Docker Fundamentals

Host Machine

The **host machine** is the physical computer or server where virtualization technologies or Docker containers run. It provides the hardware and operating system for running VMs or Docker containers.

Virtual Machine (VM)

A virtual machine emulates a physical computer using software, allowing multiple OS instances to run on the same physical hardware.

Docker

Docker is a tool for creating a portable runtime environment for any application and running it as a container.

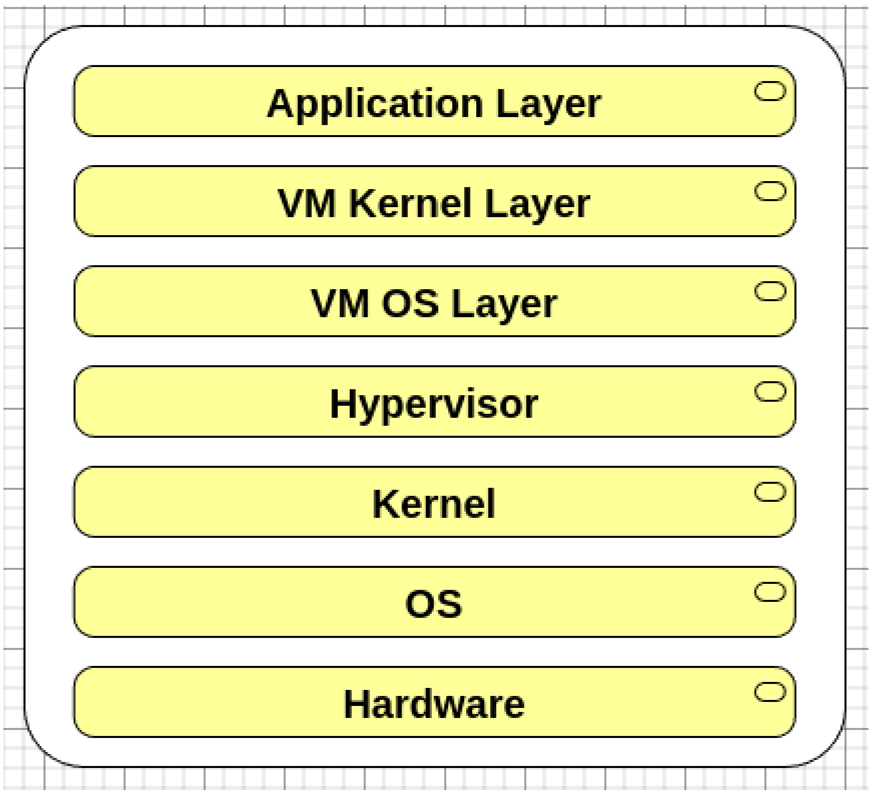
Hypervisor

A hypervisor is software, firmware, or hardware that creates and manages virtual machines (VMs) by abstracting and partitioning physical hardware resources (CPU, memory, storage, and network). It allows multiple operating systems (OSes) to run simultaneously on a single physical machine, ensuring isolation and resource allocation.

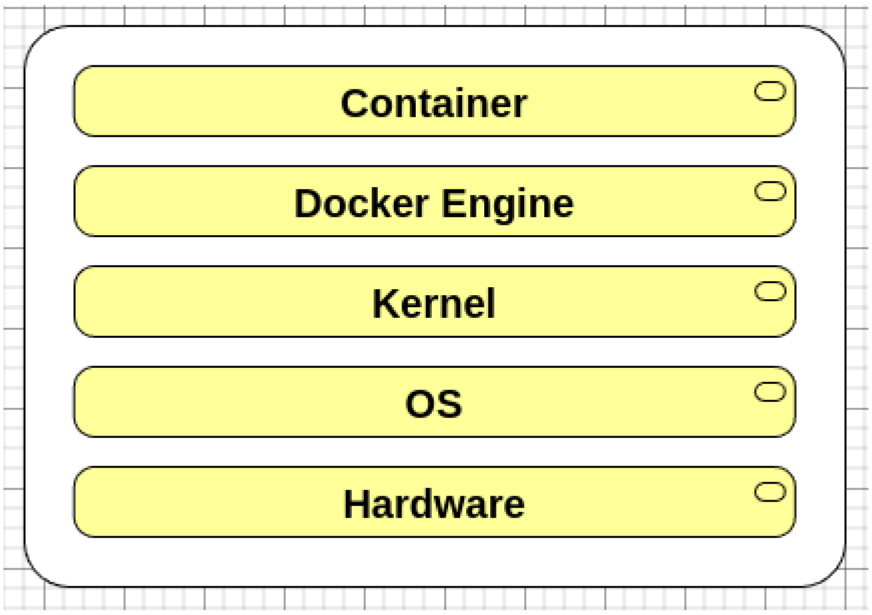
Kernel

A kernel is the core component of an operating system that acts as a bridge between the hardware and software. It manages system resources such as CPU, memory, and I/O devices, and provides essential services like process management, memory management, and device drivers.

