Docker Volume Service

Agenda

In this session, you will learn about:

- Understanding Docker Volume
- Understanding various Volume Plug-ins
- Create and manage volumes
- Start a container with a volume
- Populate a volume using a container
- Use a read-only volume
- Share data among containers

Storage for Containers

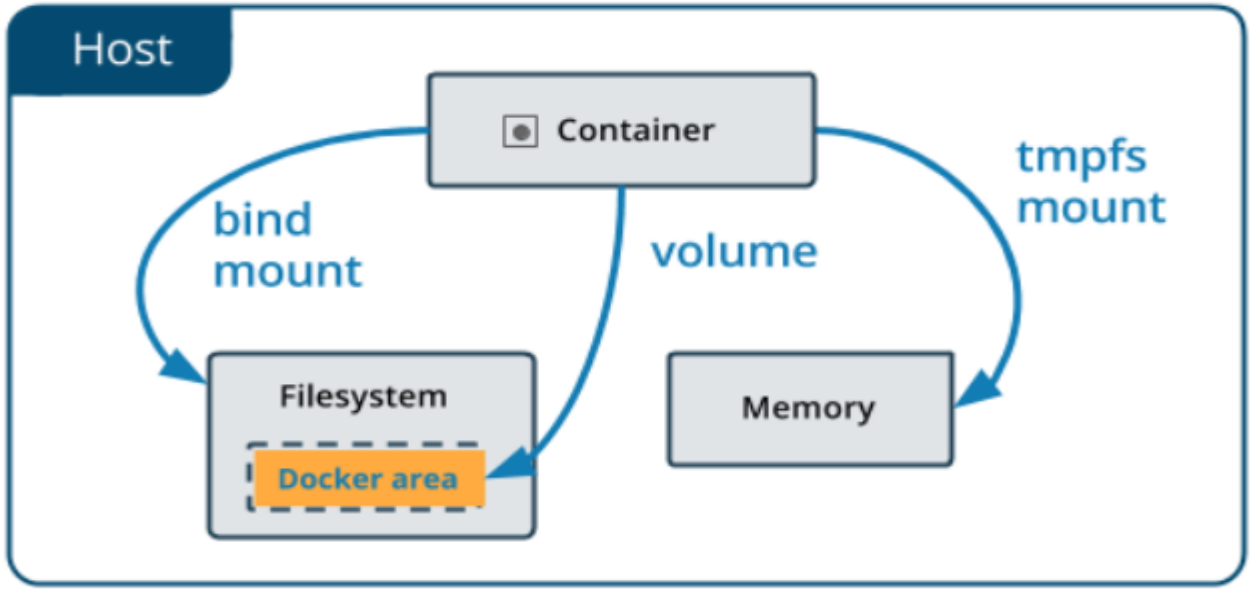
- Containers often requires storage for **capturing/saving** data beyond the container life cycle.
- Docker volume service is the best option to keep data for future use.

Manage data in Docker

- By default all files created inside a container are stored on a writable container layer.
- The data doesn’t persist when that container no longer exists.
- A container’s writable layer is tightly coupled to the host machine where the container is running.
- Writing into a container’s writable layer requires a storage driver to manage the filesystem.
- The storage driver provides a union filesystem, using the Linux kernel.
- This extra abstraction reduces performance as compared to using data volumes, which write directly to the host filesystem.
- Docker has two options for persisting data
  - Volumes
  - Bind Mounts

Docker Volumes – Life Cycle

- Volumes are often a better choice than persisting data in a container's writable layer
- Volume does not increase the size of the containers using it
- Volume's contents exist outside the lifecycle of a given container.



Docker Volumes – Life Cycle…

- **Volumes**
  - Stored in a part of the host filesystem and managed by Docker
  - Location - /var/lib/docker/volumes/
  - Non-Docker processes should not modify this part of the filesystem
  - Volumes are the best way to persist data in Docker
- **Bind mounts**
  - Stored anywhere on the host system
  - Non-Docker processes on the Docker host or a Docker container can modify them at any time
- **tmpfs**
  - Stored in the host system’s memory only, and are never written to the host system’s filesystem.

Volumes

- Created and Managed by Docker.
- Run **docker volume create** command to create volume.
- To remove unused volumes use **docker volume prune**.
- When you create a volume, it may be named or anonymous.
- Volumes can be more safely shared among multiple containers.
- Volumes are easier to Backup or Migrate than the bind mounts.

Volumes…

- When no running container is using a volume, the volume is still available to Docker and is not removed automatically.
- Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.

