Requirements

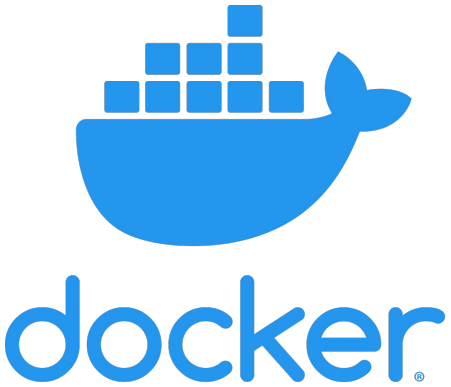
- An account in docker hub (hub.docker.com)
- Access to https://labs.play-with-docker.com/ (Our exercises will be from here)

Links

- https://github.com/DhrubajitPC/znk-docker-nightclazz-crud
- https://www.menti.com/cnz57s6di2
- https://docs.docker.com/
- https://bit.ly/intro-containers
- http://dockerlabs.collabnix.com/docker/cheatsheet/
- https://dzone.com/articles/evolution-of-linux-containers-future


zenika
<animés par la passion>
CODING
THE WORLD

Docker: The Foundation of Modern Application Development



Objectives

- Basic **Understanding of Containers** and why are they so common
- Able to **create** your own images and **run** containers
- Able to manage different services using **docker-compose**
- **Reduce learning curve** to be able to **kickstart** docker based development tomorrow
- **Refresher** on docker fundamentals
- Docker is **simple**!

Structure

- Basic Understanding of Containers
- Exploring Docker Images and Containers
- Hands-on #1
- Building custom images with Dockerfile
- Hands-on #2
- Docker networking and volumes
- Simplifying with docker-compose
- Hands-on #3
- A glimpse of docker under the hood

Requirements

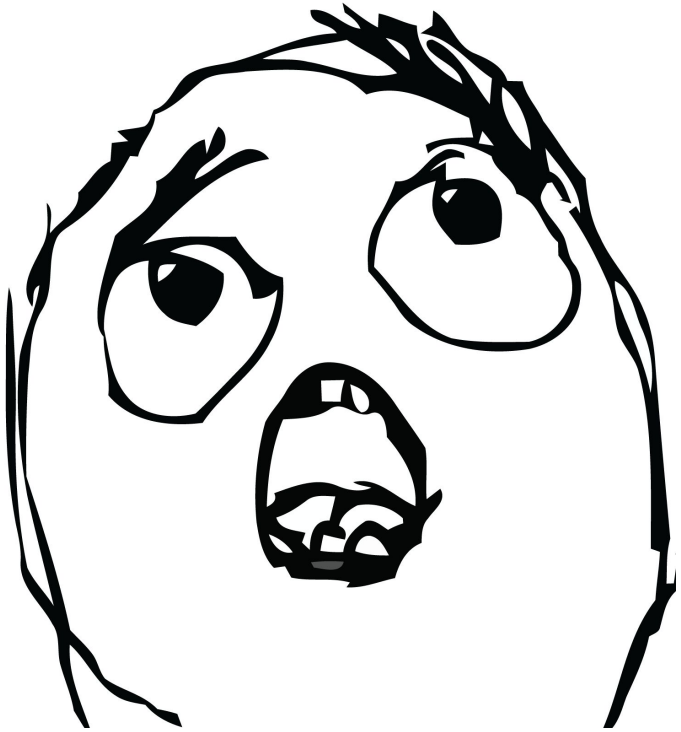
- An account in docker hub (hub.docker.com)
- Access to https://labs.play-with-docker.com/ (Our exercises will be from here)

