

Verstehen und Einordnen von Ressourcen im K8s Cluster

»Es braucht Courage, aufzustehen und zu reden. Genauso braucht es Courage, sich hinzusetzen und zuzuhören.«

Winston Churchill



Lernziele

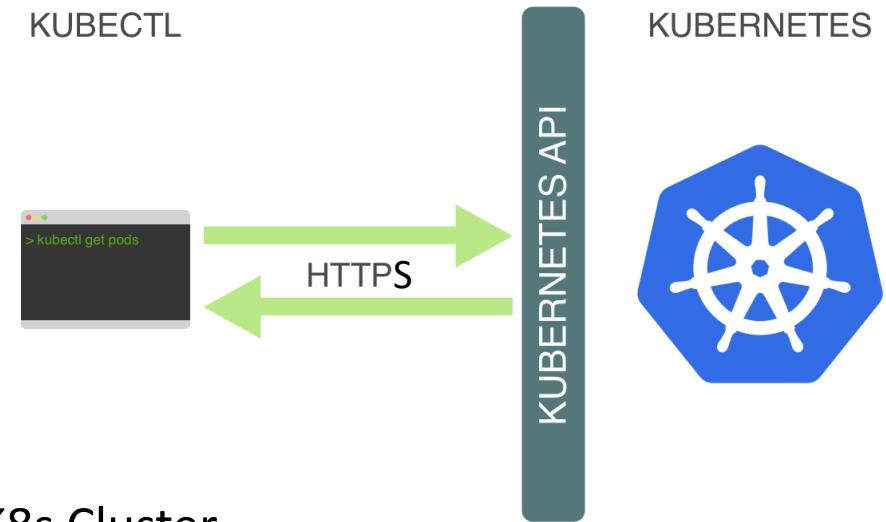
- ★ Sie haben einen Überblick über die Ressourcen im K8s Cluster und können diese Einordnen.

Zeitlicher Ablauf

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

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

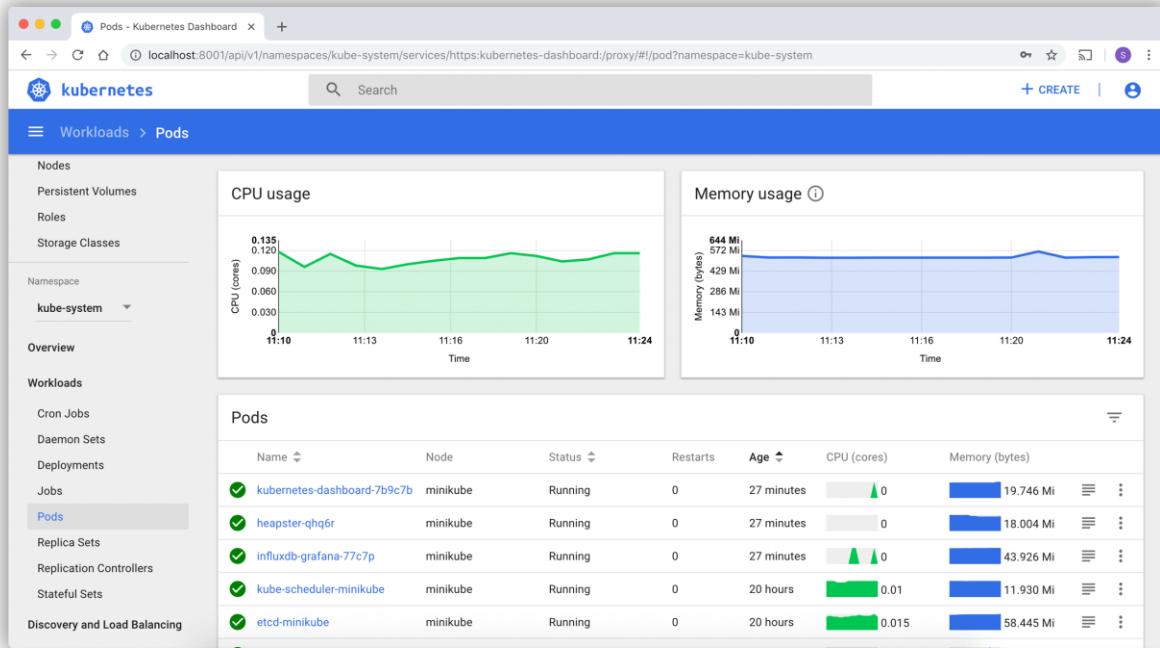


Kubectl-CLI (2)

- `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 Cluster an lokales System weiterleiten (siehe bin/weave Datei)
- `kubectl proxy` - Aufrufe an den API Service ab lokalem System ermöglichen (siehe bin/dashboard Datei)

➤ [kubectl Cheatsheet](#)

Web UI (Dashboard)



- ★ Das Dashboard ist eine webbasierte Kubernetes-Benutzeroberfläche.
- ★ Sie können das Dashboard verwenden, um containerisierte Anwendungen in einem Kubernetes-Cluster bereitzustellen, Probleme mit Ihrer containerisierten Anwendung zu beheben und die Clusterressourcen zu verwalten.
- ★ Sie können das Dashboard verwenden, um einen Überblick über die auf Ihrem Cluster ausgeführten Anwendungen zu erhalten und um einzelne Kubernetes-Ressourcen (wie Deployments, Jobs, DaemonSets usw.) zu erstellen oder zu ändern.
- ★ Das Dashboard enthält auch Informationen zum Status der Kubernetes-Ressourcen in Ihrem Cluster und zu eventuell aufgetretenen Fehlern.

- ★ Starten im CLI mittels: `./dashboard.bat`
- ★ **Einloggen** mittels **Token** welcher von `kubeps.bat` oder `source kubeenv` angezeigt wird.
- ★ <https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/>

Weave Scope



- ★ Weave Scope ist ein Visualisierungs- und Überwachungstool für Docker und Kubernetes.
- ★ Es bietet eine Top-Down-Ansicht Ihrer App sowie Ihrer gesamten Infrastruktur und ermöglicht es Ihnen, Probleme mit Ihrer verteilten containerisierten App in Echtzeit zu diagnostizieren, während diese bei einem Cloud-Anbieter bereitgestellt wird.

- ★ Starten im CLI mittels: `weave`
- ★ <https://www.weave.works/docs/scope/latest/introducing/>

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	

z.B. kubectl get po
kubectl get pods

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	

Ausgabe Liste aller Ressourcen
kubectl api-resources

YAML (Dateien)

```
# 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
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 Auszeichnungssprache (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.

API-Versionierungsdickicht

Inhalt YAML Datei

```
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 getester 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

API-Gruppen

Inhalt YAML Datei

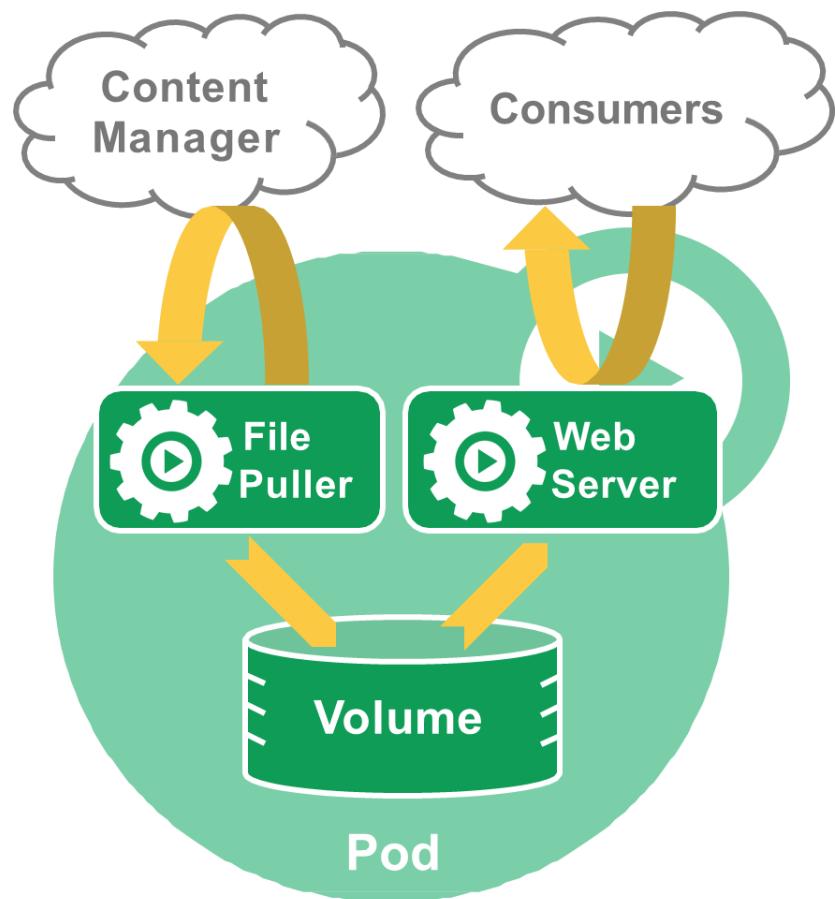
```
apiVersion: extensions/v1beta1  
kind: Deployment
```

- ★ 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](#).

