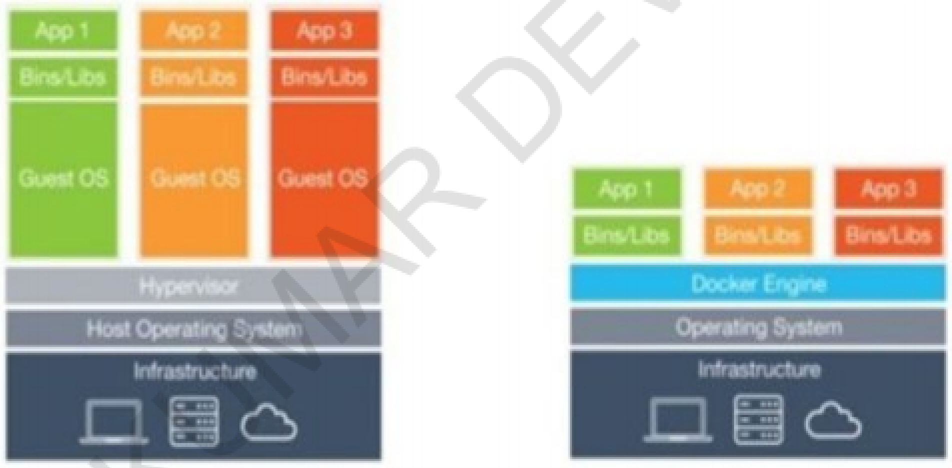
# Docker

## Container:

- Shipping an application bundled with all dependencies or libraries in an image. without OS is a container
- Containers are OS virtualisation
- Do not need full OS in container to install our applications
- Applications inside are dependent on container, further container are dependent on host OS kernel
- Containers are like any other processes in OS but these but differ in way they operate
- They operate in isolated environment
- Containers have their own process tree, network also
- Every container will have its own IP address and port on which the application inside container is running
- Generally, VM needs a Full OS and containers does not



- Containers are lightweight as it just only contains libraries and applications
- Therefore, there is an less usage of compute power and resources
- In simple words, container is self sufficient image without actual OS that helps to run an application or serve the purpose

## Docker history:

- Hykes started the Docker project in France as an internal project within dotCloud, a platform-as-a-service company.
- Docker Inc. was founded by Solomon Hykes and Sebastien Pahl during the Y Combinator Summer 2010 startup incubator group and launched in 2011.
- Docker debuted to the public in Santa Clara at PyCon in 2013.
- It was released as open-source in March 2013. At the time, it used LXC as its default execution environment.

## Docker usage:

- The main purpose of Docker is to Deploy applications to production environments, or other environments as required by Ops.
- Docker, a container management tool, is used in DevOps to manage software parts as isolated, self-sufficient containers, which can be deployed and run in any environment.
- Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.
- Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package.
- However, the highlight of Docker lies in the process of deploying it. Till Docker came into picture, the traditional approach to deployment was via Virtual Machines (VM).

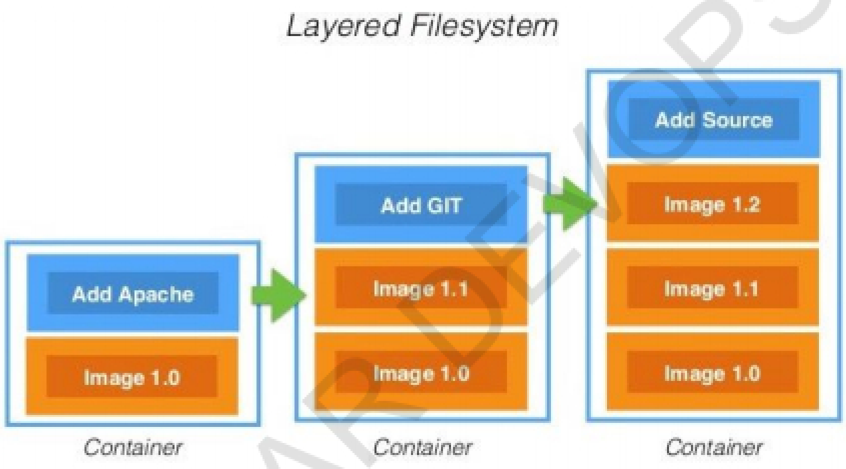
## OS-Level-Virtualization:

- Docker is a set of platform as a service (PaaS) products that uses OS-level virtualization to deliver software in packages called containers.
- Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.
- A type of server virtualization technology which works at the OS layer.
- The OS kernel will run a single operating system and provide that operating system functionality to each of the partitions.
- The physical server and single instance of the operating system is virtualized into multiple isolated partitions, where each partition replicates a real server.

