Docker Compose

Dockerfile, Container Networking, Orchestration, Using Docker Compose for Multi-Container Apps

Technical Trainers

SoftUni Team

Software University

https://softuni.bg

Have a Question?

sli.do

#Dev-Ops

Table of Contents

1. Dockerfile
2. Container Networking
3. Orchestration Overview
4. Docker Compose Orchestration Tool
5. Kubernetes Overview

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**

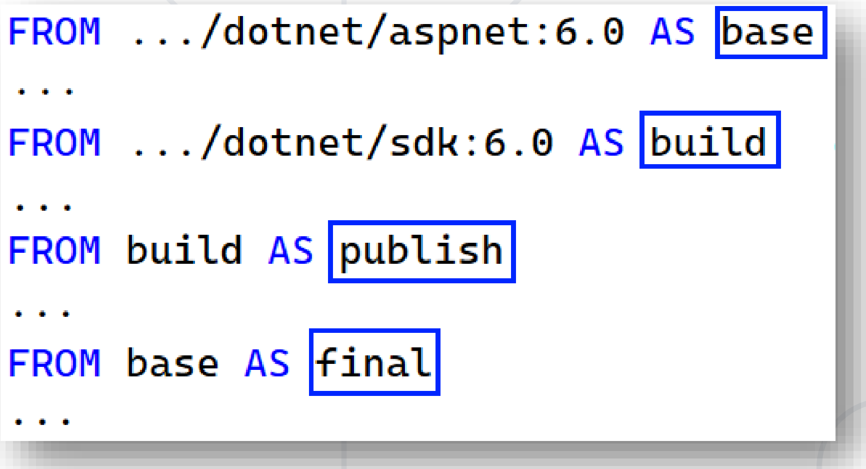
```
FROM node:16
ENV NODE_ENV=production
WORKDIR /app
COPY ["package.json", "package-lock.json*", "./"]
RUN npm install --production
COPY . .
CMD [ "node", "server.js" ]
```

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

Dockerfile: Key Instructions

- **FROM** – create an image from another image (supports multi-staging)
  - Each FROM starts a new stage

```
FROM  <image>
FROM  <image>:<tag>
FROM <image>@<digest>
```



```
FROM .../dotnet/aspnet:6.0 AS base
...
FROM .../dotnet/sdk:6.0 AS build
...
FROM build AS publish
...
FROM base AS final
...
```

Dockerfile: Key Instructions

- LABEL – add metadata in a key-value pair fashion

```
LABEL <key>=<value> <key>=<value> …
```

- RUN – execute command

```
RUN <command> [AS <name>]
RUN ["executable", "param1", "param2"]
```

- COPY – copy different files in the image, like your source code

```
COPY <src> [<src> ...] <dest>
COPY ["<src>", ... "<dest>"]
```

Dockerfile: Key Instructions

- ENTRYPOINT – define which command starts the container

```
ENTRYPOINT executable param1 param2
```

- WORKDIR – the working directory of the image, where your files are

```
WORKDIR </path/to/workdir>
```

- VOLUME – defining a volume for the containe

```
VOLUME ["<path>", ...]
VOLUME <path> [<path> ...]
```

Dockerfile: Key Instructions

- **ENV** – define environment variables
  - Like db connection strings

```
ENV <key> <value>
ENV <key>=<value> [<key>=<value> ...]
```

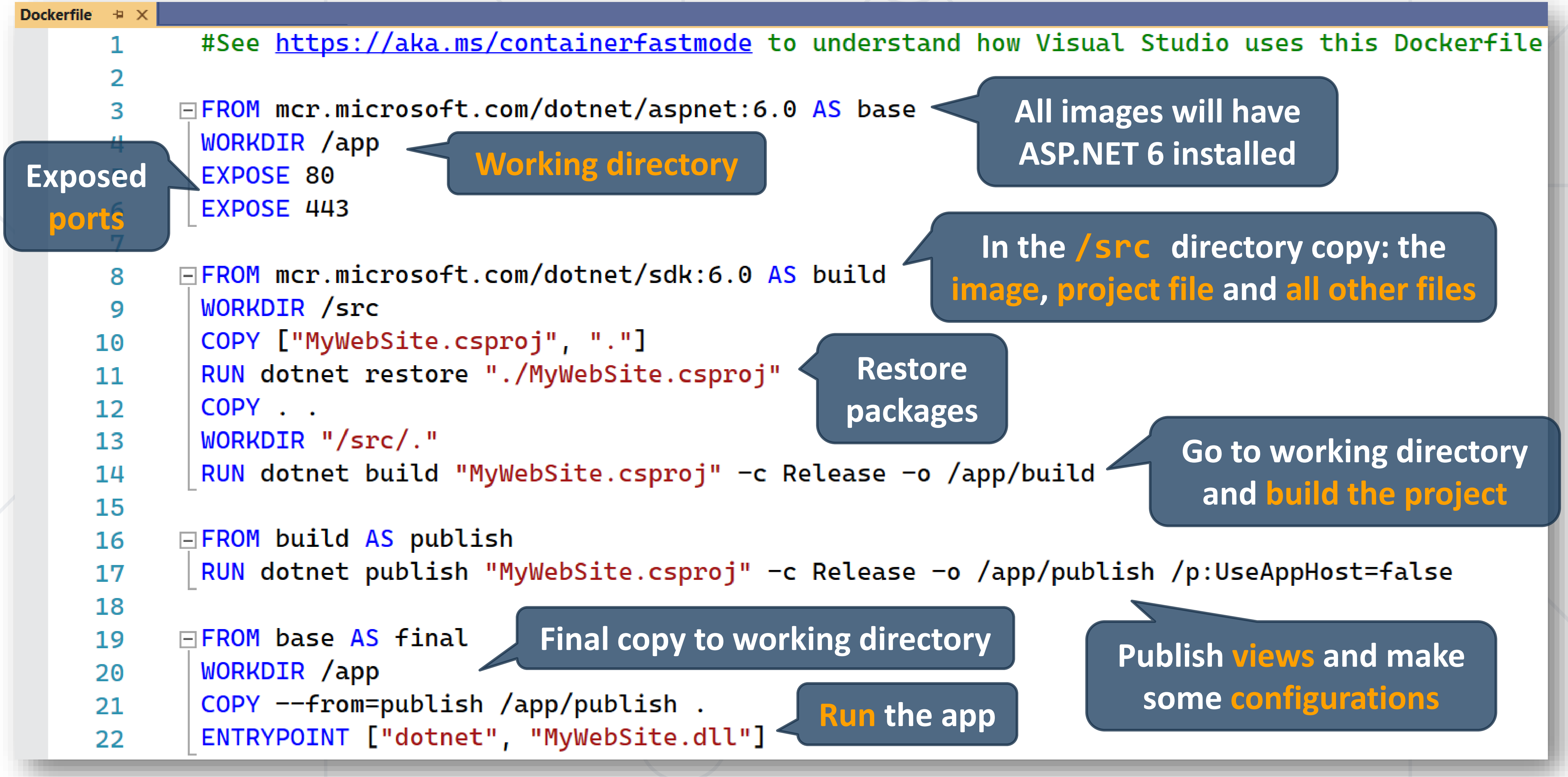
- **CMD** – execute a command-line operation

```
CMD executable param1 param2
```

