Agenda

Docker Fundamentals

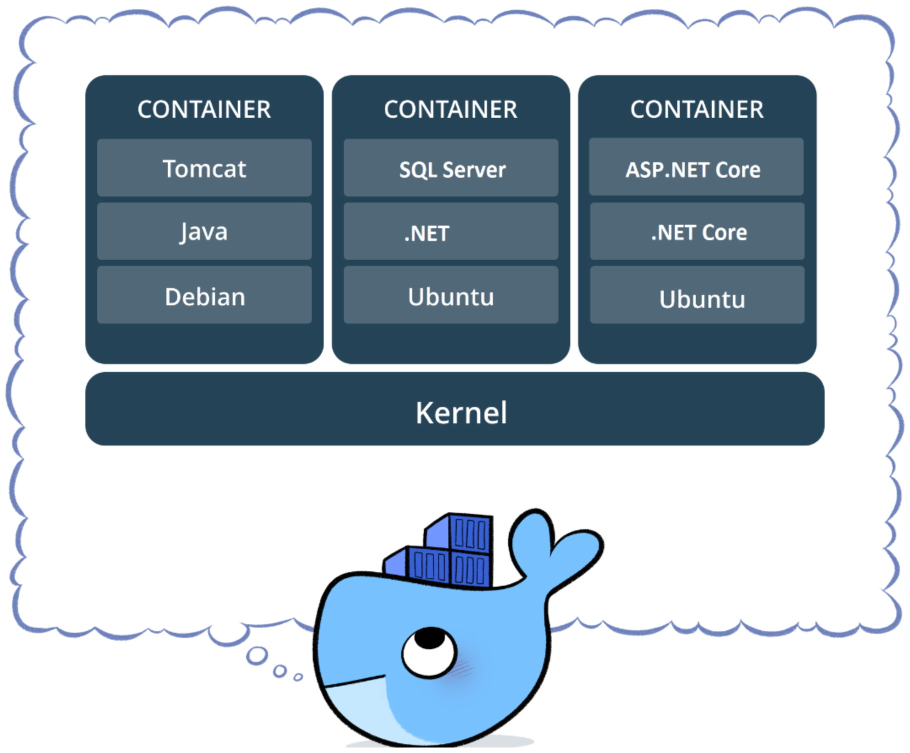
- Evolution of Computing
- What is Container?
- Virtual Machines vs Containers
- Introduction to Docker
- Docker CLI Commands
- Docker Container Life Cycle
- Docker Engine
- Docker Storage
- Docker Volume

Evolution of Computing

Development Process	Application Architecture	Deployment and Packaging	Application Infrastructure
Waterfall	Monolithic	Physical Server	Datacenter
Agile	N-Tier	Virtual Servers	Hosted
DevOps	Microservices	Containers	Cloud

What is Container?

- A container is a software that contains an application code and all its dependencies.
- Enables an application to run quickly in an isolated environment.
- Provide smooth migration from one computing environment to another.
- Share the same OS kernel
- Works with all major Linux & Windows Server



Source : www.docker.com

ScholarHat

Virtual Machine vs. Containers

- Hardware-level virtualization
- Fully isolated
- Isolated OS
- Having its own kernel
- Slower in start-up
- Many startup process
- Upfront resource allocation

- OS-level virtualization
- Process-level isolation
- Isolated processes/filesystems
- Host machine kernel is used
- Faster in start-up
- Single Start-up process
- No upfront resource allocation

ScholarHat

Introduction to Docker

- A light weight, open and secure platform for developing, shipping and running applications using container technology.
- Provides Container solutions for developers, architects, DevOps, and IT People.
- Run on most Linux distributions, Windows and Mac OS.
- Supported by most of cloud providers like AWS, Azure, Google etc.
- Provide Dev/Test, CI and DevOps platform for many use cases.

