CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Container-Orchestrierung

(6) Kubernetes (Teil IV: Services, Persistenz und Isolation)

PROF. DR.
NANE KRATZKE

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 9

Container-Plattformen

9.1 Scheduling

- Heterogenität von Workloads
- Scheduling-Algorithmen
- Scheduling-Architekturen

9.2 Orchestrierung

- Definition von Betriebszuständen
- Regelkreis: Desired vs Current State

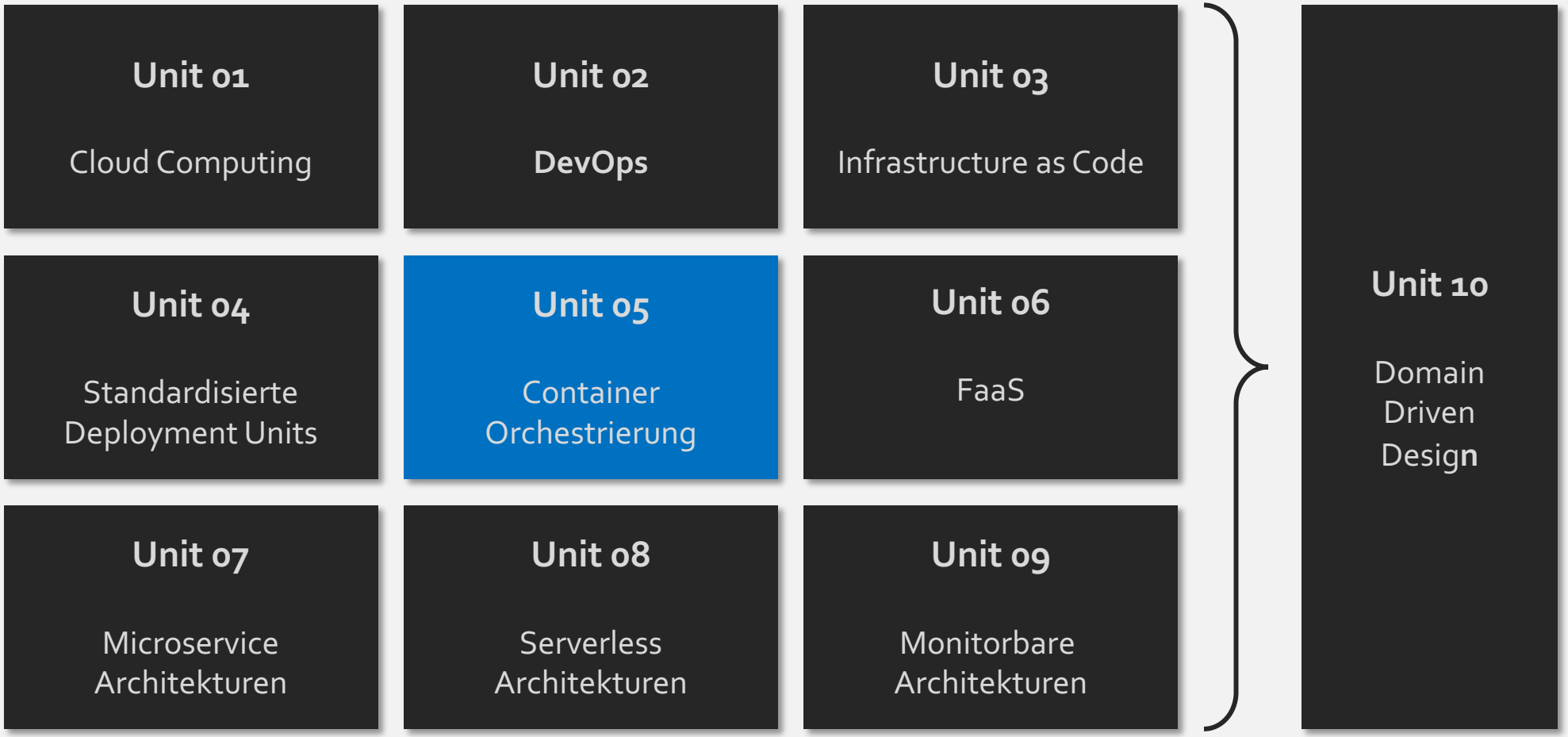
9.3 Inside Kubernetes

- Kubernetes-Architektur und Ressourcen
- Workloadarten
- Scheduling Constraints
- Automatische Skalierung von Workloads
- Exponierung von Services
- Health Checking
- Persistenz
- Isolation von Workloads

9.4 Zusammenfassung

INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



KUBERNETES

Überblick über die wichtigsten Kubernetes-Ressourcen

CLOUD NATIVE COMPUTING

Ausführbare Jobs/Workloads	Exponieren von Anwendungen und Diensten	Persistenz	Isolieren von Anwendungen mittels Namespaces
 Deployments	Primär Intra-Cluster  **Services**	 Persistent Volume Claim	  Quotas und Limit Ranges
 (Cron-)Jobs			
 Daemon Sets	Primär Extra-Cluster  **Ingress**	 Persistent Volume	 RBAC  Network Policies
 Stateful Sets			

SERVICES

Exponieren von Anwendungen und Diensten

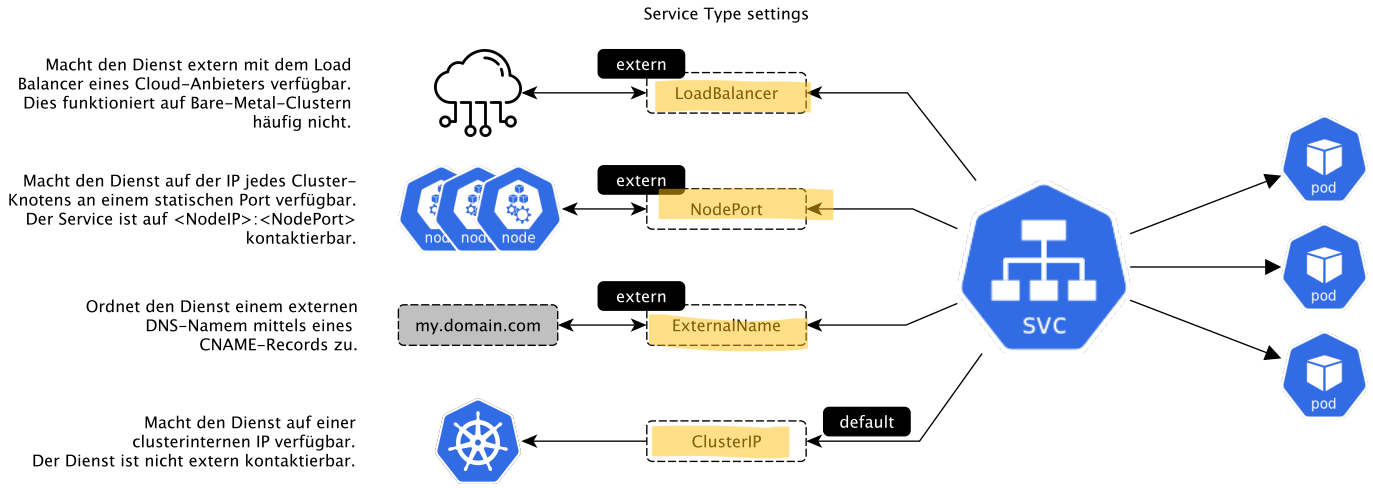
Definieren von Diensten mittels Pod-Labels und Selektoren

Dienste ermöglichen es eine Anwendung, die auf einer Reihe von Pods ausgeführt wird, als Netzwerkdienst unter einem **DNS-Namen** verfügbar zu machen.

Hierzu werden die Pods eines Services mittels ***Selektoren*** bestimmt.

Kubernetes gibt Pods ihre eigenen **IP-Adressen** und Services einen eindeutigen **DNS-Namen**.

