INTRODUCTION TO DOCKER

from first concepts to an app
in one hour

@nohwnd

JAKUB JAREŠ

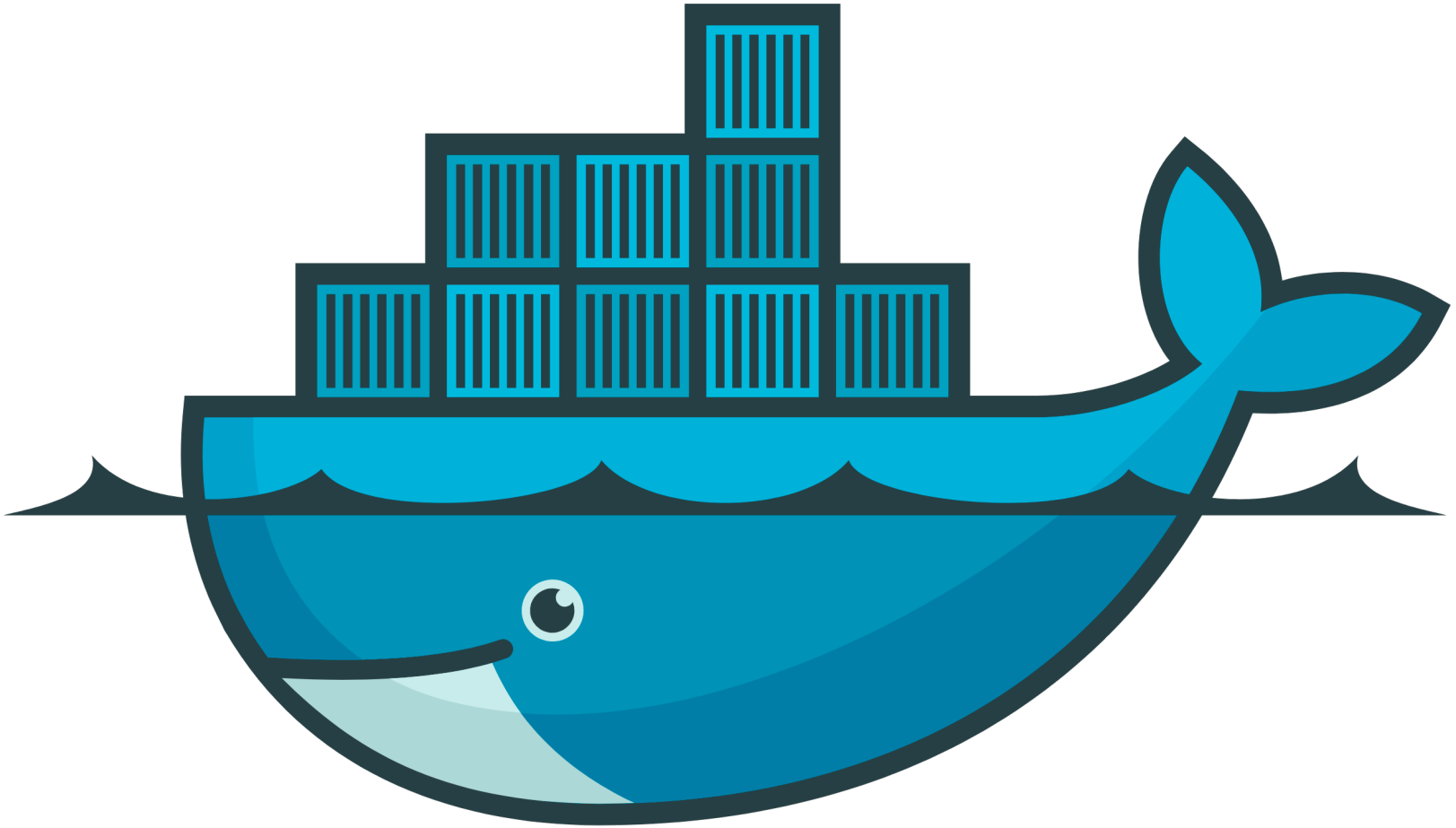
.NET

C# & F#

PowerShell

Microsoft MVP

@nohwnd



docker

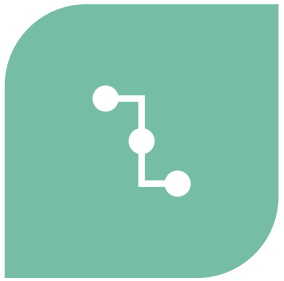
WHAT IS DOCKER?

