Docker Compose

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

Technical Trainers

SoftUni Team

Software University

SoftUni

https://softuni.bg

Software University

Have a Question?

Software University

sli.do

#Dev-Ops

Table of Contents

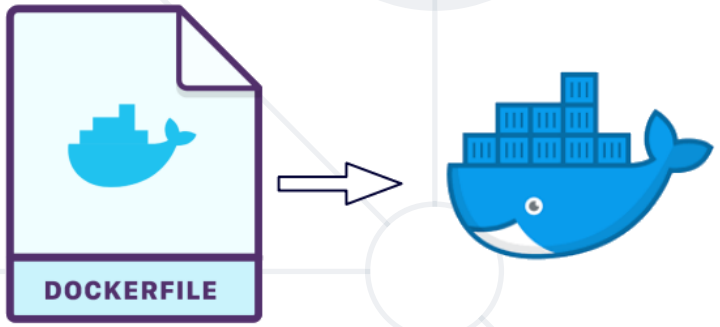
Dockerfile

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

```
FROM ubuntu
ENV APP nginx
RUN apt-get update && apt-get install -y APP
WORKDIR /var/www/html/
ADD index.html ./
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```



Docker file

Docker Image

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
- **LABEL** – add metadata in a key-value pair fashion
- **RUN** – execute command
  - For example, `npm install`
- **COPY** – copy different files in the image, like your source code

```
FROM <image>
FROM
FROM
```

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

```
LABEL <key>=<value> <key>=<value> ...
```

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

```
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>
```

- **EXPOSE** – expose a port externally

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

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

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

- **VOLUME** – defining a volume for the container

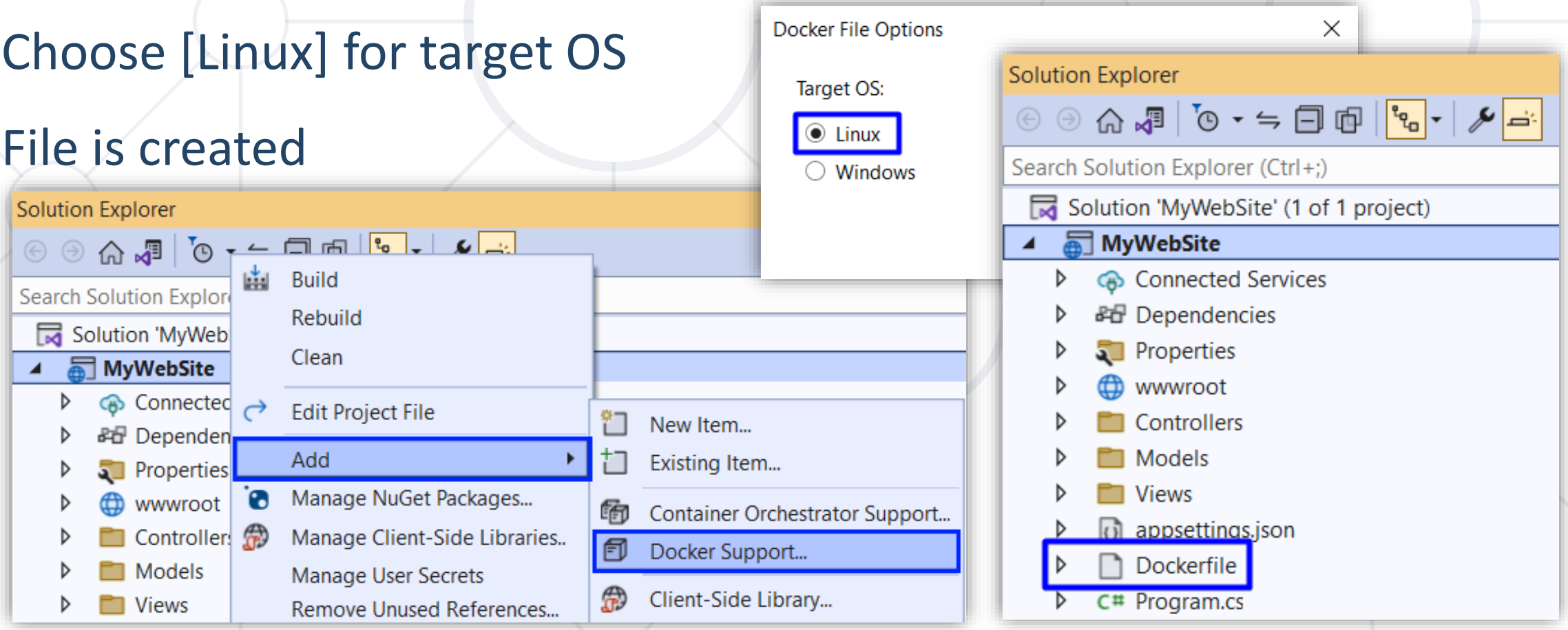
```
VOLUME ["<path>", ...]
VOLUME <path> [<path> ...]
```

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

```
CMD executable
param1 param2
```

Dockerfile from VS

- You can easily **create a Dockerfile** from **Visual Studio**
  - Right-click on the project → [Add] → [Docker Support]
  - Choose [Linux] for target OS
  - File is created



Dockerfile Structure – Example

Software University

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"]
```

All images will have ASP.NET 6 installed

Working directory

Exposed ports

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

Restore packages

Go to working directory and build the project

Publish views and make some configurations

Final copy to working directory

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

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

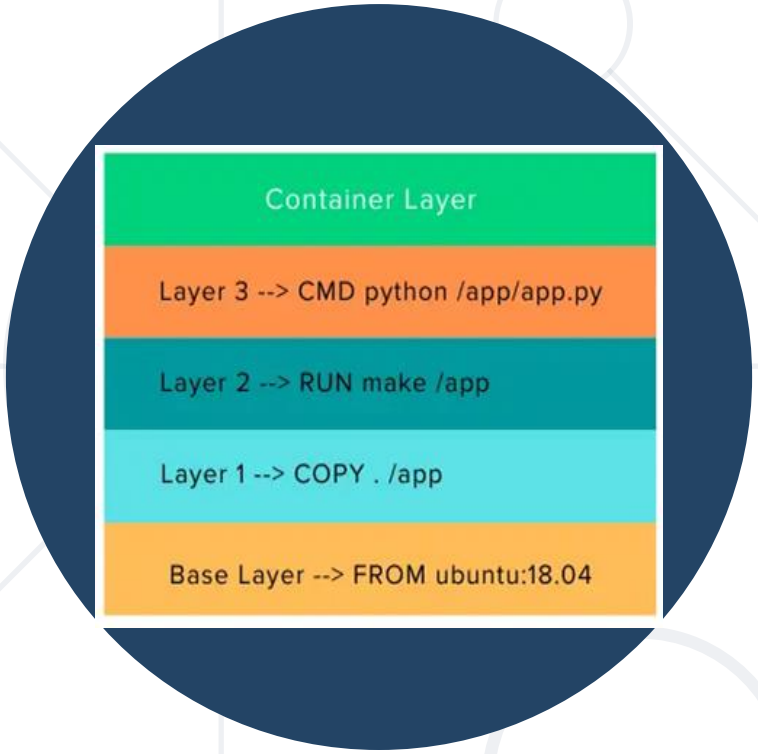
2nd stage

3rd stage

4th stage

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

Publish it in Docker Hub and Run It as a Container

Build a Custom Image (1)

