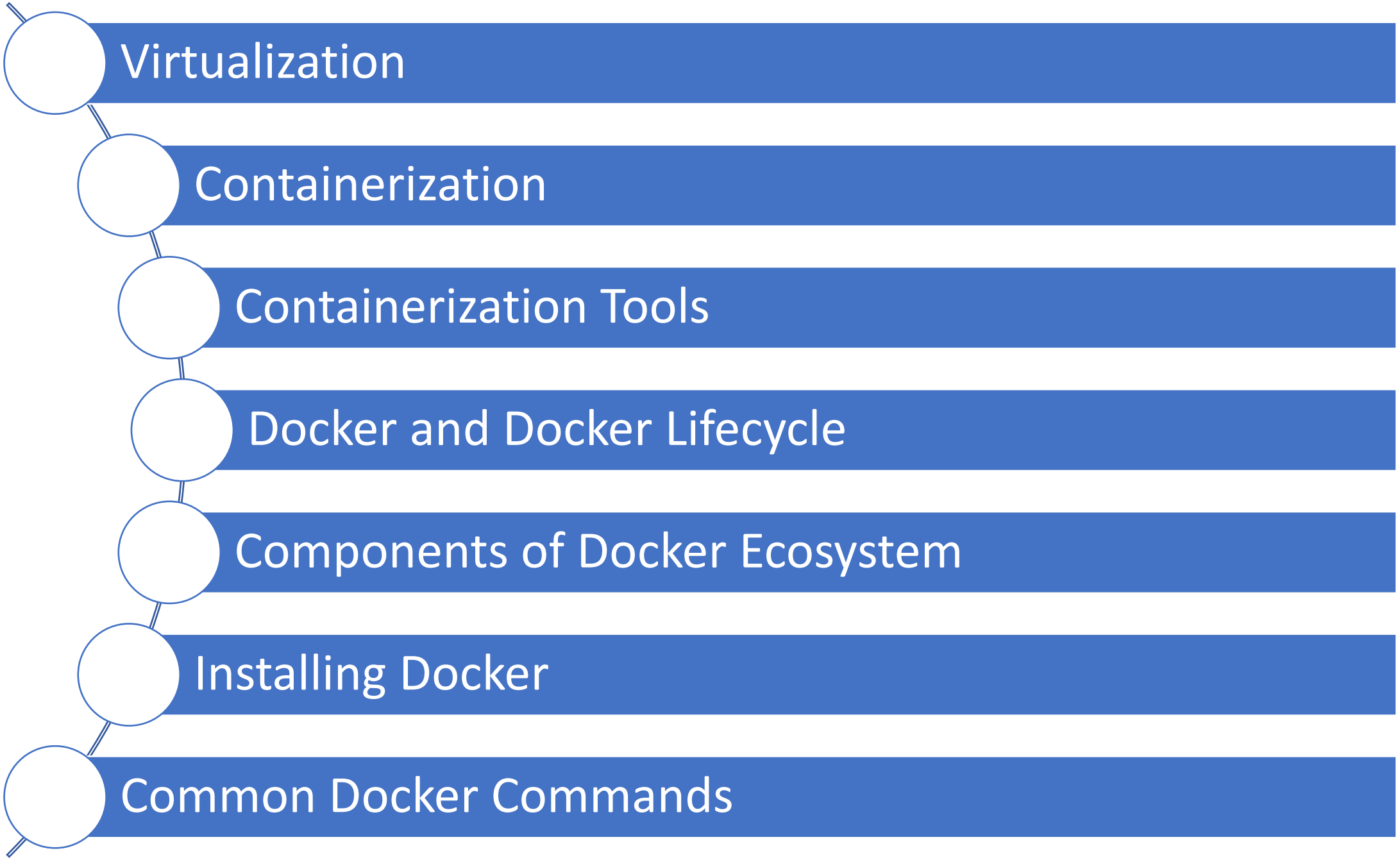
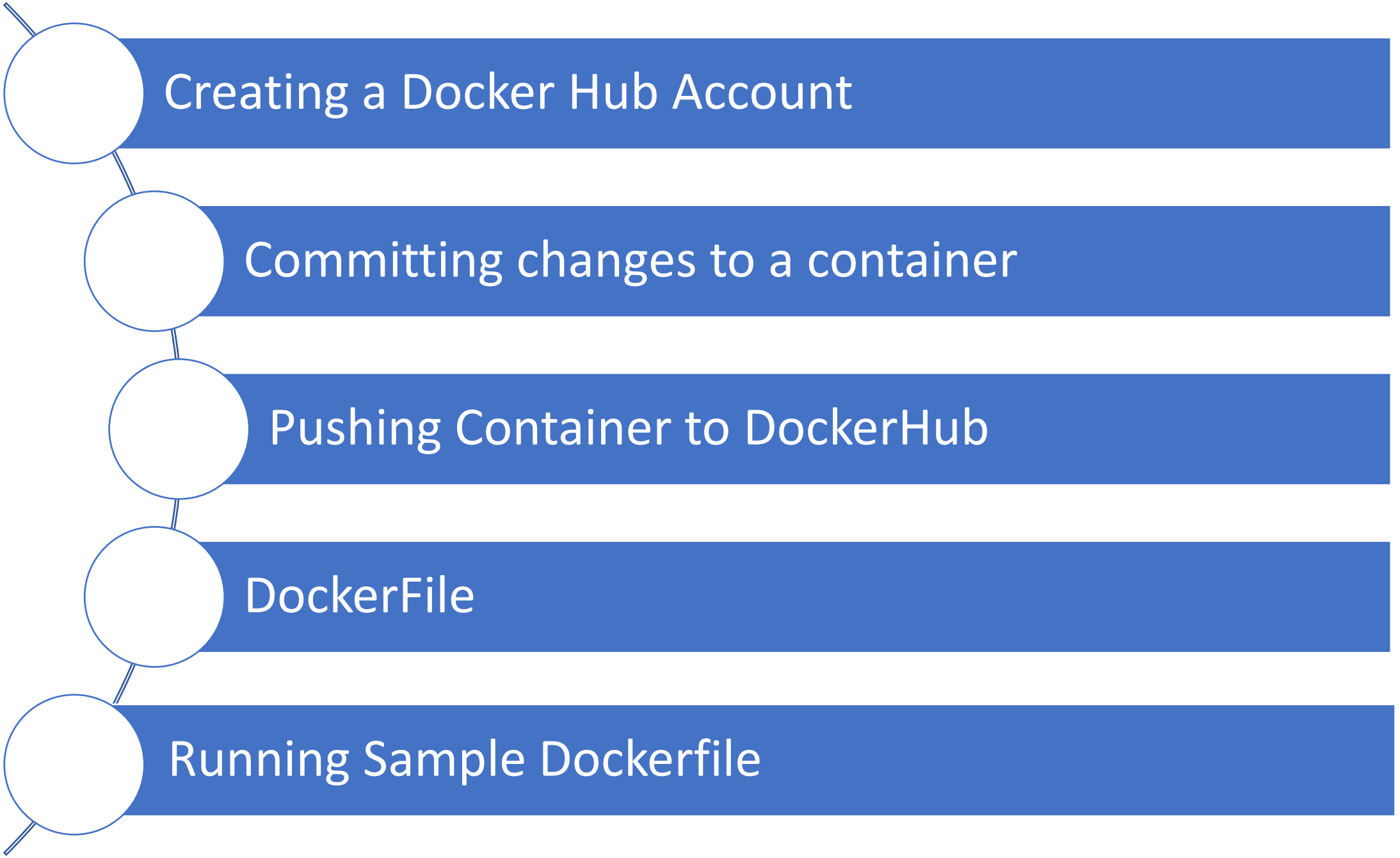
Containerization