- **EXPOSE** – expose a port externaly

```
EXPOSE <port> [<port> ...]
```

Dockerfile Structure – Example



```
#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"]
```

All images will have ASP.NET 6 installed

Exposed ports

Working directory

In the /src directory copy: the image, project file and all other files

Restore packages

Go to working directory and build the project

Final copy to working directory

Publish views and make some configurations

Run the app

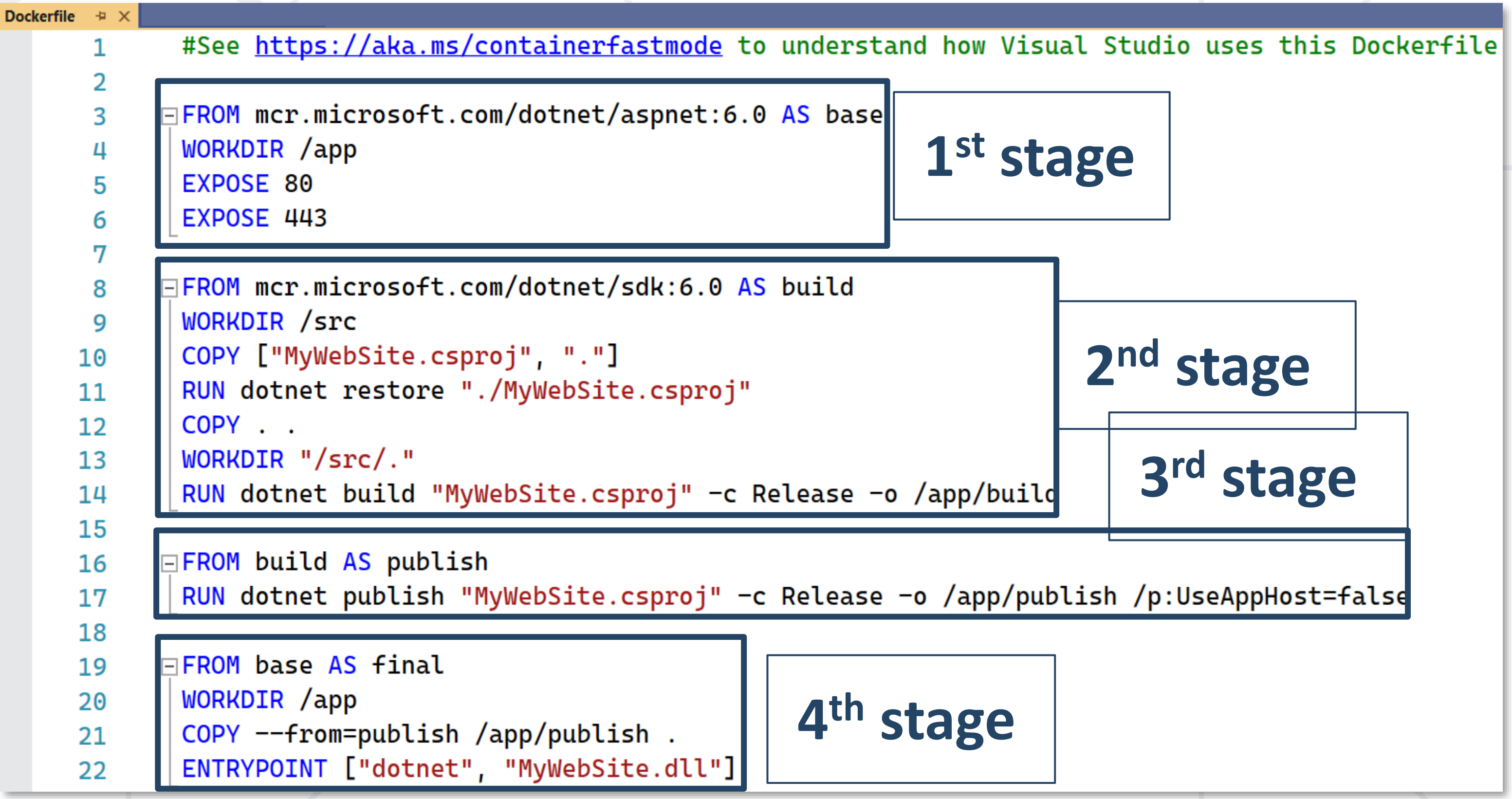
Multistaging – Example

- 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



```
#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/buil

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

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

RUN vs CMD vs ENTRYPOINT

- **RUN** executes command in a **new layer**
  - Used for installing packages, for example
  - Multiple RUN commands are acceptable
- **CMD** sets an **auto-run command** to execute at startup
  - It can be **overridden** from the command line
- **ENTRYPOINT** sets an **auto-run command** to always execute at startup
  - It is **not meant to be overridden** from the command line
- More information is available here
  - https://goinbigdata.com/docker-run-vs-cmd-vs-entrypoint/

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

Commands

- **Build** an image

