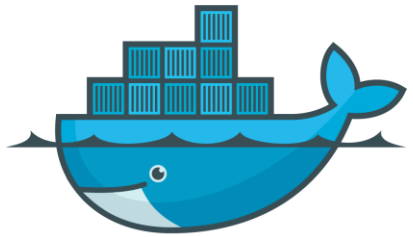
Before we start

- Install Docker Desktop
- Download code from https://aka.dataminer.services/dockerworkshop
  - Extract under c:\

Containers what, how and why?

Workshop about ~~Docker and~~ Kubernetes.

kubernetes



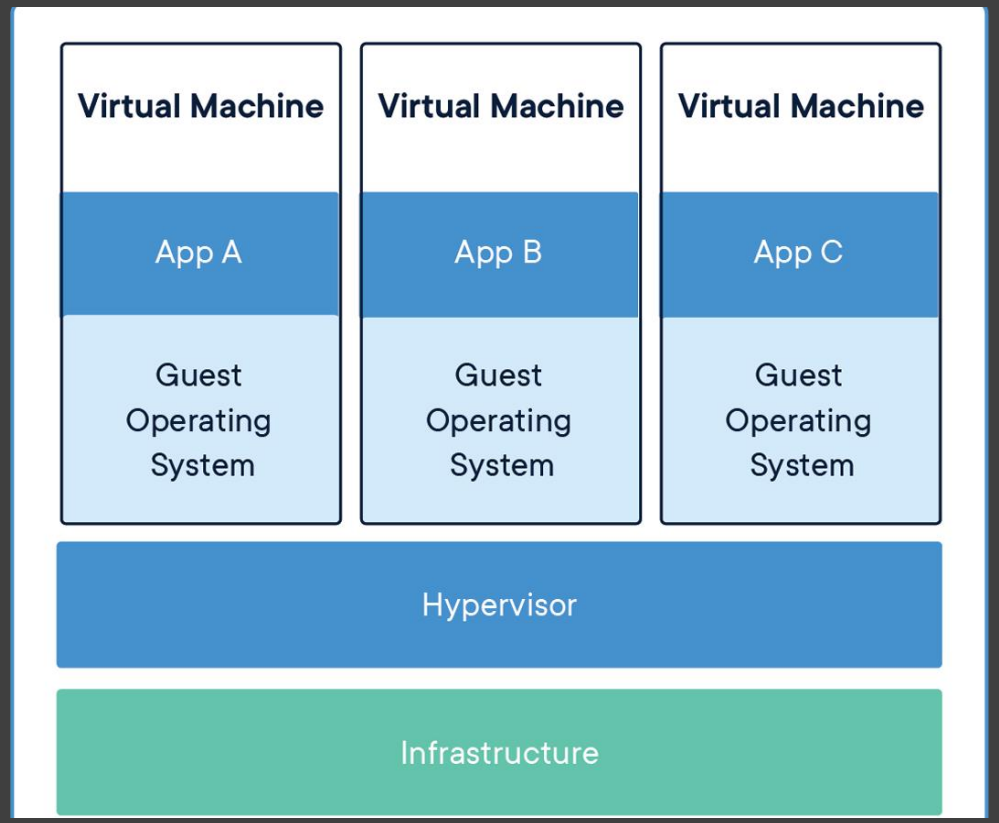