Links

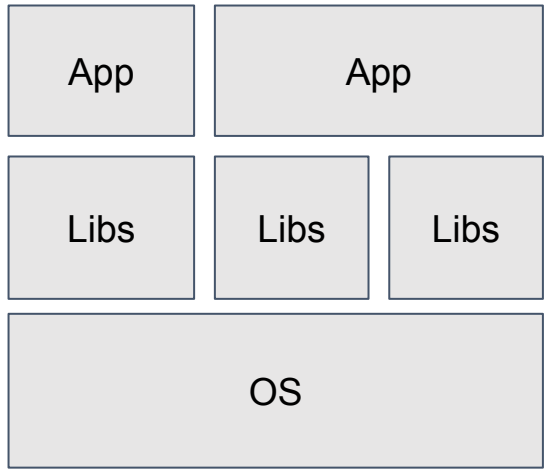
- https://github.com/DhrubajitPC/znk-docker-nightclazz-crud
- https://www.menti.com/cnz57s6di2
- https://docs.docker.com/
- https://bit.ly/intro-containers
- http://dockerlabs.collabnix.com/docker/cheatsheet/
- https://dzone.com/articles/evolution-of-linux-containers-future

History

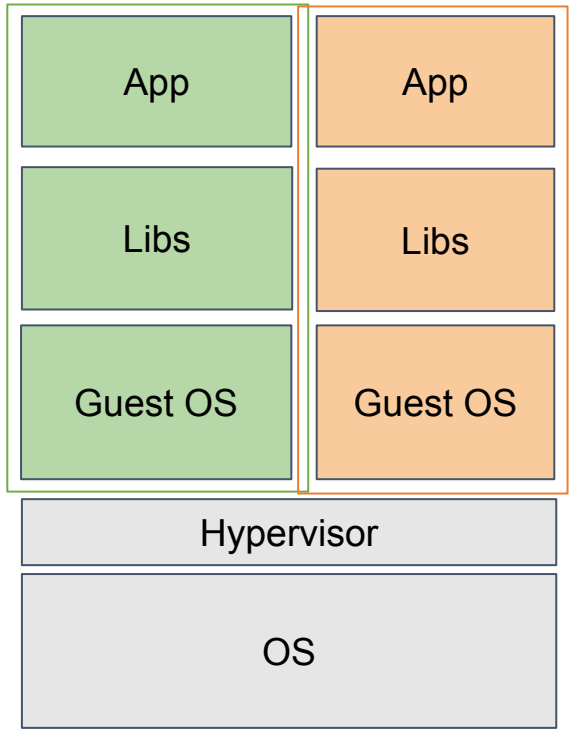
- 1979 — chroot (unix v7)
- 2000 — FreeBSD Jails
- 2001 — Linux VServer
- 2004 — Solaris Containers
- 2005 — OpenVZ
- 2006 — Process Containers
- 2007 — Control Groups
- 2008 — LXC
- 2011 — Warden
- 2013 — LMCTFY
- 2013 — Docker
- 2014 — Rocket
- 2016 — Windows Containers



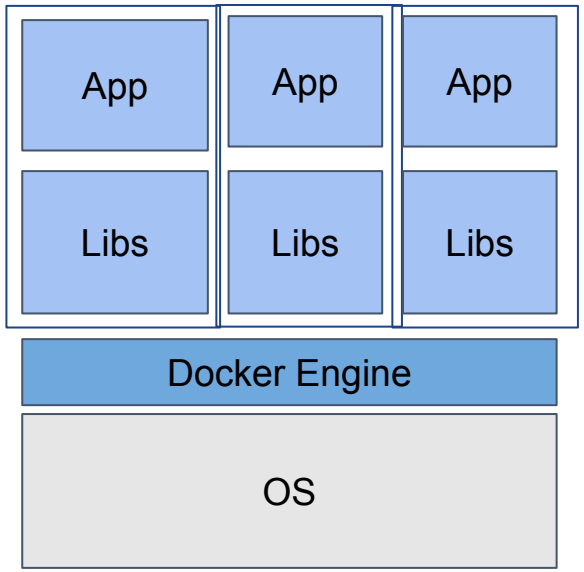
Why Containers?



