docker

up leveled

Author: Martin Knecht

Docker in the Software Universe

Frontend → Backend → DevOps

HTML

CSS

React

NodeJS

Databases

Cloud

Netlify
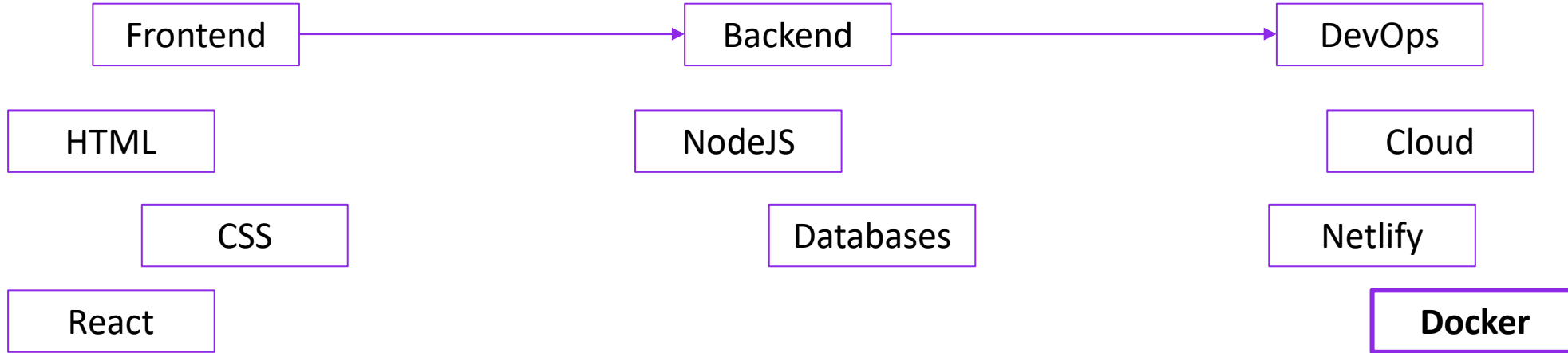
Docker

What is Docker?

- Docker makes **deployment** of software packages really **easy**.

- **Docker** can be seen as a **sandbox** in which you can execute software packages.
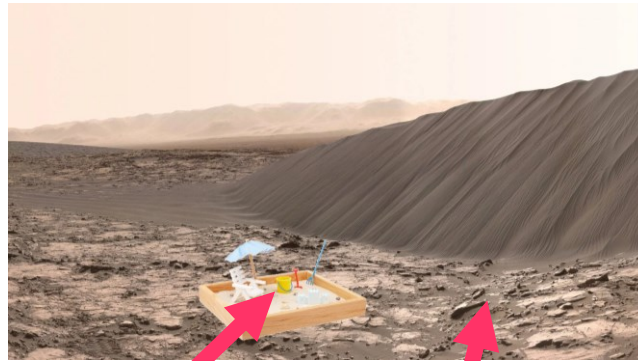
What is a Sandbox?

- Inside the Sandbox you do not care about the environment
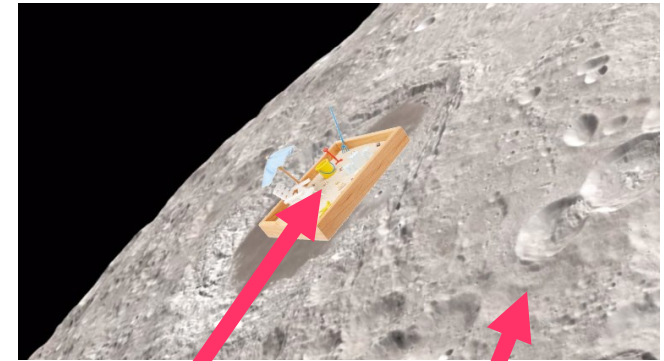- When inside the Sandbox, you can only change things inside the sandbox, not outside.

MacOS   Docker

Docker   Windows

Docker   Linux

Test Docker installation

Installation Guide: https://github.com/upleveled/system-setup/blob/master/windows.md

$ docker run hello-world

...

Hello from Docker!

This message shows that your installation appears to be working correctly.

...

Outline

Docker Architecture

Docker Images & Containers

Docker Stacks & Services

Create Images

Create Images

Stacks & Services
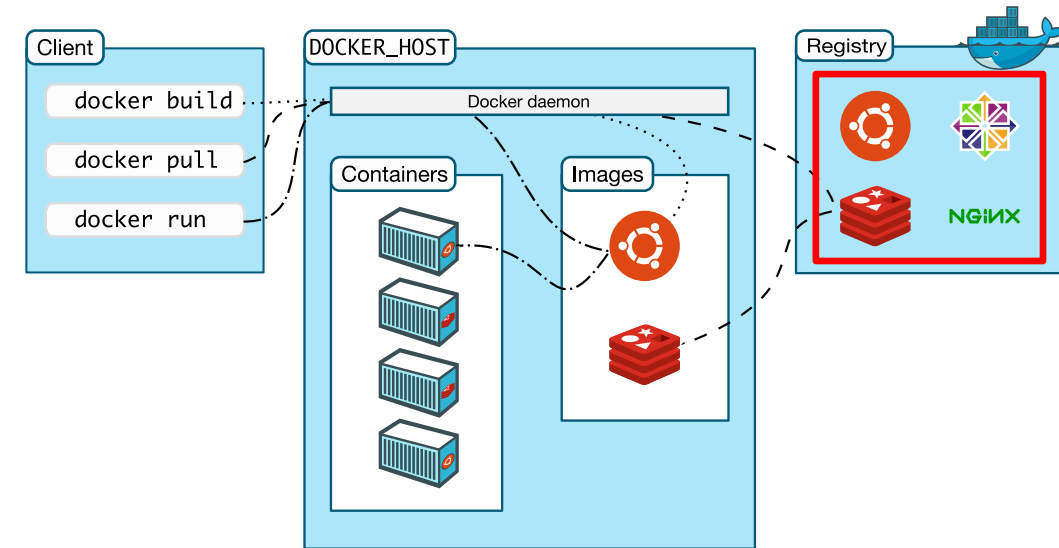
Images &
Containers

Architecture

Terminology

We can execute Software Packages inside the Sandbox.

Terminology

We can execute Software Packages inside the Sandbox.
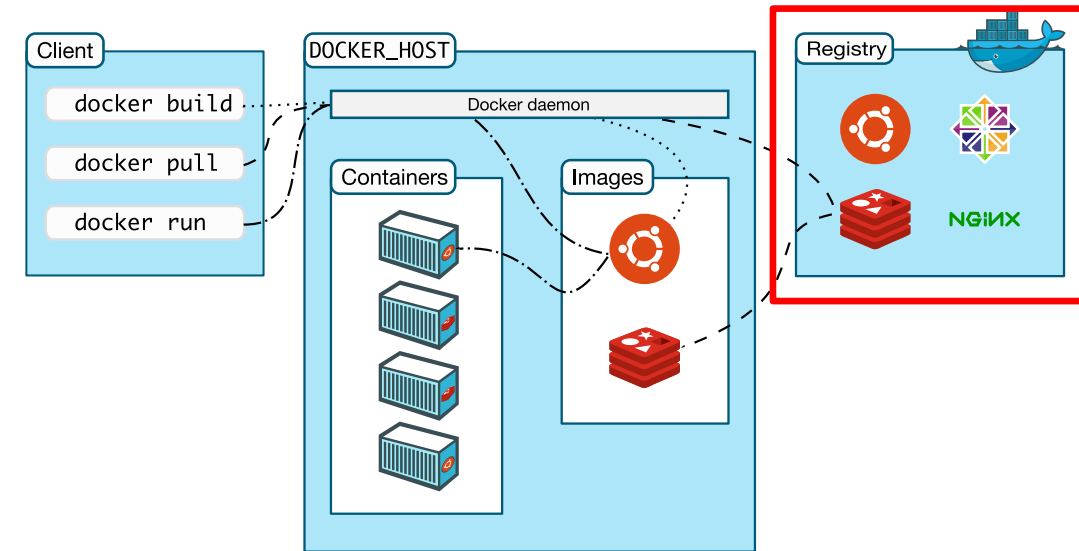
- Software Package (not executed) ➔ **Docker Image**

Terminology

We can execute Software Packages inside the Sandbox.