Pods, die über einen Service angesprochen werden unterliegen automatisch einem **Load Balancing** durch kube-proxy.



Exponieren von Diensten mittels Service Types

Dienste werden per default nur Cluster-intern bereitgestellt. Sie können aber mittels des ***serviceType*** Feldes Cluster-extern exponiert werden:

- **`Load Balancer`** bindet einen Dienst an eine Public IP eines Cloud Providers
- **`NodePort`** macht einen Service-Port auf allen IP-Adressen der Cluster Nodes bekannt.
- Mittels **`ExternalName`** können externe Dienste in den Cluster eingeblendet und wie Cluster-interne Dienste genutzt werden.

Die Exponierung mittels Loadbalancern funktioniert meist nur in Public IaaS-provisionierten Clustern.

Quelle: https://kubernetes.io/docs/concepts/services-networking/service/

SERVICES

Exponieren von Anwendungen und Diensten

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 8080
        - containerPort: 8443
```

Manifest: nginx-deployment.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  selector:
    app: nginx
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 8080
  - name: https
    protocol: TCP
    port: 443
    targetPort: 8443
```

Manifest: nginx-service.yaml

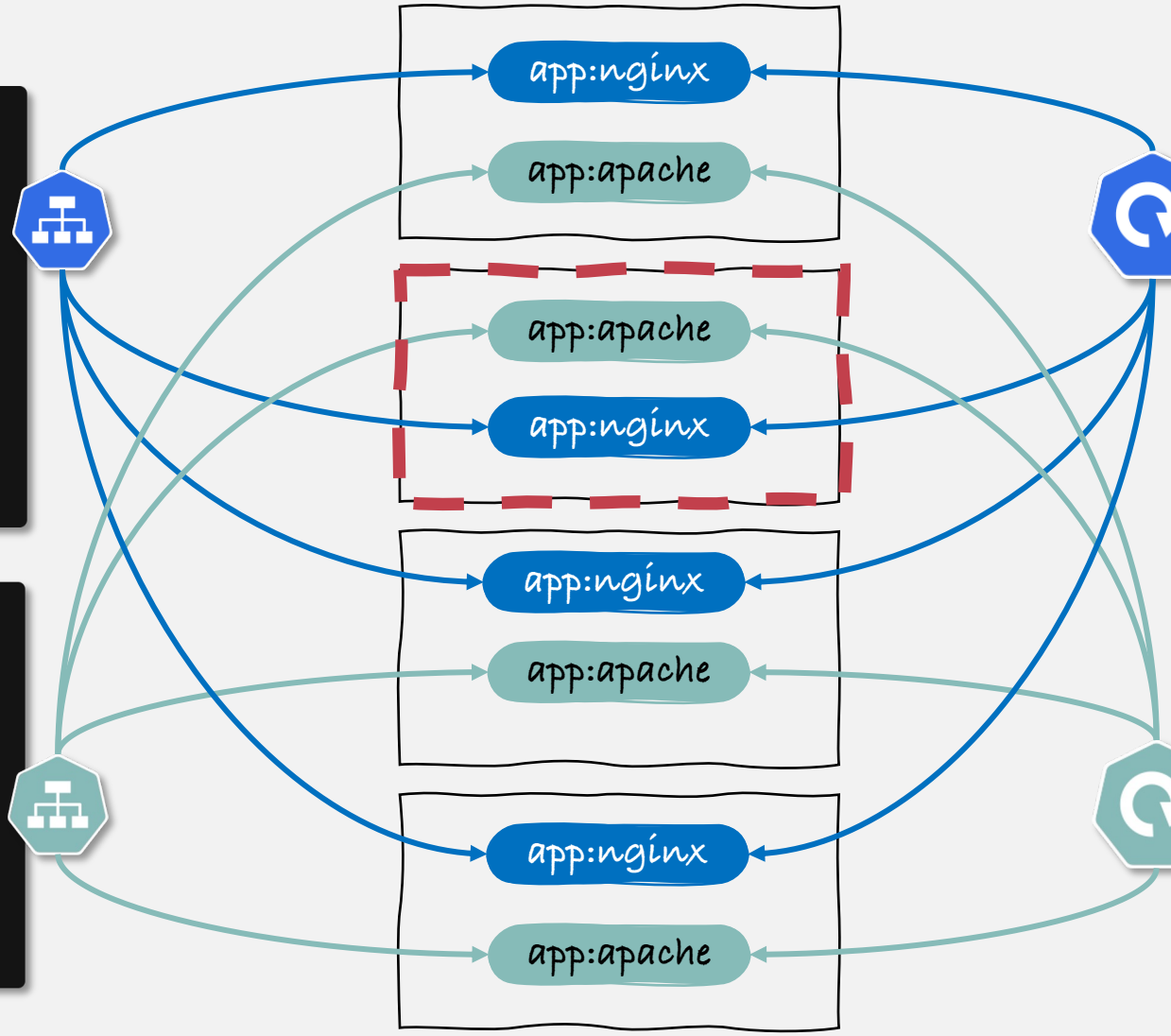
```
# Ausbringen des Deployments
kubectl apply -f nginx-deployment.yaml

# Ausbringen eines Services
kubectl apply -f nginx-service.yaml

# Auflisten aller Services
kubectl get svc

# Details zu einem Service
kubectl describe http-service

# Löschen eines Services
kubectl delete -f nginx-service.yaml
```

SERVICES

Wirkungsweise

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

```yaml
apiVersion: v1
kind: Service
metadata:
  name: apache-nginx
spec:
  selector:
    app: apache
  ports:
  - name: http
    port: 80
    targetPort: 80
```