- Build the **custom Vue.js app image**
  - Use the app from the previous session that we ran in a Docker container
- 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

```
FROM node:16

WORKDIR /app

COPY . .

RUN npm install

CMD ["npm", "run", "dev"]
```

Build a Custom Image (2)

- **Build** an **image** from a **Dockerfile**

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

```
PS C:\Users\     \MyWebsite> docker image build -t my-webapp .
[+] Building 57.7s (9/9) FINISHED
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [internal] load build definition from Dockerfile                       0.0s
 => => transferring dockerfile: 117B                                       0.0s
 => [internal] load metadata for docker.io/library/node:16                 0.0s
 => [1/4] FROM docker.io/library/node:16                                   0.1s
 => [internal] load build context                                         32.8s
 => => transferring context: 99.27MB                                      31.9s
 => [2/4] WORKDIR /app                                                     0.0s
 => [3/4] COPY . .                                                        15.4s
 => [4/4] RUN npm install                                                  8.2s
 => exporting to image                                                     1.3s
 => => exporting layers                                                    1.2s
 => => writing image sha256:975d2fe533817deb55065688f27abc59d3f2ef925eb38e27fbaa4168e  0.0s
 => => naming to docker.io/library/my-webapp                               0.0s
```

You can check the **steps** for **errors**

Publish a Custom Image in Docker Hub

- **Log in** to **Docker Hub**

```
docker login
```

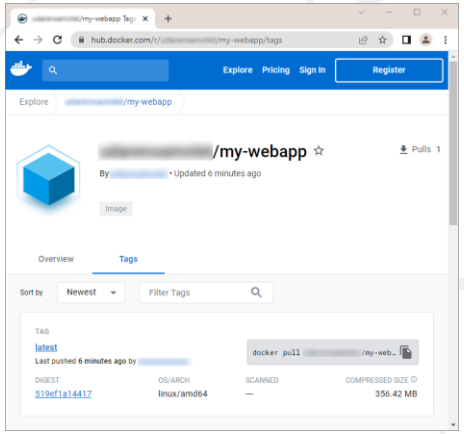
```
PS C:\Users\     \MyWebsite> docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at ht
tps://docs.docker.com/go/access-tokens/
```

- **Publish** the image

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

```
PS C:\Users\     \MyWebsite> docker tag my-webapp [redacted]/my-webapp
PS C:\Users\     \MyWebsite> docker push [redacted]/my-webapp
Using default tag: latest
The push refers to repository [docker.io/[redacted]/my-webapp]
f018d0224715: Pushed
60924be50ac8: Pushed
4179ffb128d8: Pushed
c4d7495580fd: Mounted from library/node
4aa2274f91a0: Mounted from library/node
75b410f76042: Mounted from library/node
5c7da2ce555d: Mounted from library/node
c73ad13a1488: Mounted from library/node
f584c095e67e: Mounted from library/node
ee4d330edba0: Mounted from library/node
f689d32da261: Mounted from library/node
latest: digest: sha256:519ef1a14417d7caf25b814e7af48cac1e0ab1be98205d2a0c2
d7808efd18957 size: 2633
```