docker

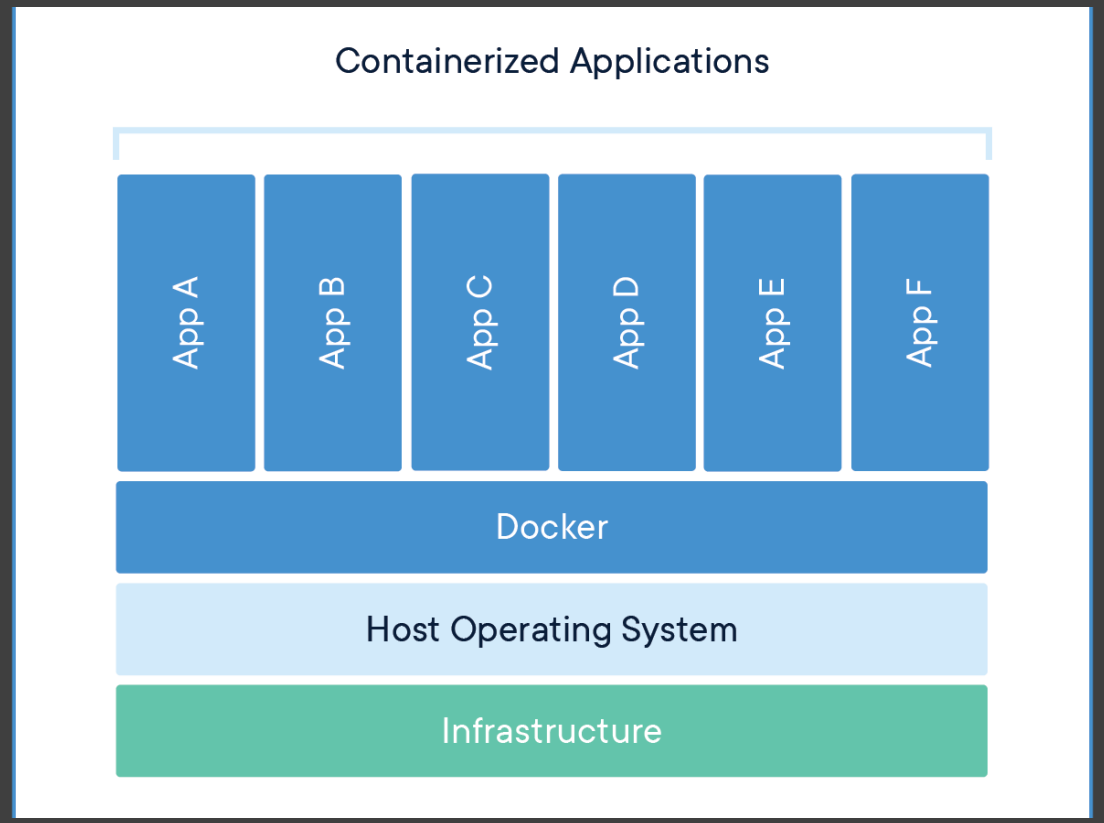
Overview

- Containers vs Virtual Machines
- Docker
- ~~Kubernetes~~

Software deployment

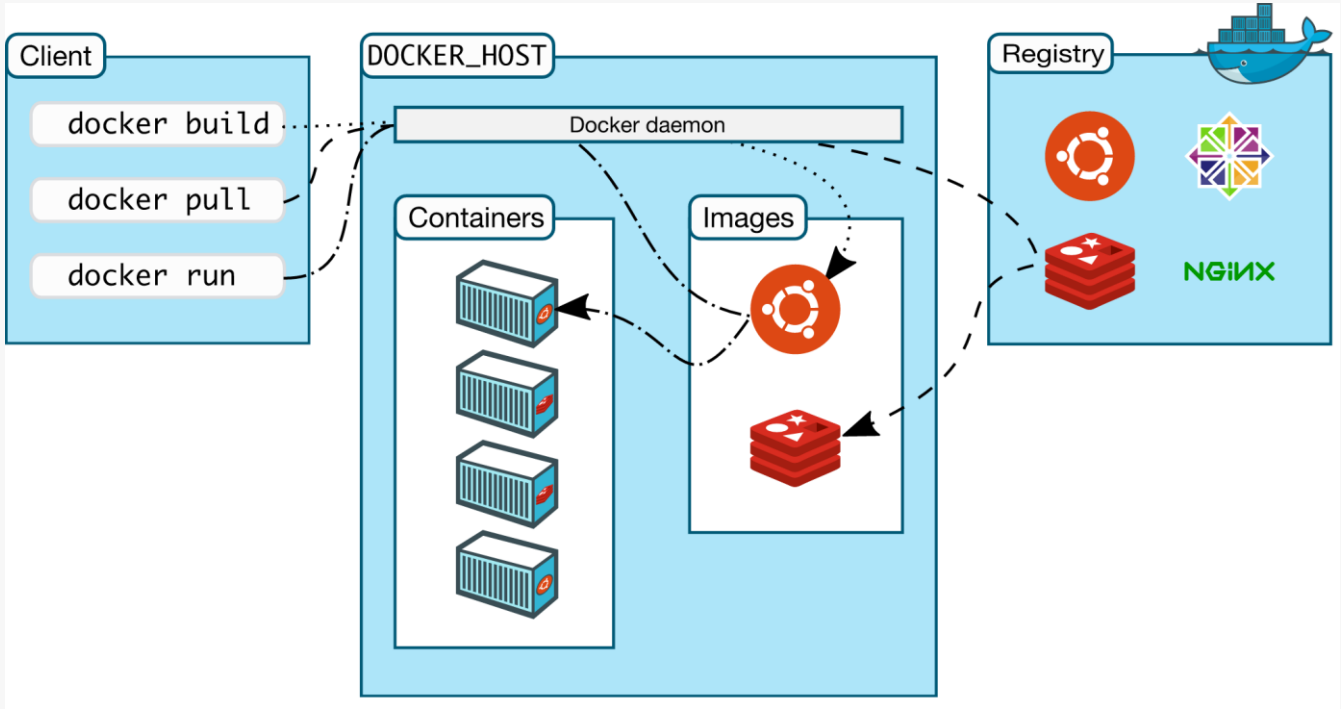
- How to deploy?
  - Service on bare metal server
  - Virtual Machine
  - Containers
  - Throw it in the cloud ☺





