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?

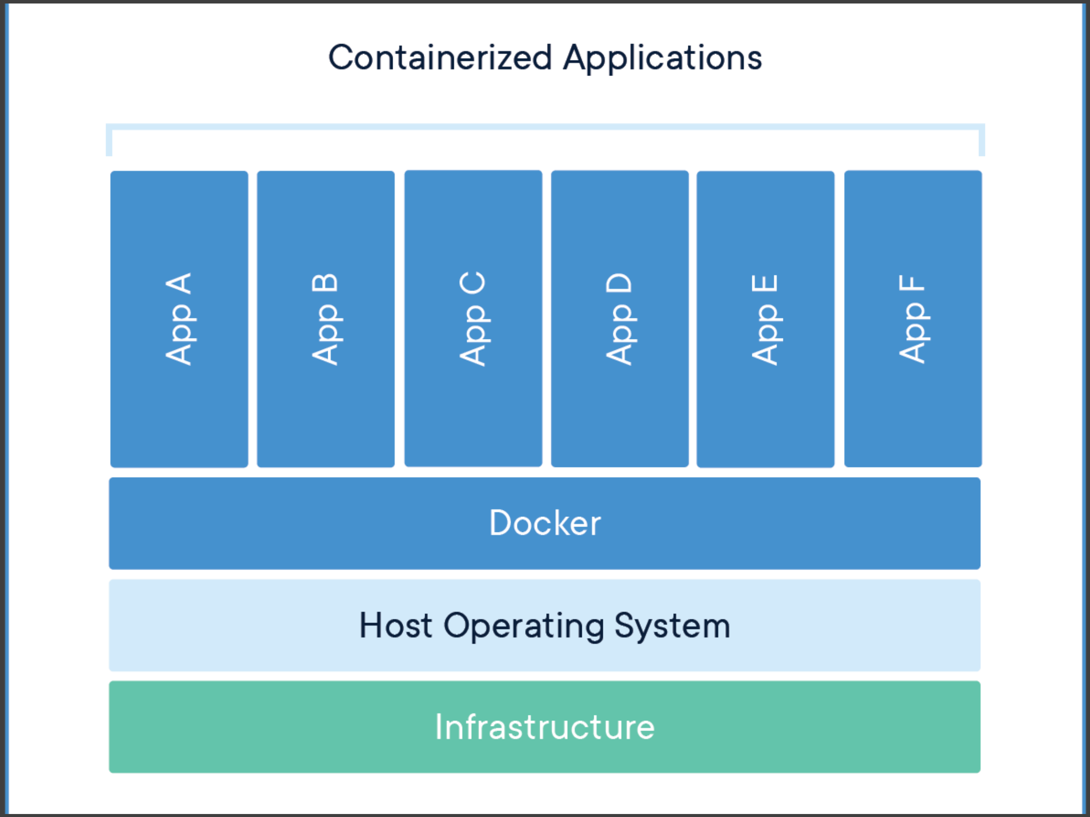
- Graduated in 2020 from KU Leuven
- Working @Skyline Communications
- Cloud Developer for 2 years
- Completed CKAD exam
- Now a Product Owner

