Why and How Developers Should Use Docker?

A Practical Guide to Docker
By Sandip Das

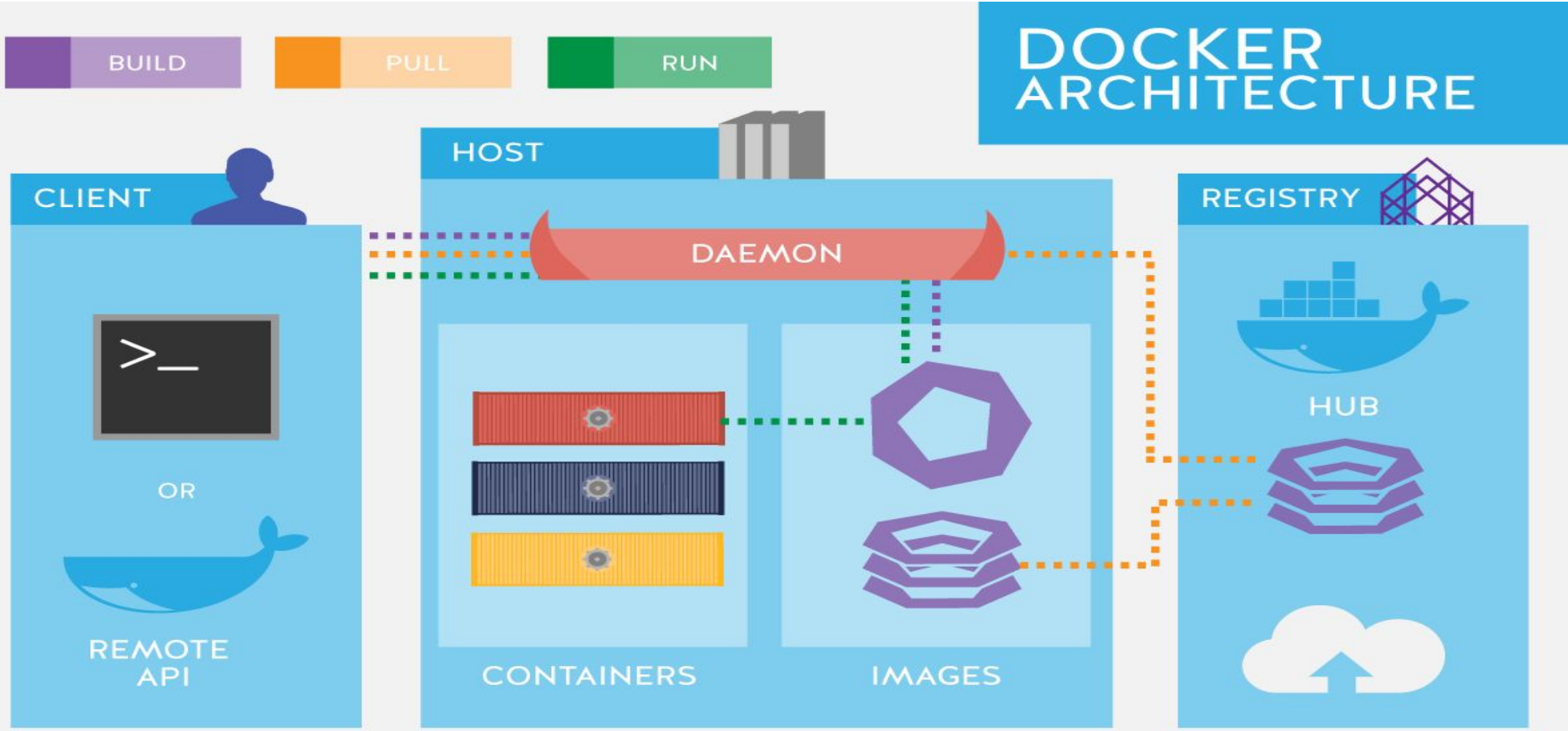
Contents

What is Docker ?

Docker is a tool designed to make it easier to create, 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. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

From: Opensource.com

Docker Architecture



Why Developers should use Docker?

Docker makes developer's life a lot easier. Below are the reasons why the Developer should use Docker:

1. Development in a cross-platform environment is easier. You can package your application and all your dependency once and run anywhere, it's an awesome feature and gets rid of this statement "It's not my fault, it works on my computer!"