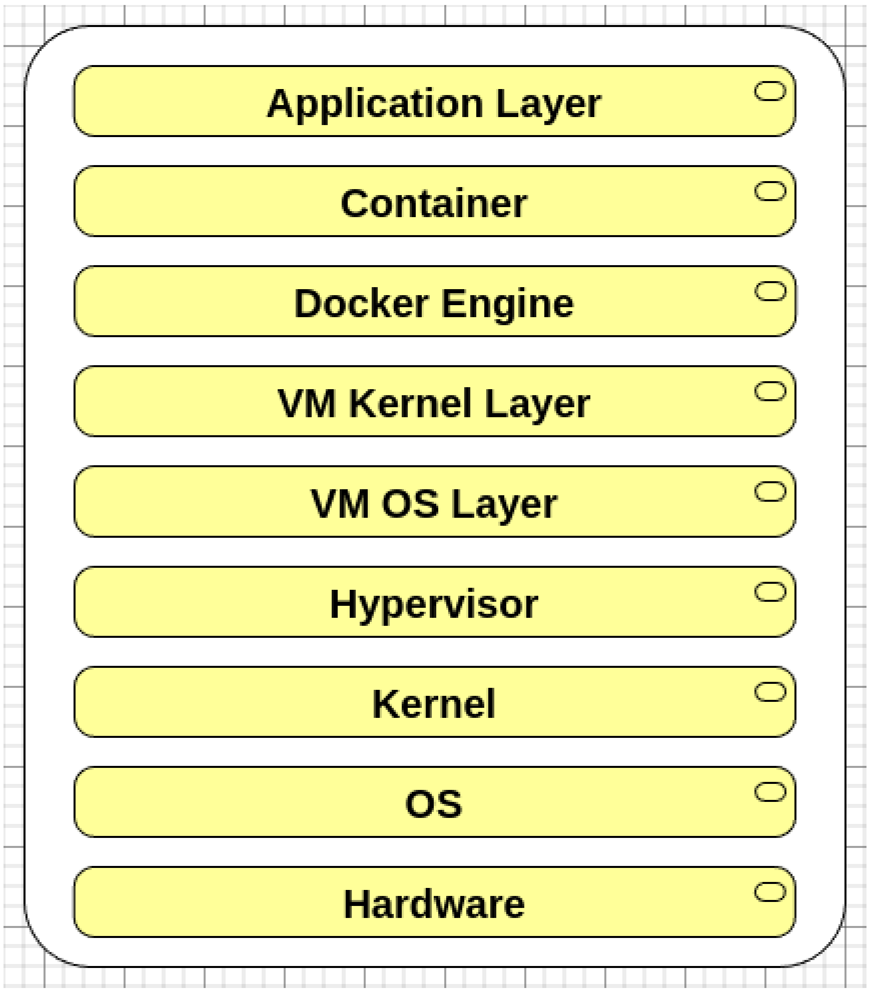
VM Layer



Docker Layer



Docker inside VM Layer



Essential Docker Components

Docker Engine

Docker Engine is an open-source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:

- A server with a long-running daemon process `dockerd`.
- APIs specify interfaces that programs can use to talk to and instruct the Docker daemon.
- A command line interface (`CLI`) client `docker`.

Docker Daemon

The Docker daemon (`dockerd`) is a background process that listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services. It runs on the host machine and directly communicate with the OS.

Docker CLI

A command line interface that passes down the commands to the docker daemon using REST API

Docker REST APIs

These APIs actually represent the main functionality of docker. These APIs are passed to the docker daemon as a specific command, which is then executed in the docker daemon. These APIs can be used or called externally to do various docker executions without directly using the CLI.

Docker Image

An image is a read-only template with instructions for creating a Docker container. In most cases, an image is based on another image, with some additional customization.

Docker Container

A container is a runnable instance of an image.

Docker Registry

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker looks for images on Docker Hub by default.

Docker Volumes

A **Docker-managed storage mechanism** on the host machine where data from a container is stored. Volumes are designed for **persistent storage** and are managed independently of container lifecycles, ensuring data persists even if the container is removed.

Docker Mount

A **more flexible method** of attaching storage to a container. Can have the following methods:

- **Volumes**: As described above, for Docker-managed storage.

- **Bind Mounts**: Directly mount a specified host machine directory into the container.
- **tmpfs Mounts**: Stores data in memory, useful for temporary data that does not need persistence.

Docker Network

Bridge Network

An isolated network, where each container is connected to the same bridge, can communicate among themselves.

To enable external access, mapping between container ports and external ports is needed

Host Network

Containers use the host network directly. All containers will have the same host IP.

No mapping is required

Overlay Network

Used in docker swarm to allow containers to communicate across different docker hosts.

