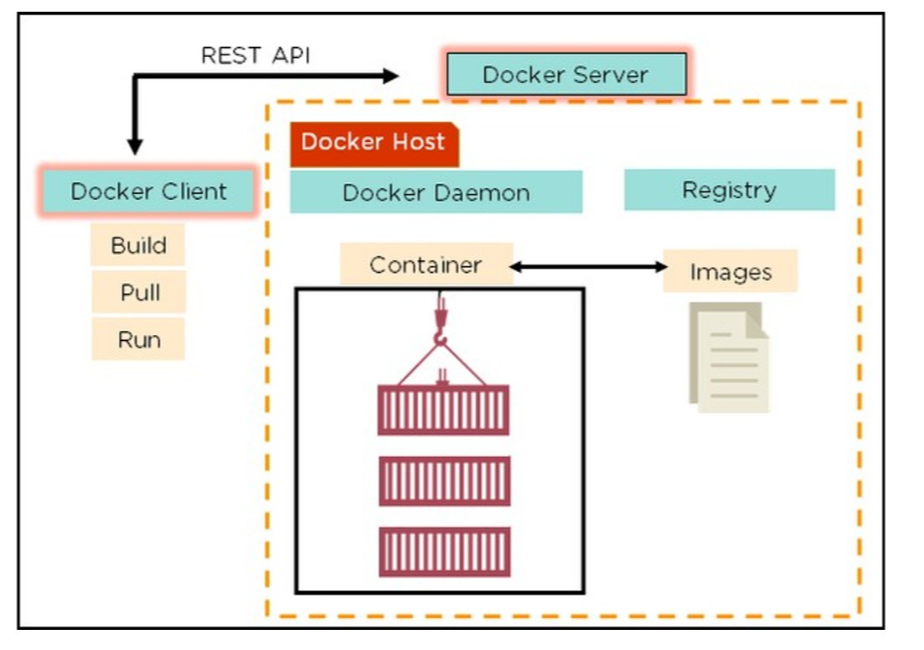
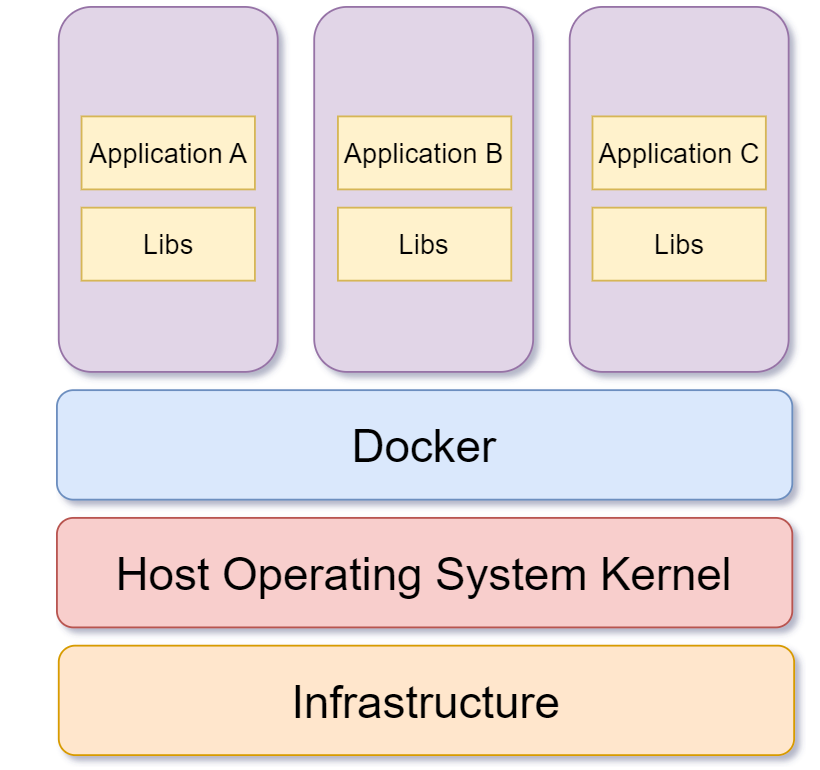
**DOCKER**

**Architecture of Docker:**

****

* Docker uses a client-server architecture.
* Docker Client is a service which runs a command.
* The command is translated using REST API and is sent to the Docker Daemon (server).
* Docker Daemon accepts the request and interacts with the operating system in order to build Docker Images and run Docker containers.
* A Docker Image is a template of instruction which is used to create containers.
* Docker Architecture consists of a Docker Engine which is a client-server application with three major components:
  + A server which is a type of long-running program called a daemon process (the docker command).
  + A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
  + A command line interface (CLI) client (the docker command).

The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. 