- Our **image** is in **Docker Hub**



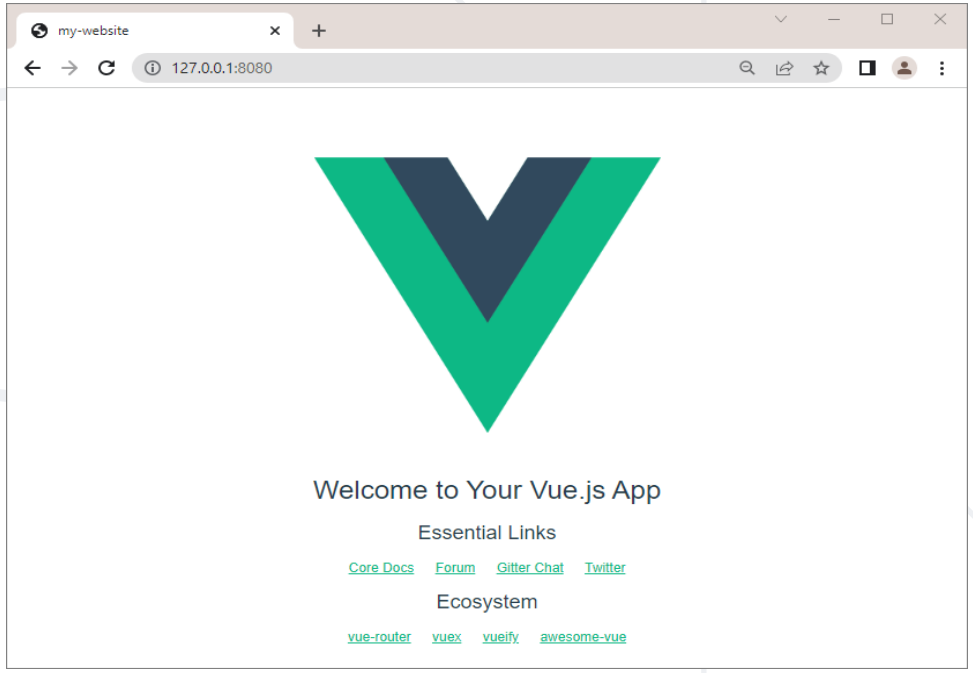
Run a Custom Image as a Container

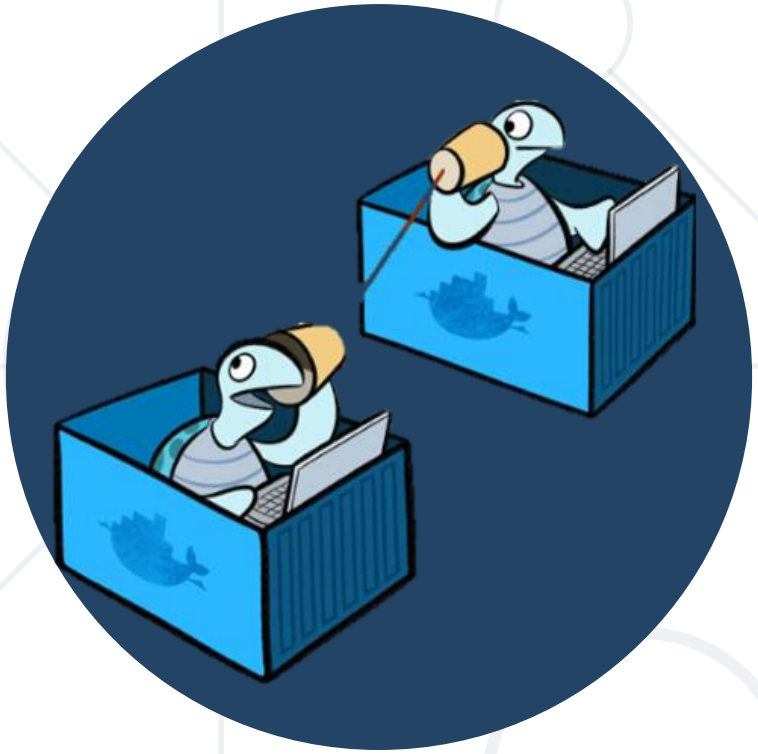
- **Run** the newly-created **image** as a **container** on the right **port**

```
PS C:\Users\      \MyWebsite> docker run -d -p 8080:8080 [redacted]/my-webapp
ab2b08161571a35ad6faa02dbf823e2524e2742d27e23b8d6c38d519397f5c1b
```

Name	Image	Status	Port(s)	Last started	Actions
kind_goldstine ab2b08161571	[redacted]/my-webapp	Running	8080:8080	3 minutes ago	

