# DOCKER

* What is docker?

Docker is a containerization platform which is used to package applications with its dependencies in the form of containers which can be easily deployed and run on any machine that supports docker. It uses container technology to isolate processes and provide a lightweight, portable solution for application deployment.

* Why do you need docker?

**Compatibility/dependency -** Certain versions of the applications will not be compatible with the OS. Also, we need to check on the compatibility between these applications and the libraries and dependencies on the OS. When we face issues like where one application requires one version of a dependent library whereas another application requires another version and When we upgrade to newer versions of these applications/services, we have to go through the same process of checking compatibility between these various applications and the underlying infrastructure(OS)

**or**

Applications require certaindependencies/libraries. So each time we install an application or upgrade an application we need to check the compatibility between the application and the underlying infrastructure(OS). There is a dependency issue which has to be resolved each time in this process. So using docker will eliminate this need.

For eg, an application may require one version of dependency similarly another application may require another version of the same dependency. There comes a clash between the two versions. In such cases docker is preferred as it packages applications in separate containers with it own dependencies  
**long setup time** – It takes a long time to setup the application while checking all the dependencies and compatibility.  
**different dev/test/prod environments**

* What can it do?

Run each application/component in a separate container with its own dependencies and libraries on the same VM and OS.  
containerize applications

* What are containers?

Containers are completely isolated environments as they have their own processes, services, network interfaces, mounts just like VM except they share the same OS kernel

* Main purpose of docker?

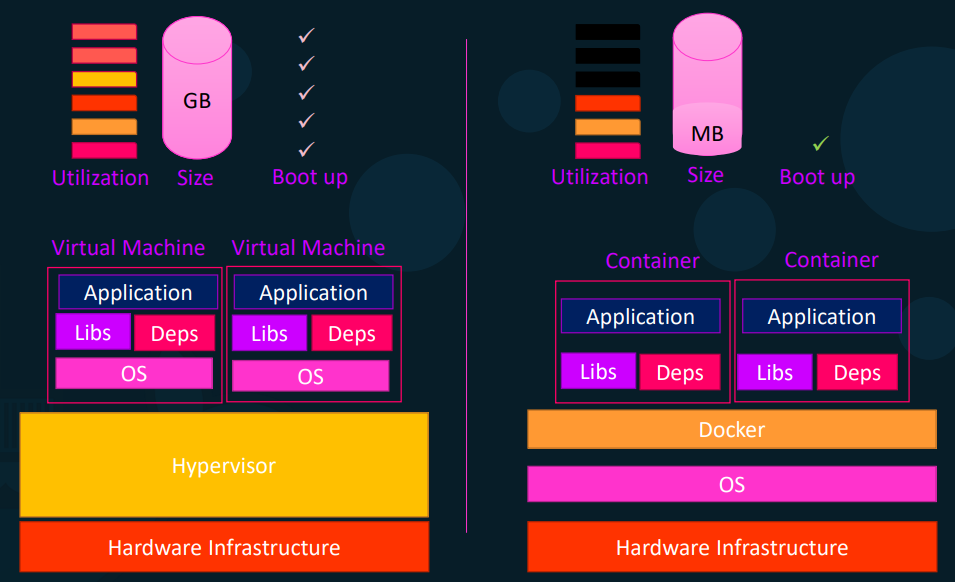
To package and containerize applications and to ship and run them anywhere

* Benefits of docker

1. Easy and faster configuration
2. Application Isolation
3. Swarm
4. Routing mesh
5. Services
6. Security Management
7. Uses less memory
8. Runs the containers in seconds instead of minutes
9. It does not a require full operating system to run applications.

* Containers vs Virtual Machines - A VM lets you run a virtual machine on any hardware. Docker lets you run an application on any operating system

1. Utilization of underlying resources is higher in VMs as there are multiple virtual OS and kernel running.
2. As there are multiple Virtual OS and kernel running, the VM also consume higher disk space as each VM is heavy and is usually in GB(gigabytes) in size whereas docker containers are lightweight and are usually megabytes in size.
3. This allows docker containers to boot up faster usually in seconds whereas VMs takes minutes to boot up as it needs to boot up the entire operation system
4. Docker has less isolation as more resources are shared between the containers whereas VMs have complete isolation from each other since VMs don’t rely on underlying OS/kernel



* Docker Hub

Public docker repository which contains containerized versions of applications.