- Software Package (not executed) ➔ **Docker Image**

- Store for Docker Images ➔ **Docker Registry**

Terminology

We can execute Software Packages inside the Sandbox.

- Software Package (not executed) ➜ **Docker Image**

- Store for Docker Images ➜ **Docker Registry**

- Sandbox ➜ **Docker Daemon**

Terminology

We can execute Software Packages inside the Sandbox.

- Software Package (not executed) ➜ **Docker Image**

- Store for Docker Images ➜ **Docker Registry**

- Sandbox ➜ **Docker Daemon**

- **Executed Docker Image** in the **Docker Daemon** ➜ **Docker Container**
    - A Container is an instance of a Docker Image

Terminology

We can execute Software Packages inside the Sandbox.

- Software Package (not executed) ➜ **Docker Image**

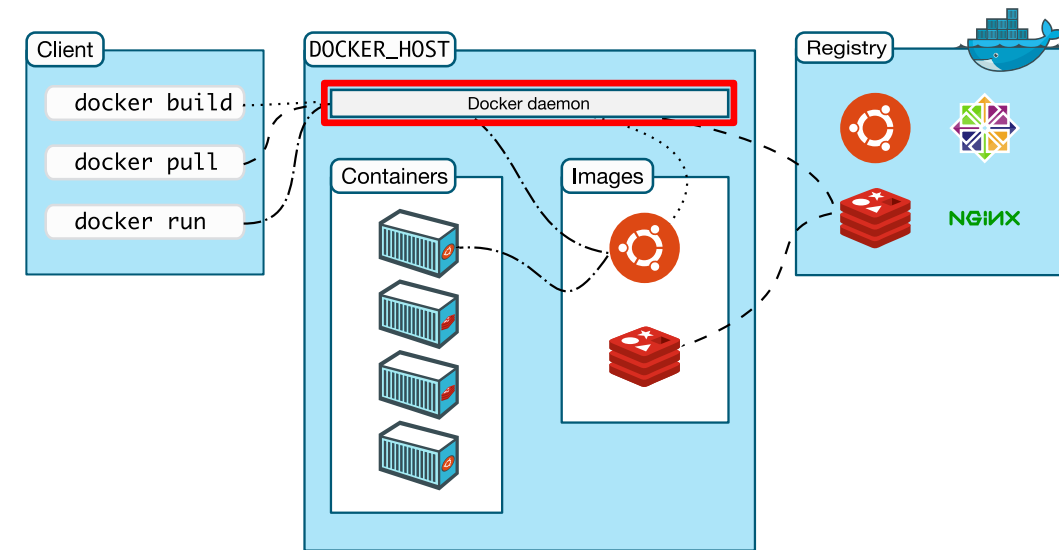- Store for Docker Images ➜ **Docker Registry**
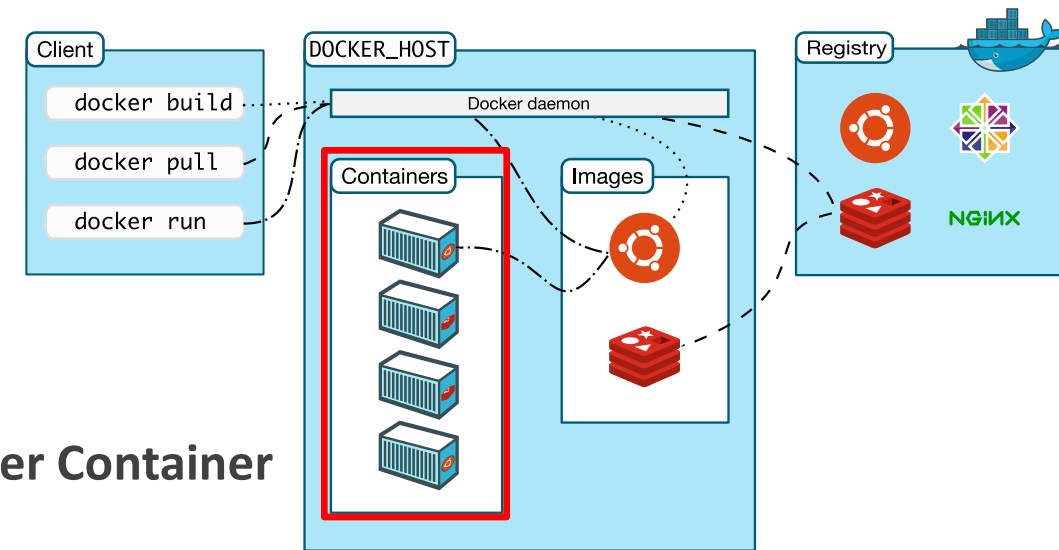
- Sandbox ➜ **Docker Daemon**

- **Executed Docker Image** in the **Docker Daemon** ➜ **Docker Container**
  - A Container is an instance of a Docker Image

- **Client** enter commands
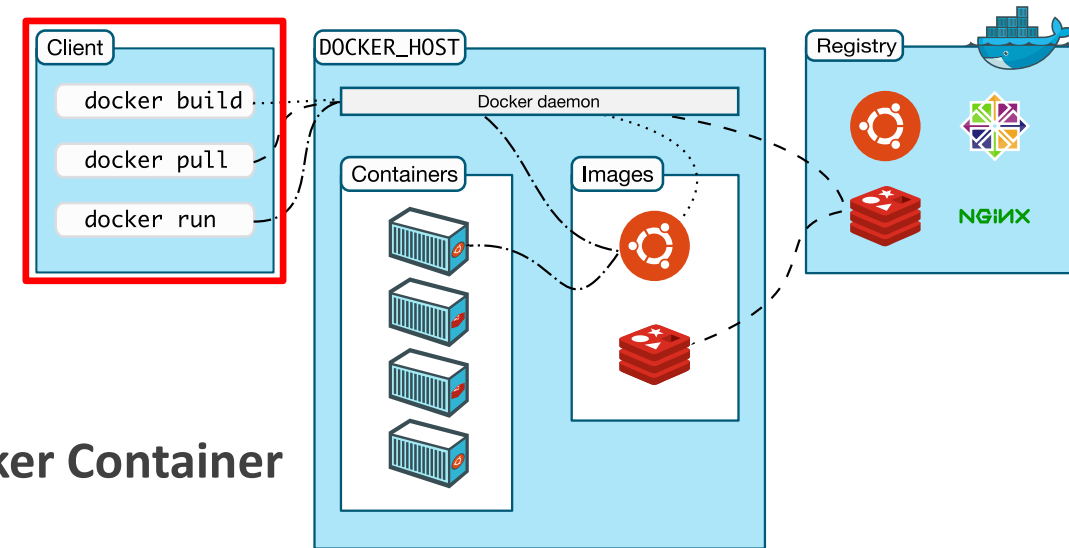
Outline

Docker Architecture

Docker Images & Containers

Docker Stacks & Services

Create Images

Docker Images & Containers

- Docker Images are stored in Docker Registries

- Largest Registry: https://hub.docker.com

- Contains tons of images

- Allows you to startup awesome stuff within minutes

Docker Images & Containers

- Let's assume we would like to run our own Analytics Tool => Matomo

- Matomo Requirements
  - PHP installed on server
  - MySQL/MariaDB Database running somerwhere
  - Apache/Nginx/IIS webserver

  ➔ https://matomo.org/docs/installation

  - **Use Docker!** ☺
    - **$ docker run –p 8080:80 matomo**
    - In your browser: localhost:8080

