Advanced Docker

Microservices

Ludovic Chevallier and Marcel Stolin

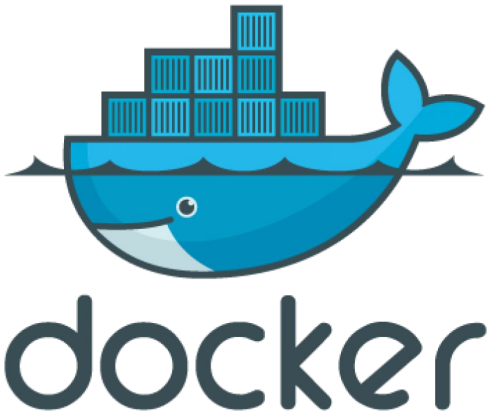
Content

- Recap
- Docker Compose
- Environment Variables
- Startup and Shutdown Order of Containers
- Restart Policies
- Persistent Storage
- Networking in Docker

Recap

Docker Recap

- Docker is a platform for shipping and running applications
- Executes applications in containers
- Containers are isolated and runnable instances of an image
- Images are templates with constructions
- Microservice architecture

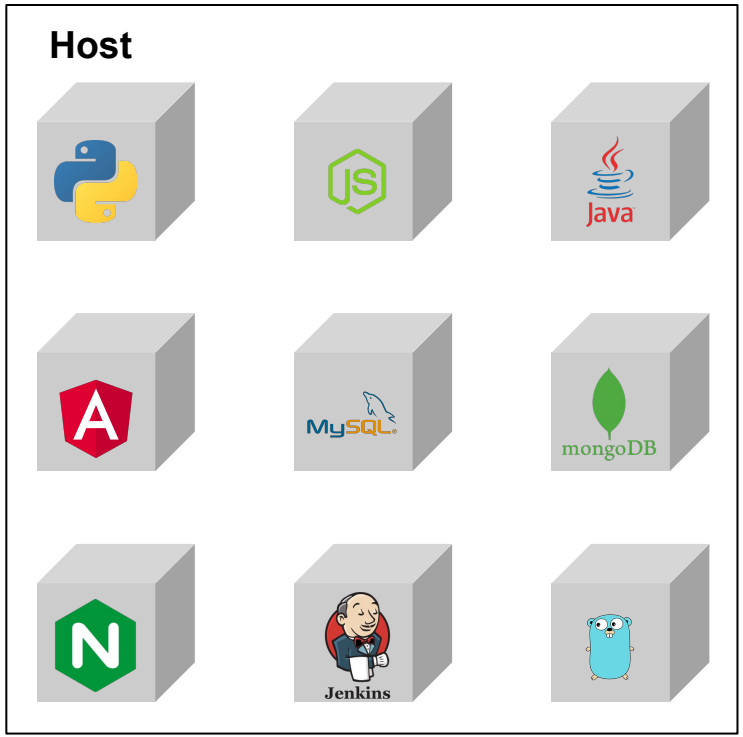


Important Docker Commands

- Run a container
  ```
  $ docker run --name Test-Container alpine:latest
  ```
- Stop a container
  ```
  $ docker stop Test-Container
  ```
- Remove a container
  ```
  $ docker rm Test-Container
  $ docker rm --force Test-Container
  ```