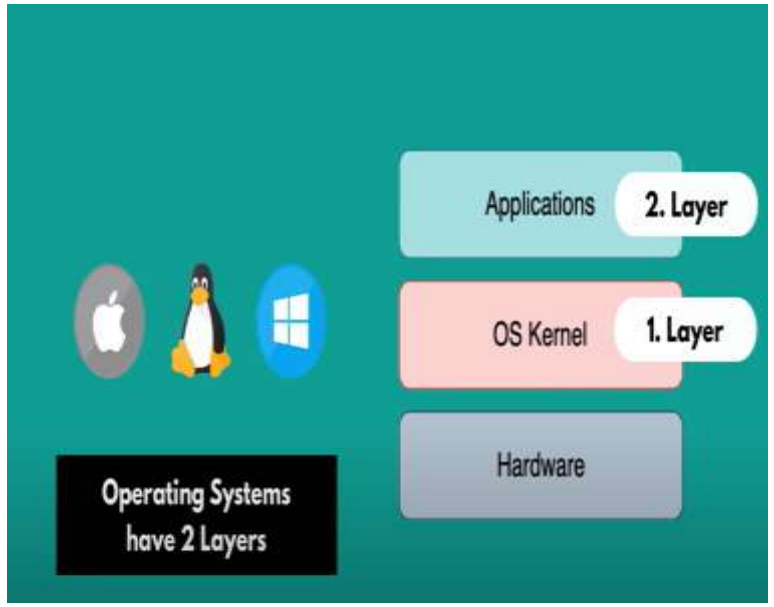
Docker

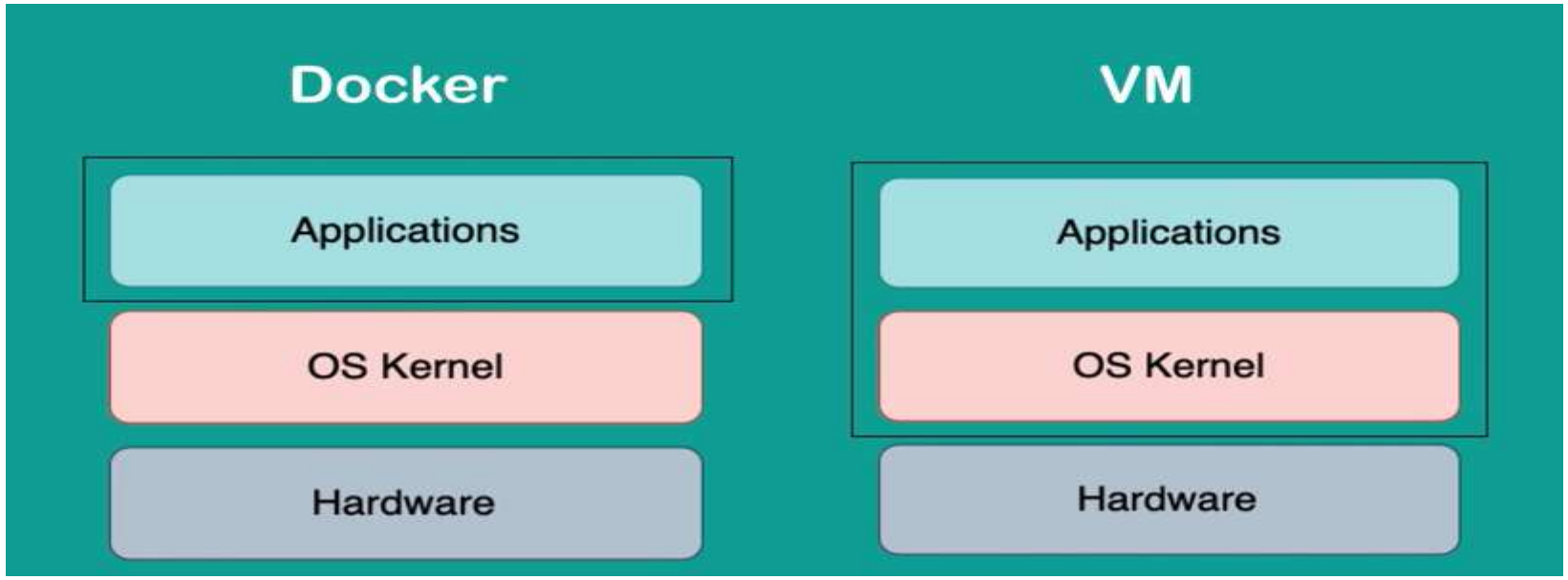
Presented by: [Santhosh Kiran J]