2. In Docker, all dependencies are already built into the image itself. When you generate the image at moment all dependencies got installed inside the docker image itself so when you start a container using a docker, the container will start very fast along with the application.

3. The distribution of your application becomes easier. Using Docker Hub and similar Docker Registry services (e.g. AWS ECR, JFrog Artifactory, Azure CR, Google Cloud GCR, etc) you can share your application image with anyone.

4. Application Deployment is easier. Normally for application deployment, you would have to install all the application dependencies, set-up environments, install libraries and what not! But using Docker image since you have already done it, so you don't have to repeat yourself anymore, by using the image you can deploy your application in all docker image deployment platforms.

How To Install Docker?

1) Install Docker in Mac:: https://docs.docker.com/docker-for-mac/install/
2) Install Docker in Windows:: https://docs.docker.com/v17.12/docker-for-windows/install/
3) Install Docker in Ubuntu: https://docs.docker.com/v17.12/install/linux/docker-ce/ubuntu/
4) Install Docker in Debian: https://docs.docker.com/v17.12/install/linux/docker-ce/debian/
5) Install Docker in CentOs: https://docs.docker.com/v17.12/install/linux/docker-ce/centos/
6) Or from Binaries: https://docs.docker.com/v17.12/install/linux/docker-ce/binaries/

Necessary Docker Commands (part 1)

Running New Container

Start new container from image:
docker run IMAGE_NAME
E.g. docker run nginx

Change the Entry Point:
docker run -it --entrypoint EXECUTABLE IMAGE_NME
E.g. docker run -it --entrypoint bash nginx

Assign name to container:
docker run --name CONTAINER_NAME IMAGE_NAME
E.g. docker run --name web_test nginx

Map Container port with Host port:
docker run -p HOST_PORT:CONTAINER_PORT IMAGE_NAME
E.g. docker run -p 8080:80 nginx

Map all ports:
docker run -P IMAGE_NAME
E.g. docker run -P NGINX

Start Container in Background:
docker run -d IMAGE_NAME
E.g. Docker run -d nginx

Managing Containers

List of Running containers:
docker ps

List of all containers:
docker ps -a

NOTE: Here CONTAINER_NAME can be any kind of identifier e.g. short id or the long id string as well the name you have specified

Stop a Running Container:
docker stop CONTAINER_NAME
E.g. docker stop web_test

Start a Stopped Container:
docker start CONTAINER_NAME
E.g. docker start web_test

Delete a container:
docker rm CONTAINER_NAME
E.g. docker rm web_test

Delete a Running Container:
docker rm -f CONTAINER_NAME
E.g. docker rm -f web_test

Delete Stopped Containers:
docker container prune

Manage Images

Download a image:
docker pull IMAGE_NAME:TAG_NAME
E.g. docker pull nginx:latest

Upload an Image to repository:
docker push IMAGE_NAME:TAG_NAME
E.g. docker push sample_image:1.0

Show List of Images:
docker images

Tag a image:
docker tag IMAGE_NAME NEW_IMAGE_NAME:TAG_NAME
e.g. docker tag ubuntu ubuntu:18.04

Build a Image from Dockerfile:
docker build DIRECTORY_PTH
E.g. docker build .

Build and tag image from Dockerfile
docker build -t IMAGE_NAME DIRECATORY_PATH
E.g. docker build -t my_test_image .
(here “.” means current directory)

Info & Stats

Show logs of a container:
docker logs CONTAINER_NAME
E.g. docker logs web_test

Show stats of running container:
docker stats

Show Processes of container:
docker top CONTAINER_NAME
E.g. docker top web_test

Show Installed Docker Version:
E.g. docker version

Get detailed info about an Object
docker inspect name
E,g, docker inspect nginx

Show all modified files in container
docker differ CONTAINER_NAME
E.g. docker diff web_test

Show mapped port of a container
docker port CONTAINER_NAME
docker port web_test

Necessary Docker Commands (part 2)

Running New Container

Assign container to host name
docker run --hostname HOST_NAME_HERE IMAGE_NAME
.e.g. docker run --hostname srv nginx

