**Docker :-** is a tool designed to make it easier to 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 ship it all out as one package

**Note - Docker container is not any specific platform. It can run on any computer, on any infrastructure and in any cloud.**

Docker makes the process of application deployment very easy and efficient and resolves a lot of issue related to deploying application

NOTE: developer will package all software (front end ,backend, libraries) into container and Docker will take care

to deploy it into different platform

**How Docker works**

Machine generated alternative text:
Developer 
Docker image 
Dockerfile 
Docker Container 
Docker Beginner Tutorial 2 - How DOCKER works I Docker Architecture 
Docker image 
Docker Container 
Staging Environment 
Pull image 
Docker image 
Pull image 
Docker 
Hub 
This resolves the issue of app working on one Dlatform and not on other 

**Docker file :** describe steps to create Docker image or a text file with instruction to build image

**Docker image:** are templates use to create Docker container

**Container:** will have application will all it dependencies ( our application run inside Docker container)

Or

Container is a running instance of image

**In Docker workflow** developer will define all the application and it dependencies and requirement in a file which is called as Docker file and this Docker file can be used to create Docker image so in Docker image we will have all application requirement and dependencies and when we run Docker image we get Docker container so Docker Container are runtime instance of Docker image and these image can also be stored in a online cloud repository

Which is called as Docker hub now these image can be pull to create container in any environment so can

Create Docker container in test environment or any other environment

**Let’s understand virtualization and containerization one by one.**

**What is Virtualization?**

Virtualization is the technique of importing a Guest operating system on top of a Host operating system. This technique was a revelation at the beginning because it allowed developers to run multiple operating systems in different virtual machines all running on the same host. This eliminated the need for extra hardware resource.

**The advantages of Virtual Machines or Virtualization are:**

Multiple operating systems can run on the same machine

* Maintenance and Recovery were easy in case of failure conditions
* Total cost of ownership was also less due to the reduced need for infrastructure

Machine generated alternative text:
App 1 
BINS 
UBS 
VMs 
Guest 
os 
App 2 
BINS 
Guest 
os 
Hypervisor 
Host OS 
App 3 
BINS 
Guest 
os 

In the diagram, you can see there is a host operating system on which there are 3 guest operating systems running which is nothing but the virtual machines.

As you know nothing is perfect, Virtualization also has some shortcomings. Running multiple Virtual Machines in the same host operating system leads to performance degradation. This is because of the guest OS running on top of the host OS, which will have its own kernel and set of libraries and dependencies. This takes up a large chunk of system resources, i.e. hard disk, processor and especially RAM.

Another problem with Virtual Machines which uses virtualization is that it takes almost a minute to boot-up. This is very critical in case of real-time applications.

Following are the disadvantages of Virtualization:

* Running multiple Virtual Machines leads to unstable performance
* Hypervisors are not as efficient as the host operating system
* Boot up process is long and takes time

These drawbacks led to the emergence of a new technique called Containerization. Now let me tell you about Containerization.

**What is Containerization?**

Containerization is the technique of bringing virtualization to the operating system level. While Virtualization brings abstraction to the hardware, Containerization brings abstraction to the operating system. Do note that Containerization is also a type of Virtualization. Containerization is however more efficient because there is no guest OS here and utilizes a host’s operating system, share relevant libraries & resources as and when needed unlike virtual machines. Application specific binaries and libraries of containers run on the host kernel, which makes processing and execution very fast. Even booting-up a container takes only a fraction of a second. Because all the containers share, host operating system and holds only the application related binaries & libraries. They are lightweight and faster than Virtual Machines.

**Advantages of Containerization over Virtualization:**

* Containers on the same OS kernel are lighter and smaller
* Better resource utilization compared to VMs
* Boot-up process is short and takes few seconds

Machine generated alternative text:
App I 
BINS 
UBS 
- App 2 
BINS 
UBS 
Container Engine 
Host OS 
App 3 
BINS 
UBS 

In the diagram on the right, you can see that there is a host operating system which is shared by all the containers. Containers only contain application specific libraries which are separate for each container and they are faster and do not waste any resources.

All these containers are handled by the containerization layer which is not native to the host operating system. Hence a software is needed, which can enable you to create & run containers on your host operating system.

