kubernetes (revisited)

containerorkestratie

Wat is kubernetes

Definitie (wikipedia): "Kubernetes, ook bekend als K8s, is een open-source systeem voor het automatiseren van de implementatie, het schalen en het beheer van ***gecontaineriseerde*** applicaties"

Ook wel: tooling voor container management (orkestratie)

Oorspronkelijk van Google (ontstaan uit Google's *Borg*)

Nu onderdeel van de Cloud Native Computing Foundation (CNCF)

Website: kubernetes.io

Niet alles ingebouwd:

- Container Runtime Interface (CRI) plugin - was docker, tegenwoordig *containerd*
- Container Network Plugin (CNI) plugin - networking ook "uitbesteed"
- Container Storage Interface (CSI) plugin - vooral voor Persistent storage

Waarom kubernetes

Voldoet aan de ontwikkeling van *monolith* systemen naar *microservices*:

- containers zijn zeer geschikt voor het runnen van allerlei microservices (payment-services, authenticatie-service, parkeer-service, …)
- containers moeten gemanaged worden: dat doet kubernetes voor je

Voordelen:

- high availability ("geen" downtijd)
- schaalbaarheid en performance
- "runs local, runs everywhere"

Nadelen:

- introduceert complexiteit
- niet voor alles geschikt

Kenmerk	Kubernetes	Docker Compose	Docker Swarm
Primair doel	Orkestratie en beheer op productie-schaal	Lokale multi-container setup	Simpele container orkestratie op kleine schaal
Schaalbaarheid	Zeer goed (cloud-native, auto-scaling, multi-node)	Niet bedoeld voor schaalvergroting	Beperkt (vergeleken met Kubernetes)
High Availability	Ja (met replica's, failover, etc.)	Nee	Ja, maar beperkter
Load Balancing	Ja (Service + kube-proxy, Ingress, etc.)	Nee	Ja, ingebouwd
State Management	Declaratief via YAML, zelfherstellend	Imperatief	Declaratief via services
Networking	CNI-plugins, policies, namespaces, DNS	Simpel (bridge/networks)	Ingebouwd overlay netwerk
Storage	Persistent Volumes + CSI	Bind mounts / named volumes	Volumes gedeeld tussen nodes (beperkt)
Extensibility	Hoog (CRDs, Operators, eBPF, etc.)	Laag	Beperkt
Monitoring/Logging	Ingebouwde integraties (Prometheus, Fluentd, etc.)	Zelf toevoegen	Beperkt
Installatie	Complexer (met kubeadm, k3s, MicroK8s, etc.)	Zeer eenvoudig (docker-compose up)	Makkelijk (docker swarm init)
Learning curve	Hoog	Zeer laag	Relatief laag

chatgpt

Veel smaken

- Mogelijkheid tot deployen van een kubernetes cluster bij een **cloud provider** (bv.: Hetzner heeft een mooi voorbeeld om dat te doen met Terraform)
- **kind** - **K**ubernetes **in D**ocker. Goed voor testen (o.a. CI/CD), lokaal
- **k3s** - lightweight, door "Rancher labs" (nu CNCF) . Goed voor IoT of Edge devices
- **microk8s** - ook een kleine footprint, redelijk eenvoudig, van Canonical (CNCF gecertificeerd). Ook voor wat grotere omgevingen.
- Setup k8s **van scratch** (met kubeadm, zie LUGN66 ;-))

Onderdelen van kubernetes

- linux namespaces - scheiden van "views" (bv. hostname, netwerk, mount-points, pid-nummers)
- cgroups - scheiden van resources (cpu, memory, disk)
- containers - runnen binnen pods. Vaak 1 container per pod
- pods - kleinste *deployable* eenheid binnen kubernetes
- deployment - declaratieve techniek voor het beheren van pods
- replicaset - bepaald aantal identieke pods, beheerd door een deployment
- services - voor toegang tot (een set van) pods middels IP-adres
- kubernetes namespaces
- networking (CNI plugin)
- configmaps - meegeven van configuratie
- secrets - meegeven van gevoelige configuratie
- storage - Persistent Volumes (PV) - straks meer
- service discovery - ip -> dns namen mbv. CoreDNS
- scheduler - scheduled pods op nodes
- controller - bewaakt de staat van je cluster

Linux namespaces en cgroups

Linux namespaces en cgroups bepalen respectievelijk de "omgeving" en "resources (limits)" van een proces (container..)