APPLICATION DELIVERY PLATFORM

APPLICATION RUNTIME

DOCKER STENGTHS

RUN ANYWHERE

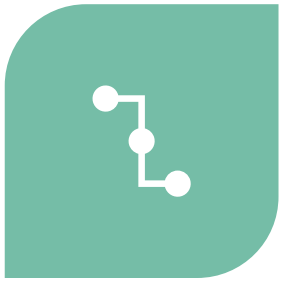


EXECUTE THE SAME WAY

DELIVER THE SAME WAY

DOCKER STENGTHS

TRY STUFF WITHOUT JUNK

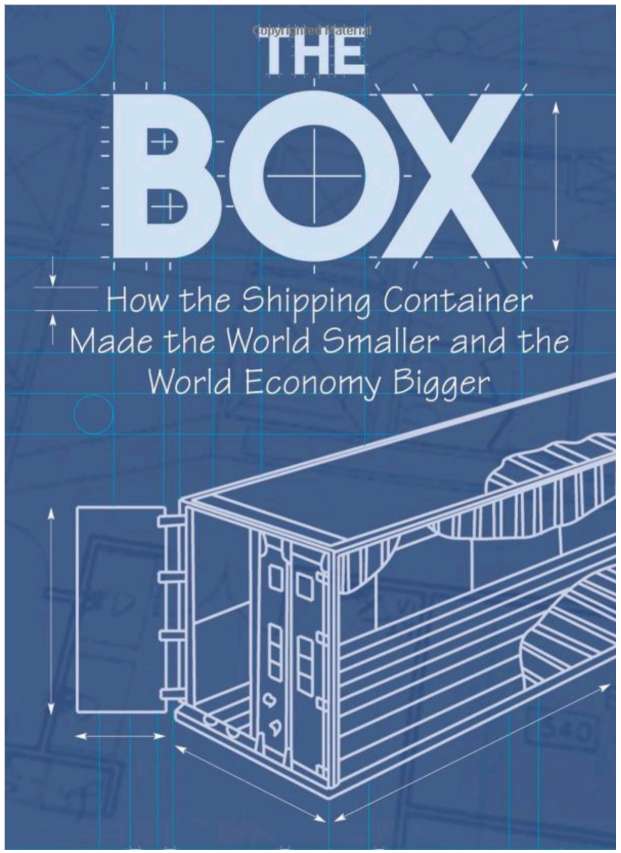


BUILDING SOFTWARE IN CI

EASY TO LEARN


STANDARDIZED!
K LINE
CAMSKIP
CAMSKIP
CAMSKIP

REAL LIFE CONTAINERS



The Box: How the Shipping Container Made the World Smaller and the World Economy Bigger

https://www.amazon.com/Box-Shipping-Container-Smaller-Economy/dp/0691136408

ALSO IT RUNS SNAKE!

```
docker run -it dyego/snake-game
```

DOCKER WEAKNESSES

NOT FOR GUI

NOT FOR DATA STORAGE

INSTALLATION

getting it into your system

INSTALLER

Docker Desktop

https://www.docker.com/products/docker-desktop

VIA A PACKAGE MANAGER

```
# windows
choco install docker-desktop

# macOS
brew install docker

# ubuntu (linux)
apt-get install docker-ce
```

CONTAINERS & IMAGES

the basic building blocks

CONTAINERS & IMAGES

Image

- contains some runtime
- contains the application files
- is made out of layers
- all layers are read-only

(like a class)

Container

- is „instance“ of and image
- gets its own part of operating system resources
- usually runs just one process

(like an instance of a class)

IMAGES INTRODUCTION DEMO

```
# pull hello world image
docker pull mcr.microsoft.com/mcr/hello-world

# show all local images
docker image list

# show layers of the image
docker history mcr.microsoft.com/mcr/hello-world
```