Containers vs Virtual Machines

Docker Architecture



Example Dockerfile

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

Base Image

Image Definition

Entrypoint

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"]
```

Advanced Dockerfile

```
PS C:\CCAAZURE\Hub\CcaHub> docker build -f .\Hub.Web\Dockerfile -t hubbuild:v1 .
[+] Building 129.2s (17/17) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 405B
 => [internal] load .dockerignore
 => => transferring context: 176B
 => [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:6.0
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk:6.0
 => [build-env 1/7] FROM mcr.microsoft.com/dotnet/sdk:6.0@sha256:b1ac02d48e170b007d2d2392119fa037e531f1ad93cf8433d911f402d00c743a
 => => resolve mcr.microsoft.com/dotnet/sdk:6.0@sha256:b1ac02d48e170b007d2d2392119fa037e531f1ad93cf8433d911f402d00c743a
 => => sha256:3ced45b0441dd414a9c6b95a627e557726bf95cf3042660f61e8947c4771d840 2.01kB / 2.01kB
 => => sha256:a65621c96b07cd4500f85cbab71a949ac3251ad056794e16566dd0bea71a11ce 31.66MB / 31.66MB
 => => sha256:6c3981608c2b6b3825fe63b9c920101d7bf4ca80ceda7ddc0dc09fdc93a5797d 14.97MB / 14.97MB
 => => sha256:b1ac02d48e170b007d2d2392119fa037e531f1ad93cf8433d911f402d00c743a 1.82kB / 1.82kB
 => => sha256:c998ae9ecc920b5f674da908d8f10462f3b30209c06c12cccffdd637a1bd8e96 7.17kB / 7.17kB
 => => sha256:14726c8f78342865030f97a8d3492e2d1a68fbd22778f9a31dc6be4b4f12a9bc 31.42MB / 31.42MB
 => => sha256:5a9070a06ae012d7ce05a03c90b76c57ebca94c3f6d434e27ff129531ccb5629 156B / 156B
 => => sha256:d757ae0f0e6a72da889c1c194992d6dcf786868b4ceff54d4022a5e3801a24fb 9.47MB / 9.47MB