Virtualization
Containerization
Containerization Tools
Docker and Docker Lifecycle
Components of Docker Ecosystem
Installing Docker
Common Docker Commands


Creating a Docker Hub Account
Committing changes to a container
Pushing Container to DockerHub
DockerFile
Running Sample Dockerfile

Docker vs Virtual Machine (VM)





- Size of Docker Image is much smaller
- Docker Containers can start and run much faster
- VM of any OS can run on any OS host where as Docker is limited to Linux.

Virtualization

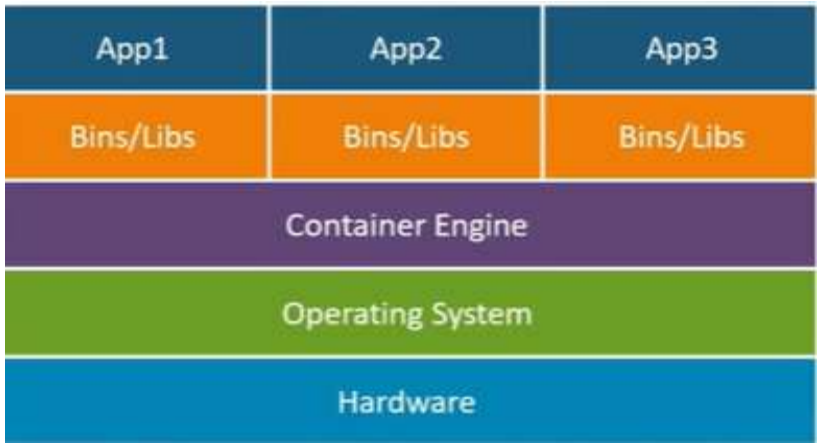
Definition: Virtualization is the process of creating a virtual version of something, such as a server, storage device, network, or even an operating system.

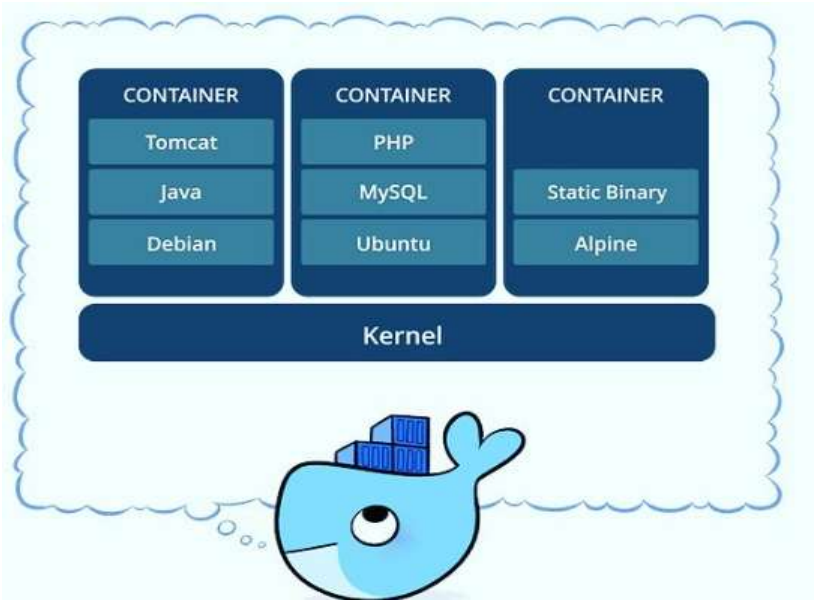
Benefits: Improved resource utilization, isolation, and flexibility.

Examples: VMware, Hyper-V, VirtualBox.

Containerization

Definition: Containerization is a lightweight form of virtualization that packages an application and its dependencies together. It is a portable artifact, easily shared and moved around.
Benefits: Portability, efficiency, scalability, and consistency across environments. Makes development and deployment processes more efficient.
Examples: Docker, Kubernetes, LXC.





What is a Container?

