Manage Data in Docker

By default the data of a container is stored on an writable container layer. This means that —

(i) When a container is deleted, and no layer is now then the data is not persisted to that container and can't be used by other process as well.

(ii) An writable container layer is tightly coupled with the host machine. It is very difficult to move data from other containers.

(iii) Storage driver is responsible for writing in writable container layer. Storage driver use union file system running on linux kernel. This extra abstraction decrease the performance of container while docker volume directly work on the host file system.

So to persist the data we use 2 mechanism —

(i) Data Volume { docker volume }
(ii) Bindmounts.

In window we use named pipe to persist out data and use tmpfs mounts for linux system.

The difference between these mechanism are based on where data where it live —

(i) Volumes — volumes are stored as a part of docker host system which is managed by Docker & present at C:\ProgramData\docker\volumes in linux. The non docker process should not change the file we have stored in volume. Volume are the best way to persist data.

(ii) Bind mounts - They are present at any where in ~~docker~~ host system.
In Bind mounts the non docker processes can modify or change at any time.

(iii) tmpfs mounts - They are present at ~~non docker~~ host's memory only. ~~They are not write~~ Only system information are used for writing.

(*) Docker volumes are different from ~~Bind mounts because~~ docker volumes are managed by docker and are isolated to the core functionality of host system.

Docker volumes :
(i) Mechanism used for persisting data created and used by containers.
(ii) Docker volumes are created explicitly as well with Docker create volume command and can be created at the creation of containers service.
(iii) Multiple containers can have single volume as well. If a container is not utilizing a volume, the volume will still not be deleted.
(iv) Volumes can be named and anonymously named by docker.
(v) Volume can used volume driver to store data at cloud providers.

Good use cases of volumes -
(i) Sharing data among multiple running containers. Containers that sharing data can perform read, write operations.
(ii) It helps in decoupling of configuration of docker host system from the container.
(iii) It is used to store data at cloud provider rather than locally.
(iv) It help to backup, migrate from one docker host to another which is the best case.
(v) When an application requires high performance I/O in Docker desktop.
(vi) When an application requires behaviour of fully native file system on docker desktop -

How docker volumes are different from Bind mounts :—

(i) Volumes can migrate and backup easily.
(ii) Volumes are managed by Docker CLI command and Docker API.
(iii) Multiple containers can shared data more safely in volumes than Bind mounts.
(iv) Volumes have storage volume which is is used to store data at cloud provider rather than locally.
(v) Volumes have higher performance than bind mounts in mac and windows.
(vi) Works on both windows and linux.

Volumes are better option than writable container layers as —

(i) Whenever we create volume for a container it doesn't change the size of container.
(ii) The volume which we made is present outside container lifecycle.

```
docker volume create weshvolume
docker volume ls
docker volume inspect weshvolume
docker volume rm weshvolume
```

start a container with volume —

```
docker run -d \
  --name devtest \
  --mount source="pwd/", target=/app \
  nginx:latest
```

using -v

```
docker run -d \
-- name = devtest \
-v pwd1:/app \
nginx:latest
```

Bind Mounts :

(I) Bind mounts have less functionality the docker volumes.

(II) When we use bind mount the directory or file of the host file is mounted to the container.

(III) The file directory is referenced by its full path of host machine.

(IV) Bind mounts are on demand created if not exists.

(V) Bind mounts are performant but they rely on docker host machine file system which have specific directory.

* One major problem with Bind mounts are that we change the host file system through running process of containers which include creating, deleting and updating important file directories. This powerful ability has security implication and this impact the non-docker process in that container.

```
docker run -d \
-it \
--name devdest \
--mount type="bind", source=/op/target, target=/app \
nginx:latest
```

tmpfs Mounts – (I) tmpfs mount are non persisted on disk either in host or within a container.

(II) They are used by container during lifetime of that container to store sensitive data.

(III) Used by docker swarm to mount sensitive information on swarm containers.

Named Pipes – (I) It is used to communication between docker host and container.