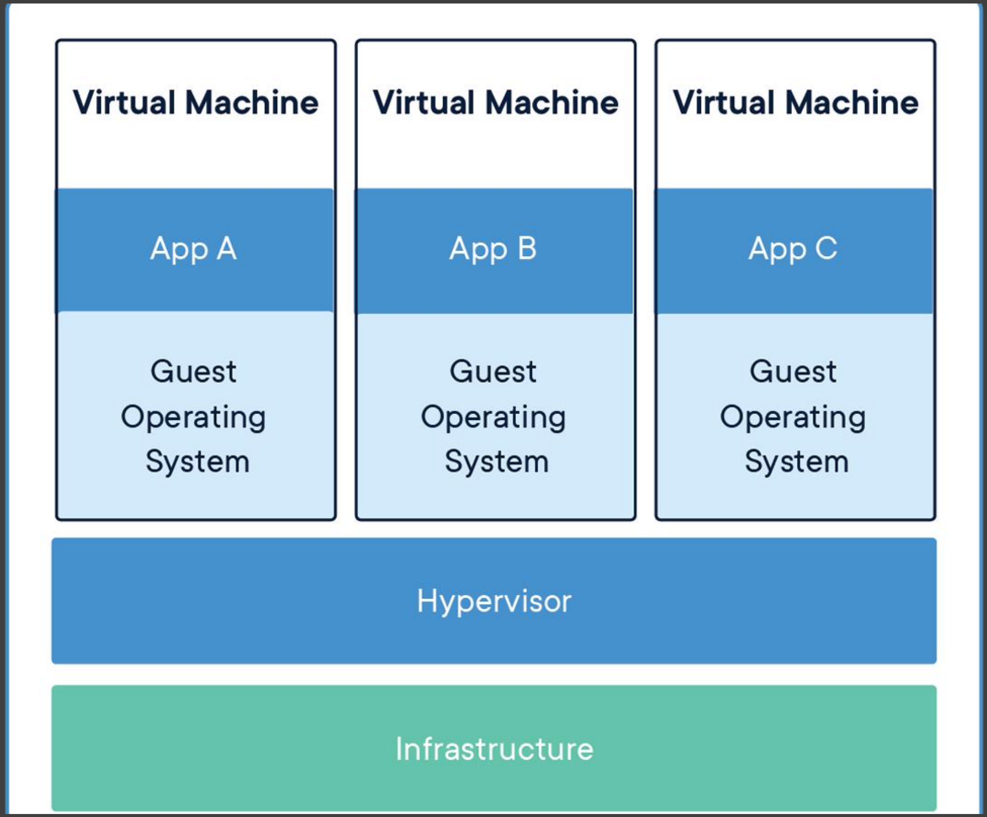
Overview

- Containers vs Virtual Machines
- Docker
- Kubernetes

Software deployment

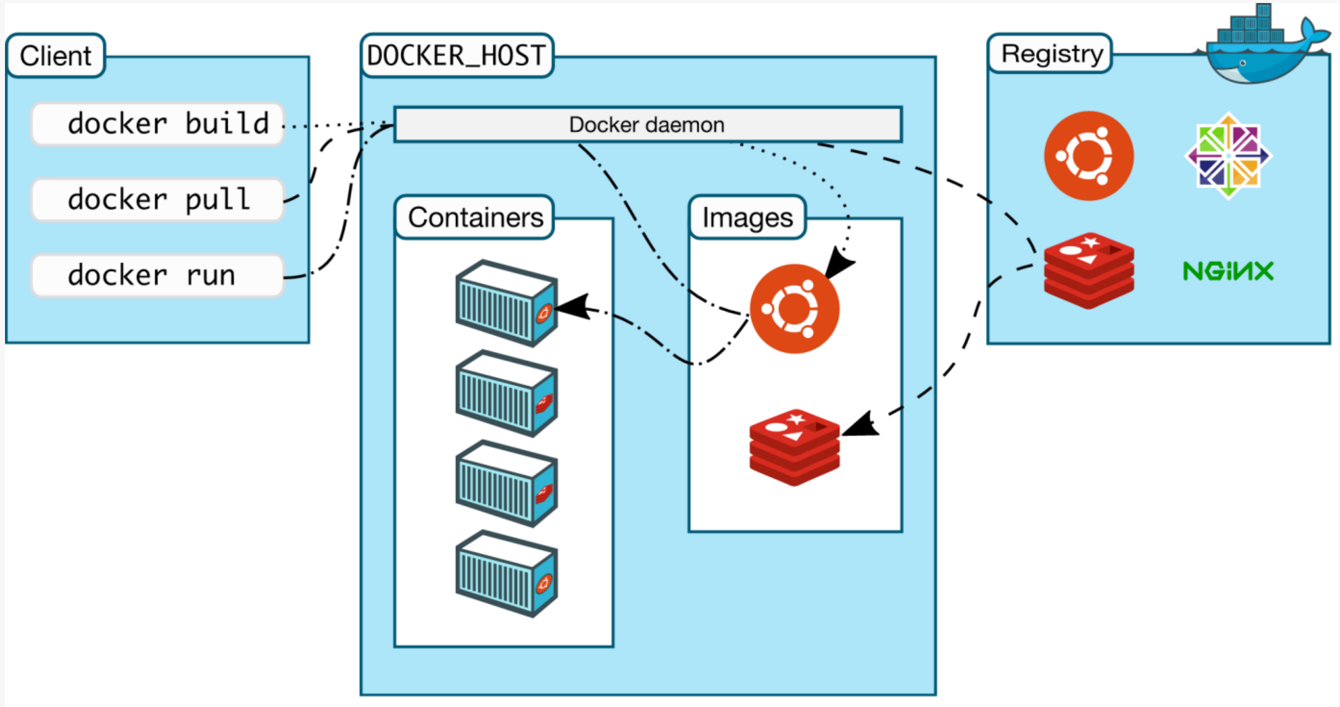
- How to deploy?
  - Service on bare metal server
  - Virtual Machine
  - Containers