 => => sha256:5dec5cf8f8b08d89661916e820d41899d37227d934347e734801a6b1d75da039 25.37MB / 25.37MB
 => => sha256:2c5489b3efcc87ebfd8bcba003c4a0f88377637eab2763a06da2eb2f0d4f2ec6 148.73MB / 148.73MB
 => => extracting sha256:14726c8f78342865030f97a8d3492e2d1a68fbd22778f9a31dc6be4b4f12a9bc
 => => sha256:60a72515fb46798c93c10ac84c0e178ee5e18ffd8349cc6322e2ba7746c6085c 13.71MB / 13.71MB
 => => extracting sha256:6c3981608c2b6b3825fe63b9c920101d7bf4ca80ceda7ddc0dc09fdc93a5797d
 => => extracting sha256:a65621c96b07cd4500f85cbab71a949ac3251ad056794e16566dd0bea71a11ce
 => => extracting sha256:5a9070a06ae012d7ce05a03c90b76c57ebca94c3f6d434e27ff129531ccb5629
 => => extracting sha256:d757ae0f0e6a72da889c1c194992d6dcf786868b4ceff54d4022a5e3801a24fb
 => => extracting sha256:5dec5cf8f8b08d89661916e820d41899d37227d934347e734801a6b1d75da039
 => => extracting sha256:2c5489b3efcc87ebfd8bcba003c4a0f88377637eab2763a06da2eb2f0d4f2ec6
 => => extracting sha256:60a72515fb46798c93c10ac84c0e178ee5e18ffd8349cc6322e2ba7746c6085c
 => [internal] load build context
 => => transferring context: 11.28MB
 => [stage-1 1/4] FROM mcr.microsoft.com/dotnet/aspnet:6.0@sha256:e855110445f4a1b48bb0d5b6de9f14cd2dd311b1daaf047bf0f3965cab3e4a25
 => => resolve mcr.microsoft.com/dotnet/aspnet:6.0@sha256:e855110445f4a1b48bb0d5b6de9f14cd2dd311b1daaf047bf0f3965cab3e4a25
 => => sha256:036528001cbbdf4a5e0ddfd164dd075e3c3372c5c51d8e6d66a91c0653fcab2f 3.26kB / 3.26kB
 => => sha256:14726c8f78342865030f97a8d3492e2d1a68fbd22778f9a31dc6be4b4f12a9bc 31.42MB / 31.42MB