Docker Compose

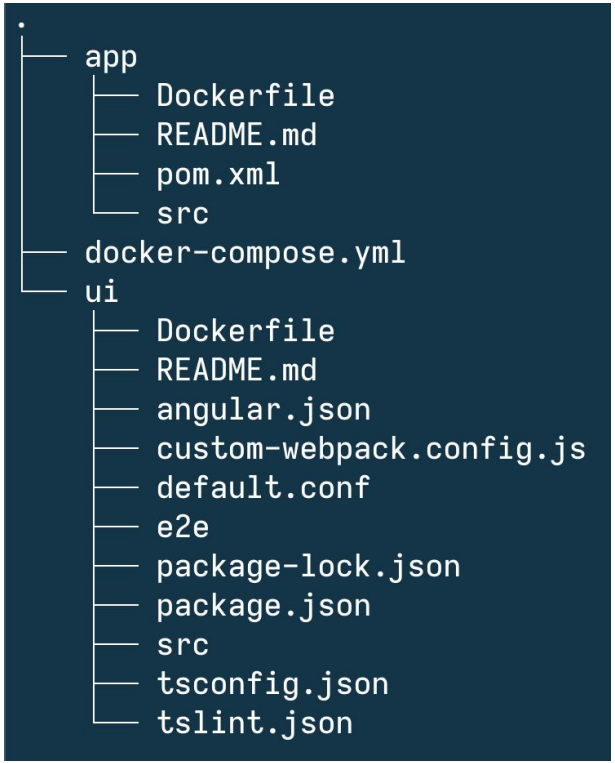
Problem Statement



Dealing with multiple microservices on Docker can become messy

Docker Compose

- Tool for multi-container/service environments
- Mostly common for development and testing workflows
- Uses a YAML file for defining the applications services
- CLI for starting, stopping, and removing an environment



Example of Docker Compose

1. Set docker-compose version
2. Define services
3. Define the **web** service

```
$ docker build -t webapp:latest \
  ./web
$ docker run --name web \
  webapp:latest
```

4. Define the **database** service

```
$ docker run --name database \
  mysql:latest
```

```
1  version:"3"

2  services:

3    web:
       build: ./web

4    database:
       image: mysql:latest
```

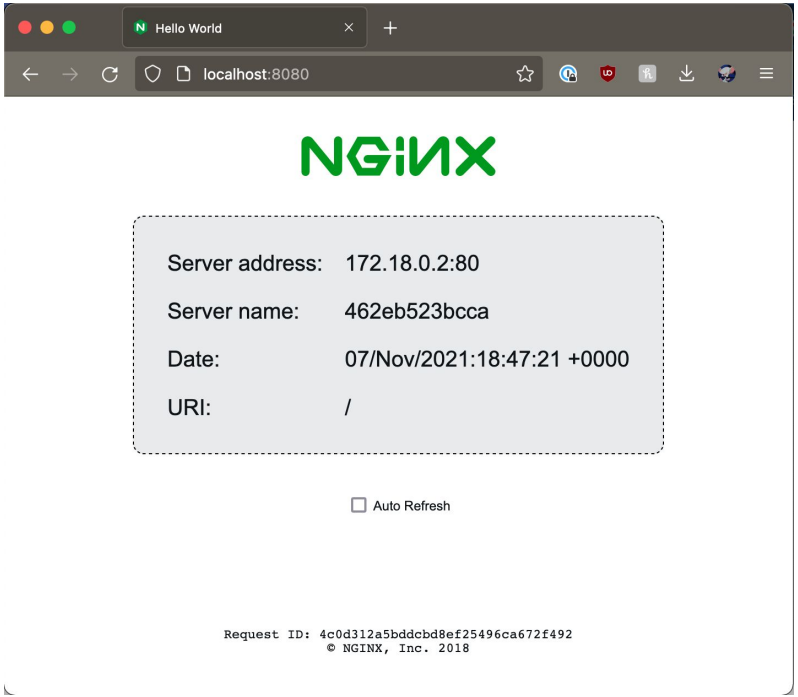
docker-compose.yml

Important Docker-Compose commands

- Execute these commands in the **same directory** as the `docker-compose.yml`
  ```
  $ docker-compose -f COMPOSE_FILE COMMAND
  ```
- Build and run the services
  ```
  $ docker-compose up
  $ docker-compose up -d
  ```
- Stopping running services
  ```
  $ docker-compose stop
  ```
- Stopping and removing running services
  ```
  $ docker-compose down
  ```

Exercise 1

- Create one service called **web**
- Use the image **`nginx:latest`**
- **Start** the environment
- **Go to** http://localhost:8080
- **Stop and remove** the environment



Environment Variables

- Environment variables affect running processes in an environment

```
$ export VALUE=”Test”
$ echo $VALUE
  Test
```

- Docker provides multiple ways defining environment variables for containers
- `--env VAR=VALUE (-e)`

```
$ docker run --env USER=admin --env PASSWORD=root alpine
```