Namespaces zijn o.a.

- mnt - view filesysteem
- utc - hostname
- pid - pid nummering
- net - network stack

Normaal: kind proces verbonden met namespace en cgroups van parent proces

Een container: eigen namespaces en eigen cgroups

cgroups: memory, cpu, disk usage

Meer weten? -> https://atcomputing.nl/blog/open-source-summit-2022.html

containers en pods

container:

- Proces met eigen namespaces (bv. hostname, netwerk, mount-points, pid-nummers, filesystem, users) en eigen cgroups (resource usage)

pod:

- Kleinste te managen eenheid binnen kubernetes
- Collectie van 1 of meer containers (vaak 1)
- I.e.g. eigen netwerk namespace, eigen hostname
- Meerdere pods kunnen dezelfde port hebben
- Alle pods op alle nodes kunnen elkaar vinden.

Deployments

Definitie: Een Deployment beheert een set van pods om een applicatie uit te voeren.

Beschrijving met een YAML manifest:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: oscarbuse/k8sflaskapp
        ports:
        - containerPort: 5000
```

selector: verteld welke pods moeten worden gemanaged ("flask-app")

template label: geeft een label aan de pods ("flask-app")

Toepassen met: kubectl apply -f "yaml file" (straks meer)

Services

pods zijn vluchtig (IP adres en locatie (node))

-> Mechanisme nodig om een constante blik te krijgen: **services**

Een **service** heeft een **fixed IP-adres** en routeert verkeer naar de juiste, bij de service horende, pods

Drie smaken:

- ClusterIP service: Default. Intern IP-adres: route naar pods
- NodePort service: port exposed op elke node. Access via node's IP adres
- LoadBalancer service: typisch voor cloud providers

Services - voorbeelden

