# Docker

## Docker commands

docker version

docker info

docker images

docker ps

docker ps -a

docker stop <container id/name>

docker start <container id/name>

docker pause <container id/name>

docker unpause <container id/name>

docker kill <container id/name>

docker exec -it <container id/name> <shell name>

docker rm <container id/name>

docker rm -f <container id/name>

docker rmi <image id/name>

docker container prune

docker image prune

docker logs <container id/name>

docker top <container id/name>

docker stats <container id/name>

docker inspect <container id/name>

docker port

docker scan hello-world

docker scan --file Dockerfile docker-scan:e2e

docker scan --file Dockerfile --exclude-base docker-scan:e2e

docker scan --json hello-world

docker scan --dependency-tree debian:buster

docker scan --severity=medium docker-scan:e2e

docker build -t <image name>:<tag> <Dockerfile path>

docker run -itd

--name <container name>

-p "<new port>:<port>" <image>

-e MYSQL\_ROOT\_PASSWORD=<your passwd> <image name>

--link <container name>:mysql -p "<new port>:<port>" <image name>

-v "<location in container>" <image name>

-v "<local path>:<container path>" <image>

--volumes-from <cont name you want to get data from> <image name>

--mount source=<volume name>,destination=<container location> <image name>

--network <your network> <image name>

docker volume ls

docker volume create <volume name>

docker inspect <volume name>

docker login

docker logout

docker tag <new image name> <dockerid>/<name>

docker pull <image name>

docker push <dockerid>/<name>

docker commit <container id> <new image name>

docker save -o <path/filename.tar> <new image name>

docker load -i filename.tar

docker cp <src-path> <container>:<dest-path>

docker cp <container>:<src-path> <local-dest-path>

docker network ls’]

docker network create --driver bridge <new network name>

docker-compose up -d

docker-compose ps

## Whats is docker ?

* Platform as a service products that use OS-level virtualization to deliver software in packages called containers.
* Containers are isolated from one another and bundle their own software, libraries and configuration files
* they can communicate with each other through well-defined channels.

## Docker Container and VM – What is the difference ?

* containers provide a way to virtualize an OS so that multiple workloads can run on a single OS instance. They are microservices
* With VMs, the hardware is being virtualized to run multiple OS instances. They are monolithic application

## what is virtualisation ?

virtualisation is the act of creating a virtual version of something, including virtual computer hardware platforms, storage devices, and computer network resources.

## what is micro services ?

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are. Highly maintainable and testable. Loosely coupled. Independently deployable. Organized around business capabilities.

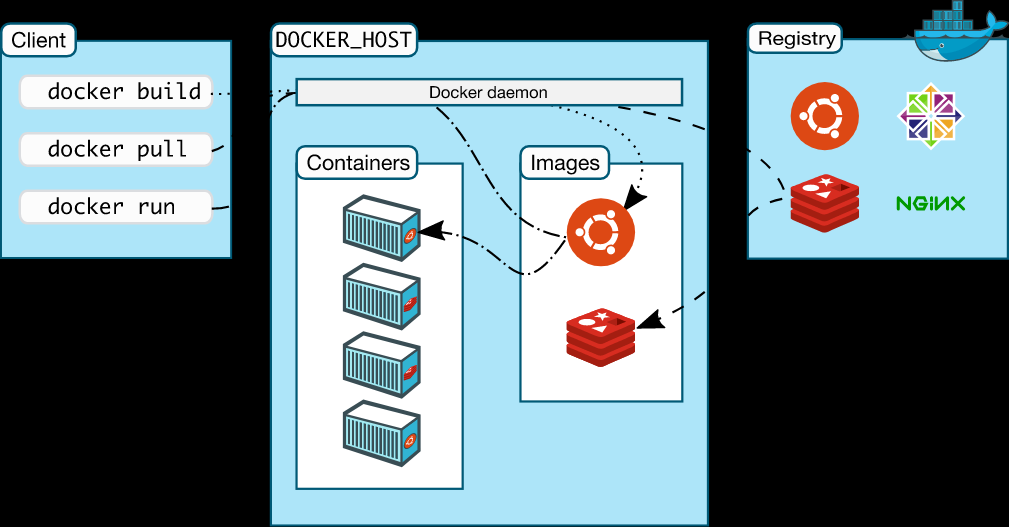
## Why should i Go Docker in my project ?