Useful for communicating among multiple nodes in the docker swarm

Essential Docker Commands

1. Images:

- **List Images:**

  ```
  docker images
  ```

  Or:

  ```
  docker image ls
  ```

- **Build an Image:**

```
docker build -t <image-name>:<tag> <path-to-dockerfile>
```

- **Pull an Image from a Registry:**

```
docker pull <image-name>:<tag>
```

- **Push an Image to a Registry:**

```
docker push <image-name>:<tag>
```

- **Remove an Image:**

```
docker rmi <image-name>:<tag>
```

- **Tag an Image:**

```
docker tag <source-image>:<source-tag> <target-image>:<target-tag>
```

- **Inspect an Image:**

```
docker image inspect <image-name>:<tag>
```

2. Containers:

- **List Containers (including stopped ones):**

```
docker ps -a
```

- **List Running Containers:**

```
docker ps
```

- **Start a Container:**

```
docker start <container-id>
```

- **Stop a Container:**

```
docker stop <container-id>
```

- **Restart a Container:**

```
docker restart <container-id>
```

- **Remove a Container:**

```
docker rm <container-id>
```

- **Run a Container (create and start):**

```
docker run <options> <image-name>:<tag>
```

- **Inspect a Container:**

```
docker inspect <container-id>
```

- **Attach to a Running Container:**

```
docker attach <container-id>
```

- **Execute a Command in a Running Container:**

```
docker exec -it <container-id> <command>
```

3. Networks:

- **List Networks:**

```
docker network ls
```

- **Create a Network:**

```
docker network create <network-name>
```

- **Inspect a Network:**

```
docker network inspect <network-name>
```

- **Remove a Network:**

```
docker network rm <network-name>
```

- **Connect a Container to a Network:**

```
docker network connect <network-name> <container-id>
```

- **Disconnect a Container from a Network:**

```
docker network disconnect <network-name> <container-id>
```

4. Volumes:

- **List Volumes:**

```
docker volume ls
```

- **Create a Volume:**

```
docker volume create <volume-name>
```

- **Inspect a Volume:**

```
docker volume inspect <volume-name>
```

- **Remove a Volume:**

```
docker volume rm <volume-name>
```

- **Prune Unused Volumes:**

```
docker volume prune
```

- **Run a Container with a Volume:**

```
docker run -v <volume-name>:<container-path> <image-name
>:<tag>
```

5. Mounts (for Bind Mounts and Volumes):

- **Bind Mount (Mount a Host Directory to a Container):**

```
docker run -v <host-path>:<container-path> <image-name>:
<tag>
```

- **Run a Container with a `tmpfs` Mount:**

```
docker run --tmpfs <container-path>:<options> <image-nam
e>:<tag>
```

- **Run a Container with Multiple Mounts:**

```
docker run -v <host-path1>:<container-path1> -v <host-pa
th2>:<container-path2> <image-name>:<tag>
```

Additional Useful Commands:

- **Prune Unused Containers, Images, Volumes, and Networks:**

```
docker system prune
```

- **View Docker System Information:**

```
docker info
```

Default File Locations

- Container Data: /var/lib/docker/containers/
- Images: /var/lib/docker/image/ or /var/lib/docker/overlay2
- Volume: /var/lib/docker/network/
- Log: /var/lib/docker/containers/<container-id>/
- Docker Daemon Configuration: /etc/docker/daemon.json
- Docker System File: /etc/systemd/system/docker.service
- Docker CLI Configuration: ~/.docker/config.json

DockerFile

```
# Step 1: Specify the base image
# You can use an official image or a custom one as a base.
FROM <base-image>:<tag>

# Step 2: Set environment variables (Optional)
# Define environment variables inside the container.
ENV <ENV_VAR_NAME>=<value>

# Step 3: Install dependencies (Optional)
# Install system packages or dependencies required for your a
RUN apt-get update && \
    apt-get install -y <package1> <package2> && \
    apt-get clean

# Step 4: Set the working directory
# This sets the directory for subsequent instructions (COPY,
WORKDIR /path/to/working/directory

# Step 5: Copy files into the container
# Copy files or directories from the host system into the con
COPY <host-path> <container-path>

# Alternatively, if you have a specific pattern of files:
# COPY ./src/ /app/
# Step 6: Expose ports (Optional)
# Expose ports the application will use to communicate with t
EXPOSE 80
EXPOSE 443

# Step 7: Set user (Optional)
# If you need to specify a non-root user.
USER <username>

# Step 8: Install application dependencies (e.g., for Node.js
# For Node.js, install npm packages; for Python, install pip
# Example for Node.js (npm install):
RUN npm install

# Example for Python (pip install):
RUN pip install -r requirements.txt

# Step 9: Add entrypoint (Optional)
# Define the command to run when the container starts.
ENTRYPOINT ["<entrypoint-command>"]

# Step 10: Default command to run (Optional)
# This will run if no command is provided at runtime.
CMD ["<command>"]

# Alternative: Combine ENTRYPOINT and CMD if needed
# ENTRYPOINT ["python", "app.py"]
# CMD ["--help"]

# Step 11: Clean up (Optional)
# You can clean up unnecessary files to reduce the image size
RUN rm -rf /tmp/*
```