CONTAINERS INTRODUCTION DEMO

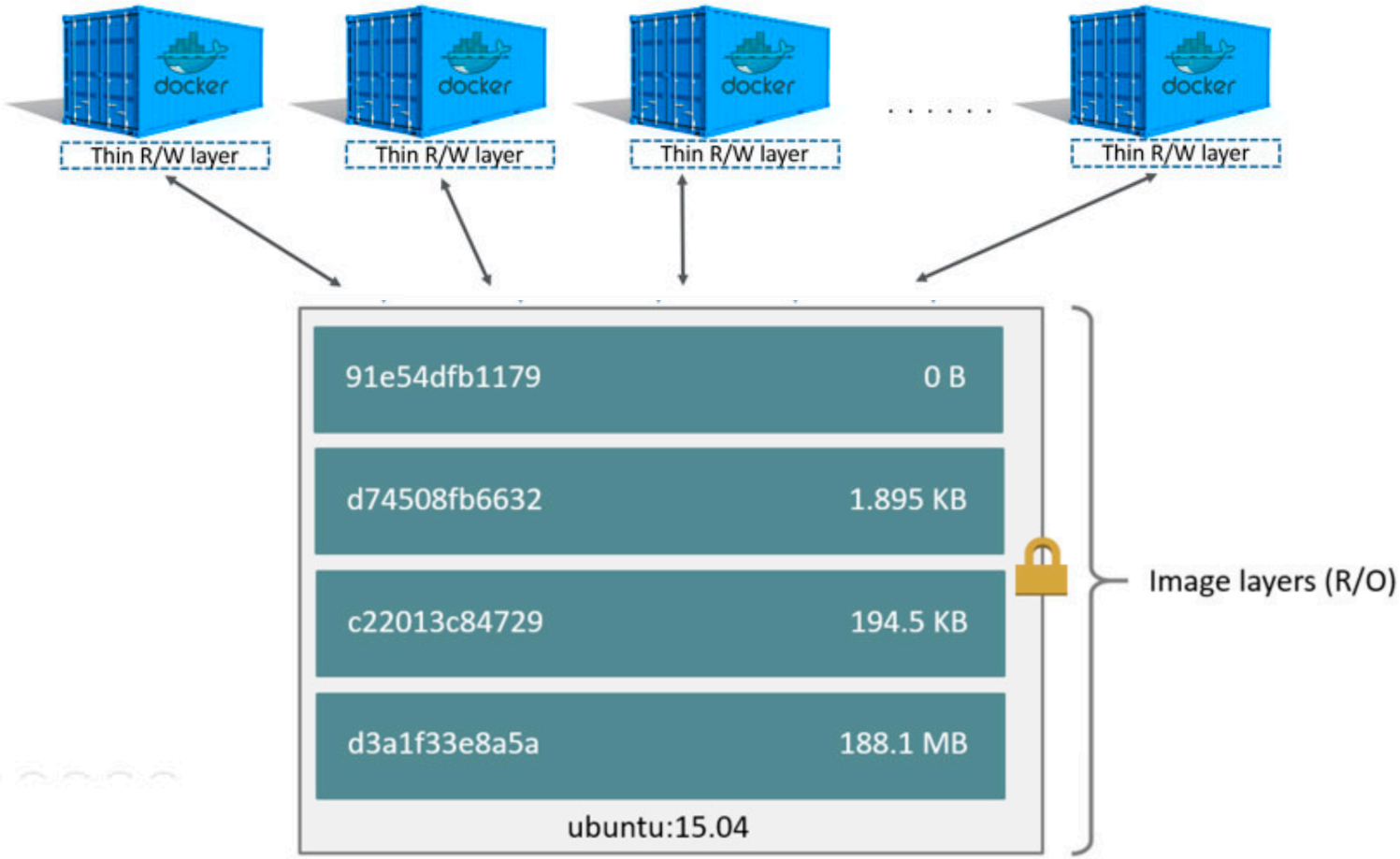
```
# run a new container
docker run mcr.microsoft.com/mcr/hello-world

# list all containers
docker container list –all

# clean up
docker container prune
```

LAYERS YOU SAY???