Containers vs Virtual Machines

Docker Architecture



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"]
```

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>

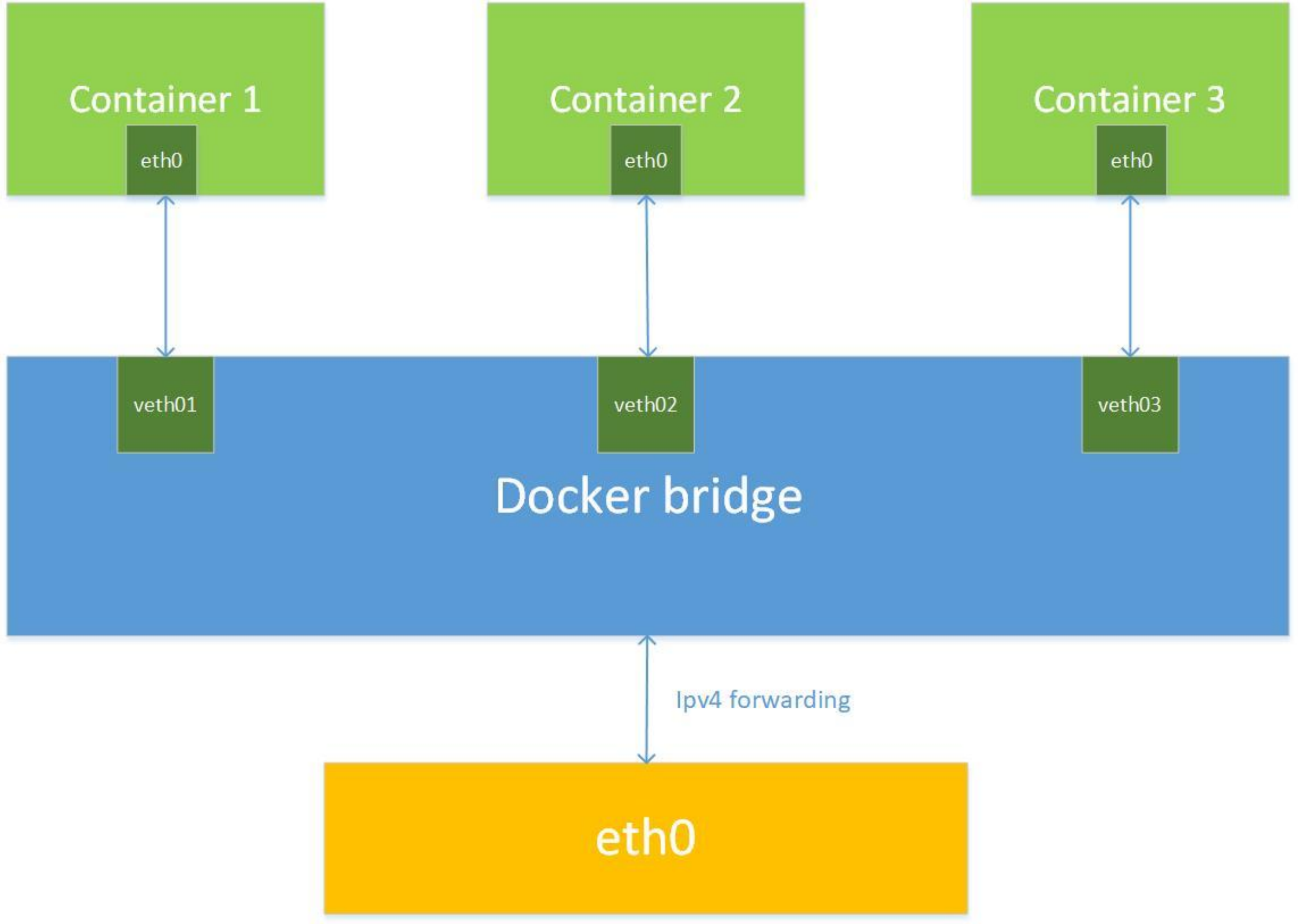
  <target image>:<tag>
- **docker push** *<image>*

Docker Networking

- **None**
- **Bridge (default)**
- **Overlay**
- Host
- Ipvlan
- Macvlan
- Custom plugin

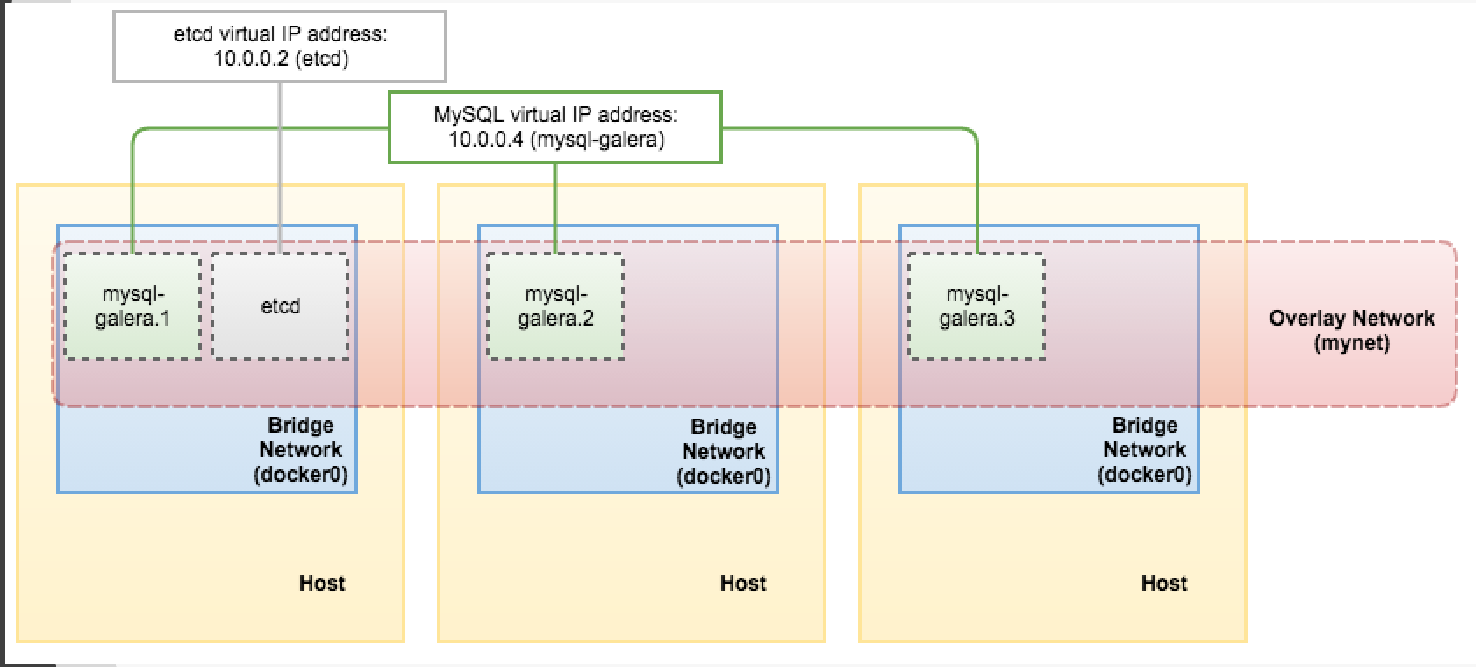
Docker Networking

Bridge network



Docker Networking

Overlay network



Docker Networking

Commands

- **docker network create**

  -d <bridge/overlay>

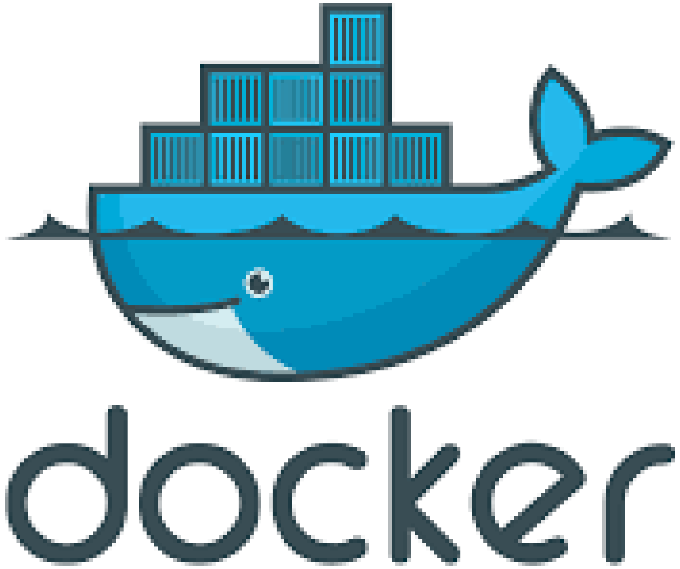
  <name>
- **docker network connect**

  <network name> <container name>
- **docker run**

  --network=<network name>
- **docker network disconnect**

  <network name> <container name>
- **docker network inspect** *<network name>*
- **docker network ls**
- **docker network rm** *<network name(s)>*

Run Docker images



Run busybox image

- **docker run** busybox
- **docker container** ls (-a)
- **docker run** busybox echo “hello from busybox”
- **docker run** -it busybox sh
  - ls
  - Vi
  - Ifconfig
- **docker network** ls
- **docker network** inspect bridge

Run basic container

- **docker run** -d -p 80:80 docker/getting-started
- **docker run** --name dockerdocs -d -p 80:80 docker/getting-started
- **docker run** --name nginx -d -p 81:80 nginx:latest

Configure bridge networking