- Layers of images

- Mostly **Linux Base Image**, because small in size
- Application image on top

postgres:10.10 — **Layer -** application image

alpine:3.10 — **Layer -** linux base image

Docker Image

- the actual package

- **artifact**, that can be moved around

not running

Docker Container

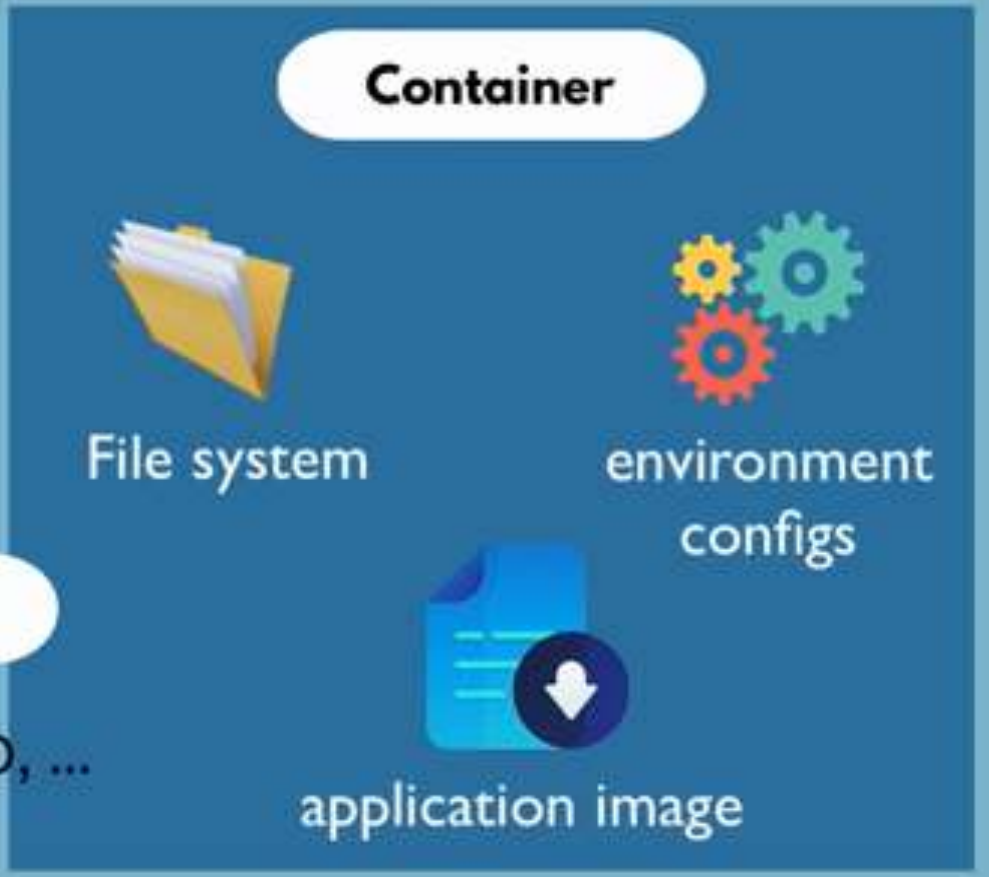
- actually **start the application**
- container environment is created

running

Difference Image and Container

- **CONTAINER** is a running environment for **IMAGE**

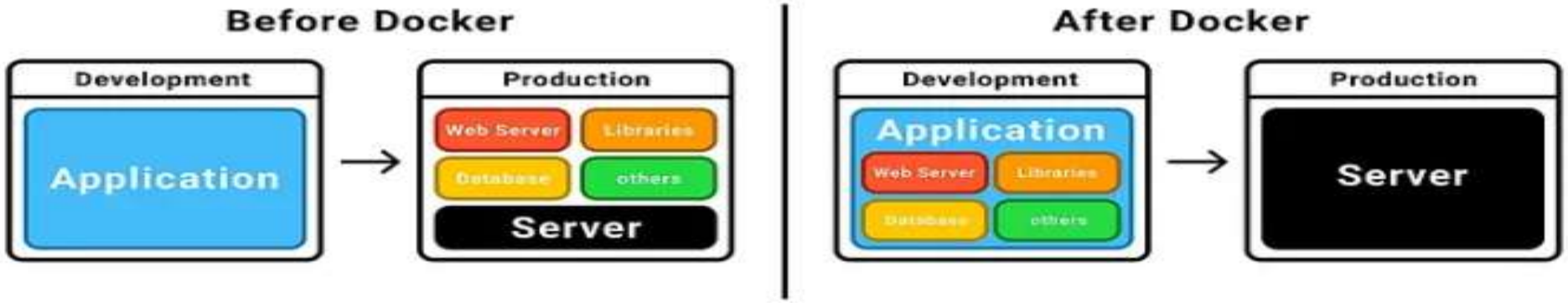
- **virtual** file system
- port binded: talk to application running inside of container
- application image: postgres, redis, mongo, ...



Containerization Tools

Docker: Open-source platform for developing, shipping, and running applications.
Kubernetes: Container orchestration tool for managing containerized applications.
AWS Fargate: Serverless compute engine for running containers.
Azure Container Instances: Managed container instances for running containers without managing the infrastructure.

Application Development – Before and After Docker



- Installation Process different on each OS environment
- Multiple installation Steps where something may go wrong

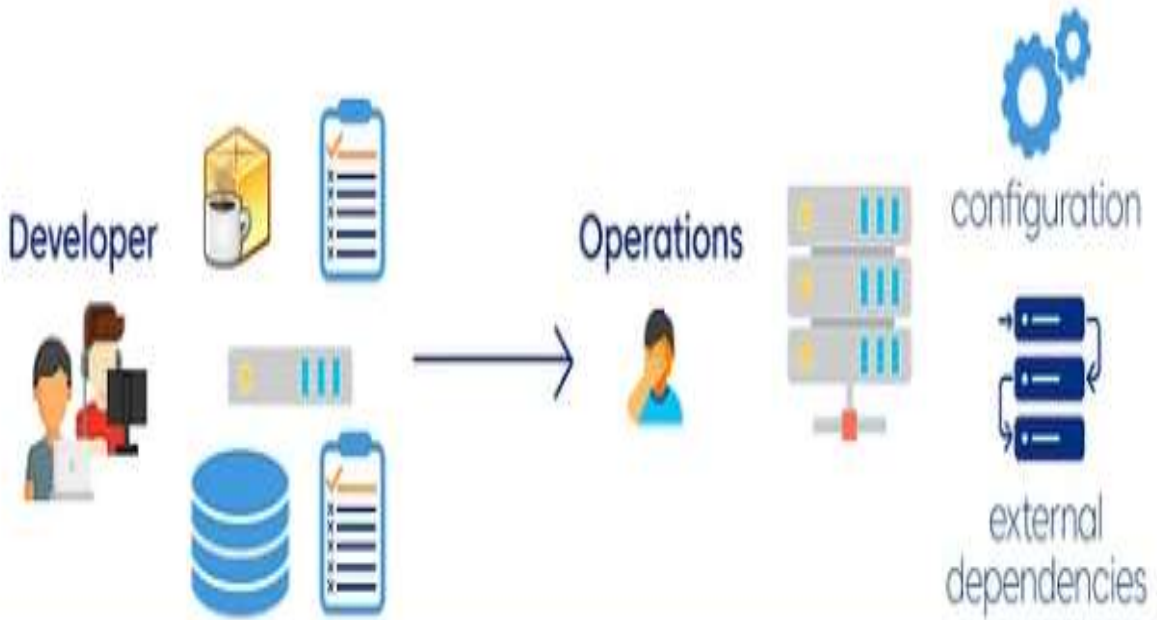
- Own isolated environment
- Packaged with all the required configurations
- One command to install the app
- Run the same application with 2 different versions

