Docker

A-1
Page
Date / /

→ Docker is an open-source centralised Platform designed to deploy & run applications.

→ Docker uses containers on the Host OS to run applications. It allows app^n to use the same kernel as a system on the host computer, rather than creating a whole virtual OS.

→ We can install docker on any OS but Docker engine runs natively on Linux distribution.

→ Docker written in 'go' language.

→ Docker is a tool that performs OS level virtualization, also known as Containerization.

→ Before Doker, many users faces the problem that a particular code is running in the developers system but not in the users system.

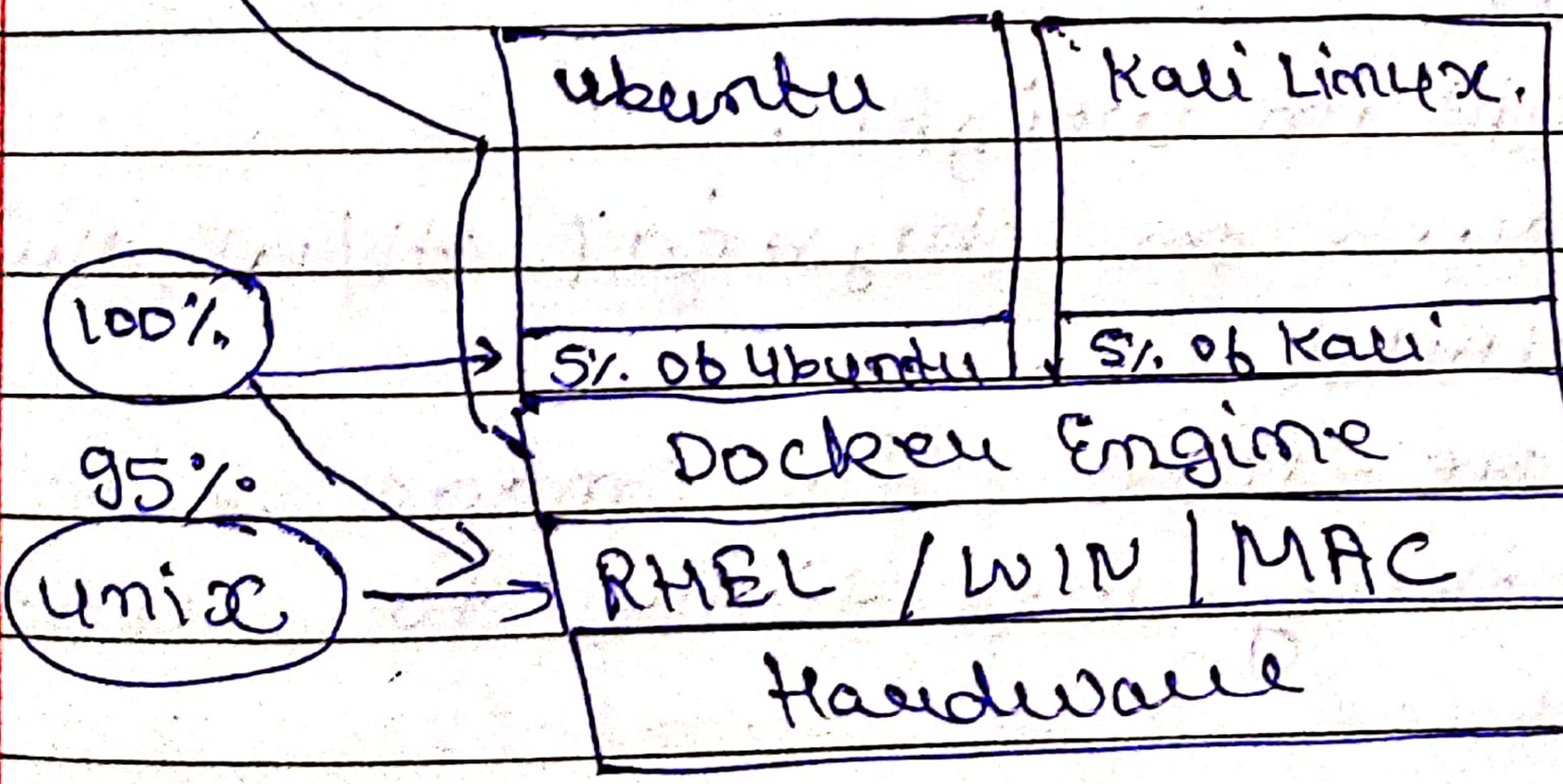
→ Docker was first release in March 2013. It is developed by solomen Hykes & sebastian Pahl.

→ Docker is a set of Platform as a service that use OS level virtualization

where as VMware uses Hardware level virtualization.

VMware	AWS EC2	Docker
		Docker Hub
VM VM / OS OS / Hypervisor / Hardware	VM VM / OS OS / Xen/Nitro / Hardware	Cont. Cont. / Docker engine / OS / Hardware

Docker Hub (everything is available on Docker Hub)



container takes 95% of the resources from the OS, that's why we called it has Operating system level virtualization.

→ Container has its own OS but only 5-10% of the total OS. another 90% takes from (RHEL) OS.