Pods (1)

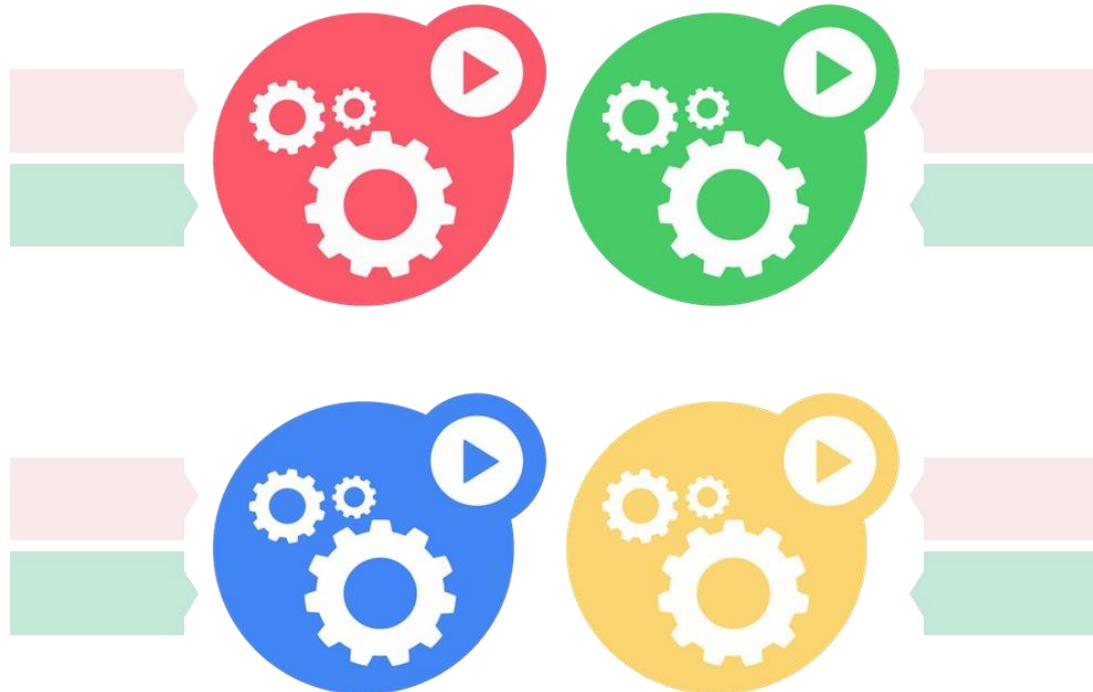


Pods (2)



- ★ 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).

Labels



- ★ Vom User bereitgestellte **Schlüssel** und **Werte** (key=value)
- ★ Kennzeichnung eines API-Objekt
labels:
`app: jupyter-base`
`group: web`
`tier: frontend`
- ★ Abfragbar mittels Selektoren (selectors), ähnlich SQL
`kubectl get pods -l tier=frontend`
- ★ Der **einige** Gruppierungsmechanismus

Selectors (1)

app: my-app
track: stable
tier: FE



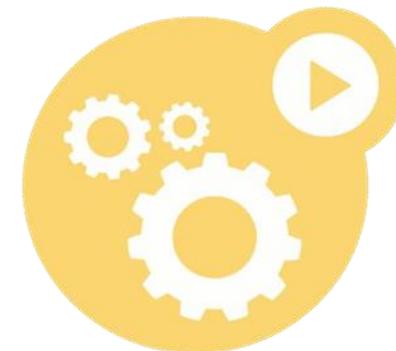
app: my-app
track: canary
tier: FE



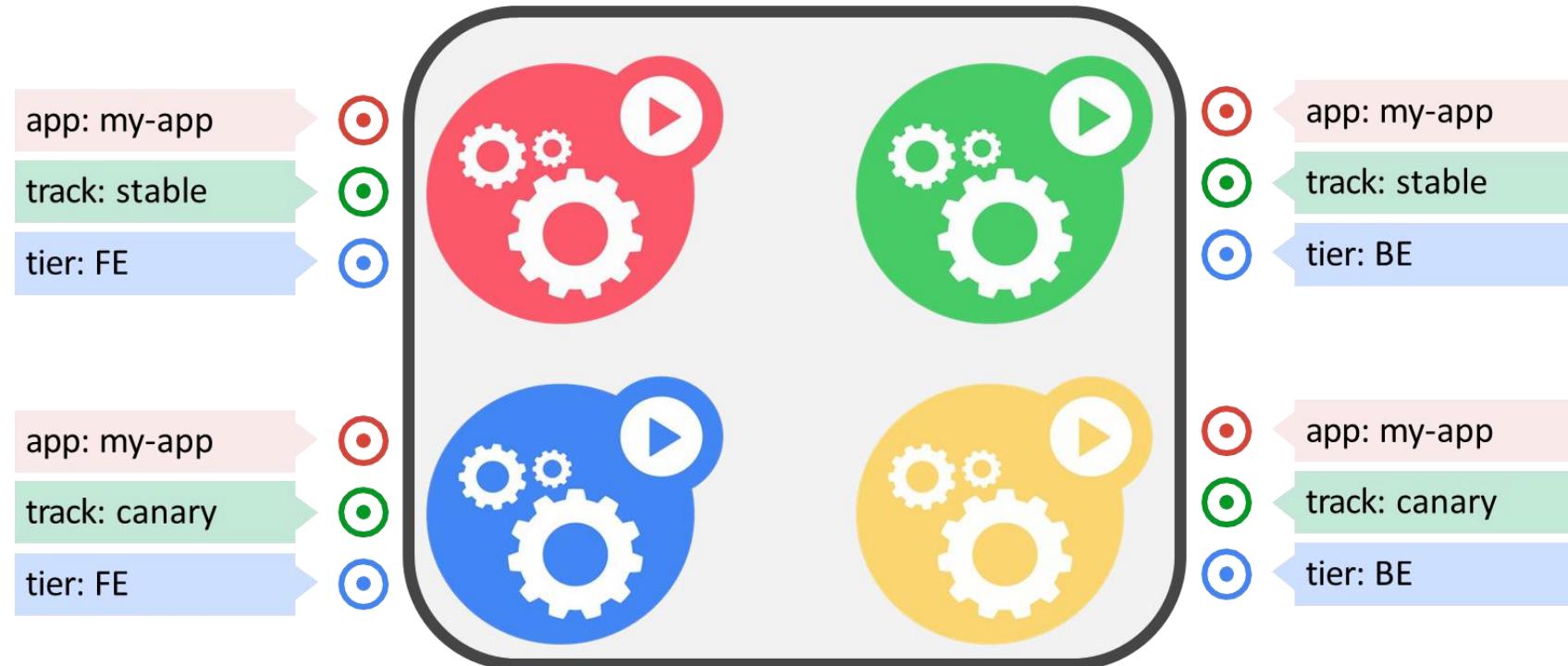
app: my-app
track: stable
tier: BE



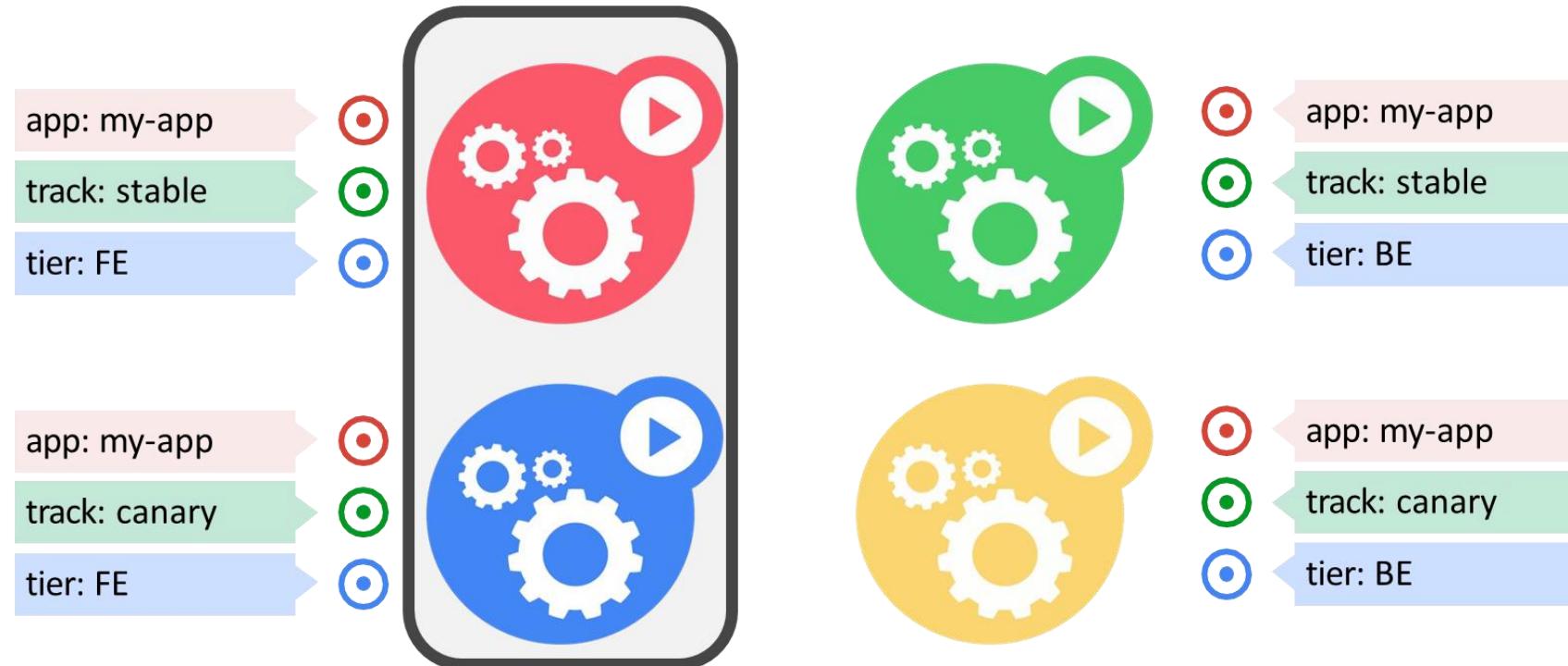
app: my-app
track: canary
tier: BE



Selectors (2)

