# Patching of Linux

yum update all

(App softwares + kernal)

Server Provisining ⟷ Server development

Production Environment :-

Cloud architect
DevOps Architech
RHCA (Redhat certified architect)

Architect :-

- Application language
- database use
- queries
- user connects
- Ram/HDD processor/bandwidth

5 server / 128 CPU / 256 GB Ram / --- IOPS

Dedicated machine / Cloud machine

to Ready all IS Provisioning

## Development

OS install
webserver setup
Firewall ~~Config~~
Configuration

## machine provisioning

⇒ dedicated machine (OS ⇒ Application ⇒ App)
(2006) virtual machine (OS ⇒ vmware ⇒ machines ⇒ OS ⇒ Application ⇒ App)
Cloud instance

Container technology

## Configuration management tool

⇒ Configuration on multiple machine
⇒ error less.
⇒ write once is recorded

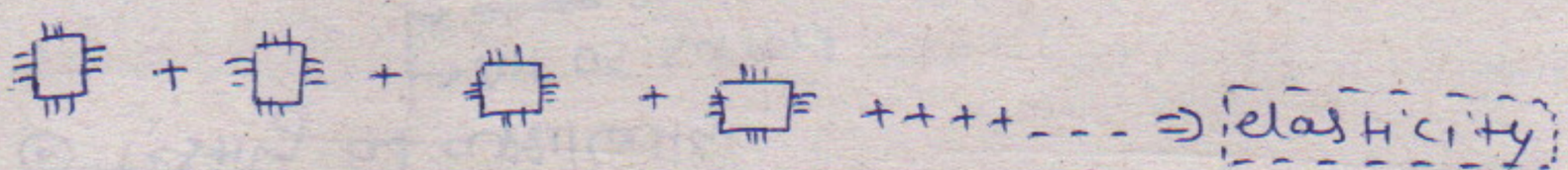
---

Cloud Computing is a software which manage virtual machines or large infra management.
On demand service, Billing dashboard.
Pay per use, migration