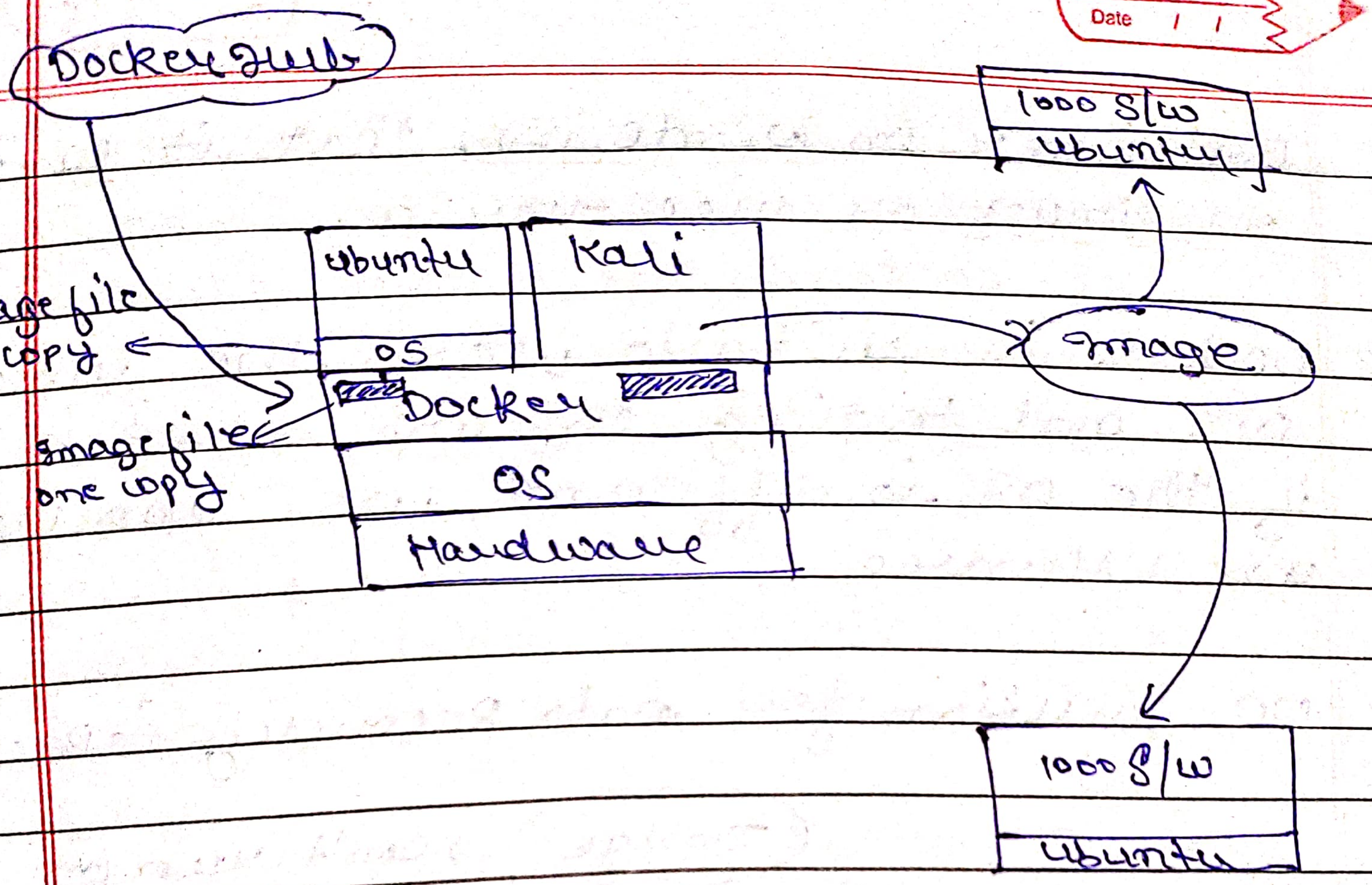
→ Container is light weight because it doesn't have its own full OS.

Advantages, Disadvantages & Architecture of Docker

↳ Advantages of Docker →

→ No pre-allocation of RAM.
→ CI efficiency → Docker enables you to build a container image & use the same image across every step of deployment process.

→ Less cost
→ It is light in weight.
→ It can run on physical H/w, virtual H/w or on cloud.
→ You can re-use the image.
→ It takes very less time to create container.



★ Image —Running→ Container

Stop

→ In Image file we can't make any changes.

Disadvantages of Dokeer →

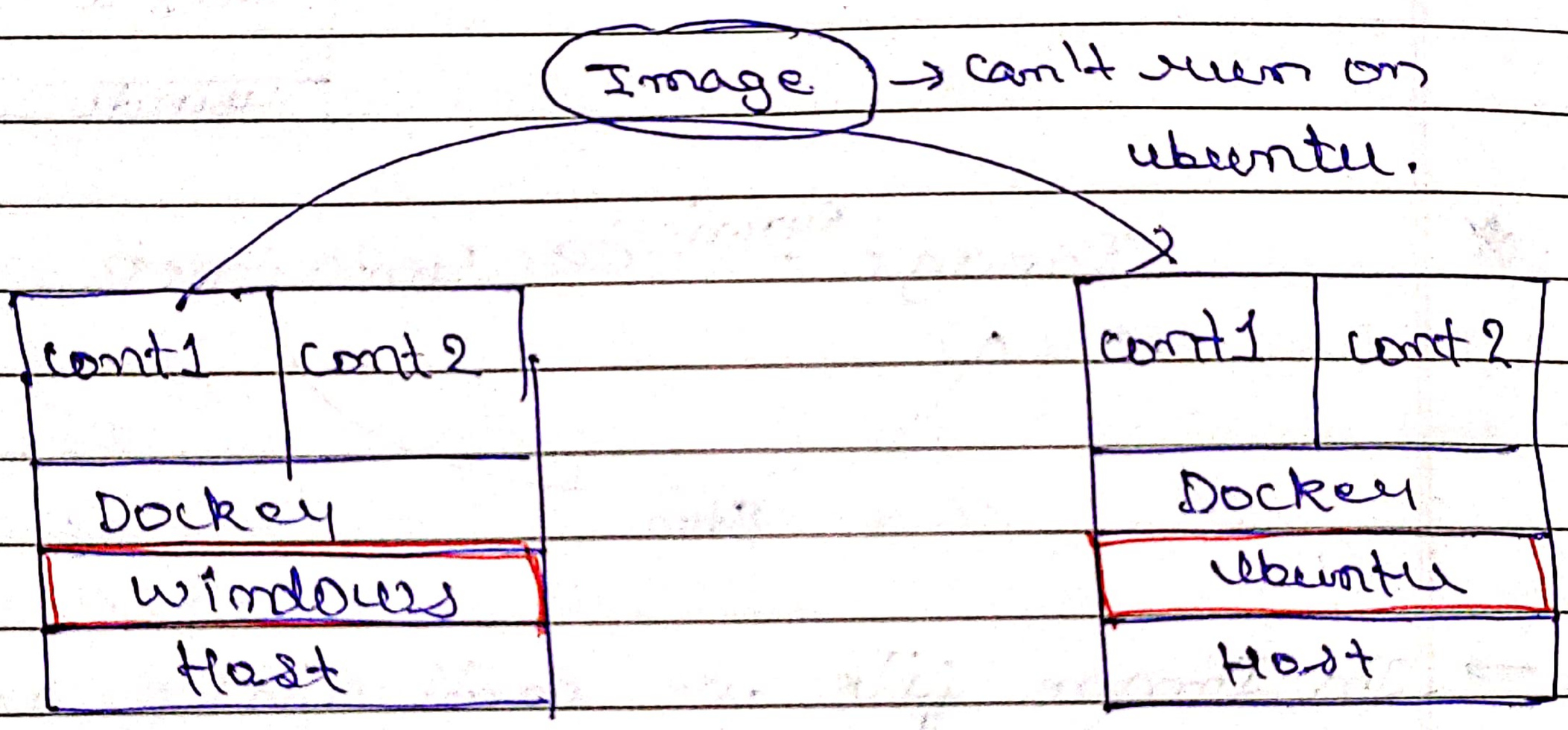
↳ docker is not a good solution of application that requires rich GUI.
→ Difficult to manage large amount of containers.
→ Docker does not provide cross-platform compatibility means if an application is designed to run in a docker.