- OS-level virtualization refers to an operating system paradigm in which the kernel allows the existence of multiple isolated user space instances.

## Layered file system:

- The layered filesystem will reuse the layers from parent images
- The image may consist of multiple layers, each of which are created once and can be reused by other images, everything is stored as references to a hash, and only when there are no more references to the hash will docker remove that layer on a docker rmi.
- A Docker image consists of several layers. Each layer corresponds to certain instructions in your Dockerfile.

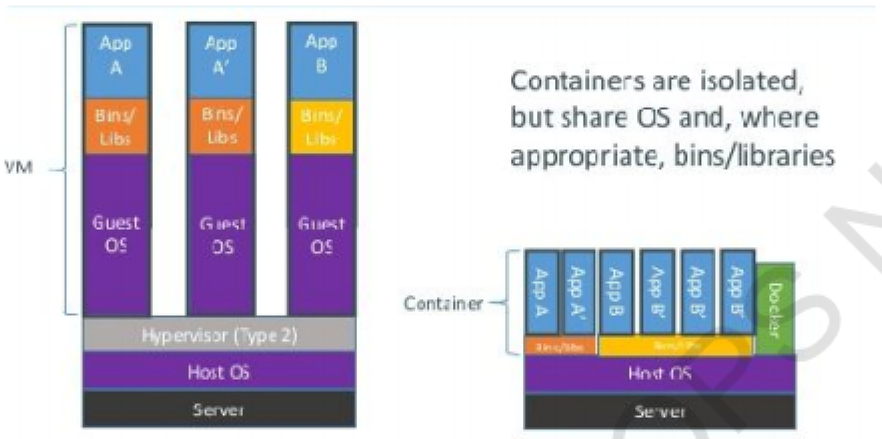


Docker container is created from a read only template called docker image.

## VMWare vs Docker:

| VMWare | Docker |
|---|---|
| A Virtual Machine, on the other hand, is not based on container technology. | Docker is container based technology and containers are just user space of the operating system. |
| VM is made up of user space plus kernel space of an operating system. | In Docker, the containers running share the host OS kernel. |
| VMWare is used for creating virtual servers. | Docker can run on virtual servers created using VMWare. |
| What VMWare does is, it abstracts the hardware and allows you to do things that you can do with raw hardware | What docker does is it abstracts the environment required by your app and only provides that much |

| | |
|---|---|
| (eg: install OS). | resources, nothing more. |
| Working in heterogeneous environment, Virtual Machines provide high flexibility | whereas Docker containers' prime focus is on applications and their dependencies. |



## Docker components, Docker workflow:

Docker components:
There are four components:

- Docker client and server
- Docker image
- Docker registry
- Docker container

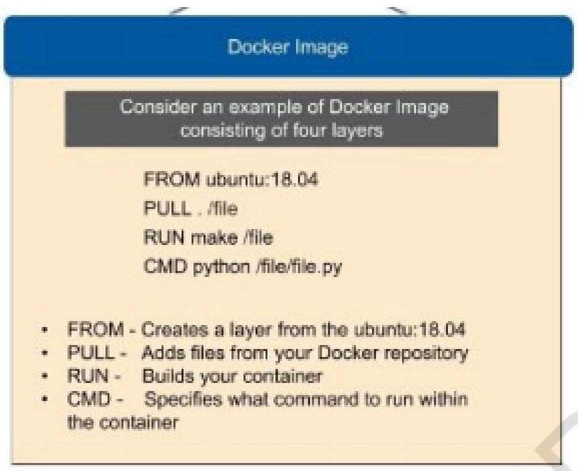
Docker Client and Server:

- This is a command-line-instructed solution in which you would use the terminal on your Mac or Linux system to issue commands from the Docker client to the Docker daemon.
- The communication between the Docker client and the Docker host is via a REST API.
- You can issue similar commands, such as a Docker Pull command, which would send an instruction to the daemon and perform the operation by interacting with other components (image, container, registry).

Docker Image

- The Docker image is built within the YAML file and then hosted as a file in the Docker registry.

- The image has several key layers, and each layer depends on the layer below it.
- Image layers are created by executing each command in the Dockerfile and are in the read-only format.