- Go to **127.0.0.1:8080** and validate the application



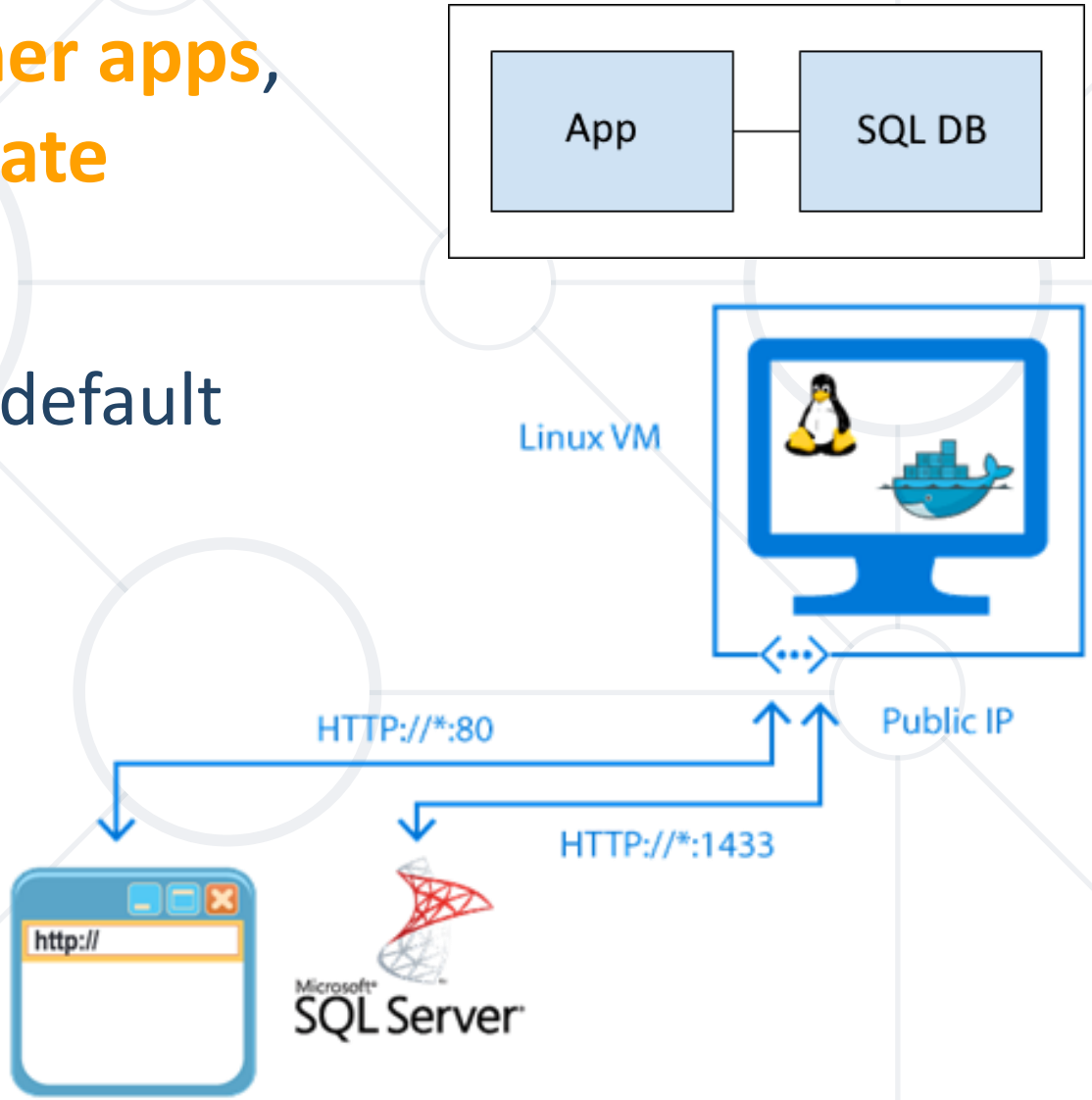


Container Networking

Communication Between Containers

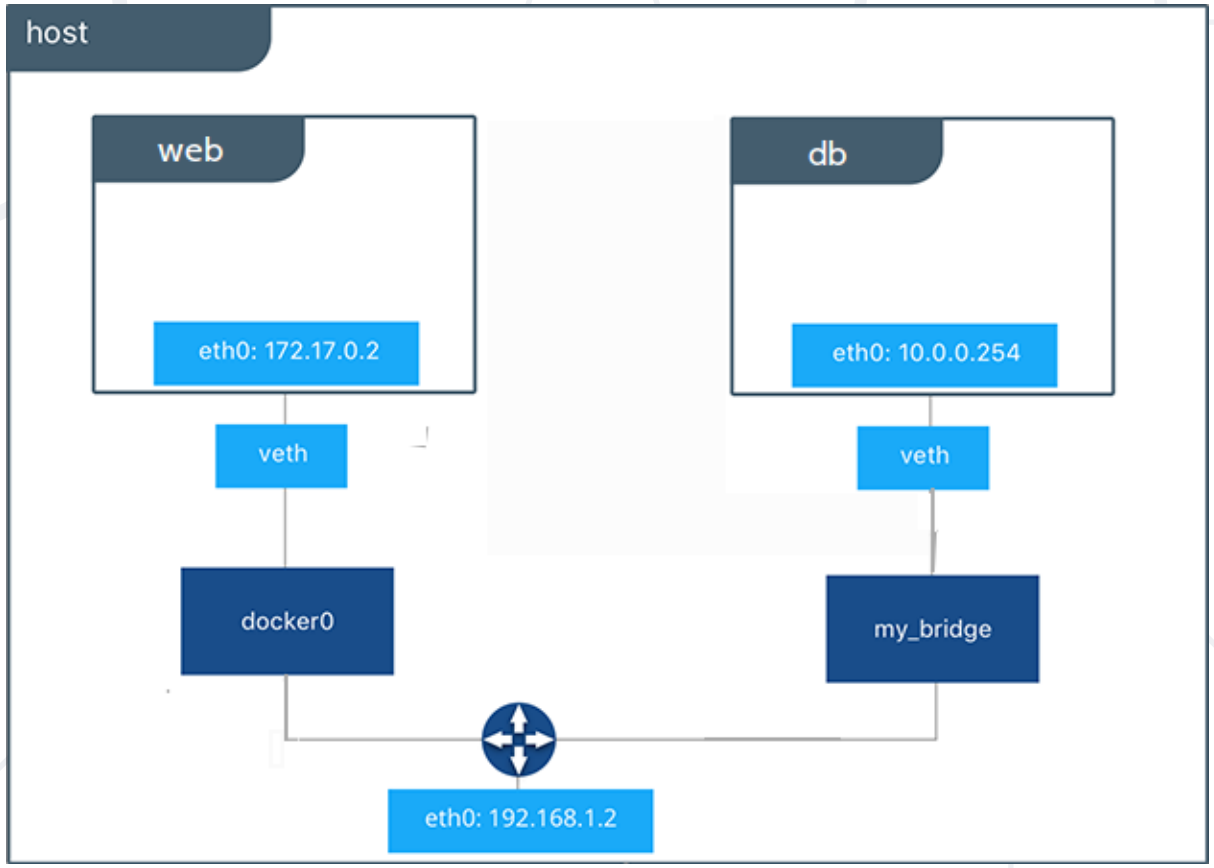
What is Container Networking?

- 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

Connecting Containers in a Network

Run App + Database Containers

Create A Network

- To connect containers, we need to **create a network** first

```
docker network create my_network
```

```
PS C:\Users\     \MyApp> docker network create my_network
36150dcdf9edafba2fe662bdfe50d378a99829be02d3d1b9e9db6f16e3b392c2
```

- Then you can **inspect it**

```
docker network inspect my_network
```

```
PS C:\Users\     \MyApp> docker network inspect my_network
[
    {
        "Name": "my_network",
        "Id": "36150dcdf9edafba2fe662bdfe50d378a99829be02d3d1b9e9db6f16e3b392c2",
        "Created": "2023-05-14T16:34:32.368057008Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.21.0.0/16",
                    "Gateway": "172.21.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

- Now we can **connect containers to the network**

No containers yet

Add MySQL Container to the Network

- Let's add the **MySQL container** to our **network**
  - **Container name** is important for other containers in the network and **should not** be random

```
docker run -dit \
--name wordpress_db \
-e MYSQL_ROOT_PASSWORD=pass \
-e MYSQL_DATABASE=wordpressdb \
-e MYSQL_USER=wordpress \
-e MYSQL_PASSWORD=wordpress \
--expose 3306 \
--expose 33060 \
--network my_network \
-v ${PWD}/data:/var/lib/mysql \
mysql
```

Container name

Network name

```
PS C:\Users\     \MyApp> docker run -dit `
>> --name wordpress_db `
>> -e MYSQL_ROOT_PASSWORD=pass `
>> -e MYSQL_DATABASE=wordpressdb `
>> -e MYSQL_USER=wordpress `
>> -e MYSQL_PASSWORD=wordpress `
>> --expose 3306 `
>> --expose 33060 `
>> --network my_network `
>> -v ${PWD}/data:/var/lib/mysql `
>> mysql
fd0e62a8b1ee19c06a0125ec831388eb24e95c36b
```