Date / /

container on windows, that it can't run on linux or vice-versa.

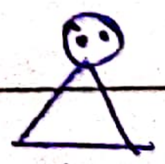
→ docker is suitable when the development O.S and testing OS are same.
if the OS is different, we should use VMware.

→ NO solution for data Recovery & Backup.

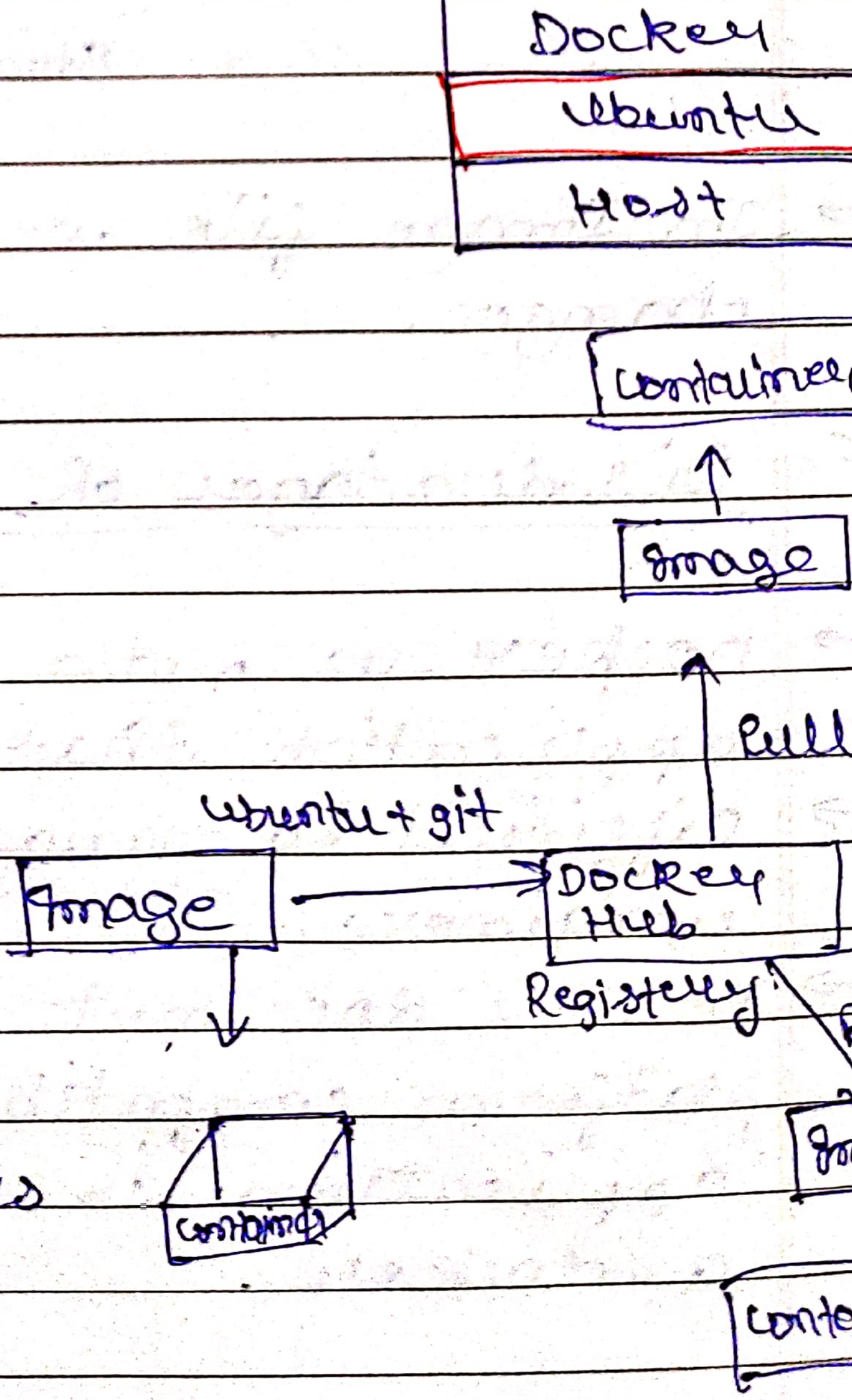


Dev

Developer



Docker file → dependencies & Req S/w



Containerd is a layered file system. Everythings done step by step is called layered file system.

(set of S/W or package) | Dockers Ecosystem

Dockers client | Dockers Daemon or Server | Dockers Hub or Registry | Dockers Image ↓ Templates | Dockers compose.

