Kubernetes

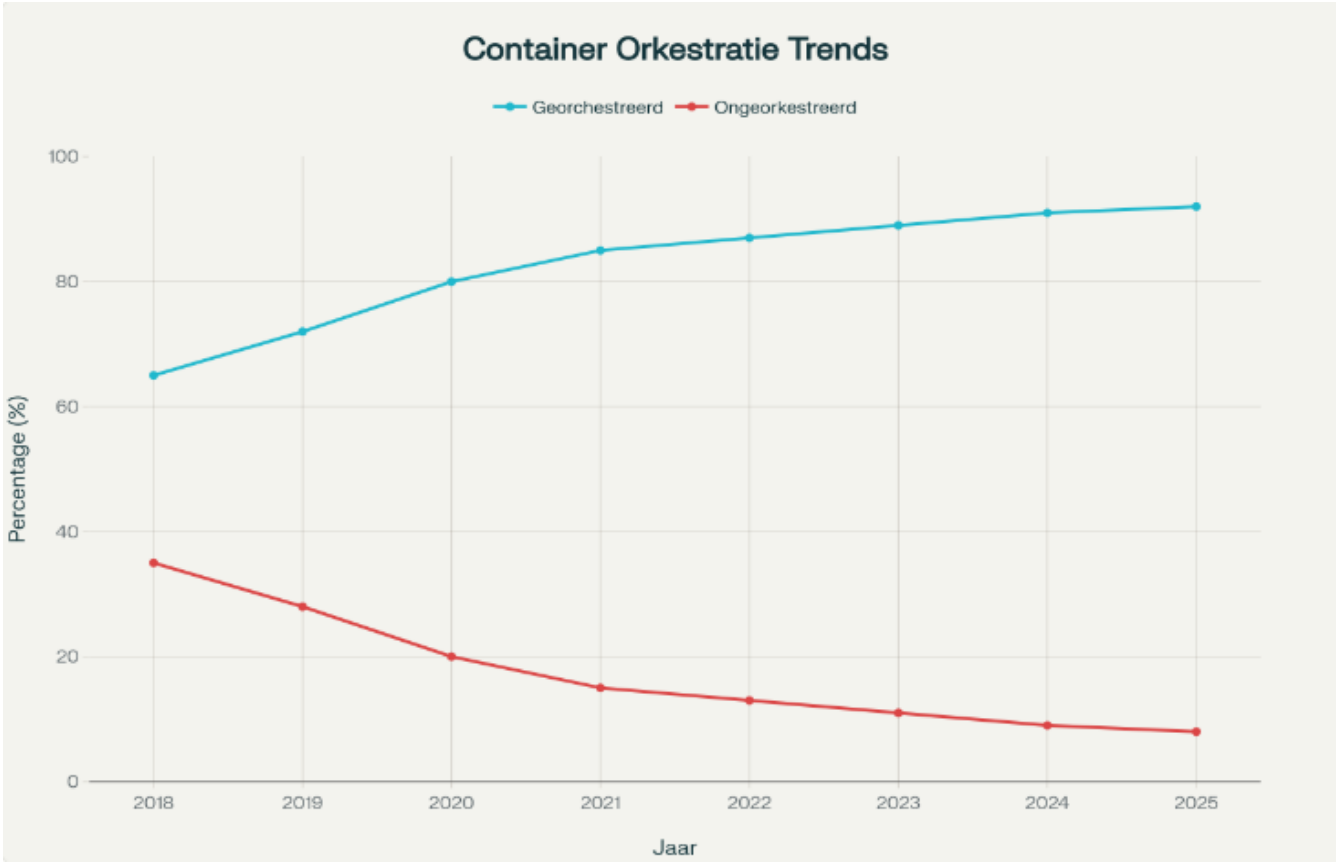
- Hoofdstuk 11

DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Inleiding

- Nu: meestal orchestration tool voor starten en stoppen containers



Kubernetes

- Open-source
- Bekendste tool voor orchestration
- Cloud en on-premises
- Geen handmatig beheer van servers meer nodig

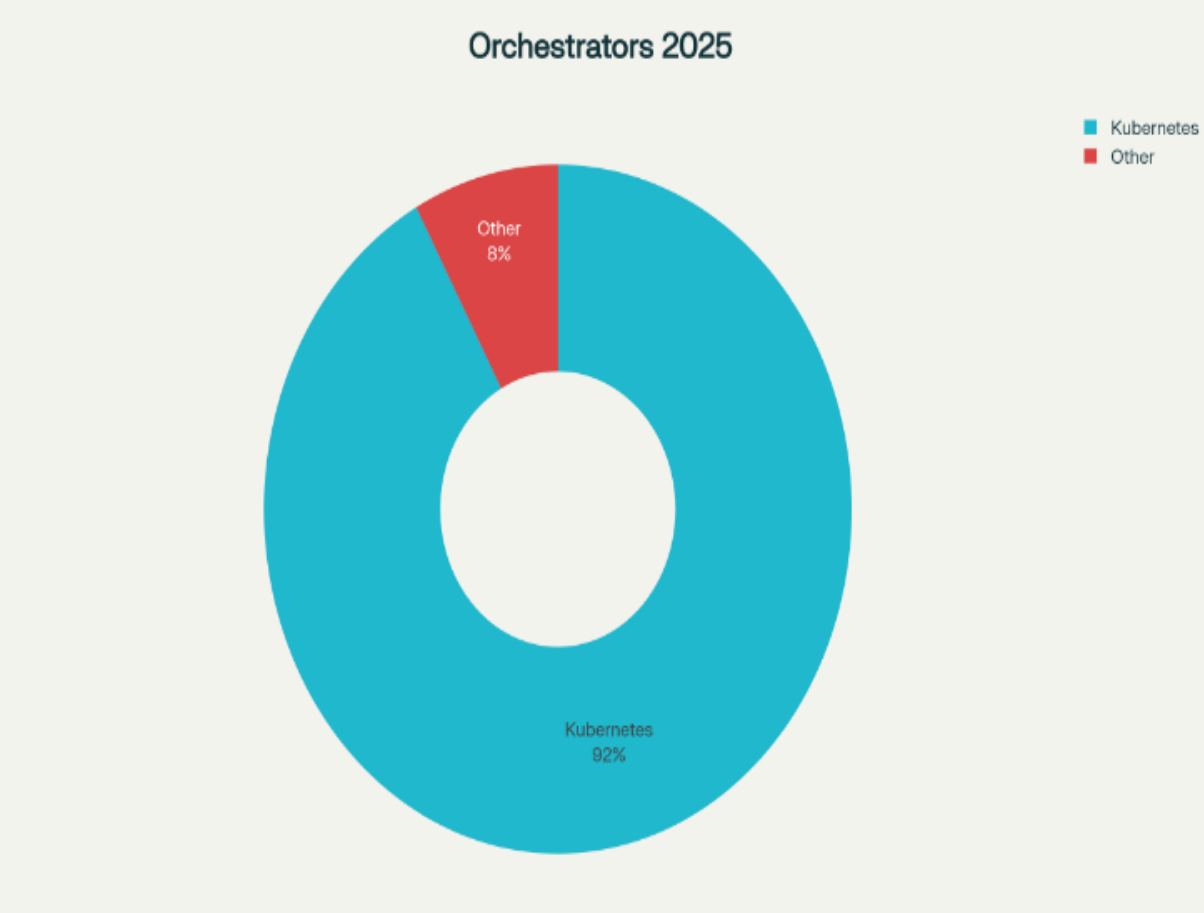
- Mogelijkheden:
  - Containers starten en stoppen
  - Schaalvergroting of -verkleining op basis van vraag
  - Load balancing
  - Zelfherstel bij fouten

Kubernetes

- Kubernetes = meest gebruikte platform voor container orchestration

- Alternatieven:
  - Docker Swarm
    - eenvoudiger, minder schaalbaar
  - Amazon ECS
    - managed AWS container orchestrator
  - OpenShift
    - Red Hat, met extra developer en security features