Bare Metal



VM

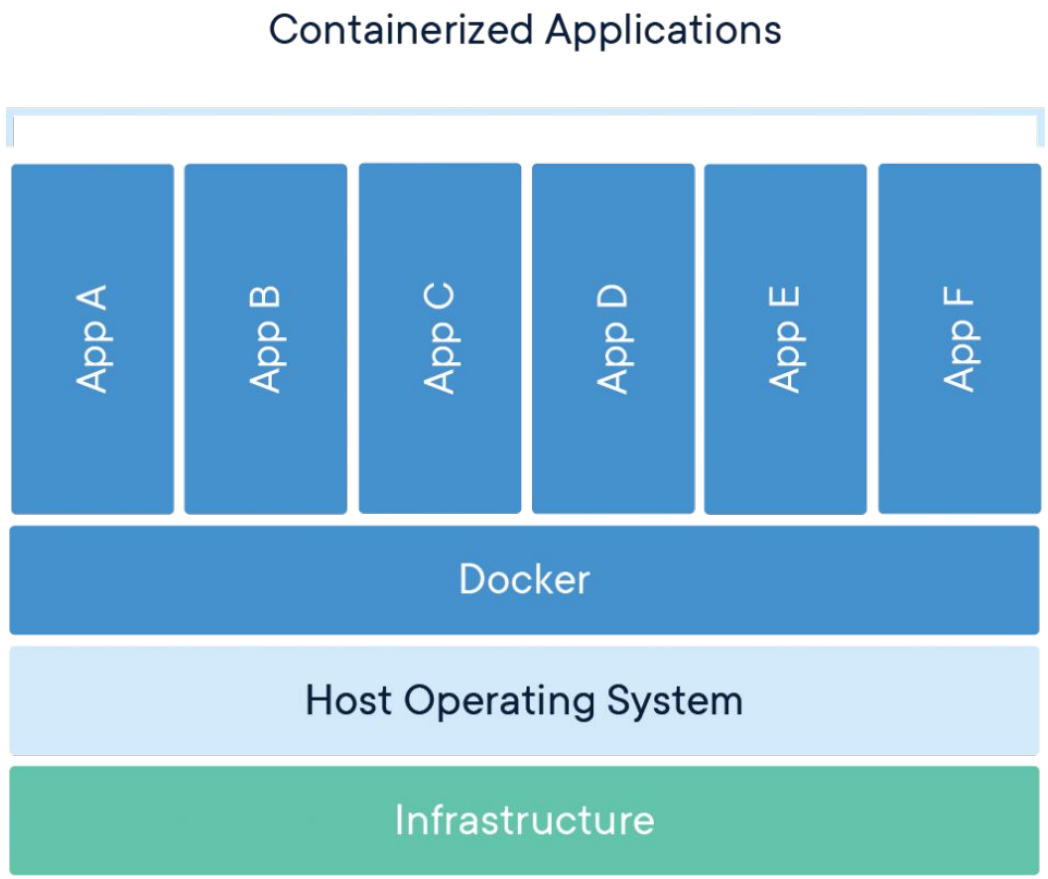


Container

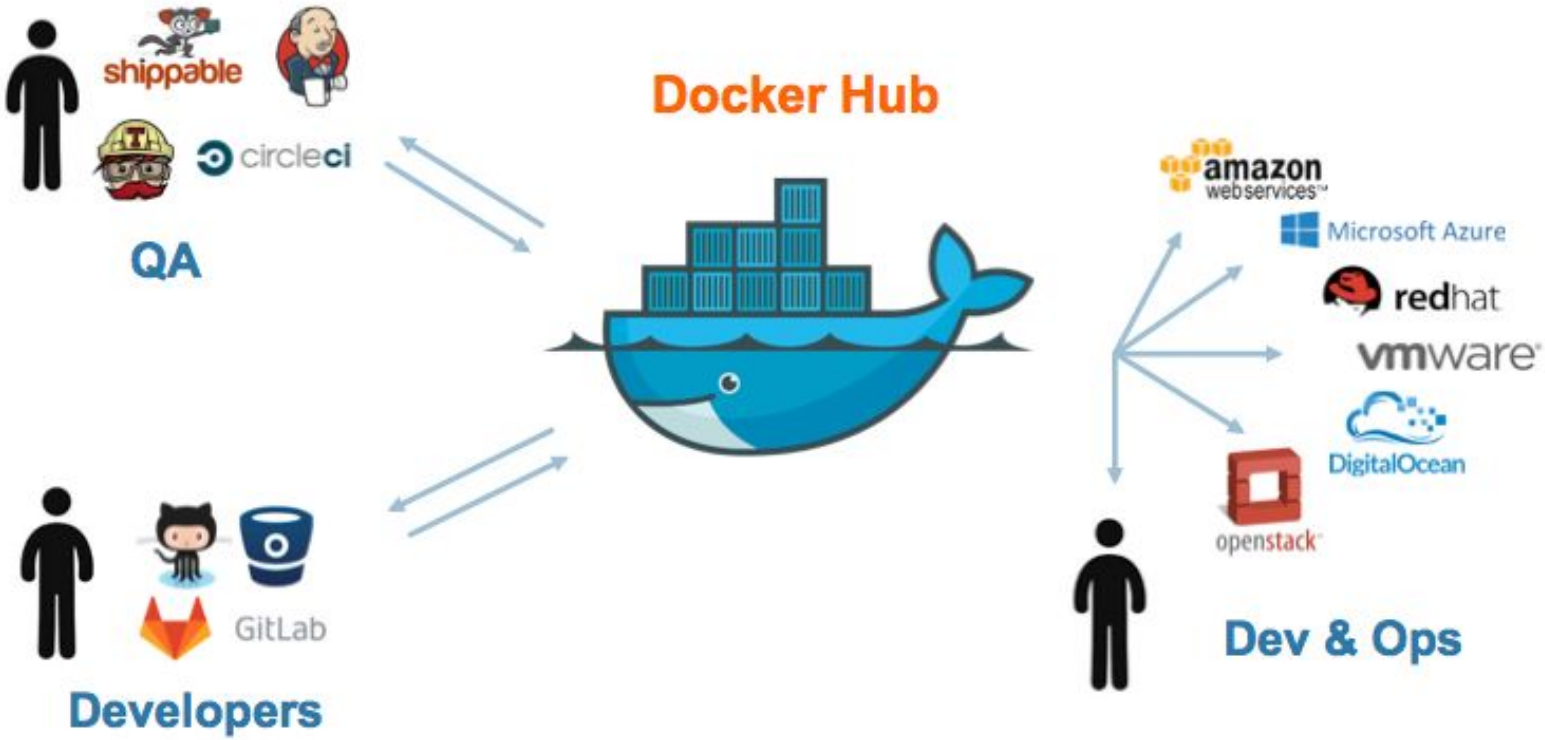
What is a Docker Image?

- A Docker **Image** is a **standard unit** of software
  - contains all the dependencies and packages required to run the application
- **Lightweight**
  - shares the machine os kernel
  - doesn't require a new os per application (no need to run a VM per application)
- **Secure** and self contained
- An image running on the docker engine during runtime creates a **container**
- **Docker hub** provides a list of images for us to use

What is a Docker Image?



What is a Docker Image?



Basics

```
// Images
docker pull <imagename>
docker images
docker rm <imagename>
docker prune images <--flag>
docker image build --tag <imagename> <folder with Dockerfile>

// Containers
docker create -d --name=<containername> -p=<hostPort:containerPort> <imagename>
<entrypoint>
docker run -it --name=<containername> <imagename> <entrypoint>
docker ps
docker ps -a
docker container rm <containername | containerhash>
```