Rack space
Amazon
Azure cloud
Alibaba
Google

openstack

private cloud + public cloud = Hybrid cloud

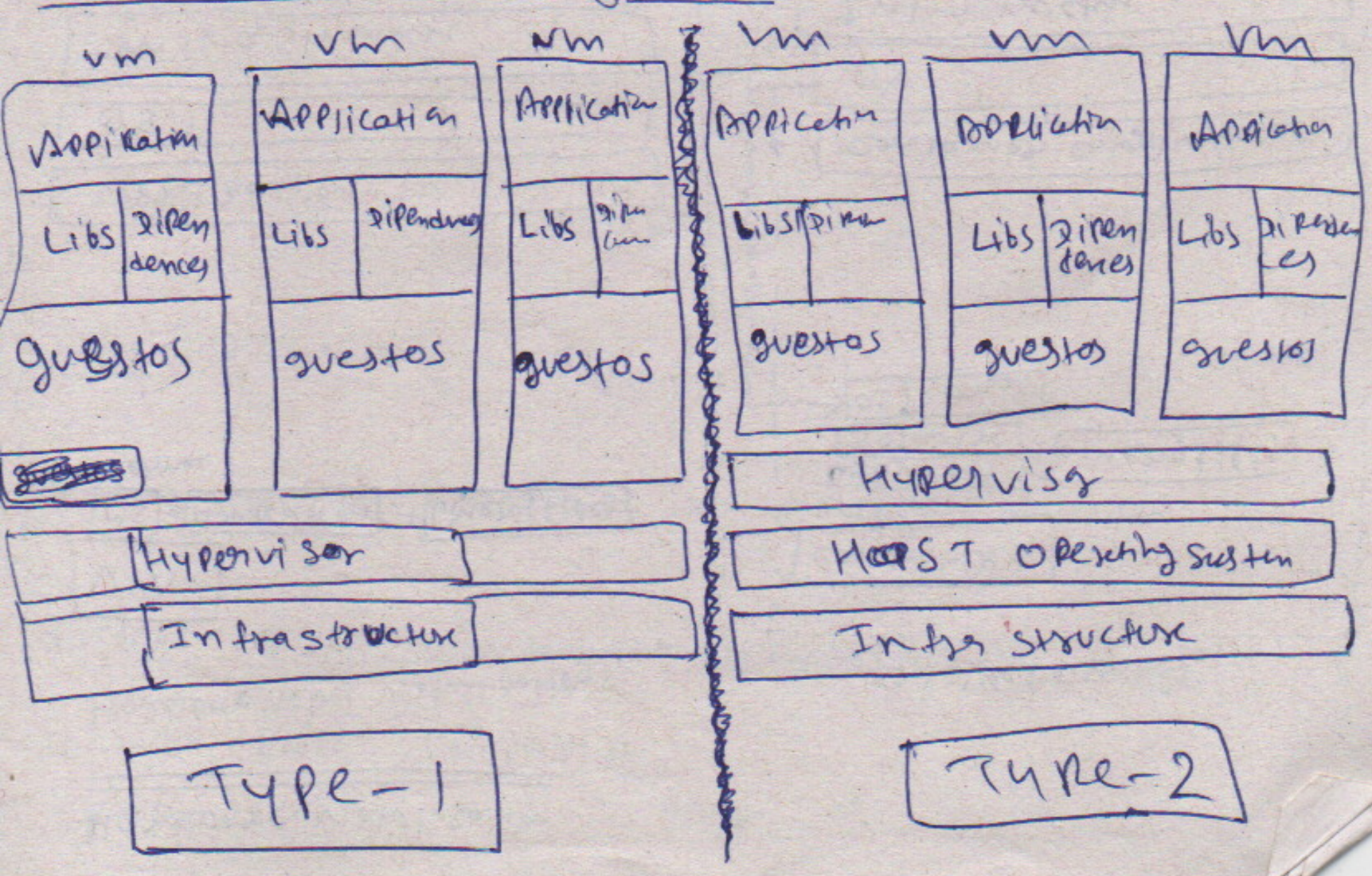



t2.micro ⇒ t1.micro ....  ⇒ Scalability

2gb ram ⇒ 4gb ram ⇒ 8gb ram ....

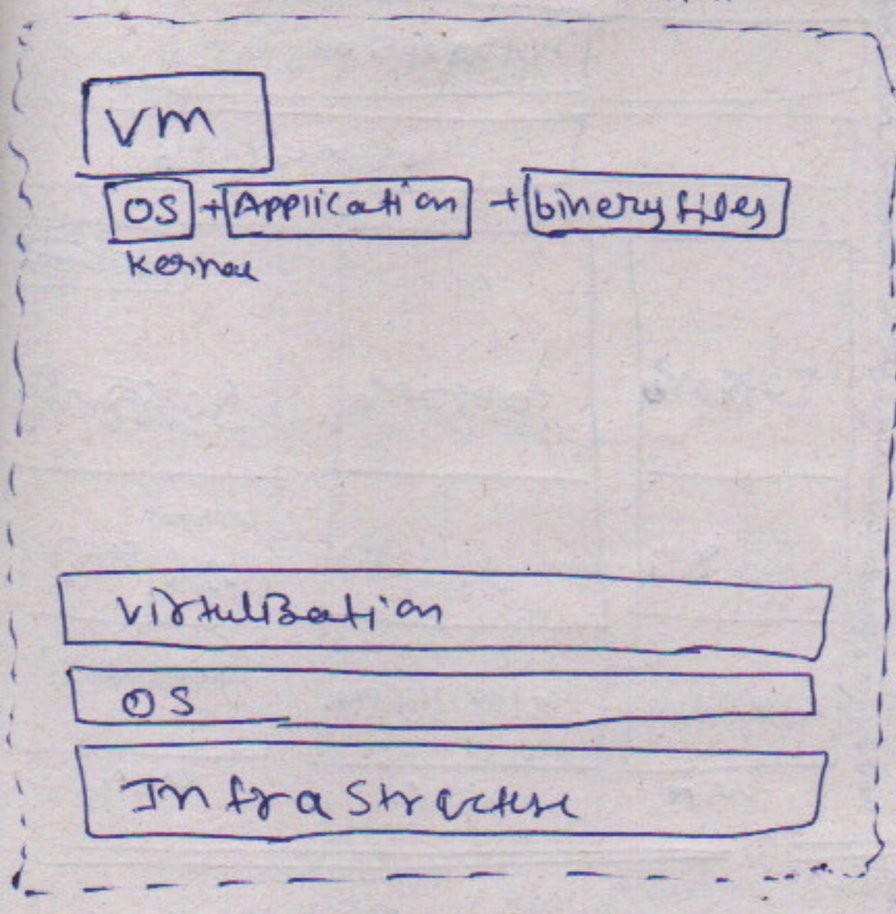
Cloud formation :→ Create template & create machine using that.