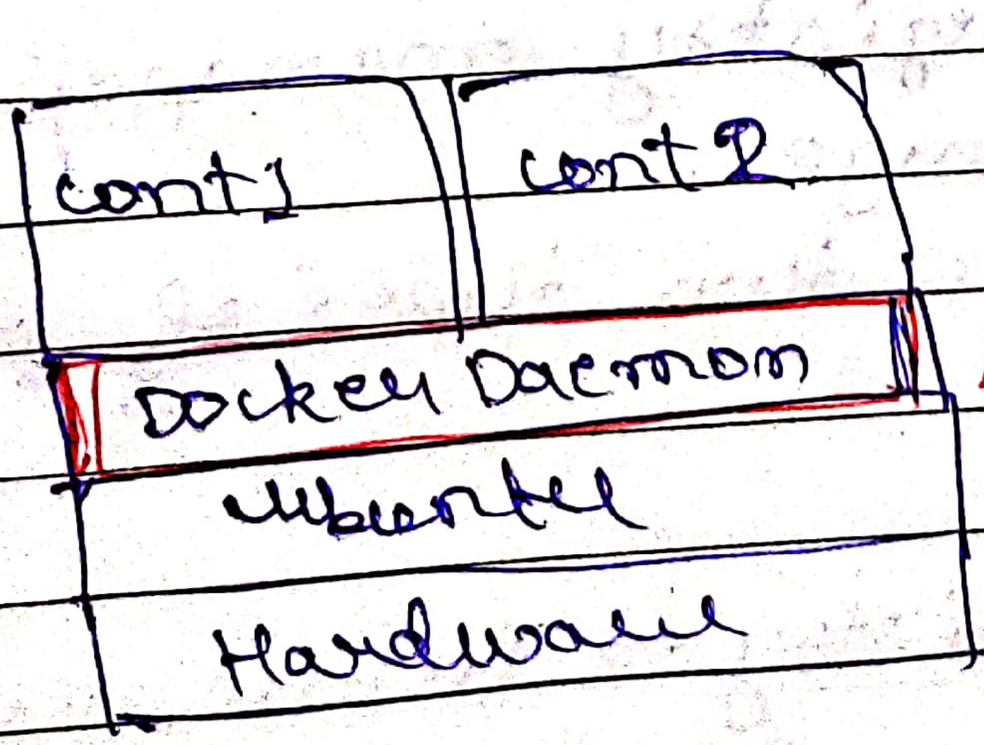
Components of Dockers

1) Dockers Daemon →

→ Dockers Daemon runs on the Host OS

→ It is responsible for running containers to manages docker services.

→ Dockers Daemon can communicate with other daemons.



2) Dockers client →

→ Dockers user can communicate with dockers daemon through a client.

→ Docker client uses command & Rest API to communicate with the docker daemon.
→ When a client run any server command on the docker client terminal, the client terminals send these docker commands to the Docker daemon.
→ It is possible for Docker client to communicate with more than one daemon.

3) Docker Host

→ Docker Host is used to provide an environment to execute and run applications.
→ It contains the Docker daemon, images, containers, networks & storages.

4) Docker Hub / Registory →

→ Docker registry manages & stores the docker image.
→ There are two types of registries in the Docker

① Public Registay ⇒ Public registry is also called as Docker hub.

② Private Registay ⇒ It is used to share images within the enterprise.

5) Docker Image →

Docker image are used to read only binary templates used to create docker containers.

OR

Single file will all dependencies & config required to run a program.

Ways to create image →

① Take image from Docker hub.
② create image from Docker file.
③ create image from existing docker containers.

6) Docker Container →

→ Container hold the Entire packages.
→ that is needed to run the application.

In other words, we can say that, the image is a template & the container is a copy of the that template.

→ Container is like a virtual Machine.
→ Images become container when they run on Docker engine.

Basics command in Docker →

→ (1) To see all images present in your local machine.

↳ docker image,
→ local image on your machine

(2) To find out images in docker Hub
↳ docker search jenkins (name of image)

(3) To download image from docker hub to local machine.

↳ docker pull jenkins
(name of image)

create + start

after running this command we will automatically get into the container.

(4) To give name to containers.
↳ docker run -it --name shivam ubuntu
(image)

interactive mode — terminal — /bin/bash → shell

(5) To check service is start or not.
↳ service docker status

(6) To start containers
↳ docker start shivam
↳ container name

(7) To go inside container.
↳ docker attach shivam

(8) To see all containers.
↳ docker ps -a
↳ all

(9) To see only running containers.
↳ docker ps
↳ process status

(10) To stop container
↳ docker stop shivam
container name

(11) To delete container
↳ docker rm shivam
↳ remove.

(12) To see docker is installed or not & where

↳ which docker.

↳ /usr/bin/docker ← location.

(13) To check which OS is run on Docker

↳ cat /etc/OS-release.

(14) get out from the container

↳ exit

Dockerfile components & 'diff' command

→ Login into AWS account & start your EC2 instance access it from putty

→ Now we have to create container from our own image.

→ Therefore create one container first.

→ docker run -it --name shivamcontainer ubuntu /bin/bash.

→ cd /tmp

→ Now create one file inside this tmp directory

↳ touch myfile.

→ Now if we want to see the difference b/w

the base image ↑ changes on it then

★ → docker diff shivamcontainer.

o/p C → /root
A → /root/.bash_history
C → /tmp
A → /tmp/myfile

C - change
A - Append
D - delete

★ → Now create image of this container
(old image) (new image)

→ docker commit new container updateimage
container name. image name we want to give.

→ docker images.

→ Now create container from this file.

→ Docker run -it --name shivam1container updateimage /bin/bash

→ check myfile is present or not in new image

→ root@cid # ls
cd /tmp
o/p → myfile { you will get all file back }

Dockerfile →

→ Dockerfile is basically a text file. It contains some set of instructions.

→ Automation of Docker image creation.

Docker components

1) FROM → for base image. this command must be on top of the dockerfile.

2) RUN → To execute command. It will create a layer in image.

