Docker Fundamentals

Shailendra Chauhan

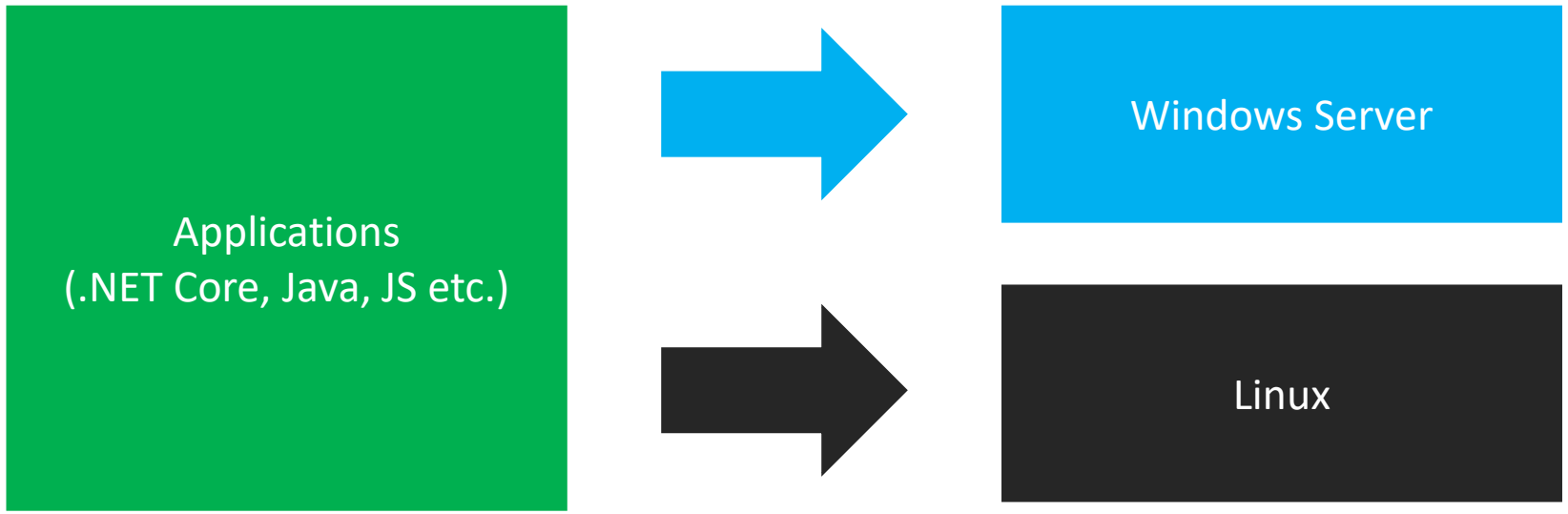
Microsoft MVP, Technical Consultant and Corporate Trainer