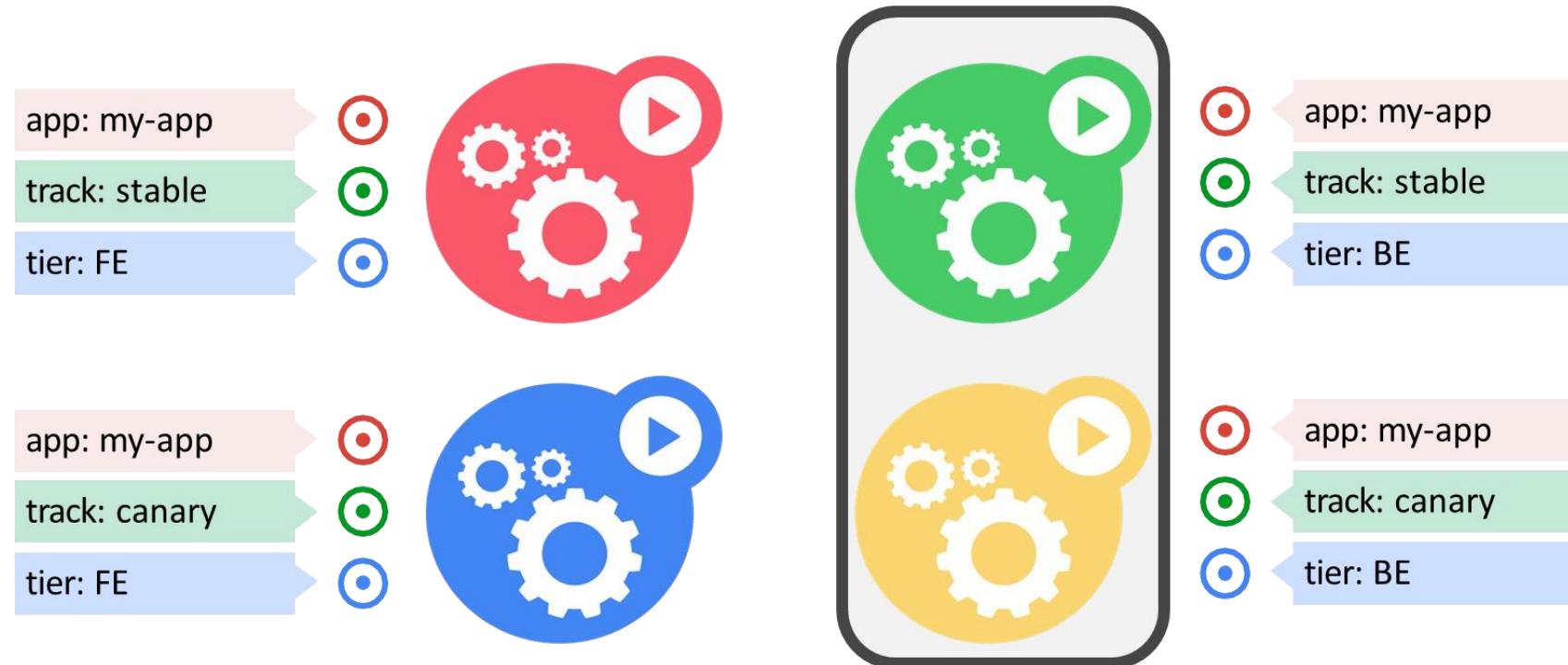


Selectors (3)



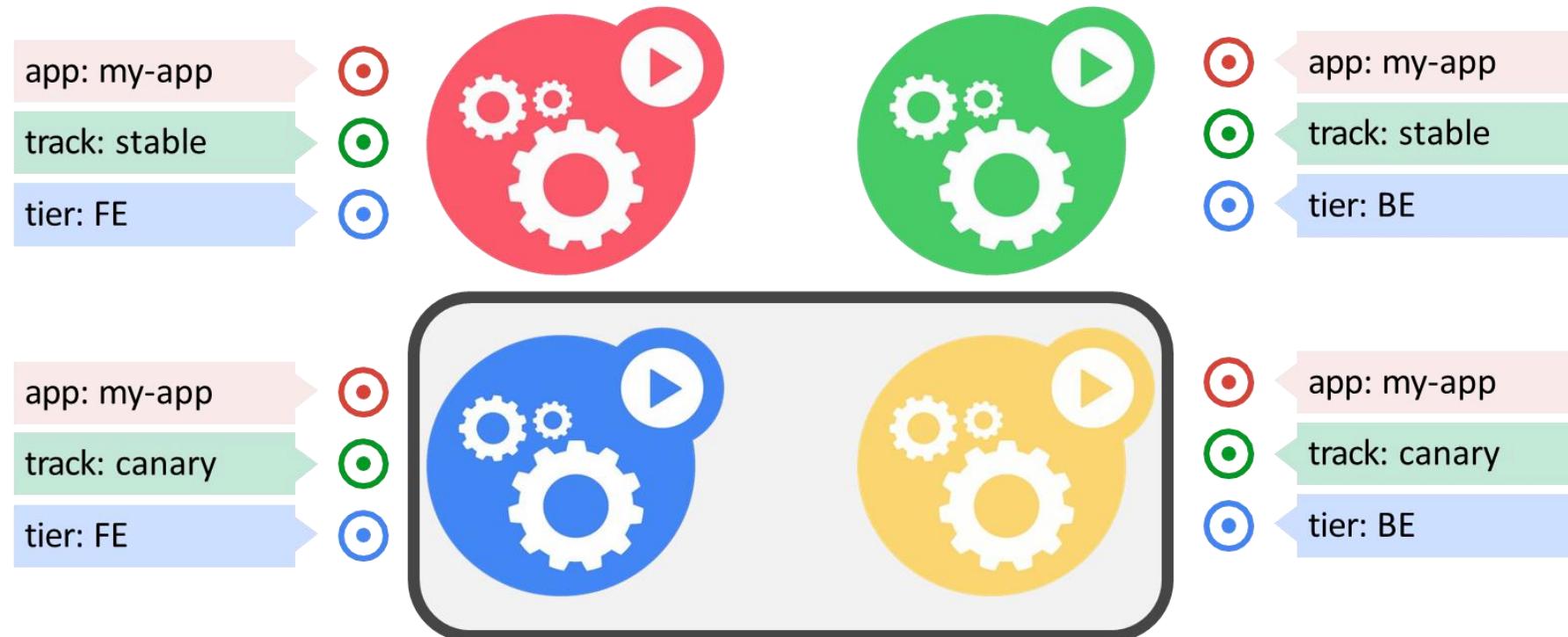
app = my-app, tier= FE

Selectors (4)



app = my-app, tier= BE

Selectors (5)



app = my-app, track = canary

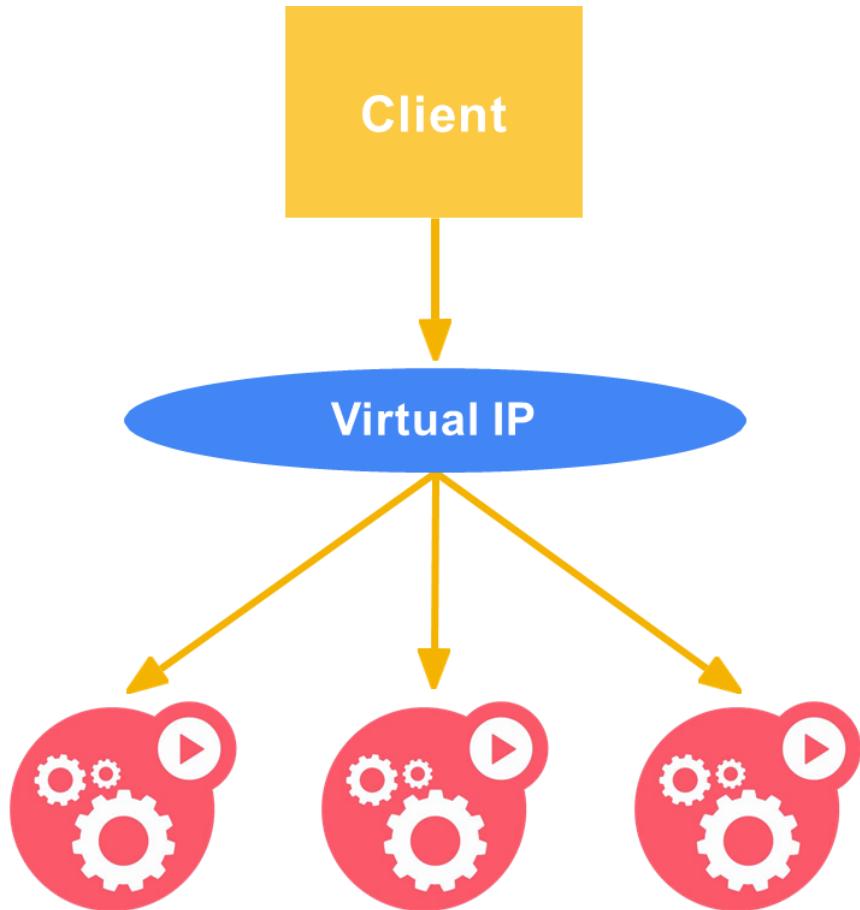
Recommended Labels (Empfohlene Labels)