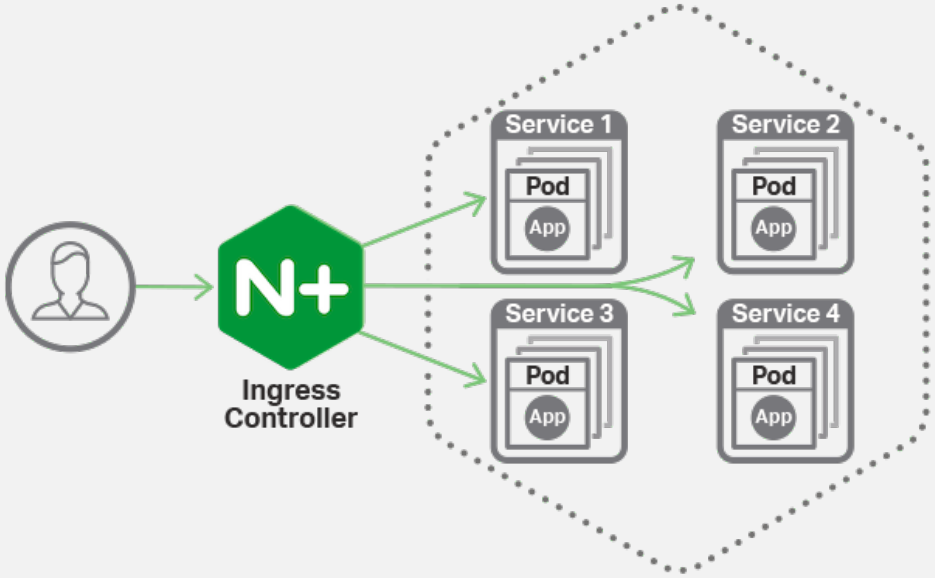

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-dep
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:stable
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-dep
  labels:
    app: apache
spec:
  replicas: 3
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: apache:stable
```

INGRESS

Exponieren von Service APIs via HTTP(S)

- Ein Ingress macht HTTP- und HTTPS-Routen von außerhalb des Clusters für Dienste innerhalb des Clusters verfügbar.
- Ein Ingress kann so konfiguriert werden, dass für Dienste extern erreichbare URLs bereitgestellt (inkl. Load Balancing, SSL/TLS Termination und namenbasiertes virtuelles Hosting) angeboten werden.
- Das Routing wird hierzu durch Regeln gesteuert, die in der Ingress-Ressource definiert sind.
- Wenn Dienste mittels anderer Protokolle als HTTP(S) exponiert werden sollen, muss normalerweise ein Service vom Typ NodePort oder LoadBalancer verwendet werden.



Quelle: https://kubernetes.io/docs/concepts/services-networking/ingress/

CLOUD NATIVE COMPUTING

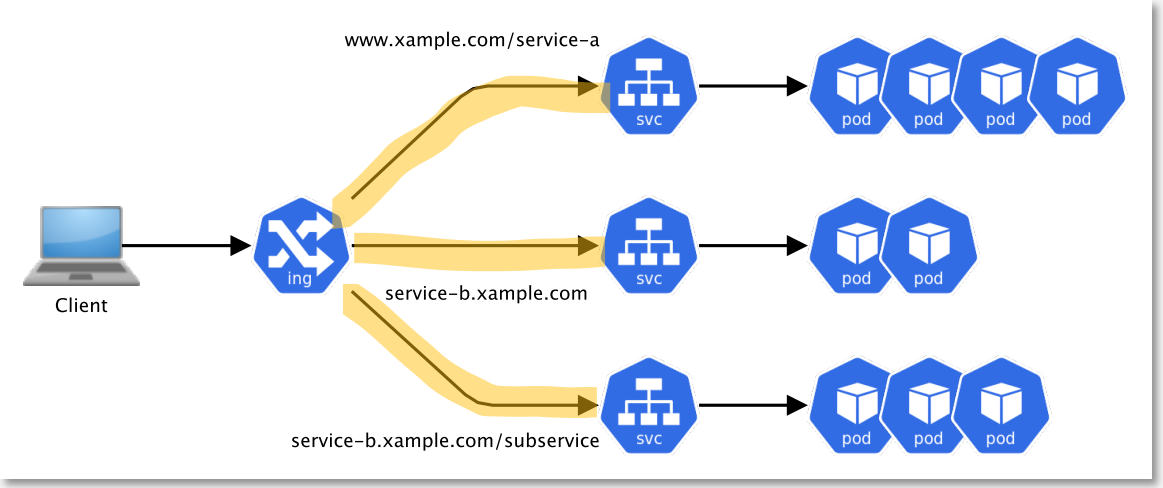
Wieso nur HTTP(S) und nicht beliebige Protokolle?

REST-basierte APIs sind einfach eine verbreitete Art und Weise Schnittstellen zu Cloud-nativen Dienste bereitzustellen.

*Beliebige andere TCP/UDP basierte Protokolle können mittels **Load Balancern** über Services Cluster-extern bekannt gemacht werden.*

INGRESS

Exponieren von Anwendungen und Diensten



Insbesondere HTTP-basierte Dienste (z.B. REST-basierte APIs) werden meist mittels Ingress Ressourcen für Zugriffe von außerhalb des Clusters bereitgestellt.

Anders als Service-basierte NodePort oder LoadBalancer Exponierungen (die pro Exponierung jeweils eine eindeutige IP-Adresse+Portnummer Kombination benötigen) braucht ein Ingress nur eine einzige IP-Adresse und eine Portnummer, um eine Vielzahl an Diensten zu exponieren. Insbesondere Public-IP4 Adressen sind mittlerweile ein knappes Gut geworden.

Ingresse laufen ferner auf Anwendungsschicht des Netzwerkstapels (HTTP) und bieten daher Funktionen wie bspw. Cookie-gestützte Sitzungsaffinitäten und können ferner mittels TLS-Zertifikaten abgesichert werden (HTTPS).

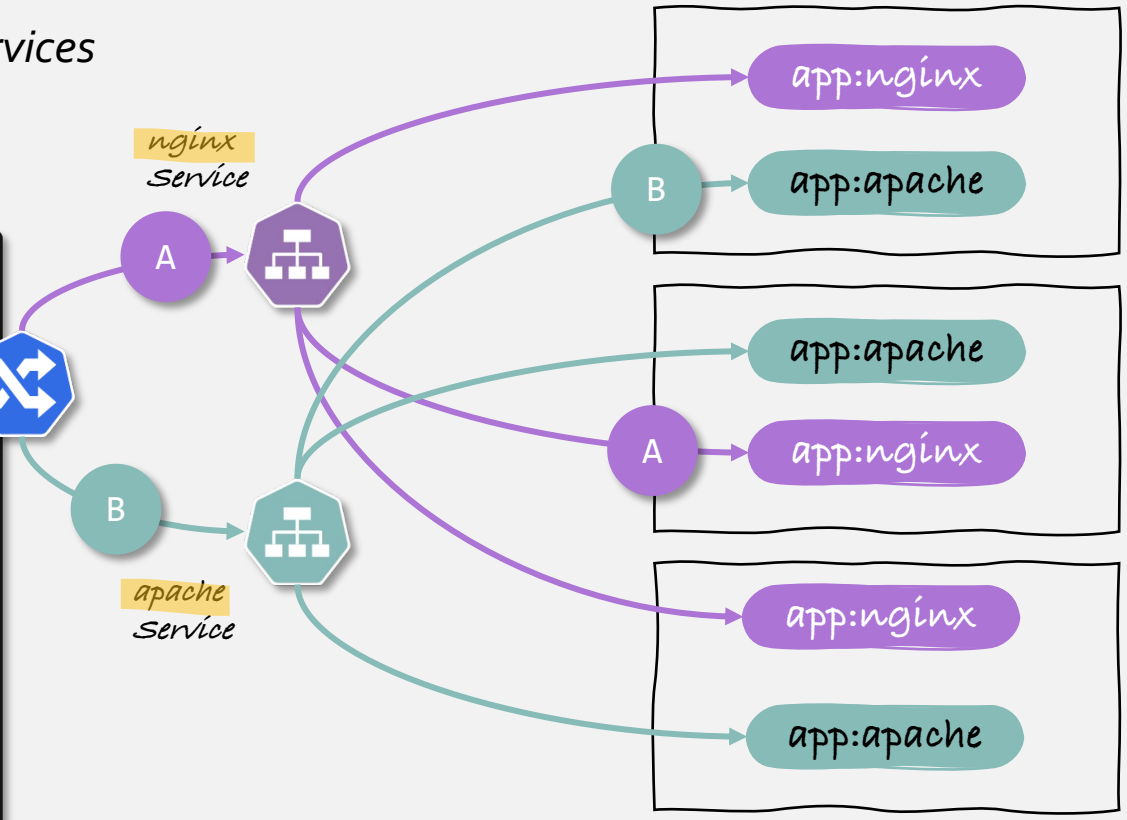
```yaml
kind: Ingress
apiVersion: networking.k8s.io/v1
metadata:
  name: xample-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: www.xample.com
    http:
      paths:
      - path: /service-A
        pathType: Prefix
        backend:
          service:
            name: service-a
            port: { number: 8080 }
  - host: service-b.xample.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service-b
            port: { number: 80 }
      - path: /subservice
        pathType: Prefix
        backend:
          service:
            name: service-b-subservice
            port: { number: 8888 }
```

Manifest: xample-ingress.yaml

INGRESS

Wirkungsweise von einem Ingress vor Services

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: eventdriven.xample.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: nginx
                port:
                  number: 80
    - host: process.xample.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: apache
                port:
                  number: 80
```




```
> curl http://eventdriven.xample.com
```

A

```
> curl http://process.xample.com
```

B

CLOUD NATIVE COMPUTING

Nginx Deployment

Apache Deployment

KUBERNETES

Überblick über die wichtigsten Kubernetes-Ressourcen

CLOUD NATIVE COMPUTING

Ausführbare Jobs/Workloads	Exponieren von Anwendungen und Diensten	Persistenz	Isolieren von Anwendungen mittels Namespaces
Deployments	Primär Intra-Cluster: Services	Persistent Volume Claim	Quotas und Limit Ranges
(Cron-)Jobs			
Daemon Sets	Primär Extra-Cluster: Ingress	Persistent Volume	RBAC
Stateful Sets			Network Policies

PERSISTENZ

Organisation persistenten Speichers in Kubernetes

Das Verwalten von persistentem Speicher ist ein anderes Problem als das Verwalten von flüchtigen Recheninstanzen.