(II) It is used to run third party tools inside a container.

Use case tmpfs mount — docker

(i) When we don't want to persist our data within a container on host.

(ii) If we want to store sensitive information.

(iii) If we want to protect the performance of our container when we want to write large volume of non-persistent data.

Limitation —

(i) Only available for linux.

(ii) Once tmpfs is used, the permission which we have used also be used.

(iii) We can't share tmpfs mounts among different containers like Bind mount and volumes does.

① Virtualization
② Utility Computing
③ SOA { Service Orien

Multiprogramming: having more than one program (Processes) in RAM.

Multi tasking : logical extension of multiprogramming. {multi tasking is like multi programming ka hone lagi hai}
Multi processing : having than one processors.

Virtualization is categorised as component.

① Full virtualization
② Para virtualization
③ Partial
④ Hybrid

OS virtualization is known as Containerization virtualization
↓
holding software & essential utilities

Binary and Library
Container is a light weight version of virtualization.
Orchestration is container tools use to manage multiple containers.

Virtualization types, symantec & asymantec processes

memory v/s storage   Project & CC ← (Vivek Dikshit) Project head of BIQ AI

Que 1 Difference b/w multiprogramming, multi tasking, multi
Que 2 What are the different kind of processors
Que 3 Distributed system

Multiprocessing – done by Tightly Coupled and Loosely Coupled.

Tightly Coupled → Share address bus & data bus → contention of process because they are accessing same storage.

Loosely Coupled → residing to a different geographical area.

Distributed

- Client Server
- Grid
- Cluster
- Cloud

wide

(i) LAN   (i) WAN
(ii) multi admin.   (ii) single administration domain

System level virtualization

- Full
- Para
- Partial

Full → virtualising the entire hardware
Partial → virtualising or some part of system i.e. RAM

address space – where the process is present

Containers / Containerization → OS virtualization
Here processes are isolated. They can't see each other processes.

Que What is the difference b/w chroot, cgroups, namespaces & jails ?

- chroot → root directory change karta hai → less isolation; changes root directory
- cgroups → give resources to applications; diffs in allocation of resources
- namespaces → more isolation; process ko apparent ...
- jails → most isolation

NDFS → windows file system

Que File system
BSD → Berkley Software Distribution

Que Windows me jo container hote hai Hyper V containers, why microsoft moves from micro v container to hyper v container.

Container – It is a software package which have software & libraries to run.

why container (to solve issue of incompatible issue) software or Hardware
↳ use hue kya
↳ container platform independent

Why use containers
(i) Isolation & allocation
(ii) Server, Websolution
Provisioning & Costs
(iv) Performance
(v) Portability
(vi) Delivery,
(vii) Scalability
(viii) Automation

Que How provisioning & costs are decreased in containers?

Popular Container → Docker (most popular), Podman, LXC {linux containers}, Vagrant

Que How ~~it~~ containers are ~~[illegible]~~ suitable for micro service?

name space – a feature that allows processes to use system resource they need but separately from other processes.

↳ focuses on infrastructure architecture

Serverless Architecture → billing is much more than serverless architecture.

Serverless Architecture → billing is based on paying for time of your code is running
↓ focus on development & coding

Monolithic	SOA	Microservice Architecture
↓	↓	
(i) tightly coupled	(i) little bit loosely coupled	(i) highly loosely coupled
(ii) faster	(ii) Auditing servers.	(ii) Service oriented approach
(iii) less auditing servers.	(iii) Used in banking application.	(iii) Maintainability is easier

orchestration - management of services

Docker - tooling & a platform {made on GO language} - works on Linux Kernal

(i) simple

(ii) Complementary tool - Docker CLI, Docker Compose & prometheus as well as various plugins for storage

Docker not good for - high performance or security, applications which have GUI, applications which have limited features.