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

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

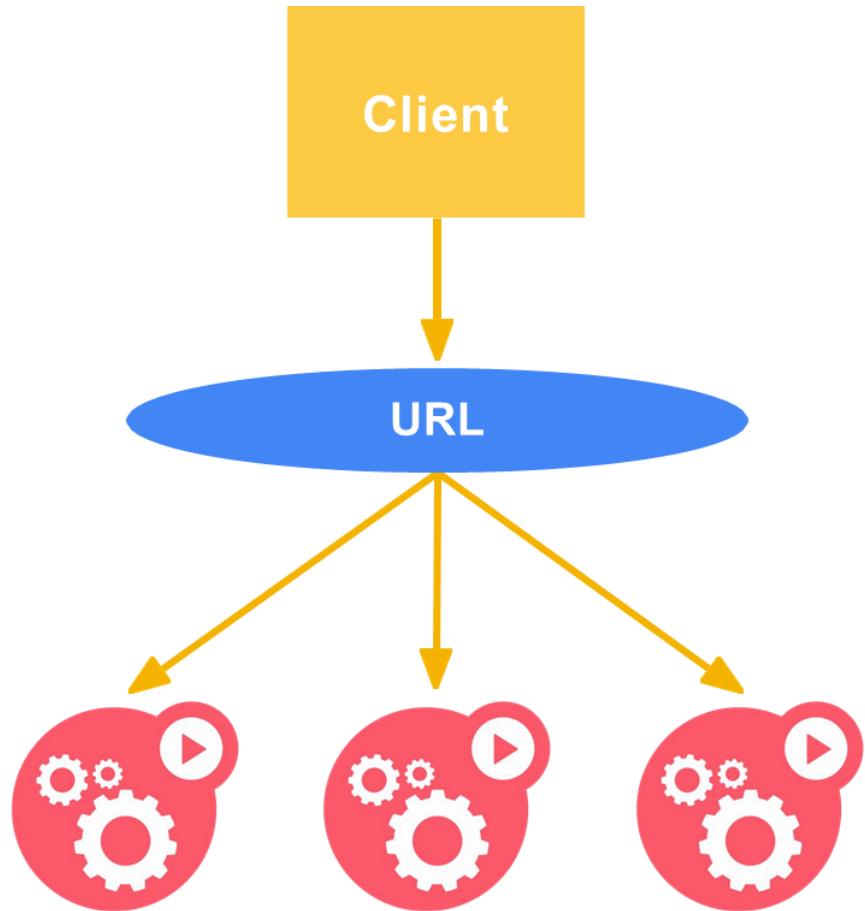
```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app.kubernetes.io/name: mysql
    app.kubernetes.io/instance: wordpress-abcxyz
    app.kubernetes.io/version: "5.7.21"
    app.kubernetes.io/component: database
    app.kubernetes.io/part-of: wordpress
    app.kubernetes.io/managed-by: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

Services



- ★ Eine Gruppe von Pods die zusammenarbeiten, Gruppiert mittels Label Selector
- ★ Erlaubt mittels unterschiedlichen Methoden auf den Service zuzugreifen:
 - DNS name
 - [DNS SRV](#) (Service Resource Record)
 - [\(Kubernetes Endpoints API\)](#)
- ★ Definiert Zugriffsrichtlinien (access policy):
 - Load-balanced: Name zeigt auf eine virtuelle IP
 - Headless: Name zeigt auf eine Liste von Pods
 - Port remapping: Interner Port wird nach aussen sichtbar
- ★ Versteckt Komplexität.
- ★ Beispiel (Jupyter Oberfläche)
 - Aussen: <http://<Cluster>:32188>
 - Innen: <http://jupyter-base:8888>

Ingress



- ★ 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. Weitere Varianten sind mittels Ingress Controllers möglich.

- ★ Beispiel:
 - <https://<Cluster>:30443/frontend/index.html>
- ★ Erfahrungsbericht Ingress
 - <https://itnext.io/kubernetes-ingress-controllers-how-to-choose-the-right-one-part-1-41d3554978d2>

Namespaces

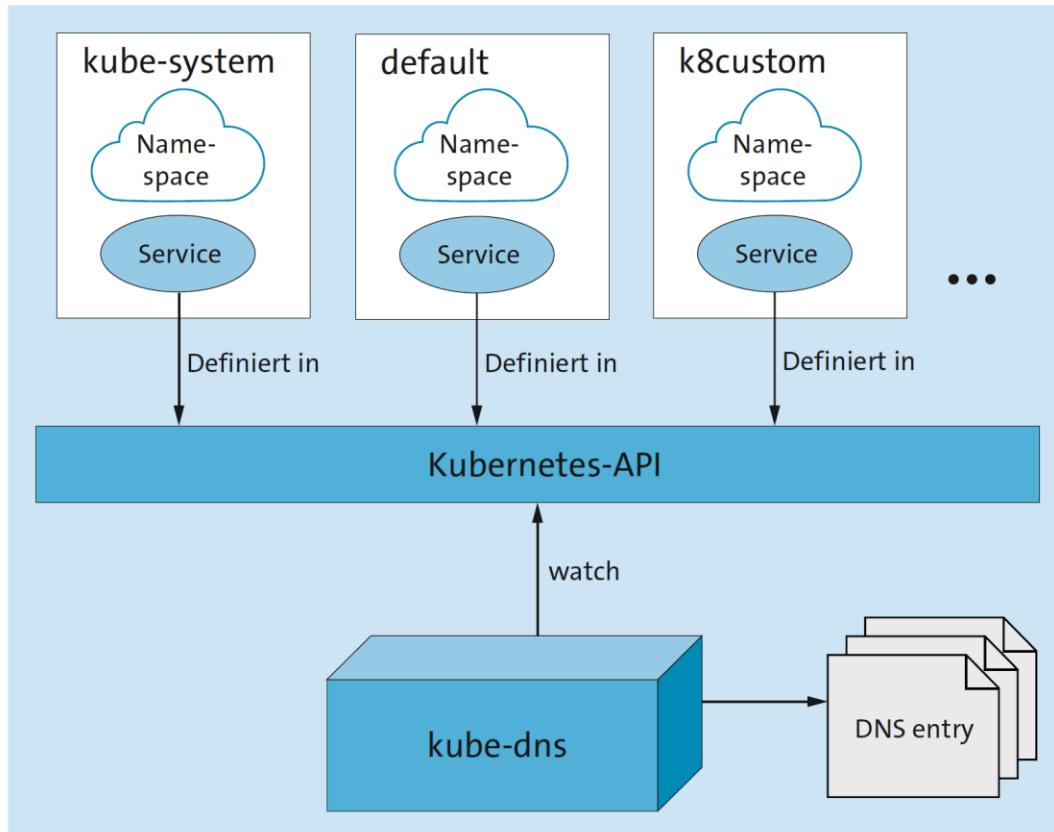


Abbildung 14.6 Zusammenspiel K8s DNS und Namespaces

- ★ Unterteilen den gesamten K8s Cluster in logische Partitionen bzw. Bereiche.
- ★ Standard Namespaces
 - **default:** In diesem Namespace werden alle neuen erzeugt, sofern wir nichts anderes angeben.
 - **kube-system:** Kubernetes-Systemdienste.
- ★ Ressourcen in Namespaces können limitiert werden (Anzahl Pod's, CPU, Memory etc.).
- ★ Der Netzwerktrafik kann, mittels [Network Policy Provider](#), eingegrenzt werden
- ★ **Achtung:** Wird ein Namespace gelöscht, werden damit auch alle in ihm gehosteten Objekte/Ressourcen gelöscht!



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.

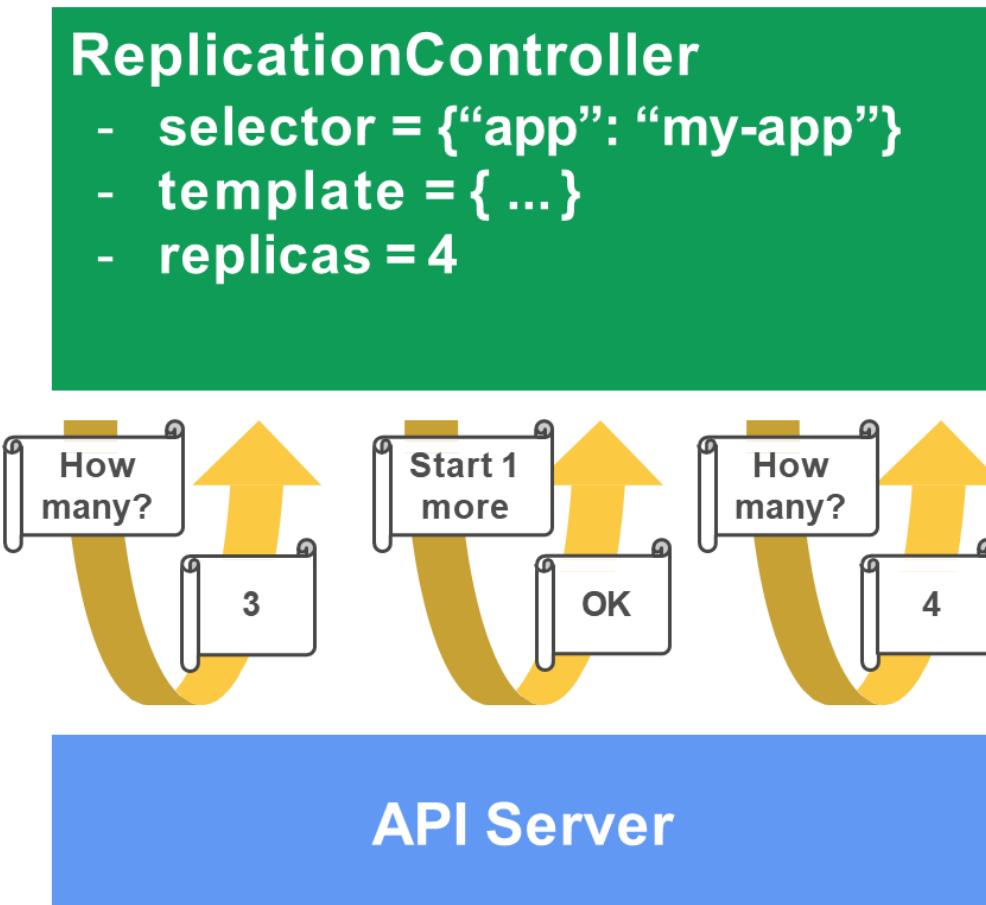
Das passiert in einer eigenen Namespace um die Resultate gezielt Darstellen zu können:

