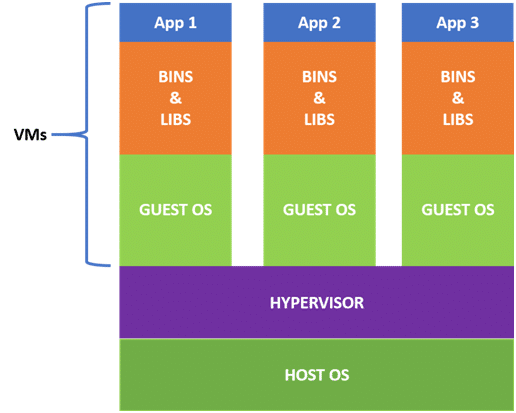
**Docker**

### **What is Virtualization?**

* Allowing developers to run multiple OS on different VMs while all of them run on the same host, thereby eliminating the need to provide extra hardware resources.



From the above VM architecture, it is easy to figure out that the three guest operating systems acting as virtual machines are running on a host operating system.

Virtualization lets us run our applications on fewer physical servers. In virtualization, each application and operating system live in a separate software container called VM. Where VMs are completely isolated, all the computing resources like CPUs, storage, and networking are pooled together, and they are delivered dynamically to each VM by a software called a hypervisor.

**Impo:**   
1) However, running multiple VMs over the same host leads to degradation in performance. As guest operating systems have their own kernel, libraries, and many dependencies running on a single host OS, it takes up large occupation of resources such as the processor, hard disk and, especially, its RAM.

1. Also, when we use VMs in virtualization, the bootup process takes a long time that would affect efficiency in the case of real-time applications. In order to overcome such limitations, containerization was introduced.

2) **What is Containerization?**

**Containerization** is a technique where the virtualization is being brought containerization to an operating system level. In containerization, we virtualize the operating system resources. It is more efficient as there is no guest operating system consuming the host resources; instead, containers utilize only the host operating system and share relevant libraries and resources only when they are required. The required binaries and libraries of containers run on the host kernel leading to faster processing and execution.

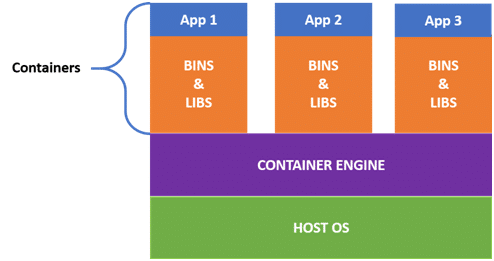
In a nutshell, containerization (containers) is a lightweight virtualization technology acting as an alternative to hypervisor virtualization. Bundle any application in a container and run it without thinking of dependencies, libraries, and binaries!

**Now, if we look into its advantages:**

* Containers are small and lightweight as they share the same OS kernel.
* They do not take much time to boot-up (only seconds).
* They exhibit high performance with lower resource utilization.

**Note:** In the case of containerization, all containers share the same host operating system. Multiple containers get created for every type of application making them faster but without wasting the resources, unlike virtualization where a kernel is required for every OS and lots of resources from the host OS are utilized.

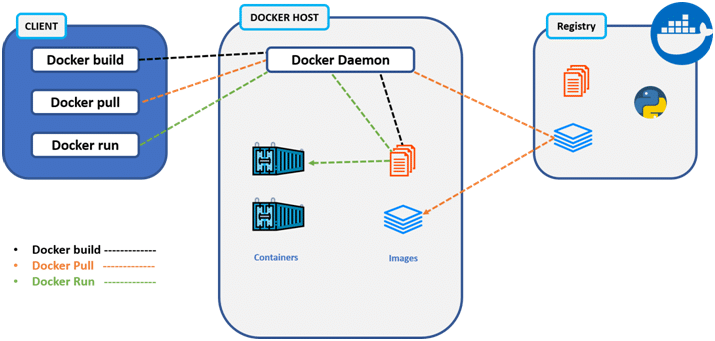
Note: <https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/>



In order to create and run containers on our host OS, we require software that enables us to do so. This is where **Docker** comes into the picture!

## **Docker Architecture**

Docker uses a client-server architecture. The Docker client consists of Docker build, Docker pull, and Docker run. The client approaches the Docker daemon that further helps in building, running, and distributing Docker containers. Docker client and Docker daemon can be operated on the same system; otherwise, we can connect the Docker client to the remote Docker daemon. Both communicate with each other using the REST API, over UNIX sockets or a network.



The basic architecture in Docker consists of three parts:

* Docker Client
* Docker Host
* Docker Registry

### **Docker Client**

* It is the primary way for many Docker users to interact with Docker.
* It uses command-line utility or other tools that use Docker API to communicate with the Docker daemon.
* A Docker client can communicate with more than one Docker daemon.