```
docker image build [OPTIONS] PATH | URL | -
```

- **Inspect** an image

```
docker images
```

- **Run** a container from the image

```
docker run -d image
```

- **Push** to a registry

```
docker push {username}/{app}
```

  - First, login to Docker Hub

```
docker login
```

Live Demo

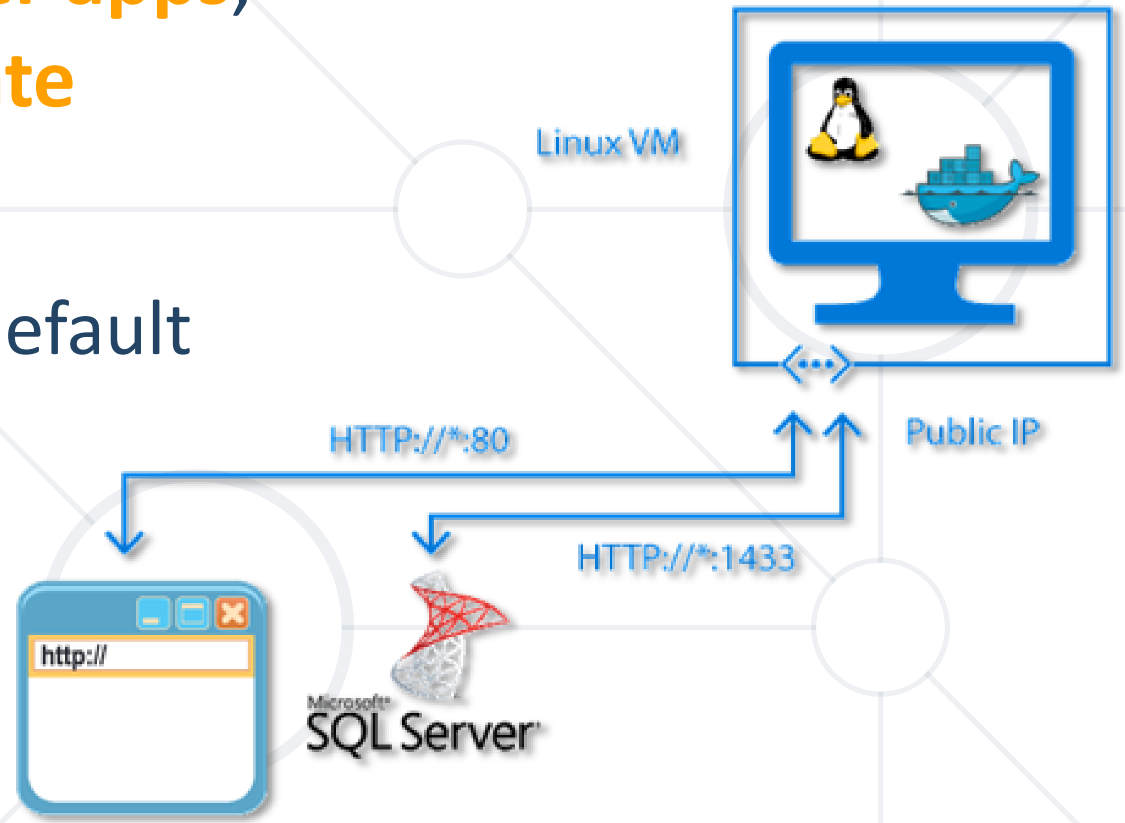
MyWebsite App:
Building a Custom Image

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**

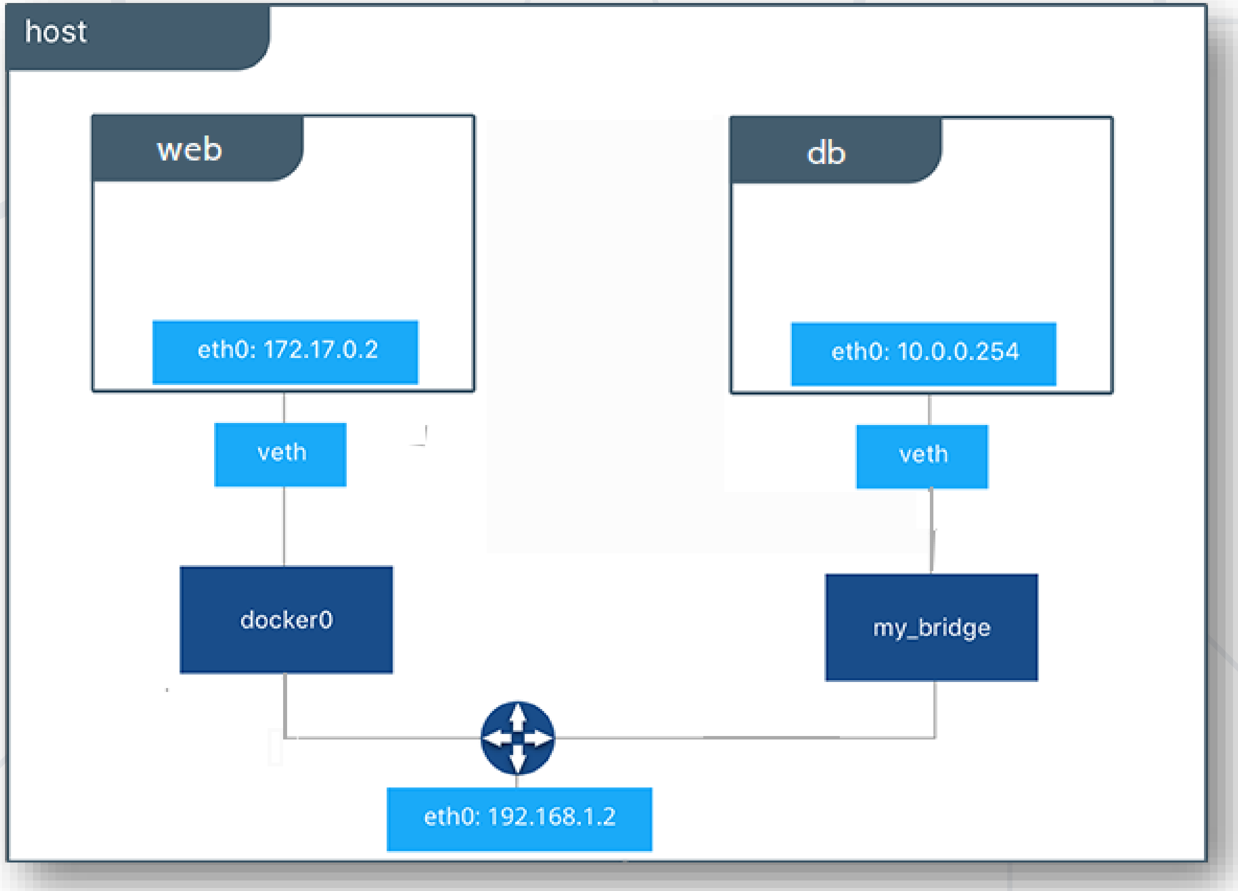


Container Networking Methods

- **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



Docker Networks

- Types of Docker networks
  - **Bridge** (**default**) → containers on a single host
  - **Overlay** → containers on multiple hosts
  - Third-party networks
- When a bridge network is **created**, it is assigned an **IP address range**