Application Deployment – Before and After Docker

Application Deployment

Before containers

- Multiple server configuration per team member could lead to the dependency version conflicts
- Textual guides could lead to misunderstandings and misinterpretations

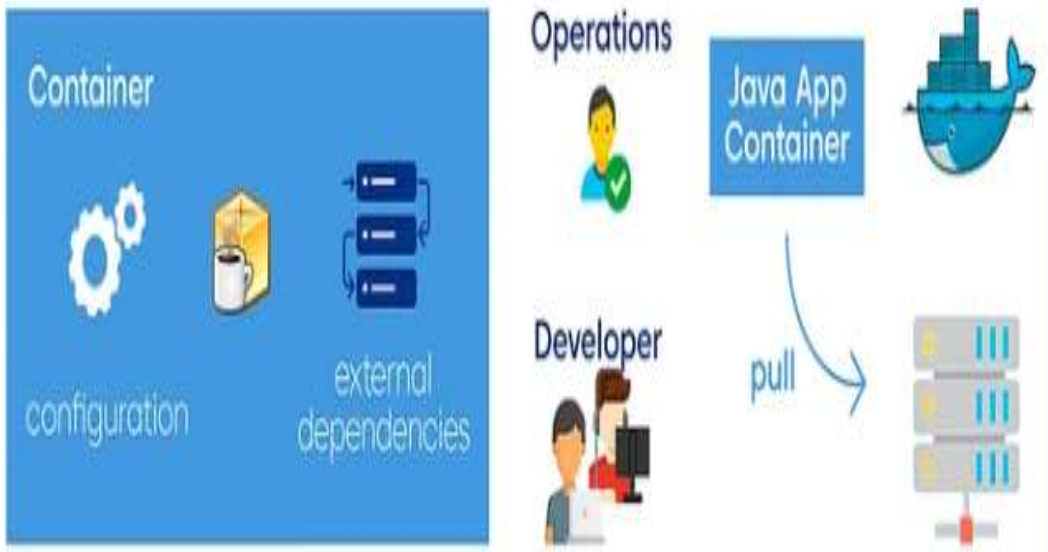


SCIFORCE

Application Deployment

After containers

- Developers and operations teams can stay on the same page through the overall development process thanks to packaging the application in a container
- The only environmental configuration needed on the server – Docker Runtime