Das Persistenz-Subsystem von Kubernetes stellt hierzu eine API für Benutzer und Administratoren bereit, die Details zur Bereitstellung von Speicher von der Art seiner Verwendung abstrahiert.

Dies erfolgt mittels zweier API-Ressourcen namens **`PersistentVolume`** und **`PersistentVolumeClaim`**.

Persistent Volume (PV)

- Ein PV repräsentiert persistenten Speicher (Festplatte) im Cluster, der von einem Administrator bereitgestellt oder mithilfe von Speicherklassen dynamisch erzeugt wurde.
- Es ist wie ein Knoten eine normale statische Ressource des Clusters. PVs haben einen Lebenszyklus, der von einem Pod unabhängig ist.
- Dieses API-Objekt erfasst die Details der Implementierung des Speichers, sei es NFS, iSCSI oder ein Cloud-Provider-spezifisches Speichersystem.

Persistent Volume Claim (PVC)

- Ein PVC ist eine Anforderung von persistentem Speicher (Festplatte) durch einen Benutzer.
- Es ist analog einem Pod, der CPU- und Memory-Ressourcen eines Knotens benötigt. PVCs verbrauchen PV-Ressourcen.
- Mittels PVCs können bestimmte QoS-, Größen- und Zugriffsmodi von persistentem Speicher angefordert werden (z. B. können sie ReadWriteOnce, ReadOnlyMany oder ReadWriteMany bereitgestellt werden).

Für K8S gibt es u.a. die folgenden Block-Storage und Filesystem Provisioner:

- *AWS Elastic Block Store*
- *Azure File*
- *Azure Disk*
- *Ceph (FS + RBD)*
- *Cinder (OpenStack)*
- *FC (Fibre Channel)*
- *Flex Volume*
- *Flocker*
- *GCE Persistent Disk*
- *GlusterFS*
- *iSCSI*
- *Quobyte*
- *NFS*
- *Vsphere Volume*
- *Portworx Volume*
- *Scale IO*
- *Storage OS*
- *Local*
- *...*

Quelle: https://kubernetes.io/docs/concepts/storage/persistent-volumes/

PERSISTENT VOLUME

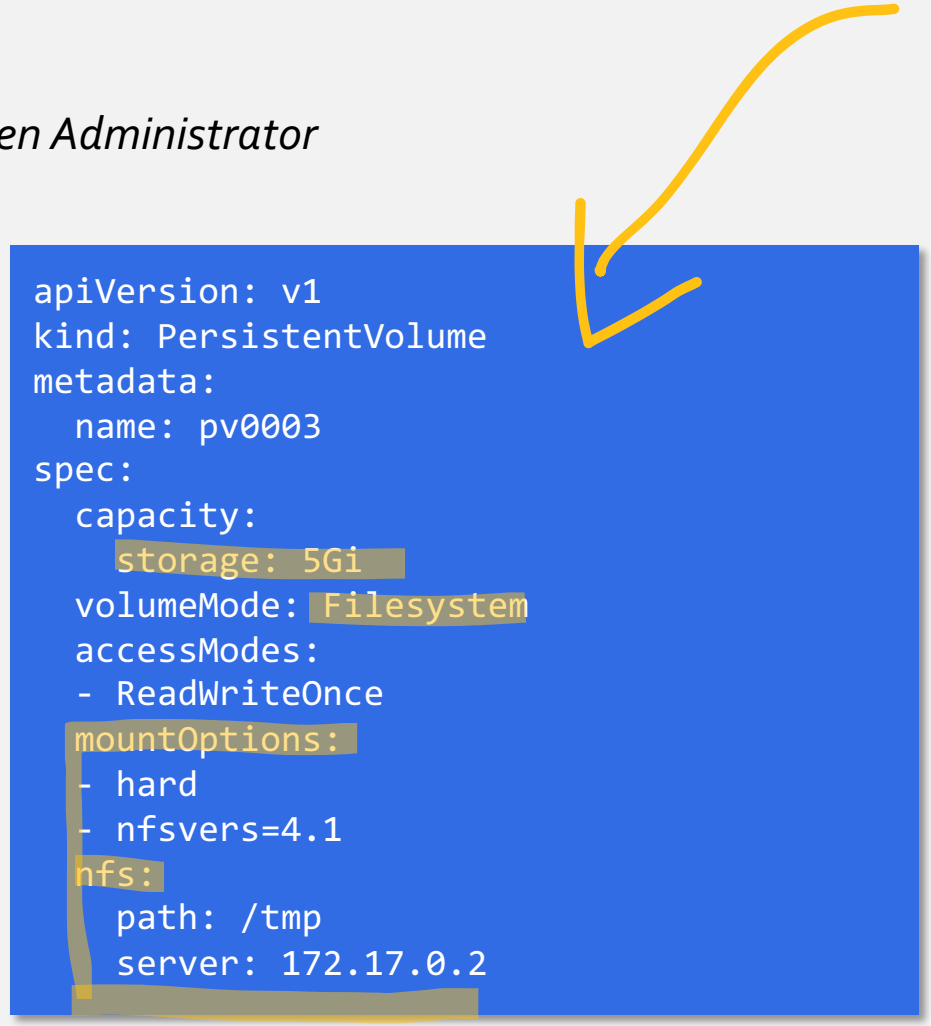
Bekanntmachung von persistentem Speicher durch den Administrator

Beispiel eines mittels NFS bereitgestellten Persistent Volumes.

Werden PVs auf diese Weise statisch dem Cluster bekannt gemacht, erfolgt dies dann normalerweise durch den Administrator. Dieser muss auch Details der Infrastruktur kennen, wie bspw. IP-Adresse des NFS-Servers, Kapazität der Platte, usw.

PVs werden daher meist nicht durch User im System angelegt.

Für User werden solche PVs meist mittels PVCs durch Storage Provisioner (Storage Classes) bereitgestellt.



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  mountOptions:
  - hard
  - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

PERSISTENT VOLUME CLAIM

Anforderung und Mounten von persistentem Speicher durch den User

Anforderung von persistentem Speicher

User können persistenten Speicher mittels PVCs anfordern. Hierzu muss eine Storage-Klasse angegeben werden, aus dem der Speicher bedient werden soll (auf die Angabe einer Storage Klasse kann verzichtet werden, wenn es eine Default Storage Klasse im Cluster gibt).

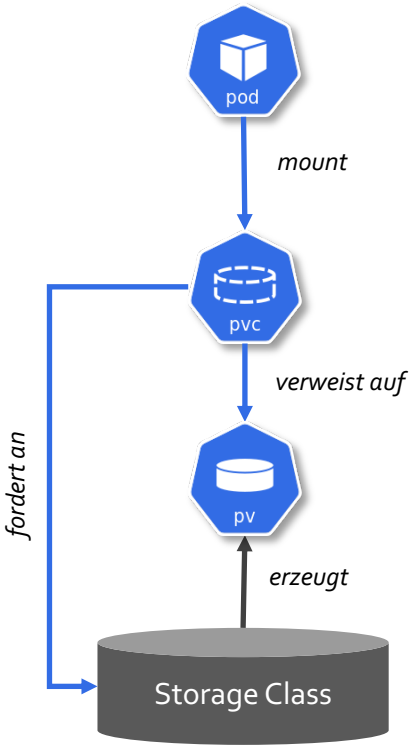
```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:["ReadWriteOnce"]
  volumeMode: Filesystem
  resources:
    requests: { storage: 8Gi }
  storageClassName: slow
```

Claims as Volumes

Pods greifen auf den so angeforderten persistenten Speicher zu, indem sie den PVC als Volume im Pod mounten. PVCs müssen sich hierzu im selben Namespace wie der Pod befinden.