What just happened?
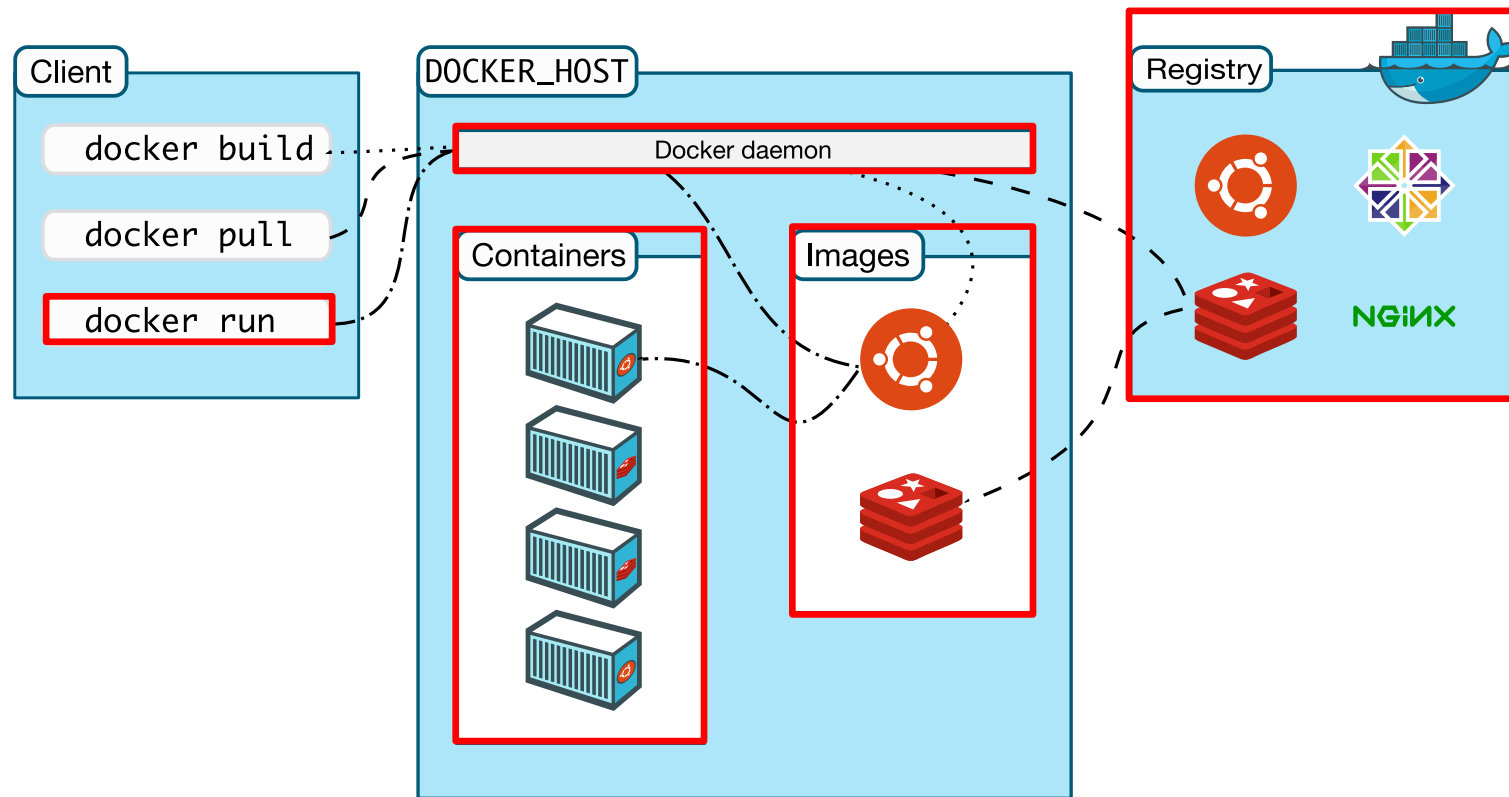
$ docker run –p 8080:80 matomo

Docker Images & Registry

- $ docker run **–p 8080:80** matomo

- Maps localhost port 8080
  to container port 80

Docker Host (your machine)

Docker Daemon (the Sandbox)

Matomo
Container

80

8080
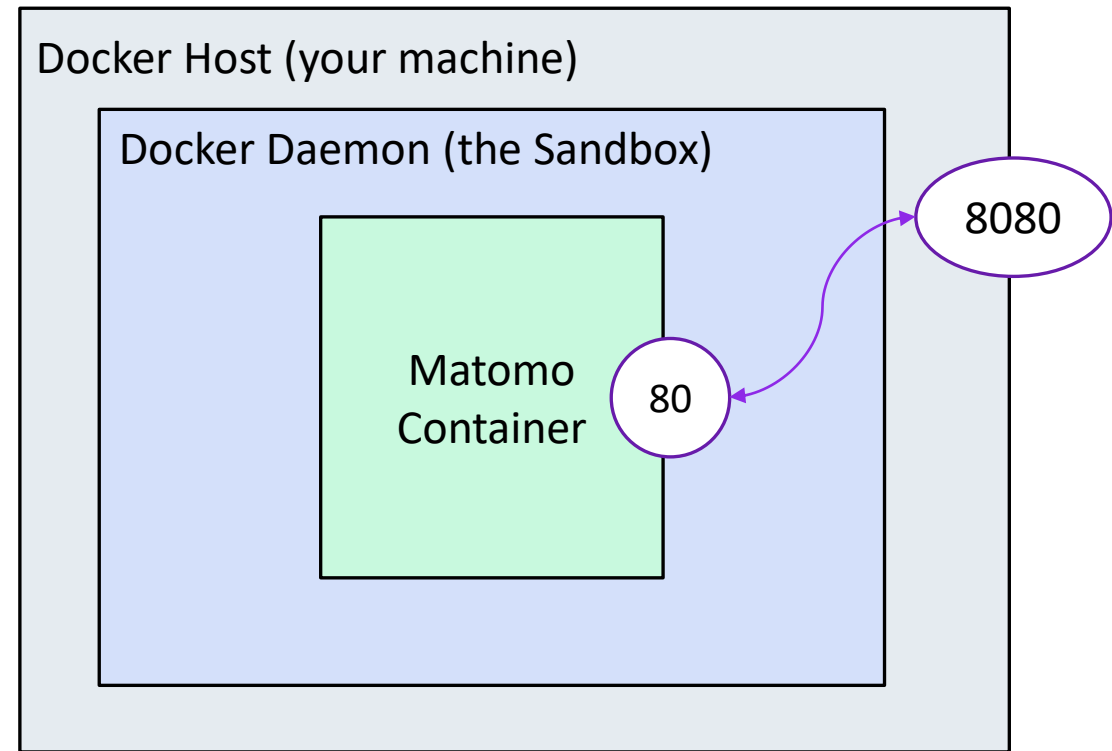
Some Docker Commands

- List all running containers: **$ docker ps**

- List all containers: **$ docker ps –a**

- Stop a container: **$ docker stop <containerId or name>**

- Start a container: **$ docker start <containerId or name>**

- Start a container and attach to output: **$ docker start –a <containerId or name>**

- See logs: **$ docker logs <containerId or name>**

- Follow logs: **$ docker logs -f <containerId or name>**

- List all local images: **$ docker image ls**

Docker Images & Containers

- Let's assume we would like to run our own Analytics Tool =>
Matomo

  - **$ docker run –p 8080:80 matomo**
  - In your browser: localhost:8080

- We are missing the MariaDB database and Matomo will not work
without it!