DotNetTricks

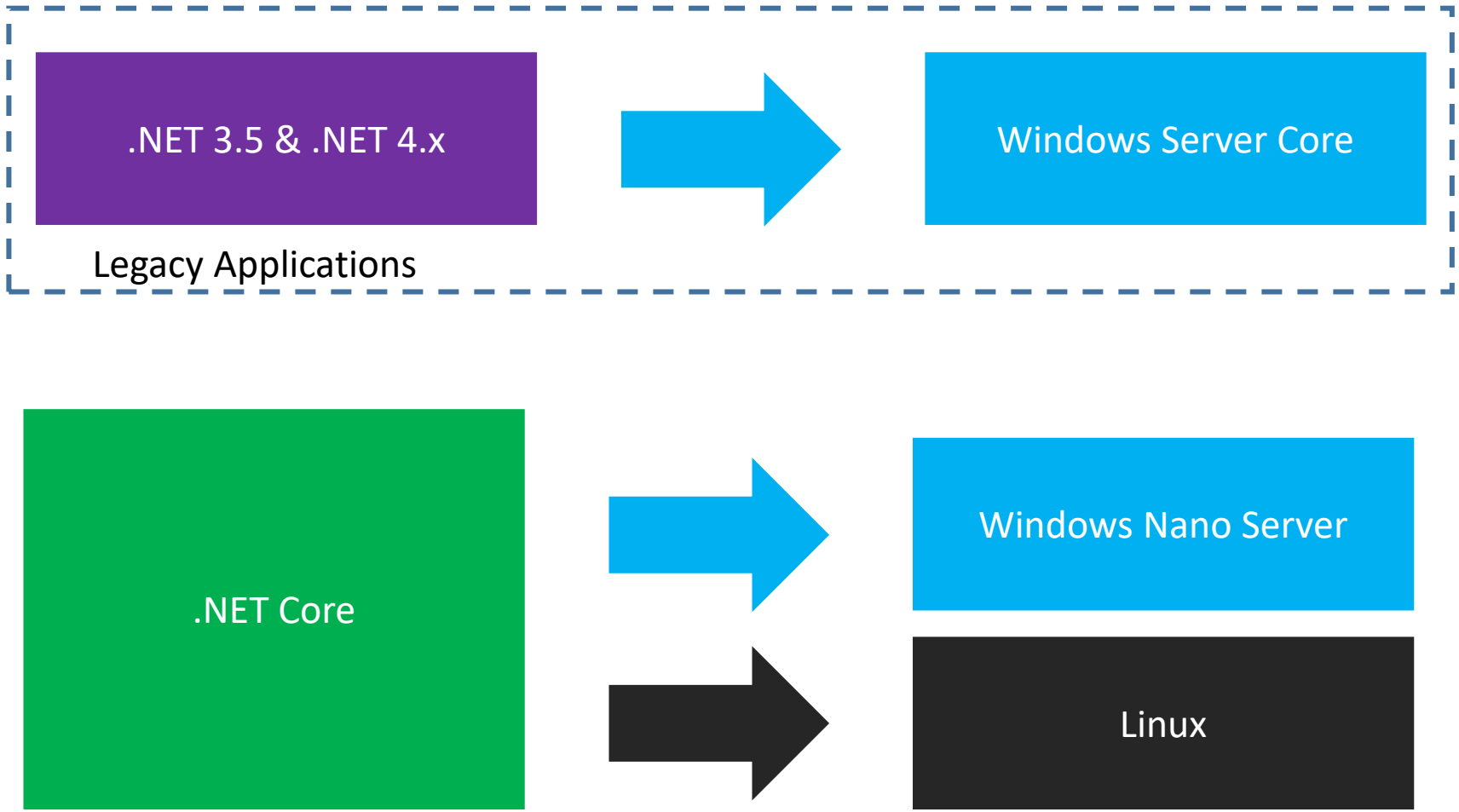
Agenda

- Docker For Developers
- Docker for .NET Applications
- Docker Community vs. Docker Enterprise
- Docker Basics - File, Image, Container and Registry
- Docker Hub
- Container Life Cycle
- Docker Engine
- Docker Storage

Docker For Developers



Docker For .NET Applications



DotNetTricks

Docker Community vs. Docker Enterprise

Docker Community

- For developers and small organizations
- Free
- Stable version (every 3 months)
- edge version (every month), with cutting edge features
- Docker's CLI and API are used for create, build and deploy application

Docker Enterprise

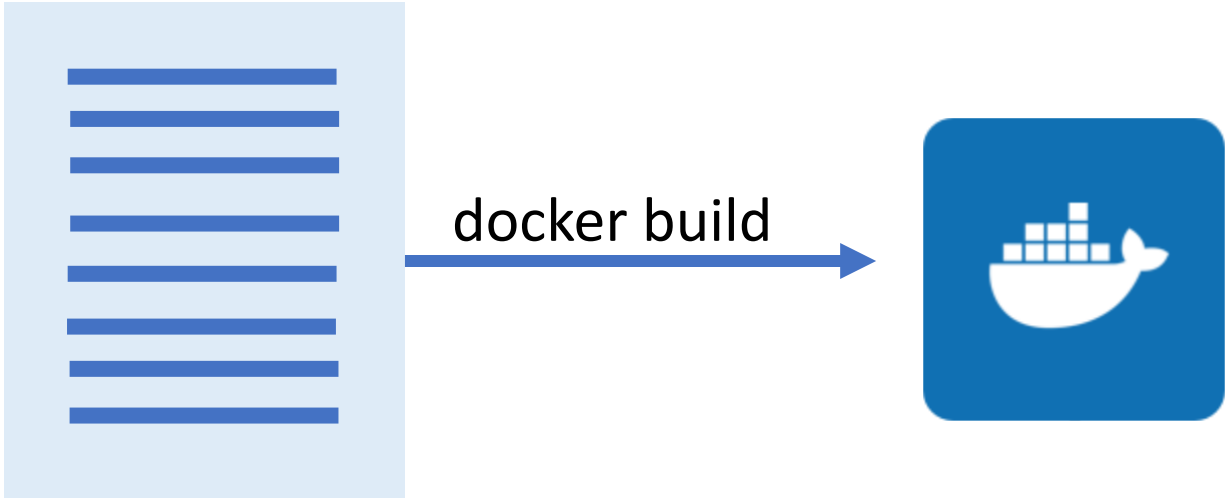
- For business and production apps
- Subscription model
- Stable version (every 3 months)
- Additional enterprise features (management, security etc.)
- Universal Control Plane is used for create, build and deploy application

Docker Basics

- Docker File
- Docker Image
- Docker Container
- Docker Registry

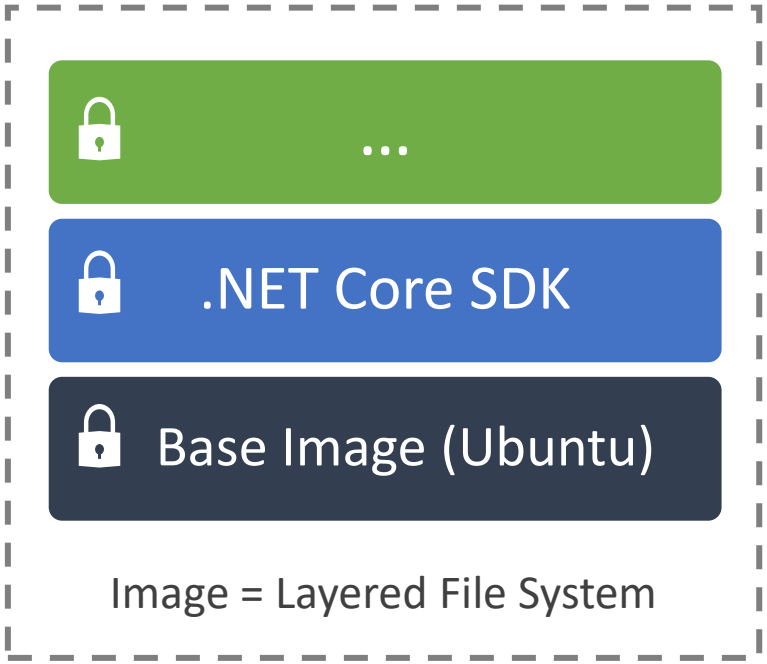
Docker File

- A simple text file that contains commands to build a docker image.
- This file doesn’t have any extension.

