Docker

Docker is the world's leading **software container platform**. Docker makes the process of application deployment very easy and efficient and resolves a lot of issues related to deploying applications.

Docker is a tool designed to make it easier to deploy and run applications by using containers.

Docker gives you a standard way of packaging your application with all its dependencies in a container
Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Difference between Virtualization and Containerization

- **Resource usage:** Virtualization requires more resources than containerization because each virtual machine runs a complete operating system, while containers share the host operating system.
- **Isolation:** Virtualization provides complete isolation between virtual machines, while containerization provides isolation at the application level.
- **Performance**: Containers typically perform better than virtual machines because they are lighter weight and faster to start up.
- **Portability:** Containers are generally more portable than virtual machines because they can be easily moved between different environments without having to worry about the underlying hardware.

Docker Architecture

Docker uses a client-server architecture, which allows the user to interact with the Docker engine through a command-line interface (CLI) or a REST API.

The Docker client is the interface that the user interacts with. It is a command-line interface that allows the user to build, manage, and deploy Docker containers. The client sends commands to the Docker daemon (server), which is responsible for managing Docker objects such as images, containers, networks, and volumes.

The Docker daemon is the background process that manages the Docker objects. It listens for API requests from the Docker client and performs the requested actions such as creating, starting, and stopping containers, building images, and managing network connections.

When the Docker client sends a command to the Docker daemon, it first checks to see if the command can be performed locally on the client. If not, the command is sent to the Docker daemon for processing. The Docker daemon then responds with the result of the command, which is displayed on the client.

The Docker client and daemon can run on the same machine, or they can run on separate machines. In a typical setup, the Docker daemon runs on a remote server, and the Docker client is installed on the user's local machine. The user can then connect to the remote Docker daemon using the Docker client, which sends commands to the remote server for processing.

Benefits of using Docker

Build app only once.

No worries that the application will not perform the same way it did on testing env.
Portability

Version Control

Isolation

Productivity

Docker simplifies DevOps.

Commands

docker info

docker run hello-world : to run hello-world image

docker images //to get list of images present locally

docker ps : to get list of running containers

docker ps -a . : to get list of all containers

Basic

- docker version
- docker -v
- docker info
- docker --help
- docker login

Images

- docker images
- docker pull
- docker rmi

Containers

- docker ps
- docker run
- docker start
- docker stop

System

- docker stats
- docker system df
- docker system prune

What are Images

Docker Images are templates used to create Docker containers.

Container is a running instance of image.

Where are Images Stored?

Registries (e.g., docker hub) Can be stored locally or remote.

docker images --help

docker pull image.

docker images

docker images -q

docker images -f “dangling=false”

docker images -f “dangling=false” -q

docker run image.

docker rmi image.

docker rmi -f image :

docker inspect :

docker history imageName

What are Containers:

Containers are running instances of Docker Images.

COMMANDS

- docker ps
- docker run ImageName
- docker start ContainerName/ID
- docker stop ContainerName/ID
- docker pause ContainerName/ID
- docker unpause ContainerName/ID
- docker top ContainerName/ID
- docker stats ContainerName/ID
- docker attach ContainerName/ID
- docker kill ContainerName/ID
- docker rm ContainerName/ID
- docker history ImageName/ID

Example

How to set Jenkins home on Docker Volume and Host Machine :

docker pull jenkins :

docker run -p 8080:8080 -p 50000:50000 jenkins :

docker run –name MyJenkins -p 8080:8080 -p 50000:50000 -v
/Users/raghav/Desktop/Jenkins_Home:/var/jenkins_home jenkins :

docker run –name MyJenkins2 -p 9090:8080 -p 50000:50000 -v
/Users/raghav/Desktop/Jenkins_Home:/var/jenkins_home jenkins :

docker volume create myjenkins :

docker volume ls :

docker volume inspect myjenkins :

docker run –name MyJenkins3 -p 9090:8080 -p 50000:50000 -v myjenkins:/var/jenkins_home jenkins :

docker inspect MyJenkins3

In case you face issues like installing plugins on this Jenkins, can setup jenkins with this command:

$ docker run -u root –rm -p 8080:8080 -v /srv/jenkins-data:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock –name jenkins jenkinsci/blueocean

Dockerfile

A text file with instructions to build image Automation of Docker Image Creation

FROM RUN CMD

Step 1: Create a file named Dockerfile.

Step 2 : Add instructions in Dockerfile.

