**In Docker, there are two main types of storage mechanisms to persist data:**

1. **Volumes (managed by Docker)**
2. **Bind mounts (use a specific file or directory from the host)**

**These mechanisms allow you to store and persist data across container restarts and deployments, making them essential for applications requiring stateful data storage.**

**1. Docker Volumes**

A **Docker volume** is a managed storage mechanism, where Docker controls the storage location on the host. Volumes are easier to use because Docker abstracts the underlying file system structure.

* **Volumes** are stored in a part of the host filesystem that is managed by Docker (/var/lib/docker/volumes on Linux).
* Volumes are typically the preferred way to persist data in containers, as they offer better portability and ease of use.

**Why Use Volumes?**

* **Data persistence**: Keeps data even after containers are stopped or deleted.
* **Ease of backup and sharing**: Volumes are easy to back up or share between containers.
* **No direct dependency on the host's directory structure**.

**2. Bind Mounts**

A **bind mount** allows you to specify an exact directory or file on the host machine that should be accessible inside the container. The directory or file is **directly mounted** into the container.

* With **bind mounts**, Docker doesn’t manage the content or its lifecycle.
* The path for bind mounts must exist on the host machine.

**Why Use Bind Mounts?**

* **Flexibility**: Useful when you need to map specific host files/directories into the container.
* **Control**: You can map an exact directory on the host to the container for use cases such as logging, application development, etc.

**Key Differences:**

| **Feature** | **Docker Volume** | **Bind Mount** |
| --- | --- | --- |
| **Managed by Docker** | Yes | No |
| **Host Path** | Managed internally (/var/lib/docker/...) | User-defined absolute path on host |
| **Permissions** | Can be defined with options | Inherits permissions from the host |
| **Portability** | Highly portable across systems | Dependent on exact host path |
| **Backup** | Easy to back up and share | Must handle backups manually |

**Setup of Persistent Volumes for Docker**

**Steps to Create, Attach, Detach, and Delete a Docker Volume:**

1. Install Docker

If Docker is not already installed, use the following command to install it:

**yum install docker -y**

2. Start Docker Service

Make sure the Docker service is running:

**systemctl start docker**

Check if Docker is active:

**systemctl status docker**

3. Create a Docker Volume

To create a volume named my-app1, use the following command:

**docker volume create my-app1**

You can list all volumes to verify the creation:

**docker volume ls**

4. View Volume Directory on Host

Docker volumes are stored in /var/lib/docker/volumes/. Navigate to this directory:

**cd /var/lib/docker/volumes/**

**ls**

You should see the directory my-app1.

Navigate to the \_data directory inside my-app1, which is where the data for this volume is stored**:**

**cd my-app1/\_data/**

5. Run a Container with the Volume Attached

To attach the volume my-app1 to an Nginx container and bind it to the /usr/share/nginx/html directory (where Nginx serves files), use the following command:

**docker run -d -p 80:80 --mount source=my-app1,target=/usr/share/nginx/html nginx**

Make sure the paths are correct, and the Nginx default directory should be /usr/share/nginx/html

6. Verify the Running Container

To check if the container is running, use:

**docker ps**

7. Access the Running Container

If you need to enter the running container to verify or inspect things, use docker exec:

**docker exec -it <container\_id> /bin/bash**

Replace <container\_id> with the actual container ID from the docker ps output.

8. Detach and Stop the Container

To stop and remove the running container:

**docker stop <container\_id>**

**docker rm <container\_id>**

9. Delete the Volume

After stopping and removing the container, you can delete the volume using:

**docker volume rm my-app1**

**Additional Commands:**

* **List Docker Volumes:**

**docker volume ls**

* **Inspect Docker Volume:**

**docker volume inspect my-app1**

1. Create a Directory on the Host for Bind Mount

First, create a directory or file on your host machine that you want to bind to the Docker container.

For example, create a directory called my-app and a file index.html inside it:

**mkdir /path/to/my-app**

**echo "<h1>Hello from Docker Bind Mount</h1>" > /path/to/my-app/index.html**

Make sure you replace /path/to/my-app with the actual absolute path you intend to use.

2. Run a Docker Container with a Bind Mount

Run the container and mount the directory my-app from your host machine into the container. For example, mounting it to the Nginx container's HTML directory:

**docker run -d -p 80:80 --mount type=bind,source=/home/ec2-user/my-app2,target=/usr/share/nginx/html nginx**

Explanation:

* **type=bind**: Specifies that this is a bind mount (as opposed to a volume).
* **source=/path/to/my-app**: The absolute path on your host machine that you want to bind to the container.
* **target=/usr/share/nginx/html:** The path inside the container where the directory or file will be mounted. Nginx serves files from this directory by default.

Now, if you access http://localhost in your browser, you should see the content from your index.html file served by Nginx.

3. Inspect the Running Container and Bind Mount

To see the details of the running container, including the bind mount, run:

**docker inspect <container\_id>**

Look for the "Mounts" section in the output, which will show details of the bind mount, including the source and target directories.

4**. Make Changes to the Bind Mount on the Host**

Because a bind mount is a direct link between the host and the container, any changes made to the directory or file on the host will immediately be reflected in the container.

For example, if you update the index.html file on the host:

**echo "<h1>Updated Content from Host</h1>" >/home/ec2-user/my-app2/index.html**

If you refresh the webpage (http://localhost), you'll see the updated content.

5. Stop and Remove the Container

To stop the running container:

**docker stop <container\_id>**

To remove the container after stopping it:

**docker rm <container\_id>**

**Stopping and removing the container does not delete the bind mount directory on the host. The files in /path/to/my-app will remain intact.**

6. Detach the Bind Mount

**There is no explicit "detach**" operation for bind mounts like there is for volumes, but once you stop and remove the container, the bind mount will no longer be in use. The bind mount is only active while the container is running.

**Persistent Volumes in Docker Compose**

You can also manage volumes in a **docker-compose.yml** file to handle persistent volumes across multiple containers.

Here’s an example:

version: "3.9"

services:

web:

image: nginx

volumes:

- my\_volume:/data

volumes:

my\_volume:

In this example:

* The volume my\_volume is defined under the volumes section.
* It’s then mounted in the nginx container at /data.

**Lab**

docke

**Best Practices for Docker Volumes and Bind Mounts**

1. **Use Volumes for Data Persistence**: For databases or applications where data needs to be preserved across container lifecycles.
2. **Use Bind Mounts for Development**: When you need to share source code between your host and container, or for sharing logs and configuration files.
3. **Backup Volumes**: Always keep backups of important data stored in volumes, especially if it's critical for your application.
4. **Limit Permissions on Bind Mounts**: Ensure correct user permissions when using bind mounts to avoid security risks.