**Docker image:**

* Docker image is the source of Docker container. In other words, Docker images are used to create containers. Images are created with the build command, and they will produce a container when started with run.
* Images are stored in a Docker registry such as http://registry.hub.docker.com because they can become quite large, images are designed to be composed of layers of other images, allowing a minimal amount of data to be sent when transferring images over the network.

**Docker container:**

* Docker containers include the application and all of its dependencies, but share the kernel with other containers, running as isolated processes in user space on the host operating system.
* Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud. Now explain how to create a Docker container.
* Docker containers can be created by either creating a Docker image and then running it or you can use Docker images that are present on the Docker hub.
* Docker containers are basically runtime instances of Docker images.

**Docker hub:**

* Docker hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker cloud so you can deploy images to your hosts.
* Docker images create docker containers. There must be a registry where these docker images live. This registry is Docker Hub.
* Users can pick up images from Docker Hub and use them to create customized images and containers. Currently, the Docker Hub is the world’s largest public repository of image containers.

**Docker file:**

* Docker can build images automatically by reading the instructions from a file called Docker file.
* A Docker file 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.

The Docker file provides the instructions to build a container image through the `docker build -t [username/][:tag] ` command.

It starts from a previously existing Base image (through the FROM clause) followed by any other needed Docker file instructions.

**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.

**Life cycle of a Docker Container:**

* Create a container
* Run the container
* Pause the container(optional)
* Un-pause the container(optional)
* Start the container
* Stop the container
* Restart the container
* Kill the container
* Destroy the container

**Installing docker on Ubuntu Linux:**

Docker is supported on the following 64-bit versions of Ubuntu Linux:

• Ubuntu Xenial 16.04 (LTS)

• Ubuntu Wily 15.10

• Ubuntu Trusty 14.04 (LTS)

• Ubuntu Precise 12.04 (LTS)

Prerequisites:

* Docker only works on a 64-bit installation of Linux.
* Docker requires Linux kernel version 3.10 or higher (Except for Ubuntu Precise 12.04, which requires version 3.13 or higher).

Update APT sources:

This needs to be done to access packages from Docker repository.

1. Log into your machine as a user with sudo or root privileges.

2. Open a terminal window. Update package information, ensure that APT works with the https method, and that CA certificates are installed.

First, update your existing list of packages:

1. sudo apt update

Next, install a few prerequisite packages which let apt use packages over HTTPS:

1. sudo apt install apt-transport-https ca-certificates curl software-properties-common

Then add the GPG key for the official Docker repository to your system:

1. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Add the Docker repository to APT sources:

1. sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"

This will also update our package database with the Docker packages from the newly added repo.

Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:

1. apt-cache policy docker-ce

Finally, install Docker:

1. sudo apt install docker-ce

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it’s running:

1. sudo systemctl status docker

The output should be similar to the following, showing that the service is active and running:

Output

● docker.service - Docker Application Container Engine

Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)

Active: active (running) since Tue 2020-05-19 17:00:41 UTC; 17s ago

TriggeredBy: ● docker.socket

Docs: https://docs.docker.com

Main PID: 24321 (dockerd)

Tasks: 8

Memory: 46.4M

CGroup: /system.slice/docker.service

└─24321 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Installing Docker now gives you not just the Docker service (daemon) but also the docker command line utility, or the Docker client.

**DOCKERFILE Instructions arguments:**

COMMAND DESCRIPTION

|  |  |
| --- | --- |
| FROM | Sets the base image for subsequent |
| MAINTAINER | Sets the author field of the generated images |
| RUN | Execute commands in a new layer on top of the current image and commit the results |
| CMD | Allowed only once (if many then last one takes effect) |
| LABEL | Adds metadata to an image |
| EXPOSE | Informs container runtime that the container listens on the specified network ports at runtime |
| ENV | Set an environment variable |
| ADD | Copy new files, directories, or remote file urls from into the filesystem of the container |
| COPY | Copy new files or directories into the filesystem of the container |
| ENTROYPOINT | Allows you to configure a container that will run as an executable |
| VOLUME | Creates a mount point and marks it as holding externally mounted volumes from native host or other |
| USER | Sets the username or UID to use when running the image |
| WORKD IR | Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD commands |
| ARG | Defines a variable that users can pass at build-time to the builder using --build-arg |
| ONBUILD | Adds an instruction to be executed later, when the image is used as the base for another build |
| STOP SIGNAL | Sets the system call signal that will be sent to the container to exit |

The following steps explain how you should go about creating a Docker File.

**Step 1** − Create a file called **Docker File** and edit it using **vim**. Please note that the name of the file must be "Docker file" with "D" as capital.



