# Docker

1. **Before Docker** -> Application is running on servers. Some companies have their own dedicated servers in their data centre and some companies borrows servers from cloud providers. Initially what use to happen is only one application can run on one server. So if you wanted to increase application running you need to setup another server. This was not very cost efficient and environment friendly. Vmware came up with a solution called as Virtual Machine. Using Virtual machine you can run multiple applications on same server. But the biggest problem with VM was VM requires it's own Operating Systems. OS uses RAM,CPU, hard disk etc. Even if a particular amount of resources are been allocated to a particular OS it may be possible that all the resources are not utilised. There can be issues related to licenses of OS etc. VM were better than running one application on one server, but it was not perfect. Vm were not super fast, needed dedicated OS, dedicated cpu, ram allocation. When you develop an application or website, and share your source code with your friend, and your friend tries to run the application on his PC, and he faces some issues like some version missing or depreciated, version mismatch etc. And you say " But it is working on my Machine". Such issues like having various dependencies, modules, packages issues docker is trying to solve.
2. **Containers** -> If you want to run multiple application in isolation on same Operating system, then we need to use container. In Virtual Machine we have the underlying infrastructure, on top of that there is Hypervisor. Hypervisor is used to create multiple machines on a host OS and it manages virtual machines. In containers we have the infrastructure on top of that there is a Host OS. On host OS we have container engine and many applications running on it in isolation. Using containers we can deploy application. Docker is a tool which helps us to create and manage containers.
3. **Running Docker on Windows** -> We have to install Docker Desktop. Windows containerised application cannot run on linux based docker and vice versa. Therefore windows container will require windows host and linux container will require linux host. We need to use Docker Desktop, WSL( Windows Subsystem for linux ).
4. **Running Docker on MAC** -> Virtual Machine, Docker Desktop.
5. **Docker** -> Docker is a container platform that allows you to build, test and deploy applications quickly. Using Docker we can run applications in any environment.
6. **Installation** -> For Mac and Windows, download Docker Desktop. For Linux - https://docs.docker.com/desktop/install/ubuntu/

7. **Docker Runtime** -> It allows us to start and stop the containers. Runtime is of two types, runc and container-d. The role of runc is to work with operating system and start and stop the container. The role of container-d is managing runc and container. It also does stuff like how to get data from the internet to the network(pulling images). Every docker container will have runc and container-d.
8. **Docker Engine** -> We use Docker Engine to interact with docker. When we type docker command in Docker CLI that rest API call goes to Docker daemon and daemon will tell Runtime to run the command. Docker uses client-server architecture.
9. **Orchestration** -> Orchestration allows us to manage the containers like auto scaling the containers, restarting few containers, managing rolling updates of applications in containers etc. Example of Orchestration engines are Docker Swarm and Kubernetes. Kubernetes is much more than container orchestration engine, but it is one of the functionality been provided.
10. **Docker/Container Image** -> If i want someone to run my application, i can create image of my application and share that image, and when they run that image, the application will start. Image is just a file that contains all the instructions. Container is a running instance of that image. We can also create docker images using Dockerfile. After running Dockerfile we get docker image and after running docker image we get a container.
11. **Open Container Initiative (OCI)** -> There are many container runtime like container-d, runc , rkt , lxc , crio etc. Which created confusion. So they came up with OCI. OCI is a linux foundation project to design open standards for containers. OCI currently contains two specifications: the Runtime Specification(runtime-spec) and Image Specification(image-spec).
12. **What is Devops** -> DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market. Devops lifecycle includes Planning, Code, Build, Test , Release , Deploy, Operate and Monitor.
13. **docker run hello-world** -> run is used to run an image to create a container. hello-world is an image name. When we run the command, docker says that you don't have that image on your local system. So it will download from Docker Hub and then run that image.

14. **docker images** -> shows all the docker images downloaded on my local machine.
15. **What happens when you run docker run hello-world command ?** -> When we run the command on Docker Client, client will talk to Daemon, Daemon will check whether that image is available on local system, if it is not available then it will download it from dockerhub, and run that image and forms a container and shows the output on the client.
16. **How a Docker Image works** -> Docker image also contains OS files. It contains application and also dependency for that application. Every image contains a base image which is a smaller version of OS which can be used to run main application.
17. **docker pull image_name** -> Download the docker image from docker hub. Eg- docker pull ubuntu:16.04 (if version not mentioned it pulls the latest one).
18. **docker run -it ubuntu:16.04** -> Run commands inside the ubuntu container as a root user.
19. **docker ps** -> See all the running containers. Can also use **docker container ls**
20. **docker container exec -it container_id bash** -> Bash shell should be attached to the running container. Command will not work if container is not running. You can now execute commands inside the container.
21. **docker start container_id** -> Start the container.
22. **docker stop container_id** -> Stop the running container.
23. **docker ps -a** -> It shows all the containers which are stopped as well as running.
24. **docker rm container_id** -> Remove the docker container.
25. **docker inspect container_id** -> It gives all the information about the container like id, when it was created, state, image info, path etc.
26. **docker logs container_id** -> fetches logs of a container.
27. **docker container prune -f** -> delete all the stopped containers.
28. **docker run -d image_name**-> Run the docker container in background, in detached mode.
29. **docker run image_name echo Hey** -> Run the container and print Hey in the terminal.
30. **docker logs --since 5s container_id** -> Show the last 5 sec logs of a particular container.
31. **docker rmi image_name** -> Remove a docker image.
32. **docker run -d -p 8080:80 nginx** -> Download nginx from the docker hub and run it in a container and access the nginx on a localhost port 8080 by forwarding the all the request which is been made to localhost port 8080 to the nginx container port

80 which is default port of nginx.

33. **docker commit -m "added names.txt file" container_id new_image_name** -> If you made some changes in docker container and want to share that as a image we can commit it to the container and create new docker image out of it which you can share wherever you want.
34. **docker rmi $(docker images -q)** -> Remove all the docker images.
35. **docker rm $(docker ps -a -q)** -> Delete all the stopped containers.
36. **Layers** -> Images are built in layers. Each layer is an immutable file, but is a collection of files and directories.Layers receive an ID, calculated via a SHA 256 hash of layer contents.
37. **How to create Docker Image** -> Dockerfile is used to create Docker Images. First create a Docker file and then write
    **FROM ubuntu** -> Base image
    **MAINTAINER yash patil <abc@gmail.com>** -> who is the maintainer
    **RUN apt-get update** -> run this command when the docker container is created.
    **CMD ["echo", "Hello World"]** -> CMD is the command the container executes by default when you launch the built image.
38. **docker build -t image_name:version .** -> Create Docker Image from a Docker file. And . in the command is the path where Docker file is present, . means current directory.
39. **Architecture Of Docker Engine**-> We have the Docker client and client communicates with the Docker Daemon. Docker Daemon communicates with the container-d(which starts and stops the container). Container-d is attached to shim and runc and Docker container. When we run docker run command the docker client talks docker daemon and docker daemon talks to container-d using grpc and then container-d will talk to runc will talk to OS, kernels and start a docker container. Container-d is a CNCF project. Even when the Daemon is down or updating the container would still be running.