```
! 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
```

Replication Controller (Neu ReplicaSets)

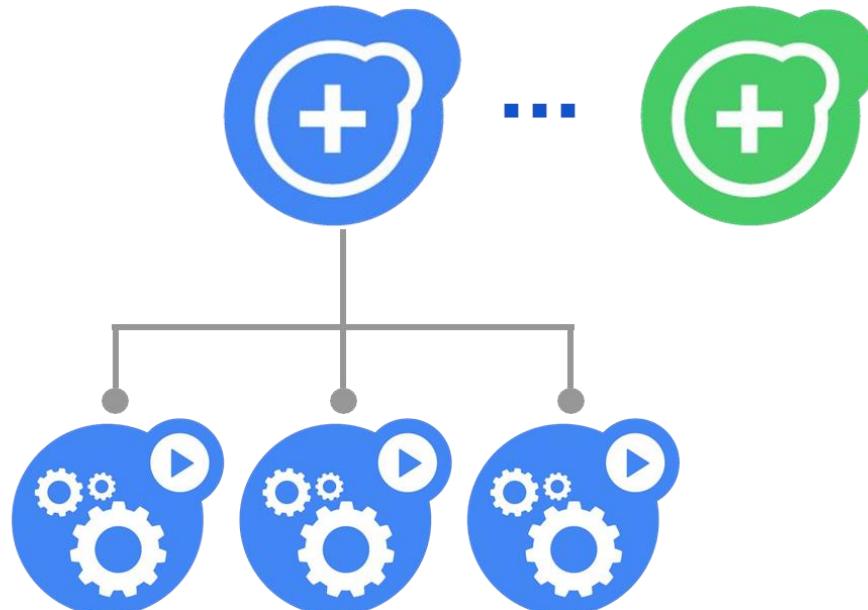


- ★ Stellt sicher, dass N Pods laufen
 - sind es zu wenig, werden neue gestartet
 - sind es zu viele werden Pods beendet
 - gruppiert durch den Label Selector
- ★ Explizite Deklaration der gewünschten Anzahl Pods
 - ermöglicht Selbstheilung
 - erleichtert die automatische Skalierung
- ★ Beispiel
 - Hochverfügbarer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

Deployment

Deployment

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



- ★ 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.



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.

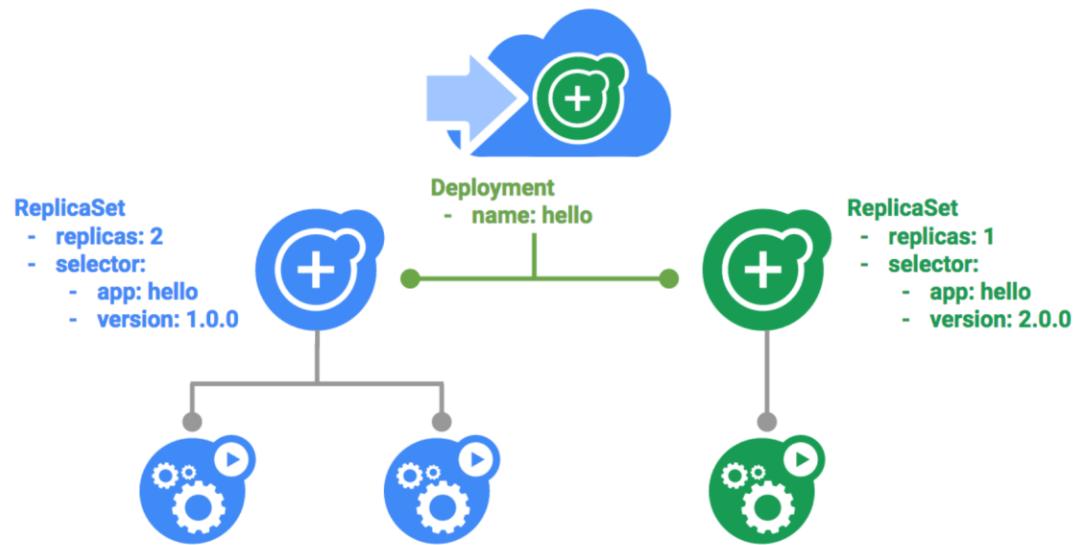
```
! kubectl run apache --image=dockercloud/hello-world --namespace rs  
kubectl run --generator=deployment/apps.v1beta1 is DEPRECATED and will be removed in a future version.  
deployment.apps/apache created
```

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

Ab `spec.containers` kommt erst der Pod.

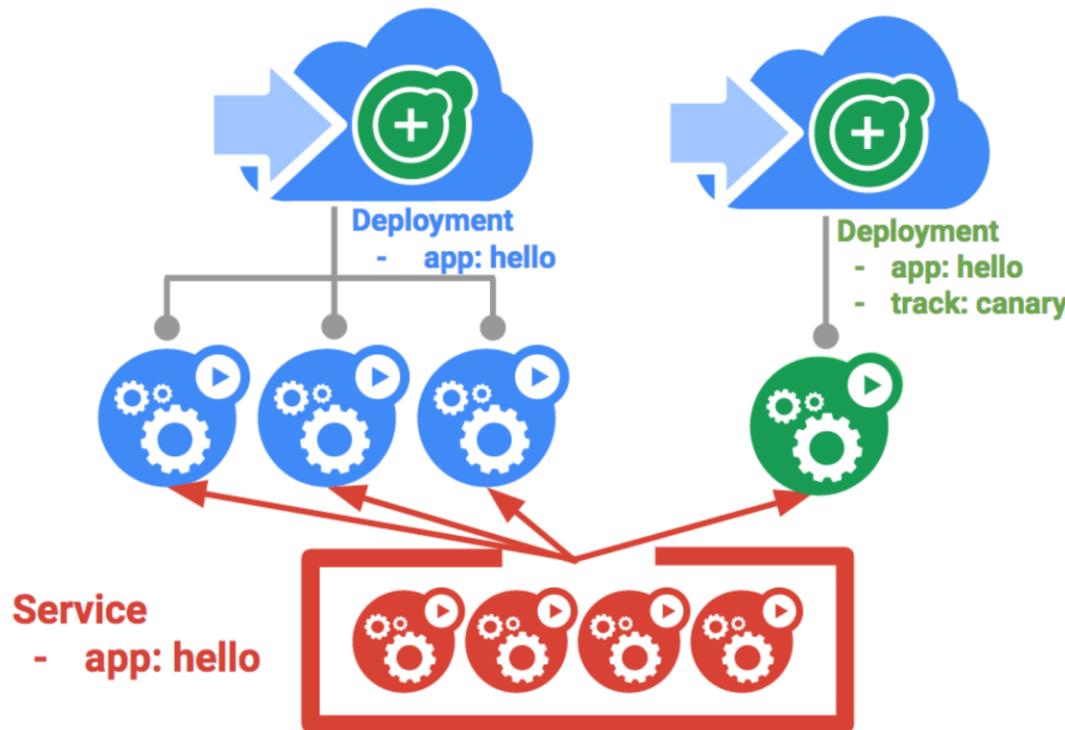
```
! kubectl get pod,deployment,replicaset,service --namespace rs
```

Rolling Update



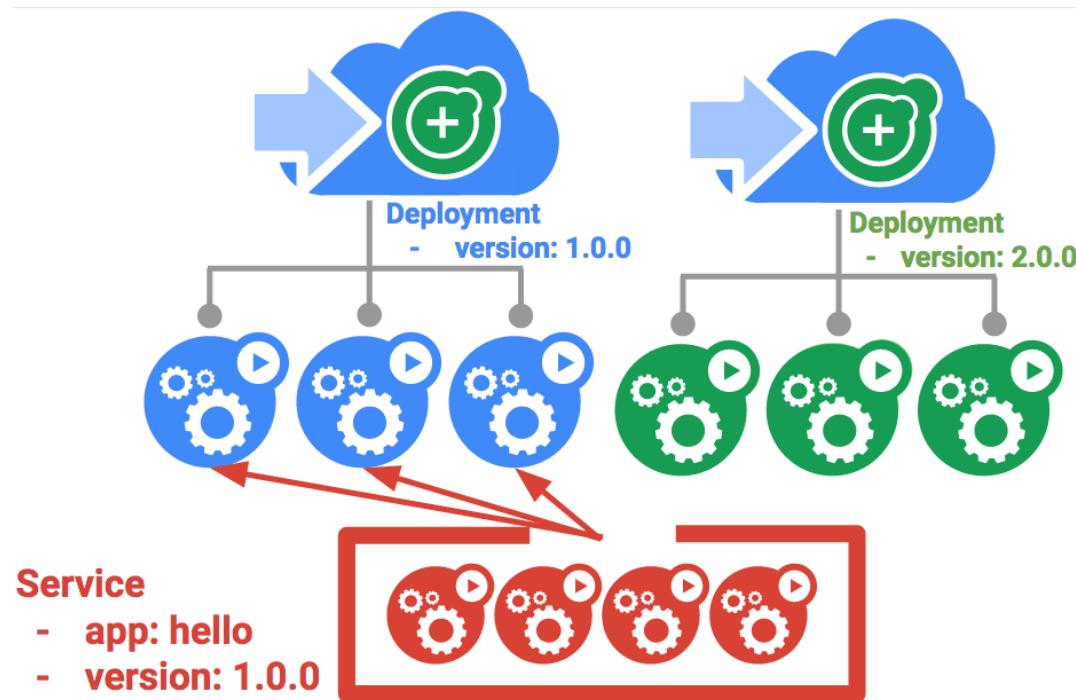
- ★ **Deployments** unterstützen das **Aktualisieren** von **Image** auf eine neue Version mithilfe eines fortlaufenden Aktualisierungsmechanismus.
- ★ Wenn ein Deployment mit einer neuen Version aktualisiert wird, erstellt es ein neues ReplicaSet und erhöht kontinuierlich die Anzahl der Replikate im neuen ReplicaSet, da die Replikate im alten ReplicaSet verringert werden.

Canary Deployments



- ★ Wenn Sie eine neue SW Version in der Produktion mit einer **kleinen Gruppe** Ihrer Benutzer testen möchten, können Sie ein «Canary» Deployment durchführen.
- ★ Bei «**Canary**» Deployment können Sie eine Änderung nur einer **kleinen Gruppe** Ihrer Benutzer **freigeben**, um das Risiko von neuen Versionen zu verringern.

Blue/Green Deployments



- ★ Rolling-Updates sind ideal, da sie Ihnen ermöglichen, eine Anwendung langsam mit minimalem Overhead, minimalen Auswirkungen auf die Leistung und minimalen Ausfallzeiten bereitzustellen.
- ★ Es gibt jedoch Fälle, in denen es sinnvoll ist, die Load Balancer so zu ändern, dass sie **erst nach der vollständigen Verteilung** auf die **neue Version** verweisen.
- ★ In diesem Fall sind so genannte Blue-Green-Bereitstellungen der richtige Weg.



Übung: Rolling Update (09-4-update)

Übung: Rolling Update

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

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