Docker Benefits

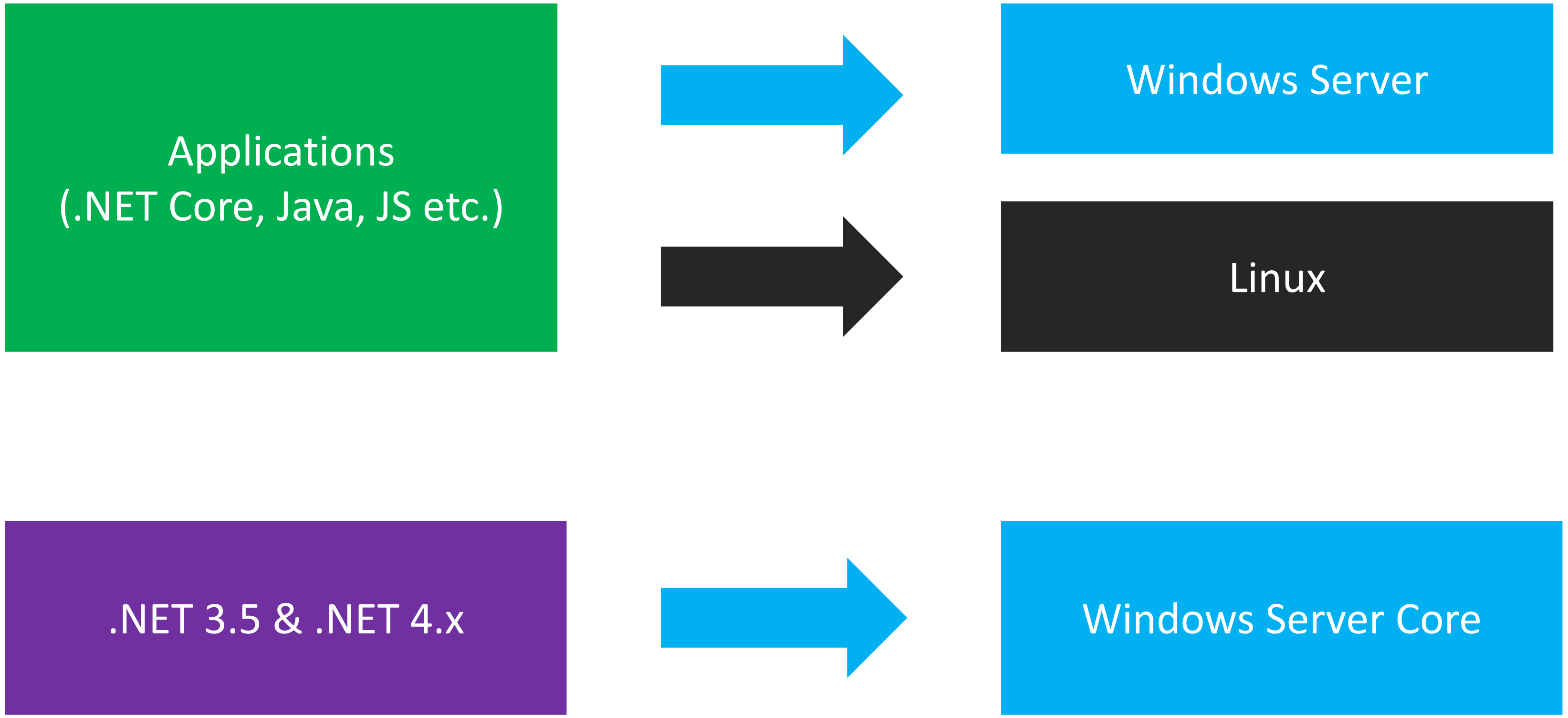
- Infrastructure Cost Savings
- Standardization and Productivity
- Isolation
- Security
- Makes app lifecycle efficient and consistent
- Continuous Deployment and Testing
- On Demand Scaling
- Multi-Cloud Platforms Support

ScholarHat

Docker Desktop

- An application for Mac and Windows to build production-ready container applications
- Enables to build and test Linux or windows container applications at local machine
- The Docker Desktop installation includes :
  - Docker Engine
  - Docker CLI client
  - Docker Compose
  - Docker Machine
  - Dashboard
- Containers and images created with Docker Desktop are shared between all user accounts on machines where it is installed.
- Available in two editions: Desktop Community and Desktop Enterprise