- Multiple environment variables can be defined in a `.env` file
- `--env-file PATH`

```
$ docker run --env-file ./vars.env alpine
```

Environment Variables in Docker Compose

1. Run **web** container with environment variables

```
$ docker run --name web \
  --env MYSQL_SERVER=database \
  --env MYSQL_USER=root \
  --env MYSQL_PASSWORD=root \
  webapp:latest
```

2. Run **database** container with environment variables

```
$ docker run --name database \
  --env MYSQL_ROOT_PASSWORD=root \
  mysql:latest
```

```
version:"3"

services:
  web:
    build: ./web
    environment:
      - MYSQL_SERVER=database
      - MYSQL_USER=root
      - MYSQL_PASSWORD=root

  database:
    image: mysql:latest
    environment:
      - MYSQL_ROOT_PASSWORD=root
```

1

2

docker-compose.yml

.env Files in Docker Compose

1. Run **web** container with an environment file

```
$ docker run --name web \
  --env-file ./web-variables.env \
  webapp:latest
```

```
MYSQL_SERVER=database
MYSQL_USER=root
MYSQL_PASSWORD=root
```

web-variables.env

1

```
version:"3"

services:
  web:
    build: ./web
    env_file:
      - web-variables.env

  database:
    image: mysql:latest
    environment:
      - MYSQL_ROOT_PASSWORD=root
```

docker-compose.yml

Startup and Shutdown Order of Containers

- In some scenarios services depend on other services
- Running and stopping container in order

```
$ docker run --name database mysql:latest
$ docker run --name web webApp:latest
$ docker stop database
$ docker stop web
```

- Docker compose provides **depends_on**

```
version:"3"

services:
  web:
    build: ./web
    depends_on:
      - database

  database:
    image: mysql:latest
```

docker-compose.yml

Restart Policies

- To control whether containers restart automatically
- Ensure that containers restart in the correct order
- `--restart`

```
$ docker run --restart always alpine
```

Flag	Description
`no`	Never restart
`always`	Always restart
`on-failure`	Only restart if the container fails with an error code
`unless-stopped`	Always restart unless stopped explicitly

Restart Policies in Docker Compose

- Run container with restart policies

```
$ docker run --name web \
  --restart=always \
  webapp:latest
$ docker run --name database \
  --restart=always \
  mysql:latest
```

- Set `restart` option

```
version:"3"

services:
  web:
    build: ./web
    depends_on:
      - database
    restart: always

  database:
    image: mysql:latest
    restart: always
```

docker-compose.yml

Guided Exercise 2

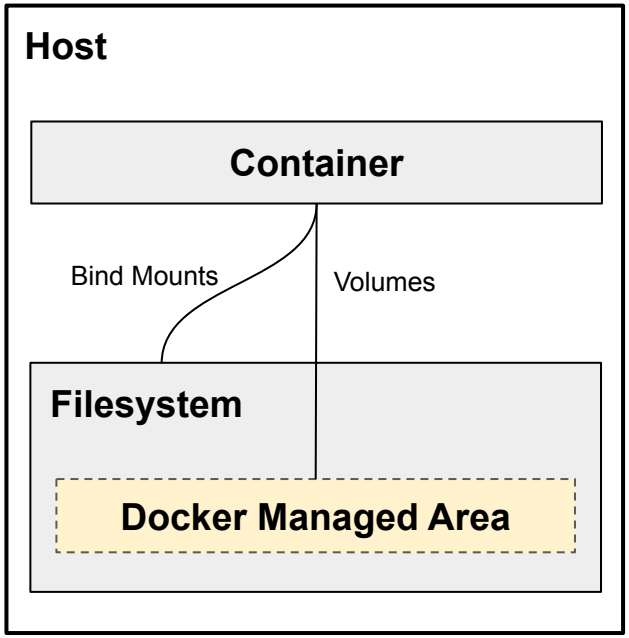
- We will create two services: mysql and phpmyadmin
- Set the environment variables
- Define the startup order: phpmyadmin depends on mysql
- Set a restart policy

Persistent Storage in Docker

Why is persistent data needed

- Data is stored inside the container
- Data inside the container is not persistent
- Better: Store data outside of the container
- **Volumes** and **Bind Mounts**