```
: ! kubectl create namespace dep1
```

Wir Erzeugen den Pod, ReplicaSet, Deployment, Service und Ingress mittels einer YAML Datei.

```
: ! wget https://raw.githubusercontent.com/mc-b/misegr/master/bpmn/bpmn-frontend.yaml >/dev/null 2>&1
! cat bpmn-frontend.yaml
```

```
: ! kubectl apply -f https://raw.githubusercontent.com/mc-b/misegr/master/bpmn/bpmn-frontend.yaml --namespace dep1
```

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,ingress --namespace dep1
```

Schauen wir uns zuerst die Ausgabe des laufenden Pods an, diesmal über den URL des Kubernetes API Servers und den via Ingress Ressourcen angehängten Prefix.

Der Prefix `frontend` ist fix in der Ingress Ressource hinterlegt, `index.html` ergibt sich wie der Container aufgebaut ist (Apache Server mit einer HTML Datei).

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.

PersistentVolume, PersistentVolumeClaims

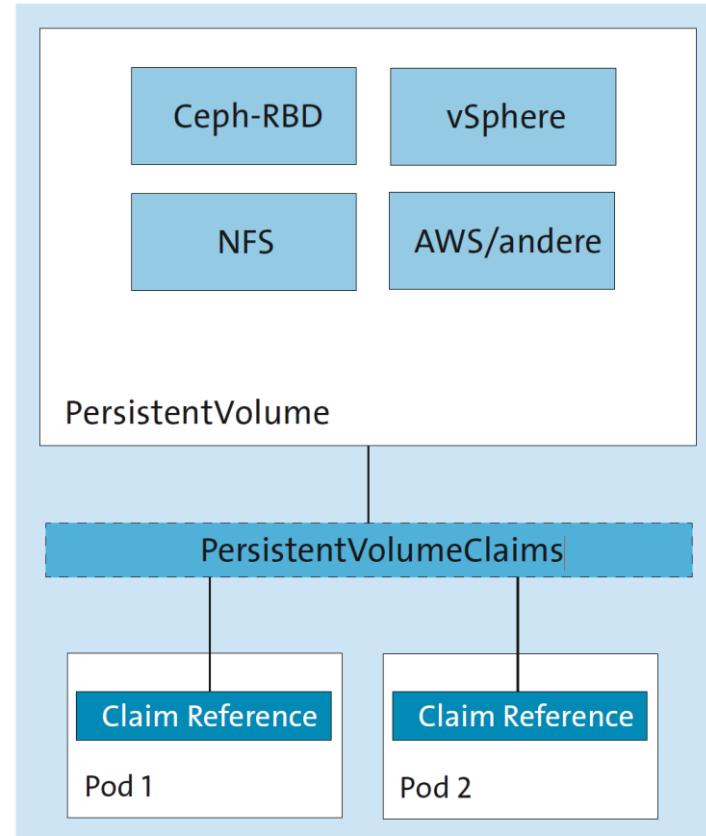


Abbildung 14.4 PersistentVolumes/PersistentVolumeClaims

★ **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.

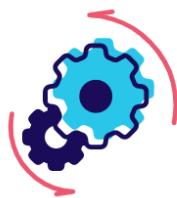
Storage Classes

```
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
```

```
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>

Persistenz: Rook a Storage Orchestrator for K8s



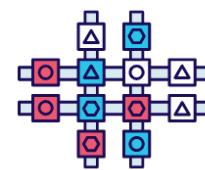
Simple and reliable automated resource management



Hyper-scale or hyper-converge your storage clusters



Efficiently distribute and replicate data to minimize loss



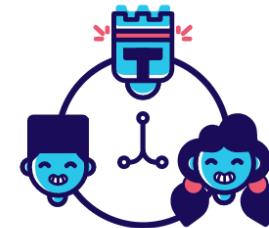
Provision, file, block, and object with multiple storage providers



Manage open-source storage technologies



Easily enable elastic storage in your datacenter



Open source software released under the Apache 2.0 license



Optimize workloads on commodity hardware

<https://rook.io/>

Container Storage Interface (CSI) ab V1.13!



- ★ CSI Kubernetes ist ein leistungsstarkes Volume-Plugin-System um Unterstützung für neue Volume-Plugins hinzuzufügen:
- ★ Volume-Plugins waren "in-tree", d.h. ihr Code war Teil des Kern-Kubernetes-Codes und wurde mit den Kern-Kubernetes-Binärdateien ausgeliefert.
- ★ Weitere Informationen:
- ★ <https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/>
- ★ <https://kubernetes.io/docs/concepts/storage/volumes/#csi>



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.

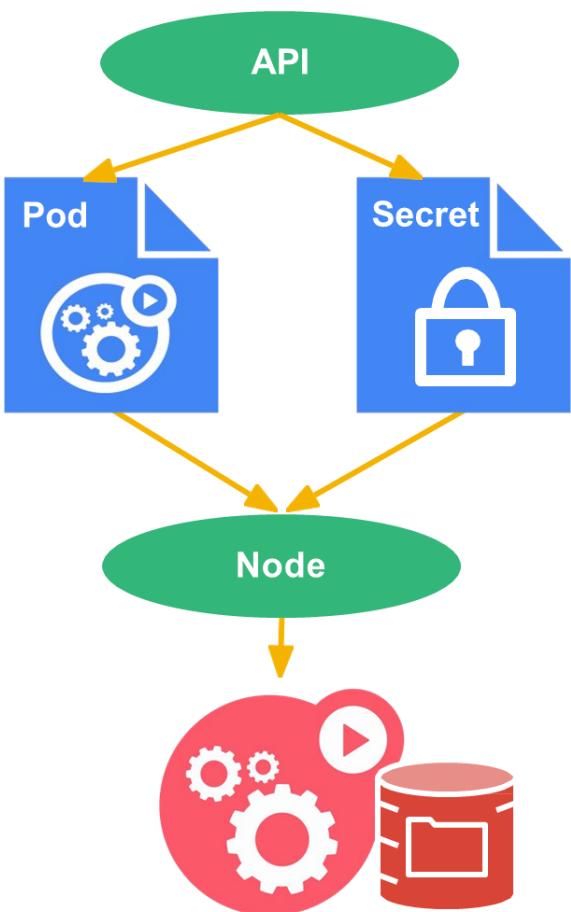
Sobald ein Pod von einem Node gelöscht wird, wird auch der Inhalt des `emptyDir` komplett und unwiederbringlich gelöscht. Selbst wenn der gleiche Pod auf dem gleichen Worker Node neu erstellt wird, kann er nicht mehr auf das Volume seines Vorgängers zugreifen. Dies bezieht sich nicht auf den Crash eines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

Typische Anwendungsfälle für `emptyDir`:

Weitere Beispiele

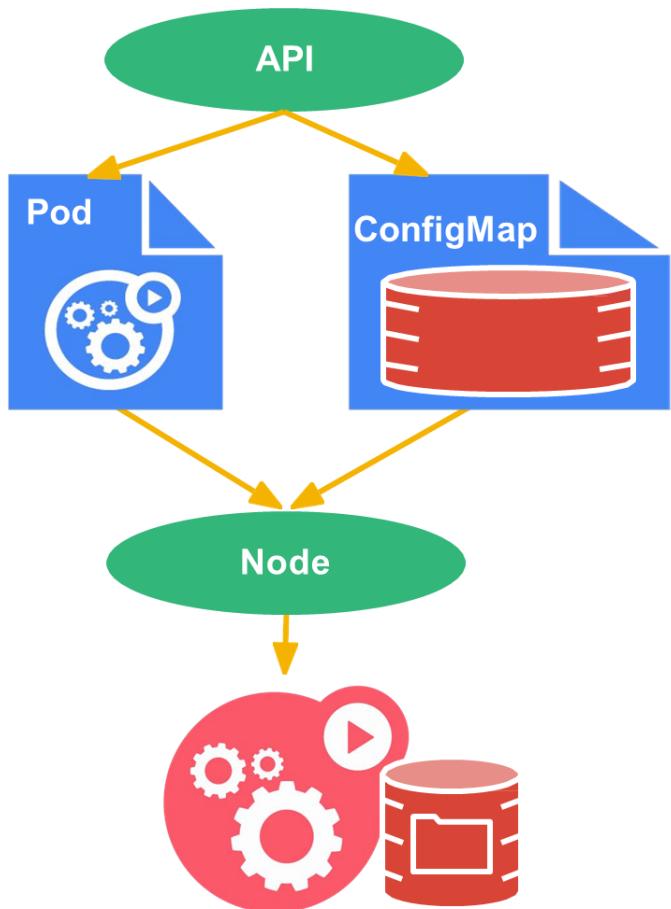
- Laufzeitdaten einer Applikation (Cache)
 - Übergabe von (Laufzeit-)Konfigurationen
- Zuerst schauen wir den Inhalt der YAML Datei:
- ```
! cat 09-5-Volume/web.yaml
```
- lernkube: <https://github.com/mc-b/lernkube/tree/master/data>
  - AWS: <https://github.com/mc-b/lernkube/tree/master/aws>
  - Azure: <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 ...

# 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>



# 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"
```