```
apiVersion: v1
kind: Service
metadata:
  name: flask-app
spec:
  selector:
    app: flask-app
  ports:
  - port: 5000
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
  - port: 80
```

Kubernetes networking

Doel: bieden van een flexibele netwerk infrastructuur waarmee pods met elkaar en met externe bronnen kunnen communiceren.

Hiermee kan Kubernetes communicatie en schaalbaarheid regelen

Basis principes die hierbij moeten gelden:

- Alle pods kunnen rechtstreeks met elkaar communiceren
- Alle nodes kunnen met alle pods communiceren, en vice versa

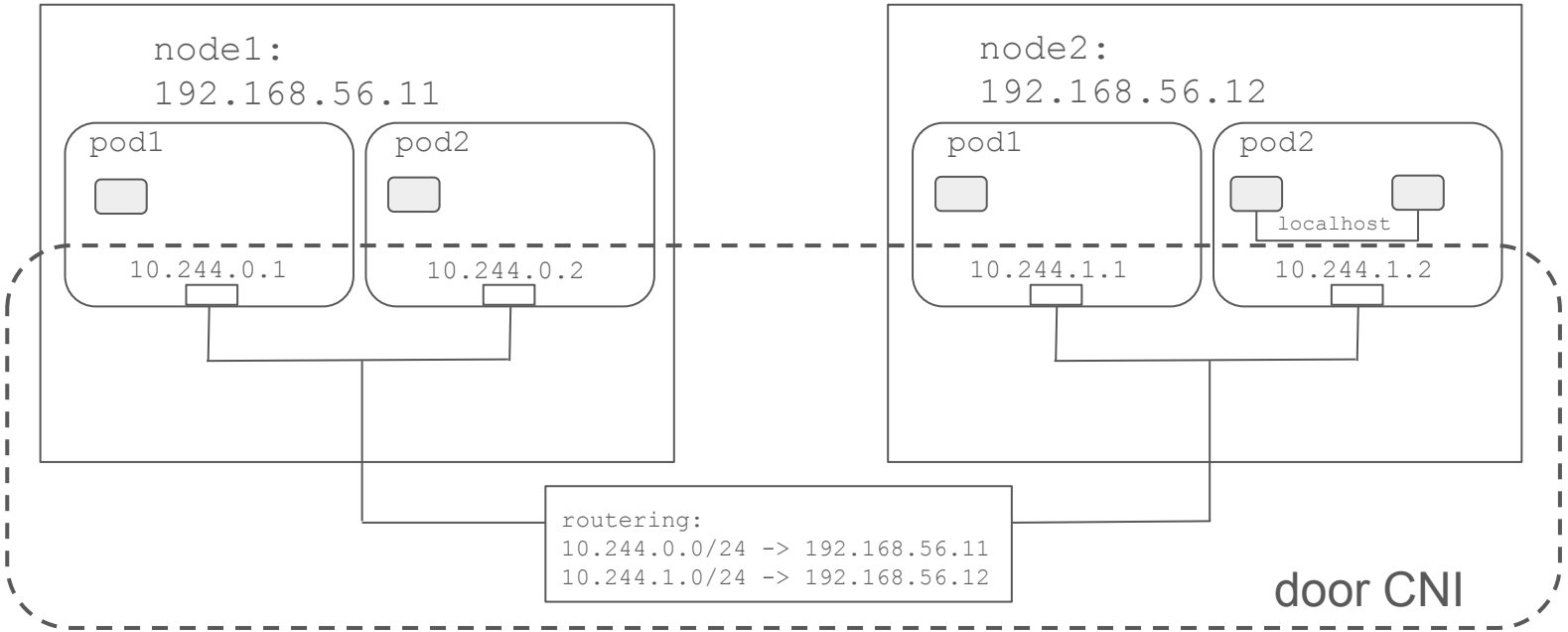
Kubernetes voorziet niet zelf in networking: gebruikt hiervoor een Container Network Interface (CNI) plugin:

- maakt netwerk interface op pods
- zorgt voor ip-adressen en routes (iptables (of eBPF) op the node)

Implementaties van CNI:

- flannel -> layer2 vlans (eenvoudige setups)
- calico -> gebruikt iptables voor routering en load balancing
- cilium -> gebruikt eBPF ipv iptables. Meer performance, beter voor grote clusters

Kubernetes networking



master (control) nodes vs worker nodes

Master (control nodes) bevinden zich in het *control plane*:

Bevat (in principe) alleen de *systeem* pods (of processen):

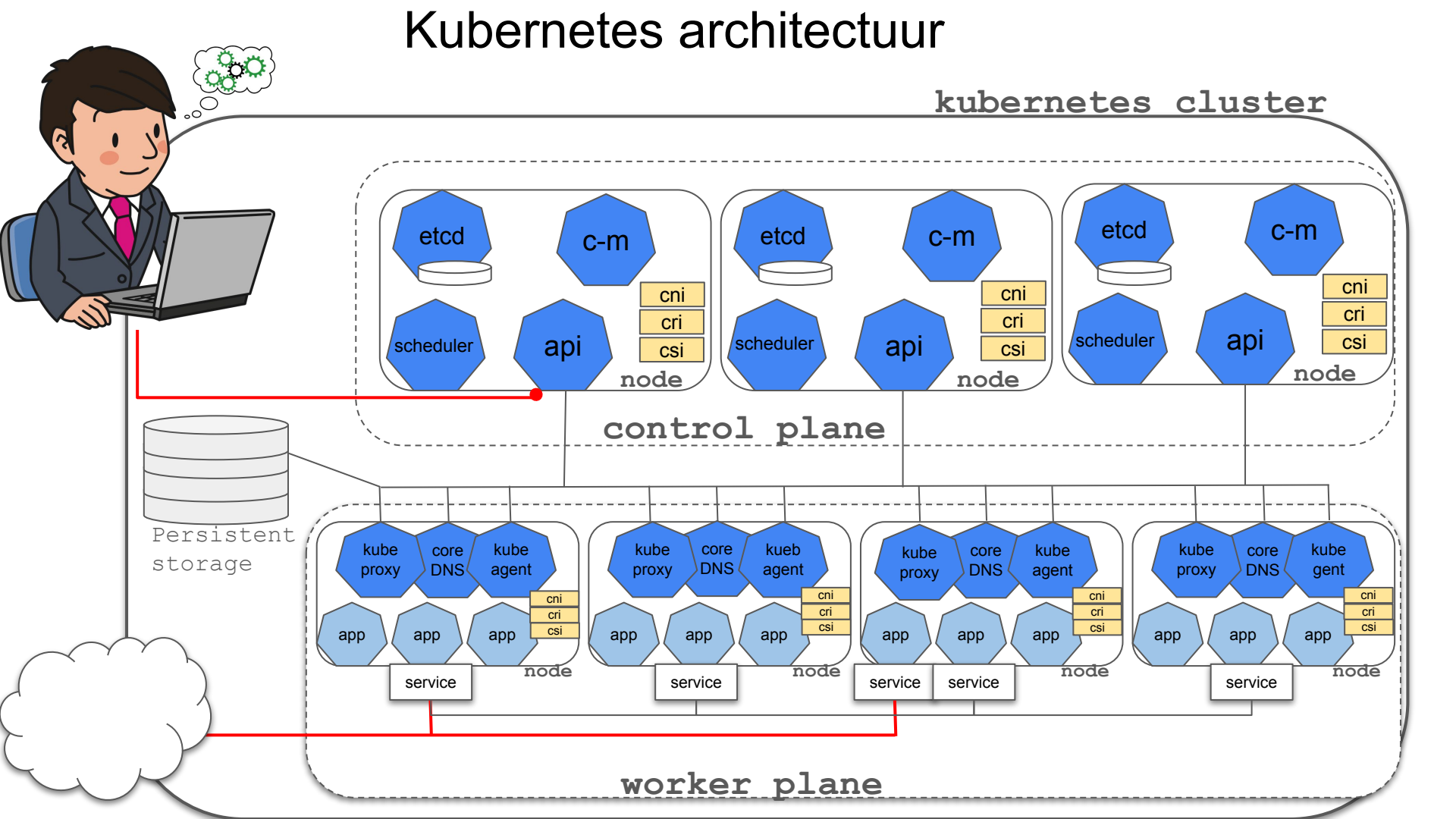
- scheduler - plaatst pods op nodes (houdt rekening met cpu/memory gebruik)
- controller (c-m) - bewaakt de staat van het cluster
- etcd: key/value store voor het cluster waarin alle resource, met values opgeslagen staan
- api - alle communicatie en commando's gaan via de api (dus met api calls)
- Minstens 3 master nodes nodig voor High Availability (HA)

Worker nodes. Bevatten de applicatie pods maar ook enkele systeem pods (of processen):

- systeem pods (of processen):
  - DNS pod - DNS-achtige service discovery (koppeling ip's <-> namen)
  - proxy pod - verzorgd de routering en load balancing rules
  - kube agent ("kubelet") - voor communicatie met de api server. Managed pods
- applicatie pods

Alle systeem pods runnen in de namespace "kube-system"

Kubernetes architectuur



Demo met microk8s

Installatie