Docker For Developers



Getting Started with Docker

Demo

ScholarHat

nginx Container

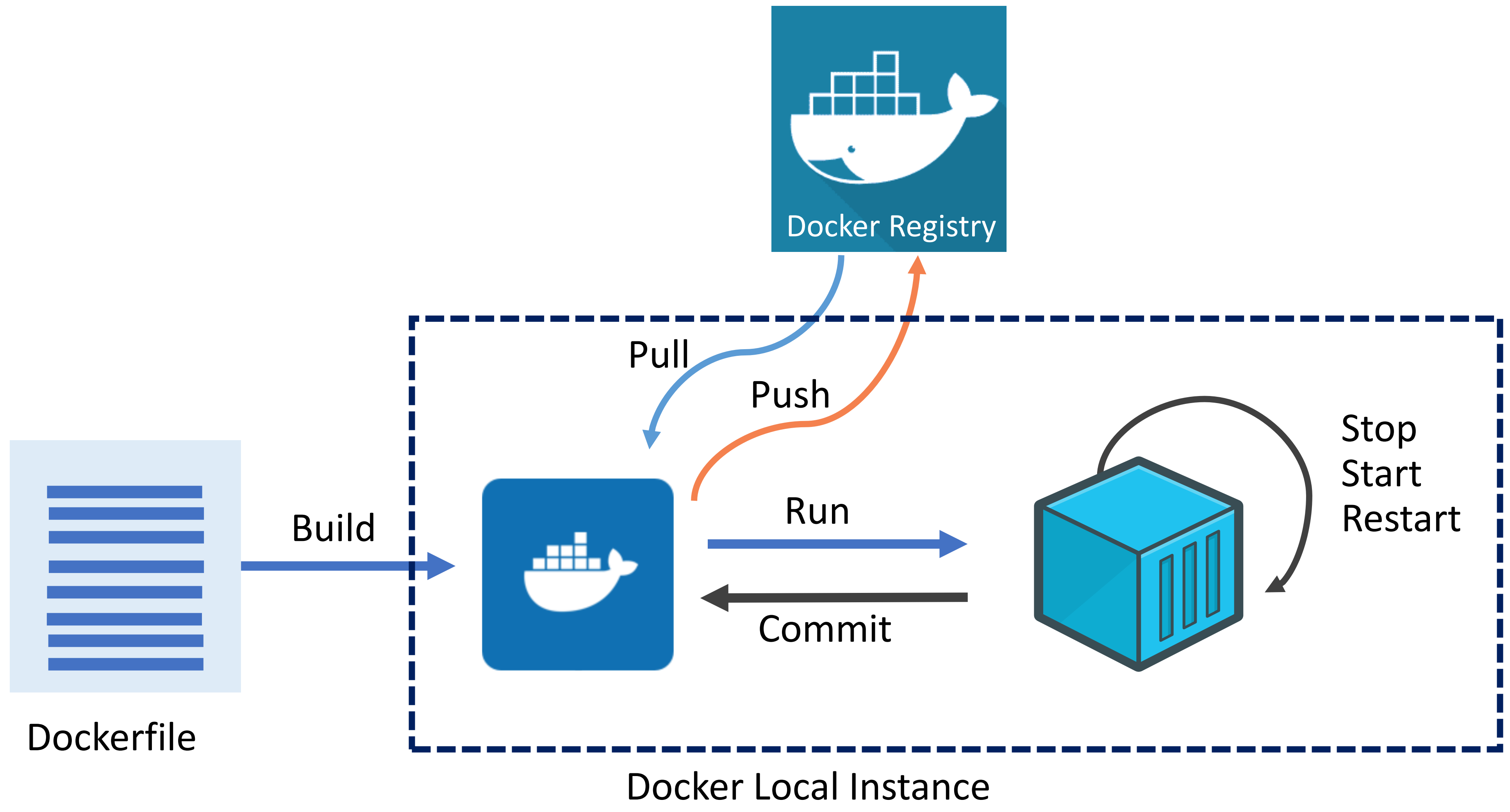
```
> docker pull nginx:1.25-alpine
> docker run --name nginx -p 8000:80 nginx:1.25-alpine
> docker run --name nginx -d -p 8000:80 nginx:1.25-alpine
```