Teraform is a Provisioning tool :→

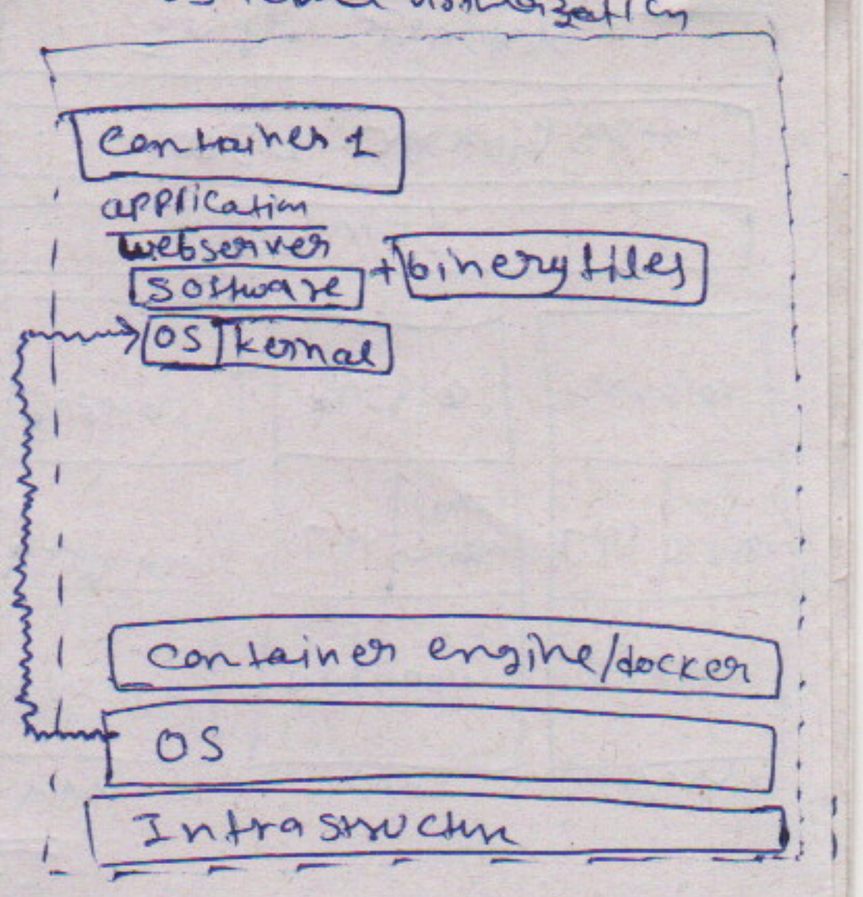


# Hardware virtulization

Hardware level virtulization




OS level virtulization




Hyper-V | enable

windows Host OS ⇒ Linux Container ✓

Linux HOST OS ⇒ windows Container ✗

## Usecase of docker

- Testing of application
  - → All OS support
  - → ~~check~~ Provide machines with diffrent OS to team with less
- Infra Cost
- To launch a container you requires a image.
  docker file ------- infrastructure as a code.
- less developer cost
- less time
- micro service architecture.

⊙ portable and movable on any cloud.

