Docker Container Networking and Security

- starts at 9:05 pm

Agenda

1. Storage
    1. Ephemeral and Persistent Storage
    2. Docker Storage
        1. Why? (Importance)
        2. Types of Docker Storage
        3. Storage in Swarm (Mini Project)
2. Docker in Docker
3. Networking
    1. Types of Networks

type = worker.

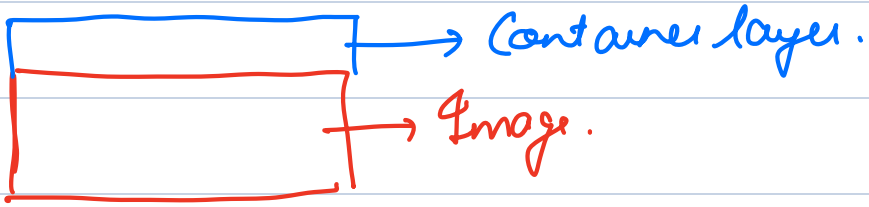
-- Constraint.

Introduction to Docker storage.

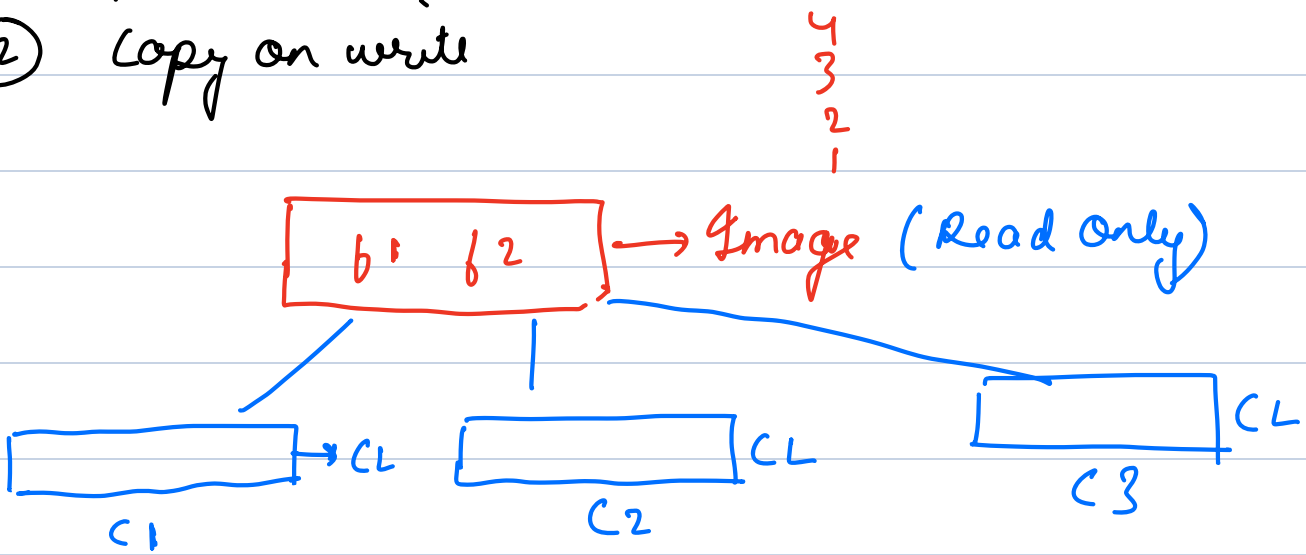
Ephemeral storage
Persistent storage.

temporary & lasts only for the container lifetime.

Container layer.



① Ephemeral
② Copy on write



f1. copy

③ Isolation.

② Persistent storage.

allows data to outlive containers lifecycle.

Docker storage.

methods and mechanisms used to persist data generated and used by Docker Containers.

① Volume
② Bind Mounts
③ tmpfs.

Importance of Docker storage.

① Efficient Resource Management
② Data Persistence.
③ Scalability and Flexibility.

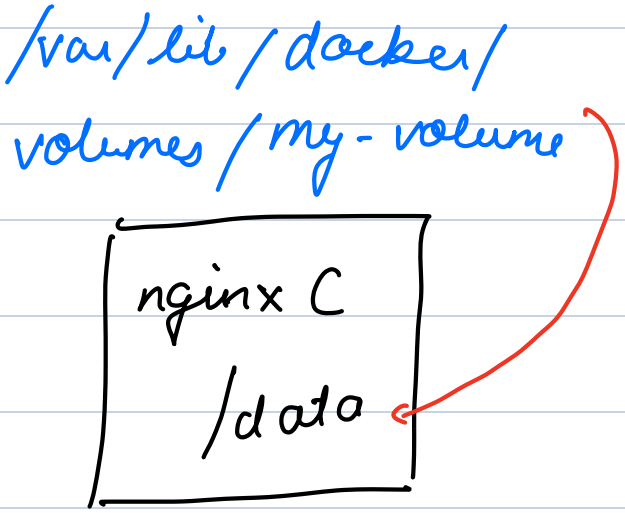
④ High Availability and Reliability.