3) MAINTAINER → Author / owner / description.

4) COPY → copy files from local system (Docker VM) we need to provide source, destination (we can't download file from internet and only remote repo)

5) ADD → similar to COPY but, it provides a feature to download files from internet, also we extract file at docker image side.

6) EXPOSE → To expose ports such as port 8080 for tomcat, port 80 for nginx etc.

7) WORKDIR → To set working directory for a container.

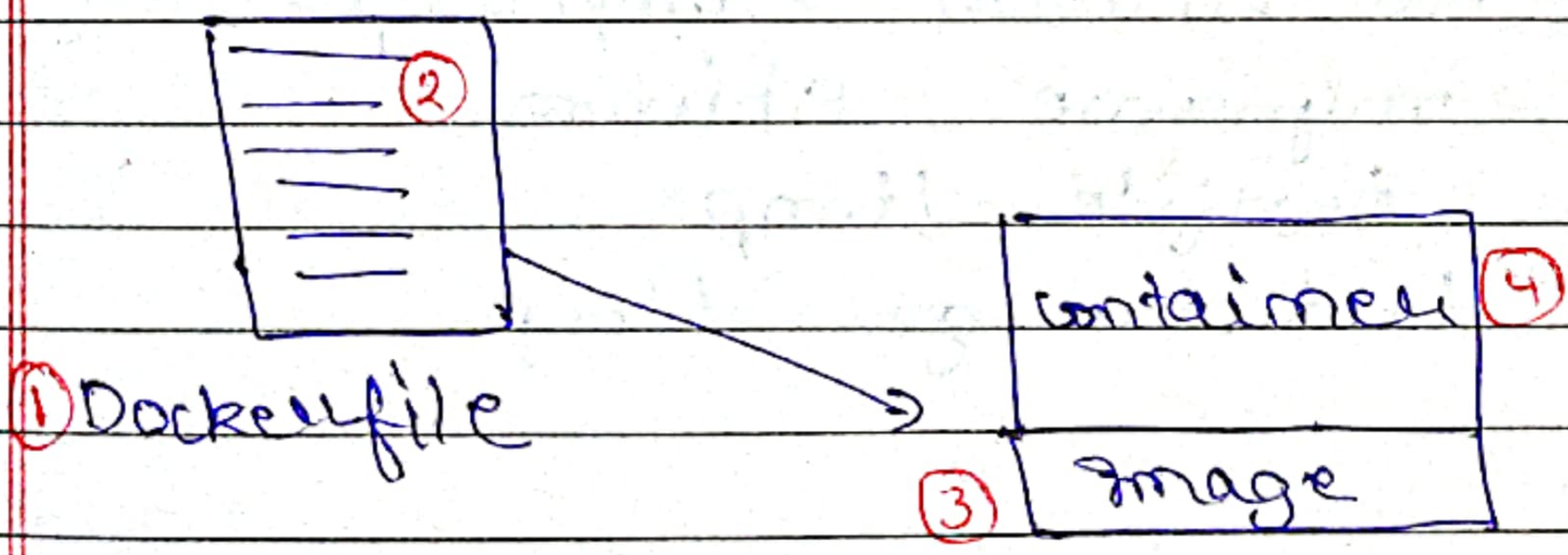
8) CMD → Execute commands but during container creation.

9) ENTRYPOINT → Similar to CMD, but has higher priority over CMD, first command will be executed by ENTRYPOINT only.

10) ENV → Environment variables.

Dockerfile →

1. create a file named Dockerfile.
2. Add instructions in Dockerfile.
3. Build Dockerfile to create image.
4. Run image to create container.



1. vi Dockerfile.
2. FROM ubuntu
   RUN echo "shivam" > /tmp/testfile.
3. To create image Dockefile,

↳ docker build -t myimage .

(image name)

tag

current directory

→ docker ps -a
↳ docker images.

④ Now create container from the above image
↳ docker run -it --name my container
myimage /bin/bash.

↳ cat /tmp/testfile.

New Dockerfile

→ vi Dockerfile.
↳ FROM ubuntu.
↳ WORKDIR /tmp
↳ RUN ECHO "Shivam" > /tmp/testfile.
↳ ENV myname Shivam
↳ COPY testfile /tmp
↳ ADD test.tar.gz /tmp.

Docker volume & How to share it.

→ Volume is simply a directory inside our container.
→ Firstly we have to declare this directory as a volume & then share volume.

→ Even if we stop container, still we can access volume.

→ Volume will be created in one containers

→ You can declare a directory as a volume only while creating containers.

→ You can't create volume from existing containers.

→ You can share one volume across any number of containers.

→ Volume will not be included when you update an image.

→ You can mapped volume in two Ways :-
1) container ⟷ container
2) Host ⟷ container.

Q How I add volume to existing docker container ?

↳ If we want to add a volume to a running container.
we can use docker commit to make a new image based on that container, and then clone it with the new volume.

Benefits of Volume

→ Decoupling Container from storage.
→ share volume among different containers.
→ Attach volume to containers.
→ on deleting container volume does not delete.

Creating Volume from Dockerfile.

① create a Dockerfile & write.
↳ vi Dockerfile
FROM ubuntu
VOLUME ["/myvolume1"]

② Then create image file from this Dockerfile.
→ docker build -t myimage .

③ Now create a container from this image & run
→ docker run -it container1 myimage /bin/bash.
(container name) (image name)
↳ Now do 'ls', you can see 'myvolume1'

(4) Now share volume with another container

container1 ⟷ container2.

→ docker run -it --name container2 (new cont)
--privileged=true --volumes-from container1 (old cont) ubuntu /bin/bash.

Now after creating container2, myvolume1 is visible whatever you do in one volume, can see from other volume.

→ touch /myvolume1/samplefile
→ docker start container1
→ docker attach container2.
↳ ls - /myvolume1.
you can see samplefile here.