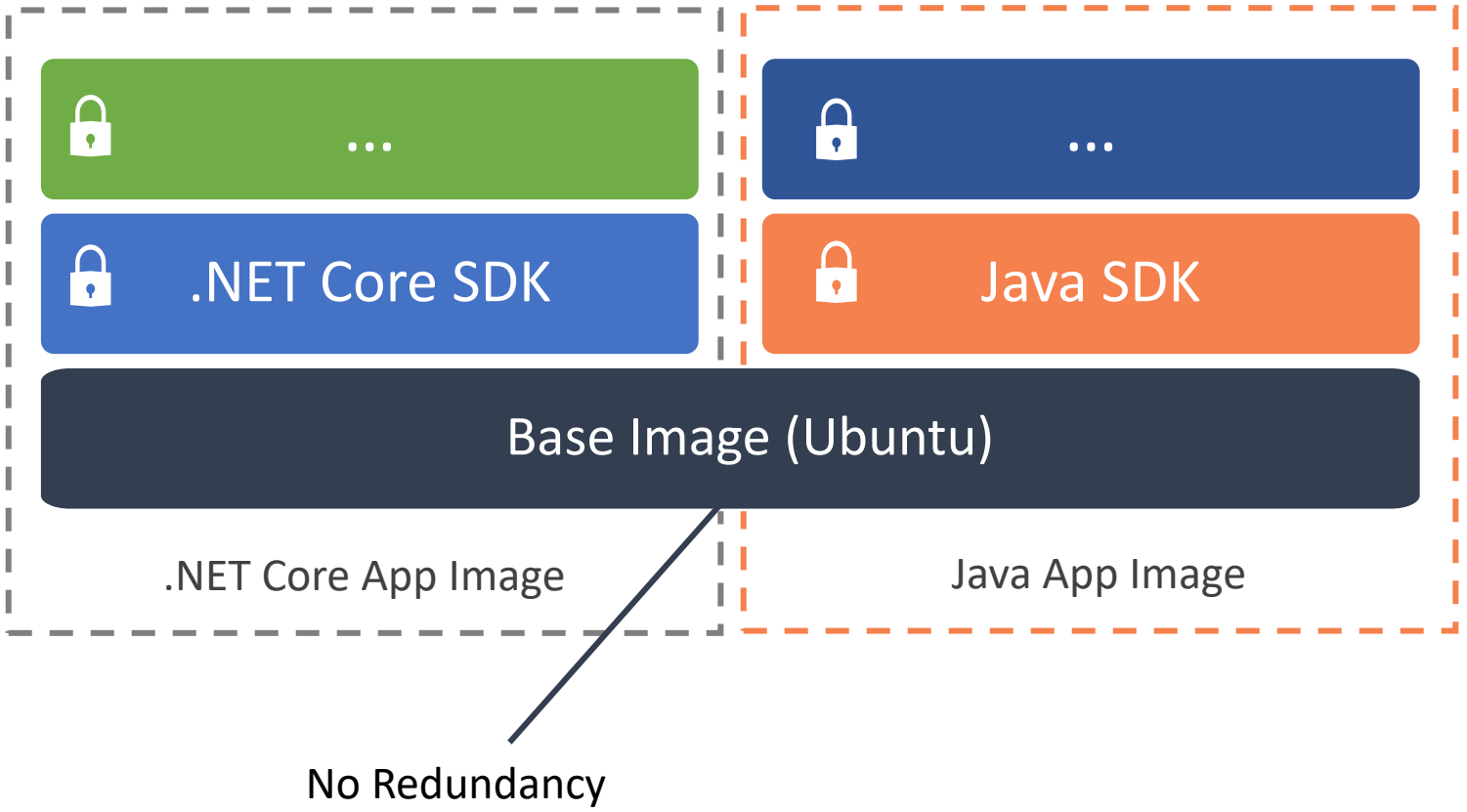


Docker Image

- A lightweight, standalone and executable package of software.
- Includes everything which is needed to run an application like code, runtime, system tools, system libraries and settings.
- An image is stack of multiple read only layers referencing another images.
- Created by *docker build command.*
- Stored in Docker registry (eg. Docker Hub).



Layer Sharing between Images



Docker Container

- Container is an running instance of a docker image
- An isolated and secured shipping container
- Run by docker run command