- Each container in it will have a **particular IP address** from the network's range

Live Demo

WordPress App with MySQL Database: Connecting Containers in a Network

Orchestration Overview

Container Orchestration

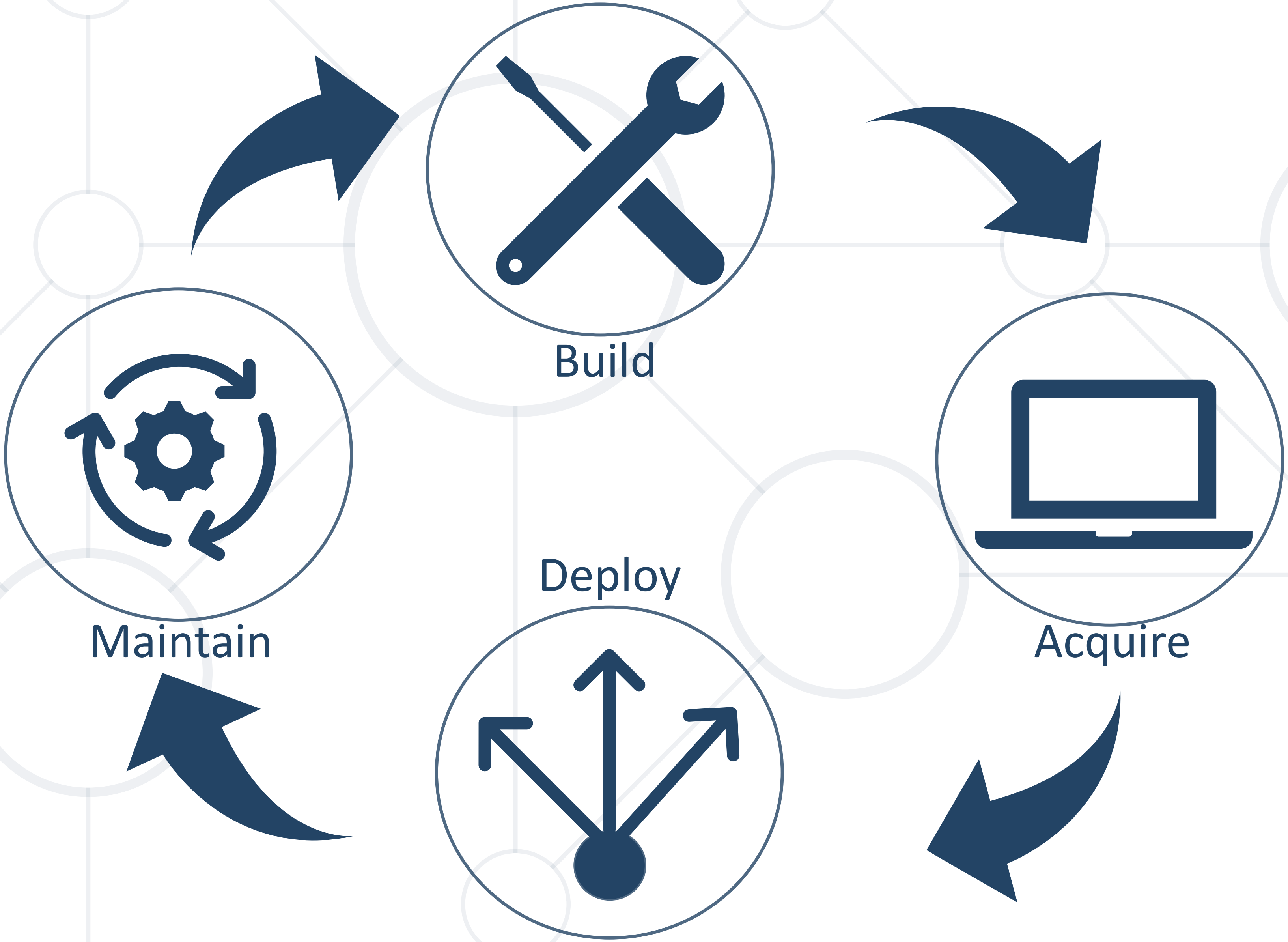
Container Orchestration

- 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**
- He also **watches them** and makes sure everyone stick to the plan
- He also may **replace** injured players when the situation demands it
- The **environment is constantly changing**, and the **coach** reacts to it

Lifecycle



Orchestration Tools

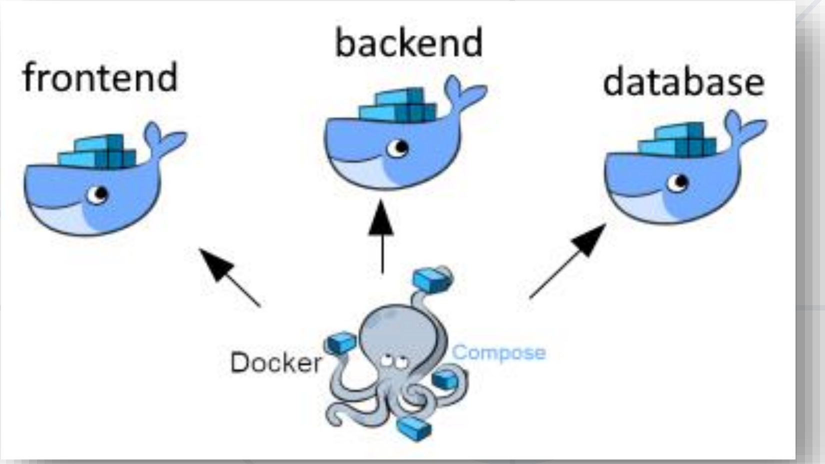
- Docker Swarm
  - Advanced feature for managing a cluster of Docker daemons
- Kubernetes (K8s)
  - Most used open-source system for container orchestration