Outline

Docker Architecture

Docker Images & Containers

Docker Stacks & Services

Create Images

Docker Stacks

- For a full Matomo setup, we also need a MariaDB database
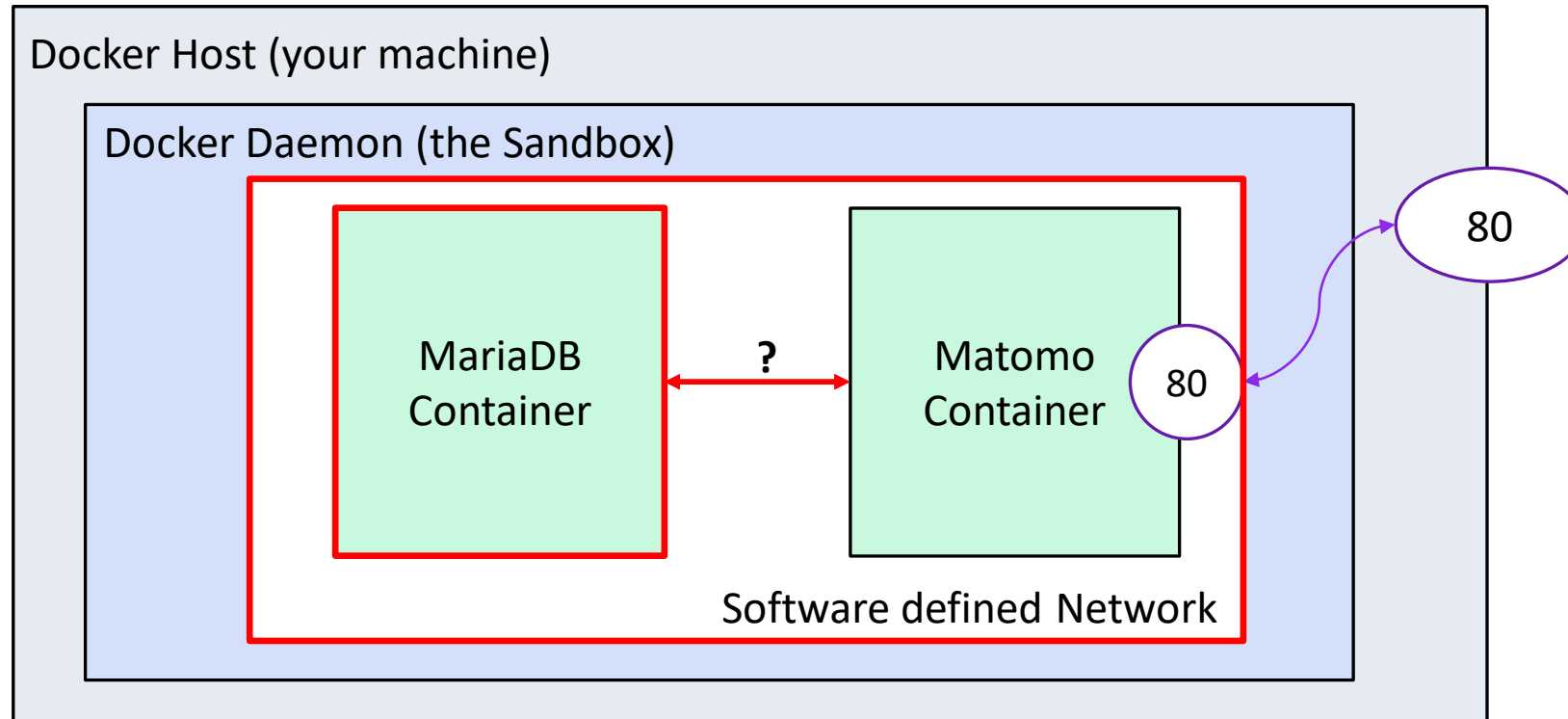
Docker Stacks

- This whole setup is called a „**Docker Stack**"

- Defined through .yml file ➔ stack.yml

Docker Host (your machine)

Docker Daemon (the Sandbox)

MariaDB Container **?** Matomo Container

80

80

Software defined Network
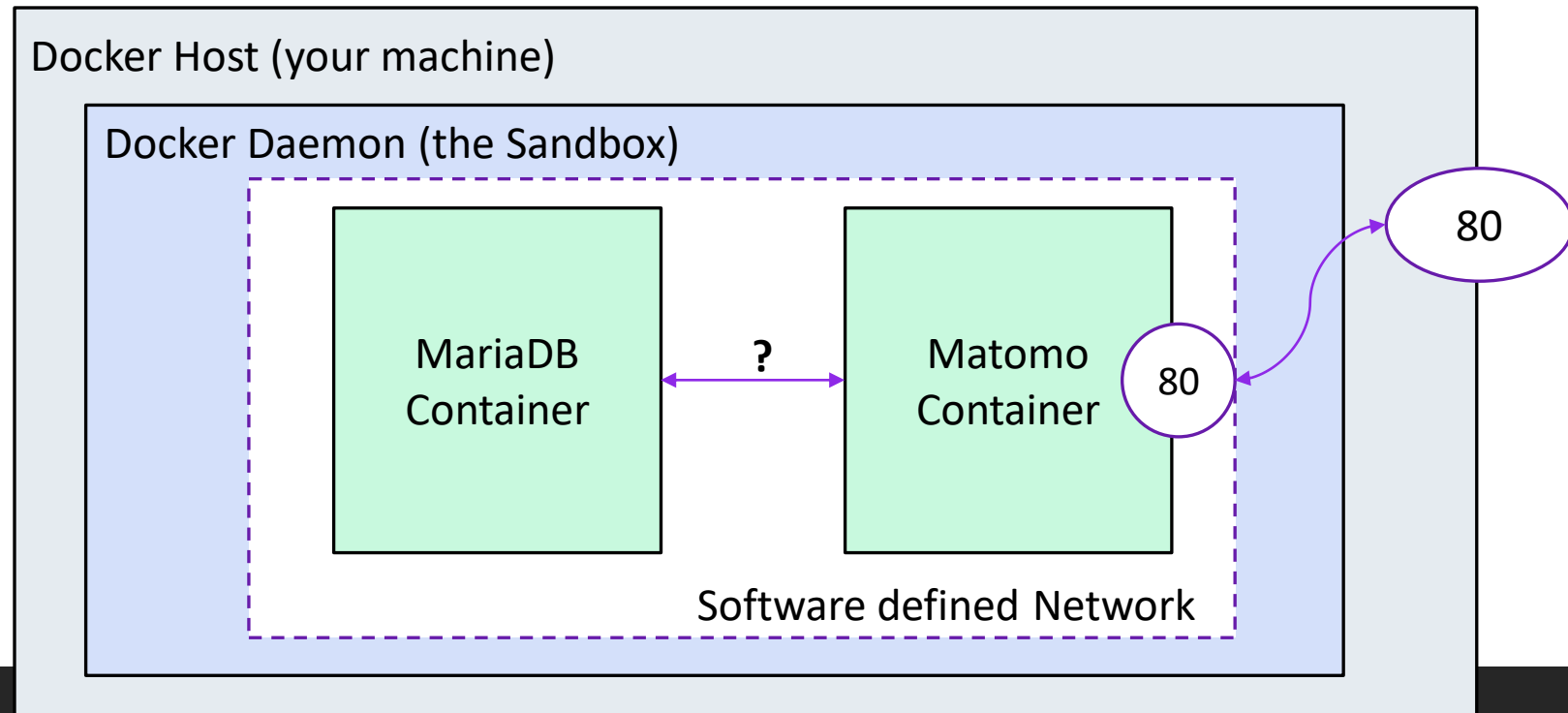
Docker Stacks

- Create new project repository and a new file called: stack.yml

```
version: „3.8"
services:
        mymatomo:
                image: matomo
                ports:
                        - 80:80
                networks:
                        - mynetwork
networks:
        mynetwork:
```

- Indents (in Spaces not Tab) are important in this file format!

Docker Stacks

- Before we can do anything with this file call:
  - **$ docker swarm init**
  - Make sure that old matomo container is not running anymore

- Now we can „deploy" this stack
  - **$ docker stack deploy –c stack.yml MyStack**

- Browser: localhost:80

What just happened?

- Docker **created** a new stack called:
  „MyStack"

- Docker **created** a software defined **network** called:
  „MyStack_mynetwork "

- Docker **created** a **service** called:
  „MyStack_mymatomo"

Some Docker Commands

- List all running stacks: **$ docker stack ls**

- List all services for this stack: **$ docker stack services <stackname>**

- List all networks: **$ docker network ls**

- Remove a stack: **$ docker stack rm <stackname>**

- Deploy a stack: **$ docker stack deploy –c <.yml file> <stackname>**

Docker Stacks – Add database service

```yaml
version: „3.8"
services:
        mymatomo:
                image: matomo
                ports:
                        - 80:80
                networks:
                        - mynetwork
        mydatabase:
                image: mariadb
                networks:
                        - mynetwork
networks:
        mynetwork:
```

Docker Stacks

- Check if deployed stack is still running
  - **$ docker stack ls**

- Update deployed stack while it is running*:
  - **$ docker stack deploy –c stack.yml MyStack**

- Do you think this is going to work?