Add DNS Entry:
docker run -add-host HOST_NAME:IP IMAGE_NAME

Map Local Directory into the container
docker run -v HOST_DIRECTORY:TARGET_DIRECTORY IMAGE_NAME
E.g.
docker run -d -p 27017:27017 -v ~/mongodb/data/db:/data/db mongo

Managing Containers

Copy a file from container to host:
docker cp CONATINER_NAME:SOURCE HOST_TARGET
E.g docker cp web_test:/index.html ./index.html

Copy a file from host to container:
docker cp TARGET_FILE CONTAINER_NAME:SOURCE
E.g: docker cp index.html web_test:/index.html

Start a Shell inside a container:
docker exec -it CONTAINER_NAME EXECUTABLE
E.g. docker exec -it web_test bash

Rename a container:
docker rename OLD_NAME NEW_NAME
E.g. docker rename web_test web_test_1

Create a image out of Container:
docker commit CONTAINER
E.g. docker commit web_test

Manage Images

Save a image to .tar file
docker save IMAGE_NAME > TAR_FILE_NAME
E.g. docker save nginx > nginx.tar

Load a image from tar file:
docker load -i TARFILE
E.g. docker load -i nginx.tar

Delete an image:
docker rmi IMAGE_NAME
E.g. docker rmi nginx

Delete Dangling images:
docker image prune

Delete all unused images:
docker image prune -a

Info & Stats

How to build image using Dockerfile & Best Practices?

Know the Commands

FROM: Initializes a new build stage and sets the Base Image

RUN: Will execute any commands in a new layer

CMD: Provides a default for an executing container. There can only be one CMD instruction in a Dockerfile

LABEL: Adds metadata to an image

EXPOSE: Informs Docker that the container listens on the specified network ports at runtime

ENV: Sets the environment variable <key> to the value <value>

ADD: Copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

COPY: Copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.

ENTRYPOINT: Allows for configuring a container that will run as an executable

VOLUME: Creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers

USER: Sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any RUN, CMD, and ENTRYPOINT instructions that follow it in the Dockerfile

WORKDIR: Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions that follow it in the Dockerfile

ARG: Defines a variable that users can pass at build-time to the builder with the docker build command, using the --build-arg <varname>=<value> flag

ONBUILD: Adds a trigger instruction to the image that will be executed at a later time, when the image is used as the base for another build

HEALTHCHECK: Tells Docker how to test a container to check that it is still working

SHELL: Allows the default shell used for the shell form of commands to be overridden

Sample Dockerfile

```
FROM node:8
WORKDIR /usr/src/app
ADD . /usr/src/app
RUN yarn
RUN yarn build
EXPOSE 4040
CMD ["yarn", "serve"]
```

This is from popular MEAN stack git repo:
https://github.com/linnovate/mean

To build image go inside the the project folder and run:
docker image build -t mean_demo:v1 .
To check working fine or not:

Start mongoDB:
docker run -d -p 27017:27017 -v ~/mongodb/data/db:/data/db mongo

Run the application:
docker container run --name mean_demo -p 4040:4040 mean_demo:v1
Now check the url: your browser **localhost:4040**

Best Practices:

Follow Principle of the 12 Factor App.
Avoid including unnecessary files.
Use .dockerignore.
Use multi-stage builds.
Don't install unnecessary packages.
Decouple applications.
Minimize the number of layers.
Sort multi-line arguments.
Leverage build cache.

How to run container using image and what more you can do with it?

Previously we just build a image called mean_demo:v1 , you can run it by just executing:

docker container run --name mean_demo -p 80:4040 mean_demo:v1
or
suppose you want to host a https site you should run it on port 443 as below:

docker container run --name mean_demo -p 443:4040 mean_demo:v1 but when you run it make sure you run it in detached mode like below:

docker container run -d --name mean_demo -p 443:4040 mean_demo:v1

But this image capability is not limited to that, you can push this image to any docker registry like Docker Hub, Amazon ECR , Google Container Registry, Microsoft Azure Container Registry and more such options available.