Ex – Images of most OS DBs and other services and tools

We can create our own image and push it to the Docker hub repository.

* Container vs Image

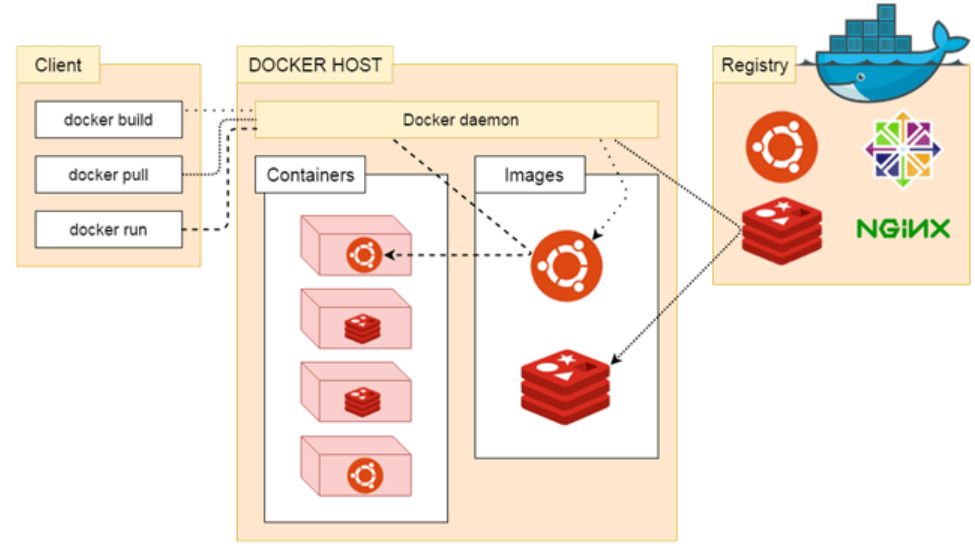
Docker image is a template which is used to create one or more containers.

Containers are running instances of images that are isolated and have their own environments.

* What is a namespace ?

Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container. These namespaces provide a layer of isolation.

* Docker Architecture



**Docker client** uses commands and REST APIs to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request.

Docker Client uses Command Line Interface (CLI) to run the commands.

**Docker Host** is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.

**Docker Registry** manages and stores the Docker images.

2 types of registry in docker – public registry(docker hub) & private registry

* What is docker daemon?

Docker daemon runs on the host operating system. It is responsible for running containers to manage docker services. Docker daemon communicates with other daemons. It offers various Docker objects such as images, containers, networking, and storage.

* Run/Start a container – **docker run nginx –** runs an instance of nginx image
* List containers – **docker ps** (lists only running containers), **docker ps -a** (lists all containers)
* Stop a container – **docker stop <containername>**
* Remove a container – **docker rm <containername>**
* List images – **docker images**
* Remove images – **docker rmi <imagename>**
* Pull-download an image – **docker run <imagename> or docker pull <imagename>**
* **docker run ubuntu sleep 5** – when the container starts it runs and goes into sleep for five seconds
* Exec-execute a command – **docker exec <containername> cat /etc/hosts**
* Run the container in background mode – **docker run -d namespace/<containername>**
* To name the docker container - **docker run -d --name webapp nginx:1.14-alpine**
* To delete all available images – **docker rmi $(docker images -aq)**
* Docker run tag – **docker run redis:4.0 ,** when no tag is given default tag will be taken as latest
* Docker run STDIN – **docker run -i namespace/<containername>** - interactive mode to provide inputs.

**docker run -it namespace/<containername>** - pseudo terminal(t)

* Docker run – Port mapping – **docker run -p 80:5000 namespace/<containername>**

Here 5000 is the internal port, 80 is the external port

docker run -p 8282:8080 webapp-color -> 8282 runs on host, 8080 runs on container

* Docker run – Volume mapping – **docker run -v /opt/datadir:/var/lib/mysql mysql** where /var/lib/mysql is the path inside container – mounts a external directory to a folder inside a docker container – so when the docker container runs it retains the data in the external directory.
* Returns details of a container – Inspect Container – **docker inspect <containername>**
* Container Logs – **docker logs <containerID>/<containername>**
* **docker run timer** – shows date and time infinitely.
* **docker run -d timer** – runs the container in background and does not show output.

