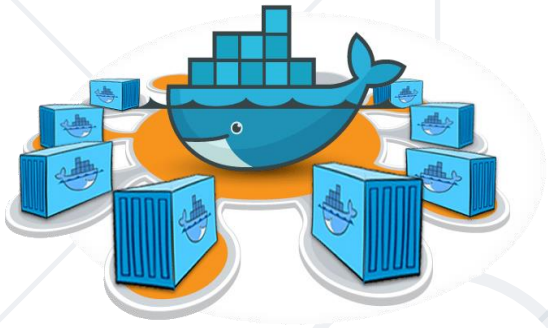
Containers, Docker and Docker Compose

Containers, Docker, Docker Compose



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#QA-BackEnd

Table of Contents

Containerization

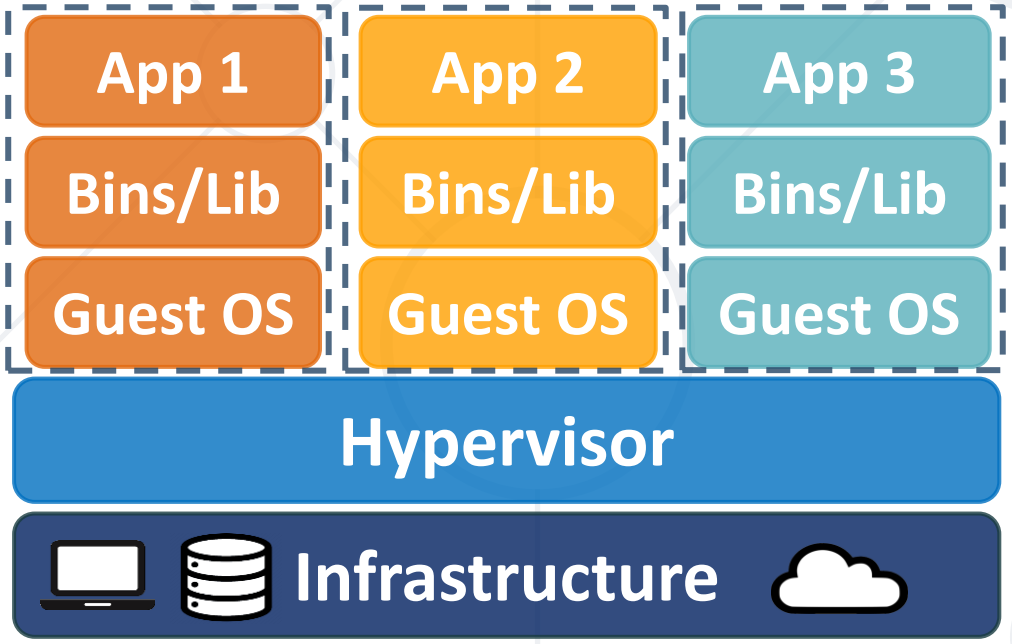
Overview, VMs VS Containers, Advantages

Containerization

- **Containerization**
  - Approach in which an **app** or **service** is packaged as a **container**
- **Image**
  - Read-only template that contains a set of instructions for creating a container
    - It contains software, packaged with its dependencies and configuration
    - Designed to run in a virtual environment
- **Container**
  - A runnable instance of an image

VMs vs Containers

- **VMs** virtualize the hardware
- Complete isolation
- Complete OS installation. Requires more resources

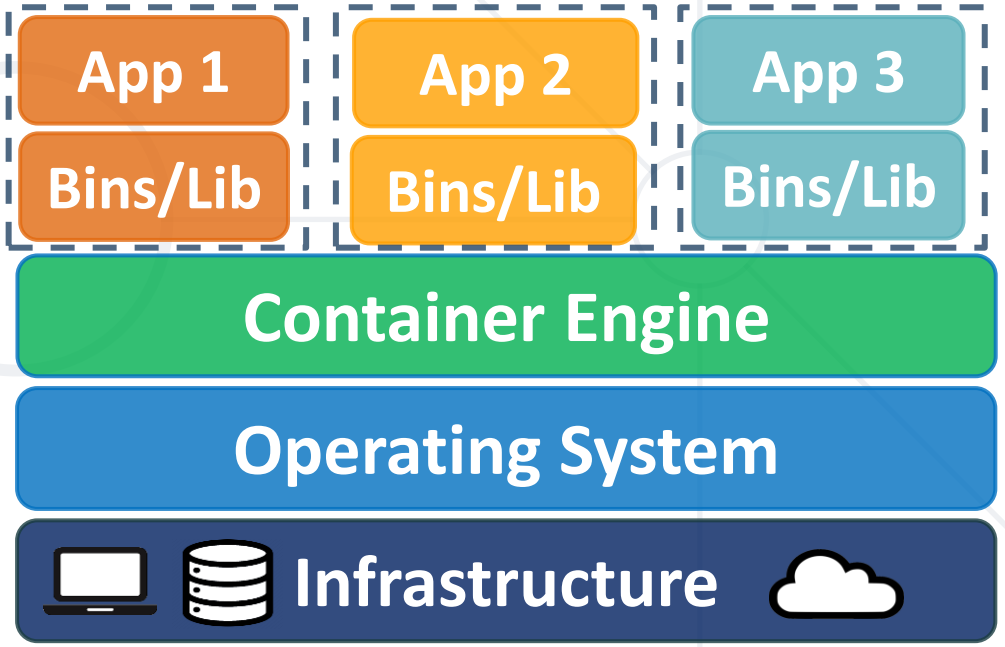


Utilization

Size

Boot Up Time

- **Containers** virtualize the OS
- Lightweight isolation
- Shared kernel. Requires fewer resources