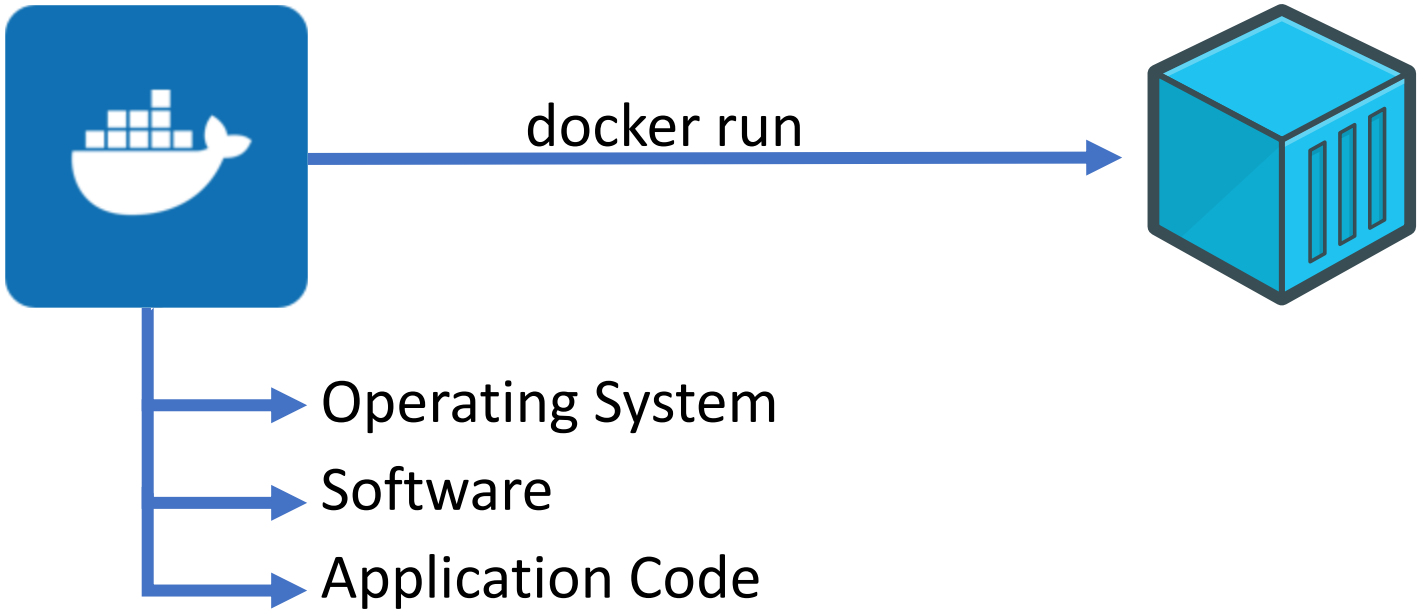
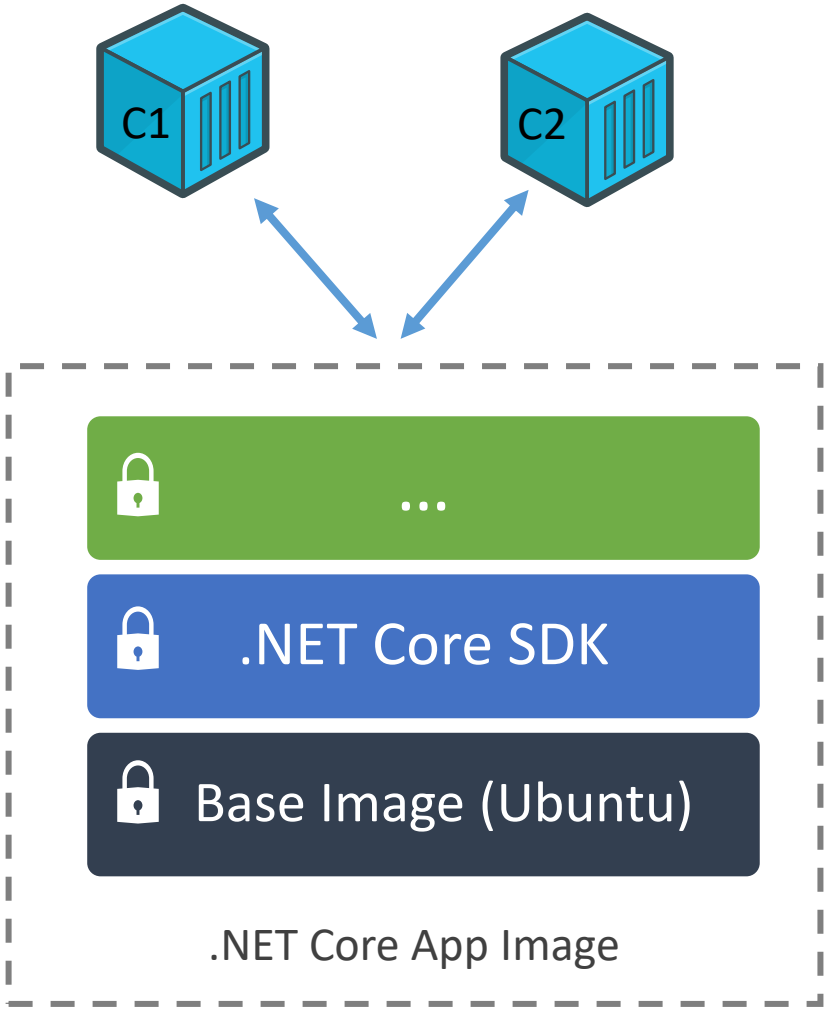
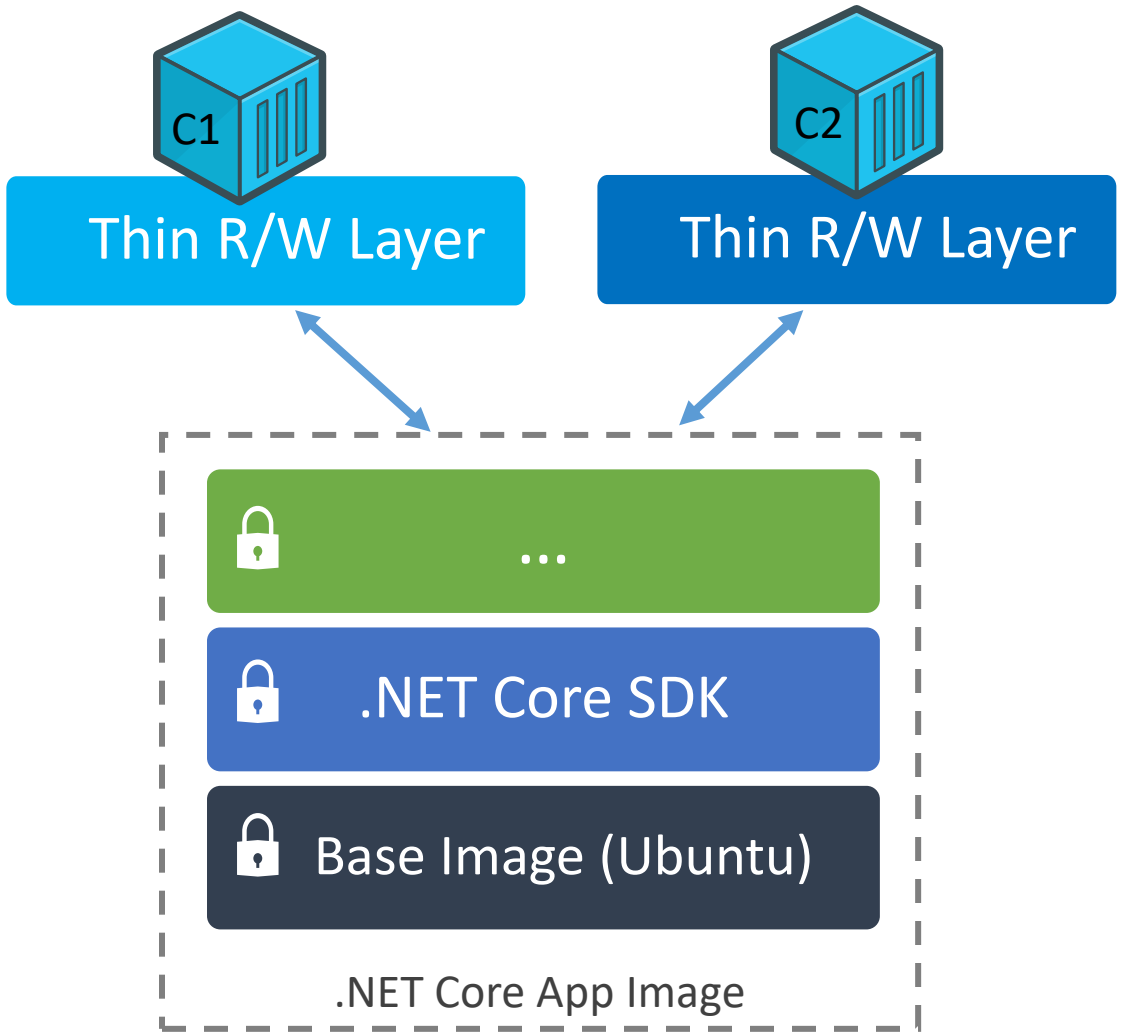