Docker Image

Consider an example of Docker Image consisting of four layers

```
FROM ubuntu:18.04
PULL . /file
RUN make /file
CMD python /file/file.py
```

- FROM - Creates a layer from the ubuntu:18.04
- PULL - Adds files from your Docker repository
- RUN - Builds your container
- CMD - Specifies what command to run within the container

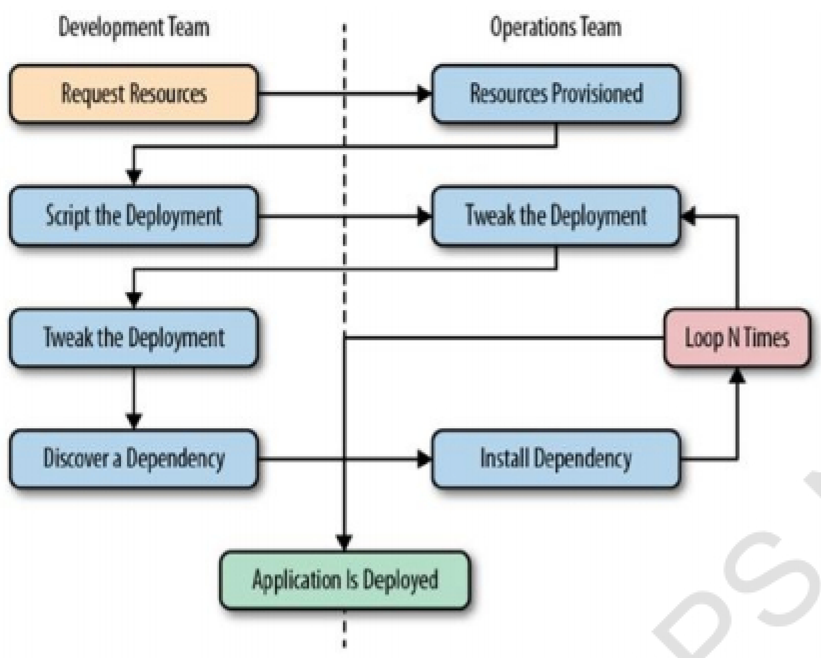
Docker Registry:

- The Docker registry is where you would host various types of images and where you would distribute the images from.
- The repository itself is just a collection of Docker images, which are built on instructions written in YAML and are very easily stored and shared.
- You can also create your own registry for your own use internally. The registry that you create internally can have both public and private images that you create.

Docker Container:

- The container is also inherently portable. Another benefit is that it runs completely in isolation.
- In simple words, a container is a runnable instance of an image.
- You can create, start, stop, move, or delete a container using the Docker API or CLI.
- When a container is removed, any changes to its state that are not stored in persistent storage disappear.

<u>Docker workflow:</u>



1. Application developers request resources from operations engineers.
2. Resources are provisioned and handed over to developers.
3. Developers script and tool their deployment.
4. Operations engineers and developers tweak the deployment repeatedly.
5. Additional application dependencies are discovered by developers.
6. Operations engineers work to install the additional requirements.
7. Loop over steps 5 and 6 *N* more times.
8. The application is deployed.

## Docker benefits, Docker images:

Docker benefits:

- Return on Investment and Cost Savings.
- Standardization and Productivity.
- CI Efficiency.
- Compatibility and Maintainability.
- Simplicity and Faster Configurations.
- Rapid Deployment.
- Continuous Deployment and Testing.
- Multi-Cloud Platforms.

Docker images:

- A Docker image is a file, composed of multiple layers, that is used to execute code in a Docker container.

- An image is essentially built from the instructions for a complete and executable version of an application, which relies on the host OS kernel.
- When the Docker user runs an image, it becomes one or multiple instances of that container.
- To see the list of Docker images on the system
  ```
  #docker images
  ```
- To run a command in a Docker container.
  ```
  #docker run image
  ```
- To remove Docker images.
  ```
  #docker rmi ImageID
  ```
- To pull image from docker registry
  ```
  #docker pull imageID
  ```
- How to run docker image
  ```
  #docker run -it <image id> <program to start>
  ```
- To search particular image
  ```
  #docker search <image id>
  ```
- How to see docker history
  ```
  #docker image history<image id>
  ```
- To see Docker image metadata
  ```
  #docker image inspect <image id>
  ```
- How to tag docker images?
  ```
  # docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGR:TAG]
  ```
- How to build an image from Dockerfile?
  ```
  # docker image build -t &lt;image_name:version&gt; &lt;path&gt;
  ```
- where are downloaded images located on docker host?
  ```
  /var/lib/docker
  ```

## <u>Docker file:</u>

- Docker can build images automatically by reading the instructions from a Dockerfile.
- The Docker daemon runs the instructions in the Dockerfile one-by-one, committing the result of each instruction to a new image
- A Docker image consists of read-only layers each of which represents a Dockerfile instruction.

