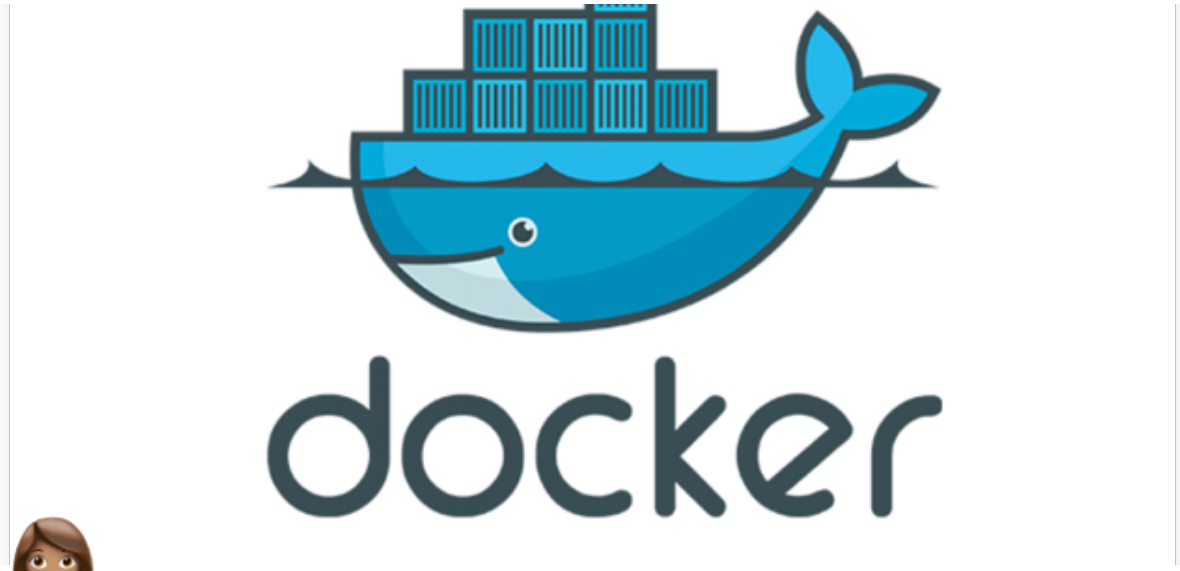
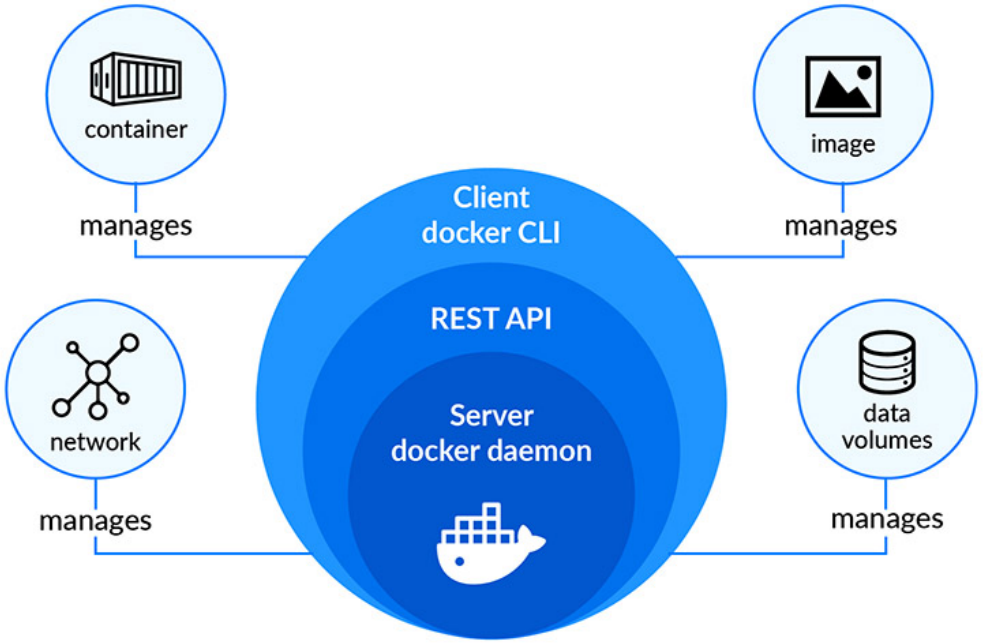


Overview of Docker

What is a docker ?

- Docker is an open-source platform for building, deploying, and managing containerized applications.
- **Why Docker?**:
  - Simplifies container creation and management.
  - Provides a consistent environment for development, testing, and production.
- **Key Terms**:
  - **Image**: A lightweight, stand-alone, and executable package that includes everything an application needs (code, runtime, libraries, dependencies).
  - **Container**: A runtime instance of an image. While images are static, containers are dynamic and can be started, stopped, or moved across environments.
  - **Dockerfile**: A text file with instructions to build a Docker image. It defines the steps to configure an environment, install dependencies, and set up the application.