Image Sharing Between Containers

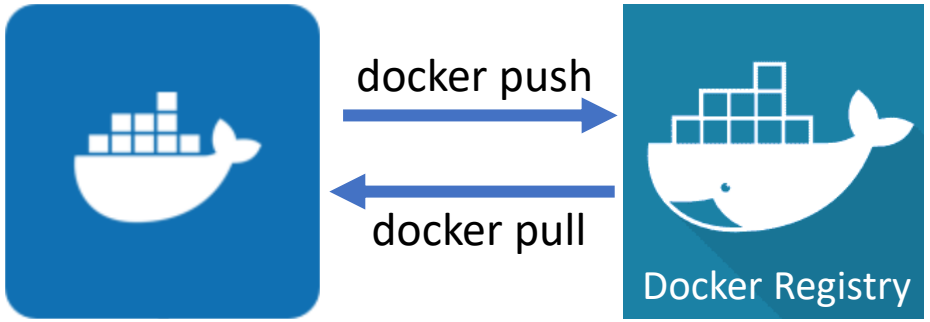


Containers and Layers



Docker Registry

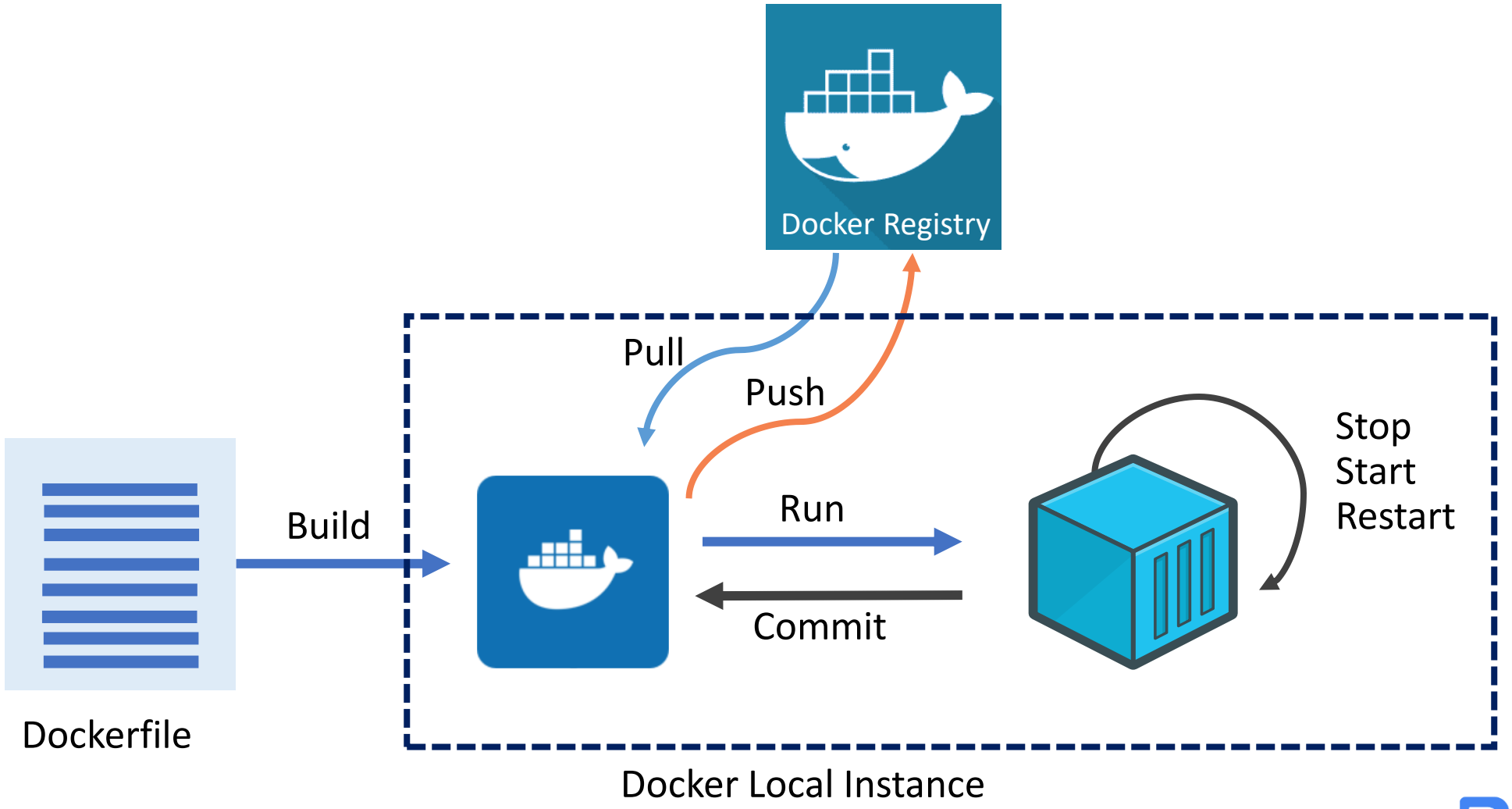
- A Service to host Docker images
- Multiple options are available:
  - Docker Hub - (Free for public images and Paid for private images)
  - Docker Trusted Registry - (on-prem or on-cloud)