### **Docker Host**

In Docker host, we have Docker daemon and Docker objects such as containers and images. First, let’s understand the objects on the Docker host, then we will proceed toward the functioning of the Docker daemon.

* Docker Objects:
  + What is a Docker image? A Docker image is a type of recipe/template that can be used for creating Docker containers. It includes steps for creating the necessary software.
  + What is a Docker container? A type of virtual machine created from the instructions found within the Docker image. It is a running instance of a Docker image that consists of the entire package required to run an application.
* Docker Daemon:
  + Docker daemon helps in listening requests for the Docker API and in managing Docker objects such as images, containers, volumes, etc. Daemon issues to build an image based on a user’s input and then saves it in the registry.
  + In case we don’t want to create an image, then we can simply pull an image from the Docker hub (which might be built by some other user). In case we want to create a running instance of our Docker image, then we need to issue a run command that would create a Docker container.
  + A Docker daemon can communicate with other daemons to manage Docker services.

### **Docker Registry**

* Docker registry is a repository for Docker images which is used for creating Docker containers.
* We can use a local/private registry or the Docker hub, which is the most popular social example of a Docker repository.

Now that we are thorough with the Docker architecture and understand how Docker works, further in this Docker tutorial, let’s get started with the installation and workflow of Docker and implement important Docker commands.

### **Running a Container**

After the installation of Docker, we should be able to run the containers. If we don’t have a container to run, then Docker will download the image in order to build the container from the Docker hub and then will build and run it.

We can run a simple ‘hello-world’ container to crosscheck if everything is working properly. For that, run the following command:

docker run hello-world

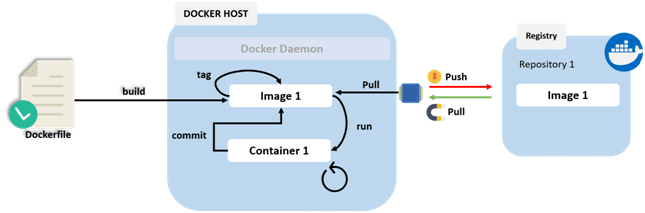
Output:

Hello from Docker!

## **Typical Local Workflow**

Docker’s typical local workflow allows users to create images, pull images, publish images, and run containers.

Let’s understand this typical local workflow from the diagram below:



Dockerfile here consists of the configuration and the name of the image pulled from a Docker registry, like a Docker hub. This file basically helps in building an image from it which includes the instructions about container configuration or it can be image pulling from a Docker registry like Docker hub.

Let’s understand this process in a little detailed way:

* It basically involves building an image from a Dockerfile that consists of instructions about container configuration or image pulling from a Docker registry, like Docker hub.
* When this image is built in our Docker environment, we should be able to run the image which further creates a container.
* In our container, we can do any operations such as:
  + Stopping the container
  + Starting the container
  + Restarting the container
* These runnable containers can be started, stopped, or restarted just like how we operate a virtual machine or a computer.
* Whatever manual changes are made such as configurations or software installations, these changes in a container can be committed to making a new image, which can further be used for creating a container from it later.
* At last, when we want to share our image with our team or to the world, we can easily push our image into a Docker registry.
* One can easily pull this image from the Docker registry using the pull command.

=> Pulling images, Running Images, Stopping and Starting Containers, Pushing an Image into the Repository, Listing, and deleting Containers and all check at:

<https://intellipaat.com/blog/tutorial/devops-tutorial/docker-tutorial/#_Introduction_to_Docker>

***Impo. Note:***

* *Contents to create DockerFile:*

*FROM python:3* : base image file (can use scratch as well)

*COPY . /usr/app/* : Copy all files from current user local dir . to the docker local user dir

*EXPOSE 5000* : Port to expose to the Docker

*WORKDIR /usr/app/*  : Define the working dir

*RUN pip install -r requirements.txt* : run and install all the libraries required

*CMD python app.py* : Run the mail app file

* Just go to the directory having the dockerfile , app file and other files and run below command, it will execute all the above commands one after the others and will create an images, later we can run that image to get the container:

***Docker build .***

* *We can push this image created by us to the docker hub so that anybody can use it.*
* *Dockerfile - > Docker image -> Docker Container*

**----------------------------------------------------------------------------------------------------**

**Some of Docker’s Commands:**

Note) docker tag 839fe37a4d2b tweeter\_analyzer:latest

* Add a tag/repository to an empty onces.

**Push Docker image to Docker hub:**

**-------------------------------------------------------------------------------------------------------**

1. Docker build . - build a docker image from docker file.
2. docker tag 839fe37a4d2b tweeter\_analyzer:latest - Add a tag/repository to an empty onces.
3. docker run -it -d tweeter\_analyzer - run above image (can pass image id as well) to get the container.
4. docker commit 6fcc10340848 shubham017/tweet\_analyzer:tweepy : commit the above container.
5. Docker login : login to Docker hub with your credentials.
6. docker push shubham017/tweet\_analyzer:tweepy : push the image to the docker Hub.