- **docker run** --network=“none” -it --name busybox busybox
  - ifconfig
- **docker network** create –d bridge mynetwork
- **docker network** connect mynetwork busybox
- **docker inspect** busybox -f "{{json .NetworkSettings.Networks }}“
- **docker network** inspect none
- **docker network** disconnect none busybox
- **docker network** connect mynetwork busybox

Configure bridge networking

- **docker run** --network=“mynetwork” -it --name busybox2 busybox
  - Ifconfig
  - Ping busybox
  - Ping <IP>
- **docker network** inspect mynetwork

Configure bridge networking

- **docker network** create –d bridge myothernetwork
- **docker network** disconnect mynetwork busybox
- **docker network** connect myothernetwork busybox

Modify and save running container

- **docker run** --it --name template busybox
  - mkdir workshop
  - cd workshop
  - touch test.txt
  - cat test.txt
  - printf 'Hellow\nWorld\n' > ./test.txt
  - cat test.txt
  - exit
- **docker container** ls -a
- **docker commit** template custombusybox
- **docker images**
- **docker run** --it --name mycustombusybox 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 http://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 http://localhost:80/

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
- **docker commit** webserver1 web-server:v3
- **docker run** --name webserver3 -d -p 82:80 web-server:v3

Surf to localhost:82 in browser or try