- Does the MariaDB container start?

* Unfortunately, does not work all the time! ;-)

Exercise 1

Find out, why MariaDB is not running.

Hint: Use the commands we've already learned

Docker Stacks
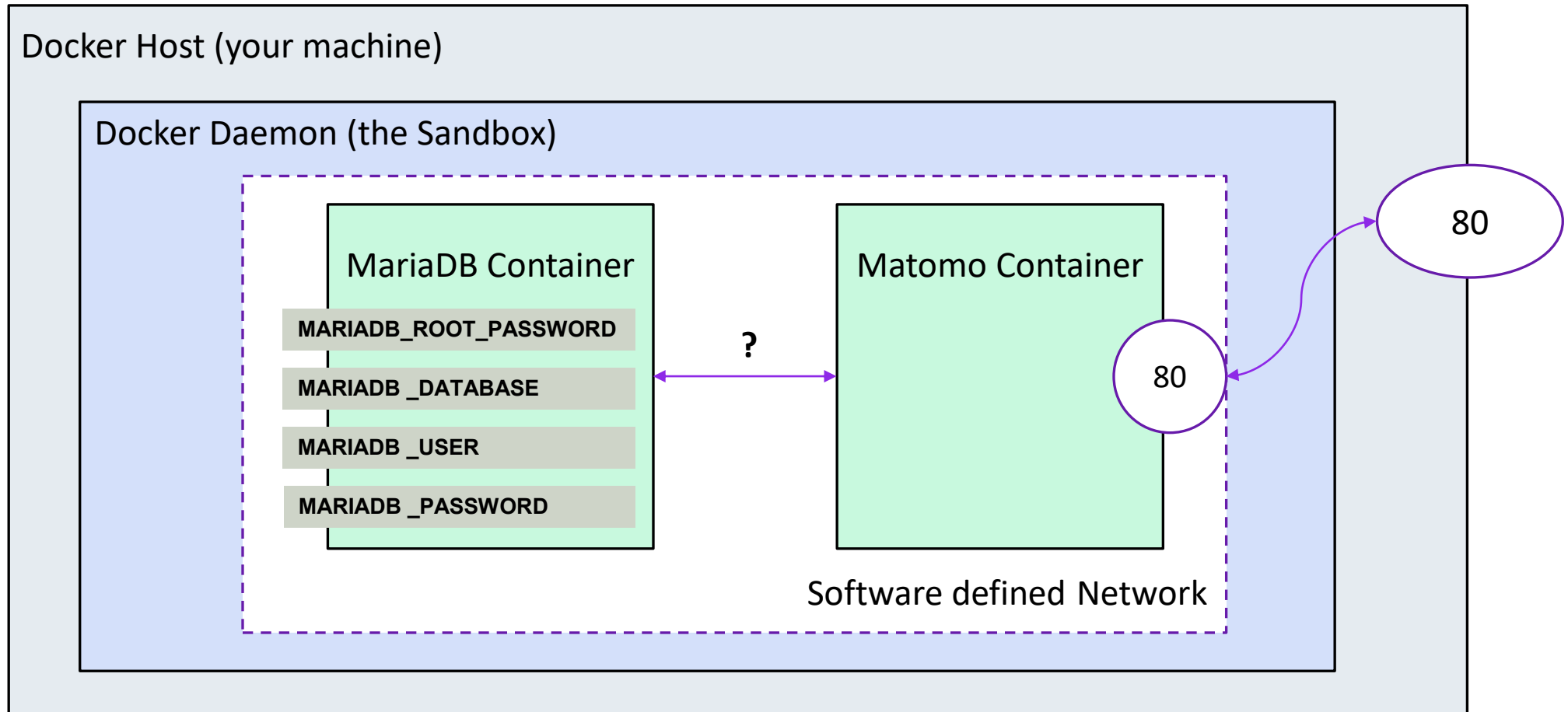
```
$ error: Database is uninitialized and password option is not specified
     You need to specify one of MARIADB_ROOT_PASSWORD,
MARIADB_ALLOW_EMPTY_ROOT_PASSWORD and
MARIADB_RANDOM_ROOT_PASSWORD
```

- MariaDB does not know enough to startup the database server

- We need to tell this to the services

- ➜ **Environment Variables!**

Docker Services - Environment Variables

Docker Services - Environment Variables

- Add environment variables at **mydatabase section** into .yml file

```
....
mydatabase:
          image: mariadb
          environment:
                    MARIADB_ROOT_PASSWORD: my-secret-root-pw
                    MARIADB _DATABASE: matomo_db
                    MARIADB _USER: matomo_user
                    MARIADB _PASSWORD: my-secret-matomo-pw
          networks:
                    - mynetwork
```
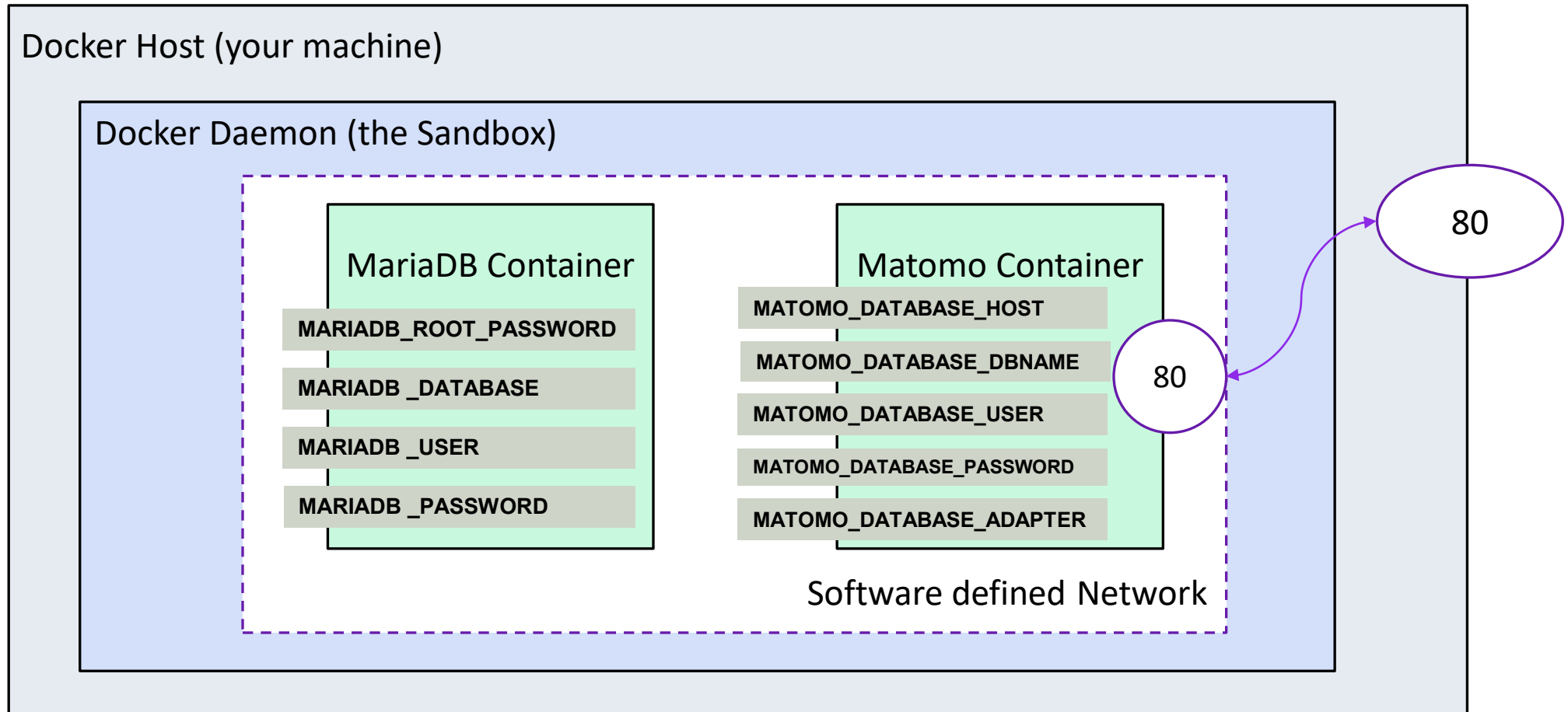
Docker Services - Environment Variables