**-------------------------------------------------------------------------------------------------------**

**References:**

1) docker run hello-world : To test if docker is working fine or not.

2) docker images : to check what all images we have pulled.

3) docker pull ubuntu : Example of pulling ubuntu image from Docker hub registry.

4) docker run -it ubuntu bash : running ubuntu image we pulled.

5) docker ps : To see what all containers are running currently

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED

STATUS PORTS NAMES

6) $ mkdir Dockercontent : Create new dir in our path.

8) $ vim dockerfile : to edit and insert the content into the file we created above.

9) $ cat Dockerfile : to check the contents of the file we have inserted above.

out:

FROM ubuntu

RUN apt-get update

CMD ["echo","Docker Container is running..."]

Example:

FROM python:3

COPY . /usr/app/

EXPOSE 5000

WORKDIR /usr/app/

RUN pip install -r requirements.txt

CMD python app.py

10) $ docker build -t dockerimg:1.0.0 . : to run the image we have created above to get the container data.

Or we can pass the image id as well instead of dockerimg:1.0.0 above.

11) $ docker run -p 5000:5000 d10d85b68ea1 : to run the image we have created. (5000:5000 it is a mapping between out port to docker host port)

12) $ docker-machine ip default : to get the docker container ip.

192.168.99.100 (we will be seeing out app at 192.168.99.100:5000/)

13) $ docker-machine stop default : to stop the docker on server.

Stopping "default"...

Machine "default" was stopped.

14) $ docker-machine start default : to start the docker on server.

Starting "default"...

15) $ docker login : to connect our console with the docker hub login page.

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: gupta020295@gmail.com

Password: EOF

16) $ docker tag mltocloud shubham017/docker\_azure-shubham : tag your image/container to the docker account.

17) $ docker push shubham017/docker\_azure-shubham : push your docker image to the docker hub repository which we have created in docker hub account.

The push refers to repository [docker.io/shubham017/docker\_azure-shubham]

1462f0c4ca9e: Pushed

cbe156e1a15b: Pushed

dd905a880807: Mounted from library/python

5bb5f8e903b2: Mounted from library/python

f2910f1f8a14: Mounted from library/python

07081806a448: Mounted from library/python

c3a984abe8a8: Mounted from library/python

18) pip freeze : this command will give us all the libraries installed in our machine.

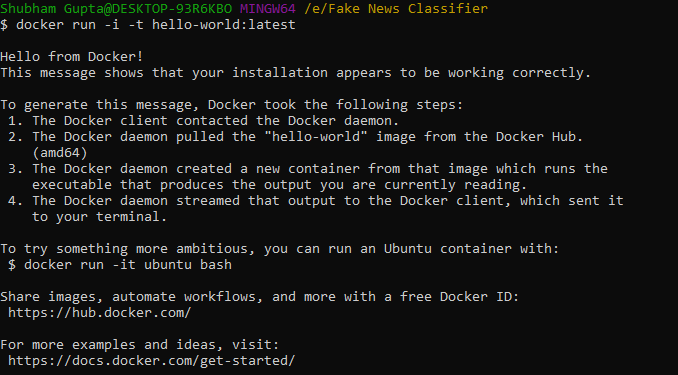
-- This we can use to create the requirement.txt file for Docker purposes.

19) To create requirement.txt file use:

pip freeze > requirement01.txt

20) https://hub.docker.com/u/shubham017 : docker hub account where i can see all my docker images.

21) docker ps -a : list out all the containers on our system (running, stopped, exited all of them).

****

**-------------------------------------------------------------------------------------------------------**

1) Shubham Gupta@DESKTOP-93R6KBO MINGW64 /e/Fake News Classifier

$ docker build .

2) Shubham Gupta@DESKTOP-93R6KBO MINGW64 /e/Fake News Classifier

$ docker push shubham017/docker\_azure-shubham:fake\_news\_classifier

3)

Shubham Gupta@DESKTOP-93R6KBO MINGW64 /e/Fake News Classifier

$ docker commit -m "fake news classifier" -a "ShubhmamGupta" 7e63814ed09a shubham017/fake\_news\_classfier:latest

sha256:4715b85c935856bf02a0b8889c51db9768f6ef0d0f1843716e258cf01ae3b695

Shubham Gupta@DESKTOP-93R6KBO MINGW64 /e/Fake News Classifier

$ docker login 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: shubham017

Password:

4)

Shubham Gupta@DESKTOP-93R6KBO MINGW64 /e/Fake News Classifier

$ docker push shubham017/docker\_azure-shubham