Docker Basics

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

ScholarHat

Docker Container Life Cycle

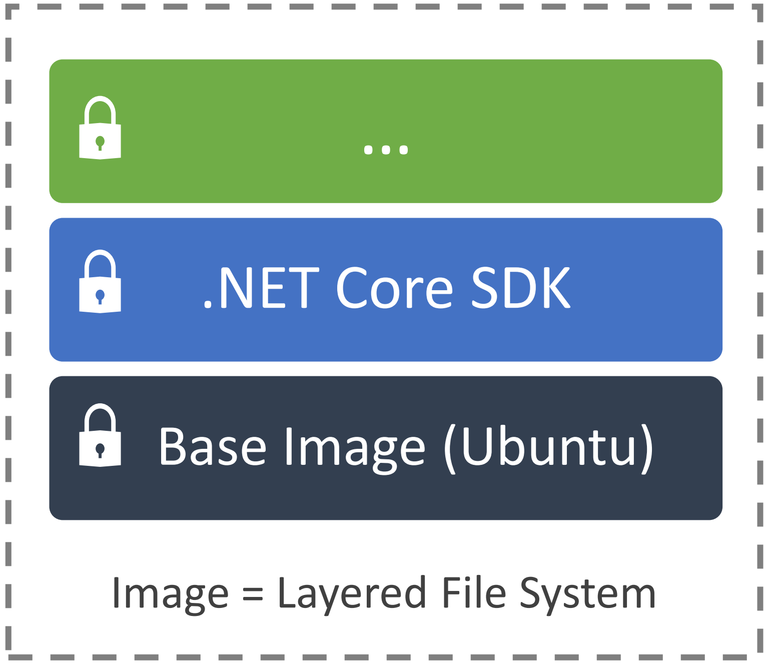


Docker Container Lifecycle Contd..

- 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

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 a stack of multiple read-only layers referencing another image.
- Created by *docker build command.*
- Stored in Docker registry (eg. Docker Hub).



Docker File Sample

```
FROM mcr.microsoft.com/dotnet/sdk:8.0
WORKDIR /app
COPY . .
ENTRYPOINT ["dotnet", "run"]
```

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

```
FROM nginx:1.25-alpine
WORKDIR /usr/share/nginx/html
COPY ./dist/angularapp .
```