Containerization – Advantages

- Easily **deploy across environments** with little or no modification
- **Immutability**
  - Once a container is created, it **doesn't** change
    - To make a change, a new container must be created
  - Ensures **consistency** across **different environments**
- **Portability**
  - Depend of container runtime, not underlying infrastructure
  - Run on any machine that supports the container runtime

Containerization – Advantages

- A containerized app can be **tested** and **deployed** as a **unit** to the host OS
- **Resource-efficient**
  - Share the same OS kernel and isolate applications from each other
- **Scalability**
  - Can be easily scaled up or down
  - Orchestrated by special tools
    - More on that later

Docker

Docker Images, Containers, Software Development

Docker

- Docker
  - Lightweight, open-source, secure **containerization platform**
- It simplifies **building**, **shipping** and **running applications**
  - On different environments
- Runs natively on Linux or Windows servers
- Runs on Windows or Mac development machines
- Relies on **images** and **containers**

Docker Image

- **Docker image**
  - Blueprint for a container
    - A **read-only template**, used to create containers
    - If you want to change something, you should create a new image
    - Holds app/service/other software
    - **Framework**, **dependencies** and **code** are "described" here
- **Docker registry**
  - A repository for images

Docker Container

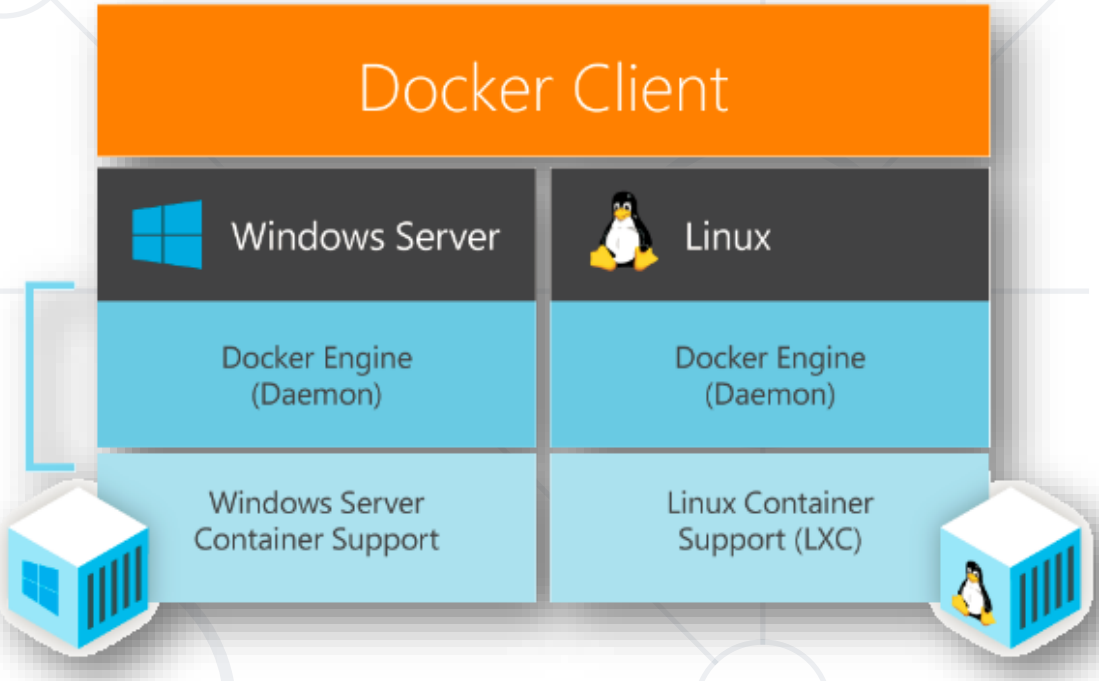
- Built **from the image**
  - Images become containers at **runtime**
- It is the actual **running environment** for your app
- **Isolated** and **secured**
- It can be started / stopped / deleted
- Different app components may reside in separate containers
  - Database, back-end, front-end, caching, messaging, etc.

Docker Desktop

- Out-of-the-box containerization software
- Runs on Windows or Mac development machines
- Includes **Docker Engine**, **CLI** and **Kubernetes**
- Complete Docker **development environment**
- Containerize any application
  - Build
  - Share
  - Run

Docker Desktop

- On Windows
  - Ability to switch between **Linux** and **Windows Server environments**
  - Typically runs Linux containers through **WSL2** technology (Windows Subsystem for Linux)
  - https://docs.docker.com/desktop/install/windows-install