① Volumes.

/var/lib/docker/volumes/abcd

Create a volume

docker volume create my-volume



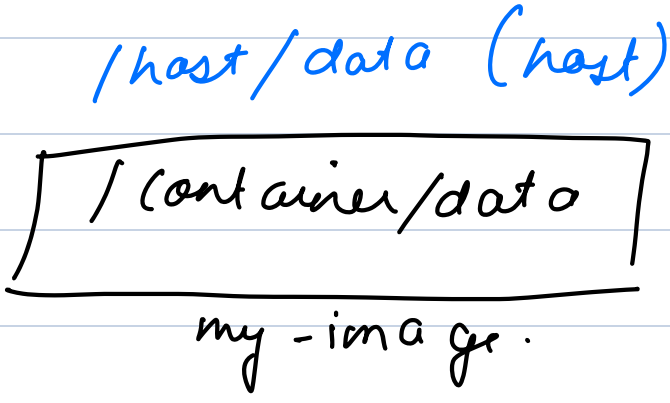
Use the volume

docker run -v my-volume:/data nginx

② Bind Mounts.

/abcd/pqr

Less Portable.



docker run -d -v /host/data:/container/data my_image

Feature	Volumes	Bind Mounts
Definition	Managed by Docker. Stored in Docker's filesystem on the host.	Directly maps a host directory or file to a container path.
Storage Location	Stored in Docker's storage location (e.g., `/var/lib/docker/volumes/` on Linux).	Stored at any specified path on the host system.
Portability	Highly portable across environments (e.g., from dev to prod).	Depends on the host system directory structure, reducing portability. **Portability Issue:** When you move or deploy the container on another machine, the host path `/path/on/host` may not exist or be different, causing potential failures or errors.
Security	More secure, as volumes are isolated from the host filesystem.	Less secure; can unintentionally expose sensitive host data to the container.

→ tmpfs mount

tmpfs creates temporary filesystem in the hosts memory (RAM)

→ operations that need high performance.
→ Reduced disk writes.

```
docker run -it --rm --tmpfs /app/cache:size=64m ubuntu bash
```

Demo Creating and Managing Docker volumes.

shared-volume

C1 → write some data to shared-volume

C2 → read some data from "

Break → 10:25 pm

Create a Volume:

docker volume create shared-volume

Inspect a Volume:

docker volume inspect shared-volume

List Volumes:

docker volume ls

Launch the First Container (Writer)

docker run -it --name writer-container -v shared-volume:/app/data ubuntu bash

Write some data to the mounter volume

echo "This data is present on shared-volume!" > /app/data/message.txt

Launch the Second Container (Reader)

docker run -it --name reader-container -v shared-volume:/app/data ubuntu bash

cat /app/data/message.txt

Demo for Backing up a volume. Shared-volume.

① Create a directory where you want bkp

docker-backup.

② use a temp. container

Shared-volume → /data

message.txt

③ use a Bind mount.

docker-backup → /backup.

④ Backup of data folder.

Create a backup directory on the host

mkdir ~/docker-backup

Use a temporary container to back up the contents of the volume.

docker run --rm \

-v shared-volume:/data \

-v ~/docker-backup:/backup \

ubuntu tar -czf /backup/shared-volume-backup.tar.gz -C /data .

Interview Question.

Why do we take backup of volume using a temporary container and not directly copy the /var/lib/docker/volumes/

folder?**

The primary reason we take backups of Docker volumes using a **temporary container** instead of directly copying files from the `/var/lib/docker/volumes/` folder is to ensure **data consistency** and **isolation** while maintaining the integrity of the volume's data.

1. **Data Consistency**

Active Containers: If a volume is being used by a running container, the data inside `/var/lib/docker/volumes/` may be actively modified.

This will result in inconsistent or incomplete backups, especially if files are being written or updated during the backup process.

Temporary Containers: By mounting the volume to a temporary container, you can ensure the data being backed up is in a stable state, as the container can explicitly lock or manage access to the volume during the backup.