```
curl http://localhost:82/
```

Clean setup

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

Kubernetes

K8s according to ChatGPT

Kubernetes is a **container orchestration** system that automates the management and deployment of containerized applications. It provides features such as **scaling, self-healing, and rolling updates** to make running containerized applications more efficient and reliable. It can be thought of as a **"control center"** for managing and coordinating multiple containers running on a cluster of machines. Essentially, it makes it **easier for developers to deploy and manage** their applications in a distributed environment.

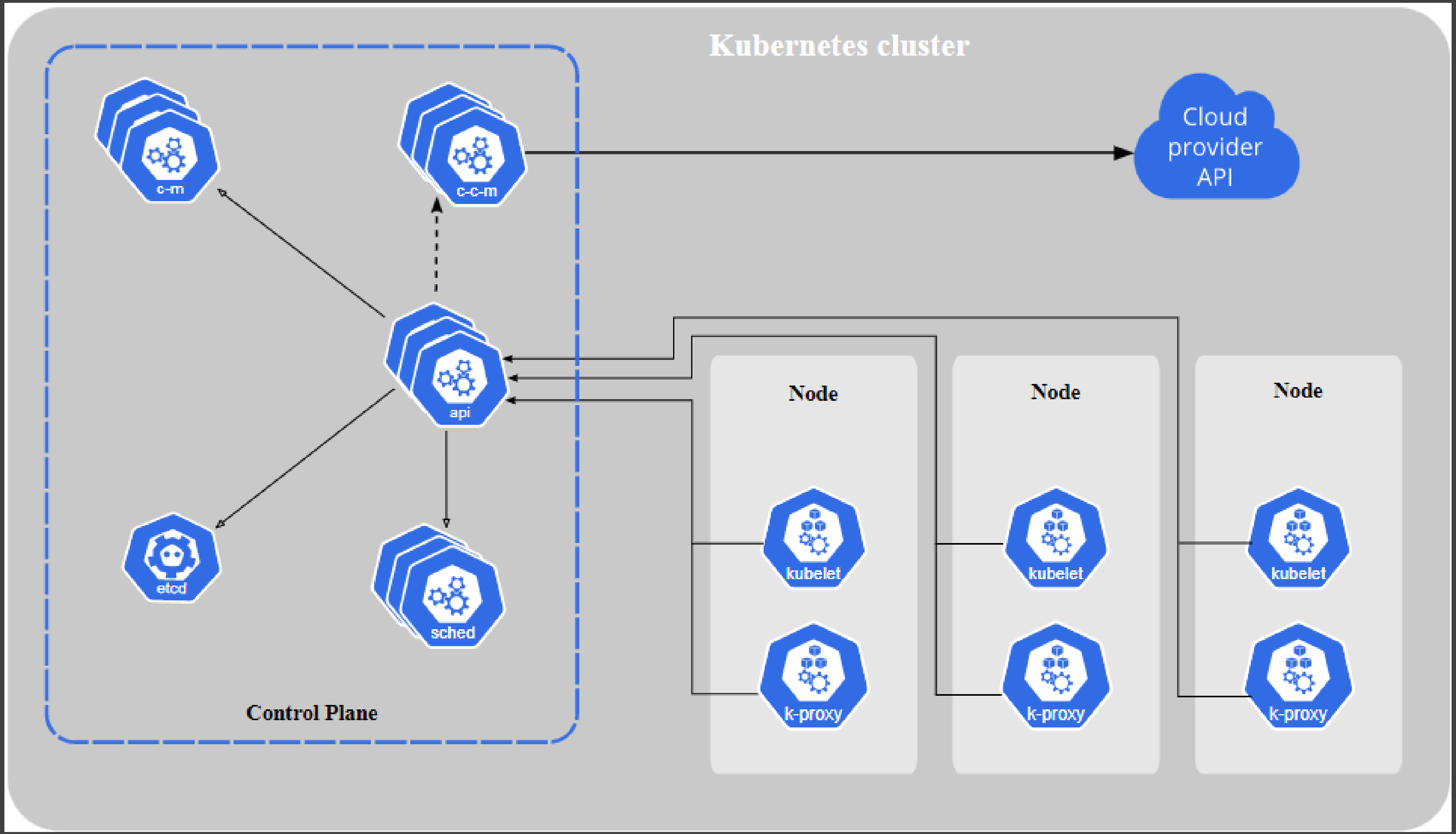
Kubernetes

- Uses containers
- Framework to run distributed system resiliently
- Provides
  - Scaling
  - Failure coping mechanisms
  - 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

K8S resources

- Namespaces
- Pods
- Nodes
- ReplicaSets
- Deployments
- Labels and annotations
- Services
- ConfigMaps & Secrets
- Ingresses

Namespaces

- Virtual environment
- Resource isolation
- Applies to:
  - Deployments
  - Services
  - Replicasets

Pods

- Smallest unit
- Group of containers with shared context
  Mostly only 1 container per pod in production setups
- Shared resources and context

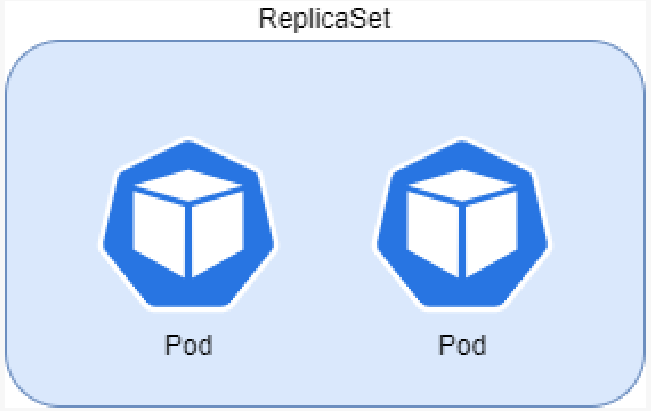
Nodes

- Physical machines
- Run all services
  - K8s system services
  - Application services

Workloads

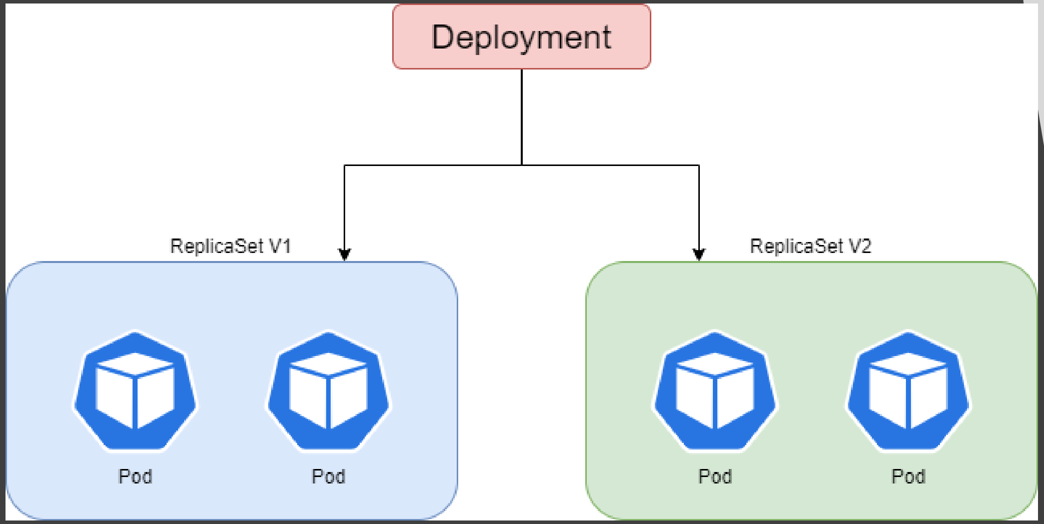
- ReplicaSet
- Deployment
- DaemonSet
- Job and CronJob

ReplicaSet



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

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
spec:
 selector:
  matchLabels:
   app: nginx
 replicas: 2
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx
```

