Before we start

- Install Docker Desktop
- Download code from https://github.com/baptistepattyn/dockerk8sworkshop

Containers what, how and why?

Workshop about Docker and Kubernetes.

Who am I?

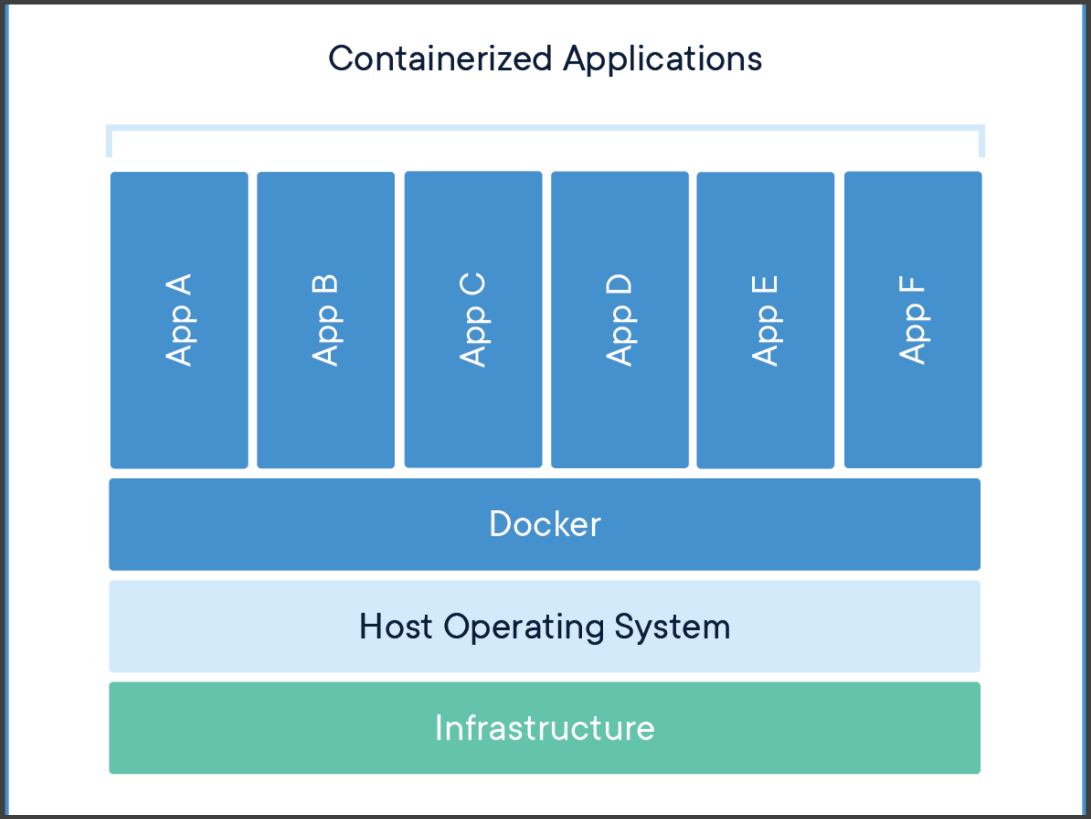
- Graduated in 2020 from KU Leuven
- Working @Skyline Communications
- Cloud Developer for 2 years
- Completed CKAD exam
- Now a Product Owner