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

ScholarHat

Docker Multi-Stage Builds

```
# Stage0: Download Base Image to host code
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app

# Stage1: Create build environment
FROM mcr.microsoft.com/dotnet/sdk:6.0 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 .

//or
docker build -t aspnet:v1 –f ./Dockerfile .

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

//OR
docker run -d –p 8080:5000 --rm --name
aspnetapp aspnet: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.

ScholarHat

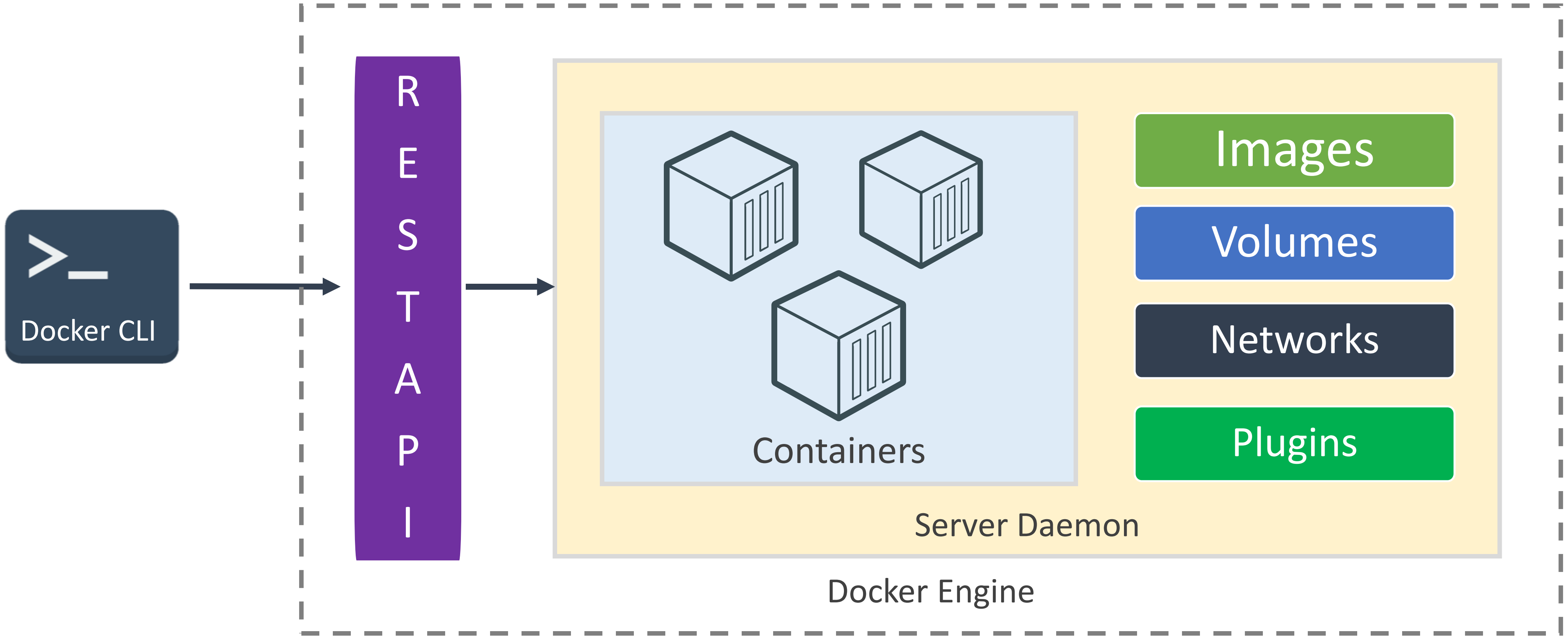
Docker Build Best Practices

- 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

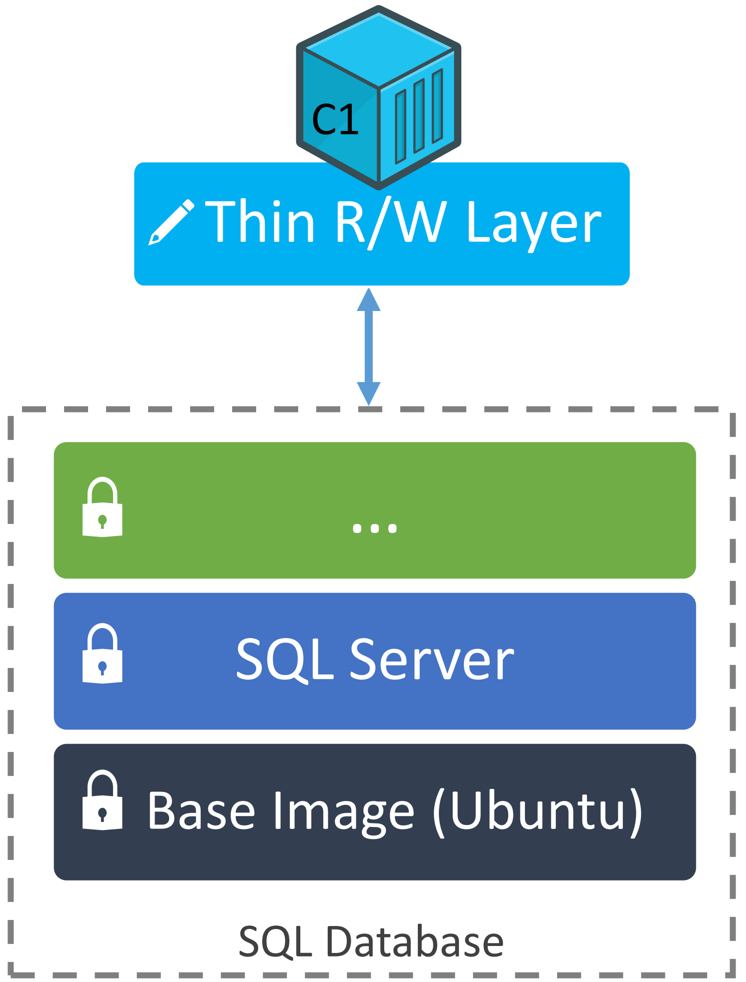


Docker Storage

- 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.

ScholarHat

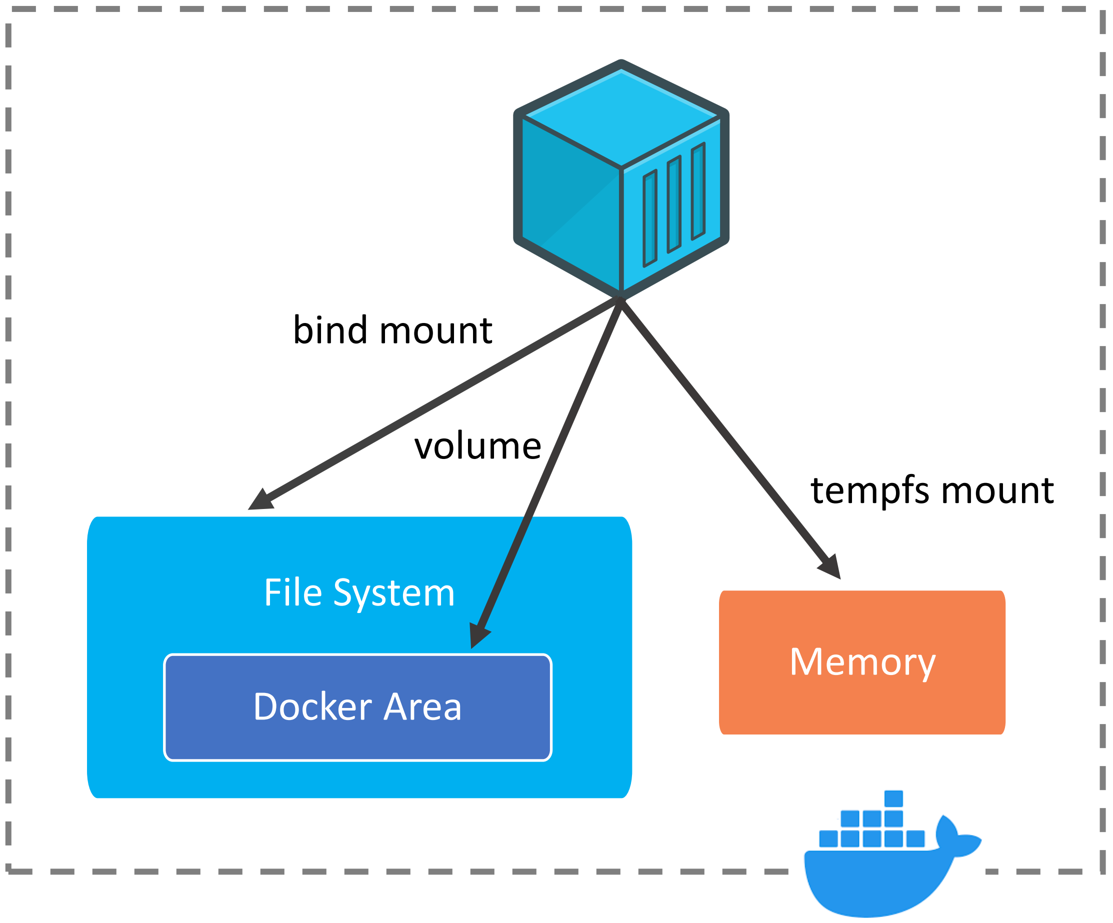
Container R/W Layer