- There are third-party solutions for Linux – DockStation, CairoDock, and more…

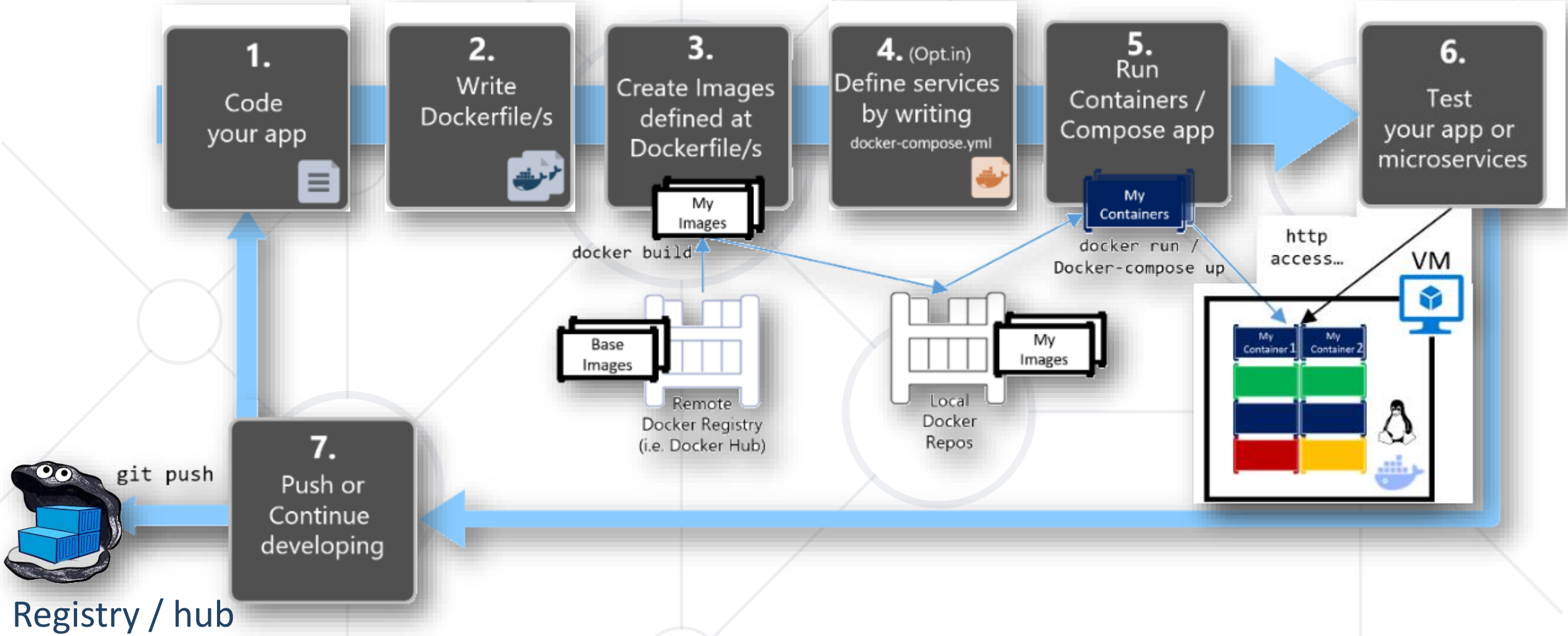
Docker Hub

- Docker Hub == cloud-based **image repository** (registry)
- Used for easy **finding** and **sharing images**
- Supports **public and private repositories**
- Automated builds and webhooks
- For every tool we use in Docker, it is recommended that we **read its documentation** first
  - As sometimes we need to perform configurations to work with the tool

Docker Compose

- Some apps combine multiple components
  - e.g., WordPress requires Linux + NGINX + PHP + MySQL
  - Each component may run in a separate Docker container
- To run **multiple connected containers**, we use **Docker Compose**

Development Workflow for Docker Apps



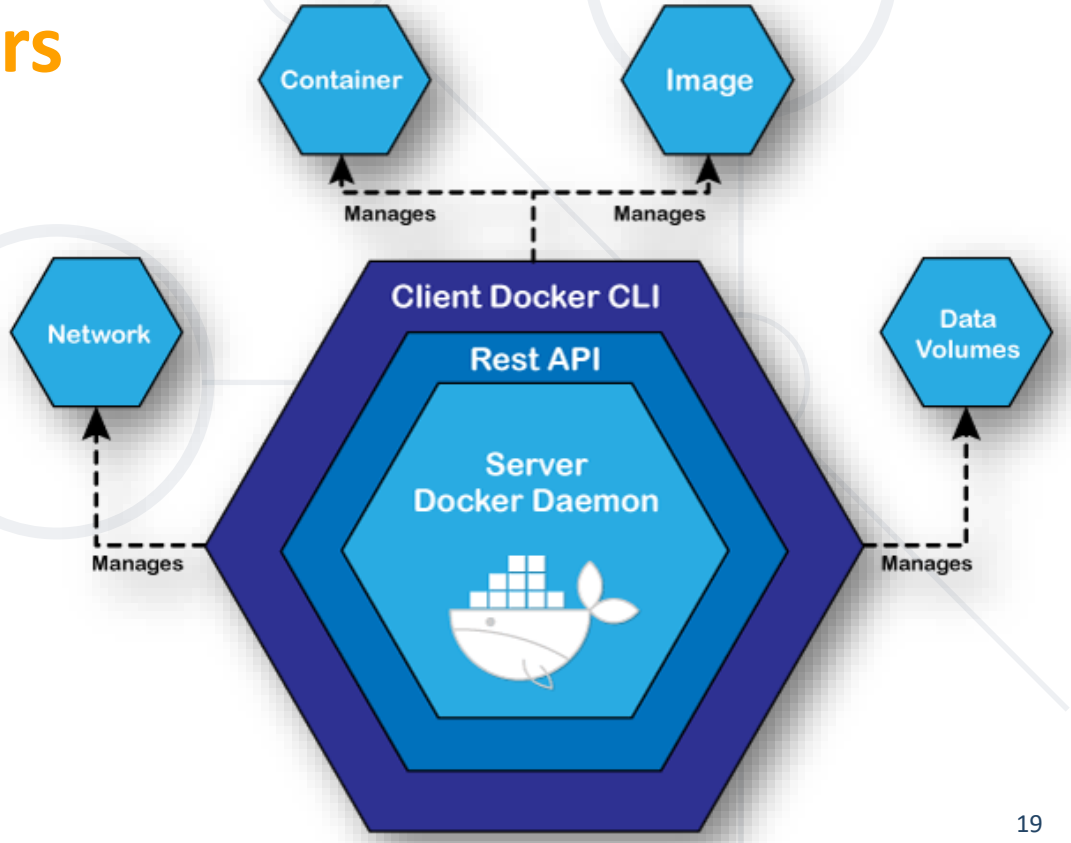
Docker CLI

Command Line Tool to Talk to the Docker Daemon

Docker CLI

- **Docker CLI** allows working with the **Docker Engine**
  - Build and manage **images**
  - Run and manage **containers**
- Example commands

```
docker pull [image]
docker run [image]
docker images
docker ps
docker logs [container]
```