Of course you can always start an instance/machine in your choice of cloud , then install docker, add rules to allow port in security group and then run the image in container or if you are managing your own Kubernetes Cluster you can use the image to run application but you must know that some excellent services offered by all major cloud service providers and you can utilize the docker image:

1) In Amazon Web Services(AWS): AWS ECS, AWS EKS, AWS Fargate, AWS Batch
2) In Google Cloud Platform(GCP): Google Kubernetes Engine(GKE), Google Cloud Run
3) In Microsoft Azure: Azure Kubernetes Service, Container Instances, Service Fabric, Web App For Containers

Expect me to cover many such topic in upcoming videos on my Youtube Channel

Docker Volumes and do you really need it ? how to utilize it?

Know about Docker Volumes

By default docker store data in non-persistent way, that means if you remove the container then the data will also get removed so it's ephemeral.
If use persistent volumes then even if the container removed the data will not get removed:

Some facts:
Non-persistent Data (use it when you don't need container data)

- By default all container use local storage
- Deleting a container delete all data inside container
- Storage locations:
  - Linux: /var/lib/docker/[STORAGE-DRIVER]/
  - Windows: C:\ProgramData\Docker\windowsfilter\
- Storage Drivers:
  - RHEL uses overlay2.
  - Ubuntu uses overlay2 or aufs.
  - SUSE uses btrfs.
  - Windows uses its own.

Persistent Data Using Volumes (use it when you want to keep container data for future use e.g. database containers)

- Use a volume for persistent data:
  - Create the volume first, then create your container.
- Mounted to a directory in the container
- Data is written to the volume
- Deleting a container does not delete the volume
- First-class citizens
- Uses the local driver
- Third party drivers:
  - Block storage (e.g. AWS EBS, OpenStack Cinder)
  - File storage (e.g. Netapp FAS, Azure File Storage, AWS EFS,)
  - Object storage (AWS S3, Ceph, MinIO, OpenStack Swift)
- Storage locations:
  - Linux: /var/lib/docker/volumes/
  - Windows: C:\ProgramData\Docker\volumes

Volume Commands

Volumes are preferred by everyone for maintaining persistent data in Docker , below are the basic commands:

List all Docker volume commands:
docker volume -h

Options:
create: Create a volume.
inspect: Display detailed information on one or more volumes.
ls: List volumes.
prune: Remove all unused local volumes.
rm: Remove one or more volumes.

Examples below:

List all volumes on a host:
docker volume ls

Create new volumes:
docker volume create test-volume1

Get the flags available when creating a volume:
docker volume create -h

Inspecting a volume:
docker volume inspect test-volume1

Deleting a volume:
docker volume rm test-volume1

Removing all unused volumes:
docker volume prune

How to Utilize it?

E.g.
Create a new volume for an Nginx container:
docker volume create html-volume

Creating a volume using that volume mount:
```
docker container run -d \
 --name nginx-volume1 \
 --mount
type=volume,source=html-volume,target
=/usr/share/nginx/html/ nginx
```

Inspect the volume:
docker volume inspect html-volume

Creating a container using that volume:
docker run -v html-volume:/html-volume -it ubuntu:16.04 bash

If you edit index.html file you can see the changes immediately

How to create container and set volume as read only:

```
docker run -d \
  --name=nginx-volume3 \
  --mount
source=html-volume,target=/usr/share/n
ginx/html,readonly \
  nginx
```

Docker Networking and how to actually use it ?

Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

- bridge
- host
- overlay
- macvlan
- none
- Network plugins

Network driver summary

- **User-defined bridge networks** are best when you need multiple containers to communicate on the same Docker host.
- **Host networks** are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.
- **Overlay networks** are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.
- **Macvlan networks** are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.
- **Third-party network plugins** allow you to integrate Docker with specialized network stacks.

Docker Network Commands:

List all Docker networks on the host:
docker network ls
docker network ls --no-trunc

Creating a network:
docker network create NAME

Getting detailed info on a network:
docker network inspect NAME

Deleting a network:
docker network rm NAME

Remove all unused networks:
docker network prune

Adding and Removing containers to a network

Create a container with no network:
docker container run -d --name network-test03 -p 8081:80 nginx

Create a new network:
docker network create br01

Add the container to the bridge network:
docker network connect br01 network-test03