Bind Mounts

- Available since the early days of Docker.
- Bind mounts have limited functionality compared to volumes.
- When you use a bind mount, a file or directory on the host machine is mounted into a container.
- The file or directory is referenced by its full path on the host machine.
- The file or directory does not need to exist on the Docker host already.
- It is created on demand if it does not yet exist.
- Bind mounts are very performant, but they rely on the host machine’s filesystem having a specific directory structure available.
- If you are developing new Docker applications, consider using named volumes instead.
- You can’t use Docker CLI commands to directly manage bind mounts.

tmpfs Mounts

- A tmpfs mount is not persisted on disk, either on the Docker host or within a container.
- It can be used by a container during the lifetime of the container, to store non-persistent state or sensitive information.
- For instance, internally, **swarm** services use tmpfs mounts to mount secrets into a service's containers.

Use cases for volumes

- When you want to store your container's data on a remote host or a cloud provider, rather than locally.
- Sharing data among multiple running containers.
  - Multiple containers can mount the same volume simultaneously, either read-write or read-only.
  - Volumes are only removed when you explicitly remove them.
- When the Docker host is not guaranteed to have a given directory or file structure.
  - Volumes help you decouple the configuration of the Docker host from the container runtime.
- When you need to back up, restore, or migrate data from one Docker host to another, volumes are a better choice.
  - You can stop containers using the volume, then back up the volume's directory (such as /var/lib/docker/volumes/<volume-name>).

Use cases for bind mounts

- Sharing configuration files from the host machine to containers.
  - This is how Docker provides DNS resolution to containers by default, by mounting /etc/resolv.conf from the host machine into each container.
- Sharing source code or build artifacts between a development environment on the Docker host and a container.
  - For instance, you may mount a Maven target/ directory into a container, and each time you build the Maven project on the Docker host, the container gets access to the rebuilt artifacts.

Use cases for tmpfs mounts

- tmpfs mounts are best used for cases when you do not want the data to persist either on the host machine or within the container.
- This may be for security reasons or to protect the performance of the container when your application needs to write a large volume of non-persistent state data.

Container and Layers

- The major difference between a container and an image is the top writable layer.
- All writes to the container that add new or modify existing data are stored in this writable layer.
- When the container is deleted, the writable layer is also deleted.
- The underlying image remains unchanged.
- Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state.
- Docker uses storage drivers to manage the contents of the image layers and the writable container layer.
- Each storage driver handles the implementation differently, but all drivers use stackable image layers and the copy-on-write (CoW) strategy.

Container size on disk

- To view the approximate size of a running container
- **docker container ls -s**
  - **size:** the amount of data (on disk) that is used for the writable layer of each container.
  - **virtual size:** the amount of data used for the read-only image data used by the container plus the container's writable layer size.

The copy-on-write (CoW) strategy

- Copy-on-write is a strategy of sharing and copying files for maximum efficiency.

Docker storage drivers

- Overlay2 (Default)
- aufs
- Devicemapper
- Btrfs
- zfs
- vfs

Docker storage drivers…

- **overlay2** - Preferred storage driver, for all currently supported Linux distributions, and requires no extra configuration.
- **aufs** - Was the preferred storage driver for Docker 18.06 and older.
- **devicemapper** - Was the recommended storage driver for CentOS and RHEL, as their kernel version did not support overlay2.
- **btrfs** and **zfs** – Supports advanced options, such as creating “snapshots”, but require more maintenance and setup.
- **vfs** - Intended for testing purposes, and for situations where no copy-on-write filesystem can be used.