Docker Services - Environment Variables

- Add environment variables at **mymatomo section** into .yml file

```
….
mymatomo:
        image: matomo
        ports:
                - 80:80
        environment:
                MATOMO_DATABASE_HOST: ???
                MATOMO_DATABASE_ADAPTER: mysql
                MATOMO_DATABASE_USERNAME: matomo_user
                MATOMO_DATABASE_PASSWORD: my-secret-matomo-pw
                MATOMO_DATABASE_DBNAME: matomo_db
        networks:
                - mynetwork
…
```

Docker Services - Environment Variables

- Add environment variables at **mymatomo section** into .yml file

```
....
mymatomo:
        image: matomo
        ports:
                - 80:80
        environment:
                MATOMO_DATABASE_HOST: mydatabase
                MATOMO_DATABASE_ADAPTER: mysql
                MATOMO_DATABASE_USERNAME: matomo_user
                MATOMO_DATABASE_PASSWORD: my-secret-matomo-pw
                MATOMO_DATABASE_DBNAME: matomo_db
        networks:
                - mynetwork

...
```
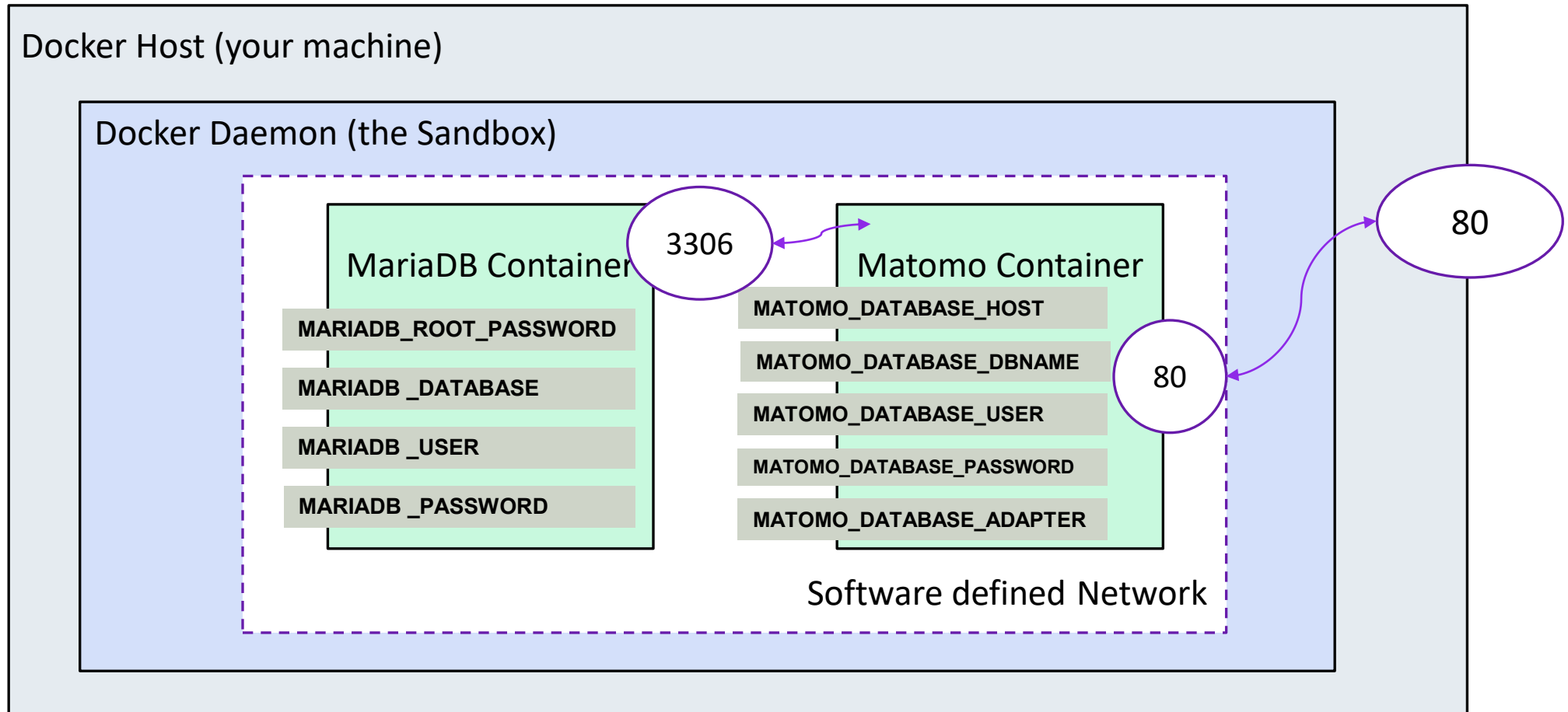
Docker Services - Environment Variables

Docker Services - Environment Variables

- Lets update our running Docker Stack:
  - **$ docker stack deploy –c stack.yml MyStack**

- Browser: localhost

- ➔ We've (nearly) successfully setup a Matomo Analytics using Docker! ☺

15min break

Docker Services - Volumes

- What happens with our data if we remove the Stack and build a new one?
  - Configure your Matomo Analytics
  - **$ docker stack rm MyStack**
  - **$ docker stack deploy –c stack.yml MyStack**

Docker Services - Volumes
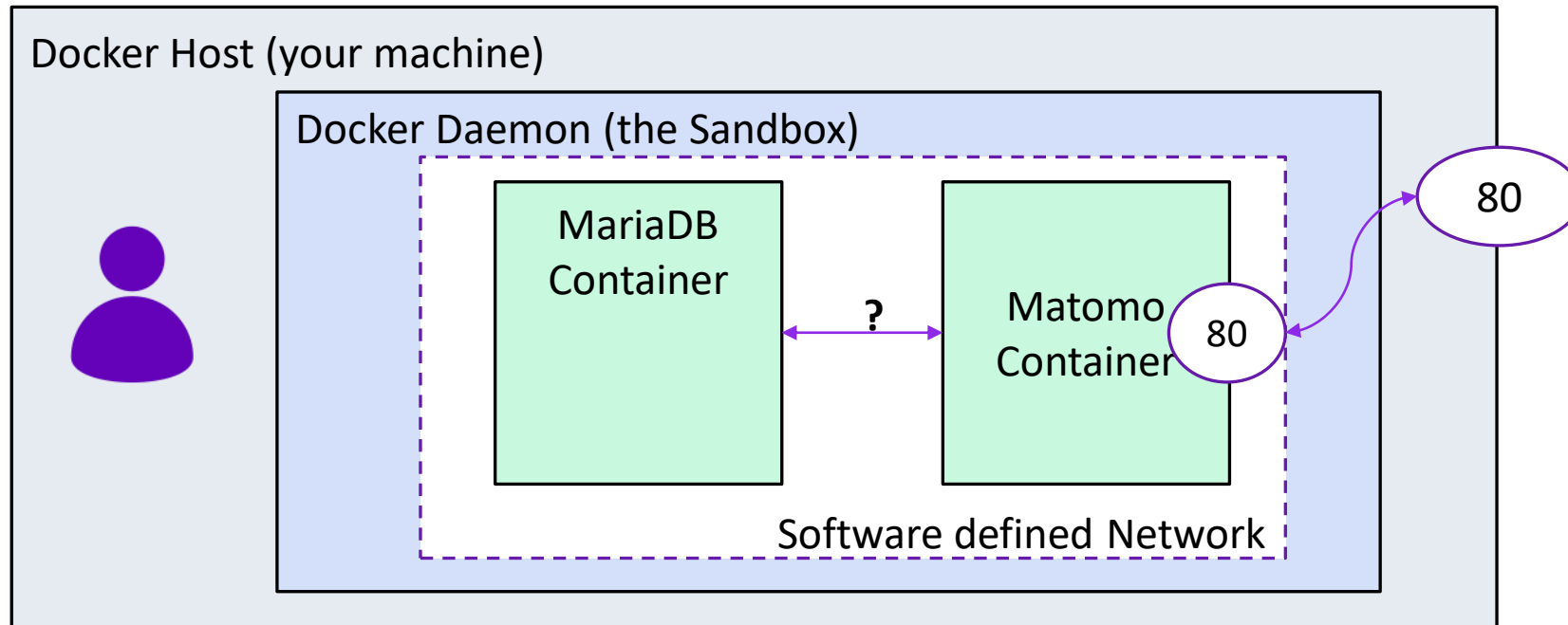
- All data is stored inside Containers
- **New Stack => New Containers => Previous Data is gone….**