 => => sha256:6c3981608c2b6b3825fe63b9c920101d7bf4ca80ceda7ddc0dc09fdc93a5797d 14.97MB / 14.97MB
 => => sha256:a65621c96b07cd4500f85cbab71a949ac3251ad056794e16566dd0bea71a11ce 31.66MB / 31.66MB
 => => sha256:e855110445f4a1b48bb0d5b6de9f14cd2dd311b1daaf047bf0f3965cab3e4a25 1.82kB / 1.82kB
 => => sha256:fa8fe66222e71452405b4487b4ebc50e6ddcd0297bd0e4da008a3c89f107dda9 1.37kB / 1.37kB
 => => sha256:5a9070a06ae012d7ce05a03c90b76c57ebca94c3f6d434e27ff129531ccb5629 156B / 156B
 => => sha256:d757ae0f0e6a72da889c1c194992d6dcf786868b4ceff54d4022a5e3801a24fb 9.47MB / 9.47MB
 => => extracting sha256:14726c8f78342865030f97a8d3492e2d1a68fbd22778f9a31dc6be4b4f12a9bc
 => => extracting sha256:6c3981608c2b6b3825fe63b9c920101d7bf4ca80ceda7ddc0dc09fdc93a5797d
 => => extracting sha256:a65621c96b07cd4500f85cbab71a949ac3251ad056794e16566dd0bea71a11ce
 => => extracting sha256:5a9070a06ae012d7ce05a03c90b76c57ebca94c3f6d434e27ff129531ccb5629
 => [stage-1 2/4] WORKDIR /app
 => [build-env 2/7] WORKDIR /app
 => [build-env 3/7] COPY /Hub.Web/nuget.config /nuget.config
 => [build-env 4/7] COPY /Hub.Web/*.csproj ./
 => [build-env 5/7] RUN dotnet restore
 => [build-env 6/7] COPY . ./
 => [build-env 7/7] RUN dotnet publish Hub.Web -c Release -o out
 => [stage-1 3/4] COPY --from=build-env /app/out .
 => [stage-1 4/4] RUN rm ./nuget.config
 => exporting to image
 => => exporting layers
 => => writing image sha256:79bc1746e88536492505060fe9dbacc7b2bb28561ed2897543e527c2996224f5
 => => naming to docker.io/library/hubbuild:v1
```

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>

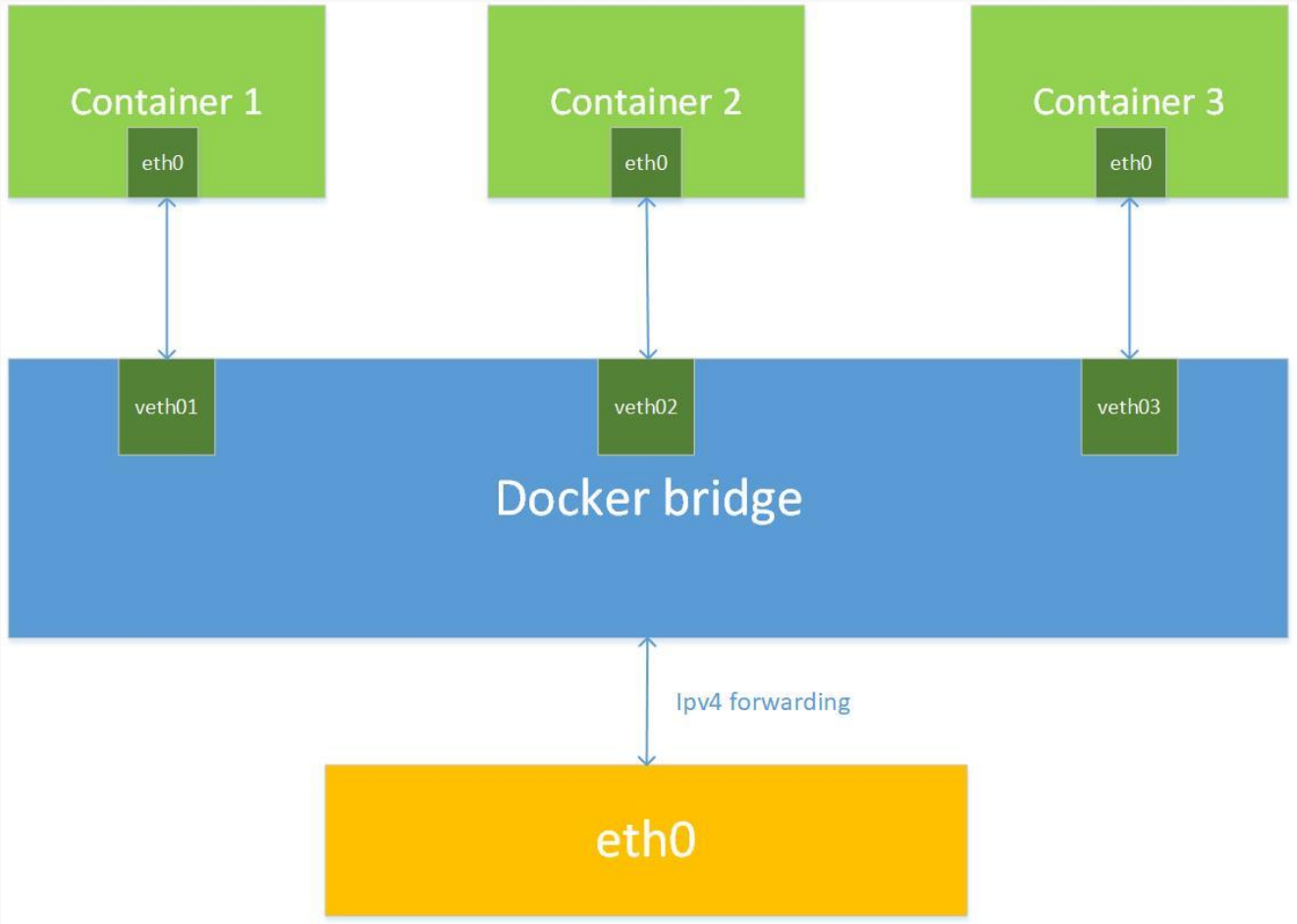