Live Demo

NGINX Server Container

File System and Volume

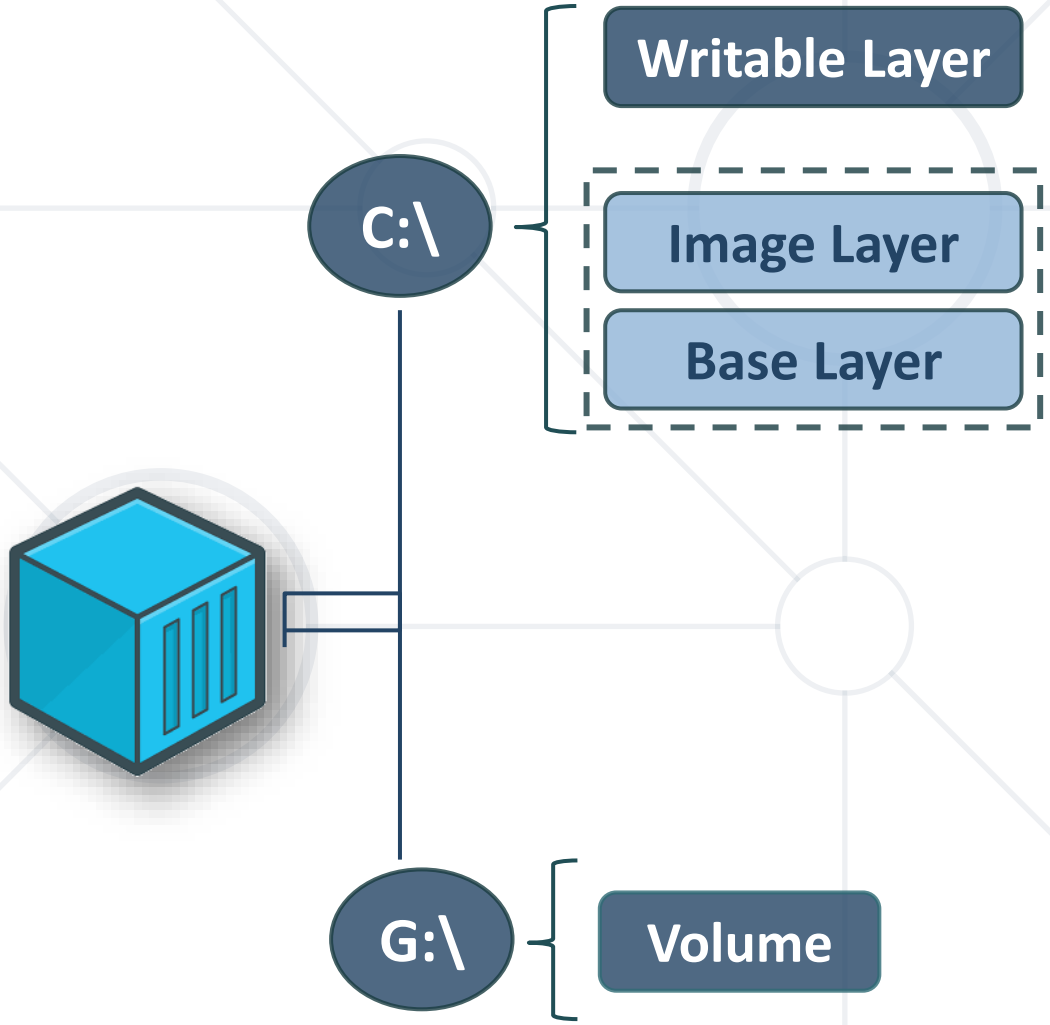
Data in Docker Containers

Layered File System & Container Isolation

- Each **image** has **file system layers**, which are read-only and isolated
  - Image layers are **reused** in different images
- **Images** share **layers**
  - Therefore they load faster once you have them
- Each container is **isolated** and has its **own writable file system**
  - By default, file system is deleted after you delete the container
    - Which is not very suitable for persistence operations

Volumes

- To persist data, use **volumes**
  - Special type of **directory on the host**
  - Mapped to the real file system
  - Can be shared and reused among containers
  - Image updates won't affect volumes
  - Persisted even after the container is deleted
  - You have full control over them



Live Demo

Vue.js App in a Container

Live Demo

Docker Container with MongoDB

Dockerfile

All Commands for Building an Image

Dockerfile

- **Dockerfile** is the way to create custom images
- Contains **build instructions**
- These instructions create an **intermediate image**
  - It can be cached to reduce the time of future builds
- Used with the **`docker build`** command
- It is like compiling a source code

Dockerfile – Example

- We have a sample **Dockerfile** for **Node.js**

```
FROM node:16
ENV NODE_ENV=production
WORKDIR /app
COPY ["package.json", "package-lock.json*", "./"]
RUN npm install --production
COPY . .
CMD [ "node", "server.js" ]
```

- Most **Dockerfiles** may be copy-pasted from the Internet

Multistaging – Example

- Each **stage** deletes the previous one but can reuse it

- In Stage 2 are created
  - **/src** with source code
  - **/app/build**

- In Stage 3
  - Source code is reused
  - **/app/publish** is created

- In Stage 4
  - **/app/publish** is copied from Stage 3
  - At the end, we have only the **.dll file**, without the source code itself

