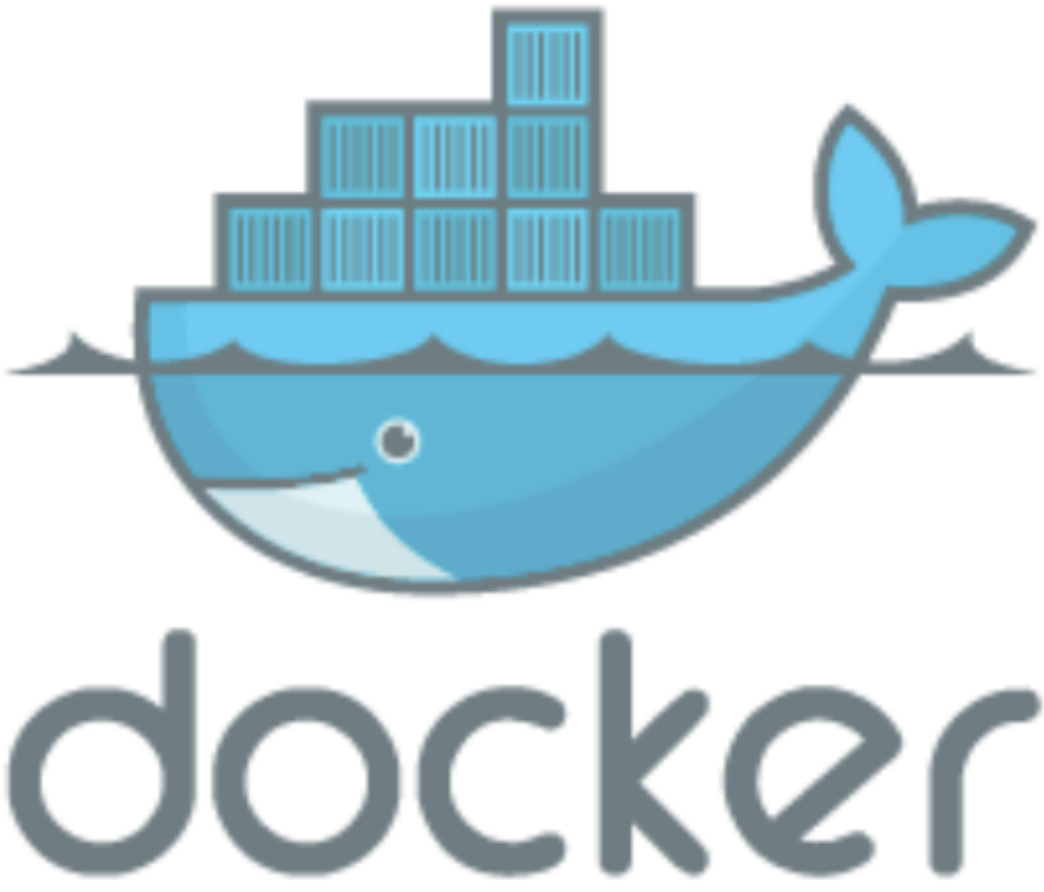
CONTAINERS AND VOLUMES



Introduction to Docker Containers

What is Docker?

Docker is an open-source platform designed to simplify the process of developing, shipping, and running applications. It uses containerization technology to package applications and their dependencies into lightweight, portable containers. These containers are isolated from each other and the host system, ensuring consistency across different environments.

Why Use Docker?

- **Isolation**: Each container runs in its own isolated environment, preventing conflicts between applications.
- **Portability**: Containers can run on any system that supports Docker, making it easy to move applications between development, testing, and production environments.
- **Efficiency**: Containers share the host system's kernel, making them more lightweight and faster to start compared to virtual machines.
- **Scalability**: Docker makes it easy to scale applications horizontally by running multiple container instances.

Key Components of Docker

- **Docker Engine**: The core component that builds and runs containers.
- **Docker Images**: Read-only templates used to create containers.
- **Docker Containers**: Runnable instances of Docker images.
- **Docker Hub**: A cloud-based repository for sharing Docker images.

Managing Containers

Starting, Stopping, and Removing Containers

- **Start a Container**:

  ```
  docker start <container_id>
  ```

  This command starts a stopped container.