- ➡ **Volumes** are used to store persistent data on the Docker Host

- Volumes **map directories** of the host into directories **inside** the container

Getting inside a Docker Container

- Get „inside" a running container: **$ docker exec -it <containerid> <command>**

Docker Services - Volumes

- Map host directory to directory inside container

```
        ….
        mydatabase:
                image: mariadb
                environment:
                        MARIADB_ROOT_PASSWORD: my-secret-root-pw
                        MARIADB_DATABASE: matomo_db
                        MARIADB_USER: matomo_user
                        MARIADB_PASSWORD: my-secret-wordpress-pw
                volumes:
                        - ./data/mariadb:/var/lib/mysql
                …
```

Docker Services - Volumes

- Redeploy our stack
- **$ docker stack deploy –c stack.yml MyStack**

Exercise 2 - Volumes

- Create a volume for **/var/www/html** in the Matomo Service

Congratulations!!!

- You have setup a Matomo Analytics Environment using Docker!!!

What we've learned so far

- How to run a Docker Container
- How to build a Docker Stack
- How to configure Docker Services
  - Ports
  - Environment Variables
  - Volumes
- How Docker Services communicate with each other in a Stack

- Everything was based on already existing matomo & mariadb images
- How do we create our own images?

Outline

Docker Architecture

Docker Images & Containers

Docker Stacks & Services

Create Images

Docker Images and Containers

- Docker Images consist of **read-only layers** of files
- Containers have a **read/write layer** on top of the read-only layers

Thin R/W layer ← Container layer

91e54dfb1179	0 B
d74508fb6632	1.895 KB
c22013c84729	194.5 KB
d3a1f33e8a5a	188.1 MB

ubuntu:15.04

Image layers (R/O)

Container
(based on ubuntu:15.04 image)
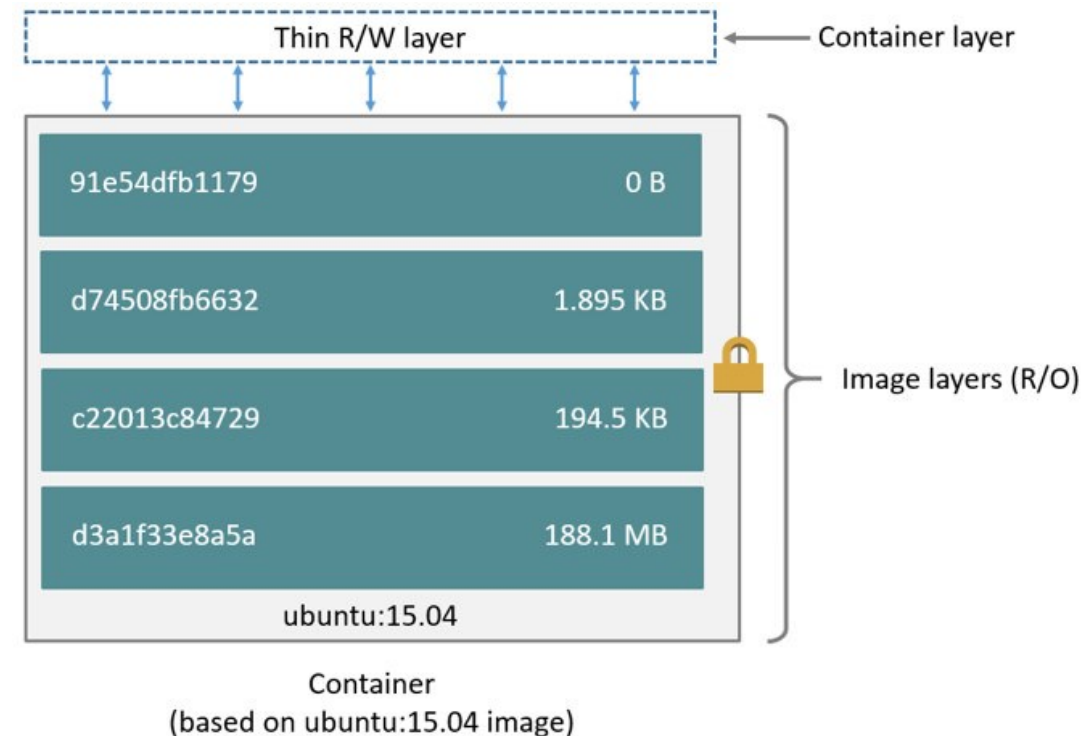
Creating Docker Images

- Docker Images are created with a Dockerfile
- Each layer in an image corresponds to a command in the Dockerfile

1. Lets create a small Express App
2. Create a Docker Image of that App

Create Express App

- $ npm install express-generator –g
- $ express myexpressapp
- $ cd myexpressapp
- $ npm install
- $ npm start

- Browser: localhost:3000

Create Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

Create Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **FROM** tells Docker which image should be the **base image** for ours

- We need an **environment** where **node** is installed.

- Tag „alpine": Alpine is a very small linux distribution

- Other tags are available: https://hub.docker.com/_/node

Create Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **WORKDIR** tells Docker in which directory subsequent calls like COPY or RUN should happen

Create Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **COPY** copies the package*.json files into our WORKDIR /app
- Why only package*.json and not the entire directory???

Create Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **RUN** executes npm install

- npm install takes a lot of time in larger projects

- However, package.json changes not too often compared to source code in our app ➔ **RUN creates a new layer in our image**

Create Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **COPY all files from directory . to the WORKDIR**

Create Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **EXPOSE port 3000 in the container**

Create Dockerfile

```
FROM node:alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

- **CMD** execute **npm** with parameter **start** when starting the container

The .dockerignore file

- Docker Images should be as small as possible. Therefore we want to ignore those files, which are not essential to run our app.