docker - CE → community edition

docker - EE → enterprise editia

rpm -q docker-ce ⇒ check docker is installed or not

add yum repo ⇒ Repo add

yum install docker-ce -y ⇒ install docker

docker -v ⇒ check version

docker --help ⇒ help

systemctl start docker ⇒

systemctl enable docker ⇒

docker images ↵ ⇒ list images

hub.docker.com ⇒ URL

docker search centos ↵ ⇒ docker hub ते search

docker pull centos ↵ ⇒ centos चा latest image

docker pull centos:7
↑ tag

docker Pull richxsl / rhel7 : min ↵

↑ username ↑ image ↑ tag

docker images ↵ docker images देखने के लिए हमारी local repo

docker run -it --name = con1 richxsl/rhel 7 : min
⇒ launch a container

docker Stop con1 ⇒ Running container को stop करने के लिए

docker start con1 ⇒ stopped वाले container को चालू करने के लिए

docker rm con1 ⇒ stoped container को Remove करने के लिए

docker rm -f con1 ⇒ forcefully Running container को delete करने के लिए

→ we will be inside container after create

- exit ↵ the container will stop then exit

- Ctrl + P , q exit without container stop

docker PS ↵ ⇒ all running container list

docker PS -a ⇒ all running + stop container list

⇒ यदि --name = _____ नहीं देंगे तो randomly कोई भी name Pick कर लेगा

★ जो docker images हमारे container में use आ रही हैं उन्हें हम docker rm centos command से remove नहीं कर सकते हैं।

docker attach con1 ⇒ बाहर आने के बाद container में वापस जाने के लिए

docker ps -q (all ids of Running containers)

docker ps -a -q (all ids of Running & stop container)

docker stop $(docker ps -a) ⇒ stop all running containers

docker rm $(docker ps -a -q) ⇒ Remove all stopped container

docker rm -f $(docker ps -q) ⇒ remove all running contain
↑ forcefully

docker images ⇒ list all local images
↑ see all

docker rmi httpd ⇒ Remove httpd image
↑ image name

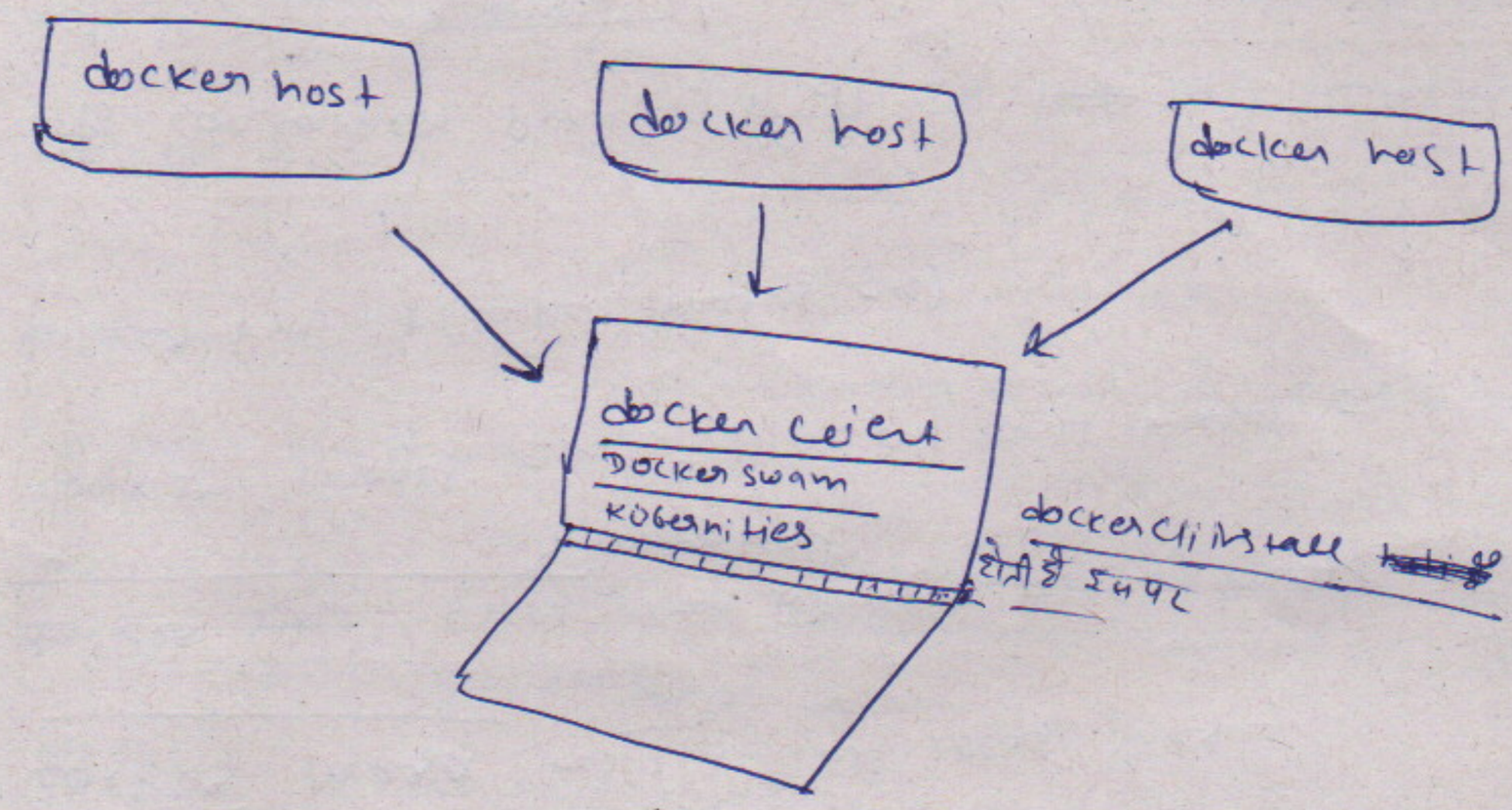
docker images -q ⇒ list all local images Id