PULLING UBUNTU IMAGE

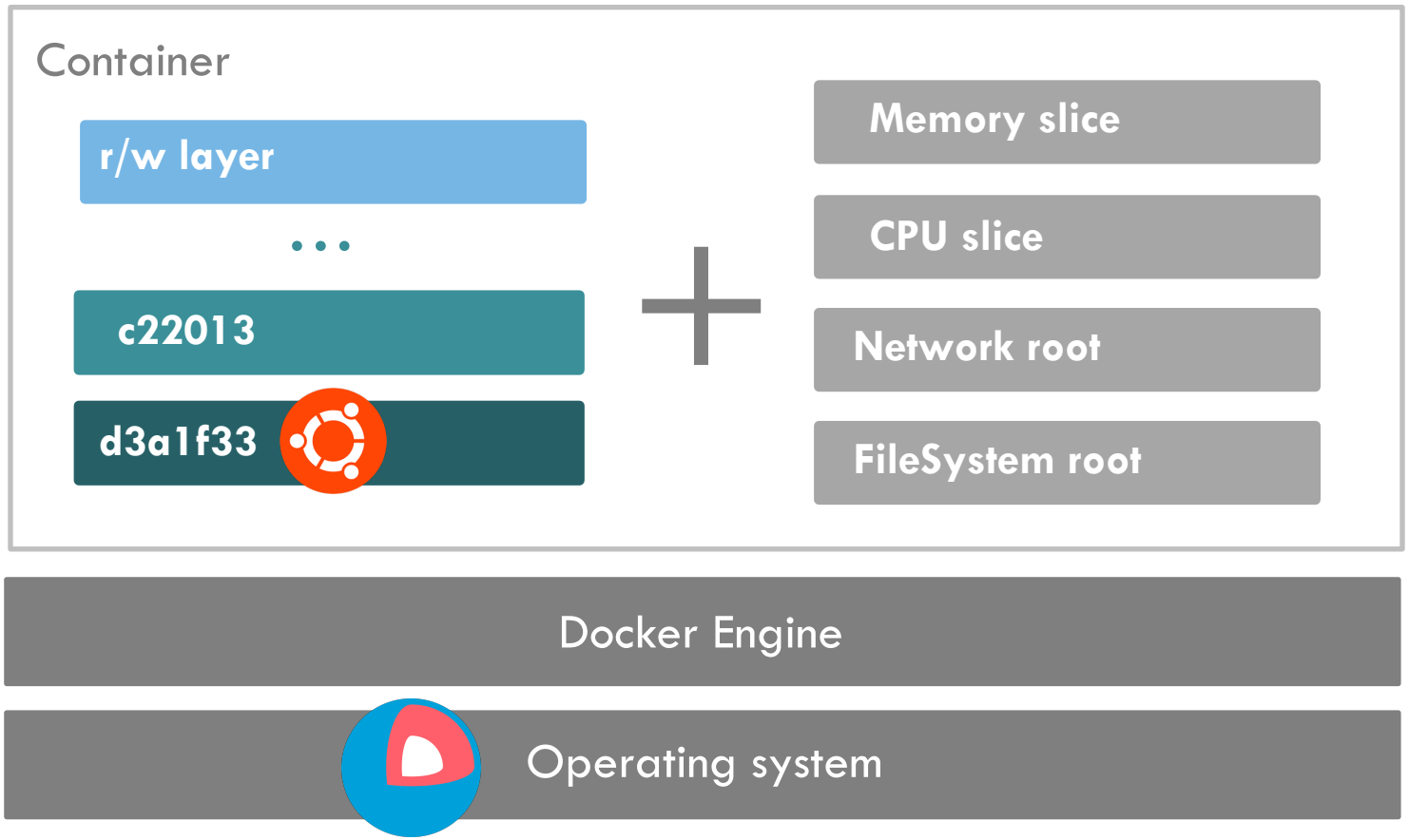
```
docker pull ubuntu:15.04
15.04: Pulling from library/ubuntu
9502adfba7f1: Pull complete
4332ffb06e4b: Pull complete
2f937cc07b5f: Pull complete
a3ed95caeb02: Pull complete
Digest:
sha256:2fb27e433b3ecccea2a14e794875b086711
Status: Downloaded image for ubuntu:15.04
```

CONTAINERS ARE R/W LAYERS



RESOLVING FILES

CONTAINER


Container
r/w layer
c22013
d3a1f33
Memory slice
CPU slice
Network root
FileSystem root
Docker Engine
Operating system

OPERATING SYSTEM

VS

BUILDING IMAGES

from scratch

DOCKERFILE INSTRUCTIONS

FROM

CMD

EXPOSE

WORKDIR

etc.