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

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

Startup Probe

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

Liveness Probe

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

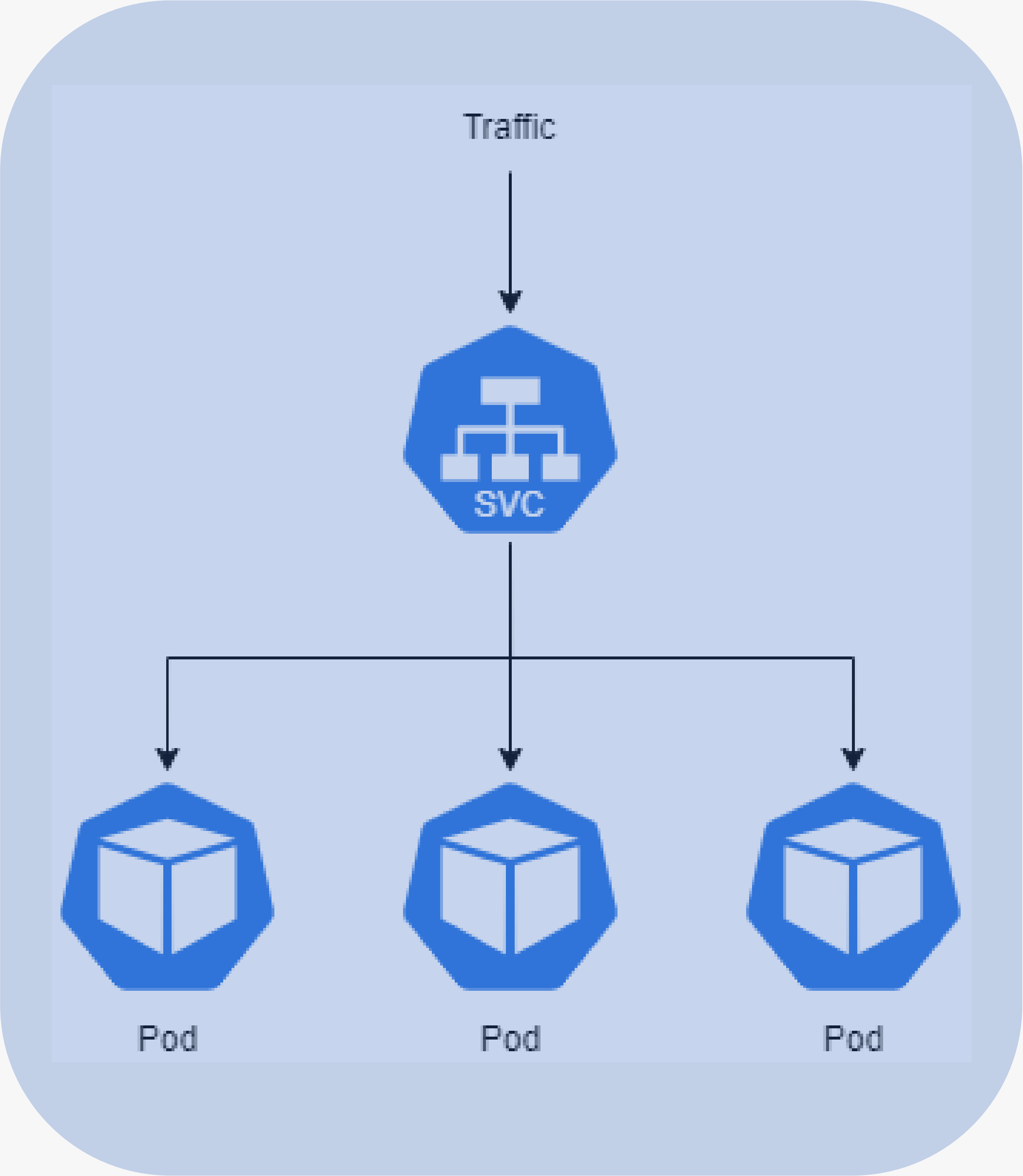
Readiness Probe

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

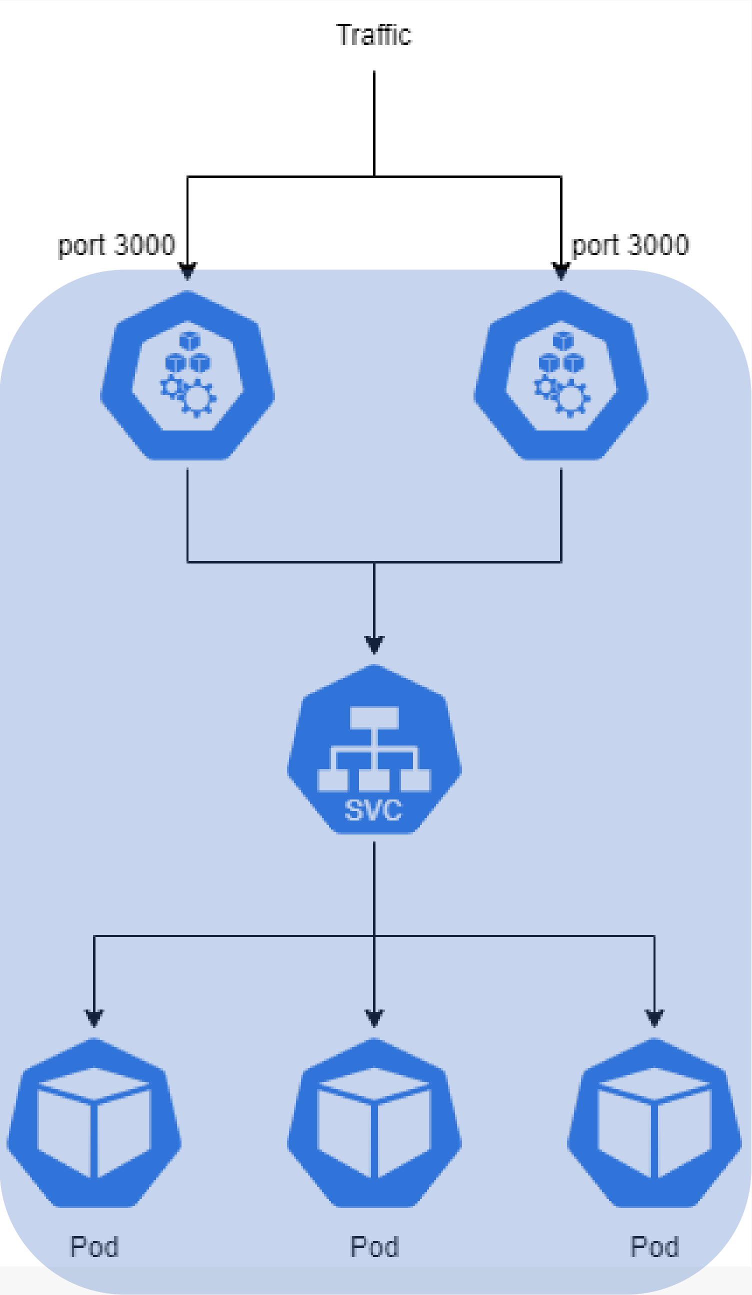
Service

- Easy access to pods in deployments
- Abstraction to define a logical set of pods
- Service types
  - ClusterIP
  - NodePort
  - LoadBalancer
  - ExternalName