Dockerfile

```
#See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile

FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["MyWebSite.csproj", "."]
RUN dotnet restore "./MyWebSite.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "MyWebSite.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MyWebSite.csproj" -c Release -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MyWebSite.dll"]
```

1st stage

2nd stage

3rd stage

4th stage

Building a Custom Image

All Commands for Building an Image

Process

- Create a **Dockerfile** in the **root** folder of the app
  - Define the base image
  - Set the current working directory
  - Copy files and folders to it
  - Install necessary dependencies
  - Run scripts
- Use **Docker commands** to manage the image
  - **Build** the image
  - **Inspect** the image
  - **Push** a container from the image

Live Demo

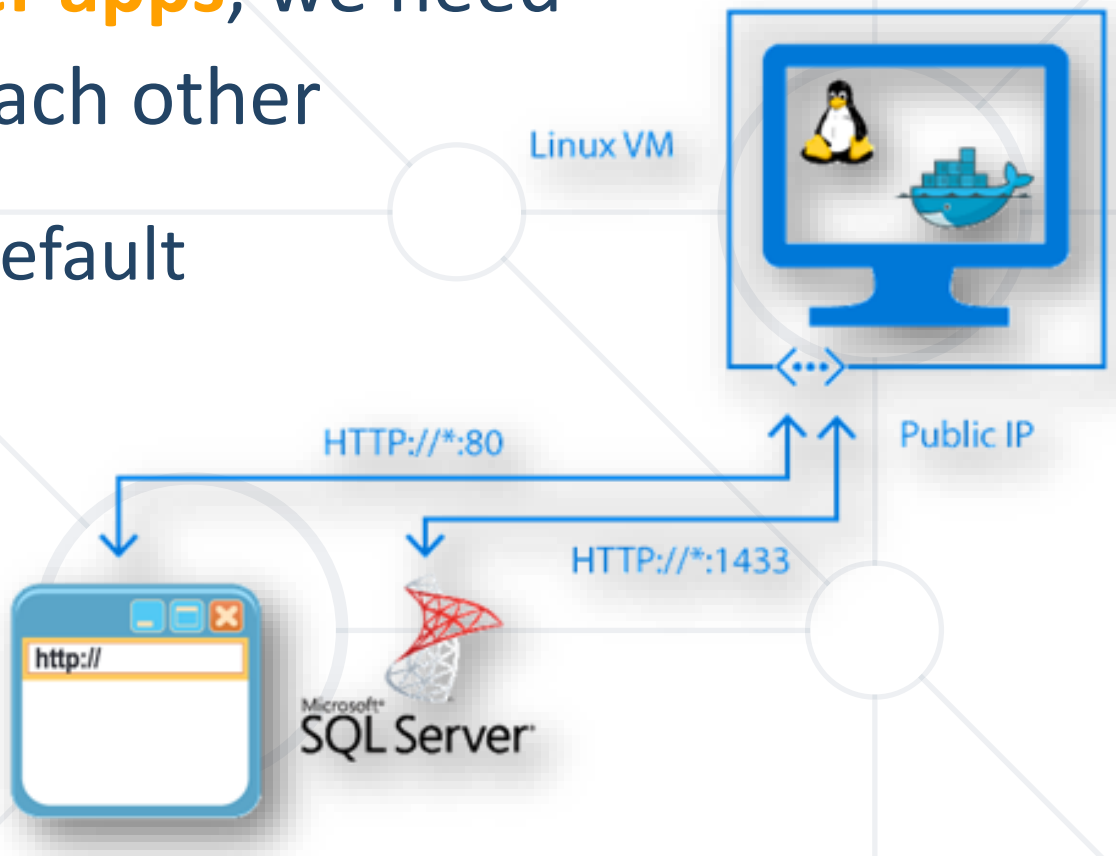
MyWebsite App:
Building a Custom Image

Container Networking

Communication Between Containers

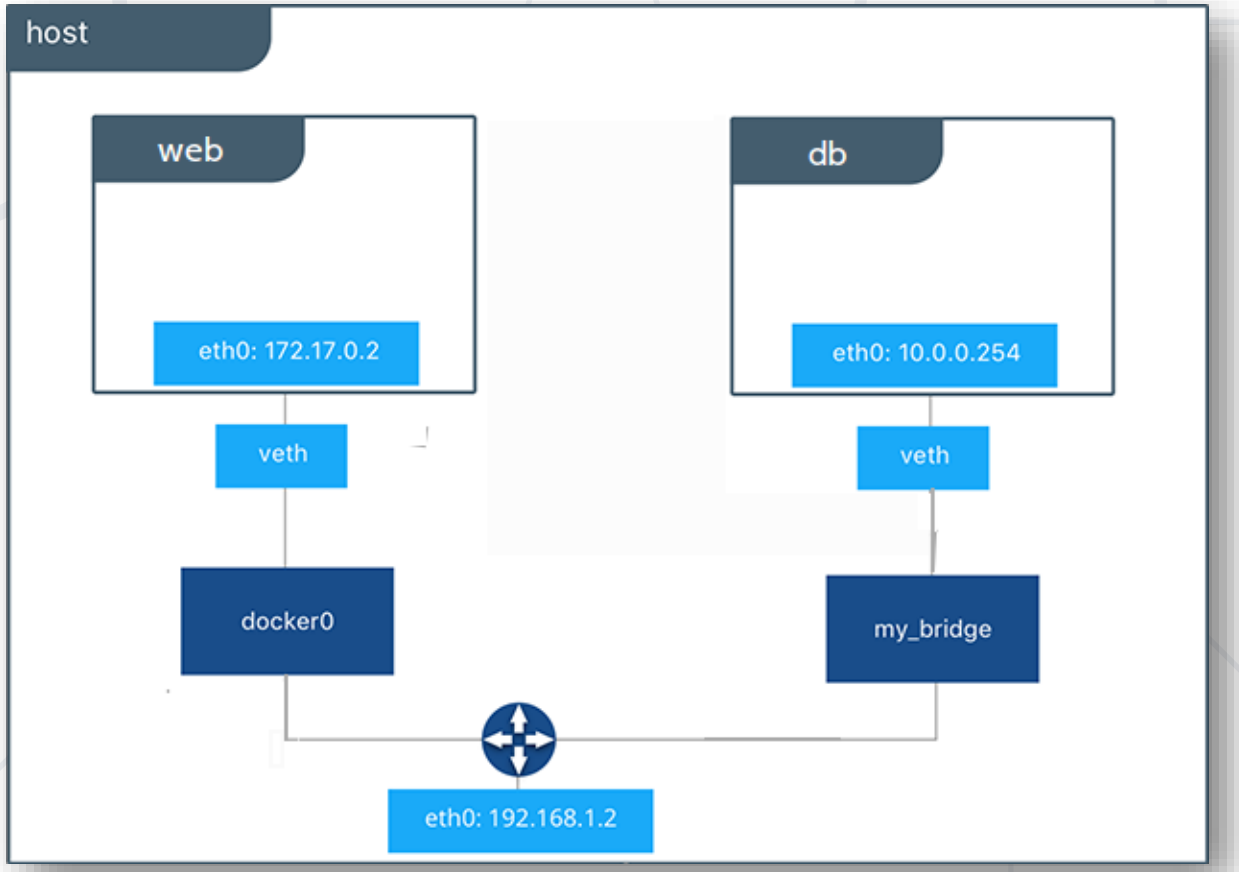
What is Container Networking?