- Mesos
  - Build and run a distributed system
- Nomad
  - Deploy and manage containers and non-containerized applications
- Rancher
  - Provision and manage Kubernetes clusters

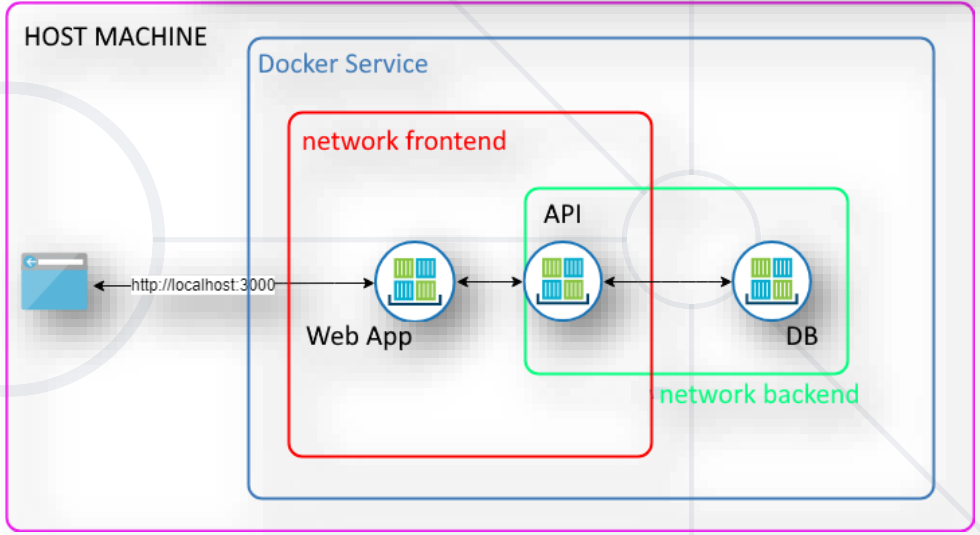
Docker Compose Orchestration Tool

Define and Run Multi-Container Docker Apps

Docker Compose

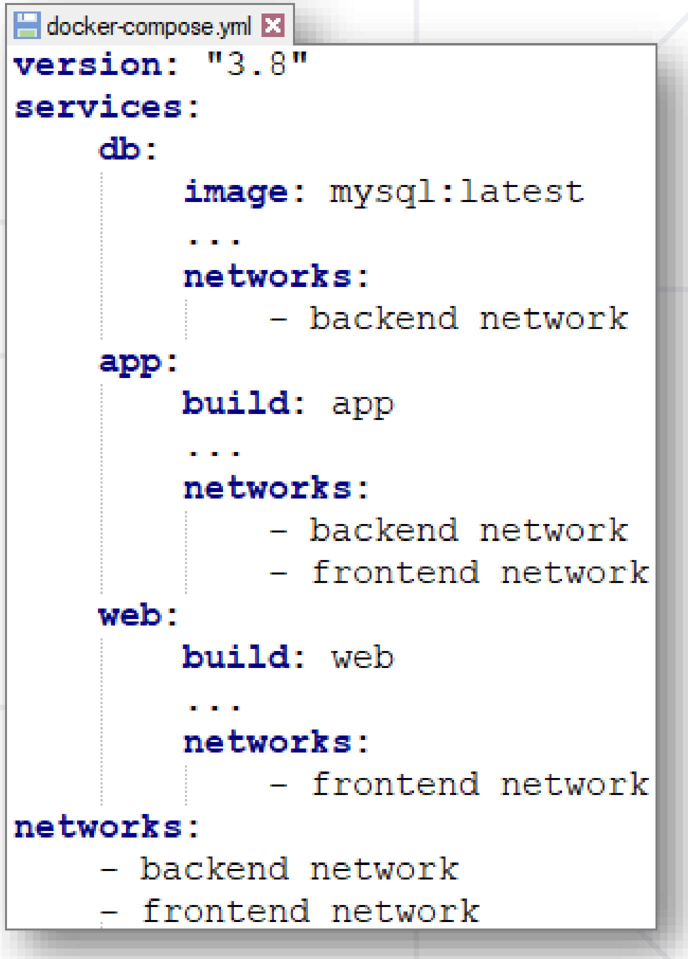
- Manages the **whole application lifecycle**
- Consider a **service** to be a **container** you manage
- **Start**, **stop** and **rebuild** services
- View **status** of running services
- Stream the log output of running services
- Run a **single command** to **run your application**





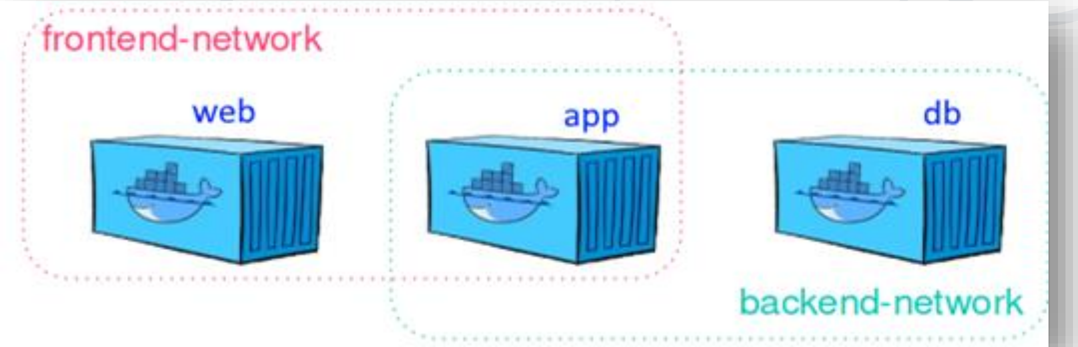
Docker Compose YAML File

- 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