Step 3 : Build dockerfile to create image

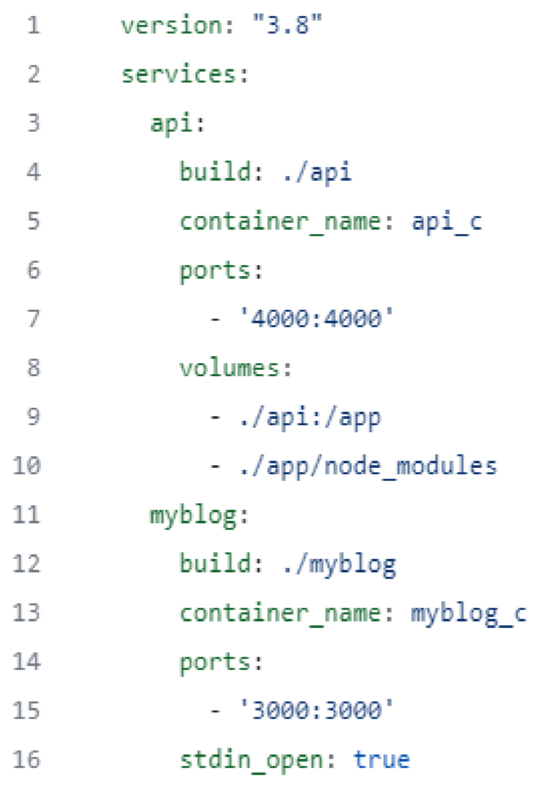
Step 4 : Run image to create container

COMMANDS

docker build.

docker build -t ImageName:Tag directoryOfDocekrfile.

docker run image.



```
version: "3.8"
services:
  api:
    build: ./api
    container_name: api_c
    ports:
      - '4000:4000'
    volumes:
      - ./api:/app
      - ./app/node_modules
  myblog:
    build: ./myblog
    container_name: myblog_c
    ports:
      - '3000:3000'
    stdin_open: true
```

Docker Compose

tool for defining & running multi-container docker applications:

use yaml files to configure application services (docker-compose.yml):

can start all services with a single command: **docker compose up**: can stop all services with a single command: **docker compose down**: can scale up selected services when required.

Step 1: install docker compose (already installed on windows and mac with docker) docker-compose -v 2 Ways 1. https://github.com/docker/compose/rel...

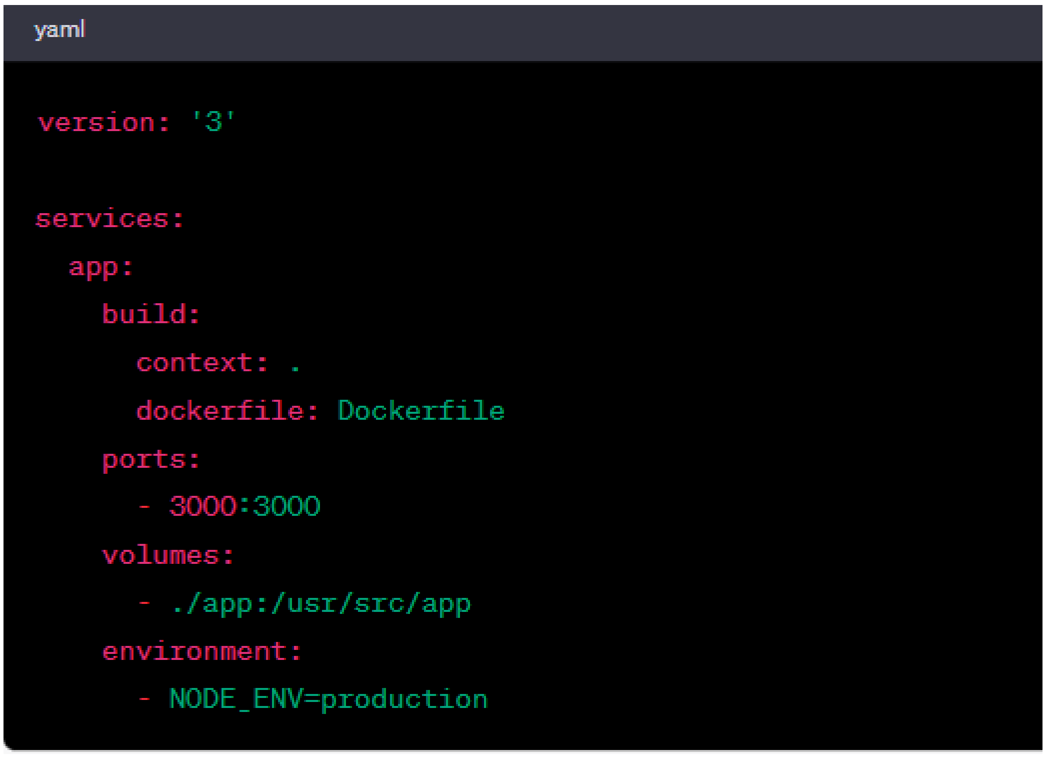
2. Using PIP pip install -U docker-compose

Step 2: Create docker compose file at any location on your system docker-compose.yml

Step 3: Check the validity of file by command docker-compose config

Step 4: Run docker-compose.yml file by command **docker-compose up -d.**

Steps 5: Bring down application by command docker-compose down TIPS How to scale services —scale docker-compose up -d --scale database=4



```yaml
version: '3'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 3000:3000
    volumes:
      - ./app:/usr/src/app
    environment:
      - NODE_ENV=production
```

Docker Volumes

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers :

docker volume //get information :

docker volume create :

docker volume ls :

docker volume inspect :

docker volume rm :

docker volume prune

Instead of deleting containers one by one of docker ps -a , we can use docker container prune. and for docker ps (running containers) we can use docker rm $(ps -aq)

Use of Volumes

Decoupling container from storage Share volume (storage/data) among different containers

Attach volume to container

On deleting container volume does not delete

By default, all files created inside a container are stored on a writable container layer

The data doesn't persist when that container is no longer running.