**docker attach <containerid>** - when this command is given, it shows the output/console of the docker container.

* **docker run --name myjenkins -p 8080:8080 -v /root/my-jenkins-data:/var/jenkins\_home jenkins** – To retain the data and configurations done in the container, we need to map the volume.
* **docker run python:3.6 cat /etc/\*release\*** - to find the base operating system information
* **docker image tag my-image localhost:5000/my-image –** to tag an image
* **docker exec <containerID> ps -eaf –** lists all process running inside the container.
* **docker system prune** – remove all stopped containers, all the networks that are not used, all dangling images and all build caches.
* **docker rm -f <containerid/containername>** - deletes a running container
* Restart policies - **docker run -d --restart unless-stopped redis**

1. Off: In this, the container won’t be restarted in case it's stopped or it fails.

2. On-failure: Here, the container restarts by itself only when it experiences failures not associated with the user.

3. Unless-stopped: Using this policy, ensures that a container can restart only when the command is executed to stop it by the user.

4. Always: Irrespective of the failure or stopping, the container always gets restarted in this type of policy.

* Restart docker container – **docker restart <containername/id>**

**DOCKERFILE**

* Dockerfile is a set of instructions which is used to build images
* Dockerfile is a text file written in a specific format that docker can understand. Its in an Instruction argument format.

Dockerfile supports the following instructions

ADD Add local or remote files and directories.

ARG Use build-time variables.

CMD Specify default commands.

COPY Copy files and directories.- copy source code of the application from local directory to the docker container.

ENTRYPOINT Specify default executable.

ENV Set environment variables.

EXPOSE Describe which ports your application is listening on.

FROM Create a new build stage from a base image.

HEALTHCHECK Check a container's health on startup.

LABEL Add metadata to an image.

MAINTAINER Specify the author of an image.

ONBUILD Specify instructions for when the image is used in a build.

RUN Execute build commands.

SHELL Set the default shell of an image.

STOPSIGNAL Specify the system call signal for exiting a container.

USER Set user and group ID.

VOLUME Create volume mounts.

WORKDIR Change working directory.

* FROM ubuntu

RUN apt-get update

RUN apt-get install -y python python-pip

RUN pip install flask

COPY app.py /opt/app.py

ENTRYPOINT FLASK\_APP=/opt/app.py flask run --host=0.0.0.0

* Docker build -t webapp:lite . – whenever docker builds an image it builds and caches each layer
* The docker build command is used to build an image from the Dockerfile. You can use the -f flag with docker build to point to a Dockerfile anywhere in your file system.

docker build -f /path/to/a/Dockerfile .

* Push to repository :-

docker login

docker push imagename

Environment Variables

To find environment variables of a container

**docker inspect containername**

**docker inspect 88142f9b11e6 | grep -i APP\_COLOR**

under Env we can find the variables

To define environment variables

**docker run -e APP\_COLOR=blue containername**

CMD

FROM ubuntu

CMD sleep 5 or CMD [“sleep”,”5”]

docker run ubuntu-sleeper sleep 10

ENTRYPOINT

FROM ubuntu

ENTRYPOINT [“sleep”]

docker run ubuntu-sleeper 10

CMD vs ENTRYPOINT

In case of CMD, the command line parameter past will get replaced entirely, whereas in case of entrypoint the command line parameter will get appended.

FROM ubuntu

ENTRYPOINT [“sleep”]

CMD [“5”]

**DOCKER COMPOSE**

Docker Compose is a YAML file which contains details about the services, networks, and volumes for setting up the Docker application. So, you can use Docker Compose to create separate containers, host them and get them to communicate with each other. Each container will expose a port for communicating with other containers.