  <target image>:<tag>
- **docker push** *<image>*

Docker Networking

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

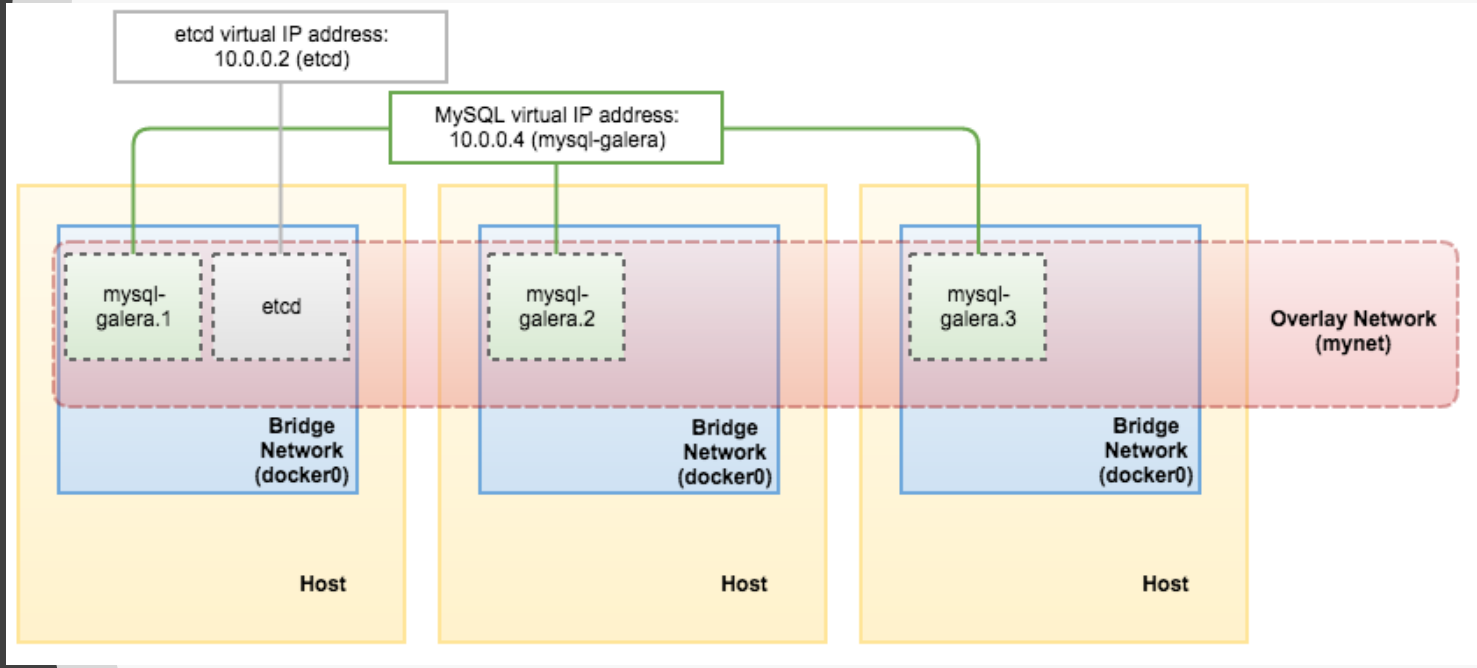
Docker Networking

Bridge network



Docker Networking

Overlay network

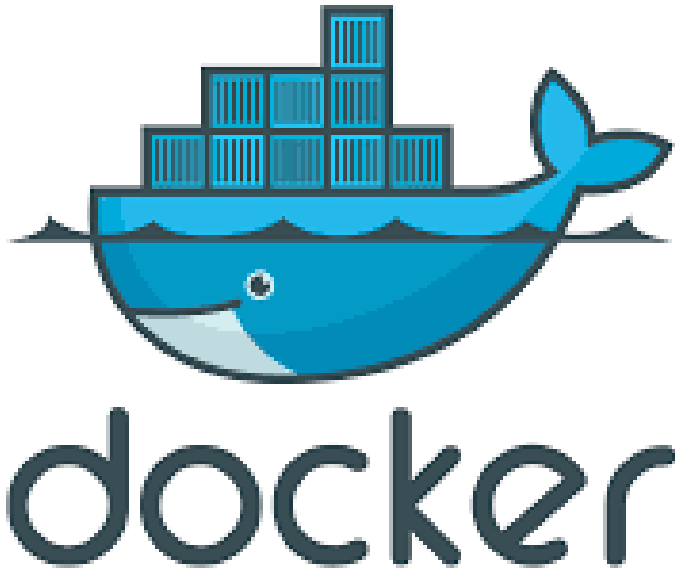


Docker Networking

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** --name shell -it busybox sh
  - ls
  - Vi
  - Ifconfig
- **docker network** ls
- **docker network** inspect bridge

Run basic container

- **docker run** --name dockerdocs -d -p 80:80 docker/getting-started
- **docker run** --name nginx -d -p 80:80 nginx:latest
- **docker stop** dockerdocs
- **docker start** nginx
- **docker run** --name nginx2 -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
- **docker network** disconnect mynetwork busybox
- ifconfig (busybox)
- ping <IP busybox> (busybox2)

Configure bridge networking

- **docker network** create –d bridge myothernetwork
- **docker network** inspect 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 'Hello\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.

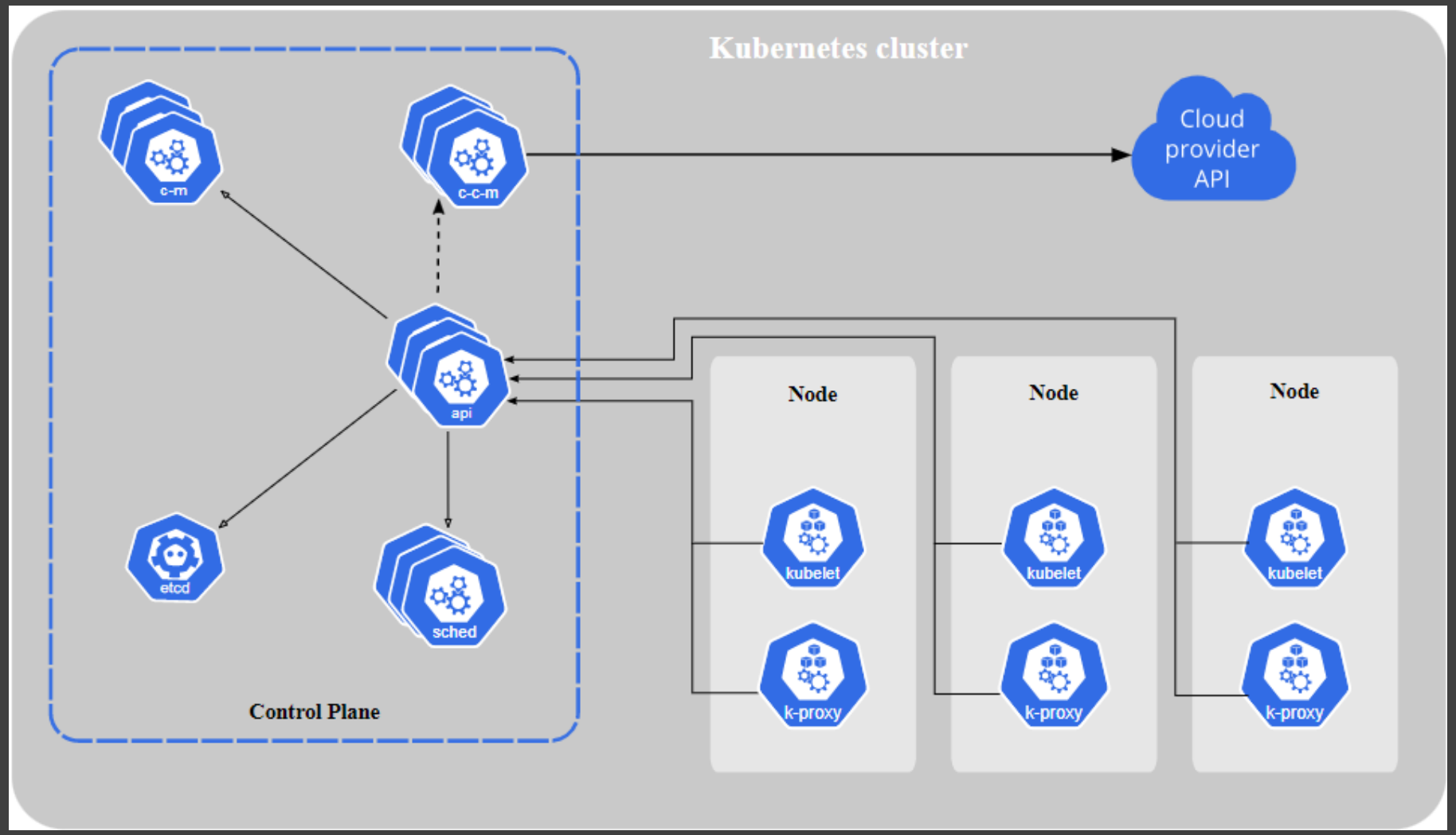
Kubernetes

- 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

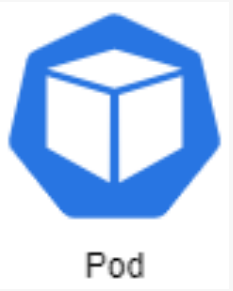
K8S resources

- 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

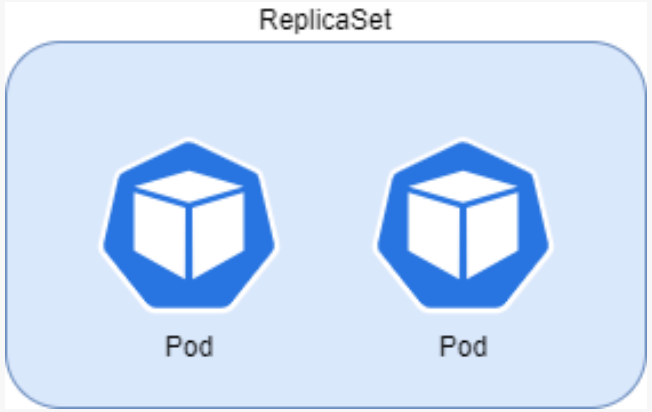
Nodes

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

Workloads

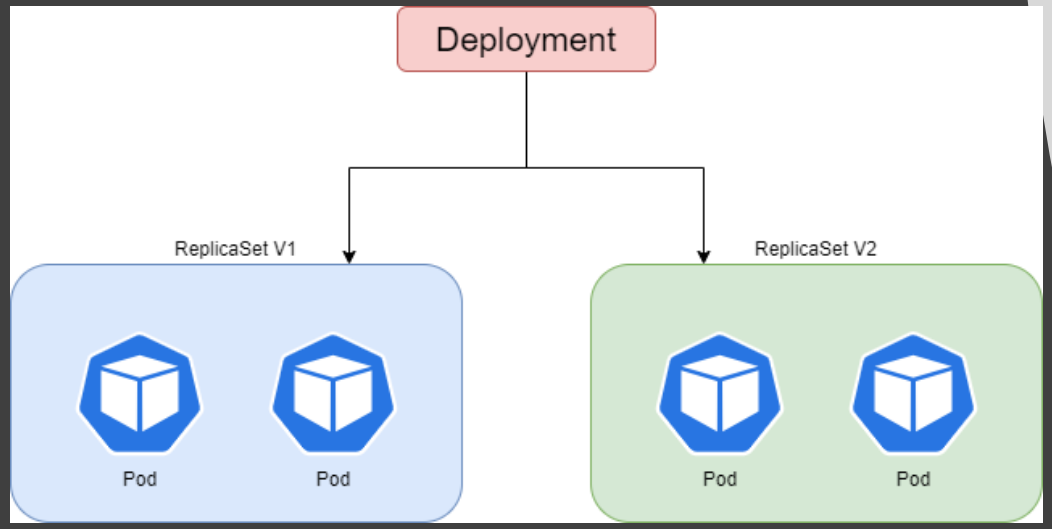
- ReplicaSet
- Deployment
- DaemonSet
- Job and CronJob

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