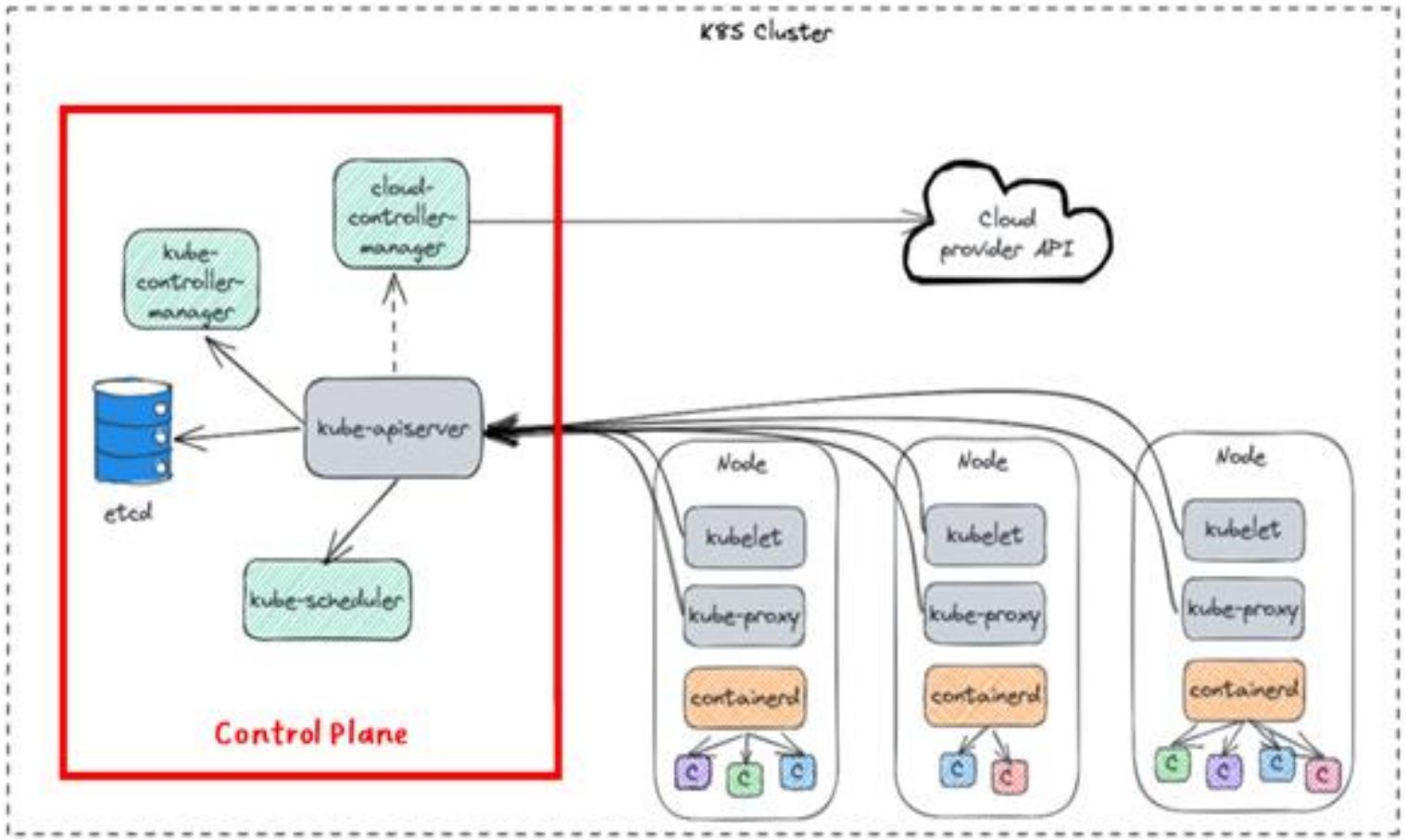
Kubernetes


Orchestrators 2025
Kubernetes
Other
Other
8%
Kubernetes
92%

Kubernetes architectuur

- Een cluster bestaat uit meerdere hosts (“nodes”) die samen containerized apps draaien.
- Hosts kunnen fysieke of virtuele machines zijn.
- De meeste clusters hebben een of meerdere worker nodes plus control plane node(s).
- Soorten nodes
  - Control plane(s) = beheerder(s) cluster;
  - worker nodes = werkpaarden cluster.

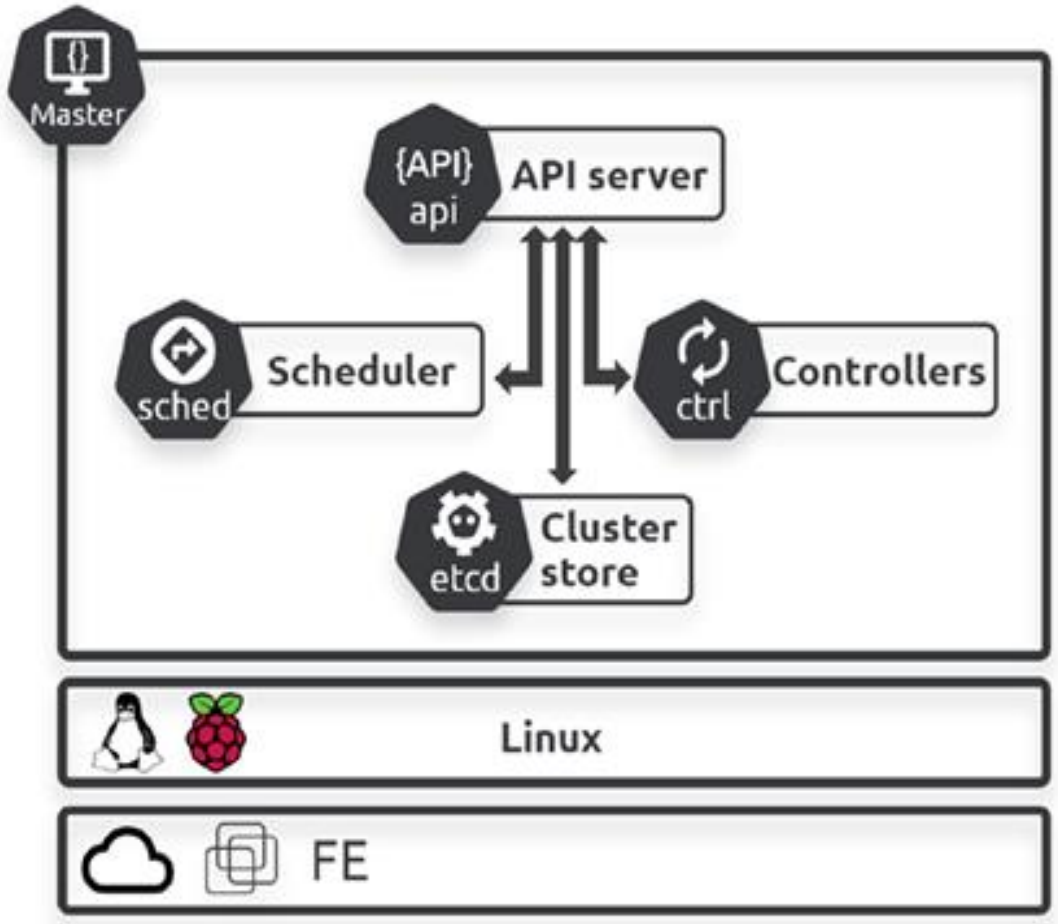
Kubernetes architectuur



Control Planes

- Bevat services die samenwerken
- Meestal 3 of 5
  - Voor HA
    - Betrouwbaarheid is belangrijk
- Control Plane-nodes bevat:
  - API server: centrale aanspreekpunt voor communicatie binnen je cluster (port 6443)
  - Cluster Store (etcd): bewaart config en status van het cluster
  - Scheduler: wijst taken toe aan gezonde nodes
  - Controller Manager: zorgt dat gewenste toestand overeenkomt met werkelijkheid
  - Cloud Controller Manager: koppelt cluster aan cloudproviders

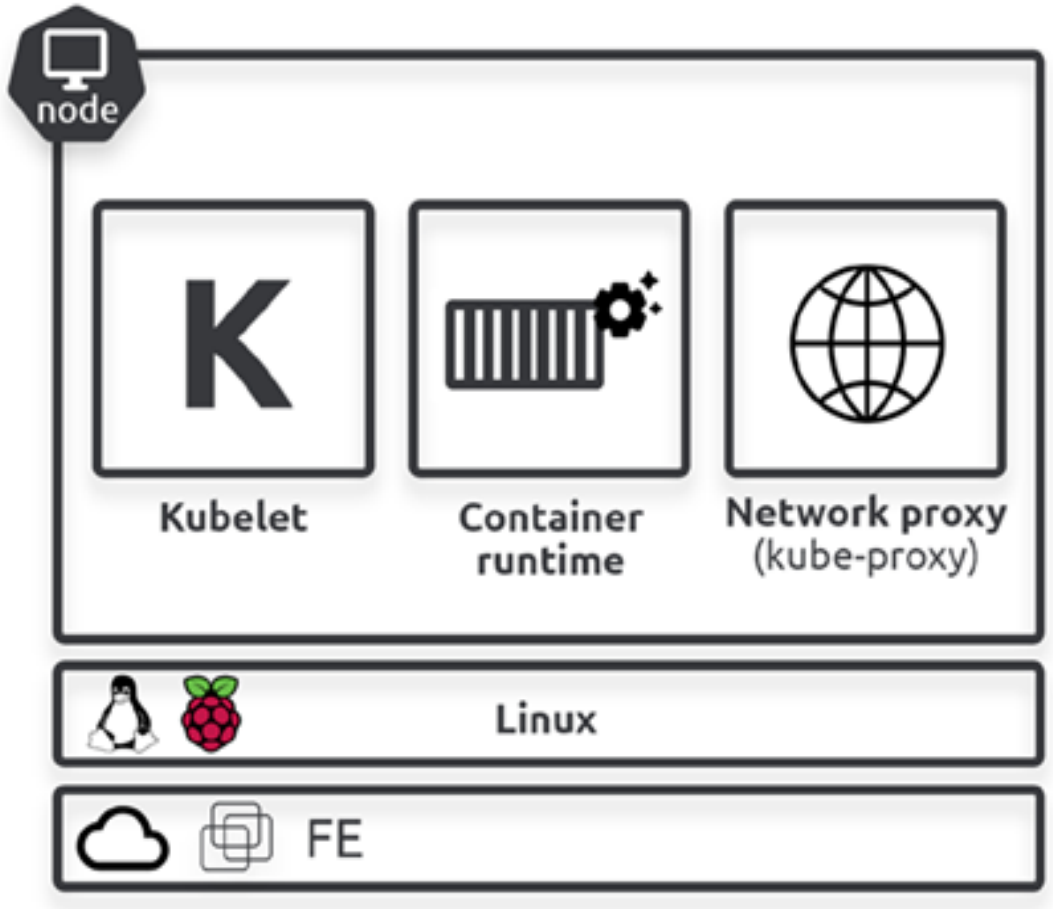
Control Planes



Worker nodes

- Zijn de werkpaarden: hosten en uitvoeren applicaties
- Taken:
  - Continu de Control Plane raadplegen via de API-server voor nieuwe taken
  - Uitvoeren van toegewezen taken
  - Rapporteren status aan de Control Plane
- Worker node bevat:
  - Kubelet: registreert node bij de API server en voert taken uit
  - Container runtime (containerd, CRI-O): start/stop containers
  - Kube-proxy: verantwoordelijk voor de netwerkregels en het afhandelen van netwerkverkeer naar de containers binnen de node