reference. https://docs.docker.com/engine/reference/builder/

OUR FIRST DOCKERFILE

```
# in dockerfile
FROM busybox
CMD echo "hello, world"

cd ~/Projects/docker/busybox-hello-world
build -t busybox-hello-world -f dockerfile .

docker run busybox-hello-world
```

BUILD IT ONE MORE TIME

```
docker build -t busybox-hello-world .

 Step 1/2 : FROM busybox
  ---> af2f74c517aa
 Step 2/2 : CMD echo "hello, world"
  ---> Using cache
  ---> e95a44c89276
 Successfully built e95a44c89276
 Successfully tagged busybox-hello-world:latest
```

BEST PRACTICE

Best practices for writing Dockerfiles

Estimated reading time: 30 minutes

This document covers recommended best practices and methods for building efficient image

Docker builds images automatically by reading the instructions from a `Dockerfile` -- a text needed to build a given image. A `Dockerfile` adheres to a specific format and set of instruc reference.

A Docker image consists of read-only layers each of which represents a Dockerfile instruction delta of the changes from the previous layer. Consider this `Dockerfile` :

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

RUNNING CONTAINERS

RUN OPTIONS

```
# remove after run
docker run --rm busybox-hello-world

# run in background
docker run -d mongo

# run interactively (exit with  Ctrl + P + Q)
docker run -i -t busybox
```

INTERACTIVE RUN DEMO

```
docker run --rm –it busybox
ls
echo “hello world”


Ctrl + P + Q
docker container list
docker container stop <hash>
```

MOUNTING VOLUMES

for your data

DON'T STORE DATA IN CONTAINERS!

MOUNT VOLUME

```
docker run -v <local path>:<container path> busybox

docker run -v /data:/mnt/data busybox
```

PERSISTING DATA DEMO

```
docker run --rm -v ~/docker/volumes/hello-world:/mnt/data -it busybox

cd /mnt/data

ls

echo “hello world” > file.txt

cat file.txt

exit


# repeat ...
```

PUBLISHING PORTS

443 → 5001

PUBLISH PORT

```
docker run -p <local port>:<container port> nginx
```

```
docker run -p 37685:80 nginx
```

EXPOSED VS PUBLISHED

```
# expose says what ports are available to publish
EXPOSE 443
EXPOSE 80

# -p selects the port to publish
docker run -p 80:80 -p 443:443 nginx

# -P publishes all exposed ports high-port numbers
docker run -P nginx
```

INTER-CONTAINER COMMUNICATION DEMO

- all conainers see all exposed ports

```
docker run --name ngx nginx
docker inspect ngx

docker run byrnedo/alpine-curl <ngx ip address>
```

TAGGING IMAGES

:latest

TAGGING

- used to mark images
- one image can have multiple tags
- allows consumers to pin to a given version
- newest available version is called "latest"
- build once, test, publish

```
docker tag <registry> <image>
docker tag <registry:tag> <registry:tag>
```

TAGGING AS A VERSIONING SYSTEM

```
:latest # give me all updates
:1 # give me only minor updates and patches
:1.0 # give me only patches
:1.0.1 # give me only rebuilds
:1.0.1-build128 # pin to exact build

# demo in few slides
```

IMAGE REGISTRY

nohwnd/webapi

IMAGE NAME

- image name is registry + path + name

```
# library docker hub image
nginx
# "custom" docker hub image
dyego/snake-game
# own registry
mcr.microsoft.com/mcr/hello-world
# self-hosted registry
localhost:5000/team1/product1/app1/component1
```

FULL TAGGING IMAGE DEMO

```
# tag as latest for my registry

docker image tag busybox-hello-world
registry.company.com/busybox-hello-world:latest


# tag as 1.0 for my registry

docker image tag busybox-hello-world
registry.company.com/busybox-hello-world:1.0


# then push

docker push -u <user> -p <pass>
registry.company.com/busybox-hello-world:1.0
```

DOCKER-COMPOSE

to compose apps

DOCKER-COMPOSE

- tool and file format for defining multicontainer applications

- allows composition, builds, scaling, overrides, parametrization, converting to kubernetes manifests etc. etc.