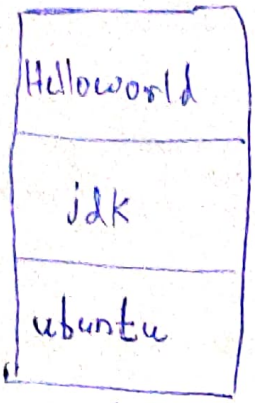
Que Why containers are not suited for using such GUI feature?

Docker - follows client server architecture

docker run -it --name = ubicon ubuntu
bt

Docker host → Docker daemon

How to manage data in docker?

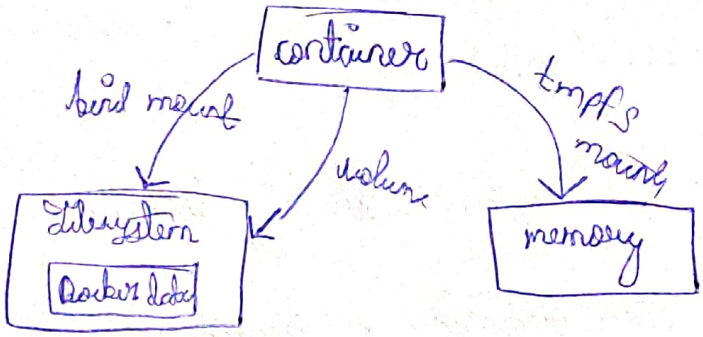


⇒ container writable layer

{ Volumes are much more secure than bind mounts.

(i) doesn't persistent
(ii) It effects the performance,
(iii) writable layer is tightly coupled to host machine.
(iv)

2 concepts → volumes and bind mounts
tmpfs mounts



when we have to share data b/w 2 containers
↓
when containers are used?

Docker volumes can be create — (i) After container is made
(ii) Before container is made

Anonymous → is anonymous, default name is given by docker,

uses cases of volumes — (i) sharing (ii) performance (iii) backup
(iv) fully native file system (v) store container data on a remote host or cloud provider,
(vi)

project micro service

docker run -dt --name=devtest
--mount source=myvol

Create volume

-v myvol2:\app

create volume - docker volume create my-vol
↳ name of volume

list volume - docker volume ls

inspect a volume docker volume inspect my-vol

remove volume - docker volume rm my-vol

```
docker run -d \
--name devtest \
--mount source=myvol2,target=/app \
```

```
docker container stop devtest
docker container rm devtest
docker volume rm devtest
```

(/var/lib/docker/volumes/on Linux)

Q What are configuration file, give a real life example

tmpfs - used to store credentials or secret data.

Volumes performance are better than bind mount.

```
docker run -d \
--name devtest \
--mount source=myvol2,target=/app \
nginx:latest
```

```
docker run -d \
--name devtest \
-v myvol2:/app \
nginx:latest
```

Bind Mounts - create folder outside the docker. It's used through API.

Uses of Bind Mounts

(i) sharing source code b/w a development environment.

pwd - present working directory

why manage data? - (i) Data doesnot persist.

Qu (i) Run a container with docker volume
(ii) Inspect whether the volume is attach with container or not
(iii) Create a file inside a volume.
(iv) Stop container, Remove container, Remove volume

(i) create volumes

docker run -d --name=vjesh --mount source=vjesh,target=/newfol nginx:latest

docker volume create vjesh

(ii) ~~docker~~ volume ~~inspect~~ ~~vjesh~~
docker inspect vjesh;

(iii) docker exec -it vjesh bash
apt-get update
example.txt
ls /var/lib/docker/volumes

(iv) docker container stop vjesh
docker container rm vjesh
docker ~~container~~ volume rm vjesh

(i) docker run -d \
--name vjesh \
-v myvol3:/app \ → folder
nginx:latest

Q) Create a docker file i.e using ubuntu OS & is capable of running a Hello world java program stored in the same folder as that of the docker file. After creating the img from the docker file push the img into your repos on docker hub. Thereafter pull img & run the container.