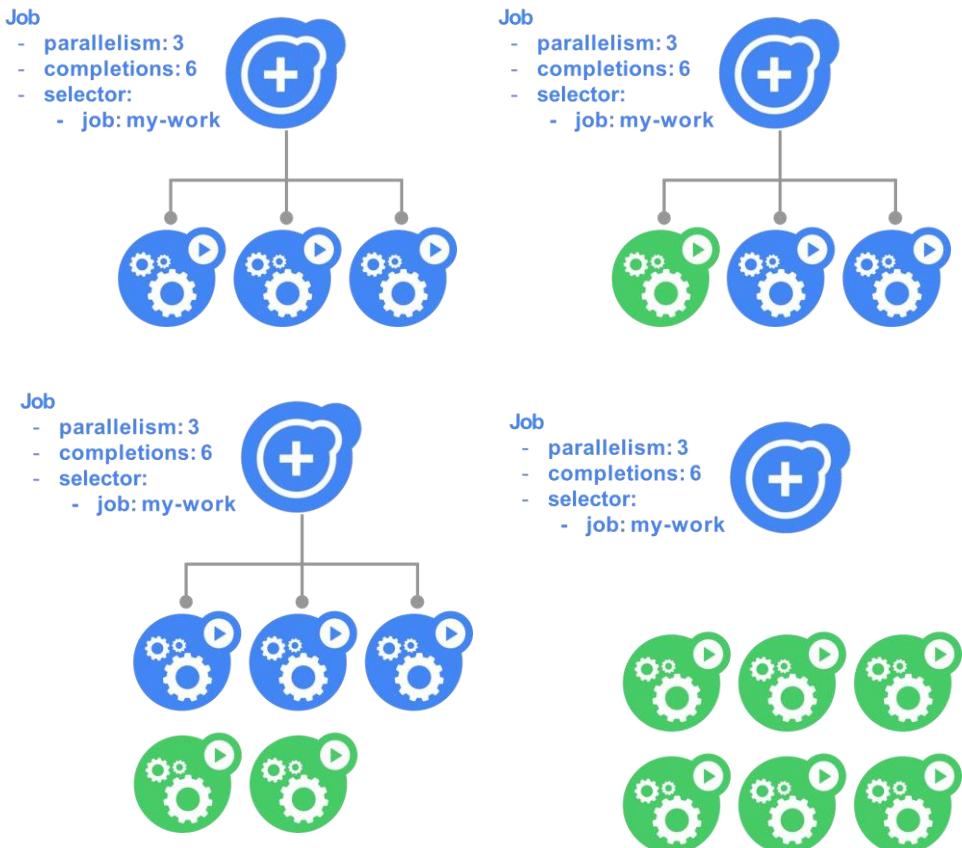
Anschliessend folgen die `Claims`:

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

## Weitere Beispiele

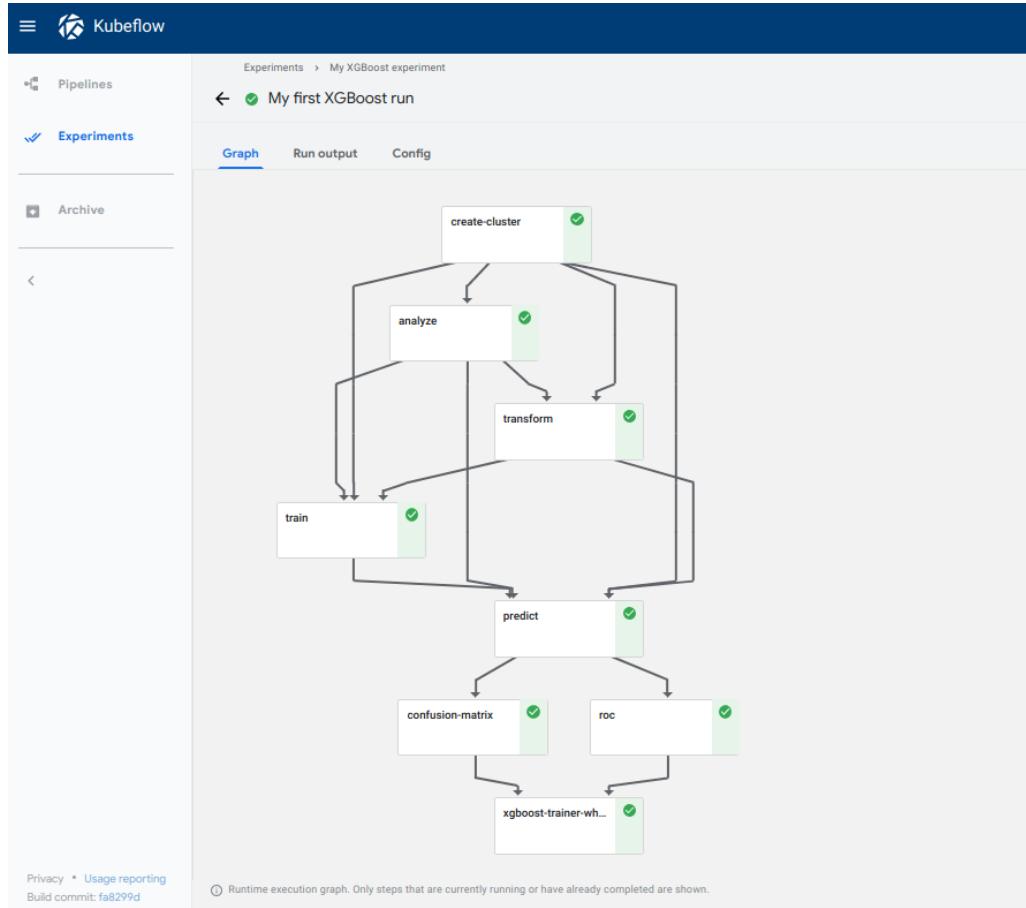
- [Deploying WordPress and MySQL with Persistent Volumes](#)

# Jobs



- ★ 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.

# Jobs: am Beispiel Kubeflow



- ★ Das Kubeflow-Projekt widmet sich der Bereitstellung von Arbeitsabläufen (Pipelines) für maschinelles Lernen (ML) auf Kubernetes, die einfach, portabel und skalierbar sind.
- ★ Überall, wo Sie Kubernetes ausführen, sollten Sie in der Lage sein, Kubeflow auszuführen.
  
- ★ Mehr Informationen:
  - <https://www.kubeflow.org/>
  - <https://github.com/volcano-sh/volcano>
  - <https://github.com/kubernetes-sigs/kube-batch>



# Übung: Jobs (optional)

★ Spielt das Beispiel laut <https://github.com/mc-b/duk/tree/master/mysql> durch.