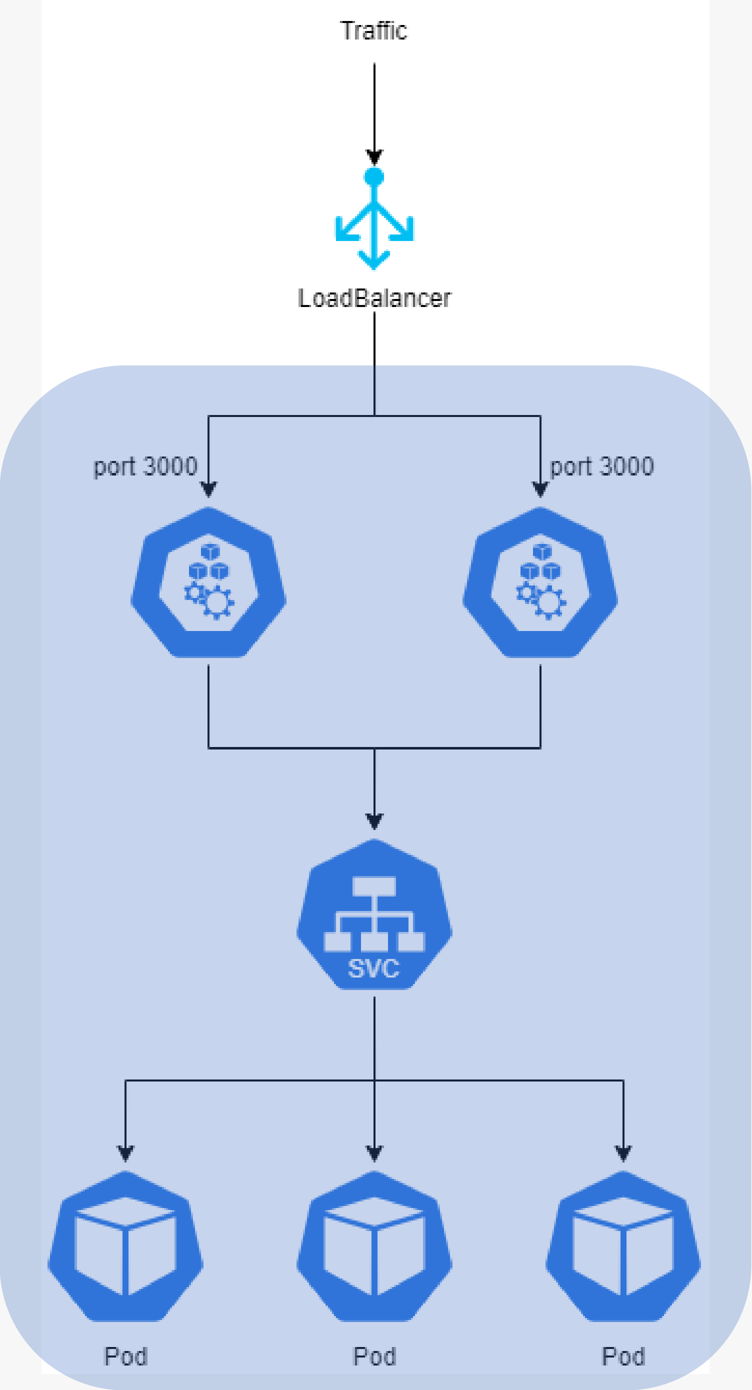
ClusterIP Service



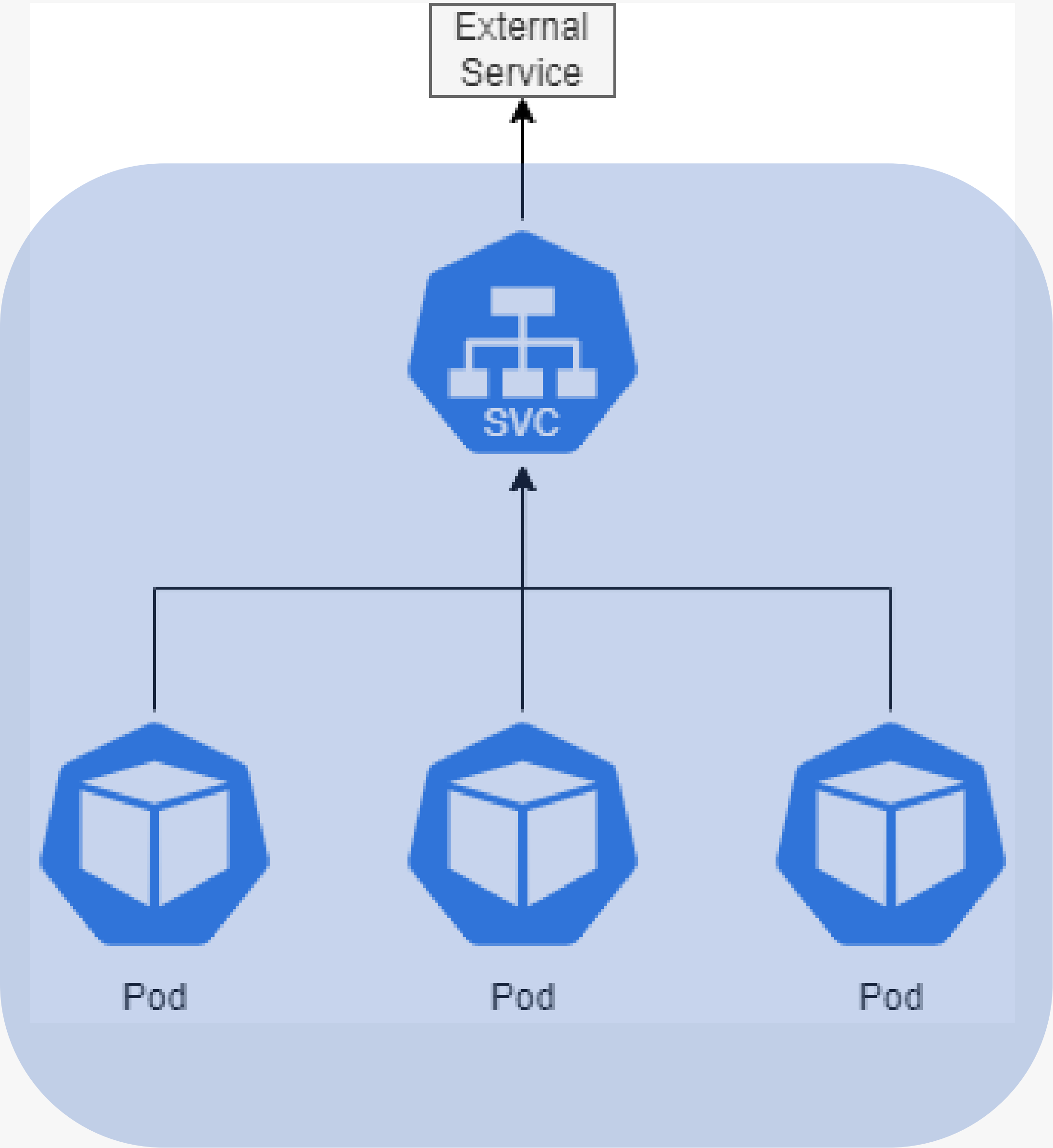
NodePort Service



LoadBalancer Service



ExternalName Service



Configuration

- ConfigMap

  Store configuration for objects in the cluster to use

- Secret

  Contains small amount of sensitive data

Hands-On

Connect to Cluster

- Check resource group and name of cluster
  - Resource group: cca-develop-weu
  - Name: cca-develop
- Connect
  - az aks get-credentials –g cca-develop-weu –n cca-develop

Create Namespace

```
kubectl create namespace workshop
```

Now on to pods

- k run *<podname>* --image=*<imageName>*

  --restart=Never

  --labels="*<name>*=*<value>*"
- k get pods -o wide
- k exec --stdin --tty *<podname>* -- /bin/sh

And nodes

- k get nodes
- k describe node <*name*>
- k cordon <*name*>
- k drain <*name*>
  - --ignore-daemonsets
- k uncordon <*name*>

Play with ReplicaSets

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx
  labels:
    app: demo
    tier: frontend
    version: v1
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: nginx
        image: nginx
```

Running and updating 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
        ports:
        - containerPort: 80
```

Jobs

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-with-timeout
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34.0
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 5
```

Testing with Liveness Probe

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec3
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 10; rm -rf /tmp/healthy; sleep 20
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
      failureThreshold: 3
```

That’s a wrap.
Any questions?