- When working with **multi-container apps**, we need **containers to communicate** with each other
  - But each container is **isolated** by default
  - Here come **networks**
- **Container networking** allows containers to **communicate** with other containers or hosts to **share resources** and **data**



Container Networking Methods

- **Docker Link** — Legacy method, not used, may be deprecated soon
  - Linking one or more docker containers
- **Docker Network**
  - Create a network and connect the containers to that network
- **Docker Compose**
  - Creates an auto-created shared network



Live Demo

WordPress App with MySQL Database: Connecting Containers in a Network

Orchestration Overview

Container Orchestration

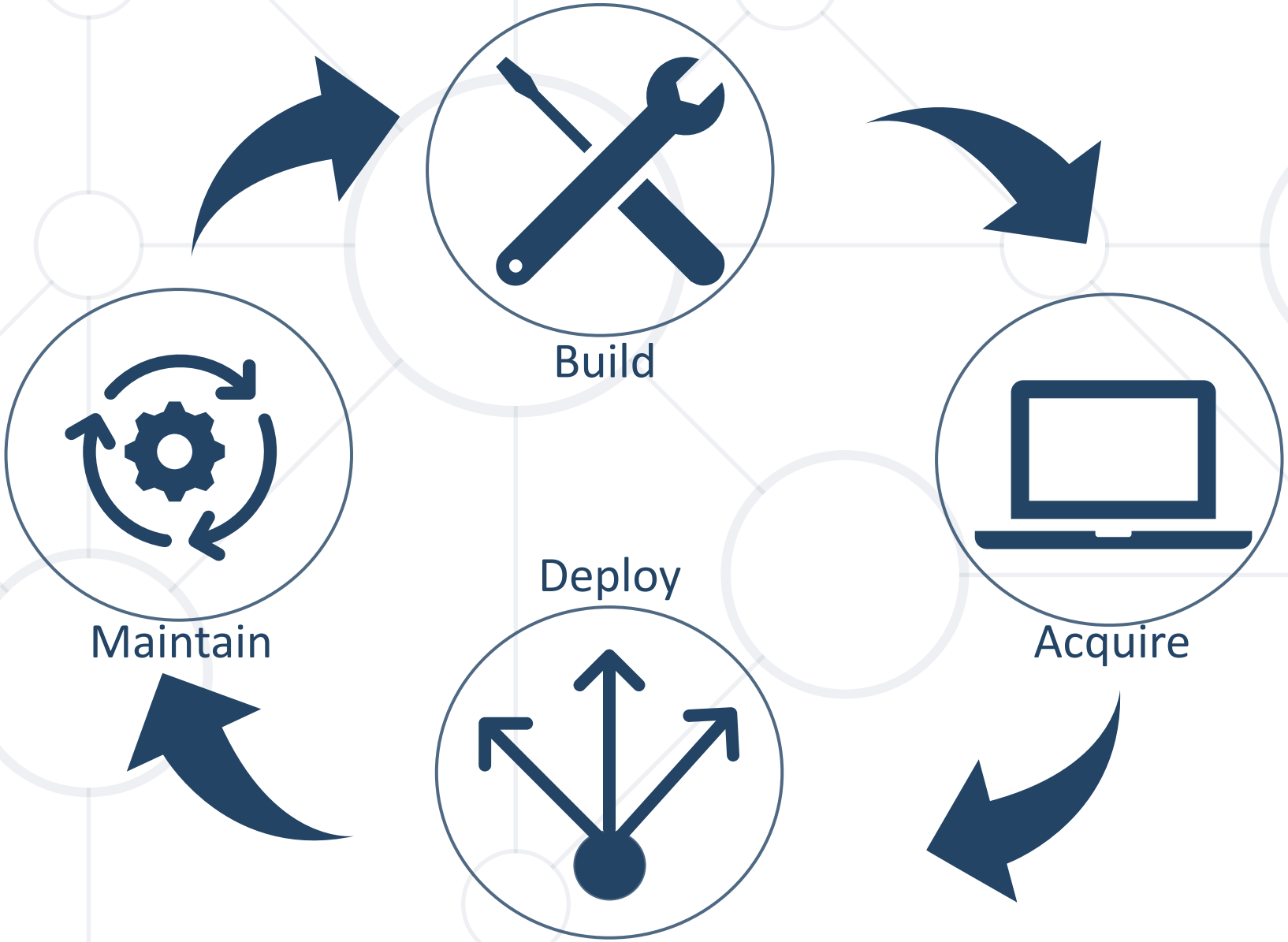
Container Orchestration

- The process of automating **arrangement**, **coordination** and **management** of complex systems, middleware and services
- Benefits
  - **Efficiency**
    - Ensure that work is evenly distributed across infrastructure
  - **Scalability**
    - Handle increased load by adding more instances
  - **Resilience**