Worker nodes



Kubelet

- Belangrijkste onderdeel worker node
- Automatische installatie en activatie bij toevoegen worker node in cluster
- Taken (draaiend houden van containerized apps)
  - Communicatie met API-server Control Plane
    - Registratie nieuwe node
    - Statusupdates versturen over uitvoering taak
    - Rapporteren als hij taak niet kan uitvoeren
  - Continu verbinding met de API-server
    - Controleren of er nieuwe taken (pods)
      - Zo nieuwe taak => kubelet zorgt voor uitvoering

Container runtime

- Worker nodes moeten containers kunnen uitvoeren
  - Container runtime nodig
    - Images ophalen
    - Starten en stoppen containers
- Welke?
  - Vroeger
    - native ondersteuning docker
  - Nu
    - containerd
      - Lichtgewicht
    - CRI-O
      - Biedt iets meer functionaliteit en bredere compatibiliteit
      - Cloudgestuurd iets populairder

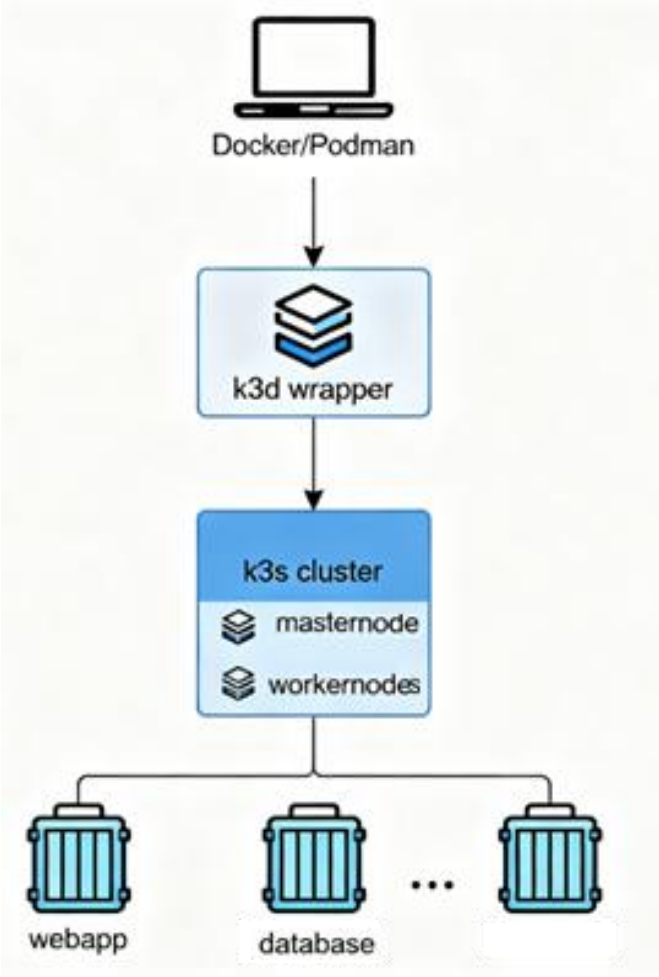
Installatie Kubernetes

- K3s
  - Lightweight Kubernetes voor testen, learning, kleine productie.
  - Kan op 1 machine draaien (wij)

  k3s ideaal om snel een complete maar eenvoudige Kubernetes-omgeving lokaal of in resourcebeperkte situaties op te zetten.
- K3d
  - Wrapper (hulpprogramma) rond k3s om **k3s-clusters in Docker-containers** te draaien.
  - Super eenvoudig om lokaal Kubernetes-clusters te starten
- K8s
  - K8s valt buiten de doelstellingen van de cursus

VM voor k3d

- RHEL 10 VM:
  - Geen GUI.
  - Hostname: virt-K8s-XX (xx zijn je initialen).
  - Gebruiker student met wachtwoord PXL.
  - Statisch IP: 192.168.112.10.
  - Bereikbaar is via SSH.



Installatieprocedure

- K3d werkt met docker
  - RHEL 9
    - Docker ondersteund
  - RHEL 10
    - Docker niet ondersteund => alternatieve repository
    - Podman
      - Gebruik minikube i.p.v. k3d omdat minikube podman beter ondersteunt
- Opmerking: openshift van RHEL: weinig verspreid

Installatieprocedure

- RHEL 9/10
  - Docker installeren
    - Je kan Podman linken aan docker maar af te raden
  - K3d installeren
  - Kubectl installeren
    - Nodig om met cluster te communiceren
- Installatie zelf: zie cursus

Kubernetes-concepten: cluster

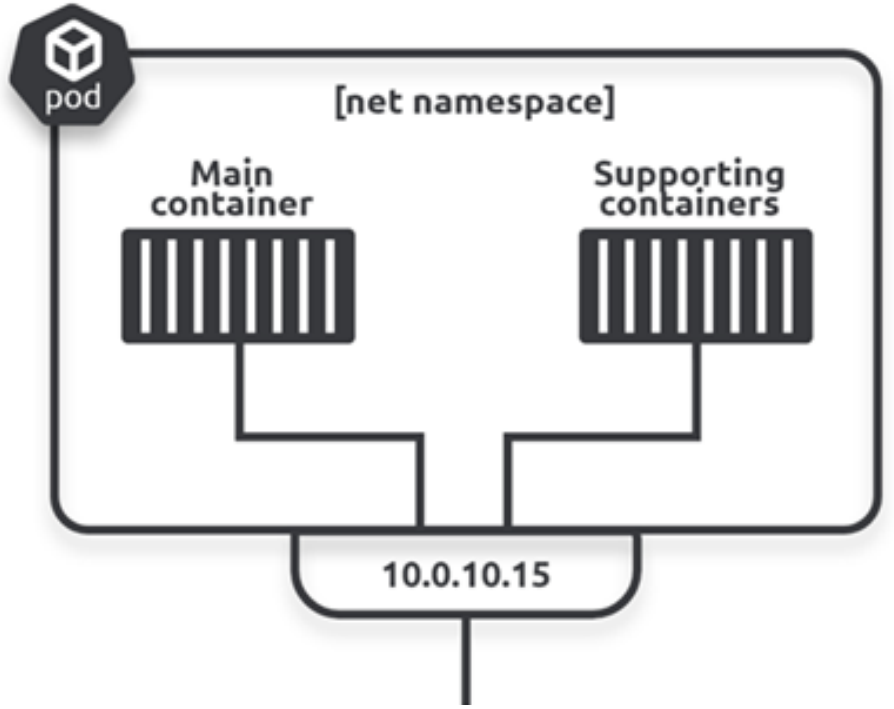
- Wat is het: De volledige Kubernetes-omgeving (bestaat uit nodes).
- Bestanddelen:
  - Master/control plane: regelt planning, scheduling, API, etc.
  - Worker nodes: draaien de containers (Pods).
- Functie: Verzamelt alle compute-resources en beheert workloads centraal.
- Voorbeeld: 1 cluster met 3 worker nodes → 15 pods draaien.

Kubernetes-concepten: pod

- Wat is het: Kleinste uitvoerbare eenheid in Kubernetes.
- Bevat: Eén of meer containers die netwerk en opslag delen.
- Kenmerken:
  - Tijdelijk (ephemeral)
  - Wordt gemaakt en beheerd door een Deployment of Job
- Doel: Een applicatie-component draaien (bijv. één microservice).
- Voorbeeld: Een nginx webserver-container in één pod.