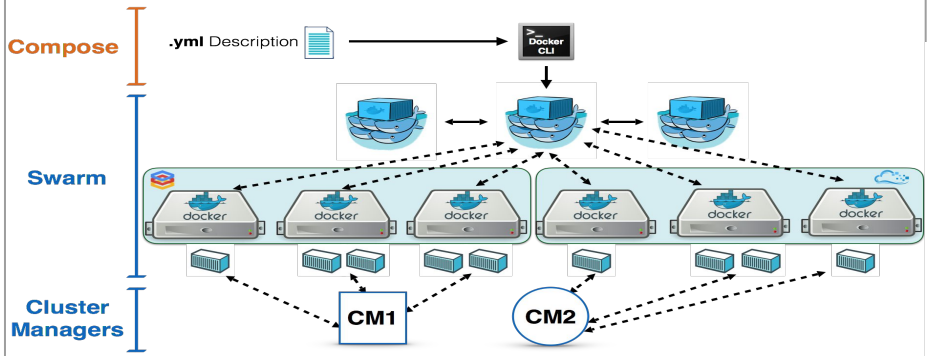
Inspect network-test03 to see the networks:
docker container inspect network-test03

Remove network-test03 from br01:
docker network disconnect br01 network-test03

Know about Docker swarm and Kubernetes (overview)

Docker Swarm

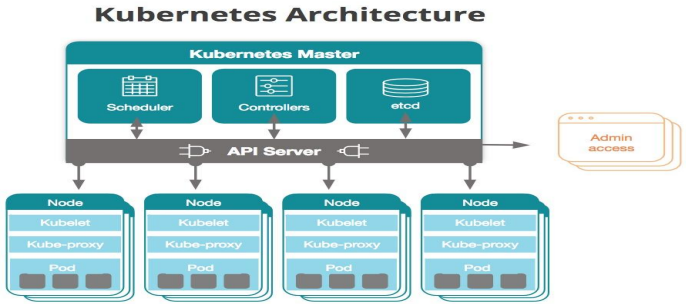
As a platform, Docker has revolutionized the manner software was packaged. Docker Swarm or simply Swarm is an open-source container orchestration platform and is the native clustering engine for and by Docker. Any software, services, or tools that run with Docker containers run equally well in Swarm. Also, Swarm utilizes the same command line from Docker.
Swarm turns a pool of Docker hosts into a virtual, single host. Swarm is especially useful for people who are trying to get comfortable with an orchestrated environment or who need to adhere to a simple deployment technique but also have more just one cloud environment or one particular platform to run this on.



Kubernetes

Kubernetes is an open-source platform created by Google for container deployment operations, scaling up and down, and automation across the clusters of hosts. This production-ready, enterprise-grade, self-healing (auto-scaling, auto-replication, auto-restart, auto-placement) platform is modular, and so it can be utilized for any architecture deployment.

Kubernetes also distributes the load amongst containers. It aims to relieve the tools and components from the problem faced due to running applications in private and public clouds by placing the containers into groups and naming them as logical units. Their power lies in easy scaling, environment agnostic portability, and flexible growth.



Check This [Docker Swarm vs Kubernetes] Article

Few More things you must learn

It's all fine till you made your image, but things never stop there, application code will have bugs, or there will be new features, so you will have to Deploy application again and again. You should be using popular container-related services offered by popular Cloud Services e.g. AWS, GCP or Azure. To learn about those cloud platforms as well, learn more about what services they are offering & cost structure and which one suits your business needs. Below are the Courses I suggest anyone who is interested in Docker, Kubernetes or DevOps in General: https://gist.github.com/sd031/f5f93cbef59361c7f6731512f27a600b

Other than that there is always something new in tech world and while very hard to keep up, there are certain personalities working hard and making our life easier, our India docker contain and Tip of Captain's Hat Award Winner 2019 , **Mr Ajeet Singh Raina,** he have started a site and github repo for Docker & Kubernetes learners / practitioners as follows:http://dockerlabs.collabnix.com/ and GitHub Link: https://github.com/Collabnix/dockerlabs , must check it out.

I will be uploading many more videos in My YouTube Channel related to Docker, Kubernetes and other Cloud services, stay tuned , if you want any specific topic need to be covered, reach out to me.

Good luck!

I hope you'll use this knowledge and build awesome solutions.

If any issue contact me in Linkedin:
https://www.linkedin.com/in/sandip-das-developer/

YouTube Channel Link and QR Code below:

https://www.youtube.com/c/SandipDas-official