Der Cluster nutzt das PVC im Namespace des Pods und verwendet es, um ein Persistent Volume zu erzeugen, welches den Anforderungen des PVC genügt. Das Persistent Volume wird dann auf dem Host und im Pod bereitgestellt.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: myfrontend
    image: nginx
    volumeMounts:
    - mountPath: "/var/www/html"
      name: mypd
  volumes:
  - name: mypd
    persistentVolumeClaim: { claimName: myclaim }
```



KUBERNETES

Überblick über die wichtigsten Kubernetes-Ressourcen

CLOUD NATIVE COMPUTING

Ausführbare Jobs/Workloads	Exponieren von Anwendungen und Diensten	Persistenz	Isolieren von Anwendungen mittels Namespaces
 Deployments	Primär Intra-Cluster  Services	 Persistent Volume Claim	  **Quotas und Limit Ranges**
 (Cron-)Jobs			
 Daemon Sets	Primär Extra-Cluster  Ingress	 Persistent Volume	 **RBAC**  **Network Policies**
 Stateful Sets			

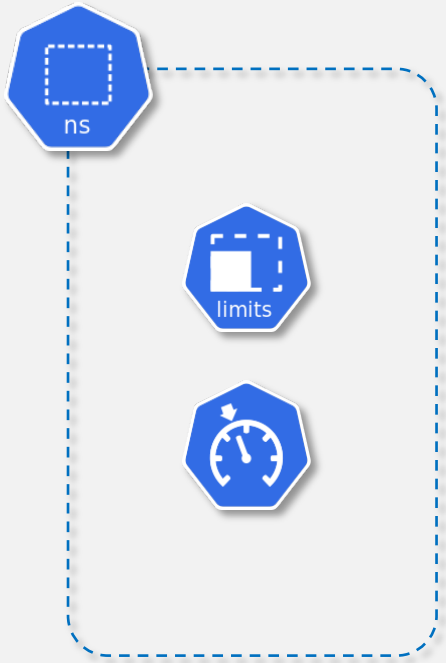
QUOTAS UND LIMIT RANGES

Isolieren von Anwendungen mittels Namespaces

Mit Ressourcen-Kontingenten **(ResourceQuota)** kann der Ressourcenverbrauch pro Namespace beschränkt werden.

Um zu vermeiden, dass einzelne Pods oder Container alle verfügbaren Ressourcen monopolisieren, kann zusätzlich die Ressourcenzuweisung pro Container limitiert bzw. mit default Werten zugewiesen werden **(LimitRange)**.

Hierzu müssen einfach nur die entsprechenden Manifeste in einem Namespace angelegt werden.



Manifest: compute-quota.yaml

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.nvidia.com/gpu: 4
```

```
kubectl apply -f compute-quota.yaml -n xample
```

Manifest: cpu-limits.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-limits
spec:
  limits:
  - type: Container
    max: { cpu: "1000m" }
    min: { cpu: "200m" }
    default: { cpu: "200m" }
```

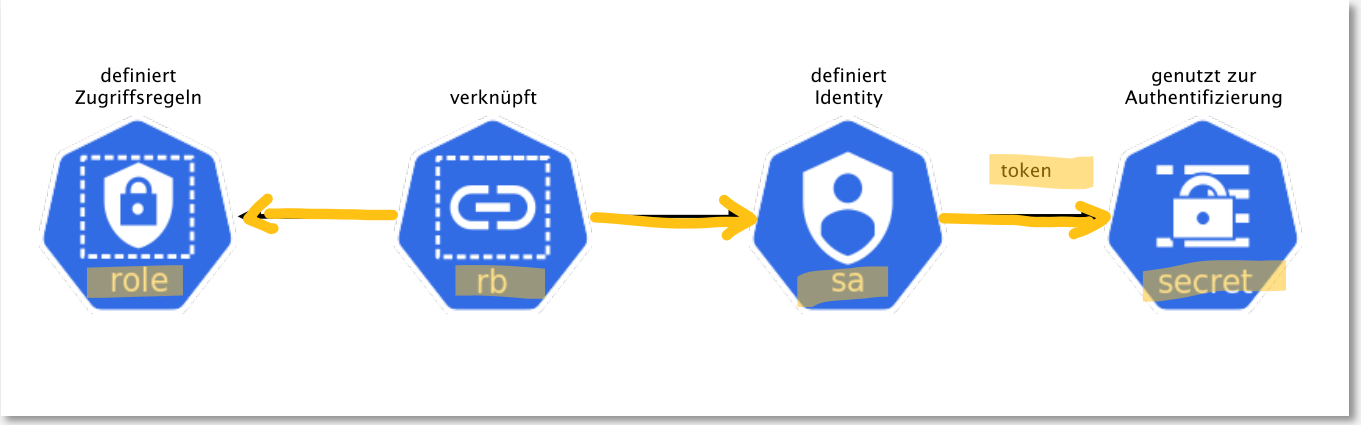
```
kubectl apply -f cpu-limits.yaml -n xample
```

Quotas: https://kubernetes.io/docs/concepts/policy/resource-quotas/

LimitRanges: https://kubernetes.io/docs/concepts/policy/limit-range/

ROLE BASED ACCESS MODEL

Isolieren von Anwendungen mittels Namespaces



In Kubernetes können Namespaces zur Isolation von Workloads angelegt werden.

Mittels Serviceaccounts und Zugriffsregeln können innerhalb eines Namespaces komplexe Rechtesysteme realisiert werden.

Jeder per `kubectl` angelegte Namespace erhält automatisch einen default Serviceaccount für den ein Access Token generiert werden kann. Weitere Serviceaccounts und Tokens können angelegt werden.

Roles weisen Rechte in einem Namespace zu (für den diese Rolle erzeugt wurde).

ClusterRoles weisen Rechte Namespace-übergreifend auf Cluster-Ebene zu.

Das Anlegen von ClusterRoles erfolgt analog zu Roles.

Quelle: https://kubernetes.io/docs/reference/access-authn-authz/rbac/

ROLE BASED ACCESS MODEL

Isolieren von Anwendungen mittels Namespaces

Dieses Beispiel zeigt, wie sich eine Rolle in Kubernetes einem Serviceaccount zuweisen lässt.

```
# Das Erzeugen eines Namespaces (mit kubcectl) erzeugt auch
# immer automatisch einen default Serviceaccount (sa)
kubectl create namespace xample
kubectl get sa -n xample

NAME      SECRETS   AGE
default   1         6m2s

# Es können beliebig weitere Serviceaccounts in einem
# Namespace angelegt werden, z.B. ein Admin Account
kubectl create sa admin -n xample