```
.git
.gitignore
node_modules
npm-debug.log
Dockerfile*
docker-compose*
README.md
LICENSE
.vscode
```

Creating Docker Images

- $ docker build .

- **Note:** Each command in the Dockerfile created a layer!

- Lets tag our image!
- Standard naming convention: dockerhubusername/imagename:version

Creating Docker Images

- $ docker build –t dockerhubusername/imagename:version .

- Push them into the Docker Hub Registry!

- $ docker login
- $ docker push dockerhubusername/imagename:version
- $ docker pull dockerhubusername/imagename:version

Congratulations!!!

- You have successfully created and pushed your first own Docker Image!!!
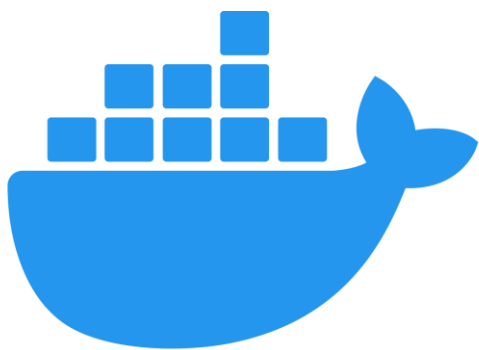
Conclusion – You've learned about…

- Docker Architecture
- Docker Images & Containers
- Docker Stacks & Services
- creating Docker Images

Conclusion – Further reading

- Have a look at the Appendix in this presentation
- Container Orchestration
  - Docker Swarm - https://docs.docker.com/engine/swarm/
  - Kubernetes - https://kubernetes.io
- Docker documentation
  - https://docs.docker.com/

Appendix - Docker Swarms

- As your awesome platform increases in popularity, so does the number of requests per second on your backend

- At a certain point your single server is going to be too weak to handle all the traffic

- You need a way to scale your platform

Docker Swarm - Scaling

- Different ways to scale:
  - Vertical Scaling: ➔ Get a faster server
  - Horizontal Scaling: ➔ Distribute work over multiple servers

- Docker Swarm helps you to easily **scale** your platform **in a horizontal way!**
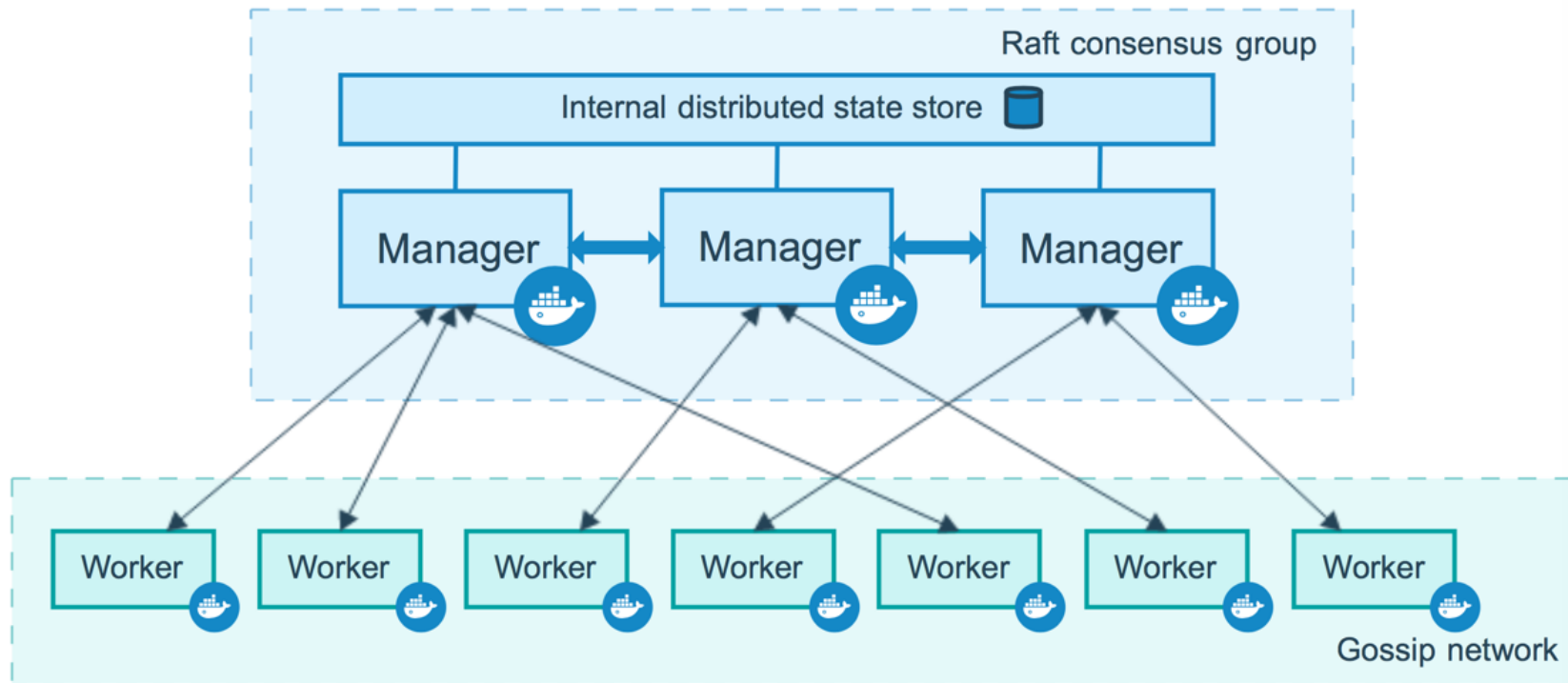
Docker Swarm - Nodes

- A Docker Swarm consists of multiple connected machines where Docker is installed
- These machines are called **Nodes**
- There are two types of **Nodes**
  - **Swarm Managers**
  - **Swarm Workers**

Docker Swarm

Docker Swarm - Nodes

- Manager Nodes are responsible for scheduling services and managing the Swarms State
- You should aim for an **odd number of Managers**
- More than 7 Manager Nodes reduce the speed of the Swarm due to communication overhead between the Managers

- Worker Nodes are pure Container executers

Docker Swarm - Nodes

- Do you remember the command: **$ docker swarm init**?
- At this point we've created a **Docker Swarm Manager Node**

```
Swarm initialized: current node (fe4uipd973z81lp2rcgjvqjrs) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3rw0mlyjv4twd8fn98sxs4ihn0k1pn4jcffdjfjavbxexu1hr0-
6pmhy2h1ul6bmll9d725i9r7r 192.168.65.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Docker Swarm - Commands

- List all Nodes in a Docker Swarm
  - **$ docker node ls**
- Get the join token for a worker node
  - **$ docker swarm join-token –q worker**
- Get the join token for a manager node
  - **$ docker swarm join-token –q manager**
- Join an existing swarm
  - **$ docker swarm join --token <token> <Swarm Managers IP Address>:2377**

Docker Swarm - Commands

- Docker Stacks can be executed on Docker Swarms!
- Define the number of replicas of each of your Services in the Stack

```
...

    deploy:

        replicas: 5

...
```

- **$ docker stack deploy –c stack.yml MyStack**

Docker Swarm - Reliability

- Docker Swarm manages load balancing between containers

- Try to stop one of your containers of MyStack

- **What happens?**

- Docker Swarm always tries to keep the number of replicas as defined in stack.yml

Appendix: Create your own registry

You can get the Docker Registry Server as Docker Image too! ☺

docker run -d -p 5000:5000 --name registry registry:2

Now you have a Docker Registry Server running at: localhost:5000

Appendix: Push image to your own registry

Tag an image to point to your registry:

docker image tag <imagename> localhost:5000/myimagename

Push the image:

docker push localhost:5000/myfirstimage

After that you can pull the image from your own registry:

docker pull localhost:5000/myfirstimage

Cheat Sheet – Docker Container

- List all running containers: **$ docker ps**

- List all containers: **$ docker ps –a**

- Stop a container: **$ docker stop <containerId or name>**

- Start a container: **$ docker start <containerId or name>**

- Start a container and attach to output: **$ docker start –a <conatinerId or name>**

- See logs: **$ docker logs <containerId or name>**

- Follow logs: **$ docker –f logs <containerId or name>**

- List all local images: **$ docker image ls**

Cheat Sheet – Docker Stack

- List all running stacks: **$ docker stack ls**

- List all services for this stack: **$ docker stack services <stackname>**

- List all networks: **$ docker network ls**

- Remove a stack: **$ docker stack rm <stackname>**

- Deploy a stack: **$ docker stack deploy –c <.yml file> <stackname>**

Cheat Sheet – Build / Push Image

- Build a Docker Image with a name
  - **$ docker build –t dockerhubusername/imagename:version .**

- Login to Docker Hub Registry
  - **$ docker login**

- Push an image to Docker Hub Registry
- **$ docker push dockerhubusername/imagename:version**

Cheat Sheet – Docker Swarm

- List all Nodes in a Docker Swarm
  - **$ docker node ls**
- Get the join token for a worker node
  - **$ docker swarm join-token –q worker**
- Get the join token for a manager node
  - **$ docker swarm join-token –q manager**
- Join an existing swarm
  - **$ docker swarm join --token <token> <Swarm Managers IP Address>:2377**