```
"Containers": {
    "fd0e62a8b1ee19c06a0125ec831388eb24e95c36bc02ee2c137252fe496b8c7e": {
        "Name": "wordpress_db",
        "EndpointID": "8d6deb5cd7482517e7c2cb9ddbd12ffd27d90dec7566fbc0671443acd9bdc0cf",
        "MacAddress": "02:42:ac:15:00:02",
        "IPv4Address": "172.21.0.2/16",
        "IPv6Address": ""
    }
},
```

Add WordPress Container to the Network

- Run the **image in a container** and attach it to our **network**

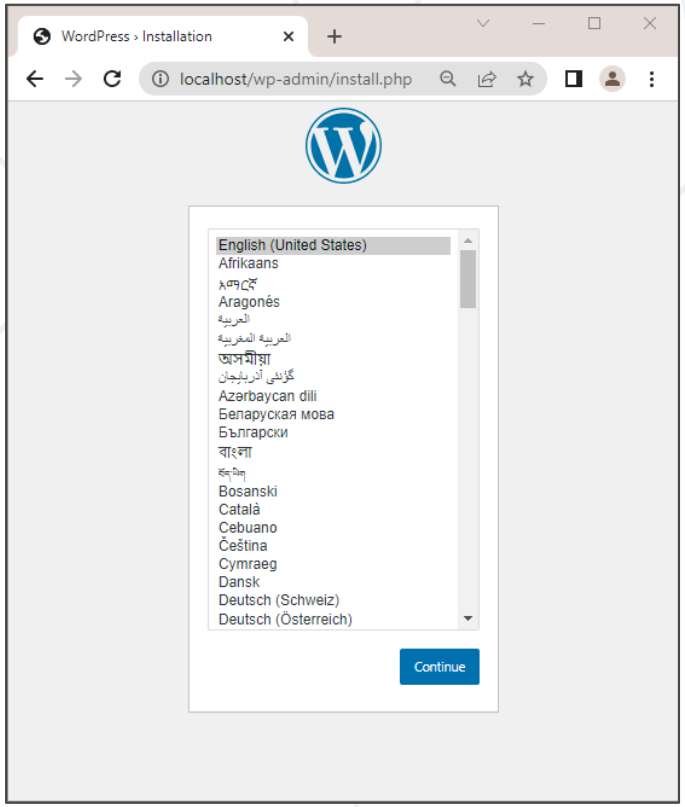
```
docker run -dit \
--name wordpress-website \
-e WORDPRESS_DB_HOST=wordpress_db \
-e WORDPRESS_DB_USER=wordpress \
-e WORDPRESS_DB_PASSWORD=wordpress \
-e WORDPRESS_DB_NAME=wordpressdb \
-v ${PWD}/wp-data:/var/www/html \
-p 80:80 \
--network my_network \
wordpress
```

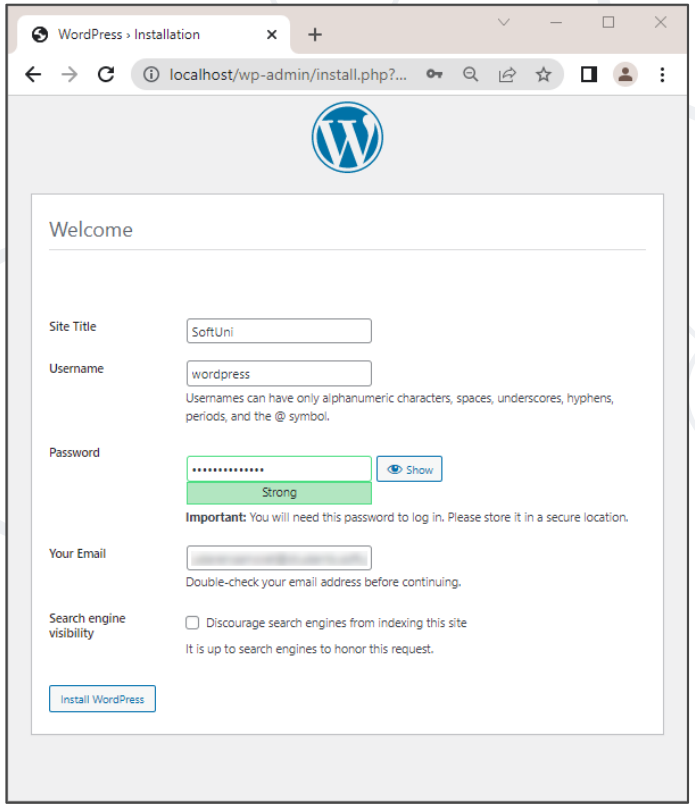
```
C:\Users\     \MyApp> docker run -dit `
--name wordpress-website `
-e WORDPRESS_DB_HOST=wordpress_db `
-e WORDPRESS_DB_USER=wordpress `
-e WORDPRESS_DB_PASSWORD=wordpress `
-e WORDPRESS_DB_NAME=wordpressdb `
-v ${PWD}/wp-data:/var/www/html `
-p 80:80 `
--network my_network `
wordpress
d6815ef6db7855f2e8b289ca80634a62dfd90218475d0387efa6c249ee1
```

- Inspect the network again to be sure that the **new** container is **attached** to it

Run App

- Go to **localhost:80**
  - You should be able to register and log in the app





Orchestration Overview

Container Orchestration

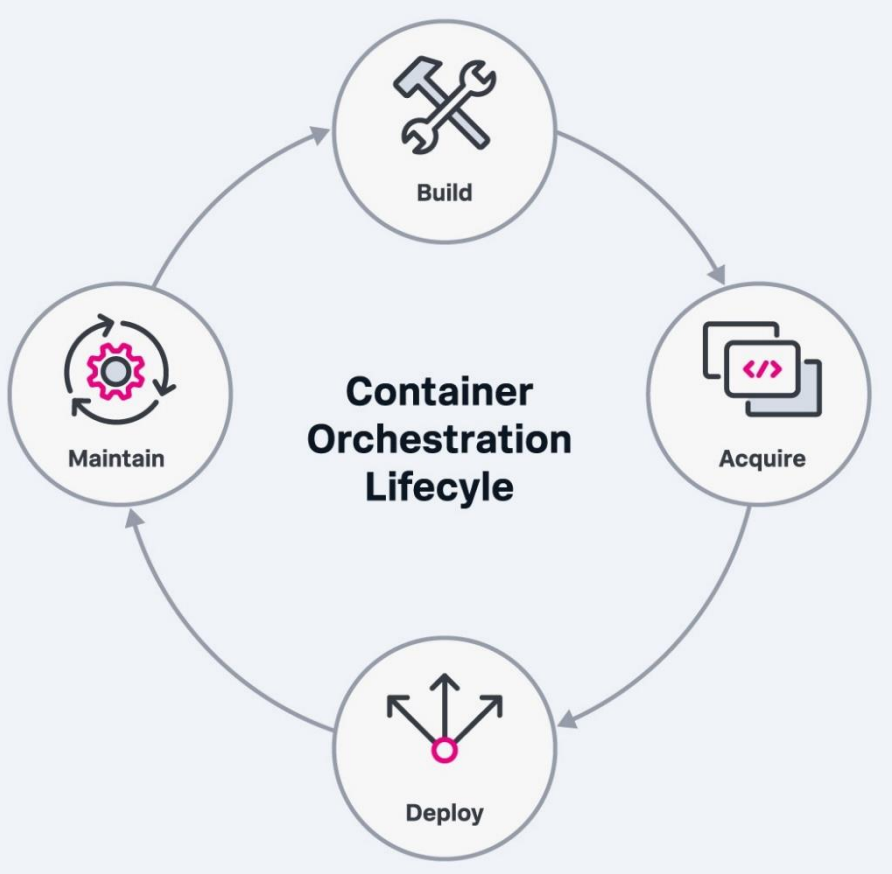
What is Orchestration?

- 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

Container Orchestration

- **Container orchestration** automates the deployment, management, scaling, and networking of containers
  - As containerizing multiple services with multiple commands is tough
- Multiple **tools**



RANCHER

Nomad

MESOS

docker SWARM

kubernetes

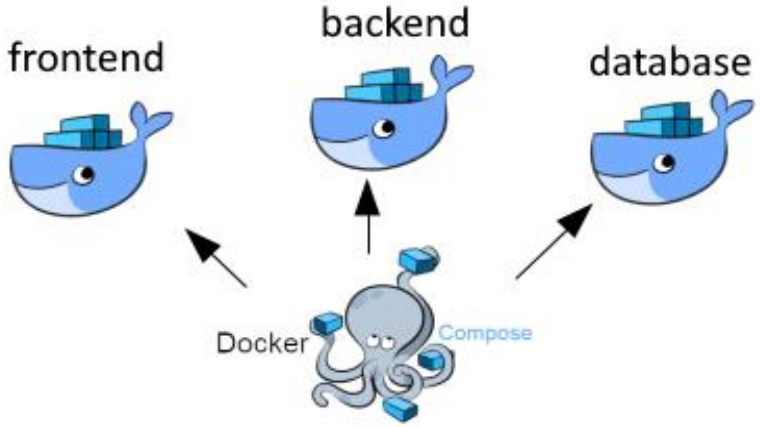
Most used is **Kubernetes**

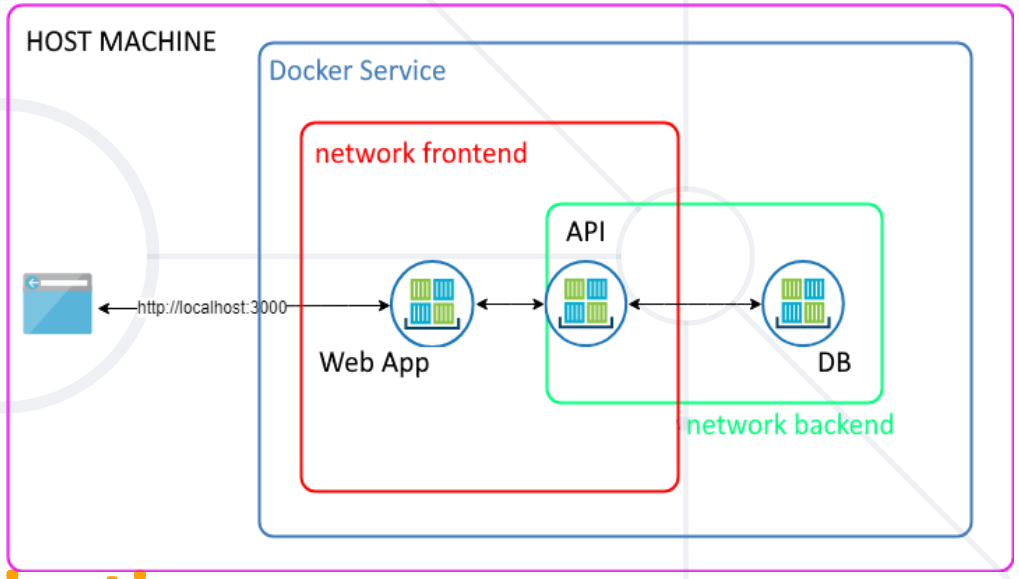
Docker Compose Orchestration Tool

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:
```