```
docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=YourStrong@Passw0rd"
-p 5020:1433 --name sqlserver -d mcr.microsoft.com/mssql/server:2019-
CU14-ubuntu-20.04
```

```
Server: localhost,5020
User: sa
Pwd: YourStrong@Passw0rd
```

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.

ScholarHat

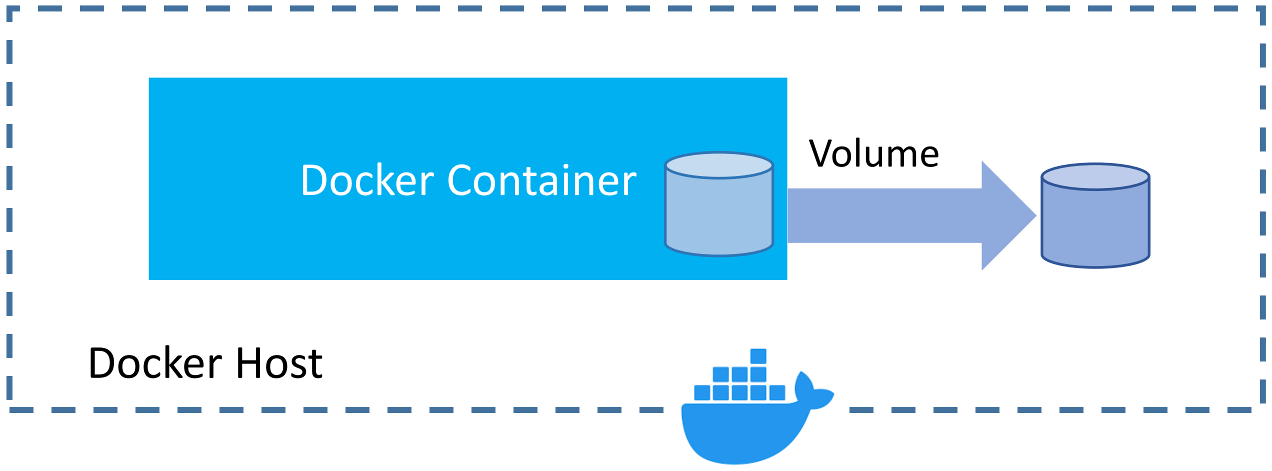
Docker Volumes

- 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 exists even after the container is deleted.
- Volumes work on both Linux and Windows containers

ScholarHat

Docker Volumes Use Case

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



Server: localhost,5030
User: sa
Pwd: YourStrong@Passw0rd

```
docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=YourStrong@Passw0rd" -p 5030:1433 --name sqlserver -v
c:/docker/databases:/var/opt/mssql/data -d mcr.microsoft.com/mssql/server:2019-CU14-ubuntu-20.04
```

ScholarHat