- **Stop a Container**:

  ```
  docker stop <container_id>
  ```

  This command gracefully stops a running container.

- **Remove a Container**:

  ```
  docker rm <container_id>
  ```

This command deletes a stopped container.

Running a Simple Container

- **Hello-World Example**:

  ```
  docker run hello-world
  ```

  This command pulls the `hello-world` image from Docker Hub (if not already available) and runs it in a container. It's a great way to verify that Docker is installed correctly.

Attaching to a Running Container

- **Accessing a Container's Shell**:

  ```
  docker exec -it <container_id> bash
  ```

  This command opens an interactive Bash shell inside a running container. Replace `bash` with `sh` for containers that don't have Bash installed.

Understanding Container Lifecycle

Lifecycle Stages

1. **Created**: The container is created but not started.
2. **Running**: The container is actively executing its processes.
3. **Paused**: The container's processes are suspended.
4. **Stopped**: The container's processes are terminated.
5. **Deleted**: The container is removed from the system.

Commands for Lifecycle Management

- **Pause a Container**:

  ```
  docker pause <container_id>
  ```

- **Unpause a Container**:

  ```
  docker unpause <container_id>
  ```

- **Restart a Container**:

  ```
  docker restart <container_id>
  ```

Monitoring Container States

- **List All Containers**:

```
docker ps -a
```

This command shows all containers, including stopped ones.

Managing Persistent Data with Docker Volumes

What are Docker Volumes?

Docker volumes are used to persist data outside containers. They are stored in a dedicated directory on the host system and are not deleted when containers are removed.

Creating and Managing Volumes

- **Create a Volume**:

  ```
  docker volume create my_volume
  ```

- **List Volumes**:

  ```
  docker volume ls
  ```

- **Inspect a Volume**:

  ```
  docker volume inspect my_volume
  ```

  This command provides detailed information about the volume, including its mount point on the host system.

Using Volumes in Containers

- **Mount a Volume**:

  ```
  docker run -d --name my_container -v my_volume:/app/data
          my_image
  ```

  This command mounts `my_volume` to the `/app/data` directory inside the container.

Bind Mounts vs Docker Volumes

Bind Mounts

- Bind mounts map a specific directory or file on the host system to a directory or file in the container.
- Example:

  ```
  docker run -v /host/path:/container/path my_image
  ```

- **Use Cases**:
    - Development environments where code changes need to be reflected immediately.
    - Accessing configuration files on the host system.

Docker Volumes

- Docker volumes are managed by Docker and stored in a dedicated directory (`/var/lib/docker/volumes` on Linux).
- Example:

  ```
  docker run -v my_volume:/container/path my_image
  ```

- **Use Cases**:
    - Persistent data storage for databases.
    - Sharing data between multiple containers.

Persistent Data Storage Across Container Restarts

Why Use Volumes?

- **Data Persistence**: Volumes ensure that data is not lost when containers are deleted or restarted.
- **Data Sharing**: Volumes can be shared between multiple containers.
- **Backup and Migration**: Volumes make it easy to back up and migrate data.

Example: Using Volumes for a Database

```
docker run -d --name mysql_db -v db_volume:/var/lib/mysql -e
        MYSQL_ROOT_PASSWORD=password mysql
```

This command runs a MySQL container with a volume for persistent data storage.

Backing Up and Restoring Volumes

Backup a Volume

```
docker run --rm -v my_volume:/volume -v $(pwd):/backup busybox
        tar cvf /backup/backup.tar /volume
```

This command creates a backup of `my_volume` and saves it as `backup.tar` in the current directory.

Restore a Volume

```
docker run --rm -v my_volume:/volume -v $(pwd):/backup busybox
        tar xvf /backup/backup.tar -C /volume
```

This command restores the backup to `my_volume`.

Dockerfile Examples

Simple Dockerfile

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y python3
COPY . /app
WORKDIR /app
CMD ["python3", "app.py"]
```

This Dockerfile creates an image with Python 3 installed and runs `app.py` as the default command.

Multi-Stage Dockerfile

```
FROM node:14 AS build
WORKDIR /app
COPY . .
RUN npm install && npm run build

FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

This Dockerfile uses multi-stage builds to create a lightweight production image for a Node.js app.