Mounts and Volumes



Volumes:

- Need to create a volume first
- Mount the volume to the container filesystem
- Data can only be modified by a Docker process

Bind Mounts:

- Mount any directory/file from the host to a container

Both can be used with `--mount` or `--volume`

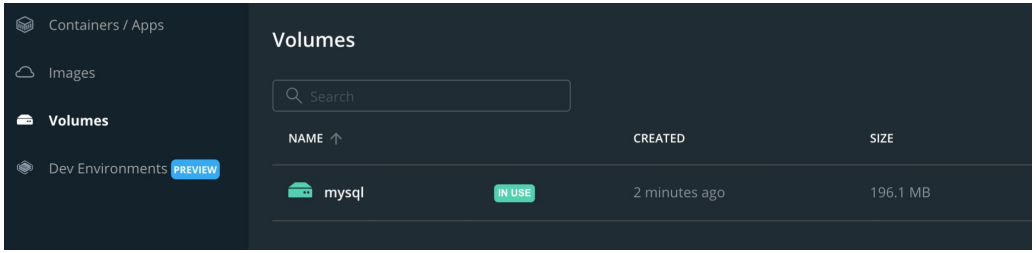
Volumes

- Mounts the volume-directory of the filesystem into the container
- Data is managed by Docker only and isolated at `/var/lib/docker/volumes`
- On Docker Desktop volumes can be seen in the Volumes section
- Create a volume

  ```
  $ docker volume create mysql
  ```

- List volumes

  ```
  $ docker volume ls
  ```



How to use Volumes

1. Create a volume

   ```
   $ docker create volume mysql
   ```

2. Mount the volume to the container

   - **`--volume`**

     ```
     $ docker run --volume mysql:/var/lib/mysql mysql:latest
     ```

   - **`--mount`**

     ```
     $ docker run --mount source=mysql,target=/var/lib/mysql mysql:latest
     ```

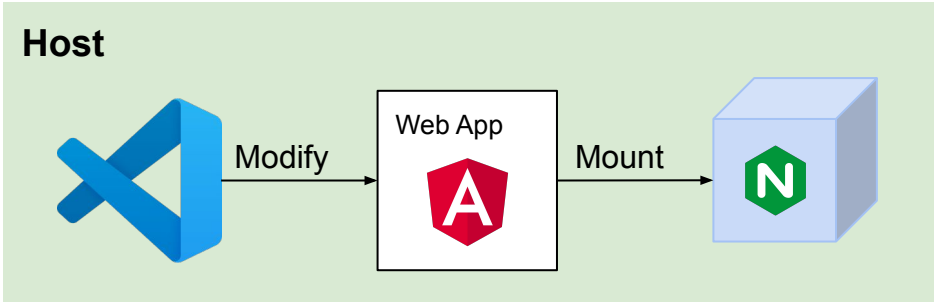
Using Volumes in Docker Compose

```yaml
version:"3"

volumes:
  mysql:

services:
  database:
    image: mysql:latest
    volumes:
      - mysql:/var/lib/mysql
```

docker-compose.yml

Bind Mounts

- Allows to mount any file or directory to the container
- Non Docker processes can modify the data (No isolation)
- If modified on the host, it is modified on the container as well
- Better suited for development purposes



How to use Bind Mounts

- Mount the file/directory to the container
  - **--volume**

```
$ docker run \
  --volume “$(pwd)”/index.html:/usr/share/nginx/html \
  nginx:latest
```