Build and Run a Multi-Container App

- Build **all images**

```
docker-compose build
```

YAML file's **directory**

```
PS C:\Users\     \mywebsitewithdb> docker-compose build
```

- Run the **containers**

```
docker-compose up
```

```
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
```

- Or in "silent" mode

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

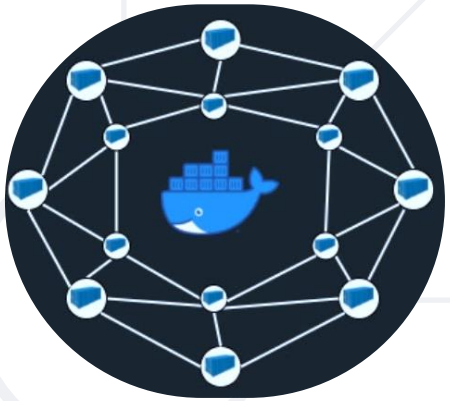
```
PS C:\Users\vikto\mywebsitewithdb> docker-compose up -d
[+] Running 2/2
 ✔ Container mywebsitewithdb-wordpress_db-1     Started
 ✔ Container mywebsitewithdb-wordpress_site-1   Started
```

- Check if services are up and running

```
docker-compose ps
```

```
PS C:\Users\     \mywebsitewithdb>  docker-compose ps
NAME                                IMAGE              COMMAND
mywebsitewithdb-wordpress_db-1      mysql:latest       "docker-entrypoint.s…"
mywebsitewithdb-wordpress_site-1    wordpress:latest   "docker-entrypoint.s…"
```

Networking in Docker Compose (1)



- 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
- For example, in our case, when you run **`docker compose up`**
  - A network called **`mywebsitewithdb_default`** was created
  - **Web** and **db containers** were created and they joined the network

```
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 (2)

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

```
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**

docker-compose.yml

```yml
version: "3.8"

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

More Docker Compose Commands

- Compose with **multiple files**

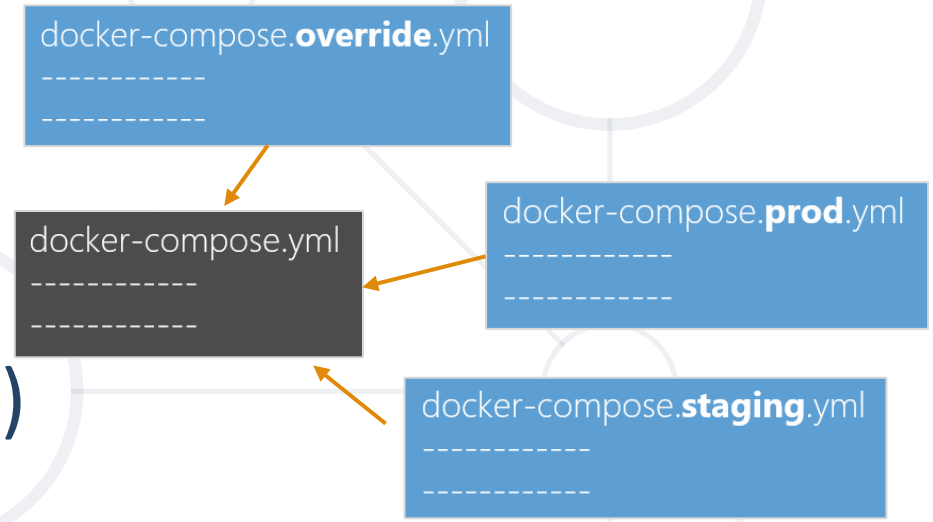
```
docker-compose -f docker-compose.yml -f production.yml up -d
```

- **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
```

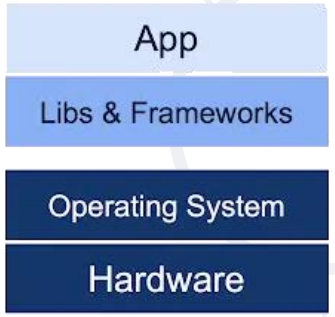


Kubernetes Overview

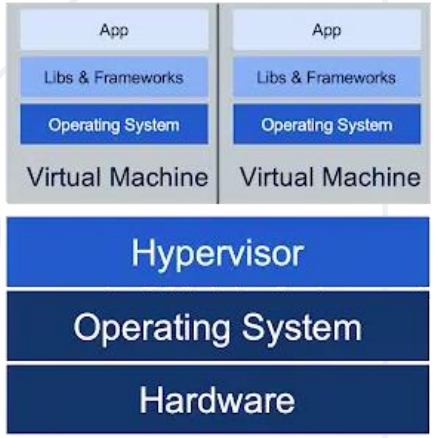
Open-source Container Orchestration Tool by Google

What is Kubernetes?