Example:

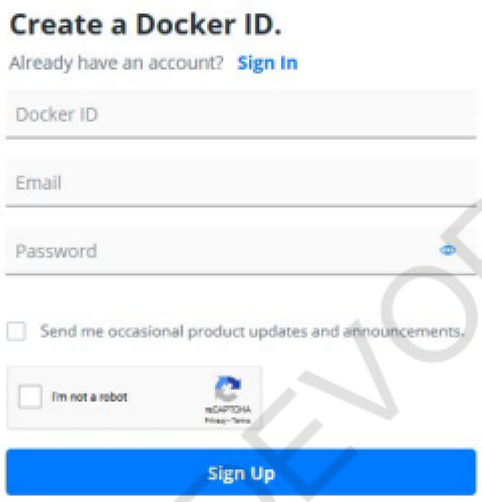
```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Each instruction creates one layer:

- FROM creates a layer from the ubuntu:18.04 Docker image.
- COPY adds files from your Docker client's current directory.
- RUN builds your application with make.
- CMD specifies what command to run within the container.

## Docker hub/registry:

1. Go to the Docker Hub sign up page.



2. Enter a username that will become your Docker ID.
   Your Docker ID must be between 4 and 30 characters long, and can only contain numbers and lowercase letters.
3. Enter a unique, valid email address.
4. Enter a password between 6 and 128 characters long.
5. Click Sign Up.
   Docker sends a verification email to the address you provided.
6. Click the link in the email to verify your address.

Note: You cannot log in with your Docker ID until you verify your email address.

- Login to a registry on your localhost

```
# docker login localhost:8080
```

- To log out from a Docker registry.

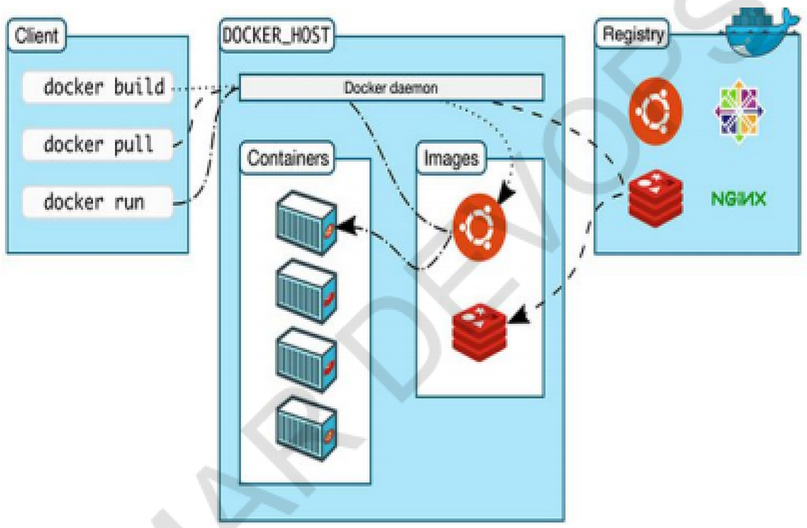
```
#docker-logout
```

- To see all running containers

```
#docker ps
```

## Docker daemon:

- The Docker daemon is a service that runs on your host operating system.
- It currently only runs on Linux because it depends on a number of Linux kernel features, but there are a few ways to run Docker on MacOS and Windows too.
- The Docker daemon itself exposes a REST API. From here, a number of different tools can talk to the daemon through this API.
- The most widespread tool is the Docker CLI. It is a command line tool that lets you talk to the Docker daemon.
- When you install Docker, you get both the Docker daemon and the Docker CLI tools together.



## Docker Install & Configure:

- Before install any docker versions, make sure that older versions of docker are removed

Reference link –

https://docs.docker.com/install/linux/docker-ce/centos/#uninstall-old-versions

- To verify if docker already exists on RHEL

  # rpm -qa I grep docker

- If using RHEL 7 version then use below command to install docker

  # yum install docker

- Once installed is completed, docker group is create
- By default, docker service is not started use below command to start the service

  # sudo systemctl start docker

  # sudo systemctl status docker

  # sudo systemctl enable docker

Note : please visit https://store.docker.com for installation of docker on specific

operating system distributions

- Add user to docker group to execute full functionality of docker commands

Q. How to verify docker installation?

  # docker –version

Run docker --version and ensure that you have a supported version of Docker:

docker --version

Docker version 17.12.0-ce, build c97c6d6

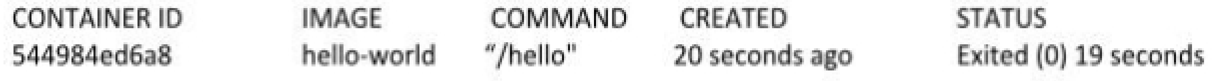
## Docker all commands:

- To check your current kernel version, open a terminal and type uname -r

  command to display your kernel version:

  # uname -r

- docker container ls --all