Hands-on #1 - Getting familiar

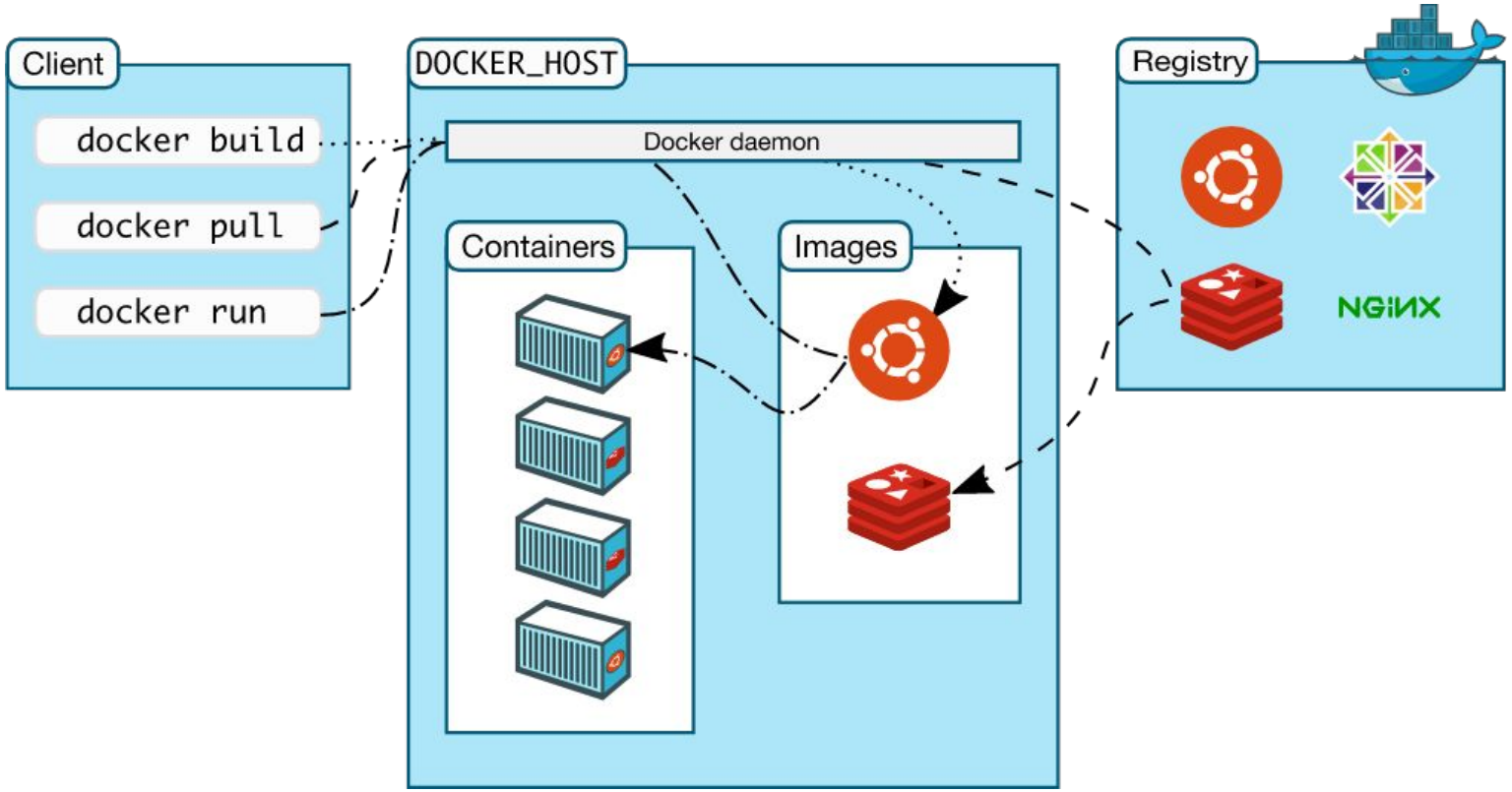
- List number of images
- pull the ubuntu image, pull the mongo image
- list the images and see their information
- create a new new mongo container
  - make sure u bind port 27017 of container to host
- curl http://localhost:27017 and you should see a response from mongo
- stop and remove the container (keep the image)

Hands-on #1 - Getting familiar (TIL)

- Using the docker CLI
- How to pull images from the remote hub
- List images
- List containers
- Creating a new container
- Stopping and removing containers
- Removing images

Docker architecture

- The Docker **client** talks to the Docker **daemon**, which does the heavy lifting of building, running, and **distributing** your Docker containers