    - Ensure high availability by distributing instances
  - **Consistency**
    - Maintain desired state of the system

Real Life Example

- Imagine a **football team**
- **Each player** has its own strengths and role
- The **coach** is responsible for the "***team orchestration***", i.e. **managing the team**
- They should have a good formation, based on the **coach's decisions**
- The coach also **watches them** and makes sure everyone stick to the plan
- The coach also may **replace** injured players when the situation demands it
- The **environment is constantly changing**, and the **coach** reacts to it

Lifecycle

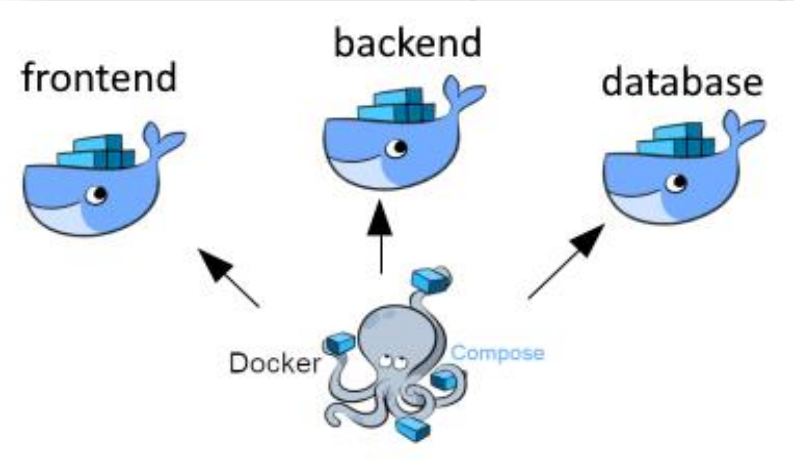


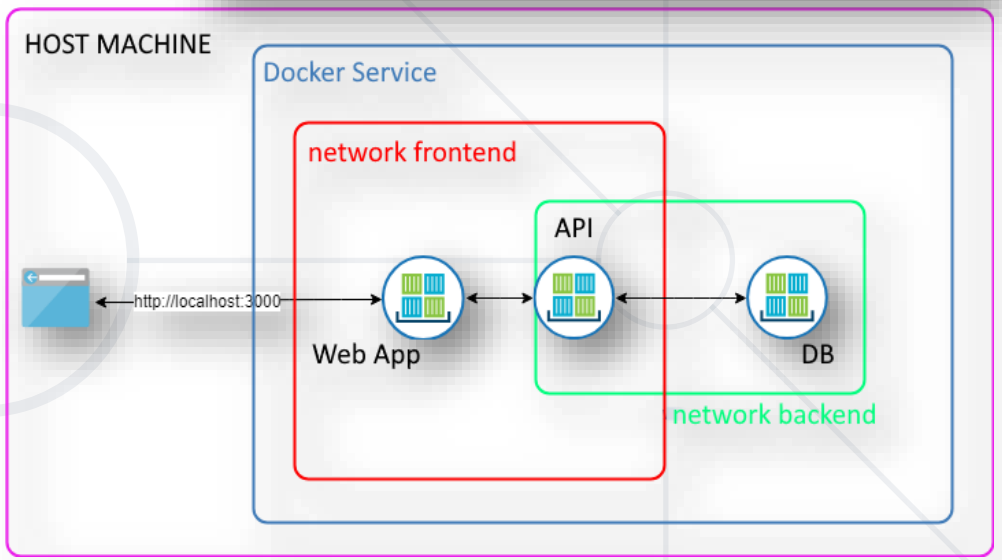
Docker Compose Orchestration Tool

Define and Run Multi-Container Docker Apps

Docker Compose

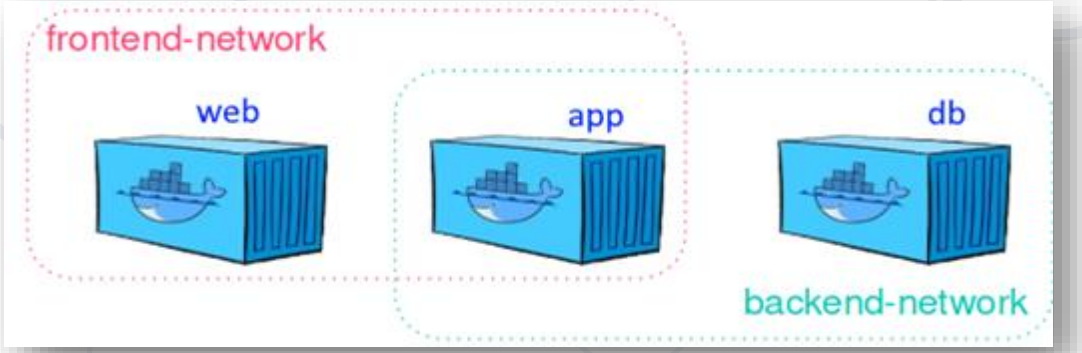
- Manages the **whole application lifecycle**
- Consider a **service** to be a **container** you manage
- **Start**, **stop** and **rebuild** services
- View **status** of running services
- Stream the log output of running services
- Run a **single command** to **run your application**