Pods

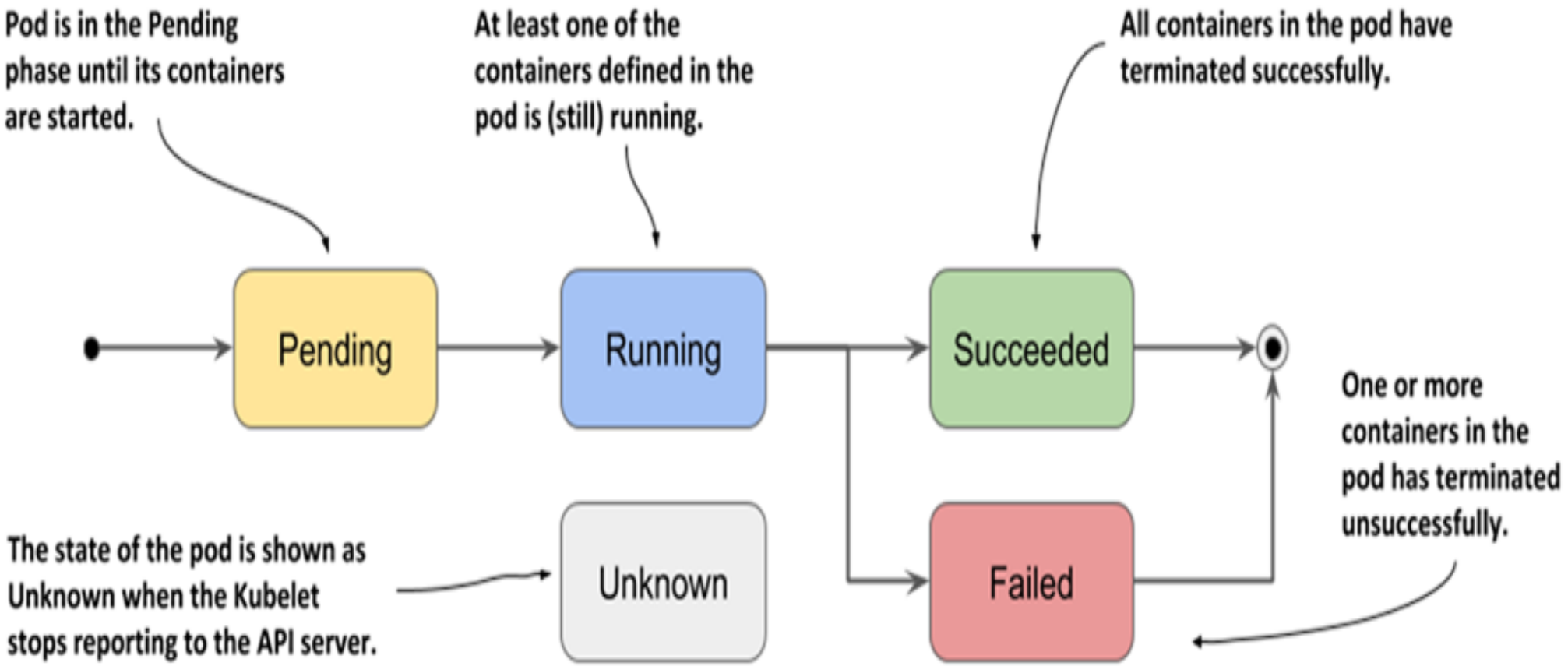
- Afgeschermde omgeving waarin één of meer containers draaien
- Normaal elke applicatie een aparte pod => onafhankelijk schalen en beheren
- Containers delen:
  - De netwerkstack (IP-adres en netwerkinterfaces)
  - Storage volumes (gevolumeerde opslag)
  - Kernel namespace (system-level isolatie)
  - Ze draaien op dezelfde fysieke node



Pods

- Elke pod:eindige levenscyclus
- Status
  - Pending
    - De pod is aangemaakt en de control plane probeert deze op een geschikte, gezonde node op te zetten
  - Running
    - Ten minste één container binnen de pod draait actief en de pod functioneert.
  - Succeeded
    - Alle containers binnen de pod zijn succesvol gestopt (beëindigd).
  - Failed
    - Eén of meer containers zijn ongepland gestopt met een fout.
  - Unknown
    - Status kan tijdelijk niet worden bepaald.

Pods

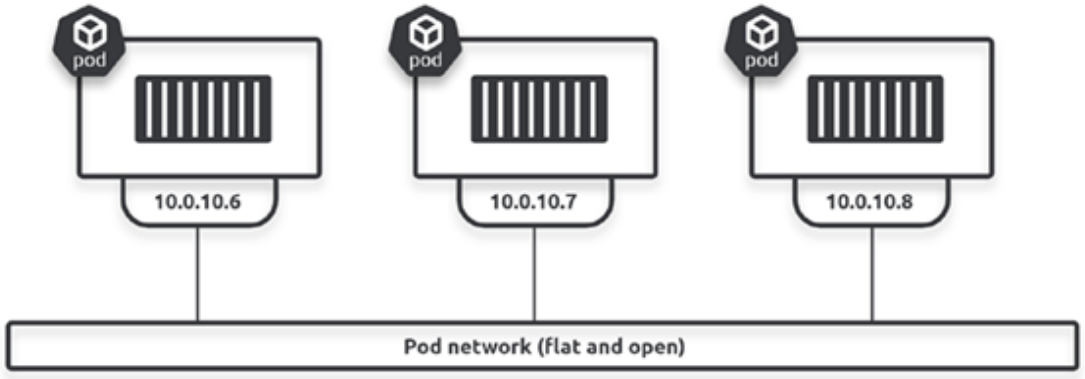

Pod is in the Pending phase until its containers are started.
At least one of the containers defined in the pod is (still) running.
All containers in the pod have terminated successfully.
Pending
Running
Succeeded
One or more containers in the pod has terminated unsuccessfully.
The state of the pod is shown as Unknown when the Kubelet stops reporting to the API server.
Unknown
Failed

Pods

- Pods en IP-adressen
  - Elke Pod krijgt een eigen IP-adres, afkomstig uit het Pod CIDR-bereik van het cluster.
  - Alle containers binnen dezelfde Pod delen:
    - hetzelfde IP-adres
    - dezelfde netwerkstack (localhost & poorten)
- Plat en open netwerk binnen cluster
  - Het Kubernetes-netwerk is standaard:
    - Plat: geen NAT tussen Pods
    - Open: alle Pods kunnen met elkaar communiceren
  - Communicatie is node-onafhankelijk — Pods kunnen elkaar bereiken ongeacht op welke node ze draaien.

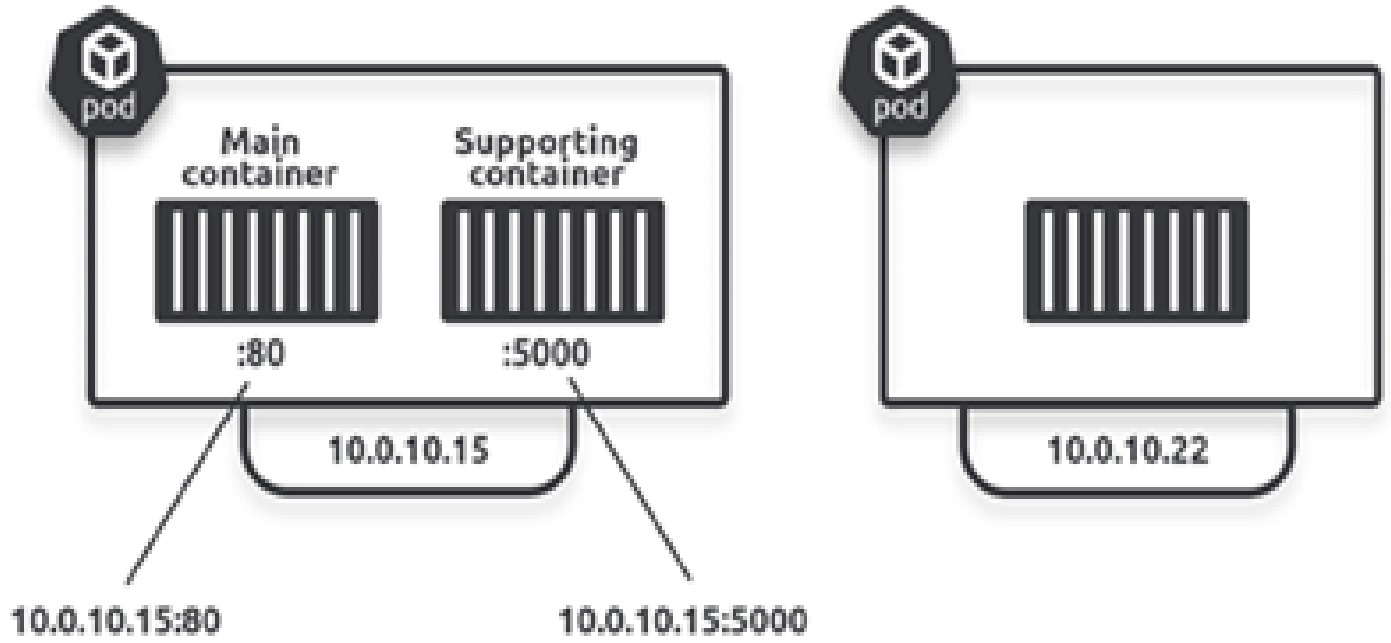
Pods

- Beveiliging met Network Policies
  - Network Policies bepalen welke Pods met elkaar mogen communiceren.
  - Kunnen verkeer beperken op basis van:
    - Pod-labels
    - Namespace
    - Poorten /protocollen
  - Verhogen de veiligheid binnen het cluster door verkeer te isoleren.