docker rmi $(docker images -a)

यदि container Running है तो हम उससे related image delete नहीं कर सकते।

docker host → जिस machine पर docker काम work कर रहे है या docker install कर रखा है वह docker host होता है जहां पर container launch होते है

**docker Client** ⇒ we are managing multiple docker host then we manage all docker host form a system that ~~is called~~ ~~do~~ machine is called docker client.




yum install -y net-tools ⇒ for ifconfig command
IP addr

**docker images are :** Application + Libraries (Read only)

**Container image :** Runable image

**Registery :** जहां पर docker image available होती है, उदा: Docker hub

**images layer :** → /var/lib/docker2/overlay2/

सभी images कि layers

**docker exec -it Con1 /bin/bash**

↓

Shell देकर open करने के लिए

docker top con1 ⇒ container की process id देखने के लिए

docker run -d Ubuntu sleep 500

↑ Background ↓ Command after catch-alike

docker attach f1358ara

↑

to bring container in foreground

docker run timer ↵

docker run jenkins ↵

docker inspect Container id ↵

see container info

docker run -p 8080:8081 Jenkins

↓ Port ↓ Host port ↑ ~~Guest port~~ Container port

Yaml is a data serialization language.

docker run -P 8080:8080 -v /data/jenkindata:/var/

Hostport Container port Host dir

jenkins_home jenkins

Container dir

-u root

User

Docker file

FROM Ubuntu ⇒ OS image

RUN apt-get update ⇒ Run Command

RUN apt-get install -y python python-pip

RUN pip install flask

COPY app.py /opt/app.py

Entry point ⇒ Run Commas

docker build .

docker build . -t my-simplee-webapp

name

```
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 login

UN —
PW —

docker push dennisinstities/~~mypro~~ imagename

## dockerfile

```
From Python:3.6
RUN Pip install flask
COPY . /opt/
EXPOSE 8080
WORKDIR /opt
ENTRYPOINT ["python", "app.py"]
```

## Environment variables →

set

```
docker run -e APP_COLOR=blue simple-webapp
```

find in running Container

```
docker inspect bissful_container
```

container name

```
Config : {
    "Env": {
        APP_COLOR = blue
        ⋮
    }
```

docker-compose install on node ~~via do~~

docker hub - docker compose
documentation ~~do~~ online

---

we use yaml to write docker compose file

~~do cto~~ vim docker-compose.yml

version-1