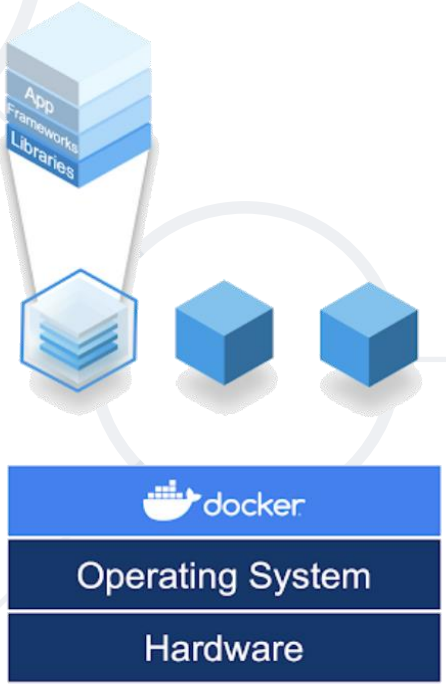
- **Kubernetes** == **container orchestration system**
  - Automates **deployment**, **scaling**, and **management** of **containerized apps**
  - Solving challenges from having **distributed apps**
  - Open-source



Traditional Deployment

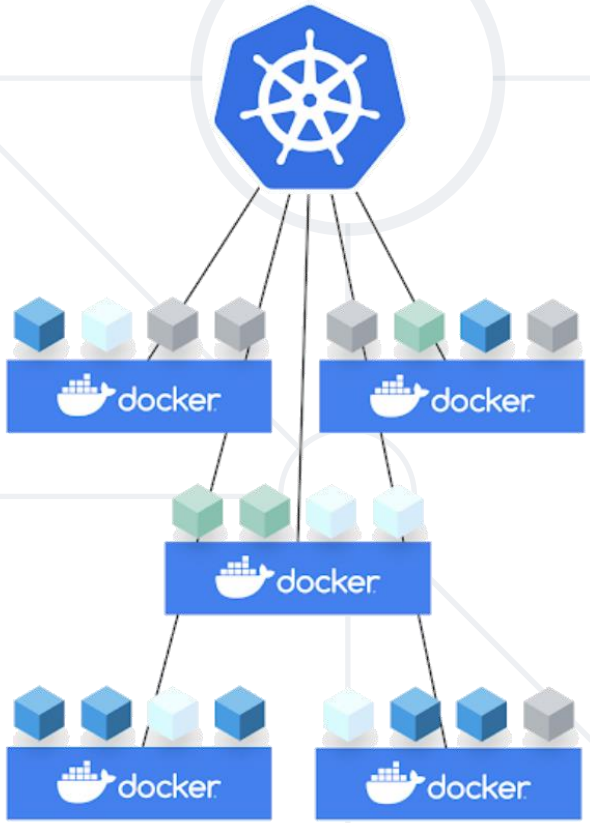


Virtualized Deployment



Container Deployment

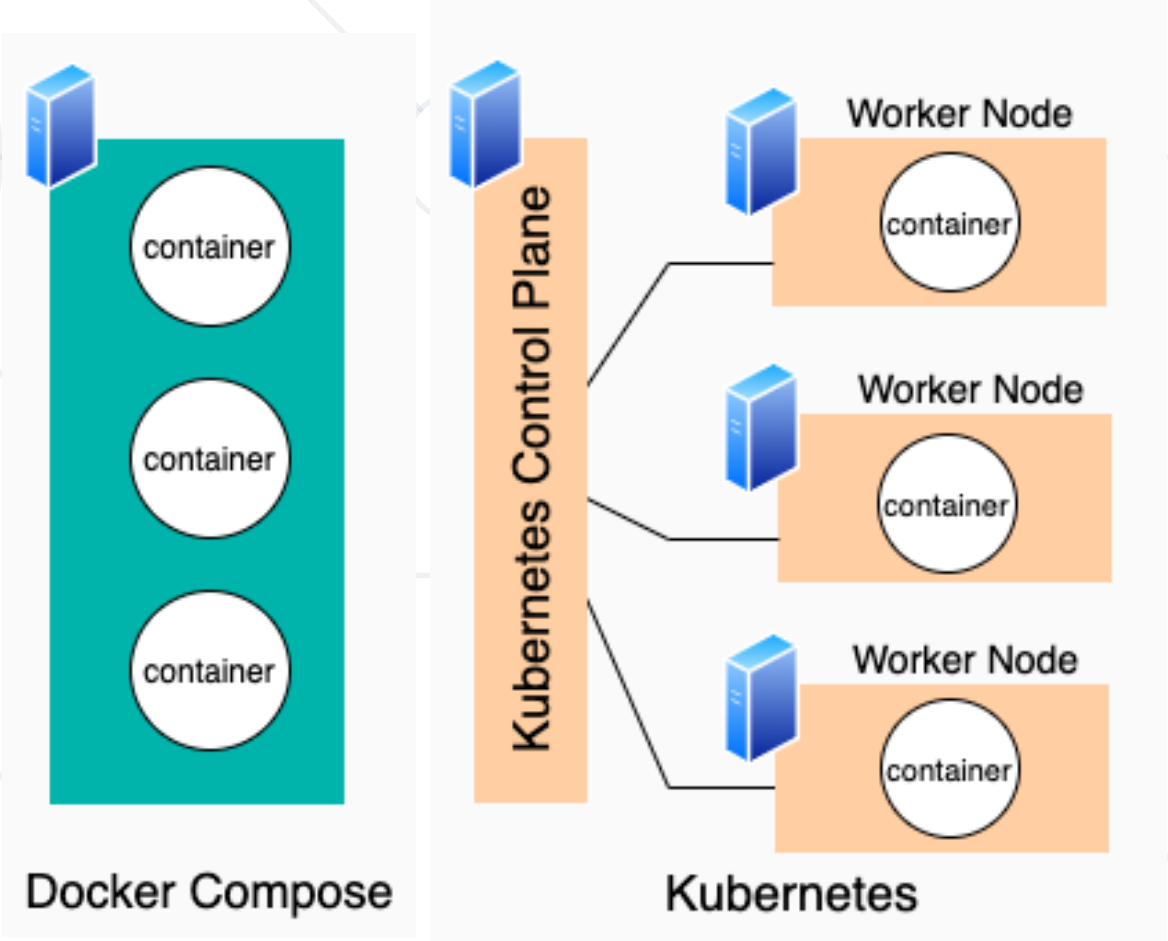
Kubernetes & Docker work together to build & run containerized applications