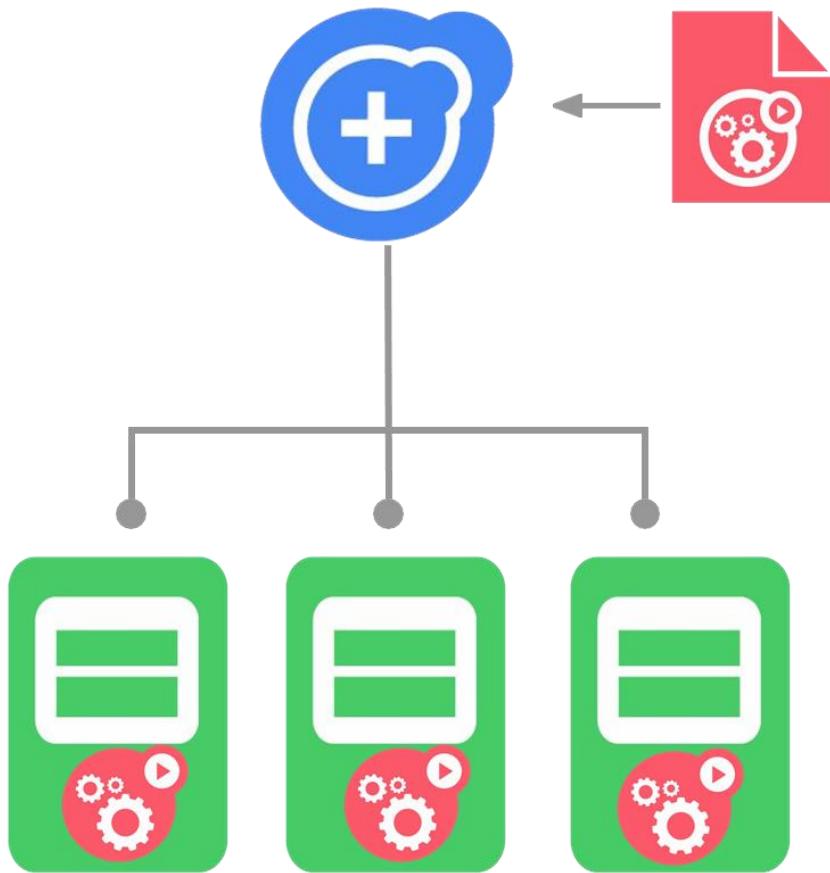
★ Anforderung:

- Ein Allgemein verwendbarer Container mit einer Datenbank, hier MySQL, soll jeweils mit anderen Daten befüllt werden.
- Dazu soll weder die YAML Datei des Containers noch der Container selber verändert werden müssen.

★ Lösung:

- Separater Job welcher sich mit dem Datenbank Container verbündet und diese mit Daten abfüllt.

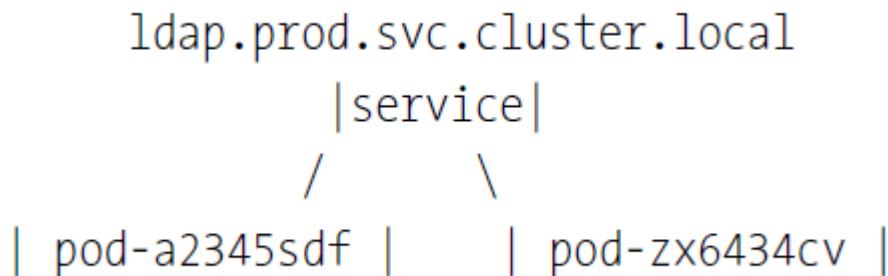
# DaemonSet



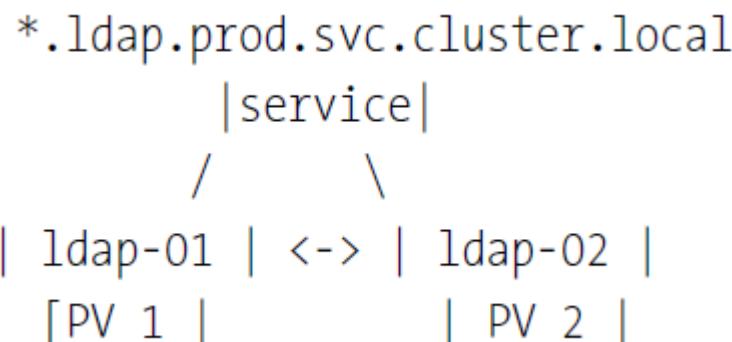
- ★ Runs a Pod on every node
  - or a selected subset of nodes
- ★ Not a fixed number of replicas
  - created and deleted as nodes come and go
- ★ Useful for running cluster-wide services
  - logging agents
  - storage systems
- ★ DaemonSet manager is both a controller and scheduler

# StatefulSets

- ★ Standard Zuordnung Service, Pods mit Hash



- ★ StatefulSets, mit fester Namensvergabe und fest zugeordneten PersistentVolumes:

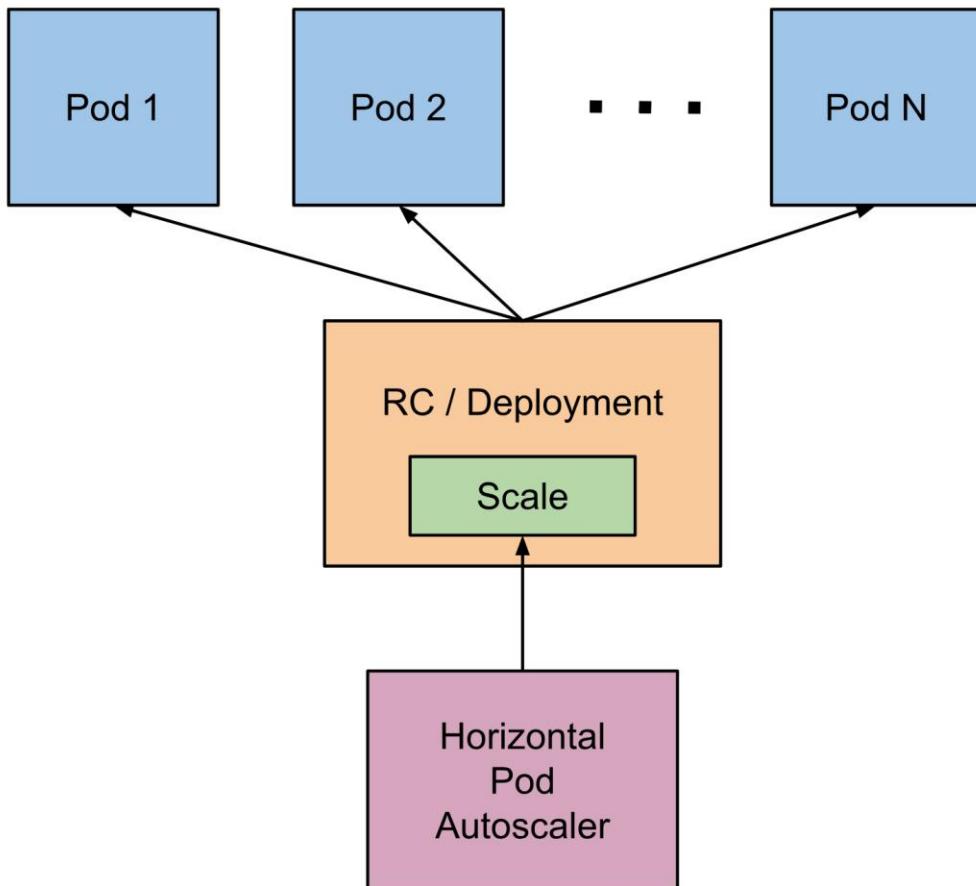


- ★ **StatefulSets:** klassische «Long Running» Services mit persistenter, hochverfügbarer Datenablage, statischen IPs (hinter einer Service-IP) und fester Namensvergabe.

- ★ 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.
- ...

# Horizontal Pod Autoscaler



- ★ Bei der horizontalen Pod-Autoskalierung skaliert Kubernetes entsprechend den vorgegebenen **Regelwerken** **vollautomatisch** die **Anzahl der Pods** in einem Replication Controller, Deployment oder ReplicaSet.
- ★ Dies kann beispielsweise auf Grundlage der überwachten CPU-Auslastung der zugehörigen Pods erfolgen.
  
- ★ Artikel:
  - <https://itnext.io/kubernetes-from-the-beginning-part-iv-autoscaling-7da4160181fc>
  - <https://sysdig.com/blog/kubernetes-autoscaler/>

# Health Probe Pattern in Kubernetes

- ★ Kubernetes überwacht den Status der Pods standardmäßig mithilfe eines seiner Controller ([Deployments](#), [ReplicaSets](#), [DaemonSets](#), [StatefulSets](#)) usw..
- ★ Ein Pod kann melden, dass er aktiv ist. Trotzdem funktioniert er nicht.
- ★ Lösung:
  - Liveness Probes
    - ★ Aktive Überprüfung, z.B. mit curl ob der Prozess im Container antwortet.
    - ★ [Kubernetes livenessProbe can be dangerous](#) - Ein einzelner DB-Fehler führt zu einem Neustart aller Container!
  - Readiness Probes
    - ★ Readiness-Tests führen dieselben Überprüfungen durch wie die Liveness-Tests (GET-Anforderungen, TCP-Verbindungen und Befehlsausführungen). Die Korrekturmassnahmen unterscheiden sich jedoch dadurch, dass eine Verzögerung mitgegeben werden kann. So hat der Container die Möglichkeit sauber zu starten.
- ★ siehe: <https://www.magalix.com/blog/kubernetes-and-containers-best-practices-health-probes>



# 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.
- ★ Rolling Updates und deren Variationen (Canary, Blue/Green) helfen die Services auf dem neusten Stand zu halten.
- ★ Detailinformationen zu jeder Ressource liefert
  - `kubectl explain <Ressource>`



# Lernzielkontrolle

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