Pods

- Multicontainer pod
  - Op niveau pod met poorten werken om andere container te bereiken



Pod aanmaken

- Op 2 manieren:
  - Imperatief
    Een pod aanmaken door het invoeren van commando’s.
  - Declaratief
    Een pod (of andere objecten) aanmaken door het gebruik van een manifest bestand.

Pod aanmaken: voorbeeld imperatief

```
student@serverXX:~$ kubectl run hello-pod --image=nginx --
restart=Never --port=80 --labels="zone=prod,version=v1"
```

Pod aanmaken: voorbeeld declaratief

```
kind: Pod                  # Geeft aan welk Kubernetes-object dit is — hier een 'Pod'.
apiVersion: v1               # De API-versie die wordt gebruikt; 'v1' is de stabiele versie voor Pods.
metadata:                  # Metagegevens over de Pod (zoals naam, labels, enz.).
  name: nginx-pod2           # De naam van de Pod — hiermee kun je hem identificeren in Kubernetes.
  labels:                  # Labels zijn key-value paren om de Pod te groeperen of selecteren.
    zone: prod               # Een label dat aangeeft dat deze Pod in de 'prod'-zone hoort.
    version: v1              # Een label dat aangeeft dat dit versie 'v1' van de Pod is.
spec:                      # De specificatie van de Pod — wat er binnen de Pod draait.
  containers:                # Een lijst van containers die in deze Pod draaien.
    - name: nginxcont        # De naam van de container binnen de Pod.
      image: nginx           # De container gebruikt de 'nginx'-image van Docker Hub.
      ports:                 # Geeft aan welke poorten deze container gebruikt.
        - containerPort: 80   # De container luistert op poort 80 — standaard voor HTTP-verkeer.
```

Kubectl

- Dit is het standaard tool voor het werken met een K8s cluster.
  - kubectl <command> <type> <name> <flags>
    - command
      Commando of operatie, bv apply, create, delete, get
    - type
      Het type van de resource of object
    - name
      De naam van de resource of object
    - Flags
      Optionele vlaggen

Kubernetes-concepten: namespace

- Wat is het: Logische scheiding binnen het cluster.
- Doel:
  - Resources organiseren (bijv. dev, test, prod)
  - Toepassen van resource quotas en role based access control
- Voordeel: Meerdere teams/omgevingen kunnen op één cluster draaien zonder elkaar te storen.
- Voorbeeld: kubectl create namespace dev

Namespaces

- Standaard: 4 namespaces

default	De standaard namespace. Als je geen -n opgeeft bij een kubectl-commando, gebruikt Kubernetes automatisch deze namespace. Alle “gewone” resources komen hier terecht, tenzij je anders specificeert.
kube-system	Hier draait de infrastructuur van Kubernetes zelf, zoals de API-server, kube-dns (CoreDNS), scheduler, controller-manager, enz. Je hoort hier meestal niet zelf applicaties te plaatsen.
kube-public	Bevat publieke informatie, zoals cluster-informatie, die voor iedereen toegankelijk is (wordt zelden gebruikt).
kube-node-lease	Wordt intern door Kubernetes gebruikt om node leases bij te houden. Dit helpt de control plane snel te detecteren of een node nog leeft. (Elke node krijgt hier een "lease object").

Kubernetes-concepten: service

- Wat is het: Toegangspunt (virtueel IP) voor één of meer Pods.
- Doel:
  - Stabiele netwerktoegang tot Pods die kunnen verdwijnen/herstarten
  - Load balancing over meerdere replicas
- Types:
  - ClusterIP (intern)
  - NodePort (extern via node)
  - LoadBalancer (cloud)
- Voorbeeld: Service die alle Pods met label app=web bereikbaar maakt op poort 80.

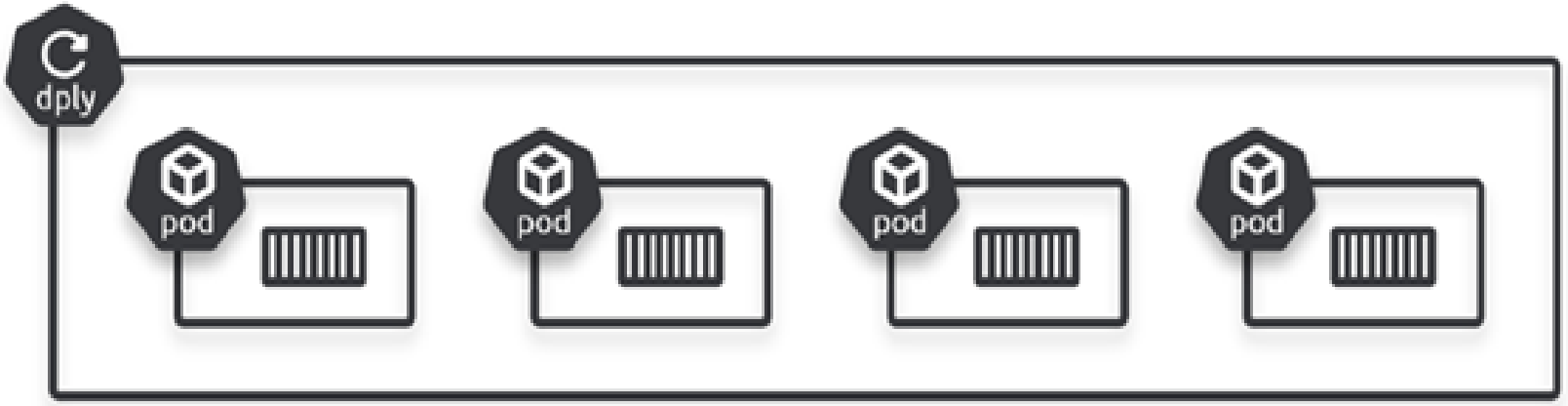
Services

- Kubernetes-netwerk:
  - Standaard plat en open → alle pods kunnen elkaar direct bereiken.
  - Communicatie via IP-adres is mogelijk, maar niet betrouwbaar.
- Probleem 1 – Veranderende IP's
  - De website-pod maakt verbinding met de database-pod.
  - Deze verbinding gebruikt het IP of DNS van de pod.
  - Bij herstart of scaling krijgt de database-pod een nieuw IP.
  - Resultaat: de website verliest de verbinding → onstabiele communicatie.
- Probleem 2 – Scaling van pods
  - De website wordt geschaald naar meerdere pods voor betere prestaties.
  - Elke pod heeft zijn eigen IP-adres.
  - Er is geen vaste toegangspoort naar de groep pods.

- Oplossing
  We hebben een mechanisme nodig om pods collectief en stabiel te benaderen — één vast aanspreekpunt = een Kubernetes Service.