Create Volume by using command -

→ docker run -it --name container3
-v /volume2 ubuntu /bin/bash

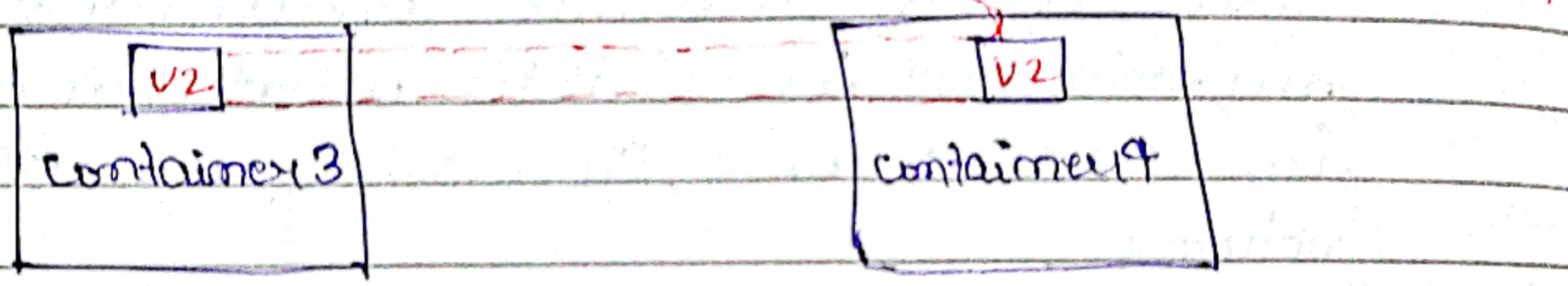
→ Do ls → cd /volume2

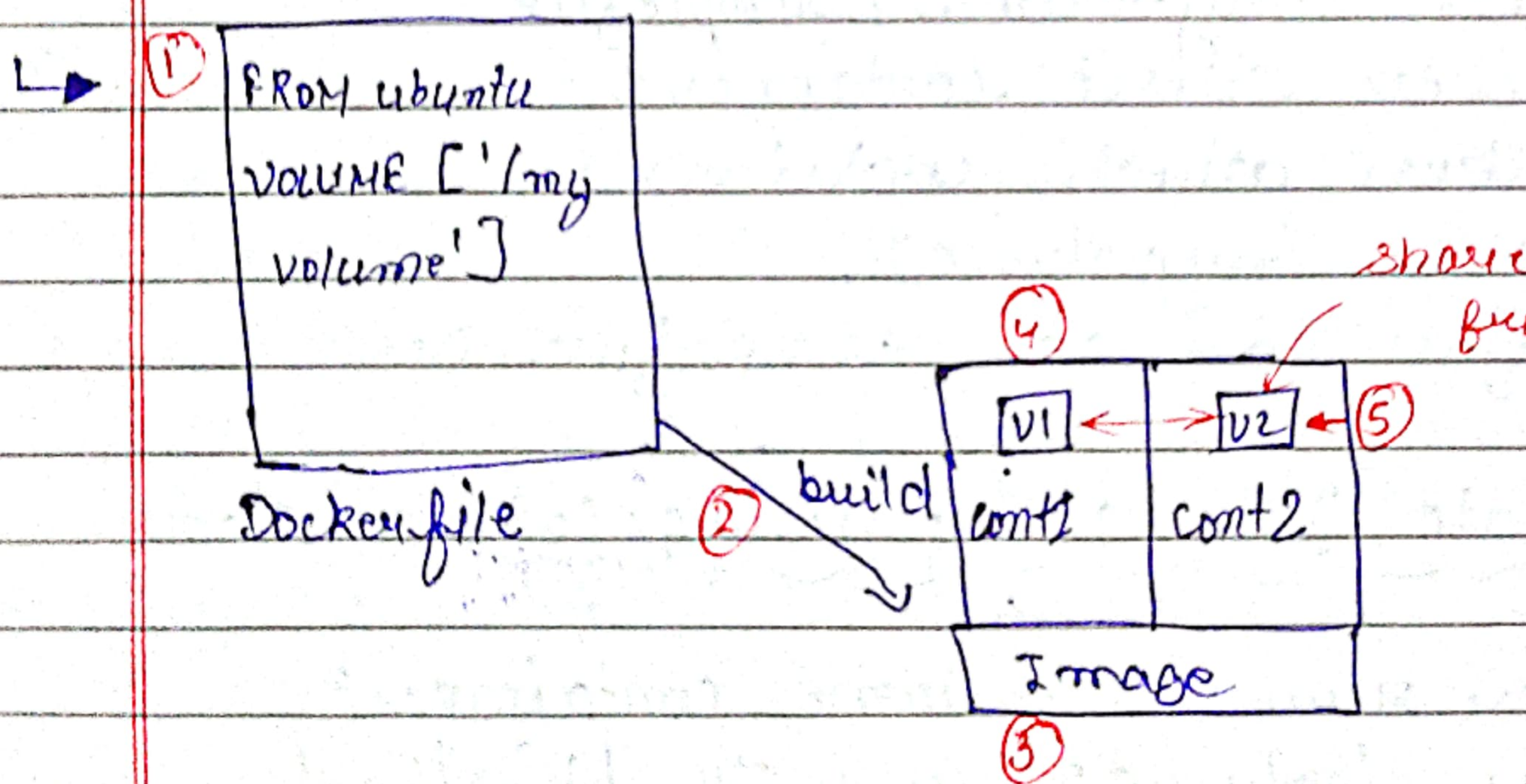
→ Now create one more container, & share volume2

→ docker run -i-t --container1 --privileged=true
--volumes-from container3 ubuntu /bin/bash.

→ Now you are inside container, do 'ls' can see volume2

→ Now create one file inside this volume and then check in container3, you can see that file.





Volume (Host - container)

→ Verify files in /home/ec2-user

run this command into /home/ec2-user

→ docker run -it --name hostcontainer -v /home/ec2-user:/shivam --privileged=true /bin/bash

Host — Container

→ cd /shivam
Do ls, now you can see all files of host machine.