```
root@mnode1~# snap install microk8s --classic

root@mnode1~# useradd -aG microk8s oscar

(op alle nodes)
```

Maak een cluster (nu met 3 control nodes ivm HA)

```
oscar@mnode1~$ microk8s add-node

oscar@mnode2~$ microk8s join 192.168.56.11:25000/304575..b688/b77d0edeb8eb

oscar@mnode1~$ microk8s add-node

oscar@mnode3~$ microk8s join 192.168.56.11:25000/404575..b688/v97bdfhhb6a2 [--worker]
```

microk8s - voorbereidingen

```
oscar@mnode1:~$ echo "alias kubectl='microk8s kubectl'" >> ~/.bash_aliases
oscar@mnode1:~$ exec bash
oscar@mnode:~$ kubectl get no[des]
    NAME      STATUS   ROLES    AGE     VERSION
    mnode1    Ready    <none>   152m    v1.32.3
    mnode2    Ready    <none>   149m    v1.32.3
    mnode3    Ready    <none>   8m21s   v1.32.3

oscar@mnode1:~$ microk8s config > /tmp/config
oscar@asterix:~$ scp mnode1:/tmp/config ~/.kube/config
oscar@asterix:~$ kubectl get no[des]
```

Eenvoudige applicatie met drie onderdelen

- 1) Webserver nginx, service met een NodePort
  - Stuurt al het verkeer naar een flask applicatie op port 5000

```
listen 80;

location / {

    proxy_pass http://flask-app:5000;