Kubernetes Deployment

Kubernetes vs Docker Compose

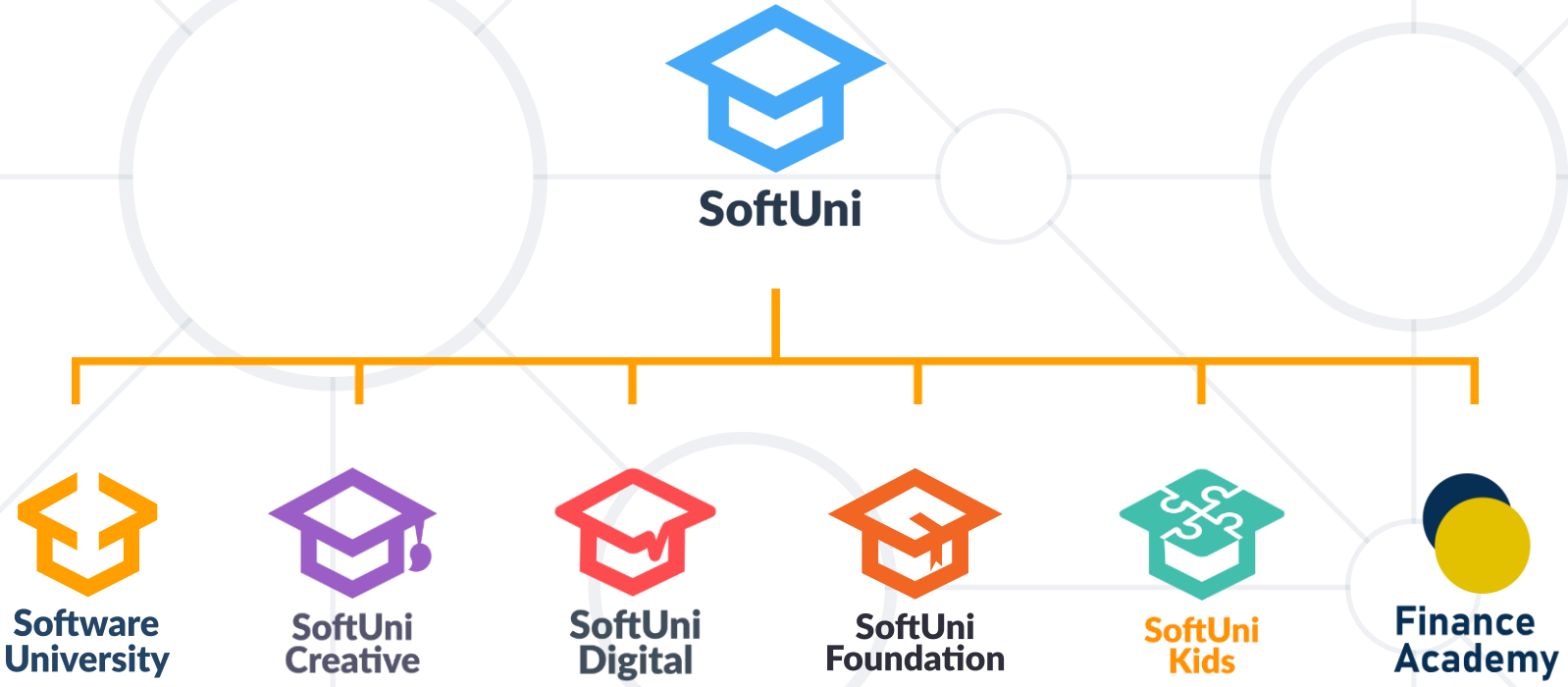
- Both are **frameworks for container orchestration**
- Main difference
  - **Docker Compose** runs containers on a **single host machine**
  - **Kubernetes** runs containers across **multiple computers**



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 Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

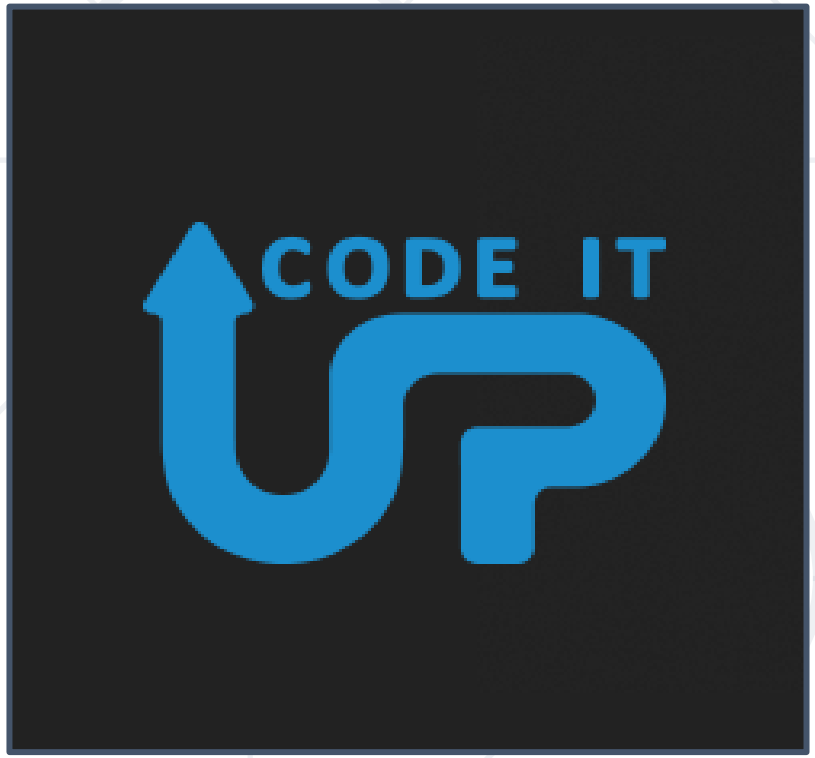
Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners



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

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
- Software University Forums
  - forum.softuni.bg