Docker containers are process-isolated and don't require a hardware hypervisor.

This means Docker containers are much smaller and require far fewer resources than a VM. Docker is fast. Very fast.

To convert monolith application to microservices we can Docker

## Docker Architecture ?



## What are registries ?

* A registry is a storage holding Docker images
* Docker images are available in different tagged versions in registry
* We interact with a registry by using docker push and pull commands.
* It is part of Docker Architecture

## What is Docker sock

docker group in linux that is being created by most docker installations. It’s purpose is to allow non-root users access to docker. You simply add a user to the docker-group using

gpasswd -a USER docker

and the USER can run docker commands freely. This works because the docker group is set to be the owner of the /var/run/docker.sock socket used for the communication with the daemon.

Be careful - this is very powerful and gives the user indirect access to the whole host machine since the docker daemon still runs as root!

## What is Docker File ?

* Dockerfile is a text file that contains a list of commands (instructions), which describes how a Docker image is built.
* The commands tells Docker to build the image by following the content (instructions) inside the Dockerfile.

## How to build DockerFile with different Name ?

$ docker build -f dockerfiles/Dockerfile.debug -t myapp\_debug .

$ docker build -f dockerfiles/Dockerfile.prod -t myapp\_prod .

## What is Docker Image ?

* A Docker image is a file used to execute code in a Docker container.
* Docker images act as a set of instructions to build a Docker container, like a template.
* An image is comparable to a AMI in AWS virtual machine (VM) environments.

## What is Docker Container ?

* Docker images are used to create a container.
* Containers are isolated from one another and bundle their own software, libraries and configuration files
* IT is a running application.
* They are mutable

## How to reduce the Docker image size ?

* Utilize the Multi-Stage Builds Feature in **Docker**.
* Use Small Base Image Like Apline Or Busybox
* Minimise Layers Used In Dockerfile
* Use Dockerignore Similar To Gitignore
* Don't Install Debug Tools Like Vim/Curl
* Use Docker Squash Tool or Docker compress while building the image
* Avoid Adding Unnecessary Layers to **Reduce Docker** Image **Size**.

## What is Docker Squash ?

* docker image build --squash
* will merge multiple layers in Docker file into a single layer/ Squash newly built layers into a single new layer

## What is Docker compress ?

* docker image build --compress
* Compress the build context using gzip

## What are image layers ?

A Docker image consists of several layers. Each layer corresponds to certain instructions in your Dockerfile . The following instructions create a layer: RUN , COPY , ADD . The other instructions will create intermediate layers and do not influence the size of your image.

## What is multi stage builds in docker ?

multi-stage builds, you use multiple FROM statements in your Dockerfile.

Each FROM instruction can use a different base, and each of them begins a new stage of the build.

You can selectively copy artifacts from one stage to another, leaving behind everything you don't want in the final image.

# syntax=docker/dockerfile:1

FROM golang:1.16

WORKDIR /go/src/github.com/alexellis/href-counter/

RUN go get -d -v golang.org/x/net/html

COPY app.go .

RUN CGO\_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest

RUN apk --no-cache add ca-certificates

WORKDIR /root/

COPY --from=0 /go/src/github.com/alexellis/href-counter/app .

CMD ["./app"]

## Difference between DockerFile vs DockerCompose ?

* Dockerfile is a text file that contains a list of commands (instructions), which describes how a Docker image is built.
* The commands tells Docker to build the image by following the content (instructions) inside the Dockerfile. **docker build -t <imagename>**
* Compose is a tool for defining and running multi-container Docker applications.
* With Compose, you use a YAML file to configure your application's services. with a single command, you create and start all the services from your configuration. **docker-compose up**

## Docker File Syntax ?

FROM

LABEL

WORKDIR

ENV

ARGS

ADD

COPY

RUN

ENTRY POINT

CMD

EXPOSE

## Difference between ADD vs Copy ?’

* Copy can send files from Local to Container
* Where are ADD can get data from url and can Extract tar files
* COPY is Same as 'ADD', but without the tar and remote URL handling.