}
```

- 2) De flask app. Kan connecten met de mysql app
- 3) Een mysql app (met data op peristent storage (nfs))

We gaan kijken naar:

- maken van de onderdelen (deployments, services) met yaml manifests
- uitvoeren van een rolling update

Troubleshoot

- node niet "ready"? (`kubectl get nodes`)
- Is de api server bereikbaar? (pod: curl -k https://cluster-IP:16443)
- journalctl -u snap.microk8s.daemon-kubelite
- journalctl -u snap.microk8s.daemon-.....

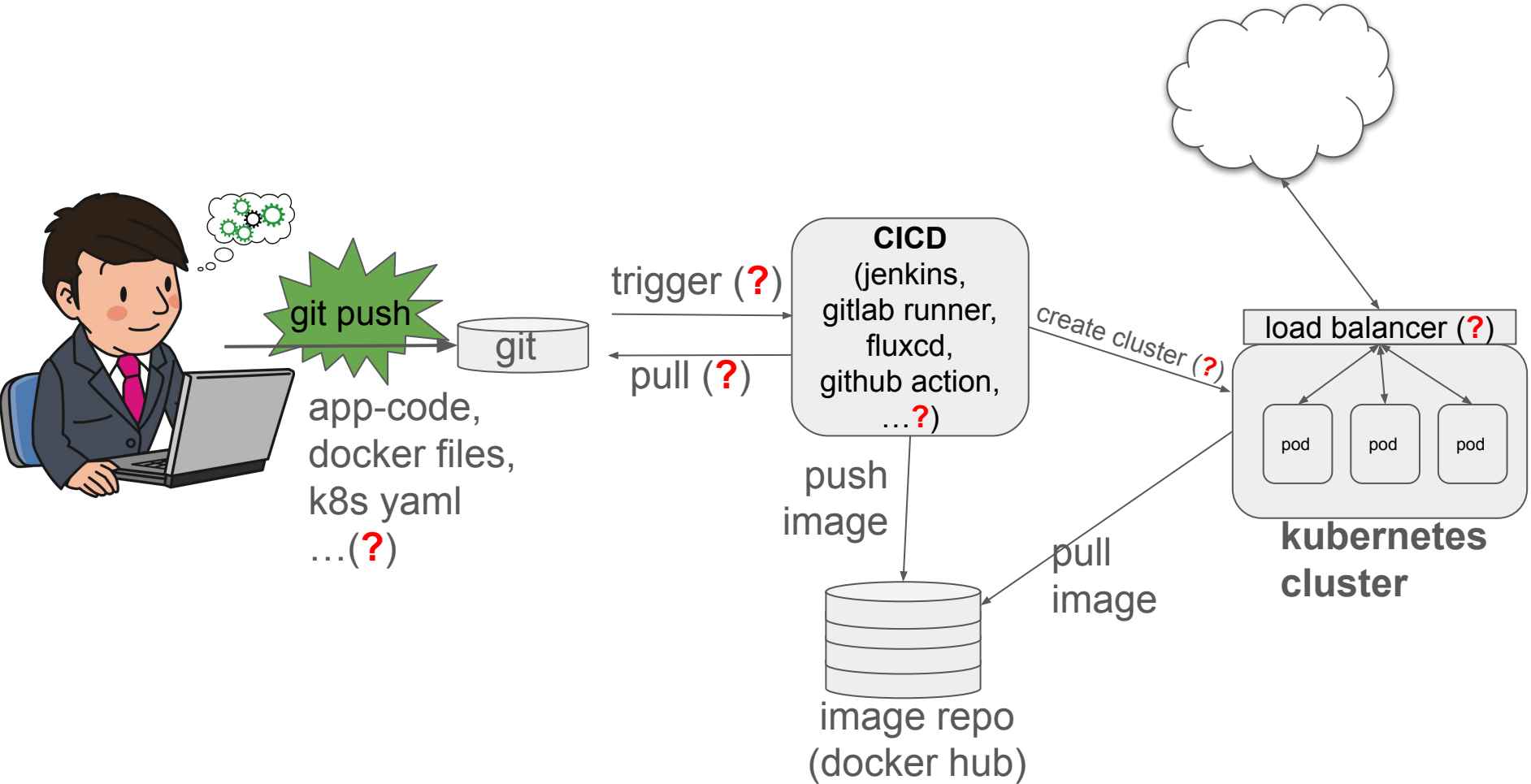
```
for service in $(systemctl list-units --type=service -all | grep microk8s |\
grep running | awk '{print $1}'); do echo "Service: ${service}"; echo;\
journalctl -u ${service} | tail; echo "hit enter"; echo; read dummy; done
```

- kubectl logs "podname"
- microk8s status
- microk8s inspect - maakt tarball met veel info over je cluster

Referenties

- https://kubernetes.io
- https://atcomputing.nl/blog/open-source-summit-2022.html

Andere keer?


git push
git
app-code,
docker files,
k8s yaml
…(?)
trigger (?)
pull (?)
CICD
(jenkins,
gitlab runner,
fluxcd,
github action,
…?)
push
image
image repo
(docker hub)
pull
image
create cluster (?)
load balancer (?)
pod
pod
pod
kubernetes
cluster

Bier?