Services

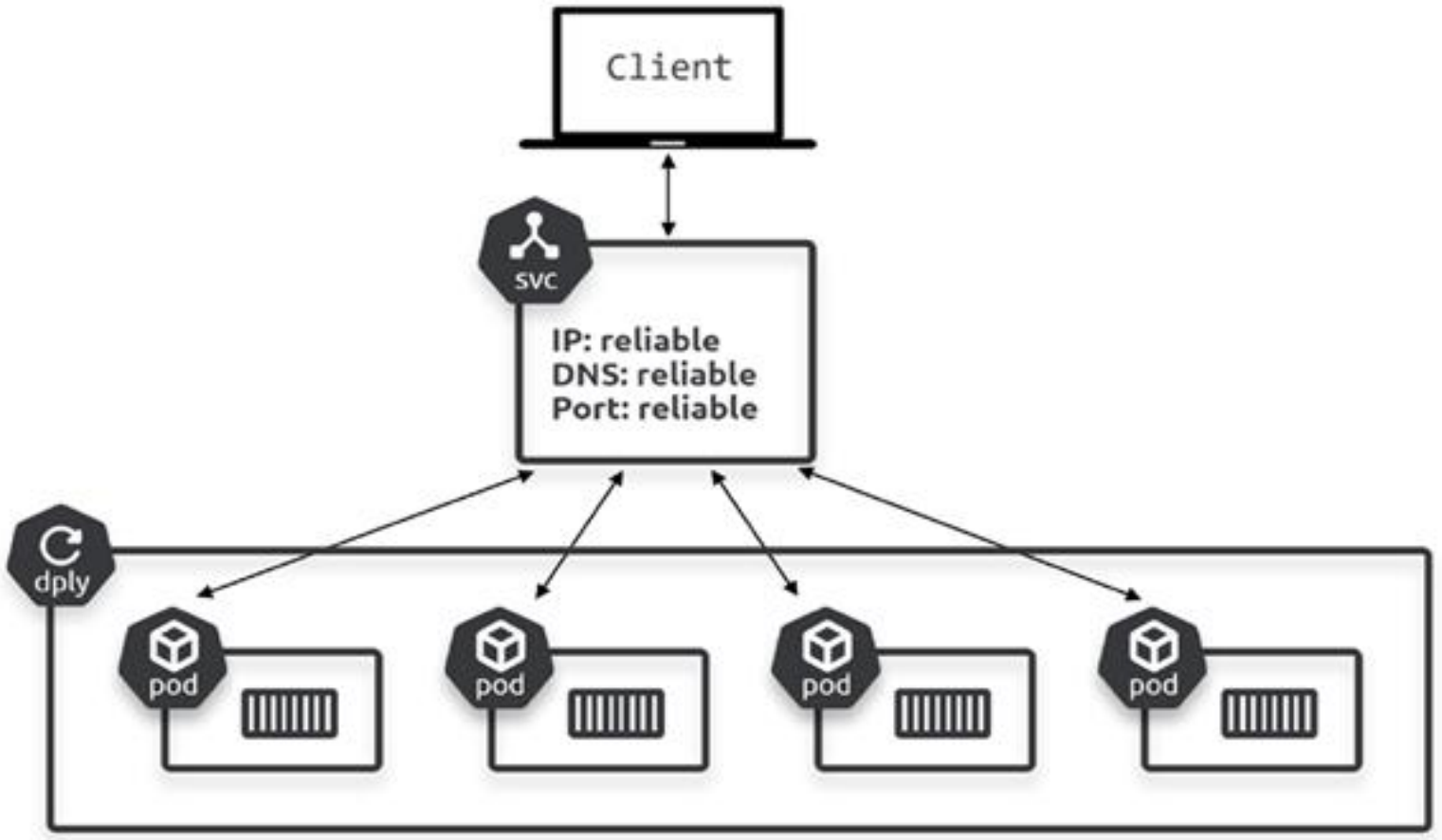
¯_(ツ)_/¯



Services

- Een Service fungeert als een vast netwerkendpoint voor een groep pods met dezelfde functie
  - Onafhankelijk van de levenscyclus van individuele pods
  - Biedt stabiele toegang tot dynamische workloads
- Eigenschappen van een Service
  - Een vast IP-adres
  - Een consistente DNS-naam
  - Één of meerdere TCP- of UDP-poorten waarlangs service bereikbaar is

Services


Client
svc
IP: reliable
DNS: reliable
Port: reliable
dply
pod
pod
pod
pod

Services

- Service zorgt voor load balancing.
  - Alle pods die gekoppeld zijn aan een Service worden continu gemonitord.
    - Als een pod niet meer gezond is, zal de Service deze automatisch uitsluiten van de load balancing, zodat alleen gezonde pods verkeer ontvangen.

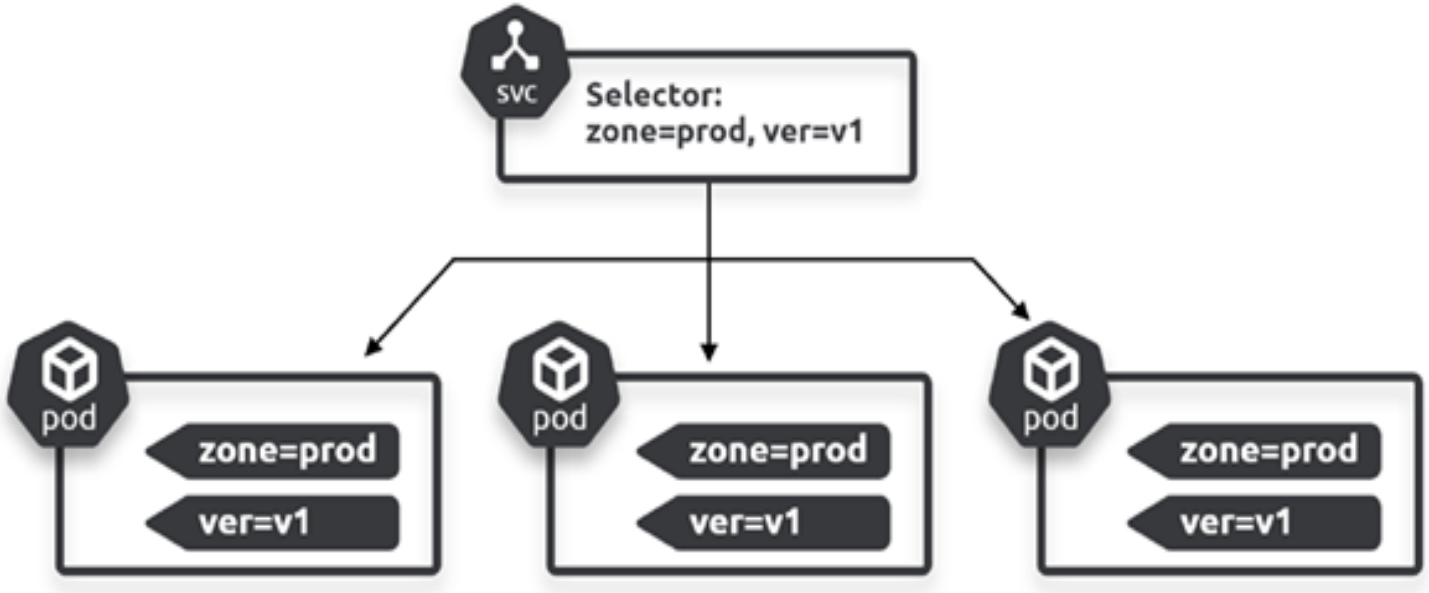
Type	Toegang vanaf	Gebruik
ClusterIP	Alleen binnen het cluster	Standaardtype, voor interne communicatie tussen pods en services
NodePort	Buiten het cluster via node-IP en poort	Eenvoudige externe toegang, vaak voor lokale of testomgevingen
LoadBalancer	Buiten het cluster via extern IP	Cloud-native toegang met geïntegreerde load balancer (bv. AWS, GCP, Azure)

Endpoint slices

- Service = Load Balancer
  - Verdeelt verkeer over gezonde pods.
  - Bepaalt targets via label selector.
- EndpointSlices
  - Bevatten subsets van IP’s en poorten van pods die bij de Service horen.
  - Worden automatisch aangemaakt bij het creëren van een Service.
  - Control Plane houdt ze up-to-date:
    - Nieuwe pods → toegevoegd
    - Niet-gezonde of verwijderde pods → verwijderd
  - Alleen Ready pods worden opgenomen.

Services aanmaken

- Wij: enkel declaratief
- In Kubernetes koppelen we altijd pods aan een Service, nooit andersom.
  - Aan de hand van labels (zie vorige slide)



Services aanmaken

- Koppeling
  - manifestfile van een pod bevat labels: laten overeenkomen met labelselector van een Service => load balancing
  - In dit geval:
    - Labelselector service: zone=prod, ver=v1.
    - Labels pod: zone=prod, ver=v1

Voorbeeld: service – clusterIP (service)

```
apiVersion: v1          # Geeft aan dat dit object gebruikmaakt van de 'v1' API-versie van Kubernetes.
kind: Service           # Definieert het type Kubernetes-resource: hier een 'Service'.
metadata:
  name: nginx-svc       # De naam van de Service.
spec:
  type: ClusterIP       # Het servicetype. 'ClusterIP' betekent dat de Service alleen binnen het cluster bereikbaar is.
  selector:
    app: nginx          # Labelselector: koppelt de Service aan pods met het label 'app=nginx'.
  ports:
    - port: 8080        # De poort waarop de Service zelf luistert (binnen het cluster).
      targetPort: 80    # De poort op de doel-pods waar het verkeer naartoe wordt gestuurd.
      protocol: TCP     # Het gebruikte netwerkprotocol; standaard is dit 'TCP'.
```

Voorbeeld: service – clusterIP (bijhorende pod)

```
apiVersion: v1          # Gebruikt de 'v1'-API-versie van Kubernetes voor het aanmaken van een Pod.
kind: Pod               # Geeft aan dat dit object een Pod is.
metadata:
  name: nginx-pod       # De naam van de Pod; hiermee wordt het object uniek geïdentificeerd in het cluster.
  labels:
    app: nginx          # Label dat aan de Pod wordt toegekend; handig voor selectie door Services of Deployments.
spec:
  containers:           # Beschrijving van de containers die binnen deze Pod draaien.
    - name: nginx       # De naam van de container binnen de Pod.
      image: nginx      # De containerimage die wordt gebruikt — hier de standaard Nginx-webserver.
      ports:
        - containerPort: 80  # De poort binnen de container waarop Nginx luistert (standaard HTTP-poort).
```