# Anlegen von Rollen
kubectl apply -f dep-pod-pvc-creator.yaml -n xample

# Zuordnen von Rollen zu Accounts
kubectl create rolebinding creator-role-binding \
  --role=deployment-pod-pvc-creator \
  --serviceaccount=xample:default -n=xample
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  # Der Name der Rolle
  name: deployment-pod-pvc-creator
rules:
- apiGroups: ["apps"]
  # Berechtigungen für Deployments
  resources: ["deployments"]
  verbs: ["create", "get", "list", "watch", "delete"]
- apiGroups: [""]
  # Berechtigungen für Pods
  resources: ["pods"]
  verbs: ["create", "get", "list", "watch", "delete"]
- apiGroups: [""]
  # Berechtigungen für Persistent Volume Claims
  resources: ["persistentvolumeclaims"]
  verbs: ["create", "get", "list", "watch", "delete"]
```

Diese Rolle ermöglicht das Anlegen, Lesen und Löschen von Deployments, Pods und PersistentVolumeClaims.

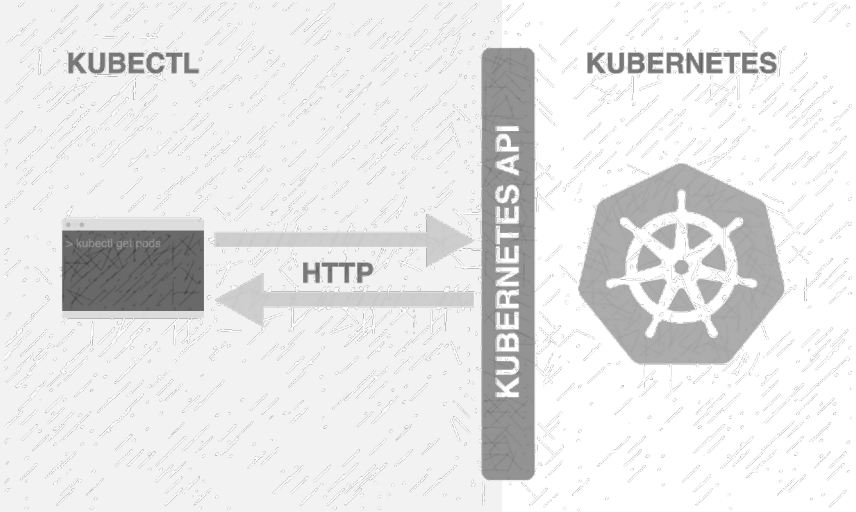
CLOUD NATIVE COMPUTING

ROLE BASED ACCESS MODEL

Einige oft genutzte API-Gruppen der Kubernetes REST-API

Einige Beispiele für API-Groups in Kubernetes sind:

- **core** oder **""** (die leere Zeichenkette repräsentiert die Core-Group für grundlegende Objekte wie Pods)
- **apps** für Anwendungen und ihre Verwaltung, einschließlich Deployments und StatefulSets
- **batch** für Batch-Jobs und CronJobs
- **storage.k8s.io** für Speicherressourcen wie StorageClasses und VolumeAttachments
- **rbac.authorization.k8s.io** für rollenbasierte Zugriffskontrollen
- **networking.k8s.io** für Netzwerkkonfigurationen wie Network Policies

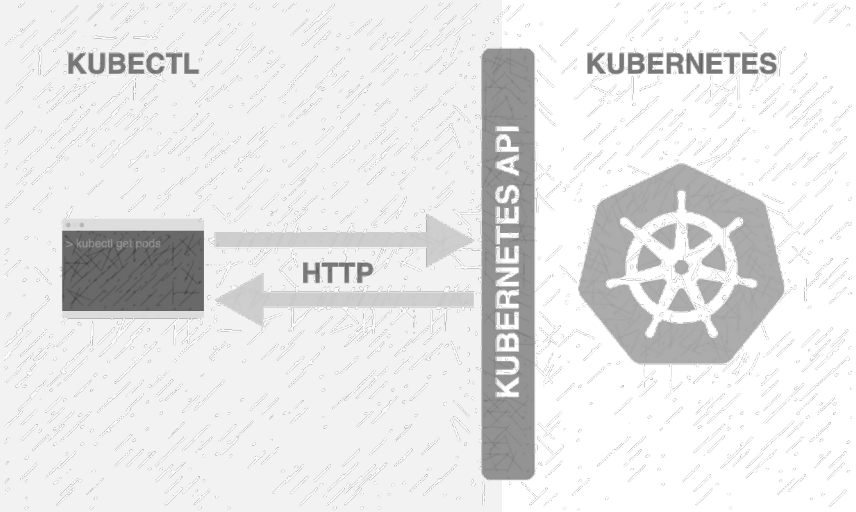


ROLE BASED ACCESS MODEL

Die Bedeutung der häufigsten Verben und ihre Bedeutung in der Kubernetes REST-API

Verben definieren die Operationen, die auf Ressourcen ausgeführt werden dürfen.

- **get:** Erlaubt das Abrufen von Informationen über eine spezifische Ressource.
- **list:** Erlaubt das Abrufen einer Liste aller Instanzen des Ressourcentyps.
- **watch:** Erlaubt das Abonnieren von Änderungen an allen Instanzen des Ressourcentyps.
- **create:** Erlaubt das Erstellen einer neuen Instanz des Ressourcentyps.
- **update:** Erlaubt das Aktualisieren einer bestehenden Instanz des Ressourcentyps.
- **patch:** Erlaubt das Aktualisieren einer bestehenden Instanz des Ressourcentyps, allerdings kann die Änderung geringfügiger sein oder sich nur auf bestimmte Teile der Ressource beziehen.
- **delete:** Erlaubt das Entfernen einer Instanz des Ressourcentyps.
- **deletecollection:** Erlaubt das Entfernen aller Instanzen des Ressourcentyps oder einer Auswahl von Instanzen basierend auf einem Suchkriterium.



NETWORK POLICIES

Isolation der Kommunikation in Multi-Tenancy Umgebungen

Standardmäßig sind Pods in einem Kubernetes-Cluster nicht isoliert; sie können daher untereinander kommunizieren. Das ist nicht immer gewollt – und insbesondere in Multi-Tenant-Umgebungen problematisch. Network Policies ermöglichen es, selektive Isolation durchzuführen, sodass nur bestimmte Verbindungen zugelassen werden.

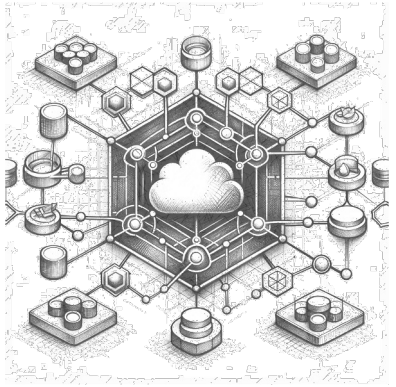
- **Pod-zu-Pod-Kommunikation**: Man kann bspw. definieren, welche Pods miteinander kommunizieren dürfen. Dies wird typischerweise auf Basis von Labels und Selektoren konfiguriert.
- **Namespace-basierte Segregation**: Man kann dies auch nutzen, um den Verkehr zwischen Namespaces zu steuern. Beispielsweise kann man Regelungen treffen, die die Kommunikation zwischen bestimmten Namespaces erlauben oder verbieten.
- **Egress- und Ingress-Regeln**: Man kann sowohl eingehenden (Ingress) als auch ausgehenden (Egress) Netzwerkverkehr konfigurieren. Ingress-Regeln steuern den eingehenden Verkehr zu Pods, während Egress-Regeln den ausgehenden Verkehr von Pods regeln.

NETWORK POLICIES

Isolation der Kommunikation in Multi-Tenancy Umgebungen

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: same-namespace-ingress-and-open-egress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector: {}
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: ingress
  egress:
  - {}
```