Machine generated alternative text:
App 1 
VMs 
Guest 
os 
App 2 
Guest 
os 
App 3 
Guest 
Hypervisor 
Host OS 
VMS - resource allocation is fixed and does not change as per application needs Machine generated alternative text:
App 1 
VMs 
Guest 
os 
App 2 
Guest 
os 
App 3 
Guest 
App 1 
lib I 
bins 
App 2 
lib 1 
bin 
App 3 
bin 
Hypervisor 
Host OS 
Virtualization 
Container Engine 
Host OS 
Containerization 

**Docker has a client server architecture**

I**n** Docker command line interface is the client and Docker server or Docker Daemon which will have all the container and Docker server receives

Command from Docker client in the form of command or a rest Api request and all the components of Docker client and Docker server together

Forms a Docker engine

Machine generated alternative text:
Docker has a client-server architecture 
Docker Daemon 
Container I 
Container 2 
Container 3 
Client 
Server 
DOCKER ENGINE 

Machine generated alternative text:
Docker has a client-server architecture 
The daemon (server) receives the commands from the Docker client 
through CLI or REST API's 
Docker client and daemon can be present on the same host (machine) or 
different hosts 
Client 
Server 

Machine generated alternative text:
Now you understand 
the basics of Docker 
Dockerfile 
Docker Images 
Docker Containers 
Docker Hub / 
Registry 
Docker Client 
Docker Server 
(daemon) 
Docker Engine 

**Advantages of Docker**

**1)**It resolves the problem of code working on one system not working on other system.

2)**Build app only once**

An application inside a container can run on any system that has Docker installed. So there is no need to build and configure app multiple times

On different platforms

Machine generated alternative text:
Docker image 
Docker Container 
Test Environment 
Docker image 
Docker Hub 
Docker image 
Docker Container 
Prod Environment 

1. With Docker you test your application inside a container and ship it inside a container.

This means the environment in which you test is identical to the one on which the app will run in production

4)

Machine generated alternative text:
Portability 
Docker containers can run on any 
platform. 
It can run on your local system, 
Amazon ec2, Google Cloud platform, 
Rackspace server, VirtualBox..etc. 
A container running on AWS can easily be ported to VirtualBox 

5)

Machine generated alternative text:
Version Control 
Like Git, Docker has in-built version 
control system 
Docker containers work just like GIT 
repositories, allowing you to commit changes 
to your Docker images and version control 
them 

6)

Machine generated alternative text:
App 1 
lib I bins 
App 2 
lib I bin 
Docker Engine 
Host OS 
App 3 
lib / bin 
Every app 
works in its 
own container 
and does not 
interferes with 
other apps 

Machine generated alternative text:
With Docker every application works 
in isolation in its own container and 
does not interferes with other 
applications running on the same 
system. 
So multiple containers can run on same system without 
interference. 
For removal also you can simply delete the container and it will 

7)

Machine generated alternative text:
Productivity 
Docker allows faster and more 
efficient deployments without 
worrying about running your app on 
different platforms. 
It increases productivity many folds. 

Docker Architecture

Docker follows client-server architecture. Its architecture consists mainly three parts.

1) **Client:** Docker provides Command Line Interface (CLI) tools to client to interact with Docker daemon. Client can build, run and stop application. Client can also interact to Docker\_Host remotely.

2) **Docker\_Host:** It contains Containers, Images, and Docker daemon. It provides complete environment to execute and run your application.

3) **Registry:** It is global repository of images. You can access and use these images to run your application in Docker environment.

Machine generated alternative text:
Client 
docker build 
docker pull 
docker run 
DOCKER HOST 
Containers 
Docker daemon 
Images 
Registry 
NGMX 

**Installation of docker**

1. sudo yum -y update
2. Install docker :- sudo yum install -y docker
3. To check version : Docker --version
4. Start docker :- sudo service docker start
5. Docker info
6. To list out image: docker images
7. To list out container :- docker ps
8. To list out all container :- docker ps -a
9. To run a image : docker run "image\_name" docker run hello-world (<https://hub.docker.com/> here we can search image)

Machine generated alternative text:
/ J # docker run hello—world 
Unable to find image 'hello—world: latest' locally 
Trying to pull repository registry. access.redhat.com/hello—world . 
Trying to pull repository docker.io/library/hello—world . 
latest: Pulling from docker.io/library/hello—world 
9db2ca6ccae0: Pull complete 
Digest: sha256: 4b8ff392a12ed9ea17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc 
Status: Downloaded newer image for docker.io/hello—world: latest 
Hello from Docker! 
This message shows that your installation appears to be working correctly. 

Machine generated alternative text:
REPOSITORY 
SIZE 
docker.io hello—world 
1.85 kB 
/ J # docker 
TAG 
latest 
images 
IMAGE ID 
2cbOd9787c4d 
CREATED 
6 weeks ago 

Machine generated alternative text:
/ J # docker 
IMAGE 
PORTS 
hello—world 
CONTAINER ID 
US 
bc60b6772fd8 
ed (0) 2 minutes ago 
COMMAND 
NAMES 
" / hello" 
modest 
CREATED 
2 minutes ago 
bohr 
STAT 
Exit 

10)Stop docker :- sudo service docker stop

1. Uninstall docker :- sudo yum remove docker

Machine generated alternative text:
You can visit - https://qet.docker.com/ 
for more installation related help 
To install docker from binaries 
https://docs.docker.com/engine/installation/binaries/ 
Installation steps for amazon ec2 
http://docs.aws.amazon.com/AmazonECS/latest/developerguide 
Idocker-basics.html 

**Explanation of command** docker run hello-world

The above command **docker run hello-world** has three parts.