①

```
docker run -d --name=ujesh
--mount source=ujesh,target=/newfol
```

(i) mkdir images
(ii) cd images
(iii) notepad Hello.java
(iv) notepad dockerfile

```
FROM ubuntu:latest
RUN apt-get update jdk
apt-get install -y default-jdk
COPY Hello.java /app/
WORKDIR /app/
RUN javac Hello.java
CMD ["java", "Hello"]
```

(v) docker build -t testimage:1.0 .

(vi) docker run -dt --name=newcon testimage
Hello World.

(vii) docker

tmpfs mounts:
① non persisted on disk

Pipe - mechanism used for inter-process communication

How IPC is done in linux system? - pipe

In Linux

fork()
↓
inside child process from parent

named pipes - ① used for communication b/w docker and container.
↓
② used for running third party tool inside

alternative for windows system.

tcp/udp → communication related b/w host and container

Docker Networking

① Helps to connect to and communicate with each other or to non-docker workloads.

When you run a container, automatically (by default) a driver is associated which help in communication.

In TCP we sent the data, and if lost, again sent.

* Each container has DNS resolution.

TLD - top level domain server

client

Publishing flag - mapping the container port with host port

Network Drivers -

| ssi of mac & ip address |
↓
(i) permanent identity
(ii) can't be changed

(I) Bridge driver - (i) default driver
(ii) Used when a single desktop have multiple containers

(II) Overlay - (i) to communicate across nodes.
(ii) connect multiple docker daemons together.
(iii) connect different hosts

(III) host - (i) remove network isolation b/w container & docker host

(IV) ipvlan → for ip address
(V) macvlan → for mac address

Qu ipv6 vs ipv4

Qu How DNS work

Bridge driver → default bridge
→ user defined custom bridge network

(I) docker network create my-net

(II) docker network connect my-net my-nginx
(my-net ↳ name of network; my-nginx ↳ name of container)

↑
OR
↓

(III) docker ~~create~~ run --name my-nginx
--network my-net \
--publish 8080:80 \
nginx:latest

(IV) docker network disconnect my-net my-nginx

(V) docker network rm my-net

Difference b/w user defined & default bridge

(i) In user defined bridge automatic DNS resolution b/w containers
default bridge uses ip address

(ii) User defined provide better isolation

(iii) default bridge network share environment variables.
default bridge can be configured.

approaches —

(i) volumes
(ii) docker compose
(iii) swarm

Qn: Check the available network in your system first.

Qn: Create a custom network of custom type.

Qn: Retrieve the information of bridge network drive.

Qn: Create a ubuntu container again which the bridge network, also check the details of container created?
Run another container & inspect the bridge network again.
Connect the container to user define container, try to ping the container which are on default bridge & customise network bridge.

(i) docker network ls
(ii) docker network create --driver=bridge my-network
(iii) docker network inspect bridge
(iv) docker run -dt --name=nettoy1 ubuntu
(v) docker run -dt --name=nettoy2 ubuntu.
(vi) docker inspect container nettoy1
(vii) " → details of ip address
nettoy2

(IX) docker exec -it nettry1
apt-get update
apt-get install
ping ipaddress nettry1

ping - checks that packet is reaching or not.

connecting container to network :

(X) docker ~~run~~ ~~--name~~ network connect my-network nettry1

(XI) docker network connect my-network nettry2

(XII) docker inspect container nettry1 ← ip address
" nettry2 ← ip address

(XIII) docker exec -it nettry1
apt-get update
ping ipaddress → nettry2

(XIV) docker network disconnect my-network nettry1

Q. Hai bhi ecommerce pl → micro services
(i) what kind of server they are using ?
(ii) server
(iii) how security is allowed
(iv) how are these being orchestrated.

Docker Compose → (i) docker local (orchestration tool)
(ii) Used to manage multiple containers

(ii) tool for defining and running multi-container Docker applications.
YAML file stores configuration of application
(Compose.YAML) → file (format key value pair)

(iv) Used in production, staging, development, testing & Continuous integration (CI) workflow.