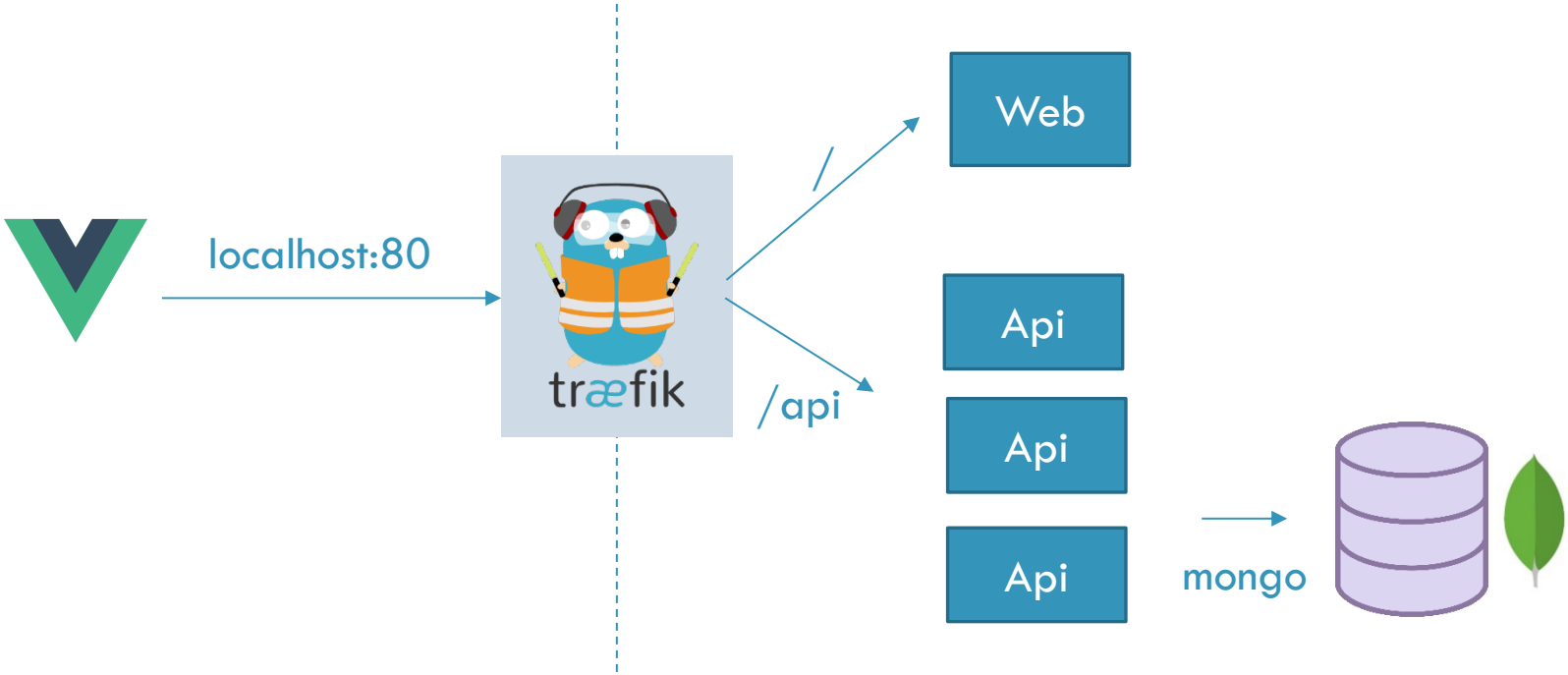
- I will show only composition and scaling

DOCKER-COMPOSE

- great for sharing the same setup among developers

-serves as a basic deployment

- easy to use with Traefik reverse proxy




```
docker-compose --file docker-compose-traefik.yml
```

DEMO

SUMMARY

if you are not asleep already

CLOSING THOUGHTS

- Docker is awesome for web app development

- Docker is awesome for CI pipeline (especially if you maintain multiple versions of build environment)

- Docker is great for trying out stuff without polluting your system

-

NEXT STEPS

- try it on your own, start a small project?

- watch few courses (youtube, pluralsight comes with your MSDN if you have any)

- Read free Microsoft e-book https://docs.microsoft.com/cs-cz/dotnet/standard/microservices-architecture/

- learn about Kubernetes ☸

Gitlab + Kubernetes = ❤️

Azure bootcamp

https://azurebootcamp.cz

27-04-2019, Microsoft

(🇨🇿 only, sorry 😔)

THAT'S IT!

@nohwnd

Q & A?