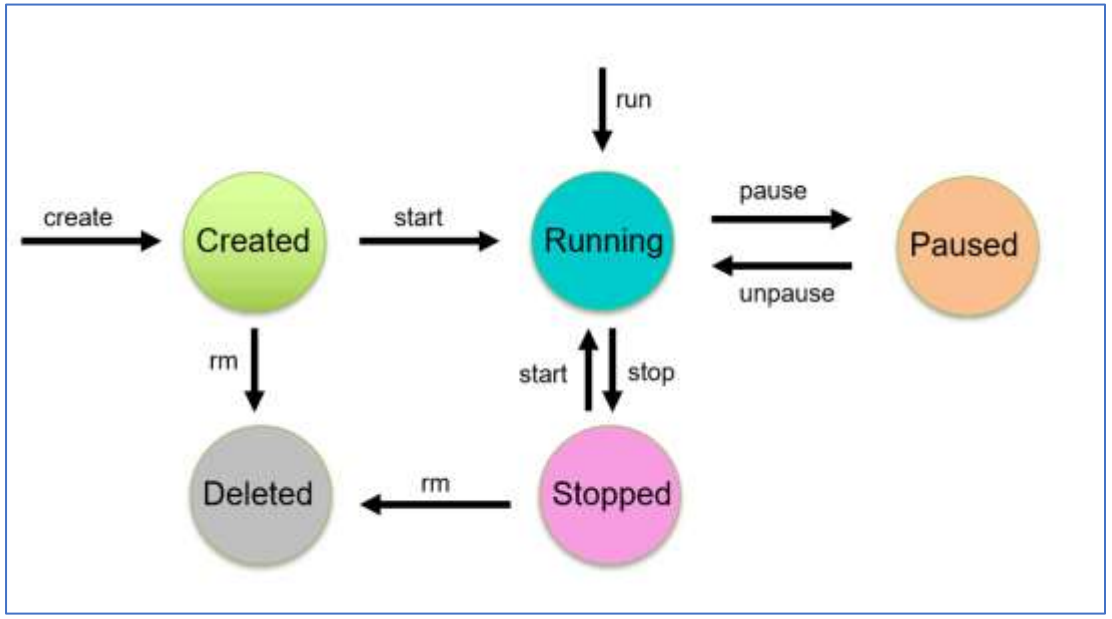


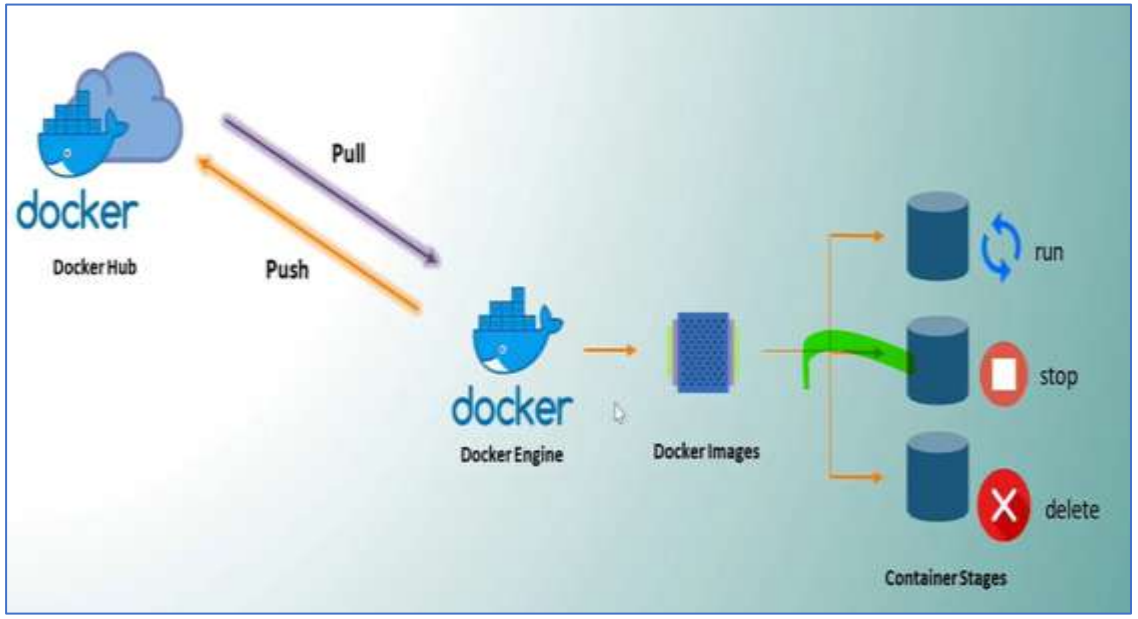
SCIFORCE

Docker and Docker Lifecycle

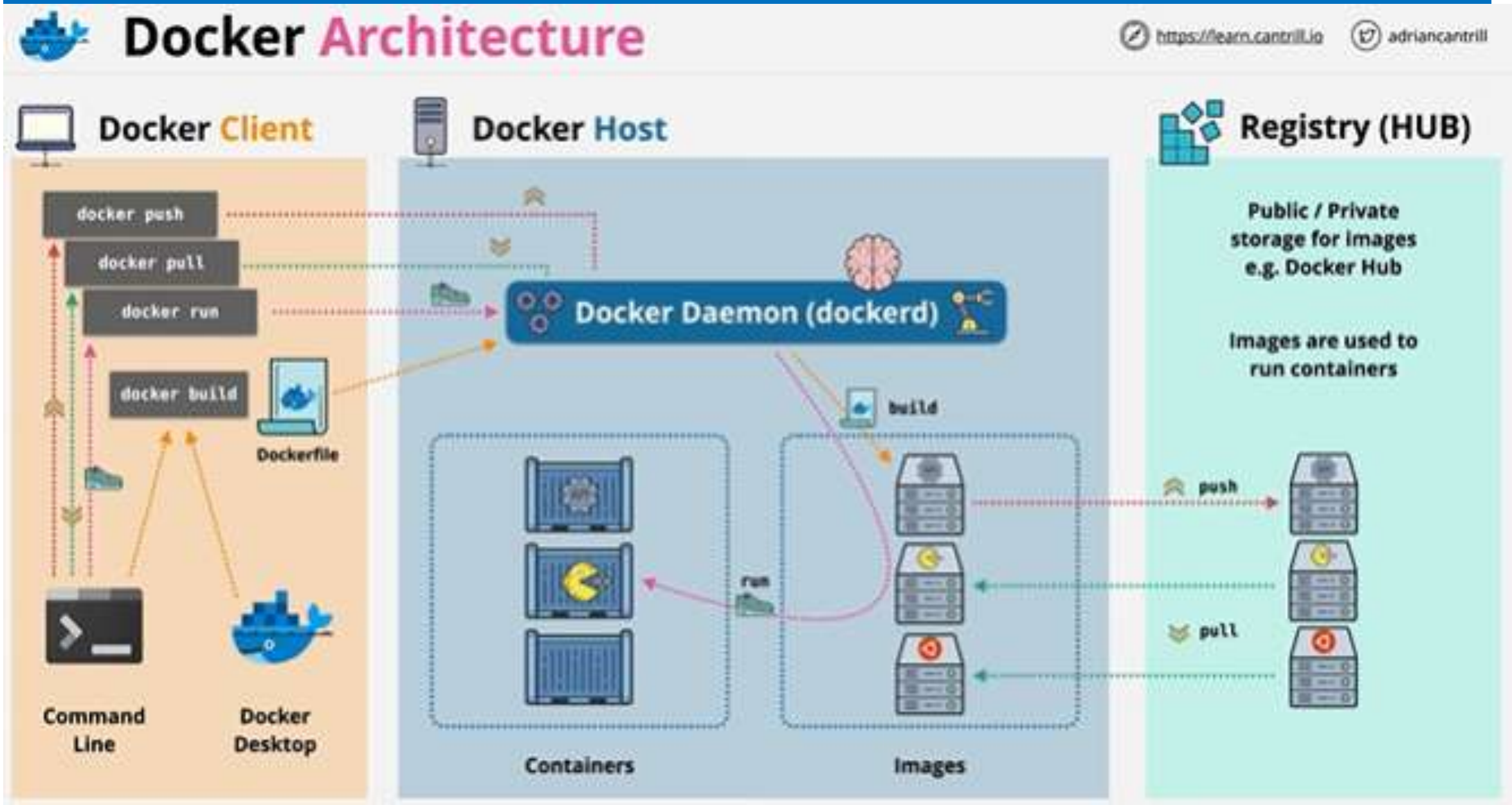
Lifecycle Stages: Create, Start, Stop, Restart, Pause, Unpause, Destroy.