## CMD vs EntryPoint What is the difference ?

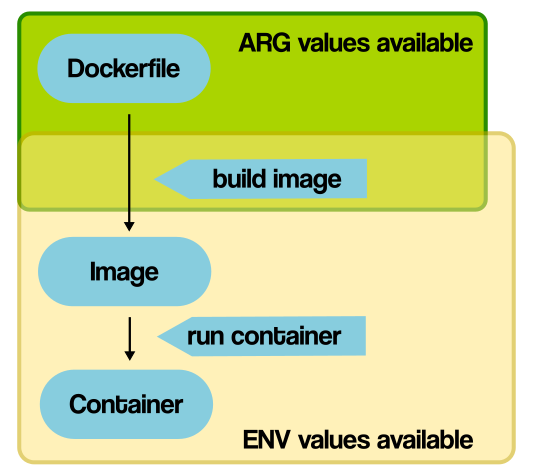
* ENTRYPOINT command and parameters will not be overwritten from command line.
* CMD sets default command for parameters, which can be overwritten from command line when docker container runs.
* All command line arguments will be added after ENTRYPOINT parameters.

## Difference between ENV and ARG in Docker file ?

ENV is for future running containers.ARG for building your Docker image.

You can’t change ENV directly during the build!

ARG values are not available after the image is built.



## What is Docker Build Context

* When we try to build a docker image, we need to send the files to the docker server.
* These files are basically the build context.
* These files are archived into a .tar file by the docker client and then they are uploaded to the docker server.

## What is .dockerignore file ?

* Dockerignore files allows you to mention a list of files and/or directories which you might want to ignore while building the image.
* This would definitely reduce the size of the image and also help to speed up the docker build process.

## To copy a file from the local file system to a container

docker cp <src-path> <container>:<dest-path>

kubectl cp <src-path> <your-pod-name>:<dest-path>

## To copy a file from the container to the local file system

docker cp <container>:<src-path> <local-dest-path>

kubectl cp <your-pod-name>:<src-path> <local-dest-path>

## Docker file we give expose and we give -p while running docker what is the difference ?

* We expose ports using the EXPOSE keyword in the Dockerfile or the --expose flag to docker run.
* Exposing ports is a way of documenting which ports are used, but does not actually map or open any ports.
* We publish ports using the --publish or --publish-all flag to docker run .

## how to find dead containers in docker ?

docker ps --filter "status=exited"

docker ps -f "status=exited"

## Difference between docker commands: up, run & start ?

* docker compose up is used to run a docker-compose.yaml file
* docker run is used to create a container
* docker start is used to start a stopped container

## Docker kill, docker pause ,Docker stop

* docker kill subcommand kills one or more containers (SIGKILL)
* docker stop will stop the container which can be resumed later (SIGTERM).
* docker stop release the memory used after the container is stopped.
* docker pause would still keep memory portion while the container is paused> This memory is used when the container is resumed.

## Why should i go Docker Volume ?

* Docker volumes are file systems mounted on Docker containers to preserve data generated by the running container.
* The volumes are stored on the host, independent of the container life cycle.
* This allows users to back up data and share file systems between containers easily.

## How to share docker data in multiple volume ?

* -v "<local path>:<container path>" <image>
* --volumes-from <cont name you want to get data from> <image name>
* --mount source=<volume name>,destination=<container location> <image name>

## Docker attach vs Docker exec?

* docker attach will leave the docker in exited status after coming out from the container
* docker exec will let the container run in background even after coming out from the container

### Docker detach ?

you can detach from a container and leave it running using the **CTRL-p CTRL-q** key sequence.

## how to push Docker image in Docker HUB ?

docker login

docker tag <new image name> <dockerid>/<name>

docker push <dockerid>/<name>

## How to add volumes to running container ?

commit the container into a new image

add the volume using docker run -v "<local path>:<container path>" <image>

## Can I limit the mem/ CPU utilization for a docker in my machine ?

docker run -it --memory=”[memory\_limit]” [docker\_image]

docker run -it --cpus=”1.0” ubuntu

## How to Link Docker Containers ?

--link is old legacy way of linking containers

now we can use docker networking to link between containers. Create a new network and create containers on them they can communicate between each other

## Docker link vs depends on ?

depends\_on expresses start order (and implicitly image pulling order), which was a good side effect of links.