Docker Compose YAML File

- Define a **docker-compose.yml** file
  - Describes **containers** to be started
- Describe **services** that will be used
- Define the **networking rules**
- Build and start up your **services**
- Manage your **services**



docker-compose.yml

```yaml
version: "3.8"
services:
    db:
        image: mysql:latest
        ...
        networks:
            - backend network
    app:
        build: app
        ...
        networks:
            - backend network
            - frontend network
    web:
        build: web
        ...
        networks:
            - frontend network
networks:
    - backend network
    - frontend network
```

Build a Docker Compose YAML File

- Just add a **docker-compose.yml** file to the **root** folder of your app
- It's like combining separate **docker run** commands

Set a ready to use **image**

Set **environment variables**

Associate **volume** with service

Expose **ports**

Used **volume**

```yaml
version: "1.0"

services:
  wordpress_db:
    image: mysql:latest
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    expose:
      - 3306
      - 33060
  wordpress_site:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
    ports:
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=wordpress_db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpressdb
volumes:
  db_data:
  wp_data:
```

Networking in Docker Compose

- By default, **Compose** sets up a **single network** for your app
  - Each **container** joins the **default network**
  - It is reachable by other containers on that network
  - It is discoverable at a **hostname**, identical to the container name
- You can also **specify custom networks**
- They let you
  - Create more complex topologies
  - Specify custom network drivers and options
  - Connect to externally-created networks

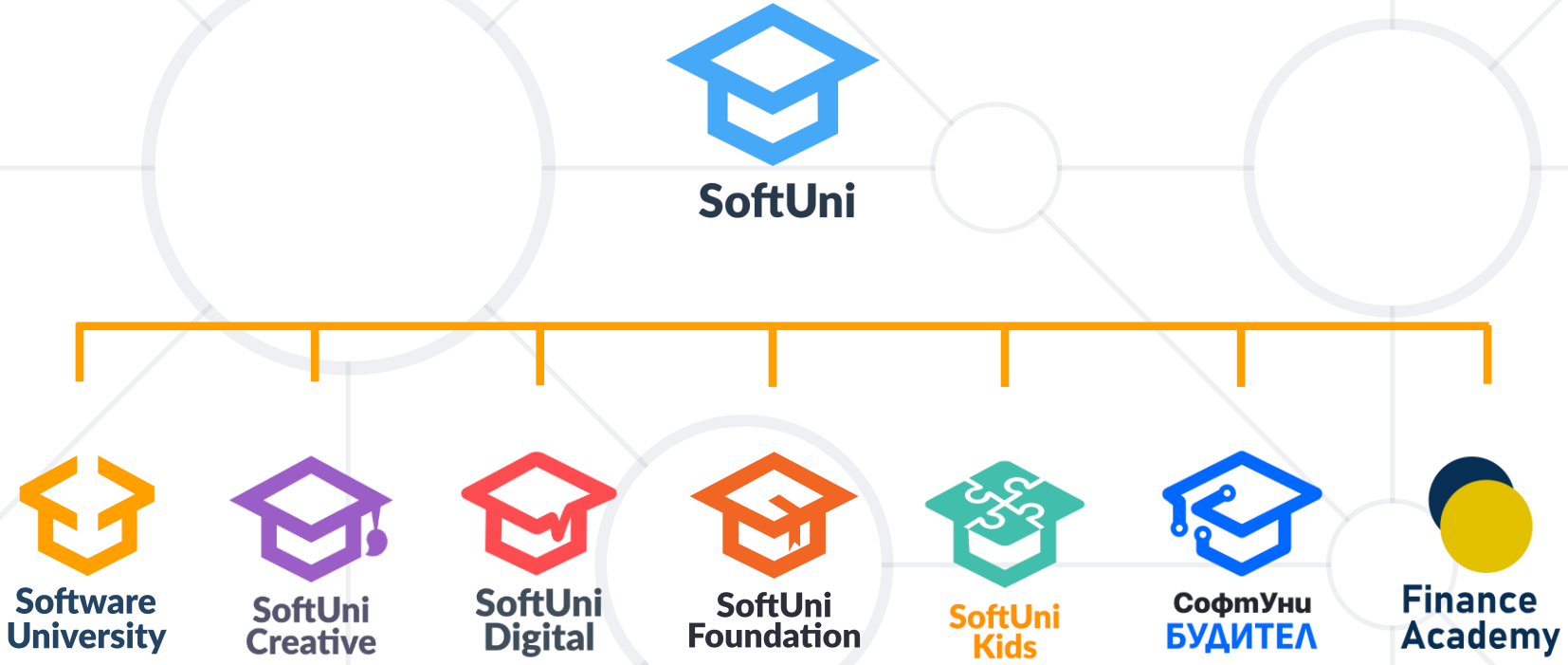
Live Demo

WordPress App with MySQL Database:
Docker Compose YAML File

Summary

- With Docker we can create and manage images, containers, volumes, etc.
  - Image == read-only template with instructions for creating a Docker container
  - Container == a runnable instance of an image
  - Volumes == the preferred mechanism for persisting data
- We can run apps in containers and also have working database in a container
- Dockerfile contains all commands for assembling an image
- Container networking allows communication between containers
- Container orchestration == automation of running and working with containerized workloads and services

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg