Modul CNT
Kubernetes Ressourcen (Objects)

September 2021

Marcel Bernet

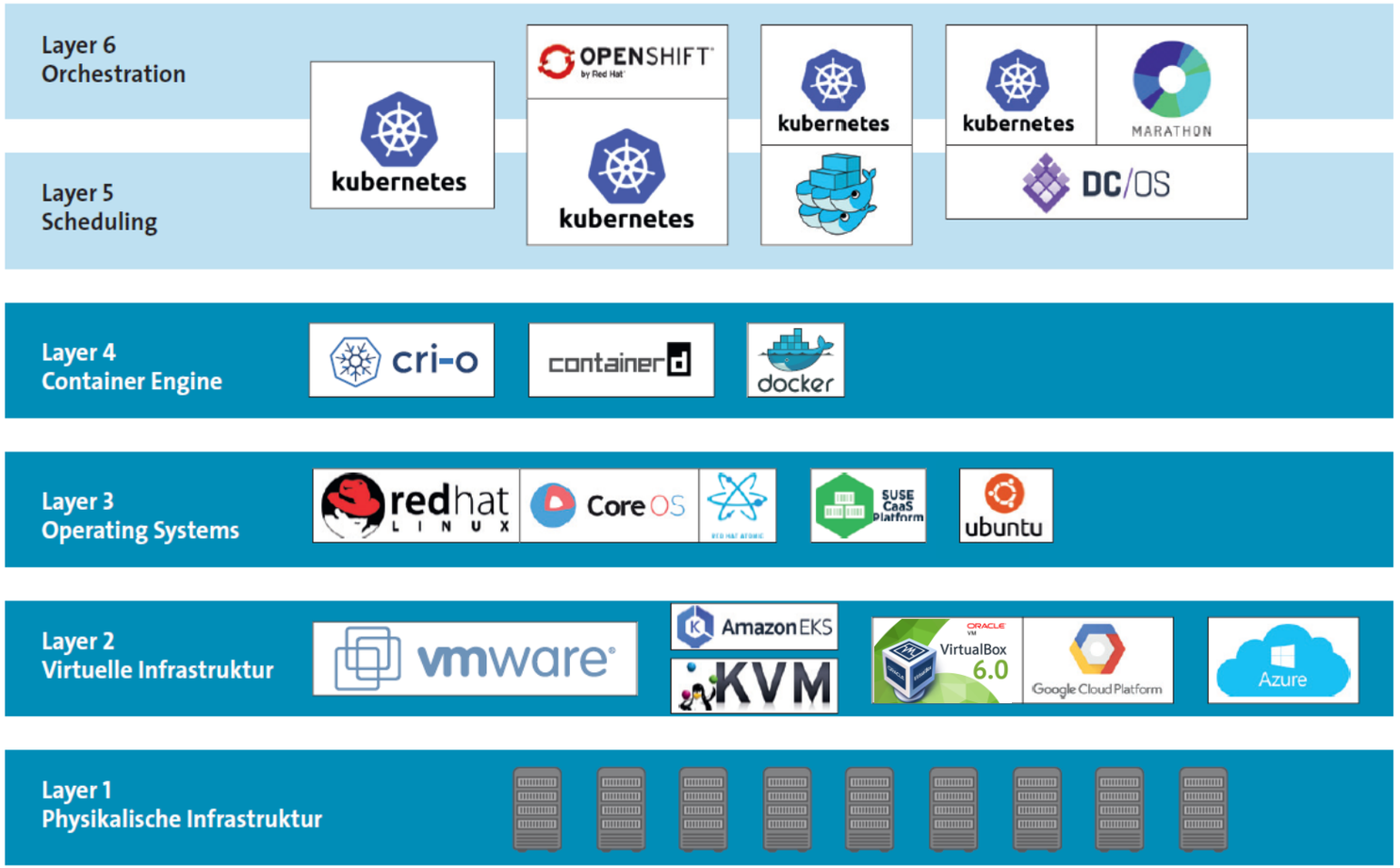
Lernziele

★ Sie haben einen Überblick über die Ressourcen im K8s Cluster und können diese Einordnen.

Zeitlicher Ablauf

- ★ Kubectl-CLI
- ★ YAML (Dateien)
- ★ K8s Ressourcen (Objects) jeweils mit 1 – 2 Übungen
  - Basis Ressourcen (Pod, Labels, Selector, Service, Namespaces)
  - Netzwerk Ressource (Ingress)
  - Ressourcen für Verteilung und Updates (ReplicaSet, Deployment)
  - Ressourcen für Persistenz (Volume, PersistenVolumeManager)
  - Ressourcen für Konfiguration (Secrets, ConfigMap)
  - Spezial Ressourcen (Jobs, DaemonSet, StatefullSet)
- ★ Reflexion
- ★ Lernzielkontrolle
- ★ Quelle K8s Folien Core primitives und Running Microservices: (https://www.youtube.com/watch?v=A4A7ybtQujA)

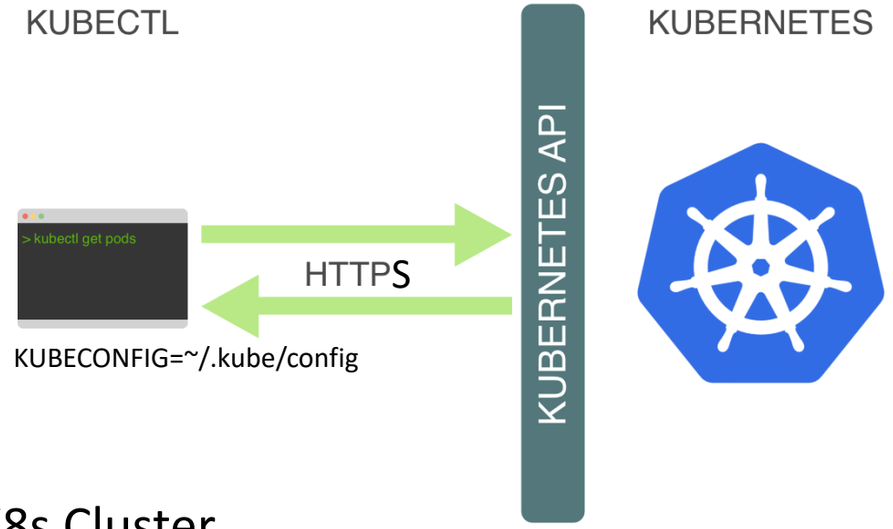
Die (vereinfachten) »Layer« der Container-Welt



Nun geht es um die Layer 5 und 6

Quelle: Buch Skalierbare Container-Infrastrukturen, Kapitel 13

Kubectl-CLI (1)



- ★ Das kubectl-Kommando stellt, eine der Schaltzentralen des K8s Clusters zur Administration der Ressourcen dar.
- ★ Die wichtigsten kubectl-Subkommandos in der Übersicht
  - `kubectl cluster-info` – Liefert Informationen zum K8s Cluster
  - `(kubectl run` *`name`* `--image=`*`image`* – startet ein Container (ähnlich docker run))
  - `(kubectl expose` – öffnet Port gegen aussen)
  - `kubectl get [all] [-o yaml]` – zeigt Ressourcen, in unterschiedlichen Formaten, an
  - `(kubectl create -f` *`YAML`* – Erstellt eine Ressource laut YAML Datei/Verzeichnis)
  - `kubectl apply -f` *`YAML`* – führt die Änderungen in der YAML im Cluster nach
  - `kubectl delete -f` *`YAML`* – Löscht eine Ressource laut YAML Datei/Verzeichnis

Quellen: Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://itnext.io/boosting-your-kubectl-productivity-b348f7c25712

Kubectl-CLI (2)

- `kubectl describe` *`type/resource`* – zeigt Details einer Ressource an
- `kubectl explain` *`pods`* – zeigt die Dokumentation zu einer Ressource an
- `kubectl logs` *`pod`* – zeige das Log eines Containers
- `(kubectl exec` – startet einen Befehl im Container oder wechselt in Container)
- `kubectl port-forward` - Port eines Service innerhalb des Clusters an lokales System weiterleiten (siehe weave.bat)
- `kubectl proxy` - Aufrufe an den API-Service ab lokalem System ermöglichen (siehe dashboard.bat)

➢ kubectl Cheatsheet und 9 kubectl commands sysadmins need to know

➢ Kubernetes 1.18 broke `kubectl run`

Quellen: Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://itnext.io/boosting-your-kubectl-productivity-b348f7c25712

Welche Ressourcen/Aktionen können im K8s Cluster via kubectl-CLI verwaltet werden?

Ressourcen-/Aktionstyp	Shortname/Alias
clusters	
componentstatuses	cs
configmaps	cm
daemonsets	ds
deployments	deploy
endpoints	ep
event	ev
horizontalpodautoscalers	hpa
ingresses	ing
jobs	
limitranges	limits
namespaces	ns
networkpolicies	
nodes	no
petset	
persistentvolumeclaims	pvc
persistentvolumes	pv
pods	po
podsecuritypolicies	psp
podtemplates	
replicasets	rs
replicationcontrollers	rc
resourcequotas	quota
cronjob	
secrets	
serviceaccount	sa
services	svc
storageclasses	
thirdpartyresources	

z.B. `kubectl get persistentvolumes`
`kubectl get pv`

Ausgabe Liste aller Ressourcen
`kubectl api-resources`

Quelle: Buch Skalierbare Container-Infrastrukturen, Kapitel 13

YAML (Dateien)

```yaml
# Order
apiVersion: v1
kind: Service
metadata:
  name: order-standalone
  labels:
    app: scsesi-order-standalone
    group: ewolff-scsesi
    tier: frontend
spec:
  type: NodePort
  ports:
  - port: 8080
    nodePort: 32090
  selector:
    app: scsesi-order-standalone
```

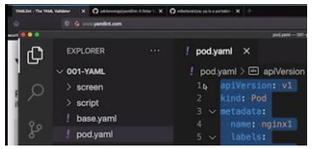
```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: scsesi-order-standalone
spec:
  replicas: 1
  selector:
    matchLabels:
      app: scsesi-order-standalone
  template:
    metadata:
      labels:
        app: scsesi-order-standalone
        group: ewolff-scsesi
        tier: frontend
    spec:
      containers:
      - name: scsesi-order-standalone
        image: scsesi_order
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8080
```

Achtung

Wichtig bei allen folgenden Yaml/JSON-Files sind natürlich insbesondere die korrekten Einrückungen, da über sie die Hierarchien der Direktiven definiert werden.

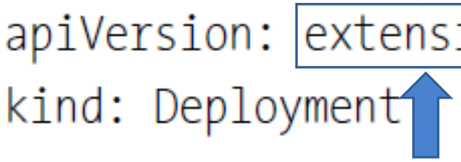
Zudem ist darauf zu achten, dass die Verwendung von Tabs bzw. die Vermischung von Tabs und Spaces problematisch sein kann. Insofern sollten für alle Einrückungen am besten durchgängig Spaces verwendet werden.

- YAML ist eine vereinfachte Auszeichnungs-sprache (englisch markup language) zur Datenserialisierung, angelehnt an XML (ursprünglich) und an die Datenstrukturen in den Sprachen Perl, Python und C sowie dem in RFC2822 vorgestellten E-Mail-Format.
- Die grundsätzliche Annahme von YAML ist, dass sich jede beliebige Datenstruktur nur mit assoziativen Listen, Listen (Arrays) und Einzelwerten (Skalaren) darstellen lässt. Durch dieses einfache Konzept ist YAML wesentlich leichter von Menschen zu lesen und zu schreiben als beispielsweise XML, ausserdem vereinfacht es die Weiterverarbeitung der Daten, da die meisten Sprachen solche Konstrukte bereits integriert haben.
- YAML tips for Kubernetes



API-Gruppen

Inhalt YAML Datei (API + Ressource)



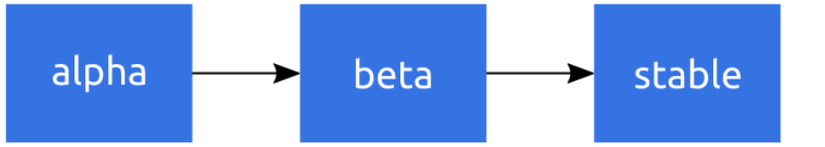
- ★ Um die Erweiterung der Kubernetes-API zu vereinfachen, wurden API-Gruppen implementiert.
- ★ Die API-Gruppe steht im REST-Pfad und im apiVersion: Eintrag.
- ★ Derzeit werden mehrere API-Gruppen verwendet:
  - Die Kerngruppe: /api/v1 und apiVersion: v1.
  - Alle anderen Gruppen: /apis/$GROUP_NAME/$VERSION und apiVersion: $GROUP_NAME/$VERSION
  - Die vollständige Liste der unterstützten API-Gruppen finden Sie in der Kubernetes API-Referenz
- ★ Es gibt zwei Möglichkeiten zum Erweitern des APIs mit benutzerdefinierten Ressourcen :
  - CustomResourceDefinition für Ressourcen mit CRUD-Anforderungen
  - Implementieren eines eigenen Apiserver mittels des Aggregation Layer.

API-Versionierungsdickicht

Inhalt YAML Datei (API + Ressource)

```
apiVersion: extensions/v1beta1
kind: Deployment
```

- Nicht jedes der vorgenannten Objekte bzw. jede Ressource kann im K8s Cluster »einfach mal so« ausgerollt werden.
- Entscheidend darüber, ob unser Kubernetes Cluster unsere Eingabe akzeptiert, ist das Attribut apiVersion in der YAML Datei, welches im Default auf V1 (Version 1) eingestellt ist.



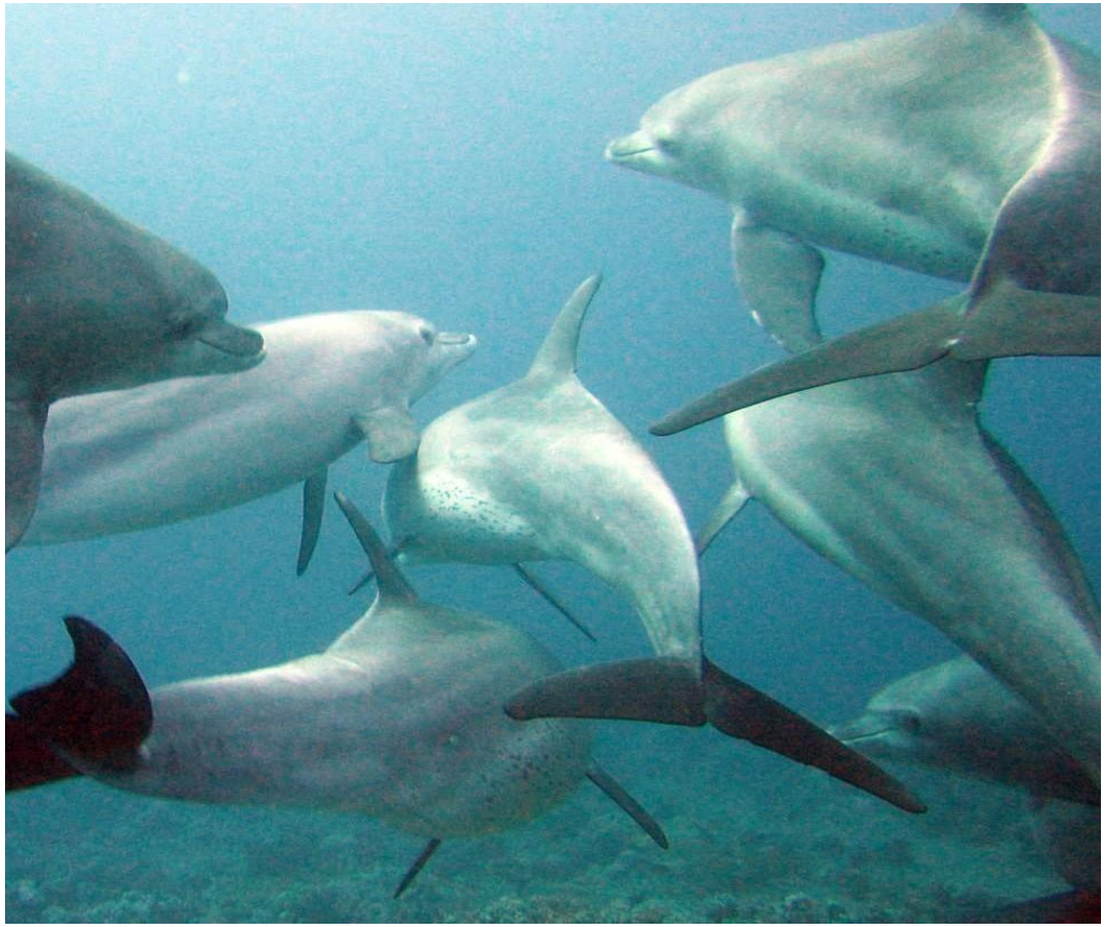
★ **Alpha**:
- eventuell buggy
- standardmäßig deaktiviert
- Support kann in zukünftigen Versionen wieder gestrichen werden
- API kann sich ändern
- nur für Testumgebungen

★ **Beta**:
- gut getesteter und (**nach den Massstäben der Container-Welt**) als sicher zu betrachtender Code
- Support für Features bleibt bestehen
- Kleine Details können sich ändern
- Schemadefinitionen können sich noch ändern
- nur für nicht »Business Critical«-Anwendungen gedacht

★ **Stable**: wie üblich

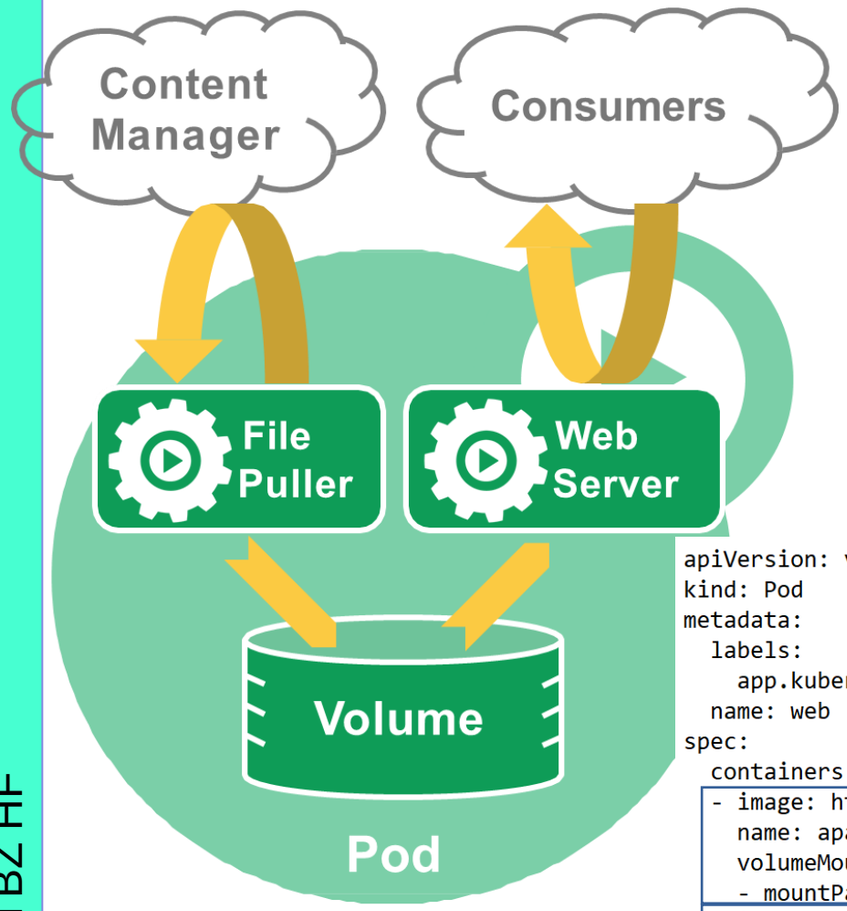
Quelle: Buch Skalierbare Container-Infrastrukturen, Kapitel 13

Pods (1)



Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/workloads/pods/pod/

Pods (2)

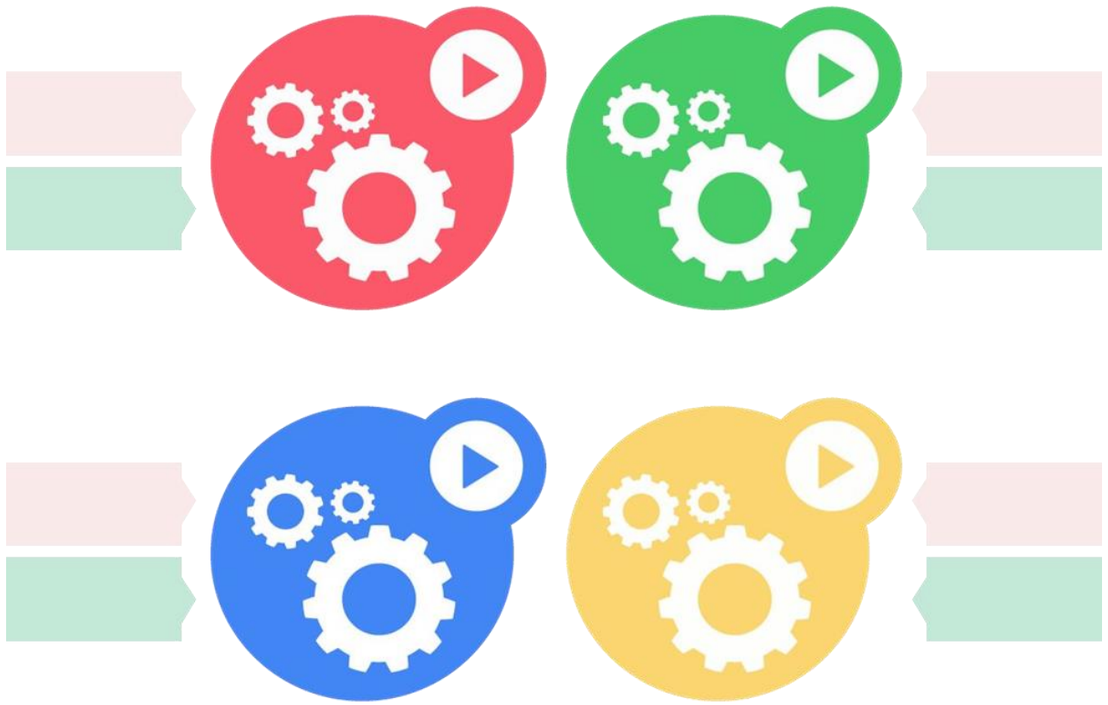



```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app.kubernetes.io/name: web
  name: web
spec:
  containers:
  - image: httpd
    name: apache
    volumeMounts:
    - mountPath: "/usr/local/apache2/htdocs"
  - image: debian:jessie
    name: file-puller
    volumeMounts:
    - mountPath: "/usr/local/apache2/htdocs"
      name: "web-storage"
```

- ★ Kleine Gruppe von Containern welche eng verbunden sind.
  - Kleinste Einheit für Replikation und Platzierung (auf Node).
  - “Logischer” Host für Container
- ★ Jeder Pods erhält genau eine IP-Adresse
  - share data: localhost, volumes, IPC, etc.
- ★ Erleichtert zusammengesetzte Applikationen
- ★ Beispiele: file puller (holt Daten, schreibt index.html), web server (zeigt Daten an).

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/workloads/pods/pod/

Labels



- ★ Vom User bereitgestellte **Schlüssel** und **Werte** (key=value)
- ★ Kennzeichnung eines API-Objekt

```
labels:
        name: web
        tier: frontend
```

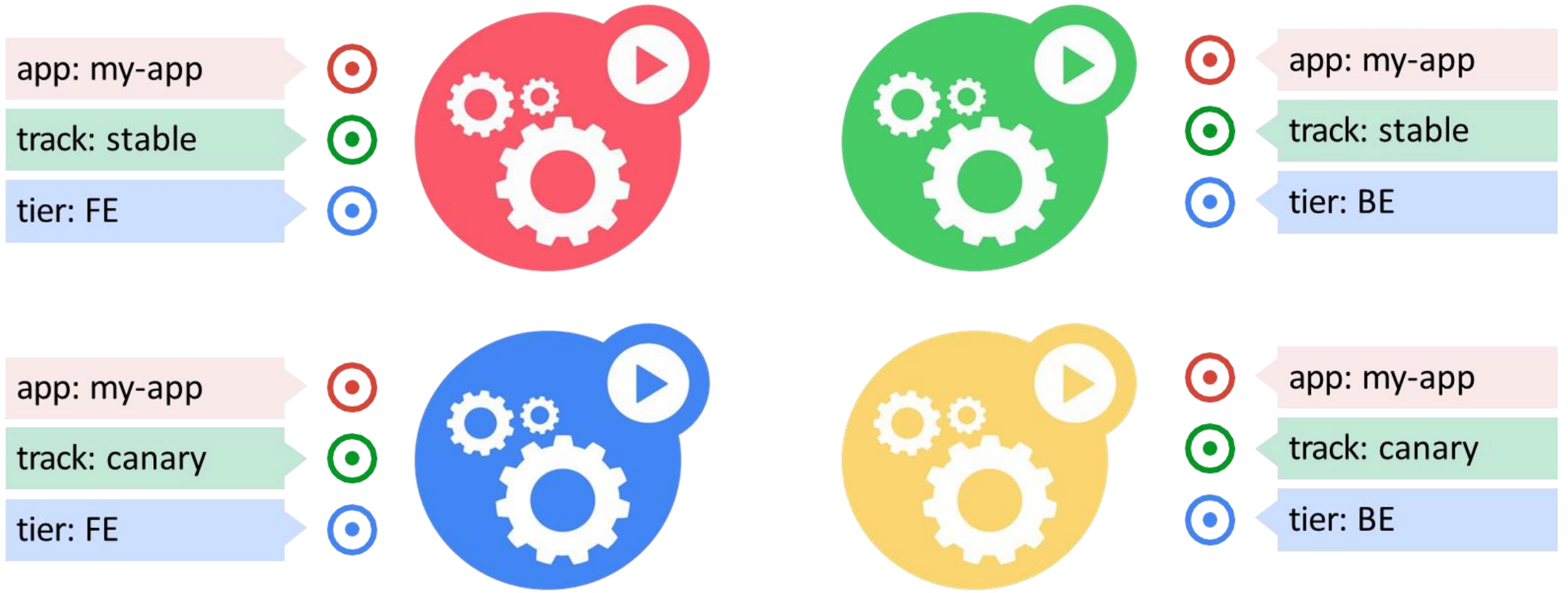
- ★ Abfragbar mittels Selektoren (selectors), ähnlich SQL

```
kubectl get pods -l tier=frontend
```

- ★ Der **einzige** Gruppierungsmechanismus

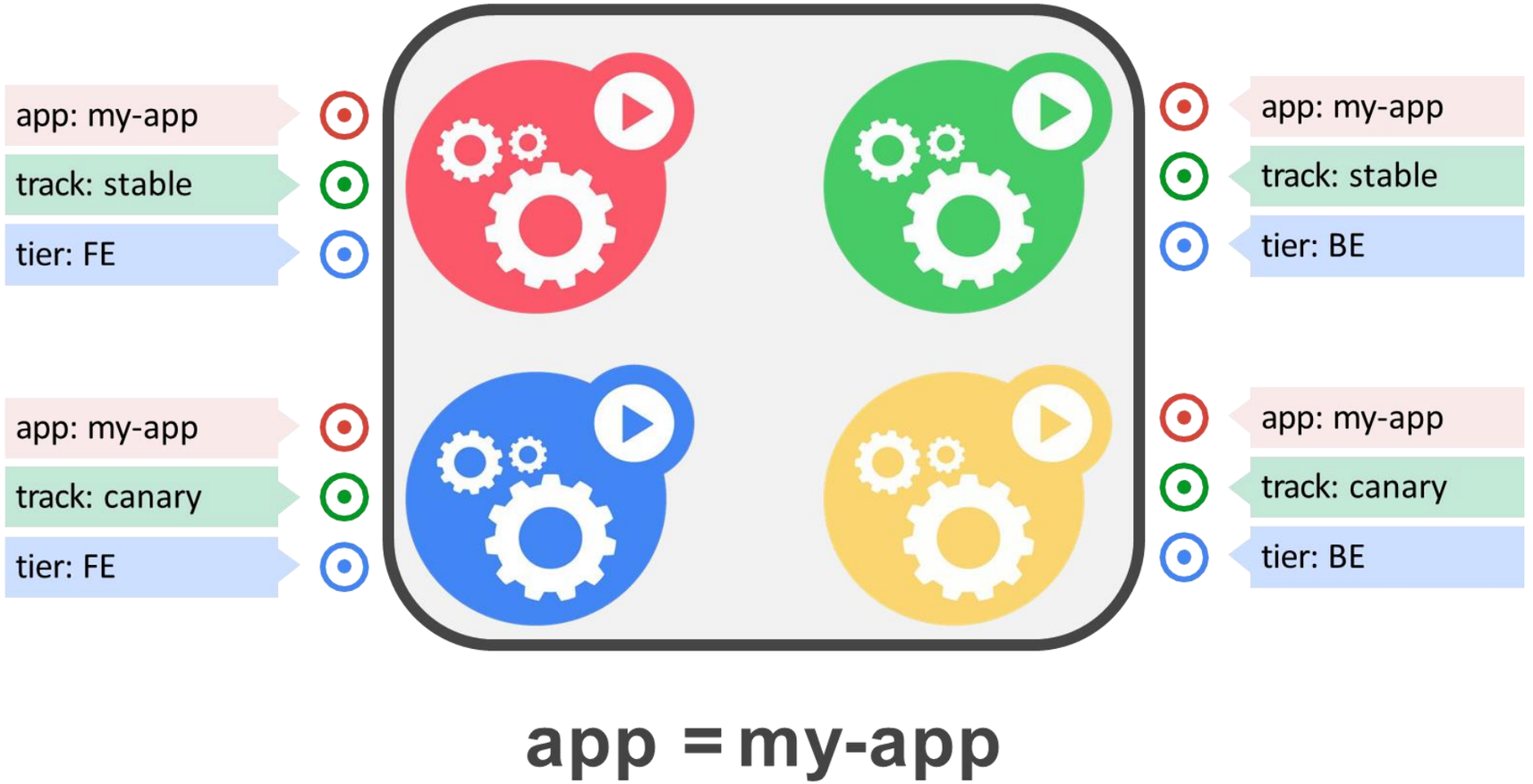
Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Selectors (1)



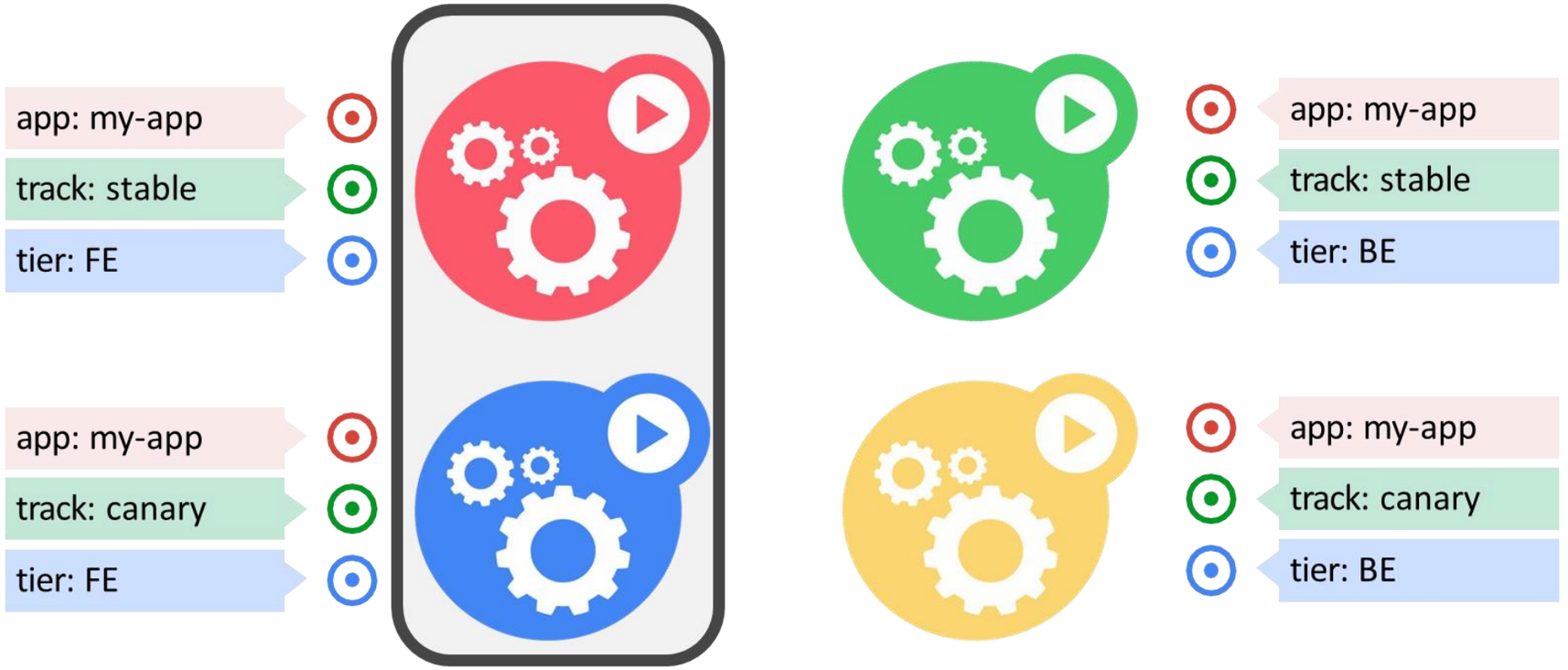
Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Selectors (2)



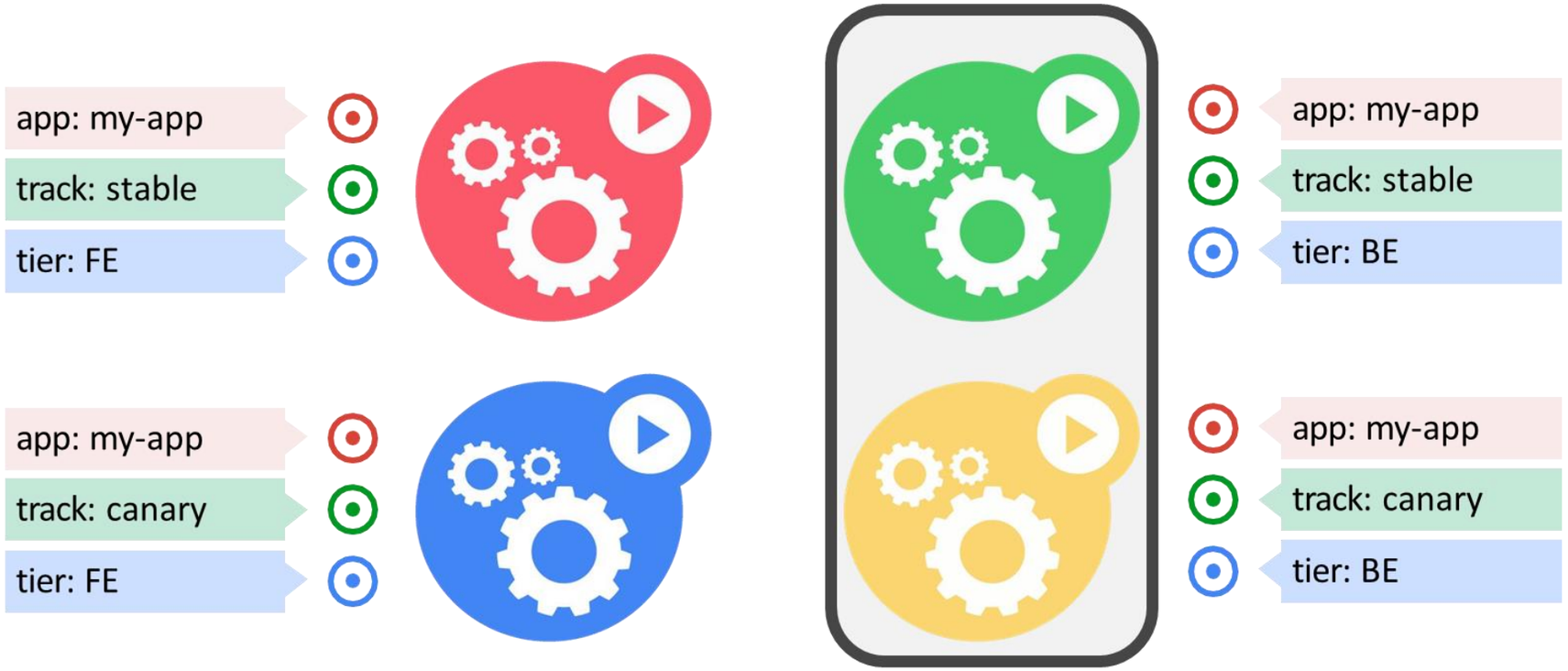
Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Selectors (3)



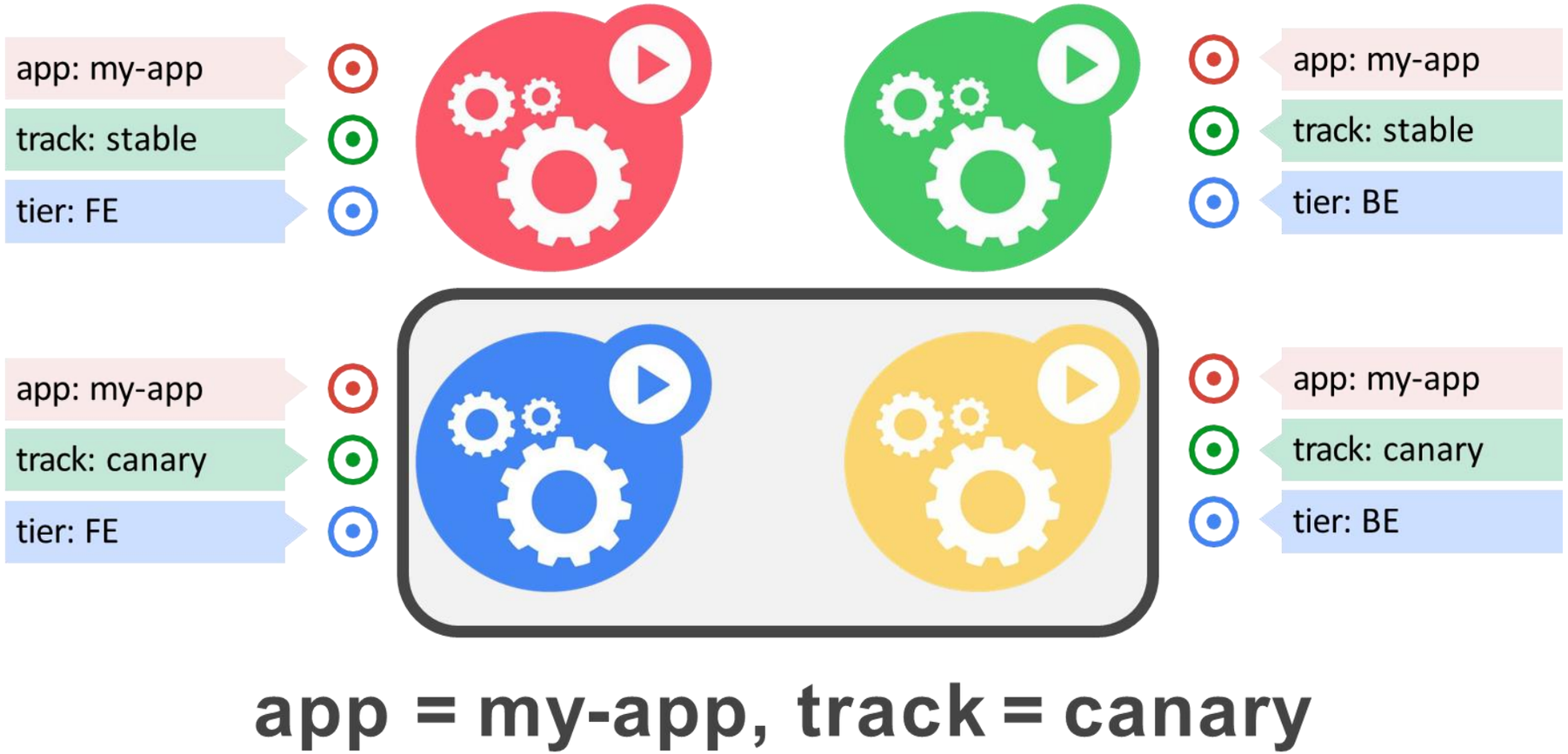
Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Selectors (4)



Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Selectors (5)



Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Recommended Labels (Empfohlene Labels)

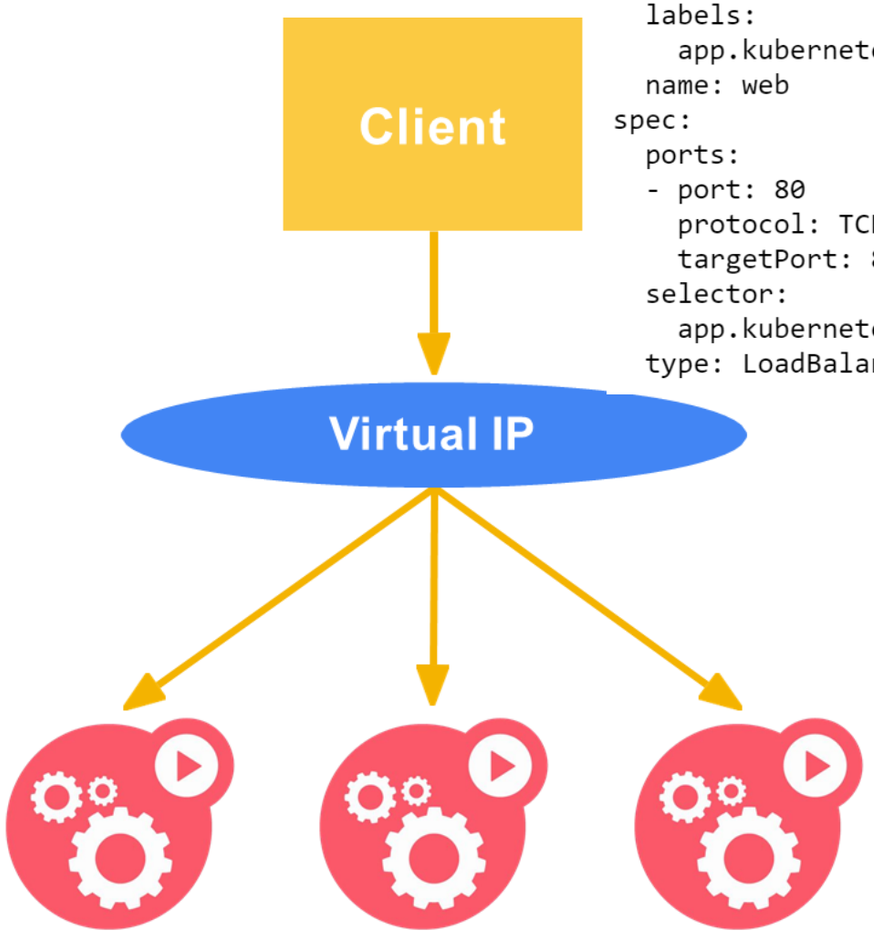
Key	Description	Example	Type
`app.kubernetes.io/name`	The name of the application	`mysql`	string
`app.kubernetes.io/instance`	A unique name identifying the instance of an application	`wordpress-abcxzy`	string
`app.kubernetes.io/version`	The current version of the application (e.g., a semantic version, revision hash, etc.)	`5.7.21`	string
`app.kubernetes.io/component`	The component within the architecture	`database`	string
`app.kubernetes.io/part-of`	The name of a higher level application this one is part of	`wordpress`	string
`app.kubernetes.io/managed-by`	The tool being used to manage the operation of an application	`helm`	string

To illustrate these labels in action, consider the following StatefulSet object:

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app.kubernetes.io/name: mysql
    app.kubernetes.io/instance: wordpress-abcxzy
    app.kubernetes.io/version: "5.7.21"
    app.kubernetes.io/component: database
    app.kubernetes.io/part-of: wordpress
    app.kubernetes.io/managed-by: helm
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/common-labels/
https://github.com/kyverno/kyverno/blob/main/samples/RequireLabels.md

Services (port-based routing)




```
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/name: web
  name: web
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app.kubernetes.io/name: web
  type: LoadBalancer
```

- ★ Eine Gruppe von Pods die zusammenarbeiten, Gruppiert mittels Label Selector (i.d.R. mehrere Instanzen eines Microservices)
- ★ Erlaubt mittels unterschiedlichen Methoden auf den Service zuzugreifen:
  - DNS Name und DNS SRV (Service Resource Record)
  - Kubernetes Endpoints API
- ★ Definiert die Sicherbarkeit des Services (ServiceTypes):
  - ClusterIP: nur innerhalb des Clusters erreichbar (default).
  - NodePort: via Port und IP Worker Node erreichbar.
  - LoadBalancer: mittels eigener IP Adresse erreichbar (dazu muss ein LoadBalancer vorhanden sein). Ansonsten verhalten wie NodePort.
- ★ Versteckt Komplexität.
- ★ **Beispiel** (Web Server)
  - ClusterIP: http://web:80
  - NodePort: http://<Worker-Node-IP>:3xxxx
  - LoadBalancer: z.B. http://192.168.137.11:80

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/services-networking/service/

Services (K8s Dienst Kube-Proxy)

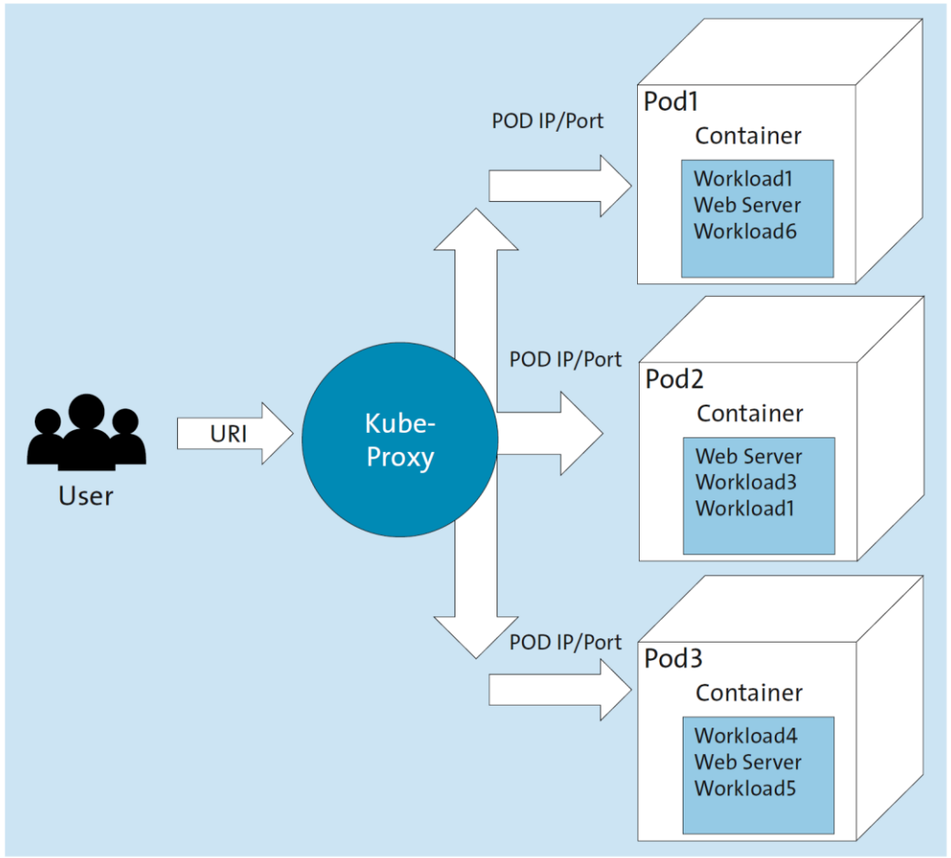


Abbildung 13.2 Arbeitsweise des Kube-Proxy exemplarisch auf einem K8s Worker Node

- ★ Der K8s-Netzwerk-Proxy – er arbeitet auf jedem Worker Node und kümmert sich um die eingehenden Requests und ihr Routing.
- ★ Er kann als primitiver Loadbalancer fungieren

```
r-01:~$ kubectl get pods --all-namespaces
NAME                                    READY
coredns-5644d7b6d9-dsxh7                1/1
coredns-5644d7b6d9-ndwjq                1/1
etcd-master-01                          1/1
kube-apiserver-master-01                1/1
kube-controller-manager-master-01       1/1
kube-flannel-ds-amd64-pn94s             1/1
kube-proxy-j7fb2                        1/1
kube-scheduler-master-01                1/1
```

- ★ Weitere Informationen:
  - How To Inspect Kubernetes Networking
  - Kubernetes NodePort and iptables rule
  - Install a Kubernetes load balancer

Übung: Basis Ressourcen (09-1-kubectl)

Übung: kubectl-CLI und Basis Ressourcen

Das `kubectl`-Kommando stellt, eine der Schaltzentralen des K8s Clusters zur Administration der Ressourcen dar.

In dieser Übung verwenden wir das `kubectl`-Kommando zur Erstellen eines Pod's und Services nutzen.

Das passiert in einer eigenen Namespace um die Resultate gezielt Darstellen zu können:

```
! kubectl create namespace test
```

```
  namespace/test created
```

Erzeugen eines Pod's, hier der Apache Web Server.

Die Option `--restart=Never` erzeugt nur einen Pod. Ansonsten wird ein Deployment erzeugt.

```
! kubectl run apache --image=httpd --restart=Never --namespace test
```

```
  pod/apache created
```

Ausgabe der Erzeugten Ergebnisse und die YAML Datei welche den Pod beschreibt:

Übung: Basis Ressourcen (09-2-YAML)

Übung: kubectl-CLI und Basis Ressourcen (YAML Variante)

Das `kubectl` -Kommando stellt, eine der Schaltzentralen des K8s Clusters zur Administration der Ressourcen dar.

Die `YAML` Beschreiben die Ressourcen und Vereinfachen so die Verwendung des `kubectl` Kommandos.

In dieser Übung verwenden wir das `kubectl` -Kommando mit `YAML` Dateien zur Erstellen eines Pods und Services.

Das passiert in einer eigenen Namespace um die Resultate gezielt Darstellen zu können:

```
! kubectl create namespace yaml
```

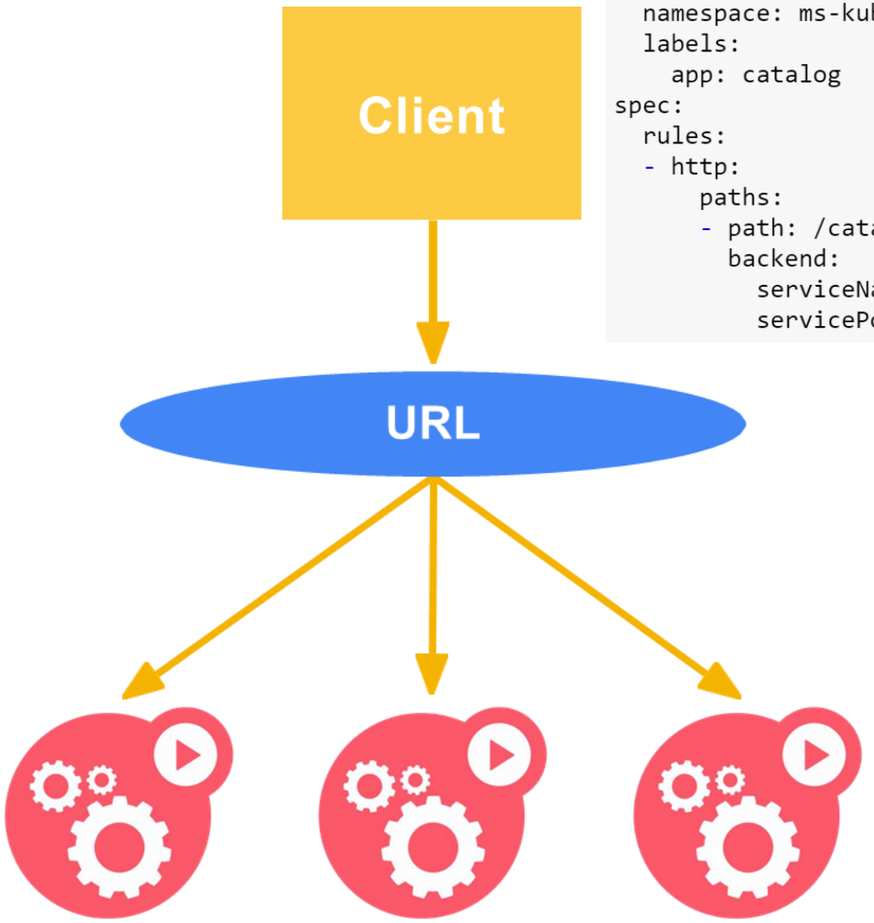
```
namespace/yaml created
```

Für den Pod haben wir, die Ausgabe von `kubectl get pod apache -o yaml` genommen und eine abgespeckte Variante als YAML Datei erstellt:

```
! cat 09-2-YAML/apache-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app.kubernetes.io/name: apache
  name: apache
  namespace: yaml
spec:
  containers:
  - image: httpd
    name: apache
```

Ingress (Reverse Proxy, URL-based routing)




```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: catalog
  namespace: ms-kubernetes
  labels:
    app: catalog
spec:
  rules:
  - http:
      paths:
      - path: /catalog/
        backend:
          serviceName: catalog
          servicePort: 8080
```

- ★ Ein API-Objekt, das den externen Zugriff auf die Dienste in einem Cluster verwaltet, in der Regel mittels HTTP.
- ★ Ingress bietet Lastenausgleich, SSL termination und namenbasiertes virtuelles Hosting.
- ★ Grob entspricht der Ingress Dienst dem Reverse Proxy Muster.

- ★ **Beispiele (nginx):**
  - https://<Cluster>:30443/order/list.html
  - https://<Cluster>:30443/customer
  - https://<Cluster>:30443/catalog

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/services-networking/ingress/

Ingress (K8s Dienste Ingress)

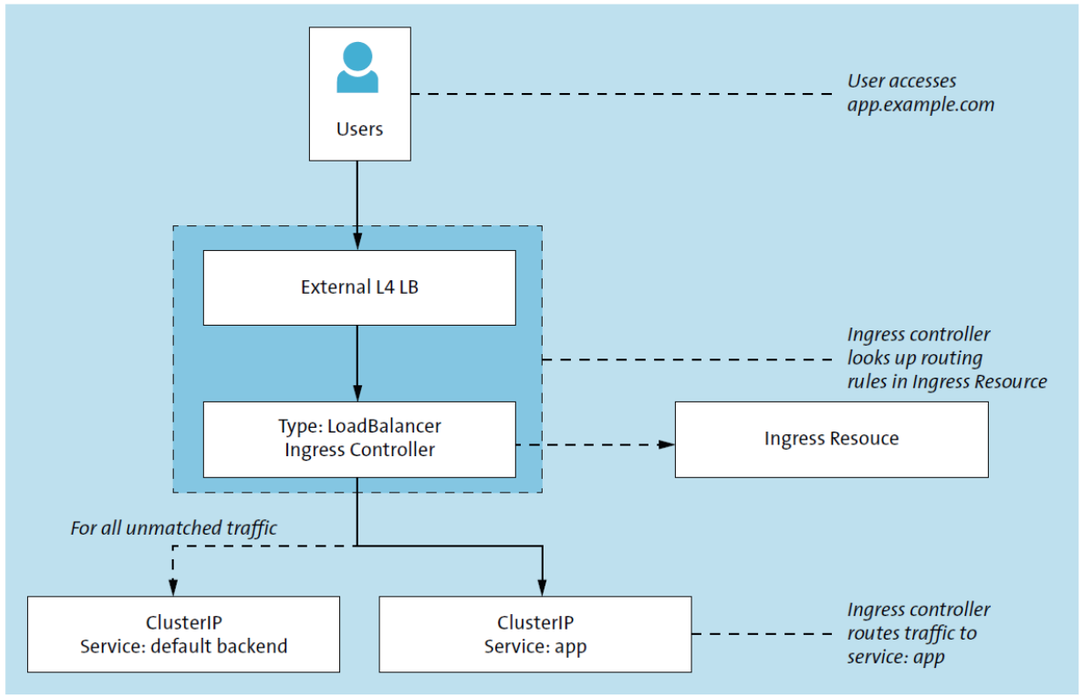


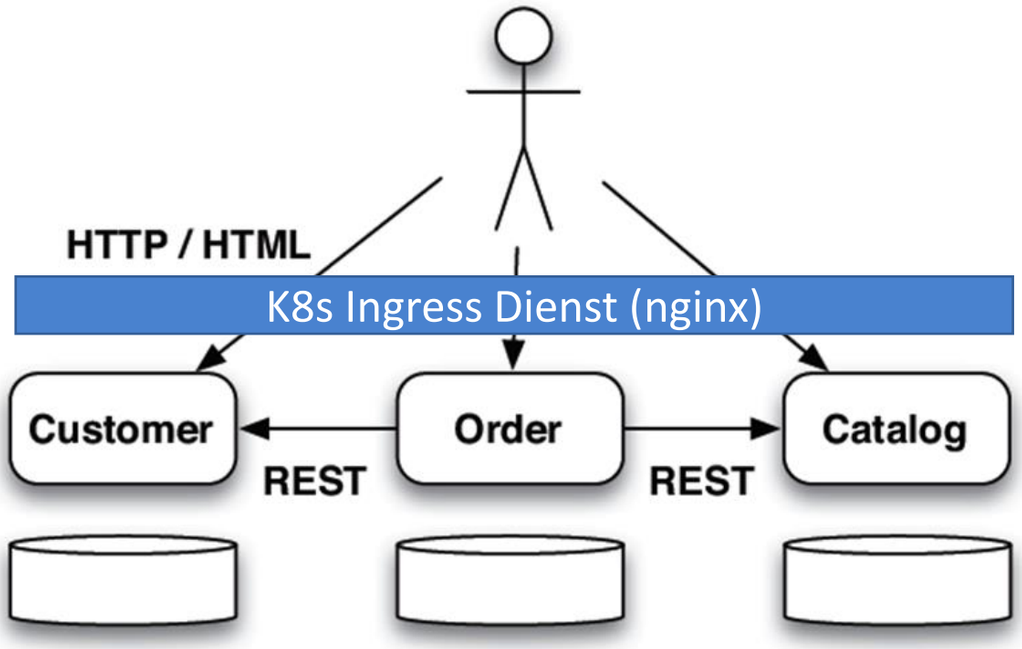
Abbildung 14.22 Ingress-Traffic-Routing

- ★ Es sind mehrere Ingress Dienste gleichzeitig möglich.
- ★ Beispiel:
  - nginx, Port **30443**
  - istio, Port **31380**

```
ingress-nginx          nginx-ingress-controller-
istio-system           grafana-d8465c484-8wxtw
istio-system           istio-citadel-67f6594c46-
istio-system           istio-egressgateway-6ff96
```

- ★ Übersicht von Ingress Controllern
- ★ Vergleich von Ingress Controllern

Übung: Ingress Ressource (09-04-Ingress)



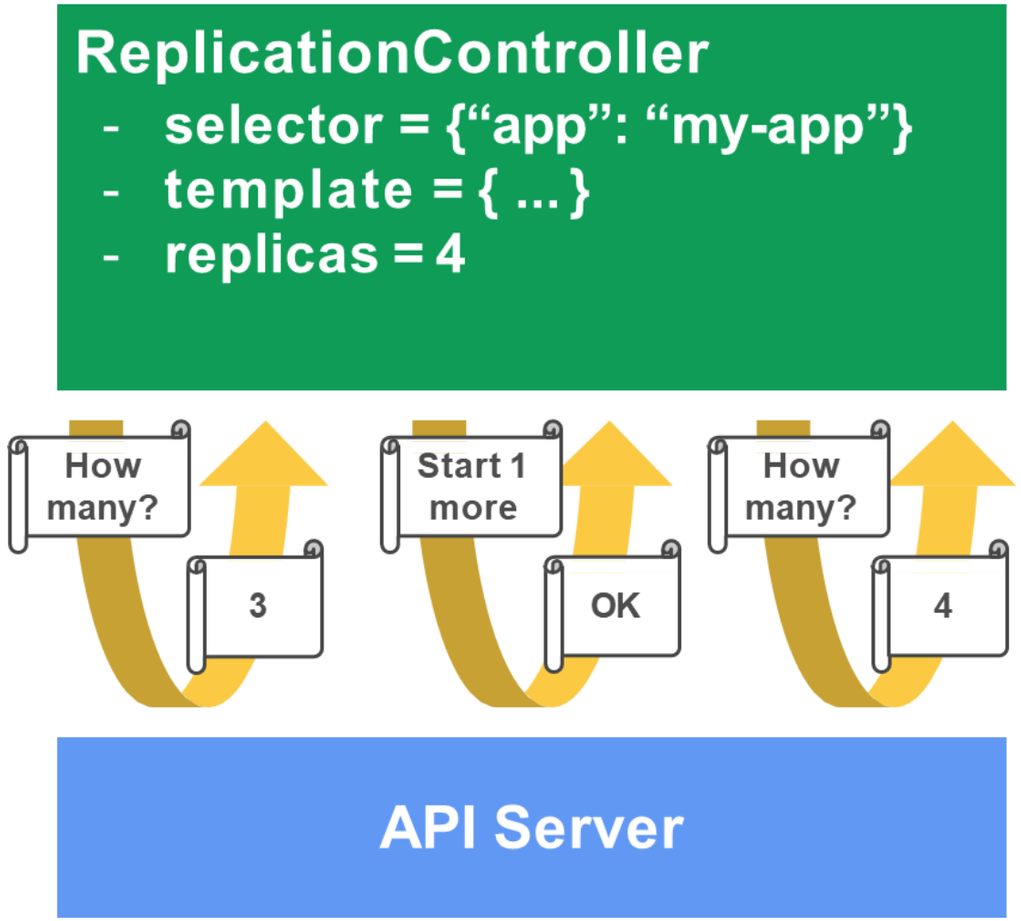
★ Für ein an Kubernetes und nginx angepasstes Beispiel siehe Microservices-BPMN.ipynb.

HTML Eingabemaske
Rechnung freigeben

BPMN Engine Orchestriert
die Microservices

Verbucht Rechnung

Replication Controller (Neu ReplicaSets)

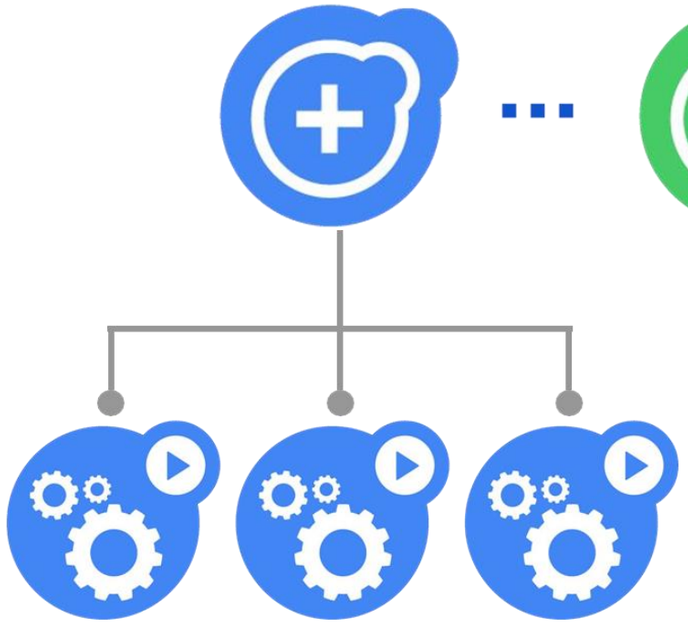


- ★ Stellt sicher, dass N Pods (Instanzen) laufen
  - sind es zu wenig, werden neue gestartet
  - sind es zu viele werden Pods beendet
  - gruppiert durch den Label Selector
- ★ Explizite Deklaration der gewünschten Anzahl Pods
  - ermöglicht Selbstheilung
  - erleichtert die automatische Skalierung
- ★ Beispiel
  - Hochverfügbarer Service, z.B. Web-Shop

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

Deployment

Deployment
- **strategy: {type: RollingUpdate}**
- **replicas: 3**
- **selector:**
  - **app: my-app**



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bpmn-frontend
spec:
  replicas: 5
  selector:
    matchLabels:
      app: bpmn-frontend
  template:
    metadata:
      labels:
        app: bpmn-frontend
        group: web
        tier: frontend
    spec:
      containers:
      - name: bpmn-frontend
        image: misegr/bpmn-frontend:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
          name: bpmn-frontend
```

- ★ Deployment eines Services (App)
  - Ersetzt ein Container Image im Pod
  - Ausgerollt durch ReplicaSet
- ★ Ermöglicht Deklarative Updates (Deployments)
  - Erzeugt und Löscht (nach Update) automatisch ReplicaSet und Pods.

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Übung: Verteilung (09-3-replicaset)

Übung: Verteilung

In dieser Übung erstellen wir mehrere Pods ab dem gleichen Image mit jeweils einem ReplicaSet, Deployment und Service.

Das passiert in einer eigenen Namespace um die Resultate gezielt Darstellen zu können:

```
! kubectl create namespace rs
```

```
namespace/rs created
```

Erzeugen eines Deployments, hier der das Beispiel von Docker mit einem Web Server welche die aktuelle IP-Adresse ausgibt.

```
! kubectl run apache --image=dockercloud/hello-world --namespace rs
```

```
kubectl run --generator=deployment/apps.v1beta1 is DEPRECATED and will be removed in a future versior
ead.
deployment.apps/apache created
```

Ausgabe der Erzeugten Ergebnisse und die YAML Datei welche den Erzeugten Ressourcen beschreibt.

Ab `spec.containers` kommt erst der Pod.

```
! kubectl get pod,deployment,replicaset,service --namespace rs
```

★ Zusätzliche Übung:

- Welche Microservices aus dem Beispiel 09-2-Ingress sind mehrfach Auszuführen?
- Überträgt die notwendigen Befehle von 09-3-replicaset nach 09-2-Ingress.

Volumes

- ★ Speicherplatz welcher automatisch an den Pod angehängt wird.
  - Lokale Scratch-Verzeichnisse, die bei Bedarf erstellt werden
  - Cloud block storage
    - ★ GCE Persistent Disk
    - ★ AWS Elastic Block Storage
  - Cluster storage
    - ★ File: NFS, Gluster, Ceph
    - ★ Block: iSCSI, Cinder, Ceph
  - Special volumes
    - ★ Git repository
    - ★ Secret
- ★ Kritischer Baustein für die Automatisierung.

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/storage/volumes/

PersistentVolume, PersistentVolumeClaims

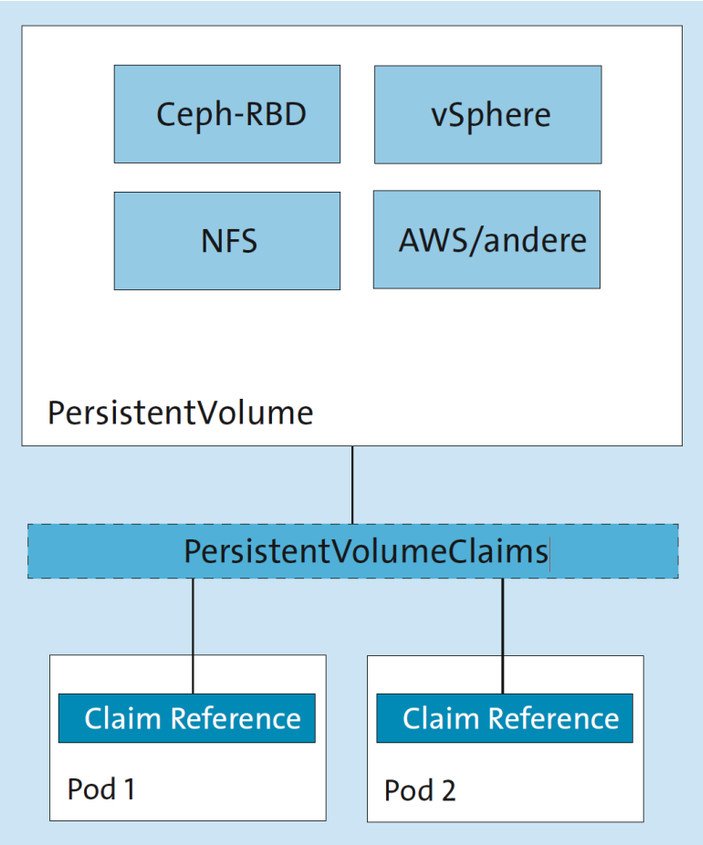


Abbildung 14.4 PersistentVolumes/PersistentVolumeClaims

- **PersistentVolume**: Bei PersistentVolumes (PV) handelt es sich um spezielle Storage Volume-Plugins, die Teile eines bestehenden SANs oder Storage Clusters auf vordefinierte Art an unsere Pods durchreichen können.
- Im Gegensatz zu den Volumes wird die Art, wie der Export des Volumes an den Pod erfolgt, abstrahiert.
- **PersistentVolumeClaims**: Der Pod mountet den Claim.

Hinweis

Als Storage-Ressource für ein PV muss nicht irgendein bestehendes SAN verwendet werden: Die PVs arbeiten mit den bereits vorgestellten PV-Typen wie Ceph/RBD, NFS, iSCSI, GlusterFS, GCEPersistentDisk oder auch AWSElasticBlockStore zusammen. Theoretisch auch mit hostPath, dies ist jedoch, wie bereits ausdrücklich erläutert, für Produktivumgebungen nicht empfehlenswert.

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Storage Classes

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azurefile
provisioner: kubernetes.io/azure-file
mountOptions:
  - dir_mode=0777
  - file_mode=0666
  - uid=0
  - gid=0
parameters:
  skuName: Standard_LRS
  storageAccount: k8sstore
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-claim
spec:
  accessModes:
  - ReadWriteMany
  storageClassName: azurefile
  resources:
    requests:
      storage: 5Gi
```

- ★ StorageClass bietet Administratoren die Möglichkeit, die "Klassen" des von ihnen angebotenen Speichers (grob: schnell/teuer, langsam/billig) zu beschreiben.
- ★ Verschiedene Klassen können Servicequalitätsstufen oder Sicherungsrichtlinien oder beliebigen Richtlinien zugeordnet werden, die von den Clusteradministratoren festgelegt werden.
- ★ Dieses Konzept wird in anderen Speichersystemen manchmal als "Profile" bezeichnet.
- ★ **Anwendung**: im Cloud Umfeld, z.B. Microsoft Azure: https://github.com/mc-b/lernkube/tree/master/azure#datenspeicherung

Übung: Volumes und Claims (09-5-hostPath)

Übung: Volume und Claims

Zuerst brauchen wir ein `PersistentVolume` worauf die `Claims` zusteuern.

Im nachfolgenden Beispiel wird der `hostPath` als Volume zur Verfügung gestellt:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: data-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/data"
```

Anschliessend folgen die `Claims`:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: data-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

Übung: Volumes und Container (09-6-volume)

Übung: Volume und mehrere Container in Pod

Diese Übung Demonstriert wie sich zwei Container innerhalb eines Pods das Verzeichnis `/usr/local/apache2/htdocs` teilen.

Der Container `apache` beinhaltet den Web Server und der Container `file-puller` schreibt alle 30 Sekunden die Datei `index.html` in das Verzeichnis `/usr/local/apache2/htdocs`.

Aus Einfachheitsgründen verwenden wir `emptyDir` als Volume.

Erläuterungen `emptyDir`

Das emptyDir-Volume wird angelegt, wenn ein Pod einem Node zugewiesen wird. Alle Container in dem Pod auf diesem (Worker-)Node können dieses emptyDir (einfach ein leeres Verzeichnis) lesen und schreiben.

Der Pfad, mit dem das emptyDir innerhalb eines Containers eingehängt wird, kann sich innerhalb der Container des Pods unterscheiden.

Sobald ein Pod von einem Node gelöscht wird, wird auch der Inhalt des emptyDir komplett und unwiederbringlich gelöscht. Selbst wenn der gleiche Pod auf dem gleichen Worker Node neu erstellt wird, kann er nicht mehr auf das Volume seines Vorgängers zugreifen. Dies bezieht sich nicht auf den Crash eines Containers des Pods.

Typische Anwendungsfälle für emptyDir-Vo

- Laufzeitdaten einer Applikation (Cach
- Übergabe von (Laufzeit-)Konfiguratior

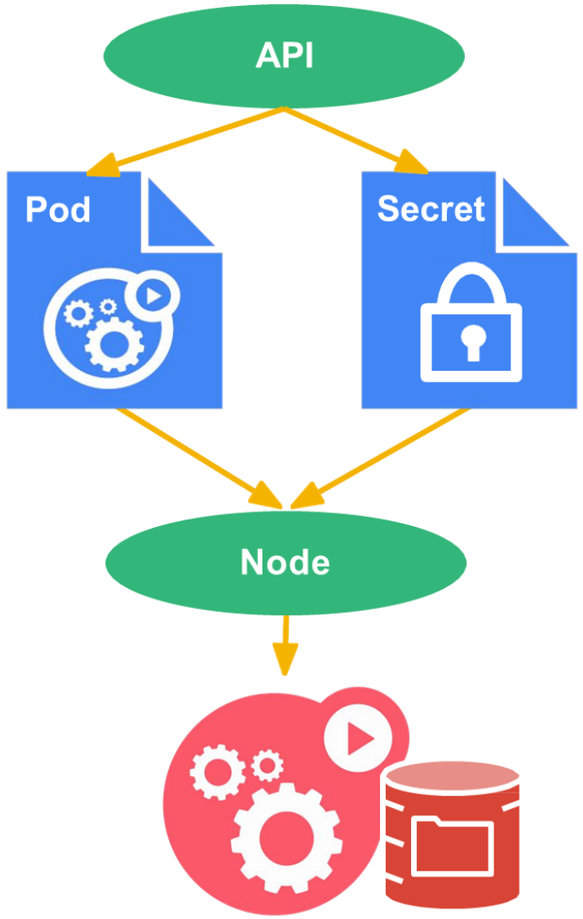
Zuerst schauen wir den Inhalt der YAML D

```
]: ! cat 09-5-Volume/web.yaml
```

Weitere Beispiele

- lernkube: https://github.com/mc-b/lernkube/tree/master/data
- AWS: https://github.com/mc-b/lernkube/tree/master/aws
- Azure: https://github.com/mc-b/lernkube/tree/master/azure
- Docker 4 Windows: https://github.com/mc-b/lernkube/tree/master/docker4windows

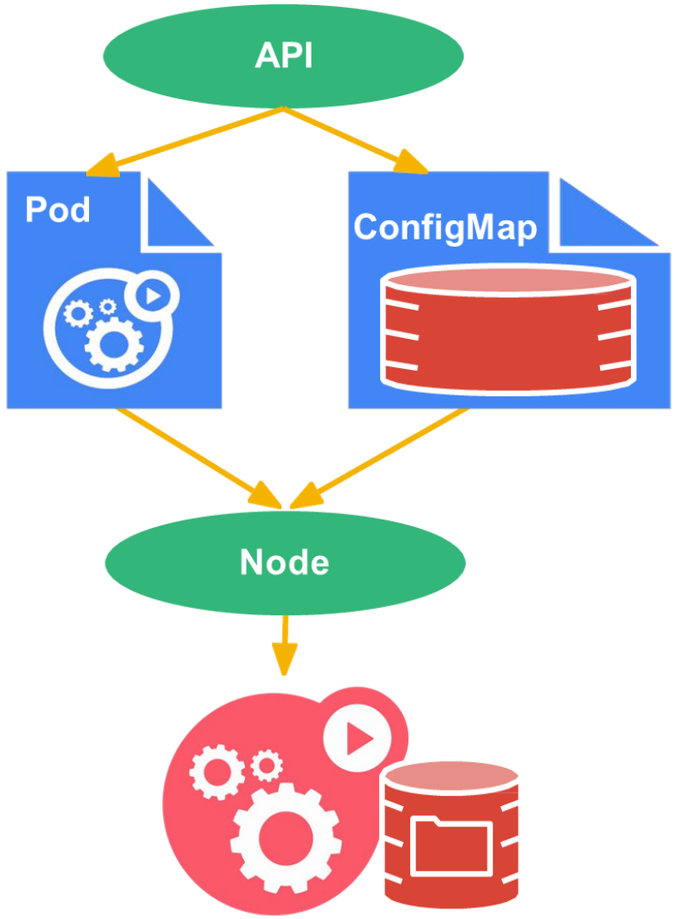
Secrets



- ★ Wie gewähre ich einem **Pod Zugriff** auf ein **gesichertes Objekt**?
  - secrets: credentials, tokens, passwords, ...
  - **Speichert sie nicht im Container**!
- ★ 12-factor says should come from the environment
- ★ Inject them as “virtual volumes” into Pods
  - not baked into images nor pod configs
  - kept in memory - never touches disk
  - not coupled to non-portable metadata API
- ★ Manage secrets via the Kubernetes API
- ★ How to explain Kubernetes Secrets …

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/configuration/secret/

ConfigMap



- ConfigMaps bieten unter K8s eine effiziente Möglichkeit, Pods auch mit komplexeren (Re-)Konfigurationen abbilden bzw. zu versorgen.
- ConfigMaps können u.a. Kommandozeilenoptionen, Umgebungsvariablen und Konfigurationsdateien sein.
- Idee ist, dass Image so standardisiert wie möglich betreiben zu können.
- D.h. Entkoppeln der Optionen, Direktiven, Umgebungsvariablen und Konfigurationsdateien vom Image.
- Über die ConfigMap-API können wir Parameter und Dateien zur Laufzeit bzw. bei der Aktivierung des Containers injizieren.
- Tutorial: https://medium.com/google-cloud/kubernetes-configmaps-and-secrets-68d061f7ab5b

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/

Übung: Secrets (09-7-secrets)

Übung: Volume und Claims

Zuerst brauchen wir ein `PersistentVolume` worauf die `Claims` zusteuern.

Im nachfolgenden Beispiel wird der `hostPath` als Volume zur Verfügung gestellt:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: data-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/data"
```

Anschliessend folgen die `Claims`:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: data-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

Weitere Beispiele

- Deploying WordPress and MySQL with Persistent Volumes

Übung: Configmap (09-8-configmap)

Übung: ConfigMap

Am Beispiel von Apache Web Server soll der Einsatz von ConfigMap Demonstriert werden.

