Docker

**Introduction:**

Docker is a container management service. The keywords of Docker are *develop, ship* and *run* anywhere. The whole idea of Docker is for Developers to easily develop applications, ship them into containers which can then be deployed anywhere.

Docker is a tool designed to make it easier to deploy and run applications by using containers. It allows developers, sys-admins, etc., to easily deploy their applications in a container (Sandbox) to run on the host operating system i.e., Linux. The key benefit of Docker is that it allows users to package an application with all its dependencies into standardized unit for software development.

**Containers:**

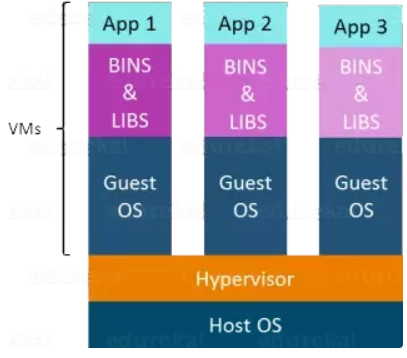
Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies and ship it all out as one package. Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers.

**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:

*Advantages*

* Multiple operating systems can run on the same machine
* Maintenance and Recovery is easy in case of failure conditions
* Total cost of the ownership is also less due to the reduced need for infrastructure



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.

*Hypervisor:* A hypervisor is a program that would enable you to host several different virtual machines on a single hardware. Each one of these virtual machines or operating systems you have will be able to run its own programs, as it will appear that the system has the host hardware's processor, memory and resources. However, it is actually the hypervisor that is allocating those resources to the virtual machines.

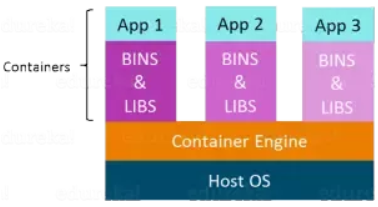
Virtualization also has some shortcomings. Running multiple Virtual Machines in the same host operating system leads to performance degradation. This is because of the guest OS running on top of the host OS, which will have its own kernel and set of libraries and dependencies. This takes up a large chunk of system resources, i.e. hard disk, processor and especially RAM.

*Disadvantages*

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

These drawbacks led to the emergence of a new technique called Containerization.

**Containerization:**

Containerization is a type of virtualization strategy that emerged as an alternative to traditional hypervisor -based virtualization. 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.  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

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.

**Virtualization vs Containerization**

The following table gives a direct comparison between Virtualization and Containerization

|  |  |
| --- | --- |
| Virtualization | Containerization |
| Virtual machines (VMs) | Containers |
| Represents hardware level virtualization | Represents Operating system virtualization |
| Heavyweight | Lightweight |
| Slow provisioning | Real time provisioning and scalability |
| Limited performance | Native performance |
| Fully Isolated and hence more secured | Process level isolation and hence less secure |

**What is a Docker?**

Docker is a tool designed to make it easier to deploy and run applications by using containers. Containers allow a developer to packaging up an application with all of the parts it needed, such as libraries and other dependencies and ship it all out as one package

*Workflow of Docker:*

* Developer defines all the application dependencies and its requirements in a file called “Dockerfile”
* The Dockerfile is used to create docker images. These images consist of all the application dependencies
* We will get docker containers by running docker images. Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies and ship it all out as one package
* So, the docker containers are the run time instances of a docker image. You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.
* The docker images can be stored in an online cloud repository which is called as “Docker Hub”

**Installation: (on Ubuntu)**

**Ref:** <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04>

Simply run the following commands in the ubuntu terminal

1. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
2. sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"
3. sudo apt-get update
4. sudo apt-get install -y docker-ce
5. sudo systemctl status docker

**Docker Hub:**

Docker Hub is a registry service on the cloud that allows you to download docker images that are built by other communities. You can also upload your own docker built images to Docker hub.

**Docker Images:**

Docker image is a combination of a file system and parameters. It is a file comprised of multiple layers, used to execute code in a Docker container.

**Docker Container:**

A container is a standard unit of software that packages up code and all the dependencies, so the application runs quickly and reliably from one computing environment to another. A docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application. Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers.

**Dockerfile:**

A Docker File is a simple text file with instructions on how to build your images. It is a text document that contains all the commands a user could call on the command line to assemble an image. Using “*docker build*” users can create an automated build that executes several command-line instructions in succession.

**Docker Client:**

In Docker command line interface is the “client”.

**Docker Server:**

Docker server consists of all the docker containers in it. It receives the commands from the docker client in the form of docker commands or Rest API request. A server which is a type of long-running program called a daemon process. A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.