**--link** - allows you to link multiple containers together and send connection information from one to another.

Docker Compose is a YAML file(yet another markup language) which is used to run multiple containers as a single service.

docker run -d --name=redis redis

docker run -d --name=db postgres:9.4

docker run -d --name=vote -p 5000:80 --link redis:redis voting-app(--link name of the container:alias for the link name)

docker run -d --name=result -p 5001:80 --link db:db result-app

docker run -d --name=worker --link redis:redis --link db:db worker

docker-compose.yml

redis:

image:redis

db:

image:postgres:9.4

vote:

image:voting-app

ports:

* 5000:80

links:

- redis

result:

image:result-app

ports:

* 5001:80

links:

* db

worker:

image:worker

links:

* db
* redis

We can replace the image line with the build line and specify the location of the directory which contains the application code and a docker file with instructions to build the docker image

build: ./vote

services:

redis:

image: redis-alpine

clickcounter:

image: kodekloud/click-counter

ports:

- 8085:5000

version: '3.0'

docker-compose up -d

docker-compose down

**DOCKER REGISTRY**

Central repository of all docker images – docker hub

2 types of registry in docker – public registry(docker hub) & private registry

**DOCKER ENGINE**

Referred to a host with docker installed on it.

When we install docker engine we install 3 different components- Docker Deamon, REST API, Docker CLI.

A screenshot of a computer

Description automatically generated

Docker Deamon is a background process that manages Docker objects such as images, containers, volumes and networks.

The Docker REST API server is the API interface that programs can use to talk to the daemon and provide instructions.

Docker CLI is the command line interface that is used to perform actions such as running a container, stopping a container, destroying images etc. It uses the REST API to interact with the docker daemon

**DOCKER STORAGE**

/var/lib/docker

* aufs
* containers
* image
* volumes

**docker volume create data\_volume** – creates data\_volume directory under volumes directory

**docker run -v data\_volume:/var/lib/mysql mysql** – creates a volume data\_volume as well as mount it to the container /var/lib/mysql – **volume mounting**

**docker run -v /data/mysql:/var/lib/mysql mysql** – mounts a directory from any location on the docker host – **bind mounting**

the above command can also be written as below

**docker run \**

**--mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql**

**docker run -d --name=mysql-db -e MYSQL\_ROOT\_PASSWORD=db\_pass123 mysql**

**docker volume ls** – list all volumes

**DOCKER NETWORK**

the ability for containers to connect to and communicate with each other.

Bridge, none, host

Bridge – **docker run ubuntu** – default network a container gets attached to, It is used when multiple docker communicates with the same docker host.

none – **docker run Ubuntu --network=none** It disables all the networking./ Completely isolate a container from the host and other containers.

host – **docker run Ubuntu --network=host** It is used when we don't need for network isolation between the container and the host.

| **Driver** | **Description** |
| --- | --- |
| bridge | The default network driver. |
| host | Remove network isolation between the container and the Docker host. |
| none | Completely isolate a container from the host and other containers. |
| overlay | Overlay networks connect multiple Docker daemons together. |
| ipvlan | IPvlan networks provide full control over both IPv4 and IPv6 addressing. |
| macvlan | Assign a MAC address to a container. |

**docker network ls**

**docker inspect alpine-1 | grep -i network**

**docker network create --driver bridge --subnet 182.18.0.1/24 --gateway 182.18.0.1 wp-mysql-network** –creating a new network

**docker run -d -e MYSQL\_ROOT\_PASSWORD=db\_pass123 --name mysql-db --network wp-mysql-network mysql:5.6** – attaching network

Deploy a web application named webapp using the kodekloud/simple-webapp-mysql image. Expose the port to 38080 on the host.

The application makes use of two environment variable:

1: DB\_Host with the value mysql-db.

2: DB\_Password with the value db\_pass123.

Make sure to attach it to the newly created network called wp-mysql-network.

Also make sure to link the MySQL and the webapp container.

docker run --network=wp-mysql-network -e DB\_Host=mysql-db -e DB\_Password=db\_pass123 -p 38080:8080 --name webapp --link mysql-db:mysql-db -d kodekloud/simple-webapp-mysql

Install docker on Linux(rhel)

1. Install the yum-utils package (which provides the yum-config-manager utility) and set up the repository.

sudo yum install -y yum-utils

sudo yum-config-manager --add-repo <https://download.docker.com/linux/rhel/docker-ce.repo>

1. To install latest version - sudo yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
2. Start docker - sudo systemctl start docker

Post installation steps to use docker as non root user

1. sudo groupadd docker – create docker group
2. sudo usermod -aG docker $USER – add user to the docker group