Features → (i) Isolated Environment (i) Isolate at feature level
(ii) Isolate at project level
(ii) Preserve Volumes
(iii) Only recreate that containers that have changed.
(To make process much more faster)
(iv) Support variables and moving a composition b/w environments.

```
mkdir composetest
cd composetest
```

(i) app.py
(ii) requirements.txt
(iii) Dockerfile
(iv) compose.yaml

redis database → in-built database.

holiday on friday ← wednesday - 11 to 12 | Not part of midterm | computer - 65536 ports 0 to 65535

concept
~~feature~~ of dockerswarm → nodes → any engine which have docker swarm

manager node — worker node

responsible for man ← manager node
→ service & task

worker node — responsible for execution of task

service command - determine ~~the what will be task be deployed~~ the defination {tell us what has to done}
↓
lead to creation of task

service command → replicated, global

* By default a manager node act as a worker node

Ingress Load Balancing

Nomad, Kubenative's, Marathon

Swarm features -

1. Cluster management integrated with docker engine
2. decentralised design
3. declarative service model
4. scaling
5. desired state reconciliation
6. service discovery
7. Load balancing - internal & ingress load balancing
8. service host networking
9. Secure by default
10. Rolling updates

Q) What are different kind of load balancers in AWS

1. Application load balancer
2. Network load balancer
3. Gateway load balancer

(IV) Classic Load balancer

need 3 Linux machine
protocol all udp, tcp

* Generally we don't prefer single manager.
Managers are always in odd no.

N of manager nodes --tolerate--> $\frac{(N-1)}{2}$ managers

* We can't have single worker node.

init → in OS
it means the very first process
or we can say that parent process

docker node promote <node name>

docker node demote <node name>

swarm
→ scaling
↳ same node
→ replicas
↳ has to go on different node

} They both are different and have difference

3 que command based
1 - 4 marks
2 - 10 marks
↳ scenario based

Docker Swarm

(I) AWS - 3 Linux Virtual machine
↳ Security Protocol → All udp
→ All tcp

(II) Install Docker
↳ sudo apt-get install docker.io

(III) docker swarm init -advertise -addr <Manager 1>

(IV) docker info

(V) docker node ls

(VI) docker swarm join - - - - - - - - W1

(VII) docker swarm join -token

(VIII) docker swarm join W2

FROM COPY, CMD

Q Write a ~~write a name~~ create a docker container with Dockerfile as mycontainer

① nano mycontainer

② Write instructions

FROM

WORKDIR . www

docker build -t myimage -f my-container

↓ tag ↓ flag file

Q Start a docker container named frothand using any webserver image and attach to a network mynetwork. ~~Use network~~

Ans docker ~~network~~ network create type=bridge my-net

docker run -dt --name=frontend nginx

docker network connect my-net frontend

Q Provide an explicit tag to container, the tag names are Hi and Hello

~~docker build -t ujesh -f Hi~~

Ans docker run -dt --name=frontend -t mycontainer:Hi

Q. Pull a specific docker image from docker hub & print its exposed ports!

Ans
```
docker pull alpine:3.0.6        { by default it starts on exposed }
docker run -dt -p 1000:8080 alpine
```

Bind Mounts —

(i) Have limited functionality as compared to Docker volume.

(ii) When we use bind mount a directory on the host machine is mounted into a container

(iii) In this the file is referenced by its full path on the host machine.

(iv) Created on demand if not exists.

(v) Very performant but rely on host machine's file system.

One of the side effect of Bind mounts is that it can change the host file system via process running in a container which includes updating, deleting system important files. This powerful ability has security implication which impact the non docker process running in a host system.

Kubernetes :- deployment & management of containized application / docker - run multi container based application on single host.
↓
not a orchestration tool, just a tool

* product of google / open source

* maintains by automatic deployment & management of containerized application.
* CICD not possible
* Has its own persistent & non persistent storage {hard disk or main memory}
* capable of integrated with every platform which makes it very popular.
* It is not a PaaS {not traditional PaaS}
* Don't provide database
* Has its own environment
* It is not opinionated about workload

In AWS → Elastic Container Service {container service} (ECS)

In AWS → Elastic Kubernetes Service (EKS)

PaaS → anchored, integrated

traditional PaaS

Ques Difference b/w Stateful & Stateless applications?