A container's writable layer is tightly coupled to the host machine where the container is running.

You can't easily move the data somewhere else.

Docker has two options for containers to store files in the host machine so that the files are persisted even after the container stops.

VOLUMES and BIND MOUNTS

Volumes are stored in a part of the host filesystem which is managed by Docker.

Non-Docker processes should not modify this part of the filesystem.

Bind mounts may be stored anywhere on the host system.

Non-Docker processes on the Docker host or a Docker container can modify them at any time.

In Bind Mounts, the file or directory is referenced by its full path on the host machine.

Volumes are the best way to persist data in Docker.

volumes are managed by Docker and are isolated from the core functionality of the host machine.

A given volume can be mounted into multiple containers simultaneously. '

When no running container is using a volume, the volume is still available to Docker and is not removed automatically.

You can remove unused volumes using **docker volume prune**.

When you mount a volume, it may be named or anonymous.

Anonymous volumes are not given an explicit name when they are first mounted into a container.

Volumes also support the use of volume drivers, which allow you to store your data on remote hosts or cloud providers, among other possibilities.

Commands

```
docker run --name MyJenkins1 -v myvol1:/var/jenkins_home -p 8080:8080 -p 50000:50000 jenkins
```

```
docker run --name MyJenkins2 -v myvol1:/var/jenkins_home -p 9090:8080 -p 60000:50000 jenkins
```

```
docker run --name MyJenkins3 -v /Users/raghav/Desktop/Jenkins_Home:/var/jenkins_home -p 9191:8080 -p 40000:50000 jenkins
```

Docker Swarm

A swarm is a group of machines that are running Docker and joined into a cluster

Docker Swarm is a tool for Container Orchestration.

A swarm is a group of machines that are running Docker and joined into a cluster.

A cluster is managed by swarm manager.

The machines in a swarm can be physical or virtual.

After joining a swarm, they are referred to as nodes.

Swarm managers are the only machines in a swarm that can execute your commands or authorise other machines to join the swarm as workers.

Workers are just there to provide capacity and do not have the authority to tell any other machine what it can and cannot do you can have a node join as a worker or as a manager. At any point in time, there is only one LEADER and the other manager nodes will be as backup in case the current LEADER opts out.

Let’s take an example:

You have 100 containers.

You need to do:

- Health check on every container
- Ensure all containers are up on every system
- Scaling the containers up or down depending on the load
- Adding updates/changes to all the containers Orchestration
- managing and controlling multiple docker containers as a single service

Tools available - Docker Swarm, Kubernetes, Apache Mesos

Step 1 : Create Docker machines (to act as nodes for Docker Swarm) Create one machine as manager and others as workers

docker-machine create --driver hyperv manager1.

docker-machine create --driver virtualbox manager1.

docker-machine:Error with pre-create check: “exit status 126” https://stackoverflow.com/questions/3...

brew cask install virtualbox;

Create one manager machine and other worker machines.

Step 2 : Check machine created successfully.

docker-machine ls

docker-machine ip manager1

Step 3 : SSH (connect) to docker machine

docker-machine ssh manager1

Step 4 : Initialize Docker Swarm

docker swarm init --advertise-addr MANAGER_IP docker node ls (this command will work only in swarm manager and not in worker)

Step 5 : Join workers in the swarm

Get command for joining as worker

In manager node run command **docker swarm join-token worker** This will give command to join swarm as worker

docker swarm join-token manager This will give command to join swarm as manager SSH into worker node (machine) and run command to join swarm as worker

In Manager Run command - **docker node ls** to verify worker is registered and is ready Do this for all worker machines

Step 6 : On manager run standard docker commands **docker info** check the swarm section no of manager, nodes etc Now check docker swarm command options docker swarm

Step 7 : Run containers on Docker Swarm **docker service create --replicas 3 -p 80:80 --name serviceName nginx**

Check the status: docker service ls

docker service ps serviceName

Check the service running on all nodes Check on the browser by giving ip for all nodes

Step 8 : Scale service up and down On manager node

docker service scale serviceName=2

Inspecting Nodes (this command can run only on manager node)

docker node inspect nodename

docker node inspect self

docker node inspect worker1

Step 9 : Shutdown node

docker node update --availability drain worker1

Step 10 : Update service

docker service update --image imagename:version web docker service update --image nginx:1.14.0 serviceName

Step 11 : Remove service

docker service rm serviceName

docker swarm leave : to leave the swarm

docker-machine stop machineName : to stop the machine

docker-machine rm machineName : to remove the machine