**Step 2** − Build your Docker File using the following instructions.

#This is a sample Image

FROM ubuntu

MAINTAINER sanjaysanajy80609@gmail.com

RUN apt-get update

RUN apt-get install –y nginx

CMD [“echo”,” Image created”]

**DOCKER-COMPOSE FILE:**

Basic example:

# docker-compose.yml

version: '2'

services:

web:

build:

# build from Dockerfile

context: ./Path

dockerfile: Dockerfile

ports:

- "5000:5000"

volumes:

- .:/code

redis:

image: redis

**Command to execute docker-compose file:**

* docker-compose start
* docker-compose stop
* docker-compose pause
* docker-compose unpause
* docker-compose ps
* docker-compose up
* docker-compose down

**Reference:**

**Building:**

web:

# build from Dockerfile

build: .

args: # Add build arguments

APP\_HOME: app

# build from custom Dockerfile

build:

context: ./dir

dockerfile: Dockerfile.dev

# build from image

image: ubuntu

image: ubuntu:14.04

image: tutum/influxdb

image: example-registry:4000/postgresql

image: a4bc65fd

**Port:**

ports:

- "3000"

- "8000:80" # host:container

# Expose ports to linked services (not to host)

expose: ["3000"]

**Commands:**

# command to execute

command: bundle exec thin -p 3000

command: [bundle, exec, thin, -p, 3000]

# override the entrypoint

entrypoint: /app/start.sh

entrypoint: [php, -d, vendor/bin/phpunit]

**Environment variables:**

# environment vars

environment:

RACK\_ENV: development

environment:

- RACK\_ENV=development

# environment vars from file

env\_file: .env

env\_file: [.env, .development.env]

**Dependencies:**

# makes the `db` service available as the hostname `database`

# (implies depends\_on)

links:

- db:database

- redis

# make sure `db` is alive before starting

depends\_on:

- db

# make sure `db` is healty before starting

# and db-init completed without failure

depends\_on:

db:

condition: service\_healthy

db-init:

condition: service\_completed\_successfully

**Volumes:**

# mount host paths or named volumes, specified as sub-options to a service

db:

image: postgres:latest

volumes:

- "/var/run/postgres/postgres.sock:/var/run/postgres/postgres.sock"

- "dbdata:/var/lib/postgresql/data"

volumes:

dbdata:

**Health check:**

# declare service healthy when `test` command succeed

Health check:

test: ["CMD", "curl", "-f", "http://localhost"]

interval: 1m30s

timeout: 10s

retries: 3

start\_period: 40s

**Network:**

# creates a custom network called `frontend`

networks:

frontend:

**External network:**

# Join a pre-existing network

networks:

default:

external:

name: frontend

**OVERALL COMMANDS FOR DOCKER:**

* List all Docker Images
  + docker images -a
* List All Running Docker Containers
  + docker ps
* List All Docker Containers
  + docker ps -a
* Start a Docker Container
  + docker start
* Stop a Docker Container
  + docker stop
* Kill All Running Containers
  + docker kill $(docker ps -q)
* View the logs of a Running Docker Container
  + docker logs
* Delete All Stopped Docker Containers
* Use -f option to nuke the running containers too.
* docker rm $(docker ps -a -q)
* Remove a Docker Image
  + docker rmi
* Delete All Docker Images
  + docker rmi $(docker images -q)
* Delete All Untagged (dangling) Docker Images
  + docker rmi $(docker images -q -f dangling=true)
* Delete All Images
* docker rmi $(docker images -q)
* Remove Dangling Volumes
* docker volume rm -f $(docker volume ls -f dangling=true -q)
* SSH Into a Running Docker Container Okay not technically SSH, but this will give you a bash shell in the container.
  + sudo docker exec -it bash
* Use Docker Compose to Build Containers Run from directory of your docker-compose.yml file.
  + docker-compose build
* Use Docker Compose to Start a Group of Containers
  + Use this command from directory of your docker-compose.yml file.
  + docker-compose up -d
  + This will tell Docker to fetch the latest version of the container from the repo, and not use the local cache.
  + docker-compose up -d --force-recreate
  + #stop docker containers, and rebuild
  + docker-compose stop -t
  + docker-compose rm -f
  + docker-compose pull
  + docker-compose build
  + docker-compose up -d
* Follow the Logs of Running Docker Containers with Docker Compose
  + docker-compose logs -f
* Save a Running Docker Container as an Image
  + docker commit <image name><name for the image>
* Show Running Containers
  + docker ps
* Show All Containers - Running and stopped
  + docker ps -a
* Build a docker image From the directory of the Docker file run:
  + docker build -t
* Stop a docker container
  + - docker kill or docker stop
* Parameter that tells docker to run the container as a background process -d
  + Example:
  + docker run -d
* List all docker images on your system
  + docker images
* Map a Host Port to a Container Port -p :
  + Example:
  + docker run -p 8080:8080
* Remove a Docker Image from your System
  + docker rmi
* Shell into a Running Docker Container
  + docker exec -it bash