```
redis:
    image: redis

db:
    image: postgres:9.4

vote:
    image: voting-app
    port:
      - 5000:80
    links:
      - redis

worker:
    image: worker-app
    links:
      - db
      - redis

result:
    image: result-app
    ports:
      - 5000
    links:
      - db
```

→ docker-compose UP

## docker file in ver-3

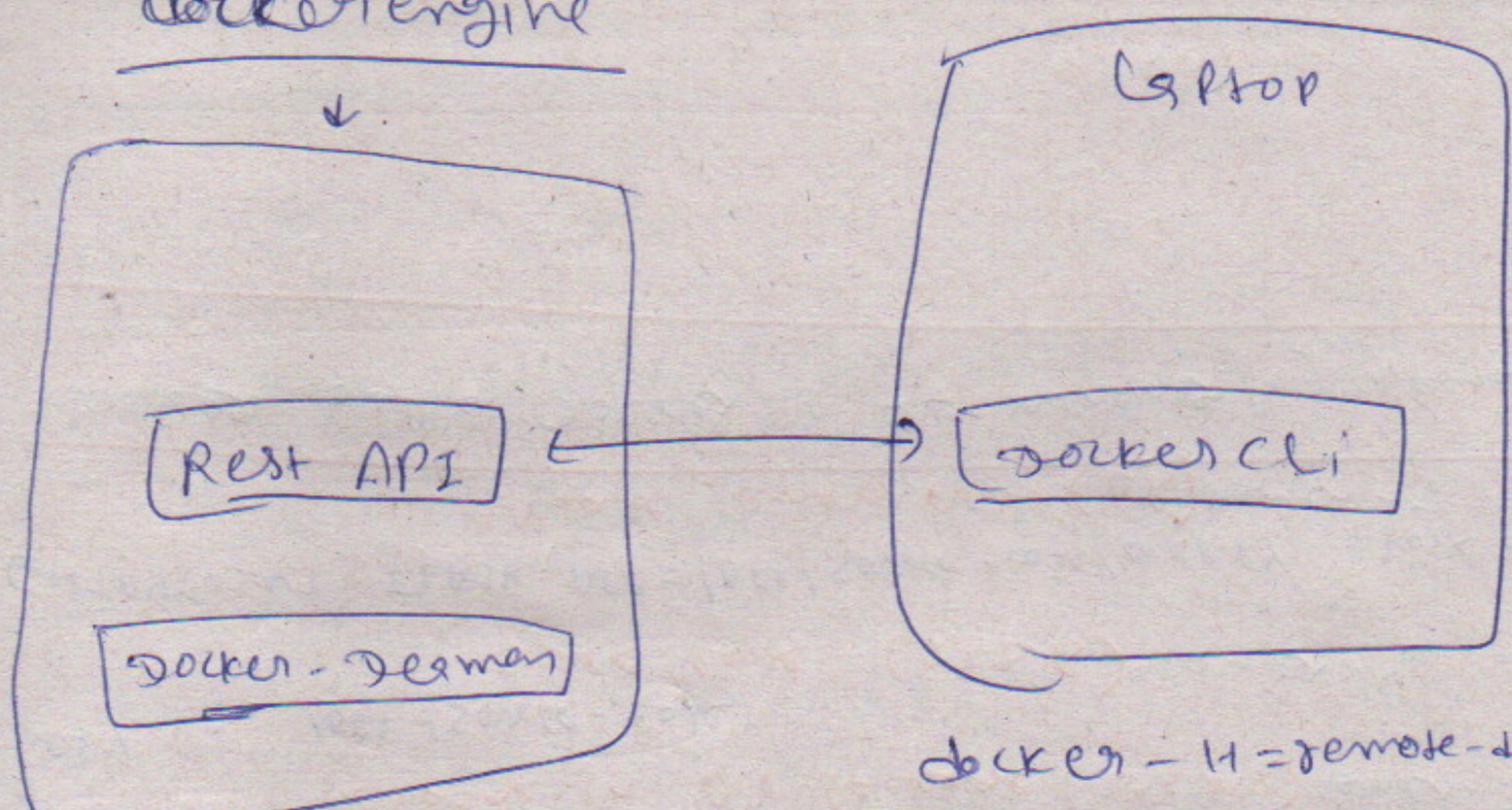
(14)

```
version: "3"
services:
    redis:
        image: redis

    db:
        image: postgres:9.4
                                  environment
    vote:                           POSTGRES_USER=admin
        image: voting-app           POSTGRES_PASSWORD=admin
        ports:
            - 5000:80