```
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

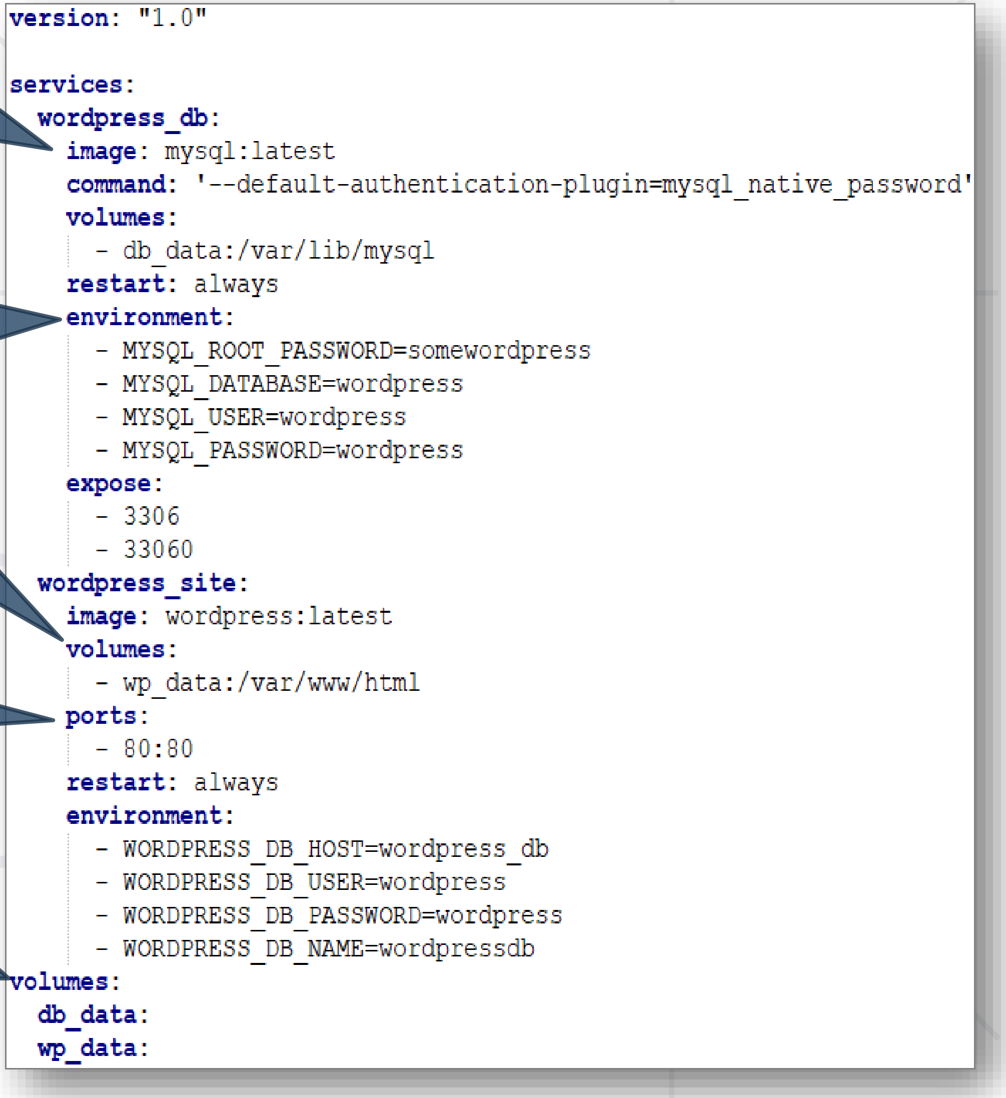
Set a ready to use **image**

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:
```

Build and Run a Multi-Container App

- Build **all images**

```
docker-compose build
```

- Run the **containers**

```
docker-compose up
```

- Or in "silent" mode

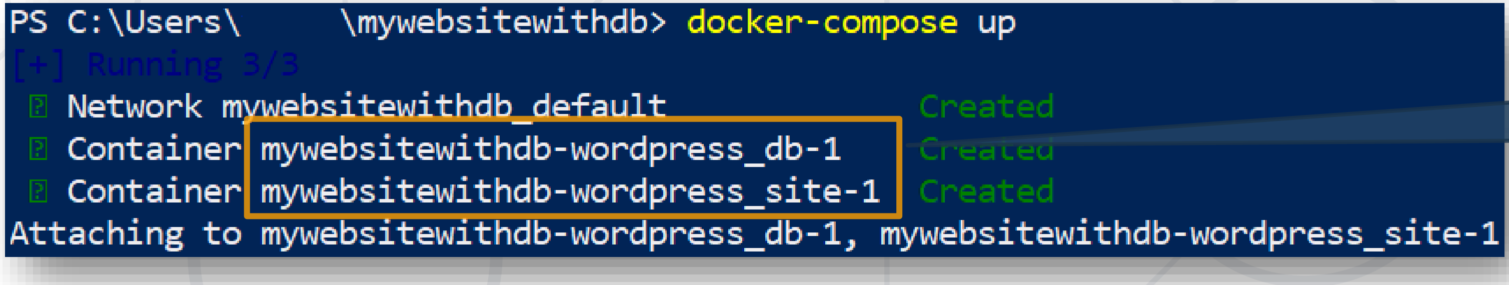
```
docker-compose up -d
```

- Check if services are up and running

```
docker-compose ps
```

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

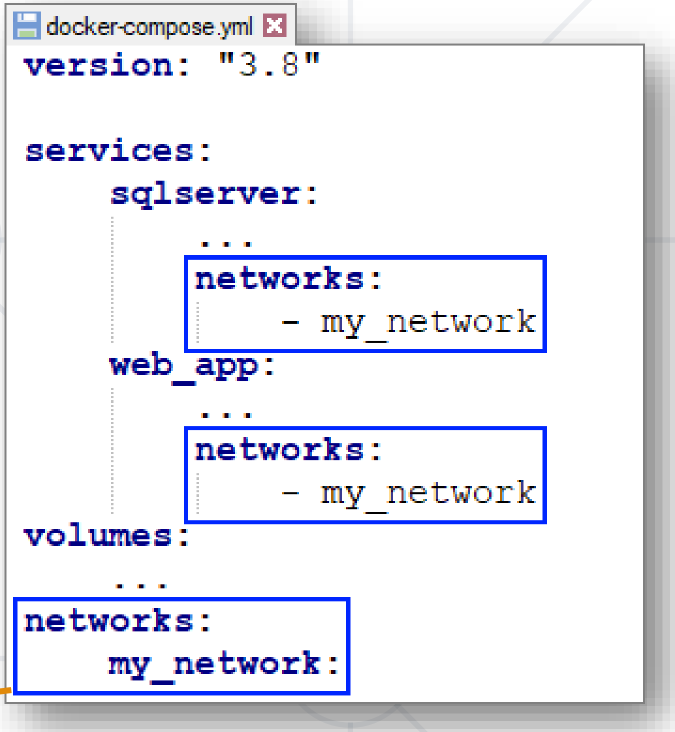