Docker Hub

- A cloud-based repository for public and private images
- Here, Docker users and partners create, test, store and distribute container images
- Users can host and share their own custom images
- For example, Nodejs image: https://hub.docker.com/_/node

Docker Container Life Cycle



DotNetTricks

Docker Container Lifecycle Contd..

- Conception - BUILD an Image from a Dockerfile
- Birth - RUN (create + start) a container
- Reproduction
  - COMMIT (persist) a container to a new image
  - RUN a new container from an image
- Sleep - KILL a running container
- Wake - START a stopped container
- Death - RM (delete) a stopped container
- Extinction - RMI (delete image) a container image

Dockerfile Instructions

- FROM : specifies the base (parent) image.
- WORKDIR : sets the working directory for the instructions that follow.
- RUN : runs a command and creates an image layer. Used to install packages into containers.
- COPY : copies files and directories to the container.
- CMD : provides command & arguments for a container. Support parameters override & one CMD.
- ENTRYPOINT : provides command & arguments for a container. Arguments persist.
- EXPOSE : exposes a port.

Dockerfile Instructions Contd..

- ADD : copies files and directories to the container.
- VOLUME : creates a directory at host to access and store persistent data.
- LABEL : provides metadata. Good place to include maintainer info.
- ENV : sets a persistent environment variable.
- ARG : defines a variable to pass to Docker at build-time.

Docker File Sample

```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1
WORKDIR /app
COPY . .
ENTRYPOINT ["dotnet", "run"]
```

```
docker build -t console:v1 .
docker run --name consoleapp console:v1
```

```
FROM nginx:1.18-alpine
WORKDIR /usr/share/nginx/html
COPY ./dist/angularapp .
```

```
docker build -t ngapp:v1 .
docker run -d -p 8080:80 --name ngapp ngapp:v1
```

Docker Multi-Stage Builds

- Contains multiple FROM statement in Docker file.
- Multistage Docker file is for creating intermediate images and from them create the final image to keep image small.
- Use *As* for labeling or naming your build stage.
- Only the instructions *RUN, COPY, ADD* create layers. Other instructions create temporary intermediate images, and do not increase the size of the build.

Docker Multi-Stage Builds

```
# Stage0: Download Base Image to host code
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1 AS base
WORKDIR /app

# Stage1: Create build environment
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build
WORKDIR /src
# Copy csproj and restore as distinct layers
COPY *.csproj .
RUN dotnet restore
# Copy everything else and build
COPY . .
RUN dotnet publish -c Release -o out

# Stage2: Build final runtime image
FROM base AS final
WORKDIR /app
COPY --from=build /src/out .
ENTRYPOINT ["dotnet", "ASPNETApp.dll"]
```

```
//create image
docker build -t aspnet:v1 .

//run container
docker run –d –p 8080:80 --rm --
name aspnetapp aspnet:v1
```