    worker:
        image: worker-app

    result:
        image: result-app
        ports:
            - 5001:80
```

---

# docker-compose up

## docker engine

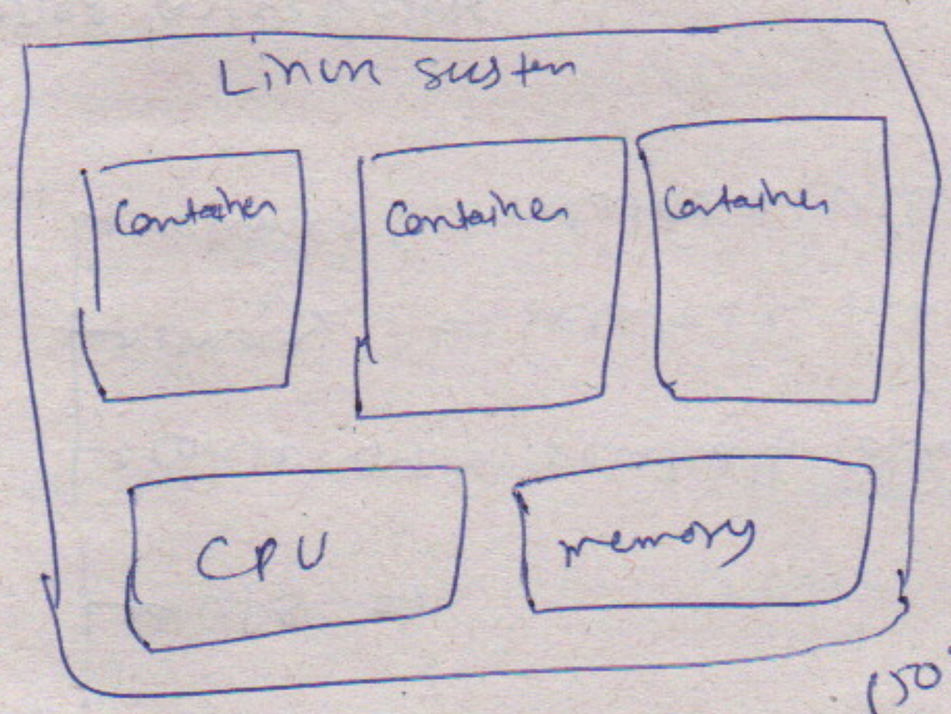



docker -H = remote-docker engine :2375

docker -H = 10.12.21:2375 run nginx

---

## Cgroup: Control Resources of Container




(50%)

docker run --CPUS = .5 Ubuntu

-- memory = 100m

## docker file in ver-3

(14)

version: "3"

## File system

(16)

### /var/lib/docker

- aufs ⇒
- containers ⇒ related to container info
- image ⇒ related to image info
- volumes ⇒ related to volume info

## layered architecture

### Dockerfile

From ubuntu  base ubuntu image (120MB) layer-1

RUN apt-get update && apt-get -y install python
change in apt package (306 MB) layer-2

~~RUN RUN~~ RUN pip install flask flask-mysql
change in pip package (6.3 MB) layer-3

COPY . /opt/source-code
source code (229 B) layer-4

ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
update entry point (0B) layer-5

---

docker build Dockerfil -t manender/mycustomapp

---

## docker volume



Create volume

docker volume create data_volume

/var/lib/docker

↳ volumes

↳ data_volume

docker run -v data_volume:/var/lib/mysql mysql

volume mounting

docker run -v data_volume2:/var/lib/mysql mysql

docker run -v /data/mysql:/var/lib/mysql mysql

bind mount

-v ⟷ --mount

docker run --mount type=bind, source=/data/mysql,

target=/var/lib/mysql mysql

### Storage drivers

AUFS - Ubuntu  BTRFS  overlay

ZFS  Devicemapper  overlay2

do clear file in ver-3

(14)

version "2"

(18)

docker networking

Bridge

docker run ubuntu

none

docker run ubuntu --network=none

docker run ubuntu --network=host

docker network ls → see networks

Embedded DNS →

container name

docker network inspect bridge

docker network create --driver bridge

--subnet 182.18.0.1/24 --gateway 182.18.0.1

wp-mysal-network

19

docker ~~ls~~ network inspect wp-mysal-network

Docker service

docker service create --replicas=3

--network frontend my-web server

-p 8080:80

~~docke~~

declare file in ver-3

(14)

version: "2"

kubectl run --replicas=1000 my-webserver

kubectl scale --replicas=200 my-web-ser

→ increase

(20)

kubectl rolling-update my-web-server

--image=web-server-2

kubectl rolling update my-webserver --rollback

K8 - Parts

api-server -

etcd → Store data & logs

scheduler - Control container

Controller - it's mind & notify to scheduler, make decision

kubelet - Agent

Container runtime

kubectl run hello-mini

kubectl cluster-info

kubectl get nodes

kubectl run my-web-app --image=

my-web-app --replicas=5