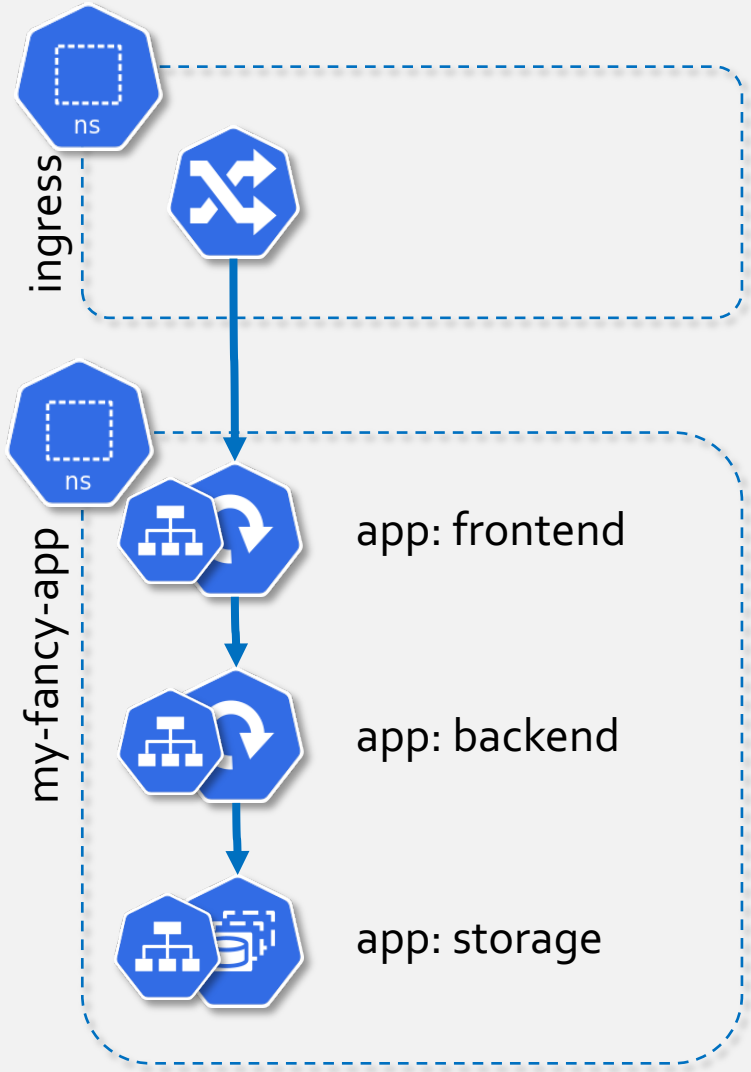
- Beispiel einer Network-Policy, die es Pods nur ermöglicht innerhalb des eigenen Namespaces zu kommunizieren oder aus dem Ingress Namespace aufgerufen zu werden.
- Die ausgehende Kommunikation der Pods (egress) wird nicht eingeschränkt.



CLOUD NATIVE COMPUTING

NETWORK POLICIES

Isolation der Kommunikation in Multi-Tenancy Umgebungen




```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-egress
spec:
  podSelector: {} # Selektiert alle Pods im Namespace
  policyTypes:
  - Egress
  egress:
  - {} # Erlaubt alle ausgehenden Verbindungen
```

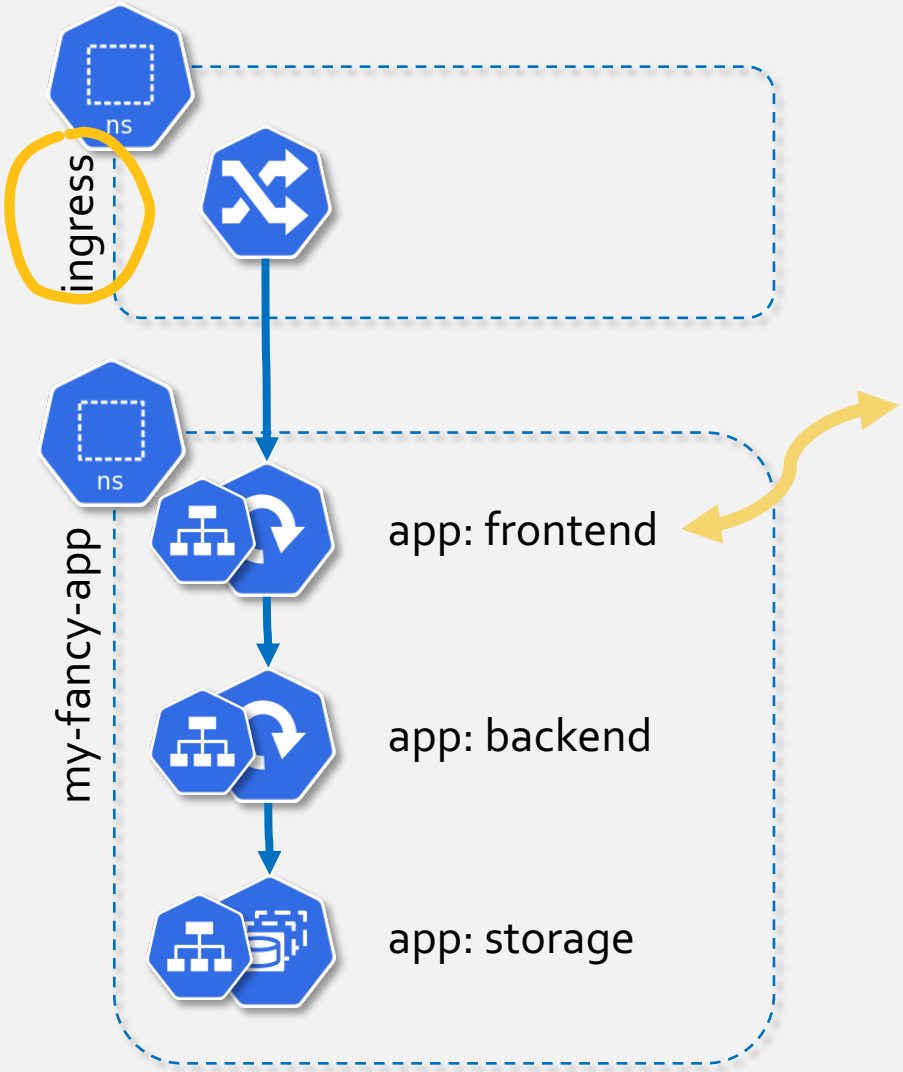
Die ausgehende Kommunikation der Pods soll nicht eingeschränkt werden.

```
> kubectl apply -f allow-egress.yaml
```

CLOUD NATIVE COMPUTING

NETWORK POLICIES

Isolation der Kommunikation in Multi-Tenancy Umgebungen




```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-ingress-policy
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: frontend
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name=ingress: ingress
```

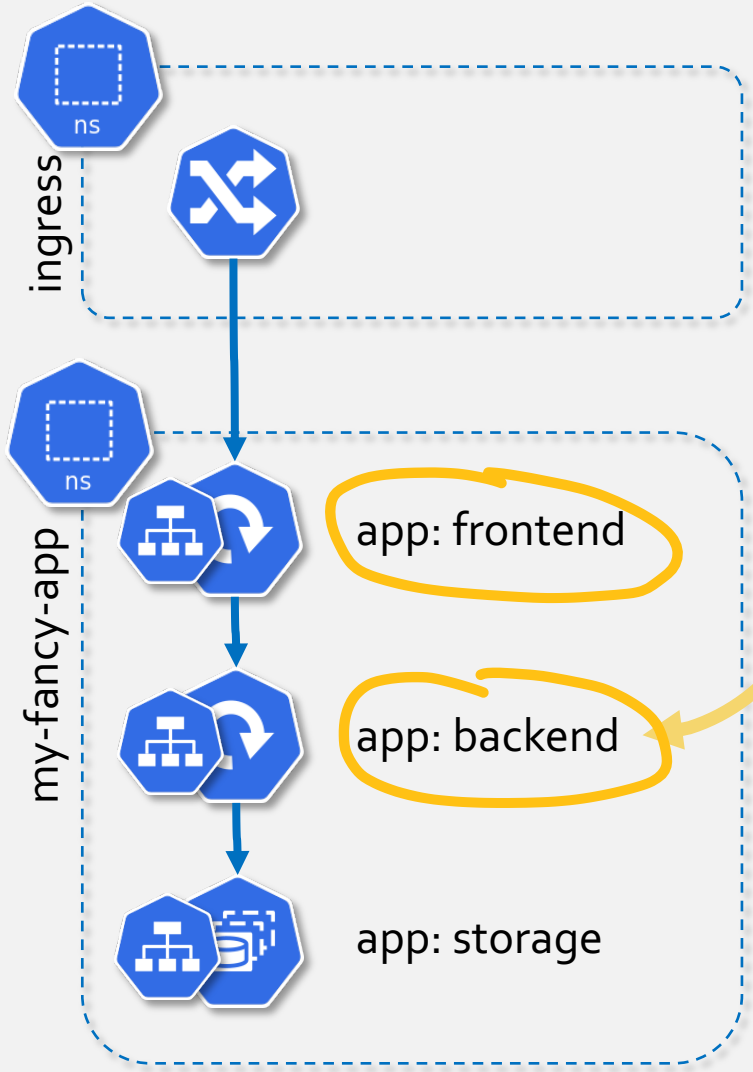
Das Frontend soll nur von Pods aus dem Ingress Namespace des Clusters erreichbar sein.

```
> kubectl apply -f allow-ingress.yaml
```