--link is also deprecated and should be replaced by a custom network.

## How to communicate between 2 containers in a different network

* Solution a) Connect one container into the other network overlay (this may not meet the constraint you have).
* Solution b) Create a third network and plug both containers into this network

## what is difference between -v and mount

-v is used to mount a local dir to a dir inside container they will synced with each other

if we have a seprate volume created by docker volume command we can use mount command to mount the volume

## how you will confirm there is no security vulnerability in public image ?

create new snyk account which can scan for vulnerabilites

docker scan hello-world

docker scan --file Dockerfile docker-scan:e2e

docker scan --file Dockerfile --exclude-base docker-scan:e2e

docker scan --json hello-world

docker scan --dependency-tree debian:buster

docker scan --severity=medium docker-scan:e2e

## how to monitor docker container ?

Prometheus

## how will you handle or store your private images in your env

Private artifactory

Jfrog (needs lisence) & sonatype Nexus

Create new instance yum install java

Download Nexus and ./nexus start -> give initial admin password and create new password

Create respository -> Docker Hosted -> give name, http 8083, give Docker v1 API access

install docker -> create /etc/docker/deamon.json -> insecure registry give publicip:8083

also do the same in Docker machine where jenkins is running the docker and dp the same /etc/docker/deamon.json

## what is the difference between bridge network and custom bridge network

* User-defined bridges provide automatic DNS resolution between containers.
* User-defined bridges provide better isolation.
* Containers can be attached and detached from user-defined networks on the fly.
* Each user-defined network creates a configurable bridge.
* Linked containers on the default bridge network share environment variables.

## Where the image layes can be found in which directory ?

The /var/lib/docker/aufs directory points to three other directories: diff , layers and mnt .

Image layers and their contents are stored in the diff directory.

## What are the 3 different directories in /var/lib/docker/aufs ?

diff/ : Differences introduced in the writable container layer, such as new or modified files.

layers/ : Metadata about the writable container layer's parent layers.

mnt/ : A mount point for each running container's unified filesystem, exactly as it appears from within the container.

## What is overlayFS ?

* OverlayFS is a modern union filesystem that is similar to AUFS, but faster and with a simpler implementation.
* Docker provides two storage drivers for OverlayFS: the original overlay , and the newer and more stable overlay2.

## How can we check the content of each layer ?

/var/lib/docker/aufs/diff -

/var/lib/docker/aufs/layers -

## How to check the layers stacked with image ?

/var/lib/docker/aufs/layers -

## What is Union Mount & AUFS ?

AUFS is a union filesystem, which means that it layers multiple directories on a single Linux host and presents them as a single directory. These directories are called branches in AUFS terminology, and layers in Docker terminology. The unification process is referred to as a union mount.

## Why use Union mount system for Docker ?

avoid duplicating a complete set of files each time you run an image as a new container

isolate changes to a container filesystem in its own layer, allowing for that same container to be restarted from a known content (since the layer with the changes will have been dismissed when the container is removed)

If you didn't have UnionFS, an 200MB image run 5 times as 5 separates containers would mean 1GB of disk space.

## How to create a bridge in container ?

docker network create --driver bridge <new network name>

docker run -itd --network <your network> <image name>

## How a container gets an internal IP ?

By default, the container is assigned an IP address for every Docker network it connects to. And each network is created with a default subnet mask, using it as a pool later on to give away the IP addresses. Usually Docker uses the default 172.17. 0.0/16 subnet for container networking.

## How kernel isolates to run the container and how resources managed by the kernel ?

Docker makes use of kernel **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.

## What is namespace and cgroups ?

cgroups limits the resources which a process or set of processes can use these resources could be CPU,Memory,Network I/O or access to filesystem

namespace restrict the visibility of group of processes to the rest of the system.

## What is docker-compose and docker-swarm ?

Docker Swarm is used to scale your web app across one or more servers

Where as Docker-compose will simply run your web app on a single Docker host.

## Can we run more than one process in a container ?

It's ok to have multiple processes, but to get the most benefit out of Docker, avoid one container being responsible for multiple aspects of your overall application. You can connect multiple containers using user-defined networks and shared volumes.