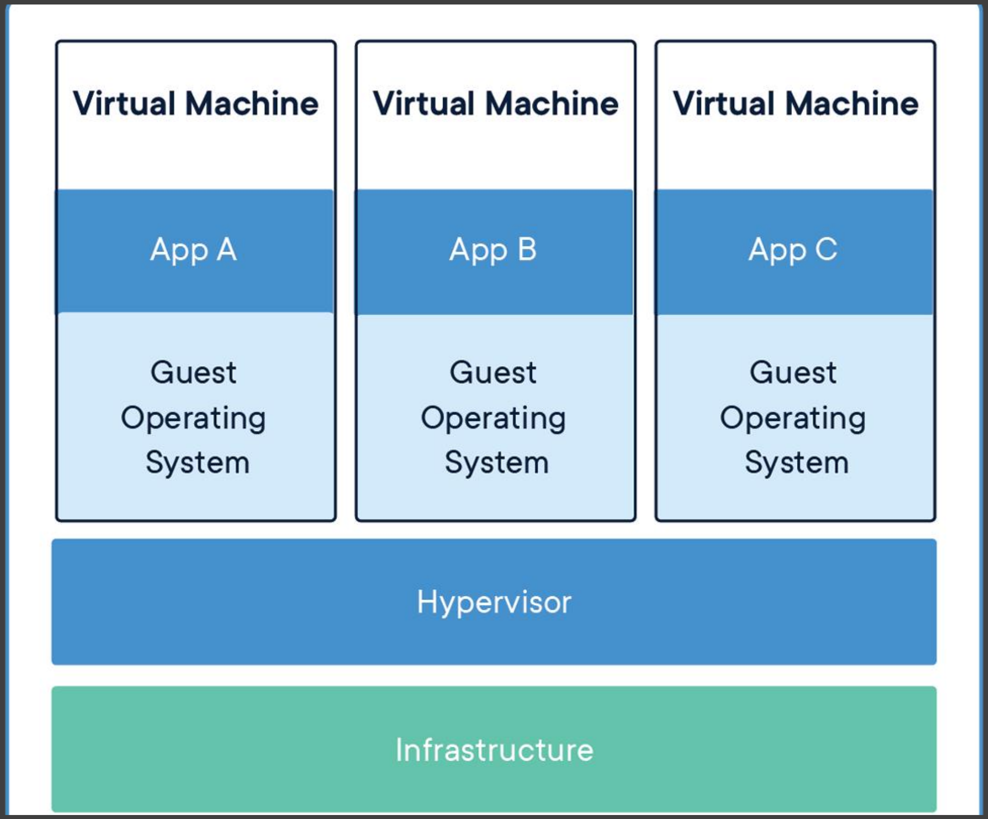
Overview

- Containers vs Virtual Machines
- Docker
- Kubernetes

Software deployment

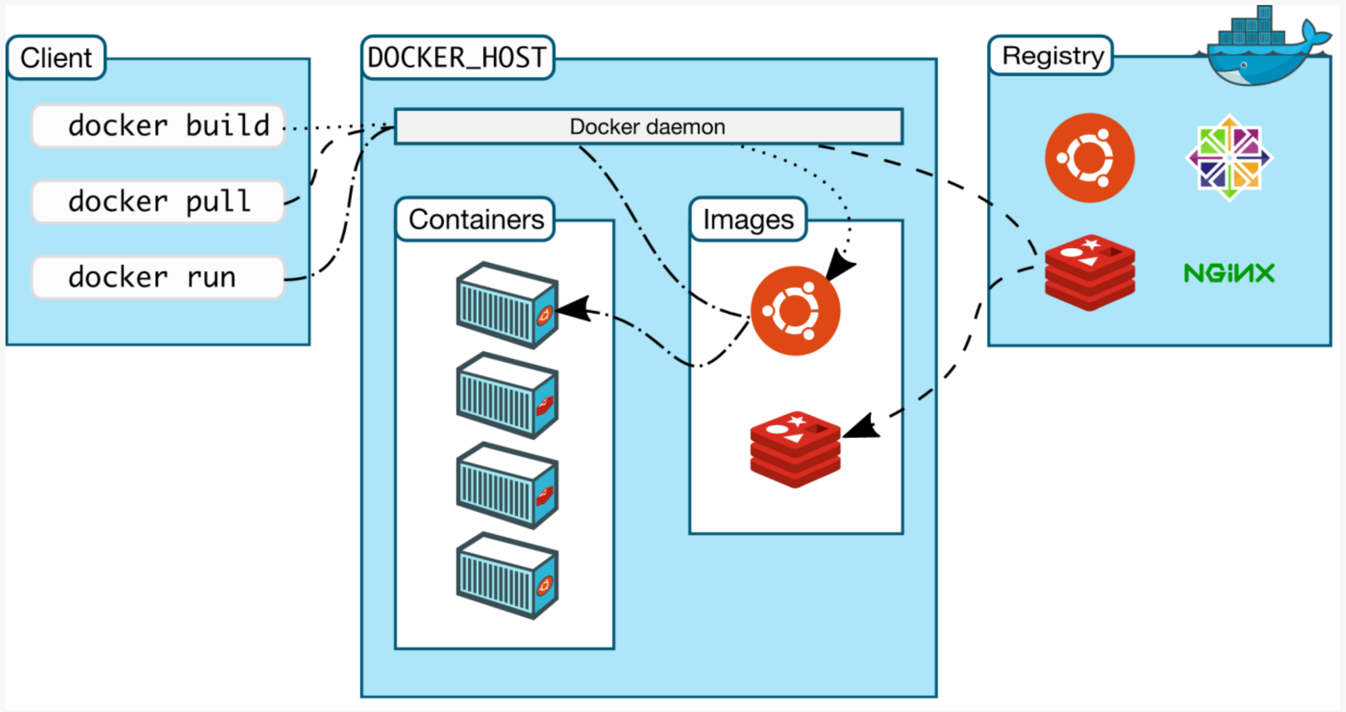
- How to deploy?
  - Service on bare metal server
  - Virtual Machine
  - Containers