→ touch shivamfile (in container)

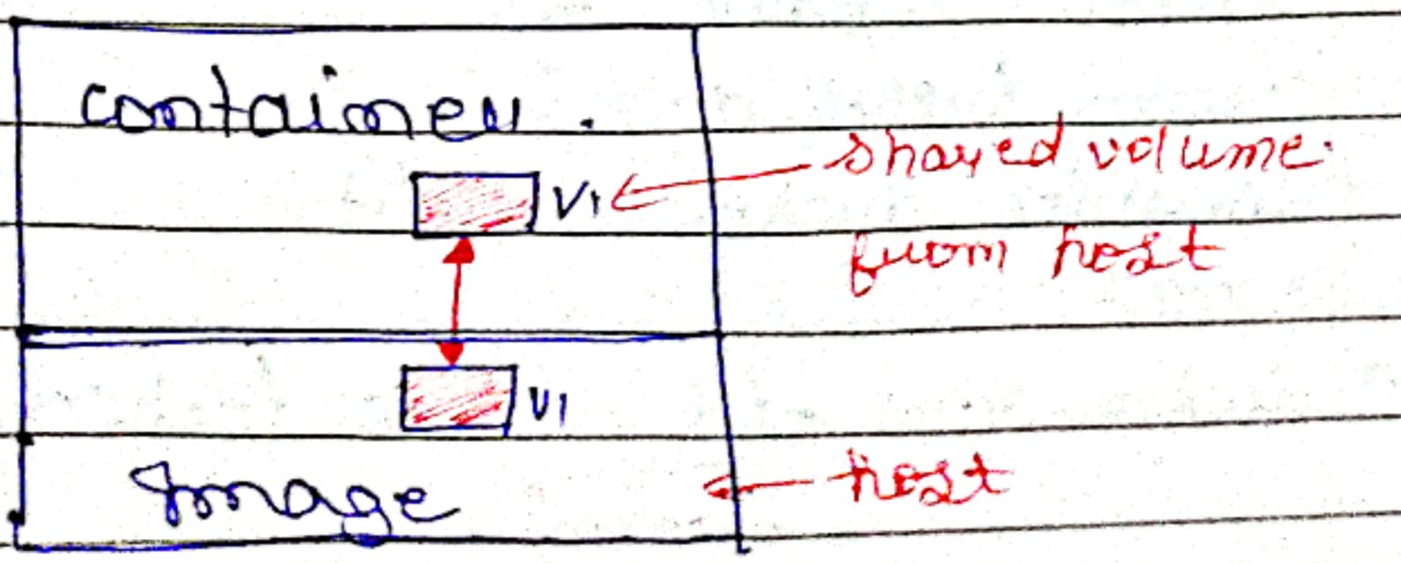
→ Now check in EC2 machine, you can see this file.

Some other commands →

→ docker volume ls

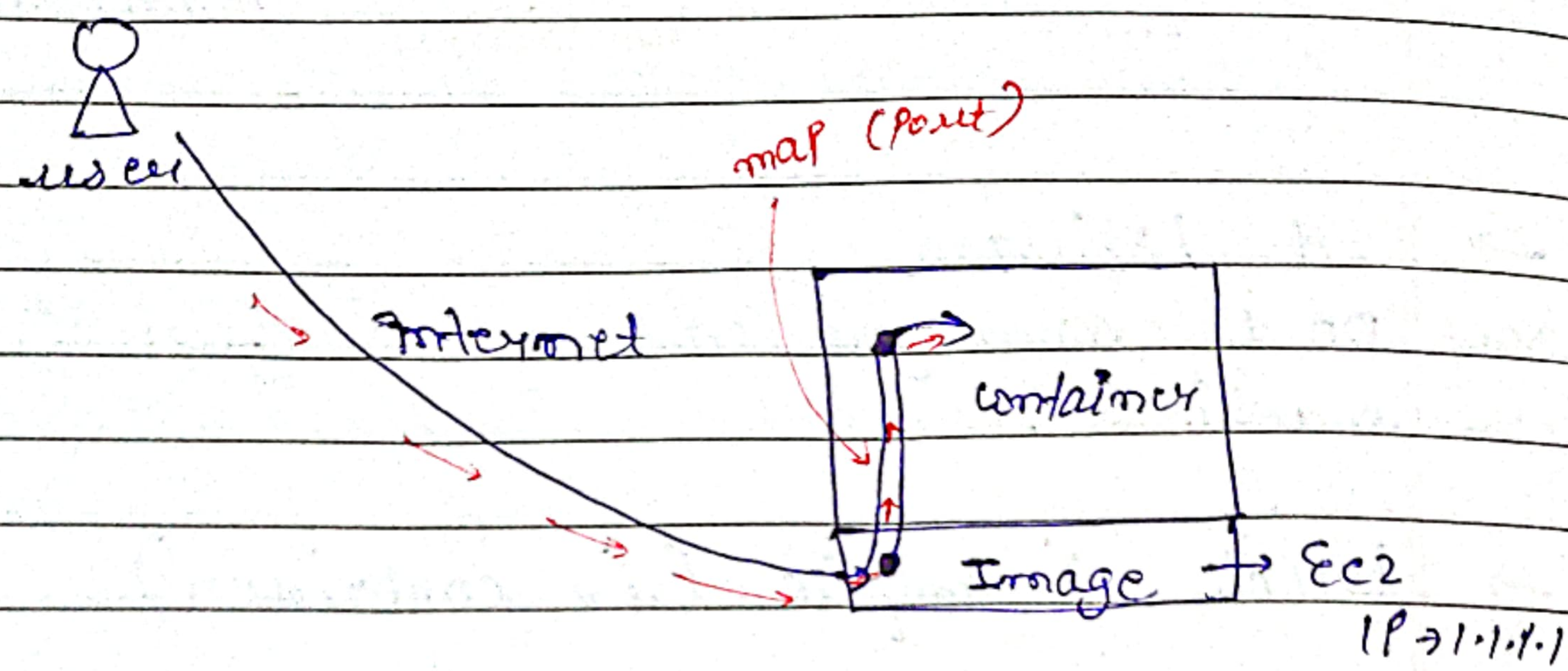
→ docker volume create <volume name>
→ docker volume rm <volume name>
→ docker volume prune
{ it removed all unused docker volume}

(all information)
→ docker volume inspect <volume name>
→ docker container inspect <container name>



Date / /

Docker Port Expose →



→ Let suppose if any user want to access the service available on container.
→ But container doesn't have IP address.
→ So we will map the EC2 port so to container port so, because EC2 have its IP address
→ So user can access EC2.