Custom Docker Image

- Possible to build our own images using the docker image build command
- Run in a folder with a Dockerfile

```
// Building image
docker image build -t <imagename> <path to folder with Dockerfile>
```

Dockerfile

set of **instructions** on how to build an image

command	description
FROM *image*|scratch	base image for the build
MAINTAINER *email*	name of the mainainer (metadata)
COPY *path dst*	copy *path* from the context into the container at location *dst*
ADD *src dst*	same as COPY but untar archives and accepts http urls
RUN *args...*	run an arbitrary command inside the container
USER *name*	set the default username
WORKDIR *path*	set the default working directory
CMD *args...*	set the default command
ENV *name value*	set an environment variable

http://dockerlabs.collabnix.com/docker/cheatsheet/

Dockerfile - example

```
// Dockerfile
FROM node:latest

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm ci
COPY . .

CMD node app.js
```

Hands-on #2 - Build It

- Let’s create a folder called custom-img
- run the cmd ‘cat hello world > helloworld.txt’
- this will create a helloworld.txt with the content ‘hello world’ inside
- create a Dockerfile here such that when the image is built and the container is run, it ‘echo’s the contents of helloworld.txt to the shell

Hands-on #2 - Build It (TIL)

- Writing custom dockerfile
- Building custom image

Network and volume

- Completely Isolated containers are not very useful
- Need to interact between containers and the host
- Storing data for containers running a db instance

Network and volume

```
> mkdir /var/quote-db
> docker run -d --name=db -v /var/quote-db/:/data/db -p 27017:27017 mongo
> git clone https://github.com/DhrubajitPC/znk-docker-nightclazz-crud.git
> vim Dockerfile
> docker image build -t server .
> docker run -d --rm --net=host server
```

Dockerfile for server

```
// Dockerfile
FROM node:13.14.0

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm ci

COPY . .

CMD npm start
```

Docker Compose

- Allows us to start multiple services at the same time
- Sets up internal network with dns resolution based on service name
- Easy to maintain

Docker Compose - structure

```
// docker-compose.yml
version: "3.3"

services:
 service1:
    image: imagename
    build:
      context: <path to dockerfile>
    depends_on:
      - service2
    environment:
      MONGO_CONNECTION_STRING: mongodb://db:27017

  service2:
    image: "mongo"
    volumes:
      - type: bind
        source: <source path>
        target: <container path>
```

Hands-on #3 - Compose It

- cd into the root of the repo znk-docker-nightclazz-crud
- write your own compose file that starts the database and the server at the same time
- run docker-compose up -d
- test it using postman

Hands-on #3 - Compose It (TIL)