Containers vs Virtual Machines

Docker Architecture



Basic commands

- **docker build**

  –t <name>:<tag>

  -f <path to Dockerfile>

  <path>
- **docker images**
- **docker run**

  --name <container name>

  - d

  - p <host port>:<container port>

  <image name>

Basic commands

- **docker container ls** *-a*
- **docker container stop** *<ID>*
- **docker tag**

  <source image>:<tag>

  <target image>:<tag>
- **docker push** *<image>*

Example Dockerfile

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Advanced Dockerfile

```
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
WORKDIR /app

COPY /Starship.Web/*.csproj ./
RUN dotnet restore


COPY . ./
RUN dotnet publish Starship.Web -c Release -o out

FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet",“Starship.Web.dll"]
```

Run Docker images

Run busybox image

- docker run hello-world
- docker run busybox
- docker container ls (-a)
- docker run busybox echo “hello from busybox”
- docker run -it busybox
  - ls
  - uptime

Modify and save running container

- docker run -it busybox sh
  - mkdir workshop
  - touch test.txt
  - exit
- docker container ls -a
- docker commit <container id> custombusybox
- docker images
- docker run -it custombusybox
  - cd workshop
  - ls

Deploy a basic container with static HTML

Create and run webserver

- cd 1
- docker build -t web-server:v1 .
- docker images
- docker run --name webserver1 -d -p 80:80 web-server:v1
- docker container ls

Surf to localhost in browser or try

```
curl localhost
```

Create and run webserver

- cd ../2
- docker build -t web-server:v2 .
- docker run --name webserver2 -d -p 81:80 web-server:v2
- docker container ls

Surf to localhost:81 in browser or try

```
curl localhost:81
```

Modify container

- docker exec -u 0 -it webserver1 sh
- apk update
- apk add nano
- cd usr/share/nginx/html
- nano index.html
  - ctrl + o (save)
  - ctrl + x (exit)
- exit

Create custom image and run it

- docker container ls

  Get container id

- docker commit <id> web-server:v3
- docker run --name webserver3 -d -p 82:80 web-server:v3