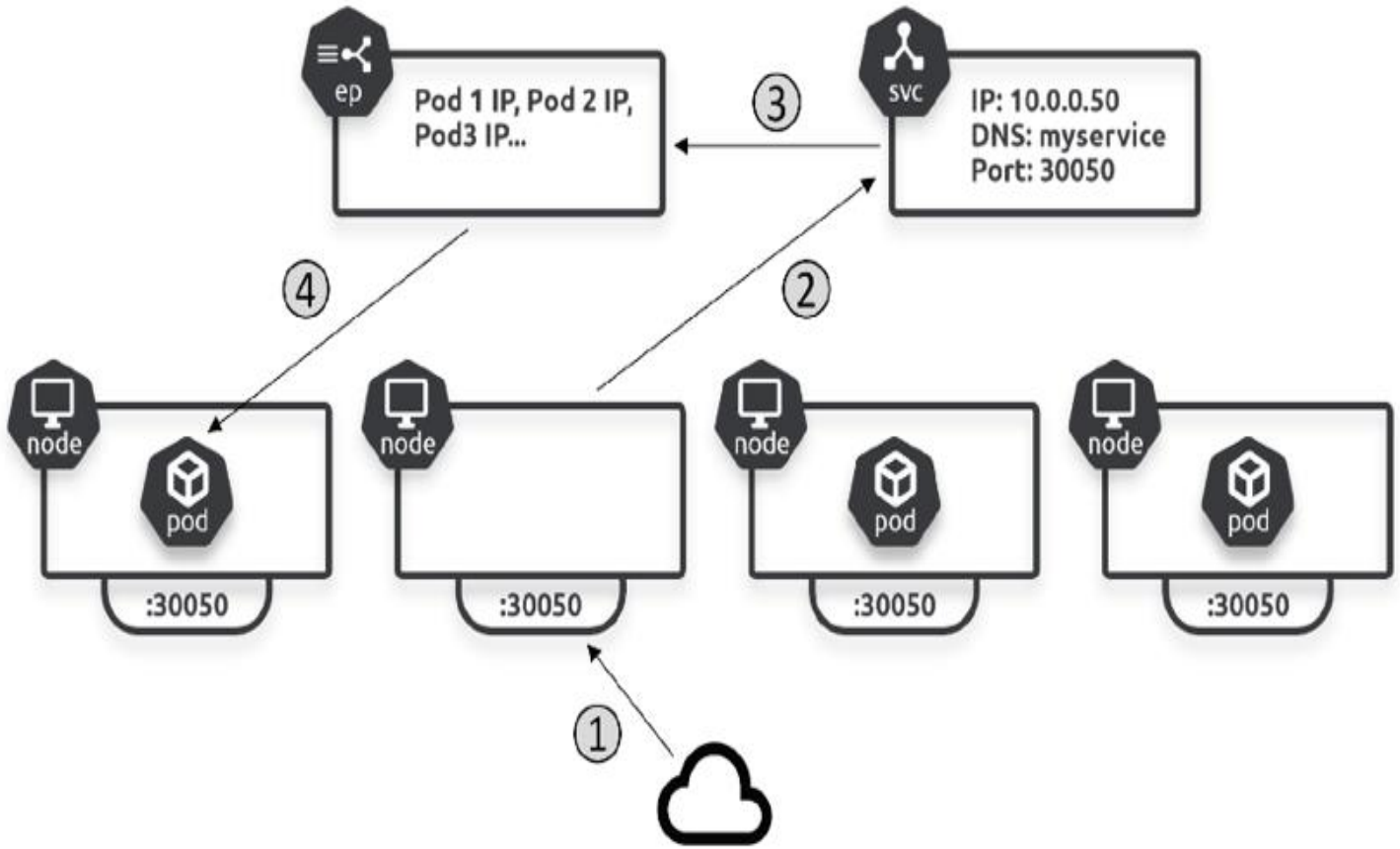
Voorbeeld: service – clusterIP (gewone pod)

```
apiVersion: v1              # Geeft aan dat de Pod wordt aangemaakt met de 'v1' API-versie van Kubernetes.
kind: Pod                   # Definieert het type Kubernetes-resource: hier een 'Pod'.
metadata:
  name: debian-pod          # De naam van de Pod; hiermee wordt het object uniek geïdentificeerd in het cluster.
spec:
  containers:               # Lijst van containers die binnen deze Pod draaien.
  - name: debian            # De naam van de container binnen de Pod.
    image: debian:latest    # De containerimage die gebruikt wordt: de nieuwste Debian Linux-versie.
    command: ["sleep", "infinity"]  # houdt de container oneindig actief.
    tty: true               # Schakelt een interactieve terminal in (TTY), zodat je via `kubectl exec` kunt inloggen.
```

Voorbeeld: service – NodePort

```
apiVersion: v1              # Versie van de Kubernetes API
kind: Service               # Type resource: Service
metadata:
  name: nginx-srv2          # Naam van de Service
spec:
  type: NodePort            # Type Service, bereikbaar via node-poort
  selector:
    app: nginx              # Selecteert Pods met label app=nginx
  ports:
    - port: 8080            # Poort waarop de Service bereikbaar is in het cluster
      targetPort: 80        # Poort waarop de Pods luisteren
      nodePort: 30050       # Poort op de nodes voor externe toegang
```

Voorbeeld: service



Kubernetes-concepten: deployment

- Wat is het: Controller die Pods beheert en updates uitvoert.
- Doel:
  - Declaratief aantal replicas instellen
  - Rolling updates & rollbacks
- Voorbeeld: replicas: 3 → altijd 3 identieke Pods online

Replicasets

- Deployments gebruiken ReplicaSets om het gewenste aantal pods te beheren
- ReplicaSets zorgen ervoor dat altijd het juiste aantal werkende pods actief is:
  - als er één uitvalt, wordt er automatisch een nieuwe gestart
    - self-healing en schalen mogelijk

Voorbeeld deployment

```
apiVersion: apps/v1          # De API-versie die gebruikt wordt; 'apps/v1' is nodig voor Deployments.
kind: Deployment             # Geeft aan dat dit bestand een Deployment definieert.
metadata:
  name: nginx-deployment     # De naam van de Deployment binnen het cluster.
spec:
  replicas: 2                # Aantal identieke Pods dat moet draaien (in dit geval 2).
  selector:
    matchLabels:
      app: nginx             # Selecteert Pods met het label 'app: nginx'.
```

Voorbeeld deployment

```
template:                # De sjabloon die bepaalt hoe de Pods eruitzien.
   metadata:
     labels:
       app: nginx          # Label dat aan de Pods wordt toegekend (moet overeenkomen met de selector hierboven).
   spec:
     containers:          # Hier definieer je welke containers in elke Pod draaien.
       - name: nginx       # Naam van de container.
         image: nginx:latest # De containerimage die gebruikt wordt, hier de nieuwste versie van Nginx.
         ports:
           - containerPort: 80   # De poort binnen de container die wordt geopend (standaard HTTP-poort).
```

Voorbeeld deployment

- Apart manifest voor Pod niet nodig als je al een deployment manifest hebt
  - Deployment bevat al een pod-template dat beschrijft hoe elke pod eruit moet zien.
  - Kubernetes gebruikt deze template om automatisch de juiste Pods te creëren en te beheren

Voorbeeld: rolling updates

```
apiVersion: apps/v1          # API-versie die gebruikt wordt voor een Deployment (apps/v1 is de juiste versie).
kind: Deployment             # Geeft aan dat dit een Deployment-object is.
metadata:                    # Metadata bevat informatie over het object (zoals naam en annotaties).
  name: nginx-deployment     # De naam van de Deployment in het cluster.
  annotations:               # Extra metadata die niet van invloed is op de werking, maar info toevoegt.
    kubernetes.io/change-cause: "Initial release"  # Beschrijft waarom deze versie is uitgerold
```

Voorbeeld: rolling updates

```
spec:
  replicas: 10              # Geeft aan dat er 10 identieke Pods moeten draaien.
  selector:                 # Bepaalt welke Pods bij deze Deployment horen.
    matchLabels:
      app: nginx            # Pods met dit label worden door de Deployment beheerd.
  revisionHistoryLimit: 5     # Het aantal oude ReplicaSets dat Kubernetes bewaart (voor rollback).
  progressDeadlineSeconds: 300  # Maximale die een update mag duren voordat K8s een fout meldt.
  minReadySeconds: 10         # Tijd dat een nieuwe Pod ‘Ready’  voor stabiel beschouwd.
  strategy:                 # Geeft aan hoe updates van de Pods uitgevoerd worden.
    type: RollingUpdate       # ‘RollingUpdate’ betekent: geleidelijk nieuwe Pods uitrollen zonder downtime.
    rollingUpdate:
      maxUnavailable: 1       # Maximaal 1 Pod mag tegelijk onbeschikbaar zijn tijdens een update.
      maxSurge: 1           # Maximaal 1 extra Pod bovenop de gewenste 10 mag tijdelijk worden gemaakt bij update.
```

Voorbeeld: rolling updates

```
Spec: (vervolg)
  template:
    metadata:
      labels:
        app: nginx          # Label dat aan de Pods wordt toegekend (moet overeenkomen met selector hierboven).
    spec:
      containers:
        - name: nginx       # Naam van de container binnen de Pod.
          image: nginx:1.24   # De gebruikte container image (specifieke versie 1.24 van Nginx).
          ports:
            - containerPort: 80   # De poort die binnen de container wordt geopend (HTTP).
```

Kubernetes-concepten: storage

- Container filesystem
  - Tijdelijk, verdwijnt bij container- of Pod-herstart
- Tijdelijke volumes (ephemeral)
  - emptyDir: gedeelde tijdelijke map, verdwijnt bij Pod-stop
  - configMap: alleen-lezen configuratiebestanden
  - Secret: beveiligde, alleen-lezen gevoelige data
  - downwardAPI: Pod-metadata als bestanden
- Persistente opslag
  - PersistentVolume (PV): opslag onafhankelijk van Pods
  - PersistentVolumeClaim (PVC): verzoek om opslag
  - StorageClass: regels voor dynamische provisioning