```
PS C:\Users\     \mywebsitewithdb> docker-compose up
[+] Running 3/3
 ⍰ Network mywebsitewithdb_default            Created
 ⍰ Container mywebsitewithdb-wordpress_db-1   Created
 ⍰ Container mywebsitewithdb-wordpress_site-1 Created
Attaching to mywebsitewithdb-wordpress_db-1, mywebsitewithdb-wordpress_site-1
```

Notice **container hostnames**

Networking in Docker Compose

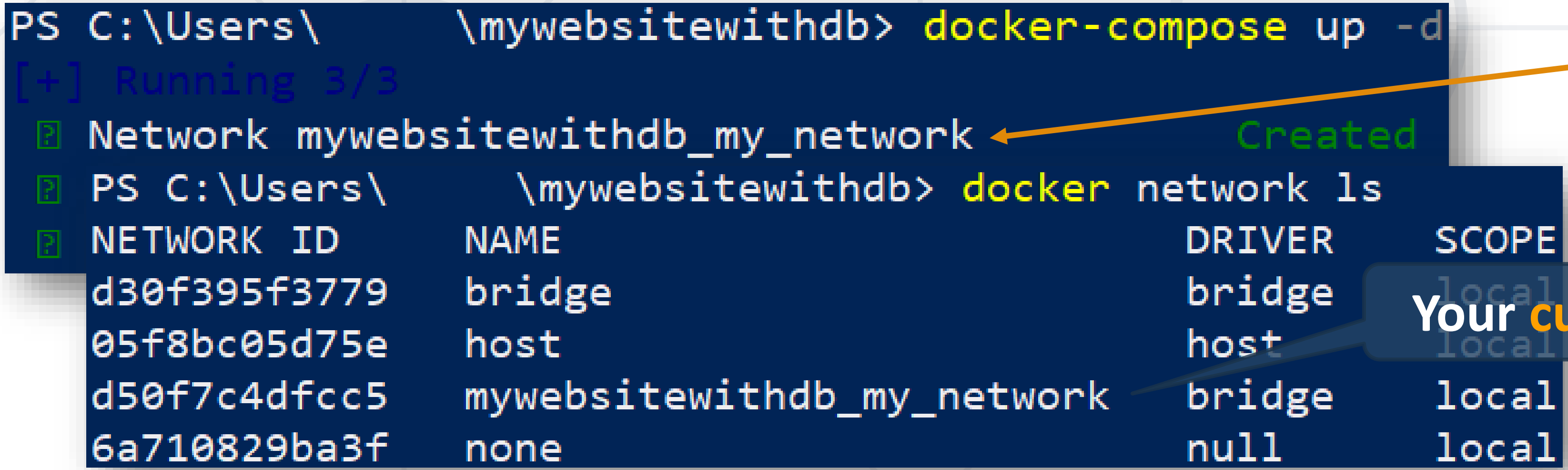
- You can also **specify custom networks**
- They let you
  - Create more complex topologies
  - Specify custom network drivers and options
  - Connect to externally-created networks



docker-compose.yml

```yaml
version: "3.8"

services:
    sqlserver:
        ...
        networks:
            - my_network
    web_app:
        ...
        networks:
            - my_network
volumes:
    ...
networks:
    my_network:
```



```
PS C:\Users\      \mywebsitewithdb> docker-compose up -d
[+] Running 3/3
 ✔ Network mywebsitewithdb_my_network                Created
 ✔ PS C:\Users\      \mywebsitewithdb> docker network ls
 ✔ NETWORK ID     NAME                          DRIVER    SCOPE
   d30f395f3779   bridge                        bridge    local
   05f8bc05d75e   host                          host      local
   d50f7c4dfcc5   mywebsitewithdb_my_network    bridge    local
   6a710829ba3f   none                          null      local
```

Your custom network

More Docker Compose Commands

- Compose with **multiple files**

```
docker-compose -f docker-compose.yml -f production.yml up -d
```

- **Redeploy** a single service

```
docker-compose build web
docker-compose up --no-deps -d web
```

- **Remove everything** (images, volumes, etc.)