CLOUD NATIVE COMPUTING

NETWORK POLICIES

Isolation der Kommunikation in Multi-Tenancy Umgebungen




```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: backend-from-frontend
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
```

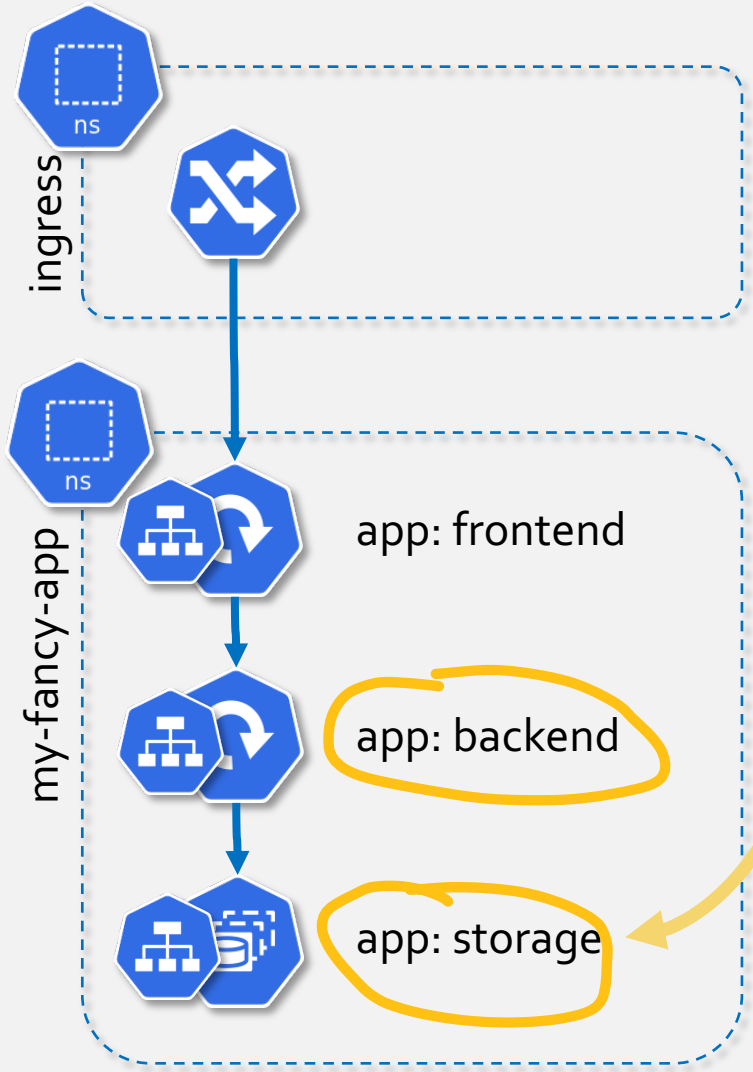
Das Backend soll nur von Pods des Frontends erreichbar sein.

```
> kubectl apply -f backend-from-frontend.yaml
```

CLOUD NATIVE COMPUTING

NETWORK POLICIES

Isolation der Kommunikation in Multi-Tenancy Umgebungen




```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: storage-from-backend
spec:
  podSelector:
    matchLabels:
      app: storage
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: backend
```

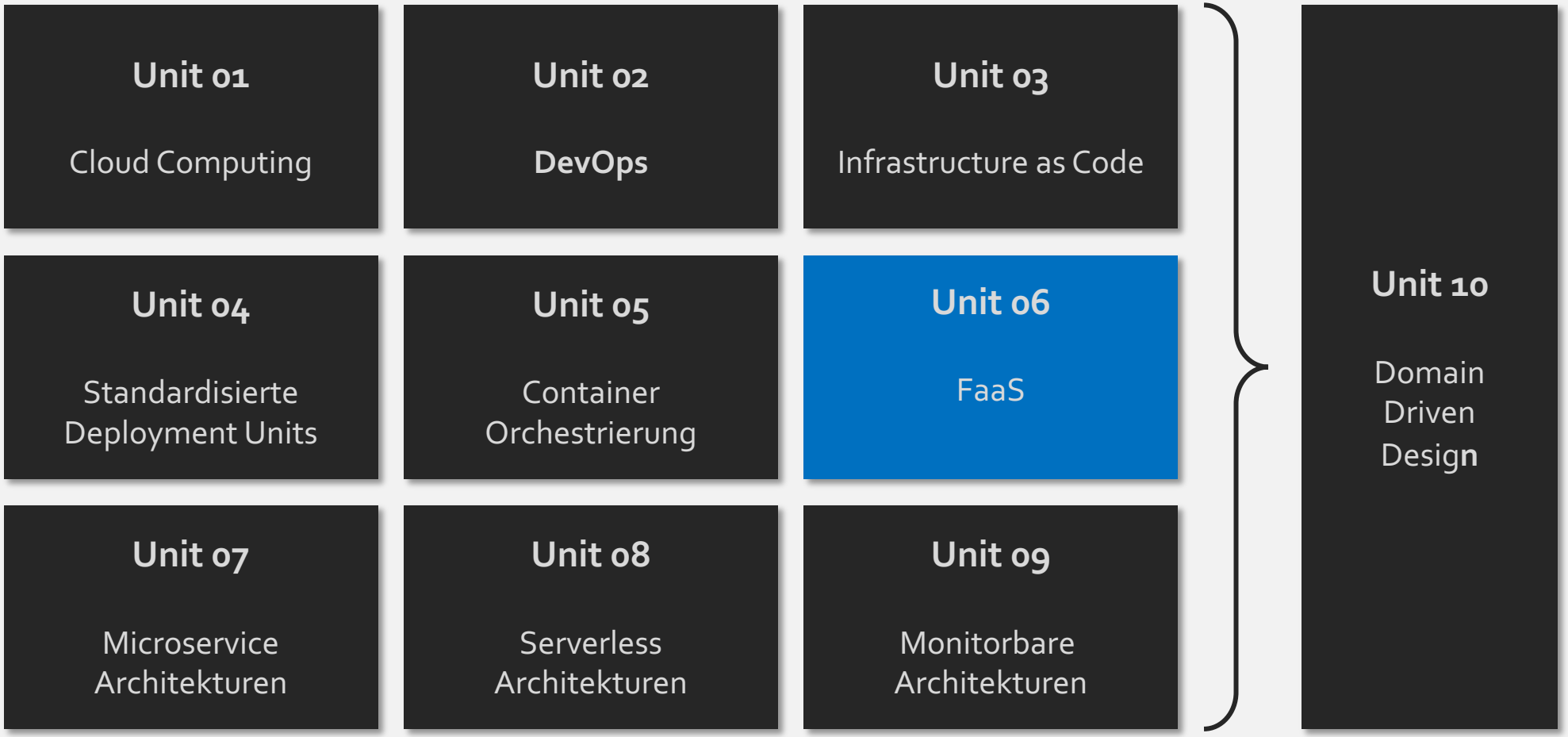
Das Backend soll nur von Pods des Frontends erreichbar sein.

```
> kubectl apply -f backend-from-frontend.yaml
```

CLOUD NATIVE COMPUTING

AUSBLICK

Überblick über Units und Themen dieses Moduls



KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de