Voorbeeld1: opslag container filesystem

```
apiVersion: v1                # De API-versie 'v1' is voor basisobjecten zoals Pods.
kind: Pod                     # Geeft aan dat dit object een Pod is.
metadata:                     # Metadata bevat informatie over de Pod, zoals naam, labels, etc.
  name: temp-data-demo        # De naam van de Pod is 'ephemeral-pod'.
spec:                         # 'spec' beschrijft de specificatie van de Pod: wat erin moet draaien.
  containers:                 # Lijst van containers die in de Pod draaien (één of meerdere).
  - name: app                 # De naam van deze container is 'app'.
    image: busybox            # De container gebruikt het 'busybox'-image (een lichte Linux-toolbox).
    command: ['sh', '-c', 'echo "data" > /tmp/file.txt && sleep 3600']
                              # De container voert een shell-commando uit:
                              # 1. 'echo "data" > /tmp/file.txt' maakt een bestand '/tmp/file.txt' met de tekst "data".
                              # 2. 'sleep 3600' houdt de container 3600 seconden (1 uur) actief
```

Voorbeeld2: tijdelijk volume

```
apiVersion: v1                    # De API-versie van Kubernetes die wordt gebruikt; 'v1' is voor basisobjecten zoals Pods.
kind: Pod                         # Geeft aan dat dit object een Pod is.
metadata:                         # Metadata bevat informatie over de Pod.
  name: pod-with-volume           # De naam van de Pod is 'pod-with-volume'.
spec:                             # 'spec' beschrijft de specificatie van de Pod.
  containers:                     # Lijst van containers die in deze Pod draaien.
  - name: app                     # Naam van de container is 'app'.
    image: busybox                # De container gebruikt het lichte 'busybox'-image.
    command: ['sh', '-c', 'echo "data" > /data/file.txt && sleep 3600']
                                  # De container voert een shell-commando uit:
                                  # 1. Schrijft de tekst "data" naar /data/file.txt.
                                  # 2. Slaapt 3600 seconden (1 uur), zodat de Pod actief blijft.
```

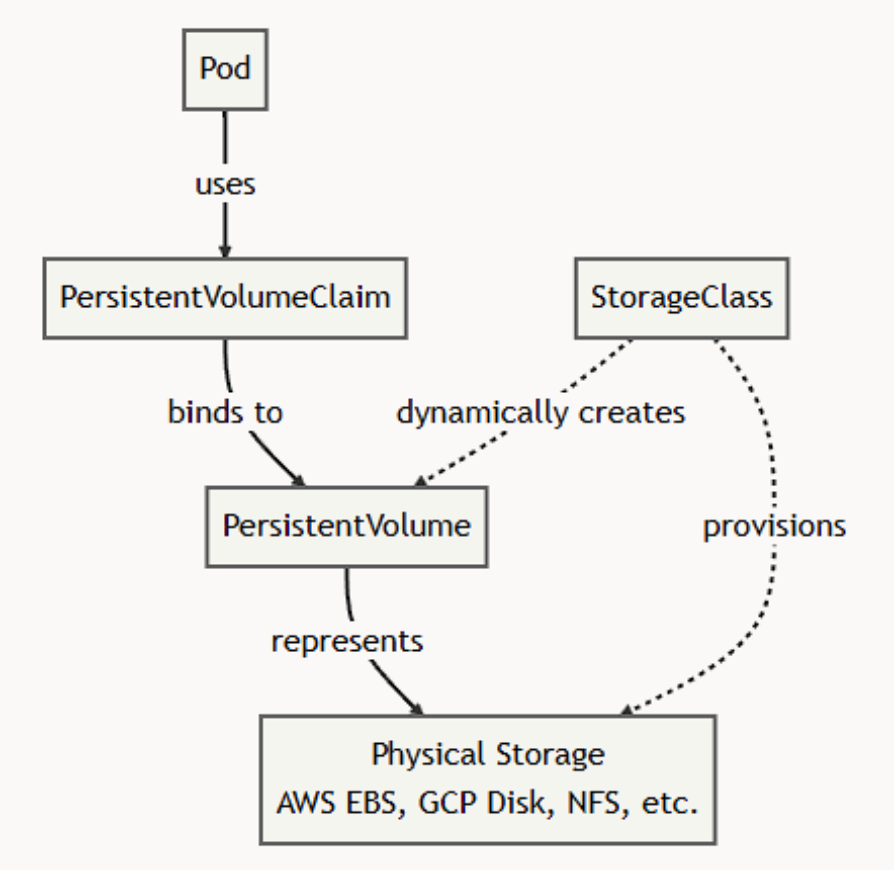
Voorbeeld2: tijdelijk volume

```
spec: (vervolg)
    volumeMounts:                 # Hiermee wordt een volume gekoppeld (mount) binnen de container.
    - name: my-volume             # De naam van het volume dat wordt gekoppeld.
      mountPath: /data            # Het pad in de container waar het volume zichtbaar wordt (hier: /data).
  volumes:                        # Definieert welke volumes beschikbaar zijn voor de Pod.
  - name: my-volume               # De naam van het volume (moet overeenkomen met volumeMounts.name).
    emptyDir: {}                  # Maakt een leeg directoryvolume aan dat leeft zolang de Pod bestaat.
                                  # Dit volume wordt aangemaakt wanneer de Pod start en verwijderd als de Pod wordt verwijderd.
                                  # Het volume blijft bestaan bij container-herstart, maar niet bij Pod-herstart.
```

Persistente opslag

- Componenten:
  - PV (PersistentVolume): fysieke opslagresource
  - PVC (PersistentVolumeClaim): aanvraag door Pod
  - StorageClass: bepaalt provisioning (cloud, NFS, etc.)
- Gebruik: Voor databases, logs, of uploads die niet mogen verdwijnen bij pod-restart.
- Voorbeeld: PVC vraagt 10 Gi opslag met ReadWriteOnce.

Persistant storage



PersistentVolume (PV)

```
apiVersion: v1                # API-versie voor het object
kind: PersistentVolume        # Type Kubernetes-object
metadata:
  name: my-pv                 # Naam van de PV in het cluster
spec:
  capacity:
    storage: 10Gi             # Beschikbare opslaggrootte
  accessModes:
    - ReadWriteOnce           # Toegangsmodus: één Node kan lezen/schrijven
  persistentVolumeReclaimPolicy: Retain  # Wat er gebeurt als PV losgekoppeld wordt
  hostPath:
    path: /mnt/data           # Fysieke opslaglocatie op de node
```

Persistant Volume Claim (PVC)

```
apiVersion: v1                  # API-versie voor het object
kind: PersistentVolumeClaim     # Type Kubernetes-object: verzoek om een PV
metadata:
  name: my-pvc                  # Naam van de PVC in het cluster
spec:
  accessModes:
    - ReadWriteOnce             # Gewenste toegang: één Node kan lezen/schrijven
  resources:
    requests:
      storage: 8Gi              # Gewenste hoeveelheid opslag (8 Gigabyte)
```

Pod die gebruik maakt van PVC

```
apiVersion: v1              # API-versie voor het object
kind: Pod                   # Type Kubernetes-object: een Pod
metadata:
  name: my-pod              # Naam van de Pod in het cluster
spec:
  containers:
  - name: app               # Naam van de container binnen de Pod
    image: nginx            # Container image die gebruikt wordt
    volumeMounts:
    - name: storage         # Naam van het volume dat gemount wordt
      mountPath: /usr/share/nginx/html  # Pad in de container waar het volume beschikbaar is
  volumes:
  - name: storage
    persistentVolumeClaim:
      claimName: my-pvc     # Naam van de PVC die aan dit volume gekoppeld is
```

Storage Class

- Storage Class maakt mogelijk dat Persistent Volumes automatisch wordt aangemaakt bij Persistant Volume Claim
  - Veel gebruikt in cloudomgevingen
  - Bij K3s is er al een StorageClass
    - Local-path

Access mode

- Persistent Volumes kunnen drie verschillende toegangsniveaus (access modes) hebben:
  - ReadWriteOnce (RWO)
    Volume kan gemount worden in Read-Write mode, maar enkel door 1 node
  - ReadOnlyMany (ROX)
    Volume kan gemount worden als Read-Only door meerdere nodes
  - ReadWriteMany (RWX)
    Volume kan gemount worden als Read-Write door meerdere nodes

Reclaim policies

- Bij het verwijderen van een PersistentVolume (PV) bepaalt de persistentVolumeReclaimPolicy wat er met de bijbehorende storage gebeurt.
  - Delete: de storage wordt automatisch verwijderd wanneer de PV wordt verwijderd (standaardoptie voor PV's die via een StorageClass automatisch worden aangemaakt).
  - Retain: de storage blijft bestaan, ook nadat de PV is verwijderd.
- In PV

  persistentVolumeReclaimPolicy: Retain

Kubernetes-concepten: samenvatting