Check your current storage driver

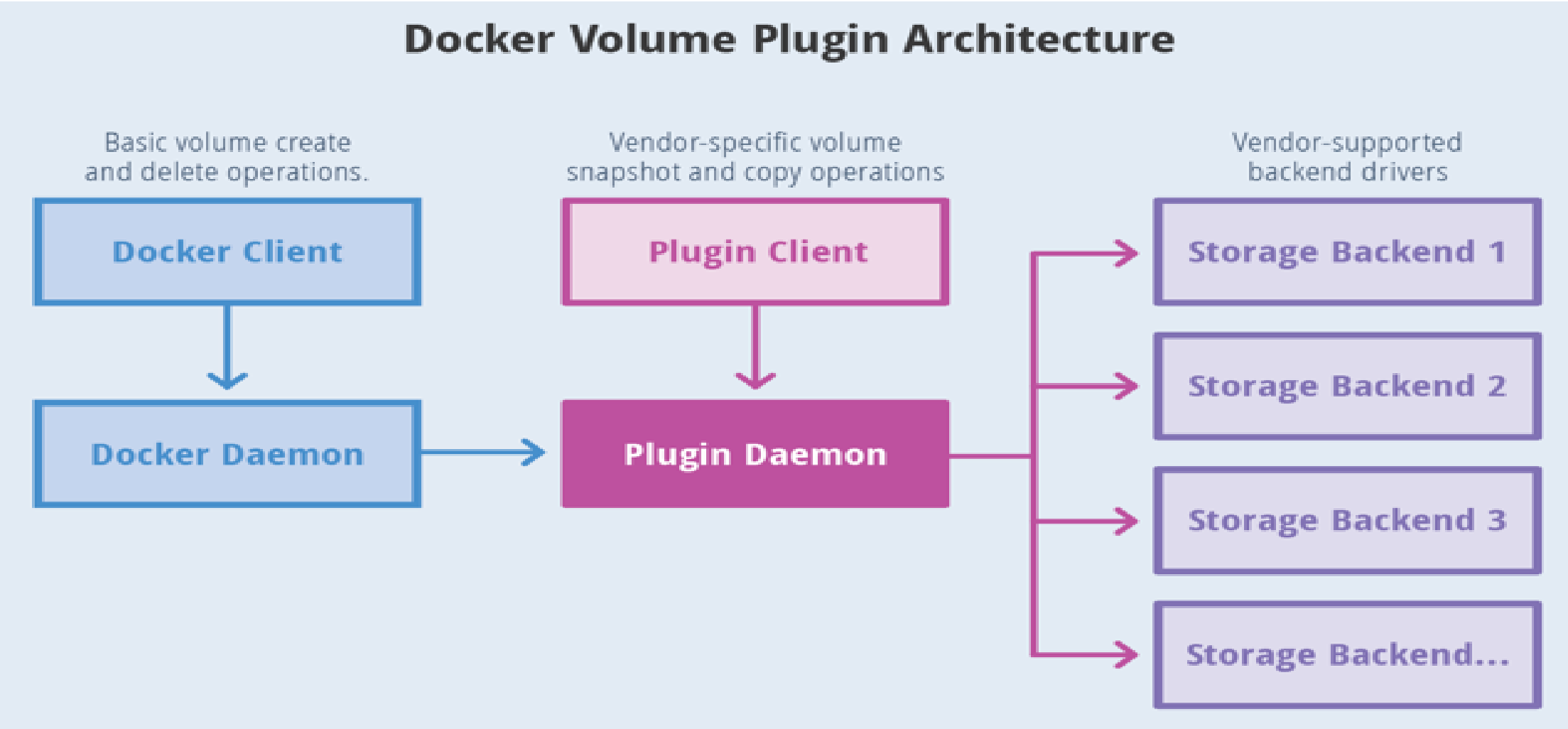
Run - **docker info** cmd

```
Server Version: 1.13.1
Storage Driver: overlay2
 Backing Filesystem: xfs
 Supports d_type: true
 Native Overlay Diff: true
```

Volume Plug-ins

- Docker has support for **plugins** to interact with 3rd party storage solutions.
- This allows the Docker to take advantage of the features of these storage solutions.
- Starting with **version 1.8**, Docker added support for 3rd party plugins.
- This enables the engine deployments to be integrated with external storage systems and volumes to persist beyond the lifetime of a single engine host.

Volume Plug-ins



Volume Plug-ins

- **Install appropriate plugin**

 # docker plugin install --grant-all-permissions <plugin>

```
[root@centos-docker ~]# docker plugin install vieux/sshfs DEBUG=1
Plugin "vieux/sshfs" is requesting the following privileges:
 - network: [host]
 - mount: [/var/lib/docker/plugins/]
 - mount: []
 - device: [/dev/fuse]
 - capabilities: [CAP_SYS_ADMIN]
Do you grant the above permissions? [y/N] y
Trying to pull repository docker.io/vieux/sshfs ...
latest: Pulling from docker.io/vieux/sshfs
52d435ada6a4: Download complete
```

Reference: https://docs.docker.com/engine/reference/commandline/plugin_install/

Create and manage volumes

- **Create a volume**:

```
# docker volume create my-vol
```

- **List volumes**:

```
# docker volume ls
```

Create and manage volumes

- **Inspect a volume:**

```
# docker volume inspect my-vol
[
    {
        "Driver": "local",
         "Labels": {},
         "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
               "Name": "my-vol",
          "Options": {},
          "Scope": "local"
    }
 ]
```

Create and manage volumes - Clean-up

- **Remove a volume**:

```
# docker volume rm my-vol
```

Start a Container with a Volume – Options

- **-v**
  - **# docker container run --name devtest -dit -v myvol:/app nginx:latest**

Clean-up Order

- `docker container stop devtest`
- `docker container rm devtest`
- `docker volume rm myvol`

Use-cases of read-only volume

- Multiple containers can mount the same volume, and it can be mounted read-write for some of them and read-only for others, at the same time.
- Container only needs read access to the data

docker container run --name nginxtest –dit -v nginx-vol:/usr/share/nginx/html:ro nginx:latest

Use a read-only volume

- Use **`docker container inspect nginxtest`** to verify the read-only mount.

```
"Mounts": [
        {
                "Type": "volume",
                "Name": "nginx-vol",
                "Source": "/var/lib/docker/volumes/nginx-vol/_data",
                "Destination": "/usr/share/nginx/html",
                "Driver": "local",
                "Mode": "",
                "RW": false,
                "Propagation": ""
        }
                ],
```

Lab - Docker Volume Service