1) **docker:** It is docker engine and used to run docker program. It tells to the operating system that you are running docker program.

2) **run:** This subcommand is used to create and run a docker container.

3) **hello-world:** It is a name of an image. You need to specify the name of an image which is to load into the container.

**Docker java example (**<https://www.javatpoint.com/docker-java-example>)

1. **Create a directory**

$ mkdir  java-docker-app

1. **Create a Java File**

**// Hello.java**

**class** Hello{

**public** **static** **void** main(String[] args){

System.out.println("This is java app \n by using Docker");

}

}

1. **Create a Dockerfile**

After creating a Java file, we need to create a Dockerfile which contains instructions for the Docker. Dockerfile does not contain any file extension. So, save it simple with **Dockerfile** name.

FROM java:8

RUN javac Hello.java

CMD ["java", "Hello"]

Write all instructions in uppercase because it is convention. Put this file inside **java-docker-app** directory. Now we have Dockerfile parallel to Hello.java inside the **java-docker-app** directory.

1. **Build Docker Image**

$ docker build -t java-app .

1. **Run Docker Image**

docker run java-app

|  |  |
| --- | --- |
|  | docker basic commands: |
|  |  |
| docker version |  |
| docker -v | give the version of docker |
| docker info | this will give detail information about docker installed |
| docker --help |  |
|  | Images basic command |
|  |  |
| docker images | list out all the images on system |
| docker images --help |  |
| dockes images -q | give images id |
| docker pull "image name" | to pull the image from docker hub |
| docker pull "image name:tag" | pull image with particular tag |
| docker rmi "image id" | to remove images |
| delete all images | docker rmi $(docker images -q) |
| dangling images | image not associated with running container |
| docker run -it ubuntu | inside ubuntu container to come out of it enter "exit" command |
| docker inspect "image name" | inspect the image what is there inside it |
| docker history "image-id" | history of image |
|  |  |
|  | Container basic Commands |
|  |  |
| docker ps --help |  |
| docker run "image name" | if it not found locally it will install image from hub |
| to run container and give name | **docker run --name Myubuntu -it(interactive mode) ubuntu bash** |
| docker start "container id" or name | start the container |
| docker stop "container id" or name | stop the container |
| delete container id | docker rm <CONTAINER ID> |
| delete all container id | docker rm $(docker ps -a -q) |
| stop all container | docker stop $(docker ps -a -q) |
| docker stats "container id" | it will tell about container id |
| docker attach "container id " |  |
|  |  |
|  | System Commands |
|  |  |
| docker stats | memory usage of container |
| docker system df | tell about all the usage of memory |
| docker system prune | will remove all stop container and images |
|  |  |
|  | **Dockerfile command ( A text file with instruction to build image)** |
|  | automation of docker image creation |
|  |  |
|  | create image using dockerfile and run image to get instance of container |
|  |  |
|  | FROM ubuntu(base image) |
|  | Maintainer saurabh<automation> ( optional) |
|  | RUN apt-get update (want to run something) |
|  | CMD ["echo","hello world"] |
|  |  |
| **note** | RUN get executed during the building of the image and command inside CMD will get executed only when we create |
|  | container out of the image |
|  |  |
| **build the image from dockerfile** | docker build -t image name |
|  |  |
| **run docker image** | docker run image name to create container |

**Docker Compose**

Tools for defining and running multi-container docker application

Use yaml files to configure application services(docker-compose.yml)

Can start all services with a single command : docker compose up

Can stop all service with a single command : docker compose down

Can scale up selected services when required

**difference between docker and kubernetes**

<https://blog.containership.io/k8svsdocker>

Let’s think about how we were deploying using the kubectl command line.

<https://medium.com/ingeniouslysimple/deploying-kubernetes-applications-with-helm-81c9c931f9d3>

Machine generated alternative text:
kubectl create -f 
. / api -deployment. yaml 
kubectl create -f 
. / ks/web- api -service. yaml 
kubectl create -f 
. / ks/worker -deployment-I. yaml 
kubectl create -f 
kubectl create -f 
. /ks/ngå nx-service. yaml 
kubectl create -f 
. /ks /nglnx-deployment. yaml 
kubectl create -f 
. /ks/worker -deployment-2. yaml 
kubectl 
command? 
./ks/ worker-service. yaml 
Deploying using kubectl 
Notice how for each resource, we have to run a manual 

**Compare that to deploying with helm**

Machine generated alternative text:
- HELM 
helm install ./ks/ -n ks 

We’re able to deploy our entire application by pointing helm at a directory containing all of our Kubernetes resources, using one command line call.

**Revision management**

Helm keeps track of how many times an application has been deployed on a cluster. You can inspect what revision number you’re currently at, and you can even rollback to previous revisions if necessary. Doing this with kubectl is possible, but it's much more difficult than just running helm rollback <APPLICATION> <REVISION\_NUMBER>.