Zuerst müssen die ConfigMaps mittels `kubectl` erstellt werden. Vorher erstellen wir eine neue Namespace `configmap` um Erstellen Ressourcen von den anderen zu trennen:

In [ ]:
```
! kubectl create namespace configmap
! kubectl create configmap web2 --from-file=index=09-8-ConfigMap/index-2.html -n configmap
! kubectl create configmap web1 --from-file=index=09-8-ConfigMap/index-1.html -n configmap
```

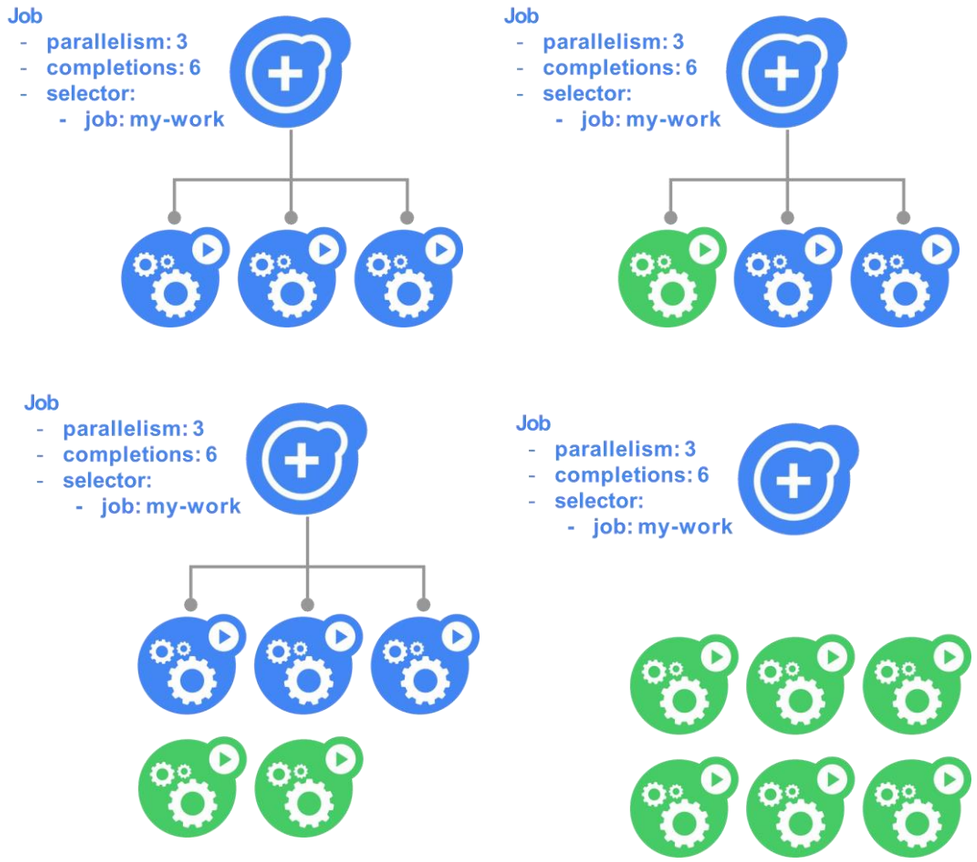
Die so erstellen ConfigMaps, hätten wir auch im YAML Format beschreiben können. Deshalb geben wir uns diese im YAML Format aus:

In [ ]:
```
! kubectl get configmap -o yaml
```

Anschliessend sind die Volumes in der YAML Datei so zu setzen, dass sie auf die ConfigMap Keys zugreifen

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app.kubernetes.io/name: web1
  name: web1
  namespace: configmap
spec:
  containers:
  - image: httpd
    name: apache
    volumeMounts:
     - name: config-volume
       mountPath: /usr/local/apache2/htdocs
  volumes:
    - name: config-volume
      configMap:
        name: web1
        items:
        - key: index
          path: index.html
```

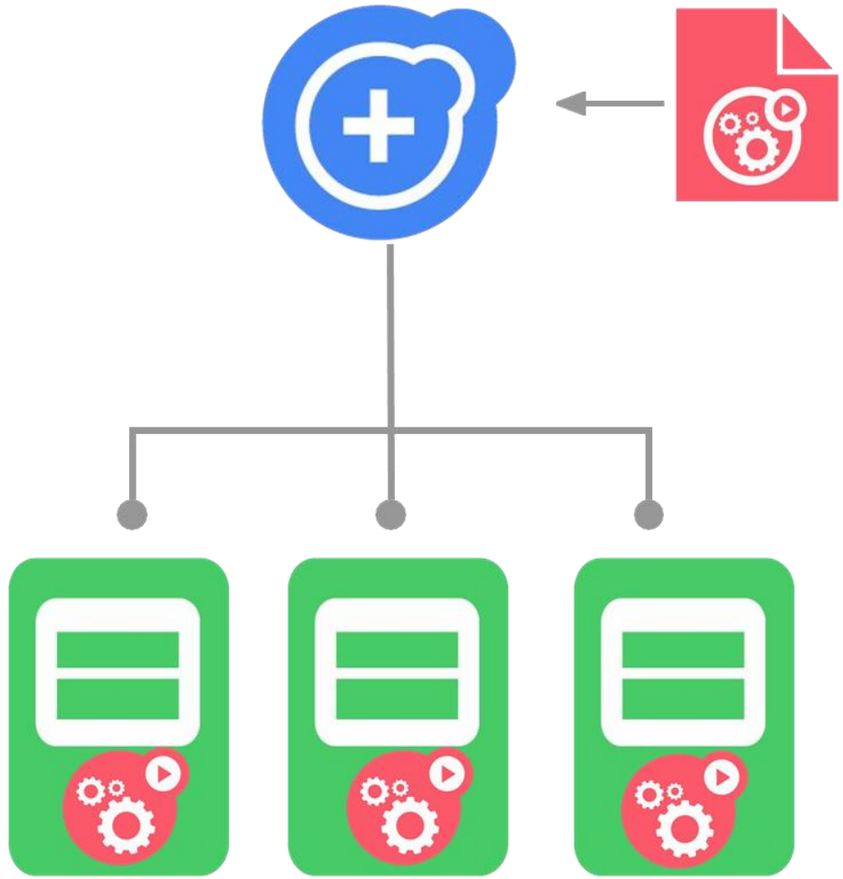
Jobs (Bestandteil vom Vertiefungsmodul)



- ★ Verwaltet Pods, die als Jobs ausgeführt werden
- ★ Ermöglicht die Anzahl der gleichzeitig ausgeführten Jobs und eine Gesamtzahl der Jobs, die durchgeführt werden müssen, festzulegen.
- ★ Ähnlich wie ReplicationSet, aber für Pods, die nicht immer neu gestartet werden
- ★ Workflow:
  - Neustart bei Fehler
  - Kontrolle ob sauber beendet
- ★ Prinzip: Aufteilen und parallelisieren von Jobs, z.B. bei grossen Berechnungen.

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/

DaemonSet

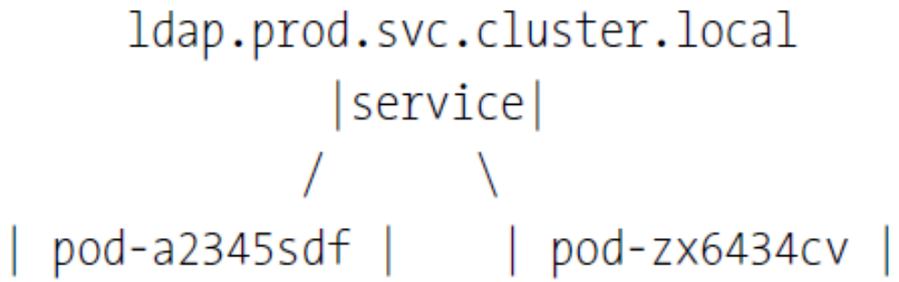


- ★ Startet einen Pods auf jeder Node
  - oder ausgewählten Nodes
- ★ Keine fixe Anzahl von Replicas (Instanzen)
  - erstellt und beendet durch hinzufügen oder wegnehmen von Nodes.
- ★ Sinnvoll für Cluster weite Services wie
  - Sammeln und Auswerten von Logfiles
  - Lokale Persistenz, Caching
- ★ Der DaemonSet Manager ist Controller und Scheduler

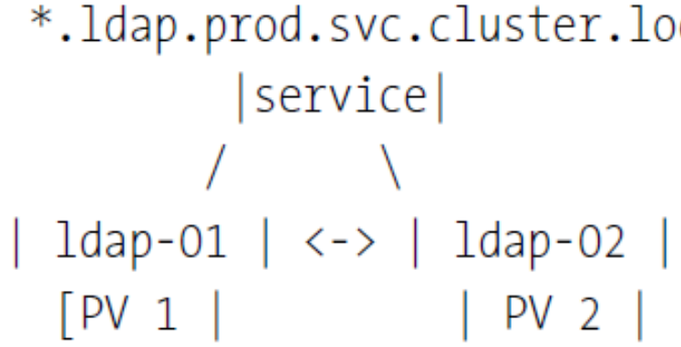
Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

StatefulSets

★ Standard Zuordnung Service, Pods mit Hash



★ StatefulSets, mit fester Namensvergabe und fest zugeordneten PersistentVolumes:



★ **StatefulSets:** klassische «Long Running» Services mit persistenter, hochverfügbarer Datenablage, statischen IPs (hinter einer Service-IP) und fester Namensvergabe.

★ Wichtigste Punkte:

- persistenter Storage für die Datenablage der StatefulSets, typischerweise Cluster-/Netzwerk-Dateisysteme
- eine feste Zuordnung im Hinblick auf IP und Namensvergabe
- eine K8s-Headless-Service-Ressource, die unabhängig von einem Deployment oder ReplicaSet erzeugt wurde, also für sich allein arbeiten kann und keine diesbezüglichen Abhängigkeiten besitzt
- StatefulSets müssen manuell aktualisiert werden.
- …

Buch Skalierbare Container-Infrastrukturen, Kapitel 13
https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

Reflexion

- ★ Kubernetes stellt eine Vielzahl von Ressourcen bereit.
- ★ Diese werden, vorzugsweise, in YAML Dateien beschrieben.
- ★ Die Ressourcen können mittels dem CLI `kubectl` oder über das K8s Dashboard verwaltet werden.
- ★ Detailinformationen zu jeder Ressource liefert
  - `kubectl explain <Ressource>`

Lernzielkontrolle

★ Sie haben einen Überblick über die Ressourcen im K8s Cluster und können diese Einordnen.