2. **Portability**

The location of `/var/lib/docker/volumes/` can vary depending on the Docker installation, operating system, or custom Docker configurations.

Relying on this path makes your backup process less portable.

3. **Permissions issue**

The `/var/lib/docker/volumes/` directory often requires root-level access. Directly accessing or modifying this directory can pose security risks or lead to accidental permission errors.

4. **Avoiding Docker's Internal Structure**

The `/var/lib/docker/volumes/` folder contains not only the data stored in the volumes but also Docker’s internal metadata and configuration.

You are building a Dockerized web application that uses a **PostgreSQL database** to store user data. The application has the following requirements:

1. **Data Persistence:** The database data must not be lost when the container is restarted or removed.

2. **Portability:** You need the ability to back up and restore the database easily.

3. **Host Accessibility:** Developers occasionally need direct access to the database files for debugging or manual modifications.

Question:

Which Docker storage type would you use in this scenario, and why?

- **A)** Ephemeral Storage (container writable layer)

- **B)** Volume

- **C)** Bind Mount

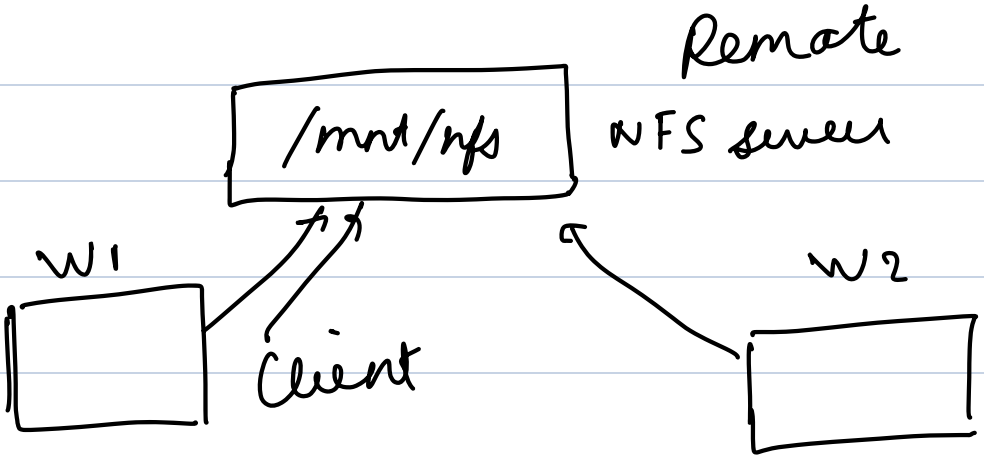
- **D)** tmpfs Mount

Correct Answer:

B) Volume

Introduction to Storage in a Swarm Cluster

→ NFS



Overview

→ (1) Set up an NFS server on manager node.

(2) Mount the NFS share on all swarm nodes. → unmount

(3) Create a volume from the NFS share

(4) Deploy a stack using the volume.

Step 1. Set up NFS on manager node

sudo apt update

sudo apt install nfs-kernel-server

Create a Directory to be shared via nfs

sudo mkdir -p /mnt/nfs_share

sudo chmod -R 777 /mnt/nfs_share

Configure NFS exports

sudo vi /etc/exports

/mnt/nfs_share *(rw,sync,no_subtree_check,no_root_squash)

Apply the changes

sudo exportfs -a

Restart the NFS server

sudo systemctl restart nfs-kernel-server

Step 2. Mount the NFS share on worker nodes.

{ only for NFS demo please unmount from worker nodes }

Install NFS client on both worker nodes

sudo apt update

sudo apt install nfs-common

Mount the NFS share on each worker node

sudo mkdir -p /mnt/nfs_share

sudo chmod -R 777 /mnt/nfs_share

sudo mount -t nfs 51.20.95.206:/mnt/nfs_share /mnt/nfs_share

umount /mnt/nfs_share (use sudo)

Step 3 → Create a volume

```
docker volume create \
 --opt type=nfs4 \
 --opt o=addr=51.20.95.206,rw \
 --opt device=:/mnt/nfs_share \
 nfs_volume
```

Step 4 → Create a Stack

docker-compose.yml file

```
version: "3.8"

services:
 nginx:
  image: nginx
  deploy:
   replicas: 5
   resources:
    limits:
     memory: 50M
  volumes:
   - nfs_volume:/data
```

```
volumes:
  nfs_volume:
    external: true
```

```
docker stack deploy -c docker-compose.yml my_stack
```