Surf to localhost:82 in browser or try

```
curl localhost:82
```

Clean setup

```
../cleandocker.ps1
```

Kubernetes

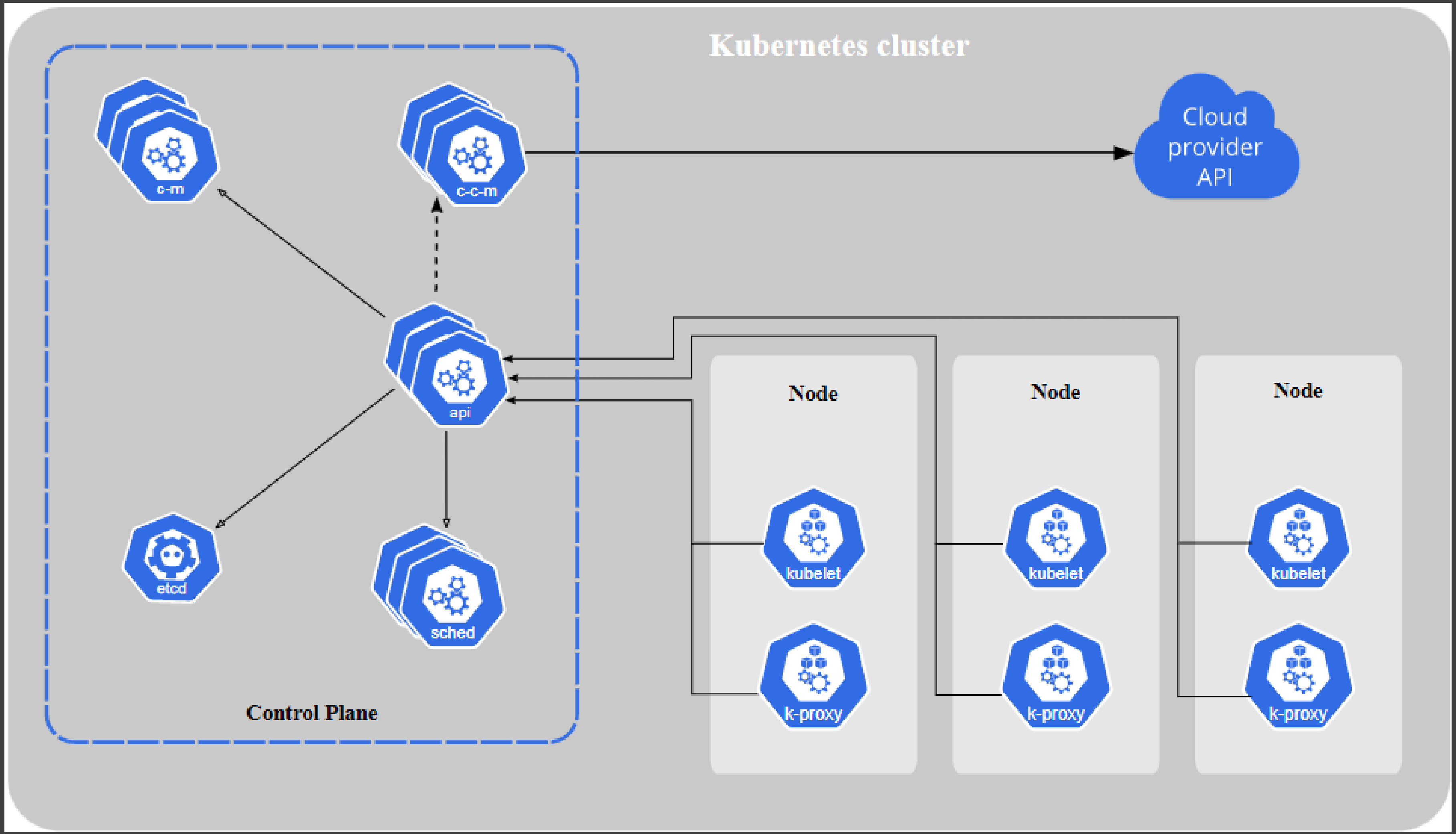
Kubernetes

- Uses containers
- Framework to run distributed system resiliently
- Provides
  - Scaling
  - Failure
  - Deployment patterns
  - …

Features

- Service discovery
- Load balancing
- Storage orchestration
- Automated rollout and rollbacks
- Self healing
- Secret and configuration management

Cluster Architecture



Basic Concepts

Describing Kubernetes objects

- Object is defined by object spec
- Describes the desired state
- Contains basic information
  - Name
  - Labels
- Used to create the object
  - kubectl: yaml
  - API request: json

Example yaml file

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Pods

- Smallest unit
- Group of containers with shared context

  Mostly only 1 container per pod in production setups

Probes

- Liveness Probe
    - Check if the container is healthy
- Readiness Probe
    - Indicates if the pod is ready to accept traffic
- Startup Probe
    - Indicates if the application in the container has started

Liveness Probe

- Probe failure will restart the container
- Can be tweaked with restart policy
- Default is success

Liveness Probe

```yaml
apiVersion: v1
kind: Pod
metadata:
 labels:
  test: liveness
 name: liveness-exec
spec:
 containers:
 - name: liveness
  image: k8s.gcr.io/busybox
  args:
  - /bin/sh
  - -c
  - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
  livenessProbe:
   exec:
    command:
    - cat
    - /tmp/healthy
   initialDelaySeconds: 5
   periodSeconds: 5
```

Readiness Probe

- Different kinds
  - Exec
  - httpGet
  - tcpSocket
- Default is success

Startup Probe

- Used for long startup time applications
- Disables liveness and readiness probes until Startup Probe succeeds
- Failed probe after “failureThreshold x periodSeconds”

Workloads

- Deployment
- ReplicaSet
- DaemonSet
- Job and CronJob

ReplicaSet

- Maintain a stable set of replica Pods at any given time
- “Pod managers”
- Defined with fields
  - Selector
  - Number of replicas it should maintain
  - Pod template

ReplicaSet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

Deployment

- Layer on top of ReplicaSet
- Describes a desired state
- Use cases
    - Rollout a ReplicaSet
    - Declare new state of Pods
    - Rollback Deployment
    - Scale up Deployment
    - Pause rollout of a Deployment
    - Cleanup ReplicaSets

Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

DaemonSet

- Run a copy of a pod on all Nodes
- Typical uses
  - Cluster storage
  - Logs collection
  - Node monitoring

Jobs

- Creates one or more pods
- Will retry execution until completion
- Used to reliably run one Pod to completion
- Run a job on schedule => CronJob
  - Will include a schedule

Jobs

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Service

- Abstraction to define a logical set of pods
- Service types
    - ClusterIP
    - NodePort
    - LoadBalancer

Configuration

- ConfigMap

  Store configuration for objects in the cluster to use
- Secret

  Contains small amount of sensitive data

That’s a wrap.
Any questions?