DotNetTricks

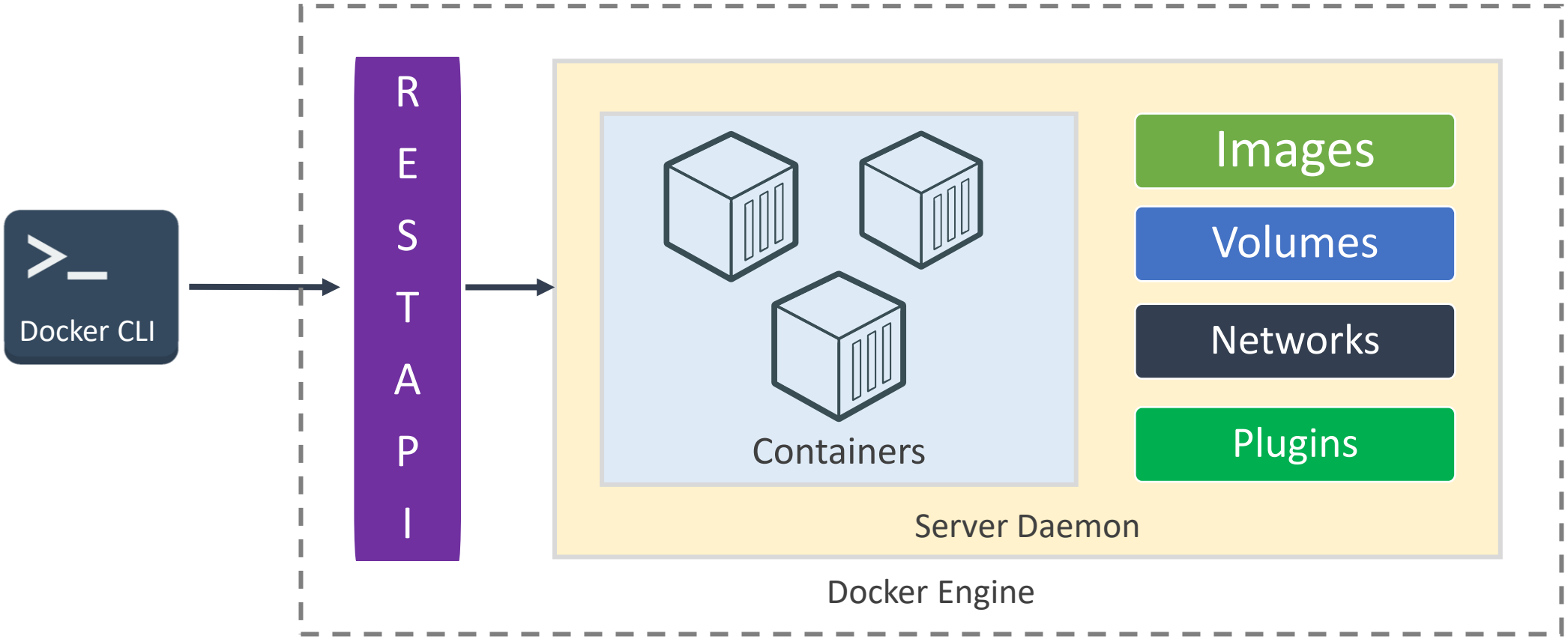
Docker Build Best Practices

- Use the smallest base image (alpine images) possible
- Reduce the amount of clutter in your image
- Use multi-stage builds
- Try to create images with common layers
- Tagging using semantic versioning
- Try to avoid installing unnecessary packages and dependencies
- Use a .dockerignore file to remove unnecessary content from the build context

Docker Engine

- A runtime to build and run container based applications which can run anywhere consistently on any infrastructure.
- Runs on various Linux (CentOS, Debian, Fedora, Oracle Linux, RHEL, SUSE, and Ubuntu) and Windows Server OS.
- Provides built in orchestration, container networking, out of the box security, volume and plugins.

Docker Engine Architecture



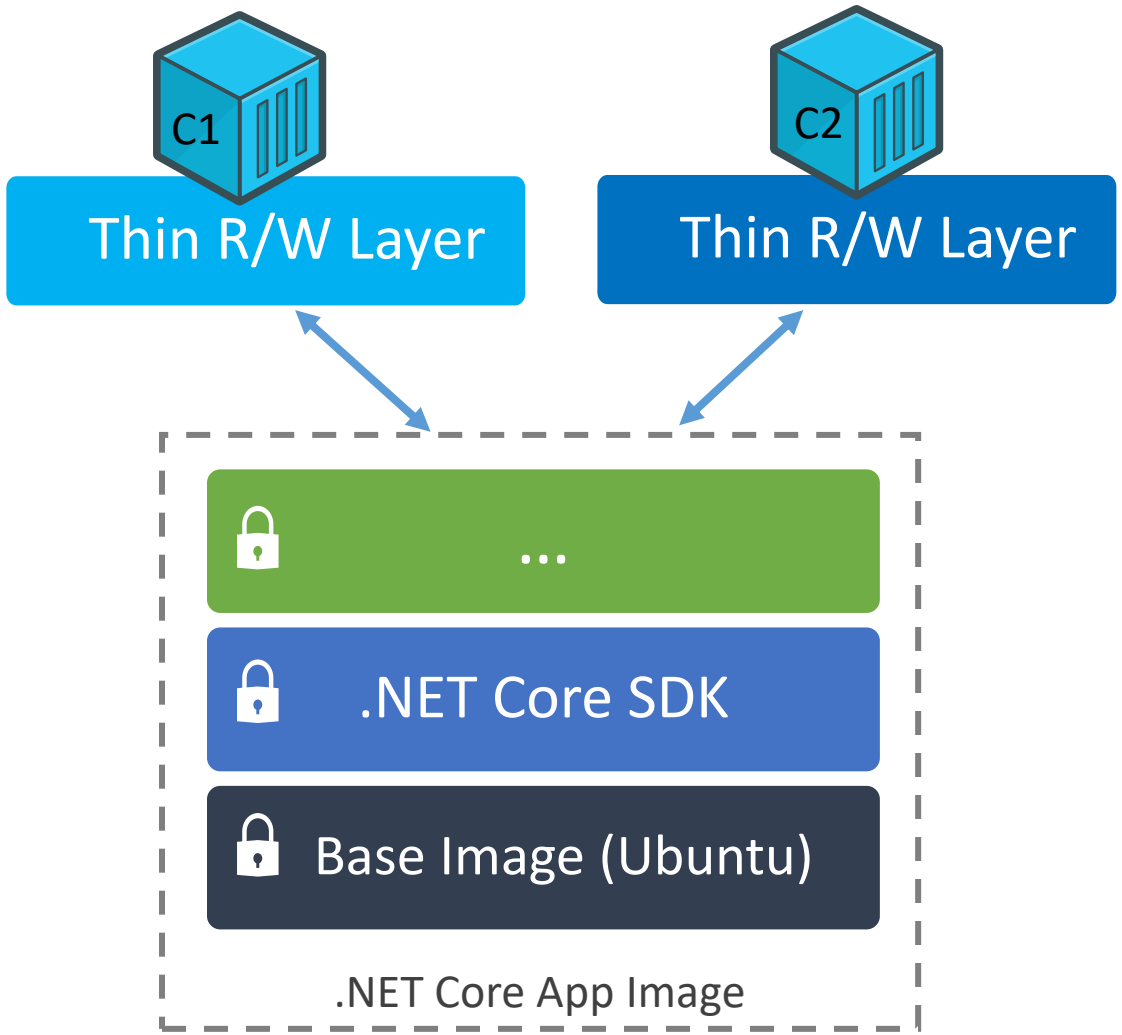
Docker Engine Contd..