**Docker Engine:**

All the components of docker client and docker server together called as Docker engine.

Note: The docker daemon (server) and docker client can be present on the same host or different hosts

**Benefits of using Docker:**

1. *Build app only once* – An application inside a container can run on any system that has a docker installed. So, there is no need to build and configure app multiple times on different platforms
2. *More Sleep and Less worry* – With docker you test your application inside the 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 the production.
3. *Portability* – Docker containers can run on any platform. It can run on your local system, ec2, Google cloud platform, Rackspace server, VirtualBox etc., A container running on AWS can easily be ported to VirtualBox.
4. 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
5. Isolation – With docker every application works in isolation in its own container and doesn’t 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 leave behind any files or traces on the system
6. Productivity – Docker allows faster and more efficient deployments without worrying about running your app on different platforms.

**Docker basic commands:**

*Basic***:**

|  |  |
| --- | --- |
| Command | Description |
| docker version | Gives the information about the docker client and docker server (Engine) |
| docker -v | Version of docker |
| docker info | Gives the detailed information about the docker that is installed |
| docker --help | To get the information on any other command  Example: *docker images --help* |
| docker login | Use this to login to the docker hub account |

*Images:*

|  |  |
| --- | --- |
| Command | Description |
| docker images | Gives the list of all the available images we have |
| docker pull | To pull the image from the repository. Example: *docker pull ubuntu* |
| docker rmi | To remove one or more docker image |

*Containers:*

|  |  |
| --- | --- |
| Command | Description |
| docker ps | List the containers |
| docker run | To run a container if it is available locally or else it would pull the image from docker hub and then run it. Example:   * *docker run ubuntu* * *docker run -it ubuntu* |
| docker start | To start the container. Example: docker start containerID/Name |
| docker stop | To stop the container. Example: docker stop containerID/Name |

*System:*

|  |  |
| --- | --- |
| Command | Description |
| docker stats | Gives all the statistics like memory usage, IO etc., for docker |
| docker system df | To check the disk usage of docker |
| docker system prune | This will remove:  - all stopped containers  - all networks not used by at least one container  - all dangling images  - all dangling build cache |

**Docker Images:**

In docker everything is based on images. A container is launched by running an image. An image is an executable package that includes everything needed to run an application such as the code, a runtime, libraries, environment variables and configuration files.

*Docker image:* Manage Images

*Syntax:*

docker image <command>

Child commands:

|  |  |
| --- | --- |
| Command | Description |
| docker image build | Build an image from a Dockerfile |
| docker image history | Show the history of an image |
| docker image import | Import the contents from a tarball to create a file system |
| docker image inspect | Display detailed information on one or more images |
| docker image load | Load an image from a tar archive or STDIN |
| docker image ls | List images |
| docker image prune | Removed unused images |
| docker image pull | Pull an image or a repository from registry |
| docker image push | Push an image or repository to a registry |
| docker image rm | Remove one or more images |
| docker image save | Save one or more images to a tar archive |
| docker image tag | Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE |

*Docker images:* List images

*Syntax:*

docker image [options] [repository[:tag]]

Options:

|  |  |
| --- | --- |
| Command | Description |
| --all, -a | Show all images |
| --digest | Show digests |
| --filter, -f | Filter output based on commands provided |
| --format | Pretty-print images using a Go template |
| --no-trunc | Don’t truncate output |
| --quiet, -q | Only show numeric ids (image ids) |

*Examples:*

Command: docker images -f "dangling=true"

output: prints the list of dangling images

dangling: Dangling images are layers that have no relationship to any tagged images. They no longer serve a purpose and consume disk space. They can be located by adding the filter flag, -f with a value of dangling=true to the docker images command.

**Docker Containers:**

A container is a runtime instance of an image. These instances can be run using the docker run command. The basic purpose of docker is to run the containers.

*docker container:* manage containers

*Syntax:*

docker container <command>

Child commands:

|  |  |
| --- | --- |
| Command | Description |
| docker container attach | Attach local standard input, output, and error streams to a running container |
| docker container commit | Create a new image from a container’s change |
| docker container cp | Copy files/folders between a container and the local file system |
| docker container create | Create a new container |
| docker container diff | Inspects changes to files or directories on a container’s filesystem |
| docker container exec | Run a command in a running container |
| docker container export | Exports a container’s filesystem as a tar archive |
| docker container inspect | Displays detailed information on one or more containers |
| docker container kill | Kill one or more running containers |
| docker container logs | Fetch the logs of a container |
| docker container ls | List containers |
| docker container pause | Pause all processes within one or more containers |
| docker container port | List port mappings or a specific mapping for the container |
| docker container prune | Remove all stopped containers |
| docker container rename | Rename a container |
| docker container restart | Restart one or more containers |
| docker container rm | Remove one or more containers |
| docker container run | Run a command in a new container |
| docker container start | Start one or more stopped containers |
| docker container stats | Display a live stream of container(s) resource usage statistics |
| docker container stop | Stop one or more running containers |
| docker container top | Display the running processes of a container |
| docker container unpause | Unpause all processes within one or more containers |
| docker container update | Update configuration of one or more containers |
| docker container wait | Block one or more containers stop, then print their exit codes |

**Dockerfile:**

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using *docker build* users can create an automated build that executes several command-line instructions in succession.