```
docker-compose down --rmi all --volumes
```

Live Demo

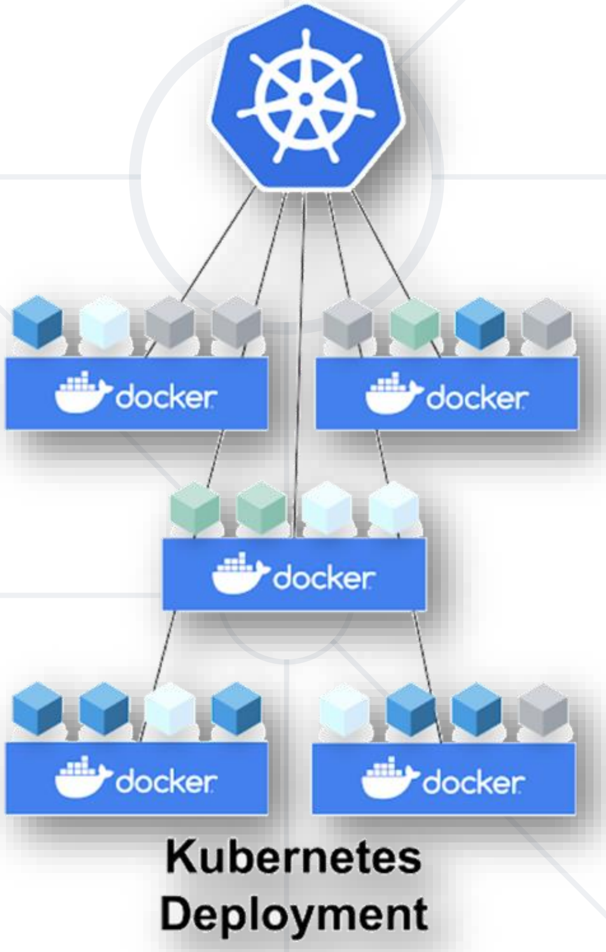
WordPress App with MySQL Database: Docker Compose YAML File

Kubernetes Overview

Open-source Container Orchestration Tool by Google

What is Kubernetes?

- Kubernetes == container orchestration system
  - Automates deployment, scaling, and management of containerized apps
  - Solving challenges from having distributed apps
  - Open-source
- Kubernetes & Docker work together to build & run containerized applications



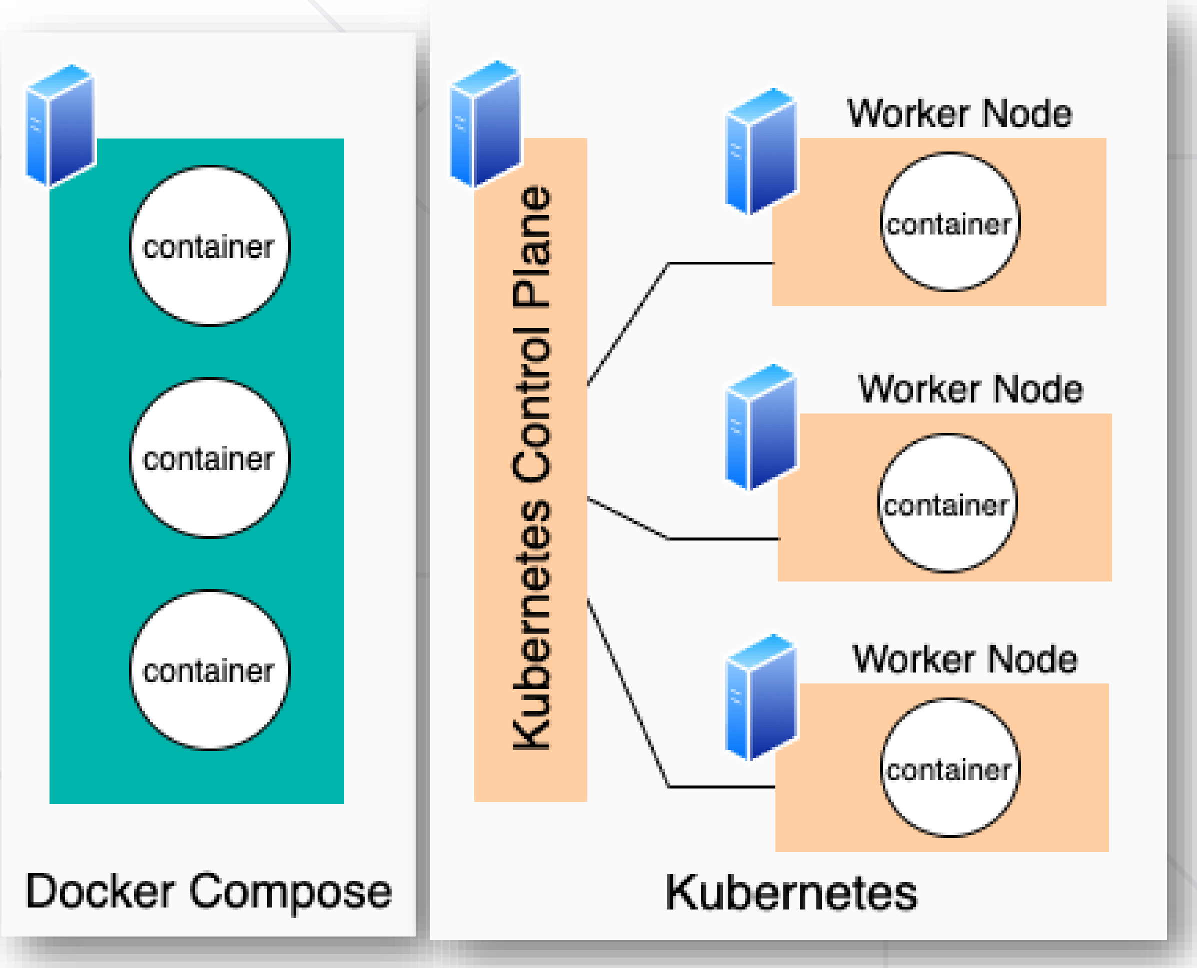
Kubernetes Deployment

Overview

- **Clusters**
  - Where containers are being run
- **Nodes**
  - Collections of clusters
  - Virtual machines or physical computers
  - The "**master**" node manages each cluster
- **Pods**
  - Smallest deployable unit
  - Can host one or more containers

Kubernetes vs Docker Compose

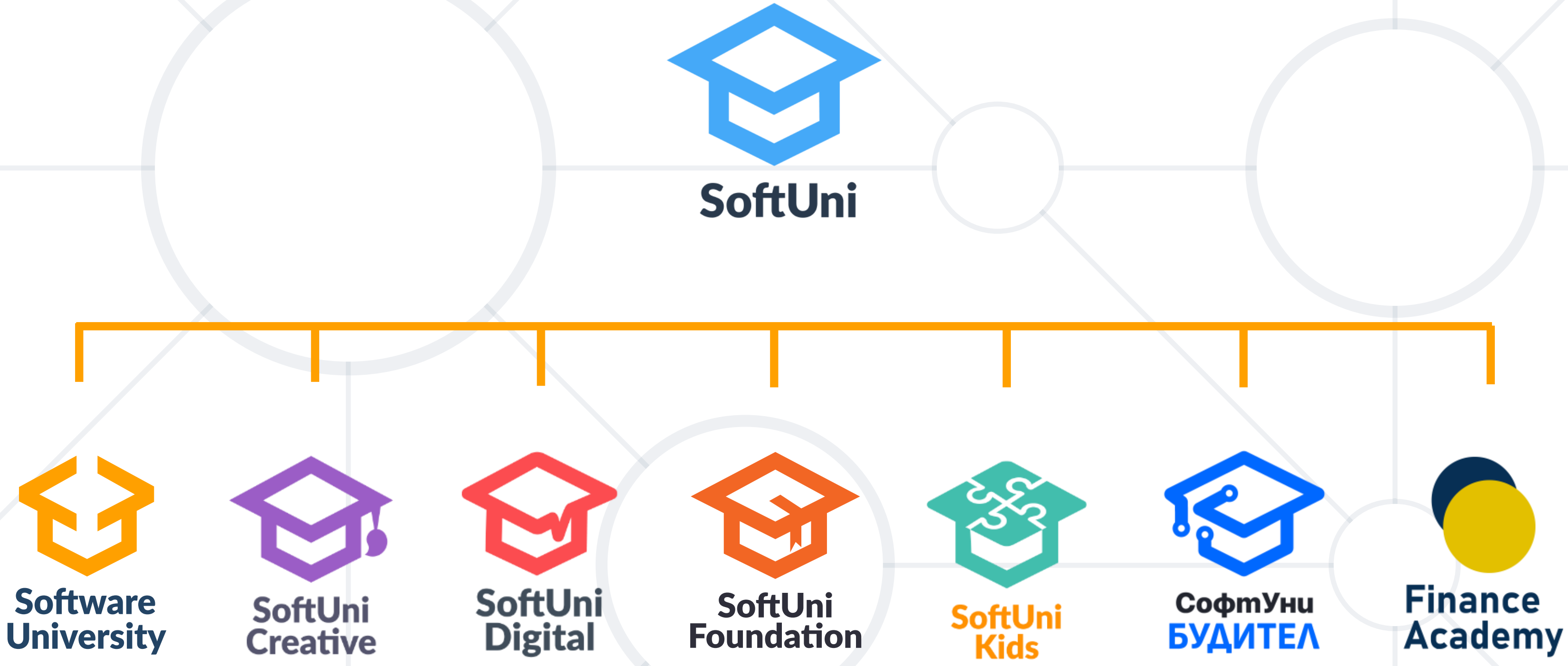
- Both are **frameworks for container orchestration**
- Main difference
  - **Docker Compose** runs containers on a **single host machine**
  - **Kubernetes** runs containers across **multiple computers**



Summary

- **Dockerfile** contains all commands for assembling an image
- We can pull and push images to **Docker Hub**
- **Container networking** allows communication between containers
  - Used for running **multi-container apps** in Docker
- **Container orchestration** == automation of running and working with containerized workloads and services
  - **Docker Compose** == Docker tool for running multi-container apps
  - **Kubernetes** == open-source orchestration system

Questions?



SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

SmartIT

createX

VIVACOM

INDEAVR
Serving the high achievers

Coca-Cola HBC
Bulgaria

Postbank
Решения за твоето утре

AMBITIONED

SOFTWARE
GROUP

SUPER
HOSTING
.BG

DRAFT
KINGS

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