- Docker engine or Docker is a client server application that builds and executes containers using Docker components.
- REST API is a primary mode of communication between Docker Client and Docker Daemon.
- Docker Daemon is a server which interacts with the operating system and performs all kind of services.
- The Docker Daemon listens for REST API request and performs the operation.

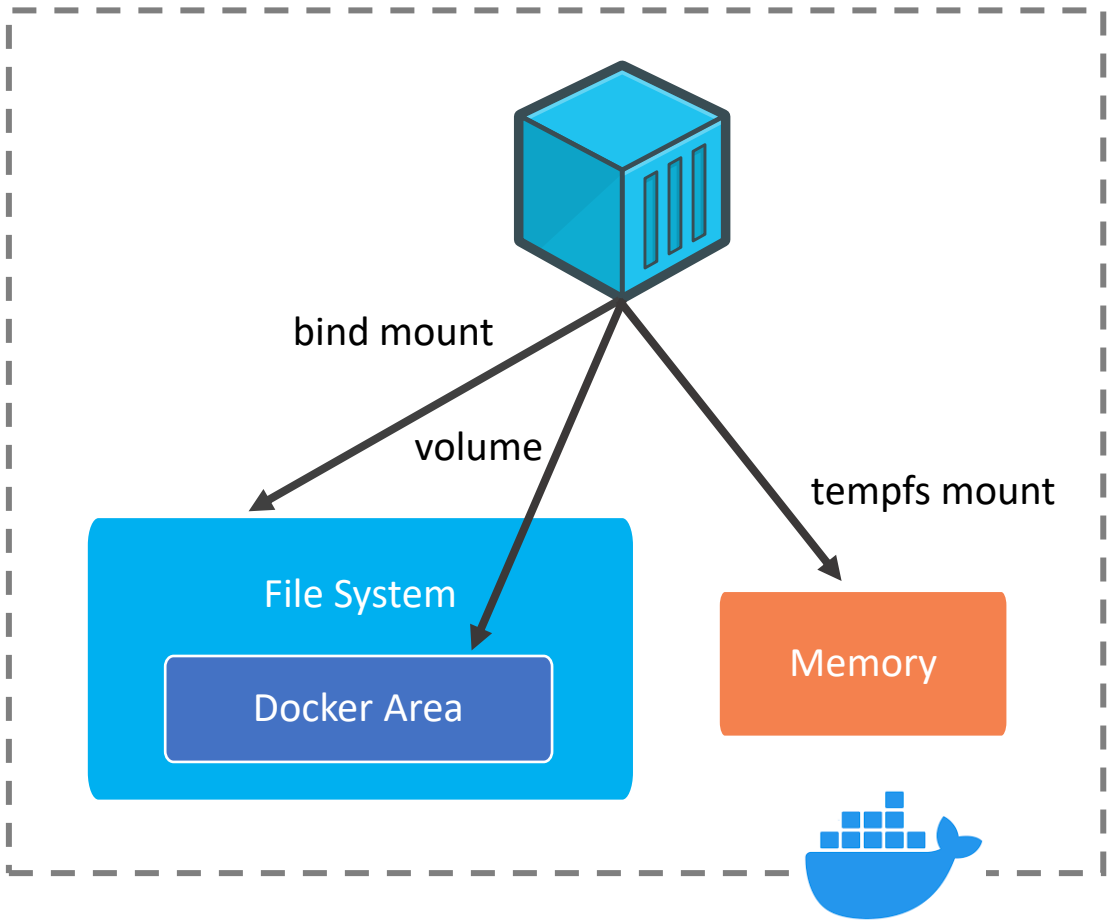
Docker Storage

- The main difference between a container and an image is the top writable layer where the container data is stored.
- When the container is deleted, the writable layer is also deleted. But the underlying image remains unchanged.
- Use Docker volume to share the same data by multiple containers.

Container R/W Layer



Docker Storage Options



Docker Storage Options Contd..

- **Bind mounts** may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host can modify them at any time.
- **Volumes** are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.
- **tmpfs** mounts are stored in the host system's memory only, and are never written to the host system's filesystem.

Docker Volumes

- Volumes are used to store data used by a container.
- Volumes can be shared among multiple containers.
- A volume does not increase the size of the containers since it exists outside the container lifecycle.
- Volumes can be used to share files between a host system and the Docker container
- A volume exist even after the container is deleted.
- Volumes work on both Linux and Windows containers

Docker Volumes Use Case

Docker volumes are helpful when you run database as a container for storing your data.