```
CONTAINER ID     IMAGE         COMMAND    CREATED           STATUS
544984ed6a8      hello-world   “/hello"   20 seconds ago    Exited (0) 19 seconds
```

- How to check the history of all containers?

  # docker ps -a

It will how both that are running and exited

- How to delete all docker containers?

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

  # docker rm -f $(docker ps -a -q) to remove forcefully.

- How to delete all docker images?

  # docker rmi $(docker images -q)

- How to see all running containers on docker host?
  Use docker ps command
  # docker ps

## Docker Volumes:

- In order to be able to save (persist) data and also to share data between containers, Docker came up with the concept of volumes.
- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
- New volumes can have their content pre-populated by a container.
- Create and manage volume

```
$ docker volume create my-vol
```

- List volumes:

```
$ docker volume ls
```

- Remove a volume:

```
$ docker volume rm my-vol
```

## Volume (container-container):

- In order to be able to save (persist) data and also to share data between containers, Docker came up with the concept of volumes.

- A data volume is a specially-designated directory within one or more containers that bypasses the Union File System.

Data volumes provide several useful features for persistent or shared data:

- Volumes are initialized when a container is created. If the container's base image contains data at the specified mount point, that existing data is copied into the new volume upon volume initialization. (Note that this does not apply when mounting a host directory.)
- Data volumes can be shared and reused among containers.
- Changes to a data volume are made directly.
- Changes to a data volume will not be included when you update an image.
- Data volumes persist even if the container itself is deleted.

volumes are designed to persist data, independent of the container's lifecycle. Docker therefore never automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container.

Example for volume creation:

1. Initialize (and mount) at run-time with the *-v* flag:

```
$ docker run -P --name web -v /webapp training/webapp python app.py
```

- This will create a new volume inside a container at /webapp. Anything written to the /webapp directory will be persisted to the host machine, available to the next container that mounts it.

2. where is the actual volume stored

```
$ docker inspect -f {{.Mounts}} web
```

3. Using the VOLUME instruction inside a Dockerfile:

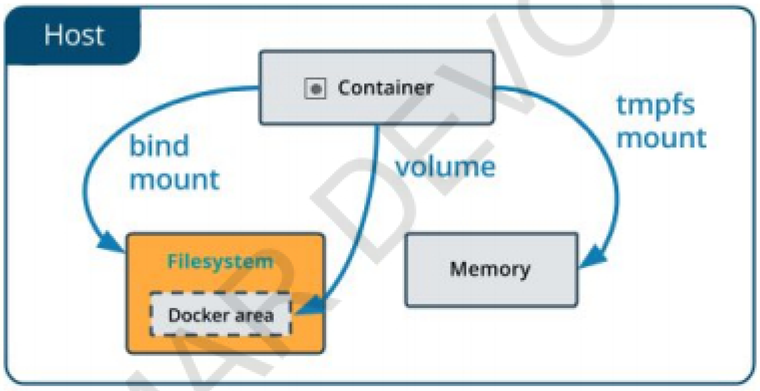
```
FROM ubuntu:latest

VOLUME /webapp
```

4. Create using the Docker Volume API introduced in Docker 1.9.