Commands: docker create → docker start → docker run → docker stop → docker restart →docker pause → docker unpause → docker rm





Docker Architecture

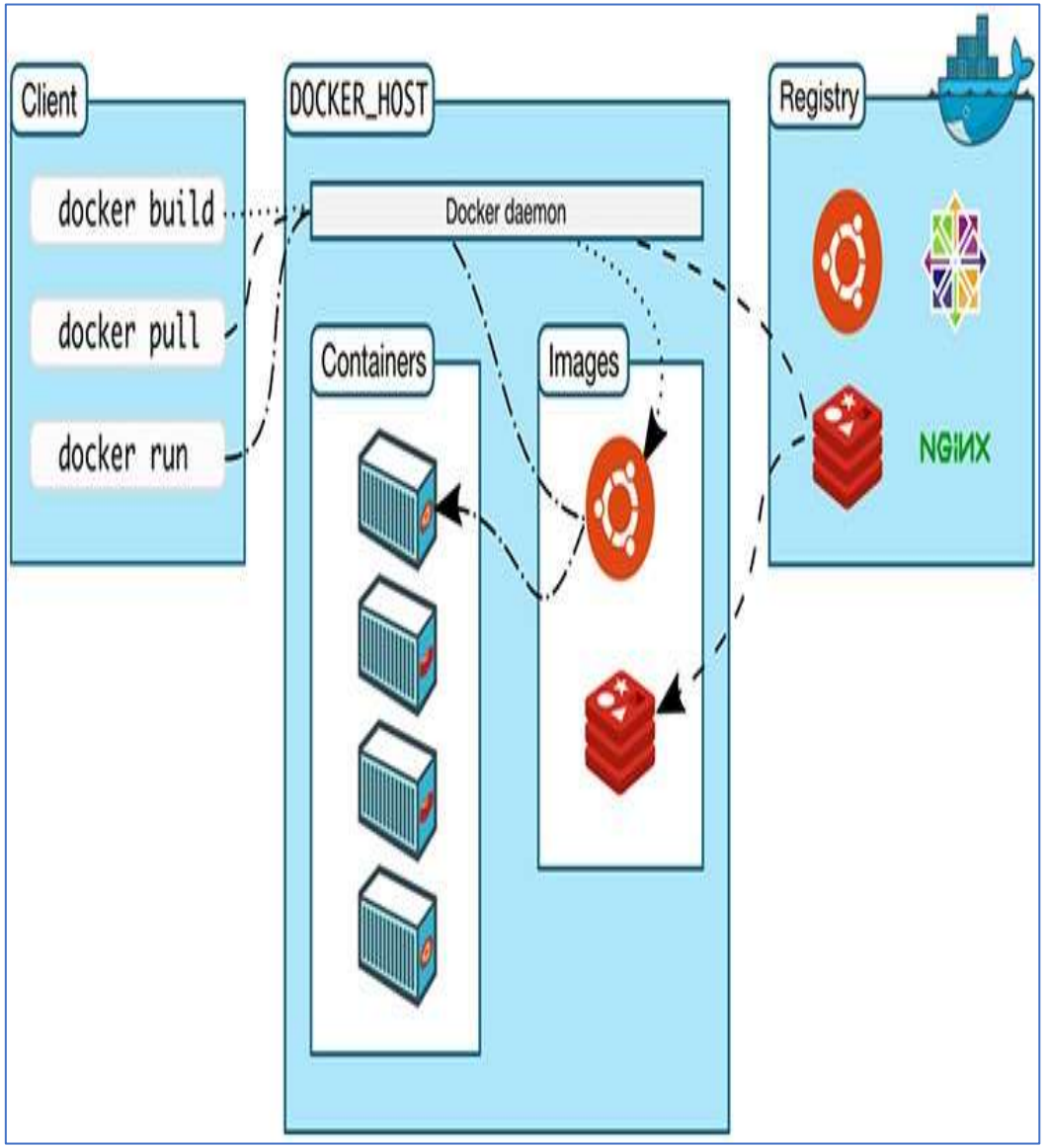


Components of Docker Ecosystem

Docker Engine: The core software that enables you to create, run, and manage containers. It allows developers to package applications and their dependencies into containers.

Docker Daemon (dockerd): A server-side program that manages Docker containers. It listens for API requests and handles the creation, running, and monitoring of containers.

Docker Client (docker): The command-line interface (CLI) that allows users to interact with the Docker daemon. Users can issue commands to create, manage, and deploy containers.



Components of Docker Ecosystem

Images: Read-only templates used to create containers. An image includes everything needed to run an application, such as code, libraries, and environment variables.

Containers: Instances of Docker images. Containers are isolated environments where applications run. Each container shares the host OS kernel but maintains its filesystem.

Docker Hub: A cloud-based registry for sharing and distributing Docker images. Developers can pull base images or push their own images to share with others.

Docker Compose: A tool for defining and running multi-container Docker applications. Using a YAML file, you can specify the services, networks, and volumes used in your application.

Components of Docker Ecosystem

Docker Swarm: A native clustering and orchestration tool for Docker. It allows users to manage a group of Docker hosts (nodes) as a single virtual system, supporting load balancing and service discovery.

Docker Network: Networking components allowing containers to communicate with each other and the outside world. Docker supports multiple types of networks (bridge, overlay, etc.) to cater to different use cases.

Docker Volume: Persistent storage mechanism for containers, allowing data to be stored outside the container filesystem. Volumes are used to share data between containers or retain data after a container is deleted.

Interaction Flow

Development: Developers create a Docker image that packages the application and its dependencies.
Deployment: The image is pushed to Docker Hub or a private registry. It can then be pulled from any host running Docker.
Execution: The Docker client communicates with the Docker daemon to run an image as a container.
Scaling: For applications requiring multiple instances, Docker Compose or Docker Swarm can be used to orchestrate multiple containers.
Networking & Storage: Containers communicate through defined networks, and persistent data is managed through Docker volumes.