```
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

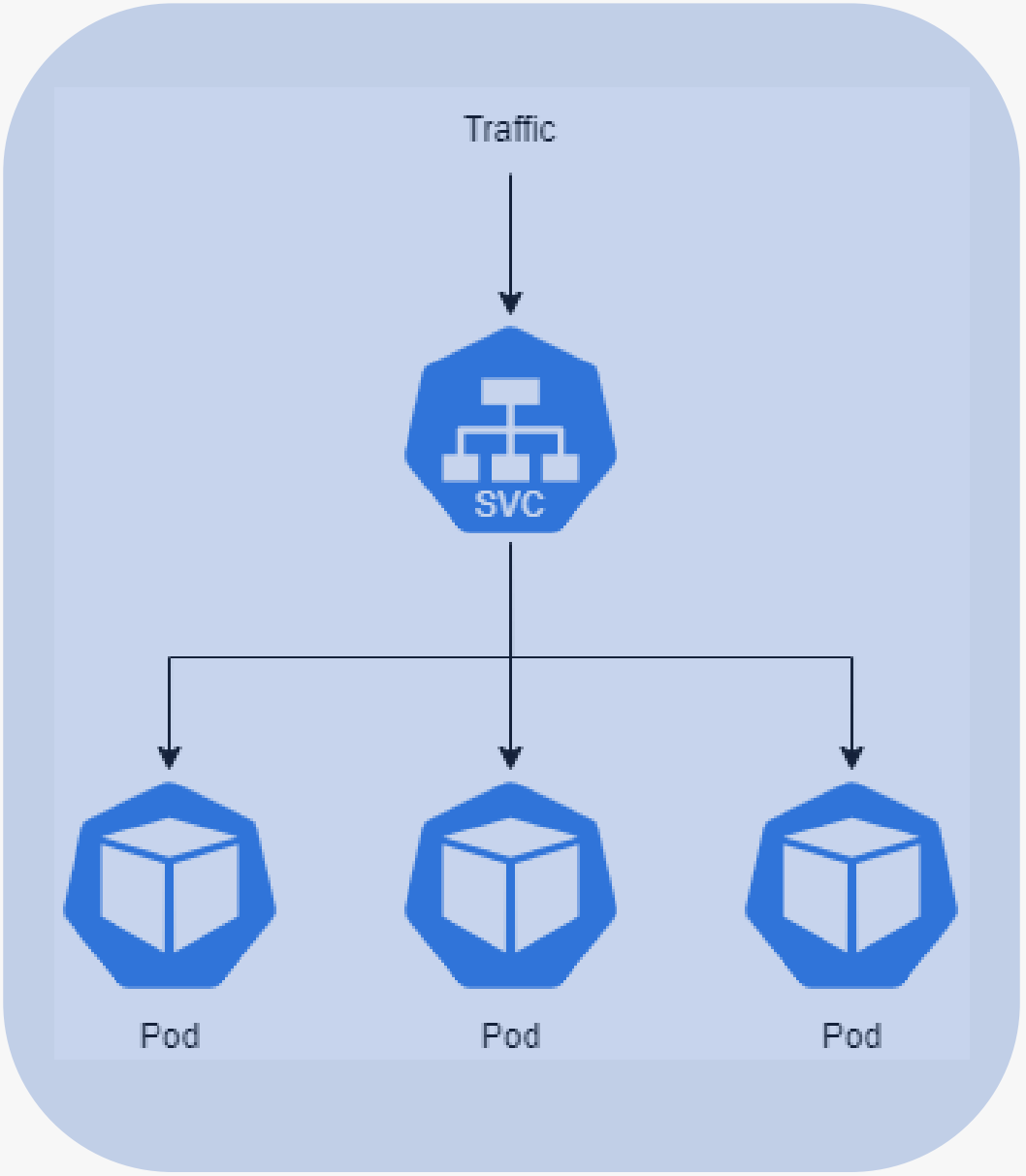
Readiness Probe

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

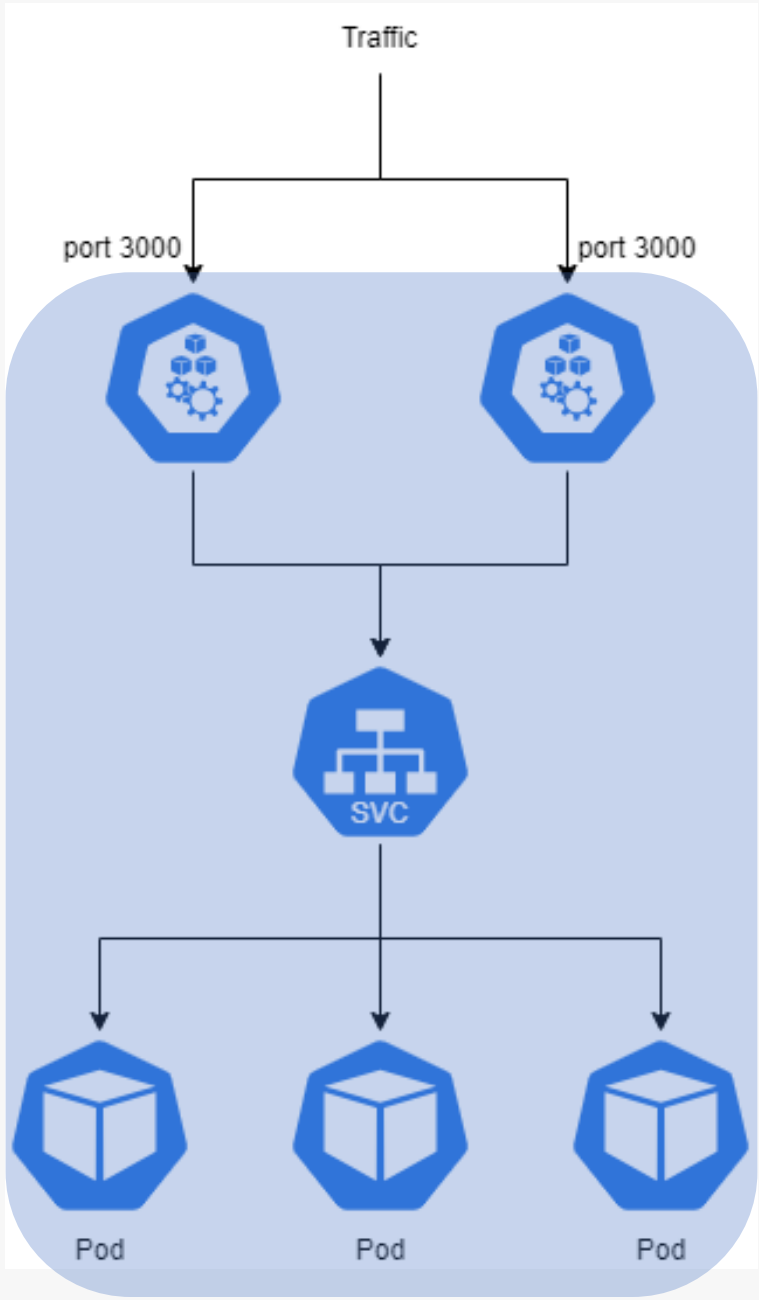
Service

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

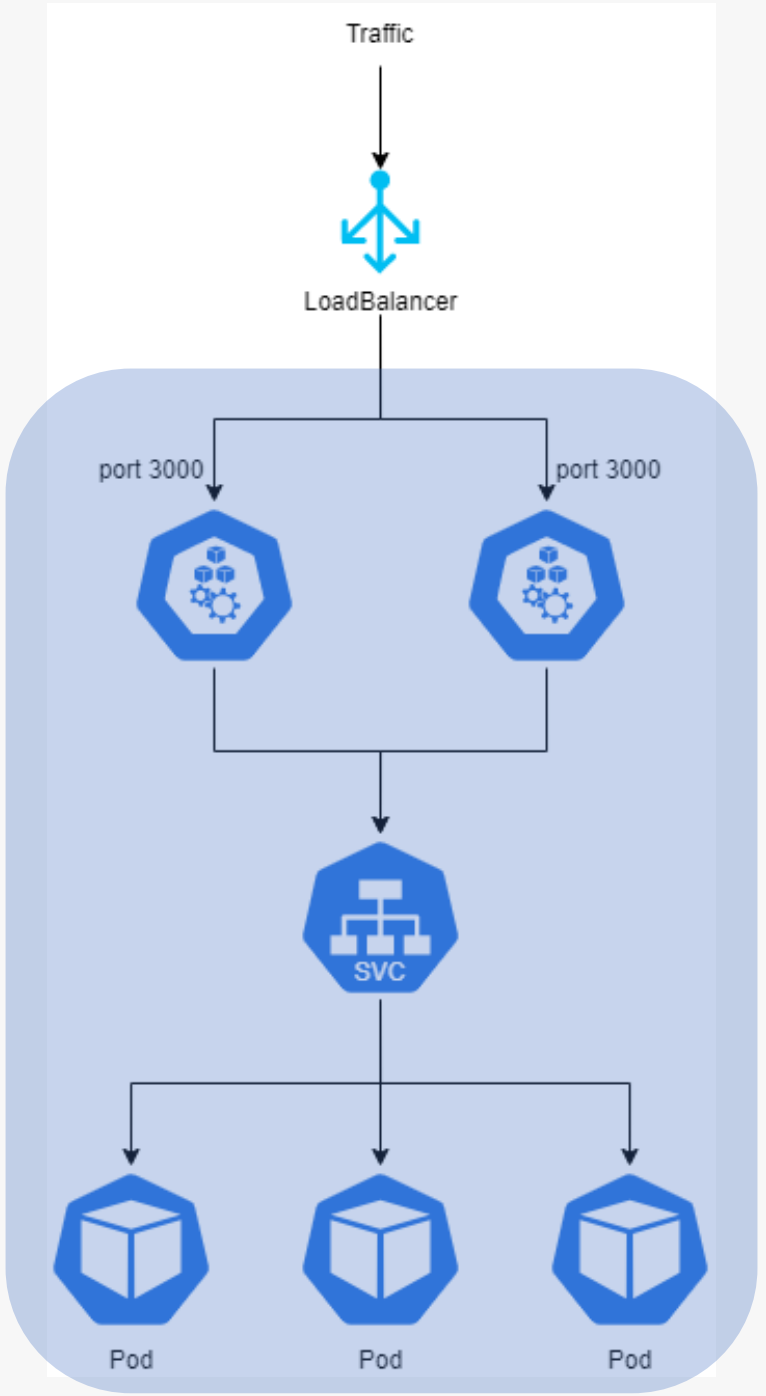
ClusterIP Service



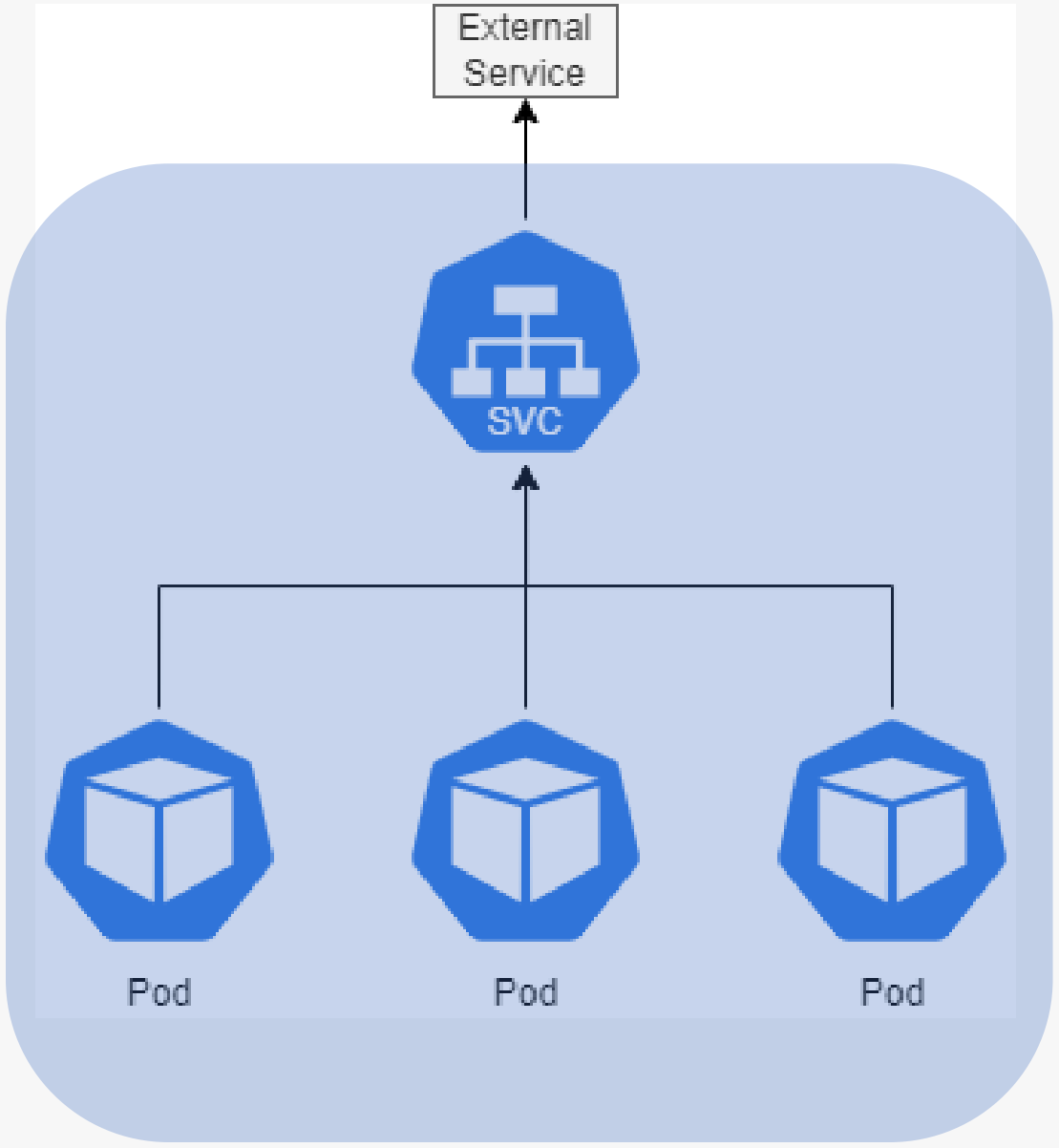
NodePort Service



LoadBalancer Service



ExternalName Service



Configuration

- ConfigMap
  Store configuration for objects in the cluster to use
- Secret
  Contains small amount of sensitive data

Hands-On

Install software

- Install Azure CLI
- Install Kubectl

Connect to Cluster

- Login on Azure
  - az login
  - az account set -subscription a476a434-0f2f-45db-bf17-cf278ef08379
- Check resource group and name of cluster
  - Resource group: rg-k8sworkshop-001
  - Name: aks-k8sworkshop-001
- Connect
  - az aks get-credentials –g rg-k8sworkshop-001 –n aks-k8sworkshop-001
- Set alias
  - Set-Alias -Name k -Value kubectl

Create Namespace

- kubectl create namespace <name>
- kubectl config set-context –current –namespace=<name>
- kubectl config unset contexts.<name of context>.<namespace>)

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

```
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
```

Basic service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: internal-svc
spec:
  ports:
  - port: 80
  selector:
    app: nginx
```

That’s a wrap.
Any questions?