- **Docker Hub**: A cloud-based repository for finding and sharing container images, both public and private.



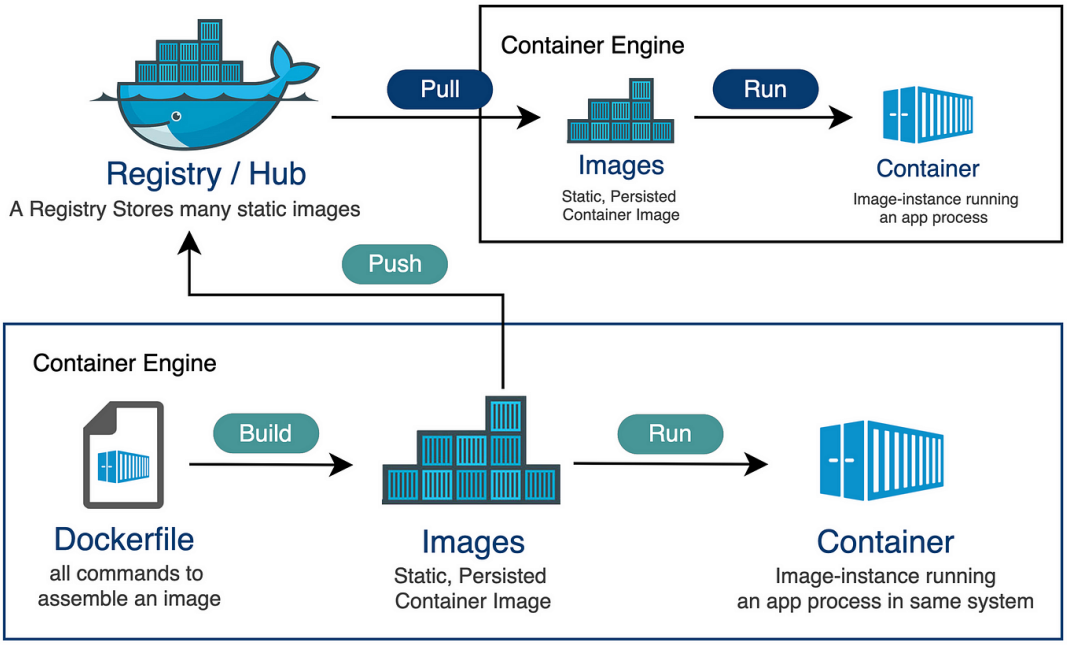
▼ Why Use Docker in Modern Development?

- **Consistency Across Environments**:
  - Docker ensures that the application runs the same regardless of where it's deployed (local machine, production server, or cloud).
- **Isolation**:
  - Containers provide process isolation, so multiple applications or microservices can run side by side without interfering with each other.
- **Scalability**:
  - Containers are lightweight and can be easily scaled horizontally (replicating containers to handle more traffic).
- **Efficiency**:
  - Compared to virtual machines, containers use fewer resources since they share the host machine's kernel, making them much faster to start and stop.
- **Portability**:

- Applications packaged in Docker containers can be easily moved across environments, cloud platforms, and OSes, ensuring smooth and reliable deployments.

▼ Key Docker Components

- **Docker Engine**: The runtime that runs and manages containers.
- **Docker Hub**: A public repository where users can publish and share container images.
- **Dockerfile**: A text file that contains instructions on how to build a Docker image.



▼ Installing Docker on Various Platforms

Windows:

- **Step 1**: Go to the Docker Desktop for Windows download page.
- **Step 2**: Download the Docker Desktop installer.
- **Step 3**: Double-click the installer and follow the prompts to install.
- **Step 4**: Ensure that **Windows Subsystem for Linux (WSL 2)** is enabled for a smoother experience.