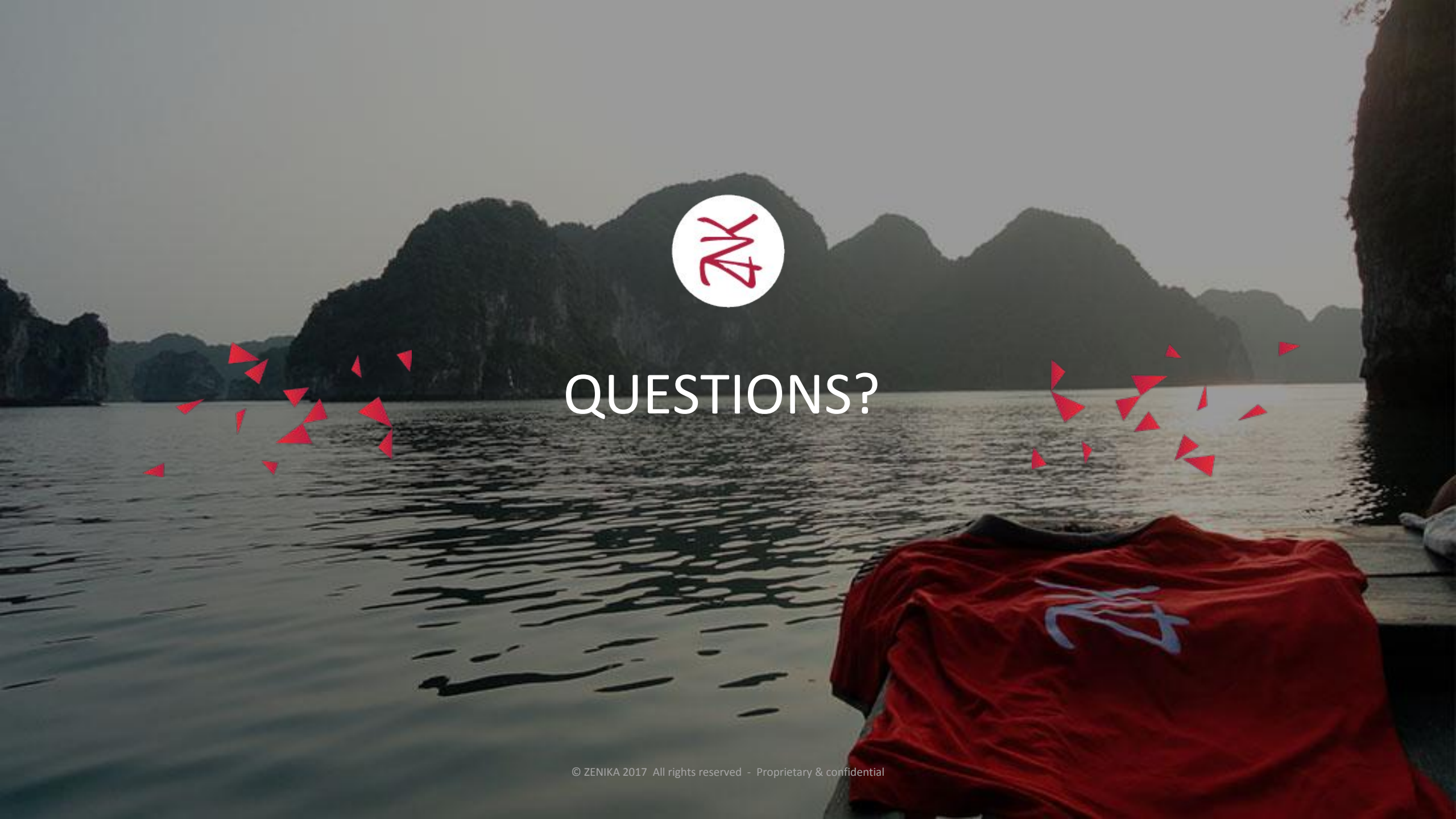
- Writing our own compose file
- Using docker compose to manage multiple services

Exploring further

- ➢ git checkout .
- ➢ git checkout feat/dockerfile
- ➢ docker-compose up -d

Under the hood

- Docker works mainly by using 3 basic linux features: **chroot**, **namespaces** and **cgroups**
- chroot: (change root) is a linux command that changes the root folder.
    - Applications inside this folder **cannot** access anything outside this folder (this being the root)
- namespaces
    - isolates applications even further by **restricting** access to other **process** from inside the chrooted environment
    - **restrict** capabilities of containers to **interfering** with other containers
- cgroups: (control groups) allows **more** control over the **physical resources** of the server
    - isolated environments only has access to provided resources
    - **prevents** one environment from **hogging** all the resource



QUESTIONS?

Training Registration

4 Days - https://bit.ly/ZENCOVIDJAVA (Java)

3 Days - https://bit.ly/ZENCOVID3D (NodeJS, Kafka, Kubernetes, React, Angular, Vue)

2 Days - https://bit.ly/ZENCOVID2D (Scrum, Ethereum, GCP, PWA, JS, Typescript)

1 Day - https://bit.ly/ZENCOVID1D (Big Data for All)

Contact

dhrubajit.chowdhury@zenika.com

https://www.linkedin.com/in/dhrubajitchowdhury

@Dhrubajit

@DhrubajitPC

< CODING THE WORLD />

<with> zenika

LAH