Kubernetes Capabilities -
(i) Automated rollouts & rollbacks
(ii) Storage orchestration
(iii) Horizontal scale
(iv) Automated Bin packing
(v) Secret & configuration management
(vi) IPV4/IPV6 are supported
(vii) Batch execution
(viii) Self healing
(ix) Self discovery & load balancing

Ques Diff. b/w DaaS & NAS & SAN

Ques What are 3 different kind of storages available in AWS azure = ?

Block, File, Object
↓ ↓ ↓
EBS EFS S3

api → set of function

Architecture of Kubernetes

master node → management

control plane & worker plane

(i) API server

kube-api server - implementation of Kubernetes API

(ii) etcd - highly available distributed key value store repository
↘ stores configuration of nodes & desired states

(iii) Scheduler - to ensure which workload has to distribute on nodes.

Overlay - in details {with cases}
docker compose features
Container linking
Orchestration - why
→ swarm → how to create manager node
→ kubernetes {architecture}

swarm vs kubernetes
↓
objects
→ replica set
v/s
→ deployment → types of deployment strategy {case base scenario}

Kubernetes services & types
↳ why this abstraction required

Kubectl {cmd tool line}
↳ 3 commands & difference

ECS v/s EKS
↓
launch types

case base question on SOA, monolithic & microservice
docker or microservice compatible kyu nahi hai or kyu hai

---

single pod deployments -

imperative - command me hi define kar diya ki kya karna hai.

Kubectl [command] [type] [name] [flags]

Imperative object configuration - it contain configuration in form of YAML or JSON file & has advantage as it can audit

{problem - configuration in file is made by developer & these changes are not been reflected by developer} ← changes are not reflected in configuration file

Declarative configuration → updates are done by this command. {

`kubectl get deployment my-dep` → command to check status

```
kubectl create -f replicaset.yaml
kubectl get pods
kubectl get rs
```

replicaset - used to meet the desire state

Kubernetes orchestration — 2 planes in architecture

- → control plane → master node (managing the entire cluster)
  - cloud controller manager: integrate with any cloud vendor
  - kube api server (frontend of control panel)
  - kube controller manager: take decision of the cluster
  - etcd repository: key value pair repository; contains configuration of whole cluster
  - kube scheduler: schedules which task has to be executed first
- worker plane → contains more than one node. It can be a physical or virtual machine

Worker

- Pods (smallest deployment unit) → a single node has multiple pods & also have multiple container in a single pod (two multiple containers)
- container engine (docker generally) →
  - * capable of integrating with any of container engine
  - * pull images & create container
- kubelet →
  - * states the current state
  - * checks the status of nodes & tell to the control plane
- kube proxy →
  - * responsible for maintaining the networking information

(i) CI/CD not possible

(ii) database not possible

(iii) PaaS (not paas) → because it does only deployment

Object - koi bhi variable or data structure used in kubernetes.

Kuber. obj: → Pods, namespaces, ~~replica~~ replica sets, deployments

replica sets → creates replica of ~~replica~~ pods (depending upon desired state)

Labels - identification technique for pods or for resource grouping

Deployments - (i) we always make deployments not replica
(ii) has more feature than replica
(iii) Handle rolling ~~update~~ updates, but replica can't
(iv) Suitable for stateless applications.

Deployment strategies :
(i) Replay
(ii) Recreate
(iii) Scale up manually or

(a) Recreate → simple setup, application version completely recreated
(b) Rolling → simple setup, rollback takes time, can't control traffic distribution.
(c) Blue green → easy to roll back & roll up., expensive, stateful application handle useful for stateful application
(d) Canary → not expensive, slow roll out, fast rollback multiple
(e) A/B testing (split testing) → used when we are designing user interface
(f) Shadow → used for various purposes