  - **--mount**

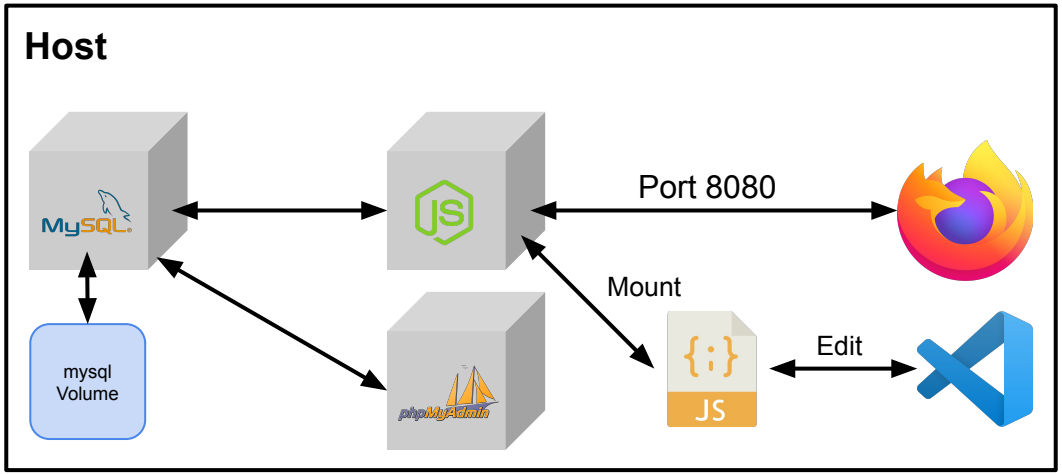
```
$ docker run \
  --mount \
  type=bind,source=”$(pwd)”/index.html,target=/usr/share/nginx/html \
  nginx:latest
```

Using Bind Mounts in Docker Compose

```
version:”3”

services:
  web:
    image: nginx:latest
    volumes:
      - ./index.html:/usr/share/nginx/html
```

docker-compose.yml

Guided Exercise 3



Volumes vs. Bind Mounts

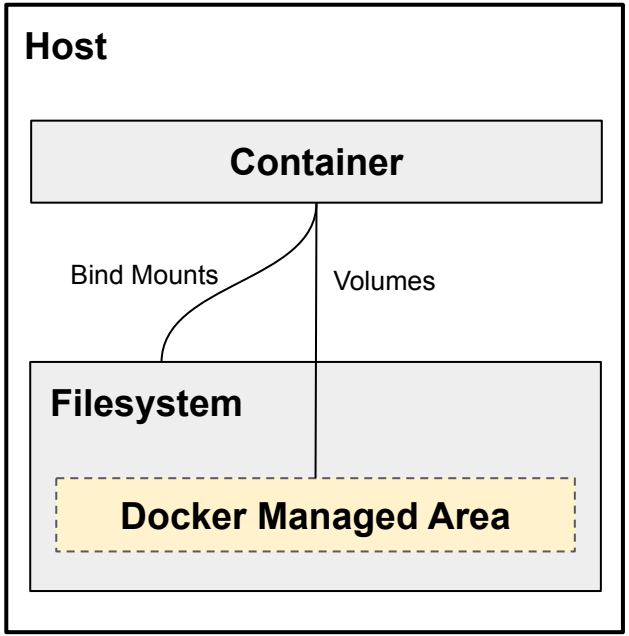
Advantages of Volumes:

- Better performance
- Managed by Docker
- Better security
- Can be shared

Use-Cases for Bind Mounts:

- Sharing host configuration, e.g. `/etc/host`
- For development environments

Use Volumes whenever possible



Docker Networks

Docker Networking

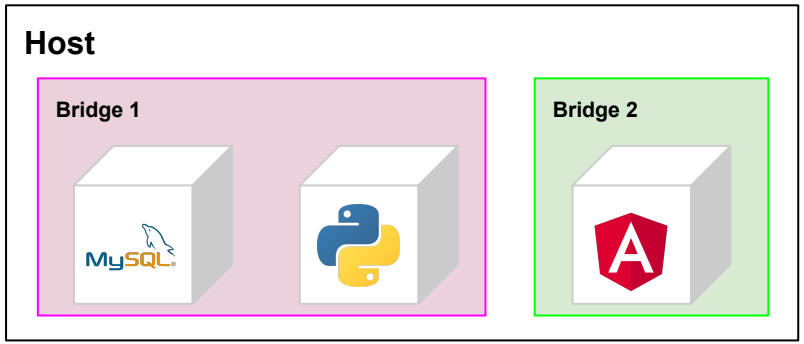
- Containers can be connected together or isolated from another
- Docker Networks is pluggable using Drivers
- Docker provides several drivers by default
- All newly created container are connected to the Bridge network