1. Create any text file named Dockerfile in any location (Ex. Desktop)

mkdir Dockerfile

1. Add instructions to the Docker file

# Getting base image ubuntu

FROM ubuntu # ubuntu is a base docker image

MAINTAINER Naveen Noorbhasha

RUN apt-get update

CMD ["echo", "Hello Naveen...! This is your first docker image"]

1. Build Dockerfile to create image

docker build -t myimage1:1.0 path\_to\_dockerfile

1. Run image to create container

docker run image\_id

**Docker Swarm:**

A swarm is a group of machines that are running Docker and joined into a cluster. Docker swarm is a tool for container orchestration. Let’s take an example:

Assume that we have 100 containers, we need to do the following –

* Health check on every container
* Ensure all containers are up and running on every system
* Scaling the containers up or down depending on the load
* Adding updates/changes to all the containers

***Orchestration:*** Managing and controlling multiple docker containers as a single service. Tools like Docker Swarm, Kubernetes, Apache Mesos will be used

***Pre-requisites:***

* Docker 1.13 or higher
* Docker machine (Pre- installed for docker on windows)

**Steps:**

1. Create Docker machines (To acts as nodes for Docker Swarm)
   1. Create one machine as a manager and others as workers. Run windows terminal as administrator and run the below command:

*docker-machine create –driver hyperv manager1*

Note: Common error will be occurred in windows – “Error with pre-create check: "no External vswitch found. A valid vswitch must be available for this command to run. Check <https://docs.docker.com/machine/drivers/hyper-v/>". Follow the given link in the error message

1. Check machine created successfully

docker-machine ls

docker-machine ip manager1

1. Create worker machines

*docker-machine create –driver hyperv worker1*

*docker-machine create –driver hyperv worker2*

1. SSH (Connect) to docker machine

*docker-machine ssh manager1*

1. Initialize Docker Swarm (We must start the swarm in manager machine)

*docker swarm init –advertise-addr <MANAGER\_IP>*

*docker node ls*

1. Join the workers in the swarm

*docker swarm join-token worker*

It will generate the following command to add workers to the swarm

**Ex:** docker swarm join --token SWMTKN-1-3mb9lcc0wxjxjkb46l9ahfa087uddccczakpnlc05im20fnwhu-7rt4r8e3t1rpxtht424xeu7pp 10.91.120.46:2377

Run the above command in the worker nodes

Check the node list in the manager node

docker node ls (Output)

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

nz4pu4k8cmei3jc9lalapw582 \* manager1 Ready Active Leader 18.09.0

gw8nblof5nitcuajnlpqrgf4c worker1 Ready Active 18.09.0

uksinsaxyd7ndtf8jb3g090fx worker2 Ready Active 18.09.0

1. On manager run standard docker commands

*docker info*

(Check the swarm section, no. of manager, nodes, etc.,)

Now check docker swarm command option

*docker swarm*

1. Run containers on Swarm

docker service create –replicas <replication count> -p <port number> –name <Service Name> <service>

**Ex:** docker service create –replicas 3 -p 80:80 –name web1 nginx

Check the status:

docker service ls

docker service ps service\_name

* Check the services running on all the nodes and on the browser by giving ip for all the nodes

1. Scale service UP and DOWN

Run the following command on manager node

docker service scale service\_name = no. of instances

Ex: docker service scale web1=4

1. Inspecting Nodes (This command only run on manager node)

docker node inspect nodename

Examples:

docker node inspect self

docker node inspect worker1

1. Shutdown Node

docker node update –availability drain node\_name

**Ex:** docker node update –availability drain worker1

1. Update the service

docker service update –image <image\_name>:<version> service\_name

**Ex:** docker service update –image nginx:1.14.0 web1

1. Remove the service

docker service rm service\_name

**Ex:** docker service rm web1

* – To leave the swarm
* docker-machine stop machine\_name – To stop the machine
* docker-machine rm machine\_name – To remove the machine