Installing Docker

Step 1: Enable WSL 2

Docker Desktop requires the Windows Subsystem for Linux (WSL) 2. Here's how to enable it:

- **Open PowerShell as Administrator**: Right-click the Start button and select "Windows Terminal (Admin)" or search for "PowerShell" in the Start menu, then right-click and choose "Run as administrator".
- **Enable WSL2**: wsl –install

```
PS C:\WINDOWS\system32> e:
PS E:\> wsl --install
Installing: Ubuntu
Ubuntu has been installed.
Launching Ubuntu...
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: bujji
New password:
Retype new password:
No password has been supplied.
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.153.1-microsoft-standard-WSL2 x86

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


This message is shown once a day. To disable it please create the
/home/bujji/.hushlogin file.
bujji@spiderman:~$
```

Installing Docker

```
PS E:\> wsl
bujji@spiderman:/mnt/e$
```

This command installs WSL and sets WSL 2 as the default version. Restart your computer if prompted.

Step 2: Download Docker Desktop

- Go to the Docker Desktop download page: Docker Desktop for Windows.
- Download the installer: Click the download button to get the Docker Desktop installer.

Step 3: Install Docker Desktop

- **Run the Installer**:
  - Locate the downloaded installer file (usually in your Downloads folder) and double-click it to run.
  - Follow the on-screen instructions to complete the installation.
- **Restart Your Computer**: You might be prompted to log out and back in or restart your computer to complete the installation.

Step 4: Configure Docker Desktop

- **Launch Docker Desktop**: After installation, open Docker Desktop from the Start menu.
- **Follow Initial Setup**: Docker Desktop will guide you through the initial setup, including enabling WSL 2 integration.

Installing Docker

Step 5: Verify Installation

- **Open a Terminal**: You can use Command Prompt, PowerShell, or Windows Terminal.
- **Check Docker Version**: docker --version (https://docs.docker.com/go/guides/)

```
bujji@spiderman:/$ docker --version
Command 'docker' not found, but can be installed with:
sudo snap install docker          # version 24.0.5, or
sudo apt  install podman-docker   # version 3.4.4+ds1-1ubuntu1.22.04.2
sudo apt  install docker.io       # version 24.0.5-0ubuntu1~22.04.1
See 'snap info docker' for additional versions.
```

```
bujji@spiderman:/$ pwd
/
bujji@spiderman:/$ cd $HOME
bujji@spiderman:~$ pwd
/home/bujji
bujji@spiderman:~$ sudo snap install docker
[sudo] password for bujji:
Download snap "docker" (2932) from channel "stable"                    100% 1.77MB/s 285ms
```

```
bujji@spiderman:~$ sudo snap install docker
[sudo] password for bujji:
docker 24.0.5 from Canonical✓ installed
```

```
bujji@spiderman:~$ docker --version
Docker version 24.0.5, build ced0996
```

Installing Docker

Step 5: Verify Installation

- **Open a Terminal**: You can use Command Prompt, PowerShell, or Windows Terminal.
- **Check Docker Version**: docker --version (https://docs.docker.com/go/guides/)

```
devops@spiderman:/mnt/e$ cd $HOME
devops@spiderman:~$ pwd
/home/devops
devops@spiderman:~$ docker --version
Docker version 27.2.0, build 3ab4256
devops@spiderman:~$
devops@spiderman:/mnt/e$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Download complete
Digest: sha256:305243c734571da2d100c8c8b3c3167a098cab6049c9a5b066b6021a60fcb966
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Common Docker Commands

- **List Images:** docker images
- **List Containers:** docker ps
- **Run Container:** docker run -itd ubuntu
- **Stop Container:** docker stop <container-id/name>
- **Remove Container:** docker rm <container-id/name>
- **Pull Image:** docker pull <container-id/name>

Creating a Docker Hub Account

- Visit Docker Hub, sign up, verify email, set up repositories.

Committing Changes to a Container

Command: docker commit <container_id> <new_image_name>
Example: docker push myusername/myrepository:latest

DockerFile

Definition: A text file containing instructions for building a Docker image.
Syntax: FROM <base_image>, RUN <command>, COPY <source> <destination>, CMD <command>.

Committing Changes to a Container

Command: docker commit <container_id> <new_image_name>
Example: docker push myusername/myrepository:latest

Running Sample DockerFile

- **Example Dockerfile:**

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

- **Build Image:** docker build -t myapp .

- **Run Container:** docker run -p 8080:8080 myapp

Common Docker Commands

Container Management

- List Running Containers: docker ps
- List All Containers (Running and Stopped): docker ps -a
- Create a Container: docker create [image]
- Start a Container: docker start [container]
- Stop a Container: docker stop [container]
- Remove a Container: docker rm [container]

Common Docker Commands

Container Management

- Rename a Container:
  docker rename [container] [new-name]

- Force Remove a Container:
  docker rm -f [container]

- View Logs for a Container:
  docker logs [container]

- Copy Files Between Container and Host:
  docker cp [file-path] CONTAINER:[path]

Common Docker Commands

Image Management

- List Images: docker images
- Pull an Image: docker pull [image]
- Build an Image: docker build -t [image] .
- Tag an Image: docker tag [image] [new-tag]
- Remove an Image: docker rmi [image]
- Search for Images: docker search [term]

Common Docker Commands

Docker Hub Management

- Create a Docker Hub Account: Visit Docker Hub and sign up.
- Push an Image to Docker Hub: docker push [username]/[repository]:[tag]
- Pull an Image from Docker Hub: docker pull [username]/[repository]:[tag]

Dockerfile Management

- Create a Dockerfile: A text file with instructions for building a Docker image.
- Build an Image from Dockerfile: docker build -t [image] .
- Run a Container from a Dockerfile: docker run -p host_port:container_port [image]

Code Sample Commands: for Node.js

docker build -t node-app:1.0 .

docker run -d -p 3000:3000 node-app:1.0