Network Drivers

Type	Meaning	Use-Case
Bridge	Default one Can communicate through IP adresse	Communication between container
Host	Use the host network	Container need to handle a large range of ports Swarm services
Overlay	Connect docker Daemon Swarm service communications	Swarm services
Macvlan	MAC adresse	Need to be connected to the physical network
None	Disable networking	For custom divers

Bridge Networks - Overview

- A link layer that forwards traffic
- Containers can connect to a bridge for communication
- Isolation from non-connected containers
- Default bridge network



Bridge Networks - User Defined Bridges

- Custom bridge networks
- Better isolation
- Better DNS resolution
- No port publishing necessary
- Create a network

```
$ docker network create custom-network
```

- Join a network

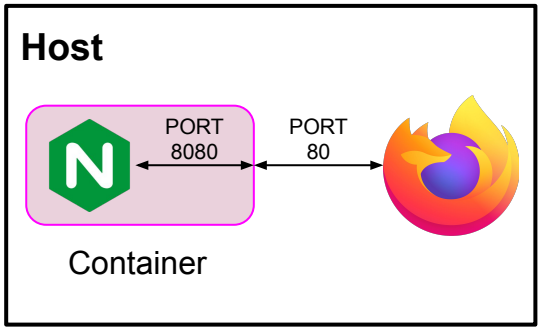
```
$ docker run --network custom-network nginx:latest
```

Exposing Ports

- By default ports are not exposed
- Services running on a container are not available
- `--publish (-p)`

```
$ docker run --publish 80:8080 nginx:latest
```

Docker host port : Container port

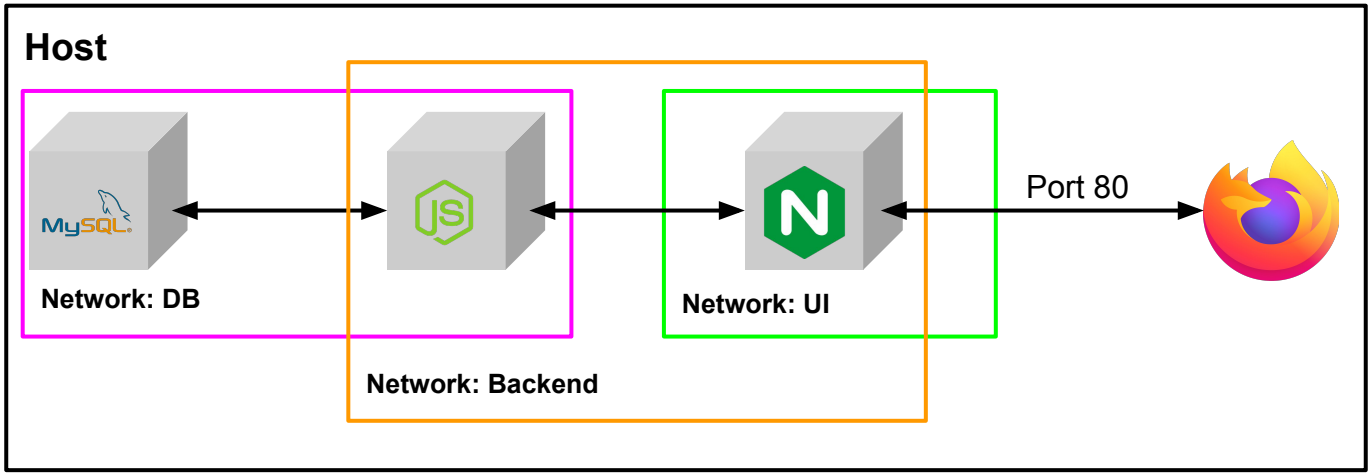


Networks in Docker Compose

```
version:”3”

networks:
  frontend:
  backend:

services:
  web:
    build: ./web
    networks:
      - frontend
    ports:
      - “8080:80”

  database:
    image: mysql:latest
    networks:
      - backend
    ports:
      - “3306:3306”
```

docker-compose.yml

Guided Exercise 4



Outlook

- Resource Management
- Docker compose in production
- Volume Backups
- Custom Network Drivers
- Scale Services using Docker Compose