```
$ docker volume create --name webapp
```

## Volume (Host- Container):

- When you use a bind mount, a file or directory on the host machine is mounted into a container.
- The file or directory is referenced by its full or relative path on the host machine. By contrast, when you use a volume, a new directory is created within Docker's storage directory on the host machine, and Docker manages that directory's contents.
- The file or directory does not need to exist on the Docker host already. It is created on demand if it does not yet exist.
- Bind mounts are very performant, but they rely on the host machine's filesystem having a specific directory structure available.
- If you are developing new Docker applications, consider using named volumes instead. You can't use Docker CLI commands to directly manage bind mounts.



## Port mapping:

- In Docker, the containers themselves can have applications running on ports.
- When you run a container, if you want to access the application in the container via a port number, you need to map the port number of the container to the port number of the Docker host.
- How to attach a port of docker host to docker container?
  # docker container run --publish container_port:host_port &lt;imatge_id&gt;

Example : # docker container run --publish 80:80 nginx

- Downloaded image &quot;nginx &quot; from Docker Hub
- Started a new container from that image
- Opened port 80 on the host IP
- Routes that traffic to the container IP, port 80

Note - We will get bind error, if host port is already being used by another Application

## Registry server:

Before you can deploy a registry, you need to install Docker on the host. A registry is an instance of the registry image, and runs within Docker.

- Use a command to start the registry container:

```
$ docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

- Push the image to the local registry running at localhost:5000:

```
$ docker push localhost:5000/my-ubuntu
```

- Remove the locally-cached ubuntu:16.04 and localhost:5000/my-ubuntu images, so that you can test pulling the image from your registry. This does not remove the localhost:5000/my-ubuntu image from your registry.

```
$ docker image remove ubuntu:16.04

$ docker image remove localhost:5000/my-ubuntu
```

- Pull the localhost:5000/my-ubuntu image from your local registry.

```
$ docker pull localhost:5000/my-ubuntu
```

- To stop the registry, use the same docker container stop command as with any other container.

```
$ docker container stop registry
```

- To remove the container, use docker container rm.

```
$ docker container stop registry && docker container rm -v registry
```

## Pull/push images from /to registry:

1.Determine the registry name:

- Choose a hostname, which specifies location where you will store the image
- Choose an image name, which can be different from the image's name on your local machine.

- Combine the hostname, your Google Cloud Console project ID, and image name:
  [HOSTNAME]/[PROJECT-ID]/[IMAGE]

2.Tag the local image with the registry name by using the command:

```
docker tag [SOURCE_IMAGE] [HOSTNAME]/[PROJECT-ID]/[IMAGE]
```

where [SOURCE_IMAGE] is the local image name or image ID.

- Push the tagged image to Container Registry by using the command:

```
docker push [HOSTNAME]/[PROJECT-ID]/[IMAGE]
```

Pulling images from a registry

- To pull from Container Registry, use the command:

```
docker pull [HOSTNAME]/[PROJECT-ID]/[IMAGE]:[TAG]
```

Or

```
docker pull [HOSTNAME]/[PROJECT-ID]/[IMAGE]@[IMAGE_DIGEST]
```

To get the pull command for a specific image:

1. Click on the name of an image to go to the specific registry.
2. In the registry, check the box next to the version of the image that you want to pull.
3. Click SHOW PULL COMMAND on the top of the page.
4. Copy the pull command, which identifies the image using either the tag or the digest.

## CMD, RUN, ENTRYPOINT:

CMD:

The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable.

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.

The CMD instruction has three forms:

CMD ["executable &quot;,&quot;param 1&quot;,&quot;param 2&quot;] (exec form, this is the preferred

form)

CMD ["param1" &quot;,&quot;param 2&quot;) (as default parameters to ENTRYPOINT)

CMD command param1 param2 (shell form)

Don&#39;t 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.

RUN:

This instruction executes on top of the existing layer and creates a new layer.

It executes commands and commit the changes

RUN has two forms –

RUN &lt;command &gt; (shell form, the command is run in a shell, which by

default is /bin/sh -c on Linux or cmd /S /C on Windows)

RUN [&quot;executable &quot;, &quot;param 1&quot;, &quot;param 2&quot;] (exec form)

RUN /bin/bash -c &#39;source $HOME/.bashrc; \

echo $HOME'

Together they are equivalent to this single line:

RUN /bin/bash -c &#39;source $HOME/.bashrc; echo $HOME&#39;

Note: To use a different shell. other than '/bin/sh'. use the exec form passing in

the desired shell. For example. Run [&quot;/bin/bash &quot;, "-c" &quot;echo hello"]

ENTRYPOINT:

This instruction helps to set the primary command for the image

COPY ./docker-entrypoint.sh /

ENTRYPOINT ["/docker-entrypoInt.sh"]

This script allows the user to interact with Postgres in several ways.

It can simply start Postgres:

$ docker run postgres