- **Step 5**: Once installed, launch Docker Desktop. You can verify the installation by running the following in PowerShell or CMD:

```
docker --version
```

macOS:

- **Step 1**: Go to the Docker Desktop for Mac download page.
- **Step 2**: Download the installer for macOS.
- **Step 3**: Open the `.dmg` file and drag **Docker.app** to the Applications folder.
- **Step 4**: Launch Docker from the Applications folder. Once it is running, verify by opening a terminal and typing:

```
docker --version
```

Linux (Ubuntu Example):

- **Step 1**: Update the package index:

```
sudo apt update
```

- **Step 2**: Install Docker:

```
sudo apt install docker.io
```

- **Step 3**: Start Docker and enable it on boot:

```
sudo systemctl start docker
sudo systemctl enable docker
```

- **Step 4**: Verify installation:

```
docker --version
```

Post-Installation Steps (All Platforms):

- **Step 1**: Run the Docker hello-world image to confirm everything is set up correctly:

```
docker run hello-world
```

- **Step 2**: Ensure you have permissions to run Docker as a non-root user:

  For Linux, add your user to the `docker` group:

```
sudo usermod -aG docker $USER
```

▼ Building and Running Applications in Docker Containers

1. **Dockerfile Structure**:
   - **Base Image**: The starting point (e.g., `openjdk:17` for Java applications).
   - **WORKDIR**: The directory inside the container where the application will reside.
   - **COPY**: Copies files from the host system into the container.
   - **RUN**: Executes commands (e.g., installing dependencies).
   - **CMD**: Defines the default command to run when the container starts (e.g., `java -jar app.jar`).

   Example `Dockerfile` for a Spring Boot application:

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY target/musicFinder-1.0.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

2. **Building the Docker Image**:
   - Command: `docker build -t my-app .`
     - This command builds the Docker image by reading the `Dockerfile`.
3. **Running the Docker Container**:

- Command: `docker run -p 8080:8080 my-app`
  - This command runs the container, mapping the container's port 8080 to the host's port 8080.

▼ Docker Best Practices

1. **Keep Images Lightweight**:
   - Use minimal base images (e.g., `alpine`) to reduce the size of the final image, leading to faster build times and fewer security vulnerabilities.
2. **Multi-Stage Builds**:
   - Separate the build environment from the final image to reduce size and improve performance.

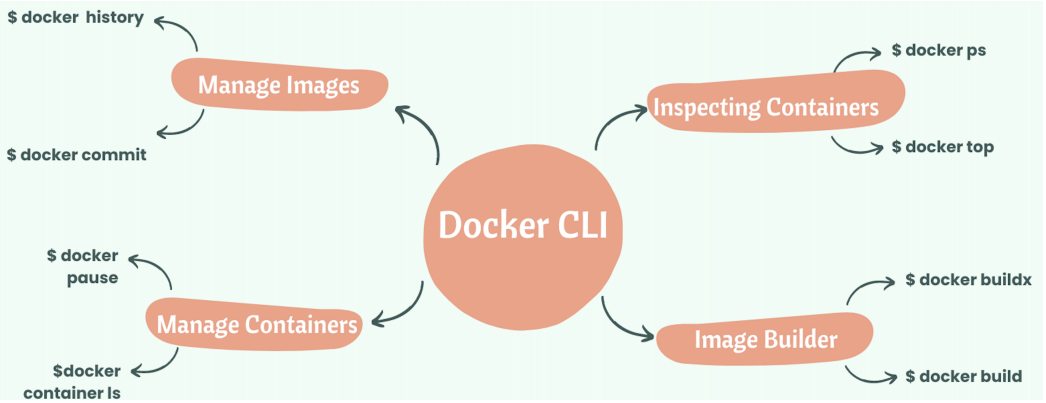
```
FROM maven:3.8-jdk-11 AS builder
WORKDIR /build
COPY . .
RUN mvn clean package

FROM openjdk:11-jre-slim
WORKDIR /app
COPY --from=builder /build/target/app.jar /app.jar
CMD ["java", "-jar", "/app.jar"]
```

3. **Use `.dockerignore`**:
   - Similar to `.gitignore`, it prevents unnecessary files from being copied into the container, optimizing build times.
4. **Tagging**:
   - Tag your images (`docker build -t my-app:v1 .`) for version control and easier management of deployments.
5. **Security Best Practices**:
   - Regularly update base images to avoid security vulnerabilities.
   - Avoid running containers as root (use non-root users).
   - Scan your Docker images for vulnerabilities (e.g., using tools like **Clair** or **Anchore**).

▼ Common Docker Commands

- **Listing Containers**:
  - `docker ps`: List running containers.
  - `docker ps -a`: List all containers (including stopped ones).
- **Stopping/Removing Containers**:
  - `docker stop container_id`: Stops a running container.
  - `docker rm container_id`: Removes a stopped container.
- **Viewing Logs**:
  - `docker logs container_id`: Shows the logs of a container.
- **Entering a Running Container**:
  - `docker exec -it container_id /bin/bash`: Opens a terminal inside the running container.



▼ Advantages of Docker in Development

- **Consistency**: Eliminates the "works on my machine" problem by providing a consistent environment across all stages of development.
- **Efficiency**: Uses fewer system resources and has fast startup times compared to traditional VMs.
- **Easy Integration**: Seamlessly integrates with CI/CD tools and workflows.