★ Steps

→ Login into AWS account create one Linux instance.
→ Now go to putty → login as → ec2-user
→ sudo su
→ yum update -y
→ yum install docker -y
→ service docker start
→
↳ docker run -td --name techserver -p 80:80 ubuntu

(daemon)

host
container
port
map

→ docker port techserver (to see expose port)
o/p → 80/tcp → 0.0.0.0/80

→ docker exec -it techserver /bin/bash.
exec → go inside container & create new process

→ apt-get update
→ apt-get install apache2 -y

→ cd /var/www/html
→ echo "This website is host on Docker!" > index.html.

→ service apache2 start.

Difference b/w docker attach & docker exec?

→ docker exec creates a new process in the container's environment while docker attach just connect the standard input/output of the main process inside the container to corresponding standard input/output error of current terminal.

→ docker exec is specially for running new things in a already started container, be it a shell or some other process.

What is the difference b/w `expose` and `publish` a docker?

↳ Basically you have three options :-

1) Neither specify `expose` nor `publish (-p)`
2) only specify `expose`.
3) specify `expose` and `-p`.

(1) If you specify neither `expose` nor `-p`, the server in the container will only be accessible from inside the container itself.

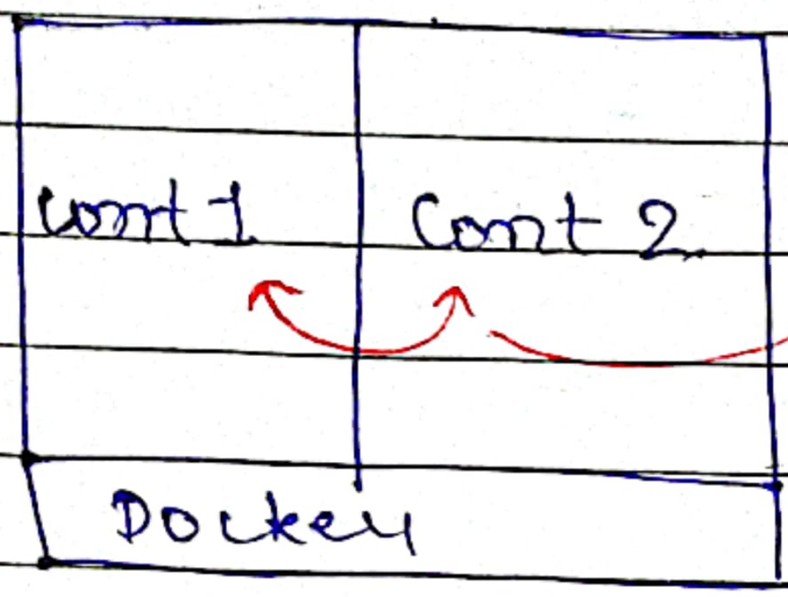
(2) If you `expose` a port, the service in the container is not accessible from outside docker but from inside other docker containers, so this is good for inter-container communication.

(3) If you expose & -p a port, the service in the container is accessible from anywhere, even outside docker.

If you do `-p` but do not `expose` docker does an implicit expose. This is because, if a port is open to the public, it is automatically also open to the other docker containers, Hence `-p` include `expose`.

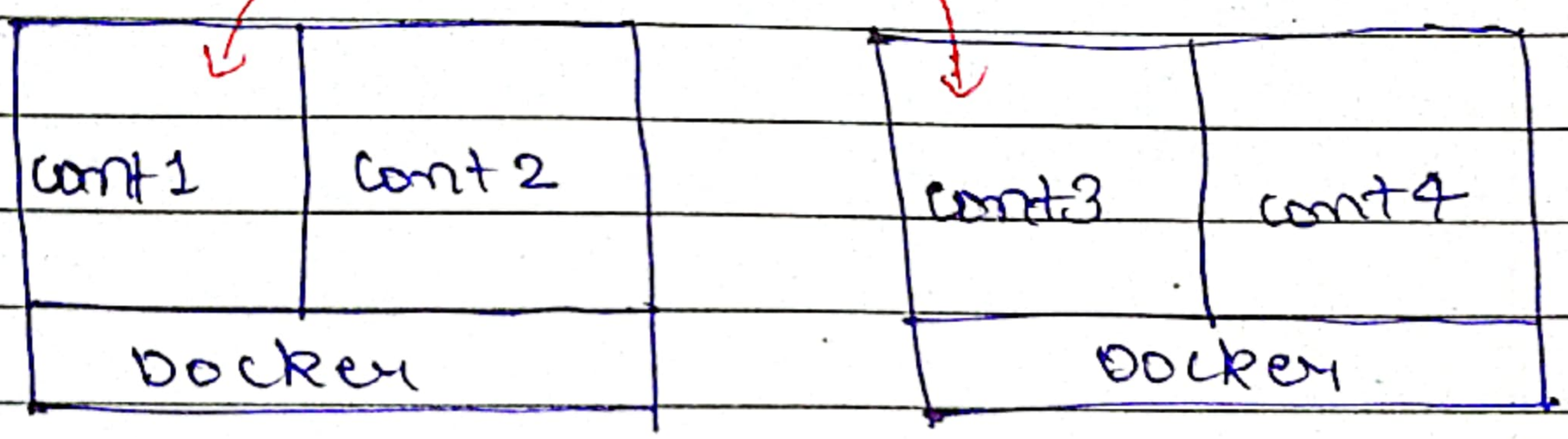
A-1
Page
Date / /

(1)



without expose they will publish they will communicate.

(2)



(3)