a file. This file should use the syntax `=value` (which sets the variable to the given value) or `=` (which takes the value from the local environment), and `#` for comments.

```
cat env.list
VARIABLE1=val1
VARIABLE2=val2
VARIABLE3=val3
USER

docker run --env-file env.list ubuntu env | grep -E 'VARIABLE|USER'
```

## Docker-Essential Commands

- Below are the list of **essential commands** for docker

Commands	Description
<code>docker version</code>	Show the Docker version information

Commands	Description
<code>docker ps</code>	List all running containers
<code>docker ps -a</code>	List all containers <b>stopped</b> , <b>running</b>
<code>docker stop CONTAINER_ID</code>	Stop the container which is running
<code>docker start CONTAINER_ID</code>	Start the container which is stopped
<code>docker restart CONTAINER_ID</code>	Restart the container which is running
<code>docker port CONTAINER_ID</code>	List port mappings of a specific container
<code>docker rm CONTAINER_ID or name</code>	Remove the stopped container
<code>docker rm -f CONTAINER_ID or name</code>	Remove the running container forcefully
<code>docker pull IMAGE_NAME:TAG</code>	Pull the image from docker hub repository
<code>docker exec -it container-name /bin/bash</code>	Connect to linux container and execute commands in container
<code>docker rmi image-id</code>	Remove the docker image
<code>docker login -u username -p password</code>	Login to docker hub
<code>docker logout</code>	Logout from docker hub
<code>docker stats</code>	Display a live stream of container(s) resource usage statistics
<code>docker info</code>	Display system-wide information

## CMD vs ENTRYPOINT

- **CMD** and **ENTRYPOINT** are **Dockerfile** instructions.
- Primary usage of **CMD** and **ENTRYPOINT** is used to run the executable when instantiating the image.
- Below are the key differences between the same.
  - **CMD** can pass default parameters to **ENTRYPOINT** if both are defined.

## CMD

- The **CMD** directive allows to specify the default command executed by the container.
- This command runs when the container starts and no other command is specified for docker run.
- If docker run specifies another command, the default command specified by **CMD** will be ignored.
- If a Dockerfile has multiple **CMD** instructions, it only applies the instructions from the last one.

**CMD** has three formats:

- Exec format: **CMD** ["executable","param1","param2"]
- **CMD** ["param1", "param2"], this format is used in combination of **ENTRYPOINT** , to provide extra parameters
- Shell format: **CMD** command param1 param2

## ENTRYPOINT

- The **ENTRYPOINT** directive allows the container to run as an application or service.
- **ENTRYPOINT** looks similar to **CMD** in that both specify the command to execute and its parameters
- You **cannot override** an **ENTRYPOINT** when starting a container unless you add the **--entrypoint** flag.

**ENTRYPOINT** has two formats:

- Exec format: **ENTRYPOINT** ["executable", "param1", "param2"] This is the recommended format for **ENTRYPOINT**.
- Shell format: **ENTRYPOINT** command param1 param2
- The parameters in **ENTRYPOINT** are always used, while the extra parameters of **CMD** can be dynamically replaced when the container starts.

```
ENTRYPOINT ["/bin/echo", "Hello"]  
CMD ["world"]  
# Output  
Hello world
```

- Note the shell format of **ENTRYPOINT** ignores any arguments provided by **CMD** or **docker run**.

```
FROM busybox  
ENTRYPOINT echo hello  
CMD world  
# Output  
hello
```