bechtle

Willkommen zum **Kubernetes** ***Basic Kurs – Part 1***

Durchgeführt von Thomas Geiger.

Mägenwil, Thomas Geiger

31.01.2025

Agenda

1. **Theorie Teil: Was ist Kubernetes?**
2. **Theorie Teil: Wichtige Kubernetes-Komponenten**
3. **Theorie Teil: Warum verwenden wir Kubernetes**
4. **Praktischer Teil 1: Minikube**
5. **Praktischer Teil 3: Erstellen eines Pods**
6. **Praktischer Teil 2: Arbeiten mit `kubectl`**
7. **Praktischer Teil 4: Erstellen eines Deployments**
8. **Praktischer Teil 5: Erstellen von ConfigMaps und Secrets**
9. **Praktischer Teil 6: Final Project: Deploying an Application**

Theorie Teil

"Kubernetes ist eine Plattform, die Ihnen hilft, genau das zu tun, was Sie in der Cloud benötigen: Anwendungen skalieren, automatisieren und verwalten – unabhängig davon, wo sie laufen."

1

Was ist Kubernetes?

Open Source

Kubernetes ist eine Open-Source-Platform zur Container-Orchestrierung.

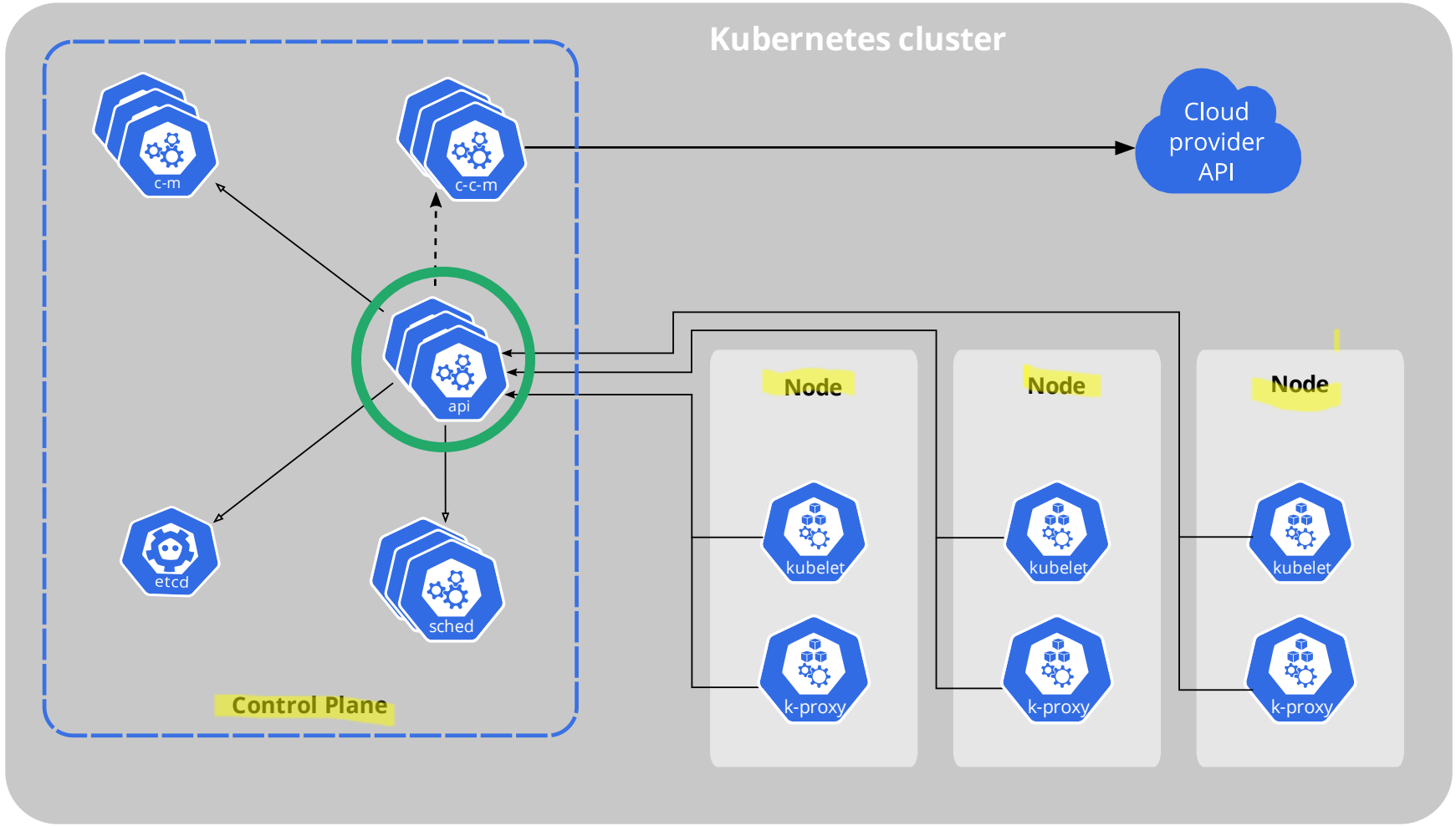
Automatisierung

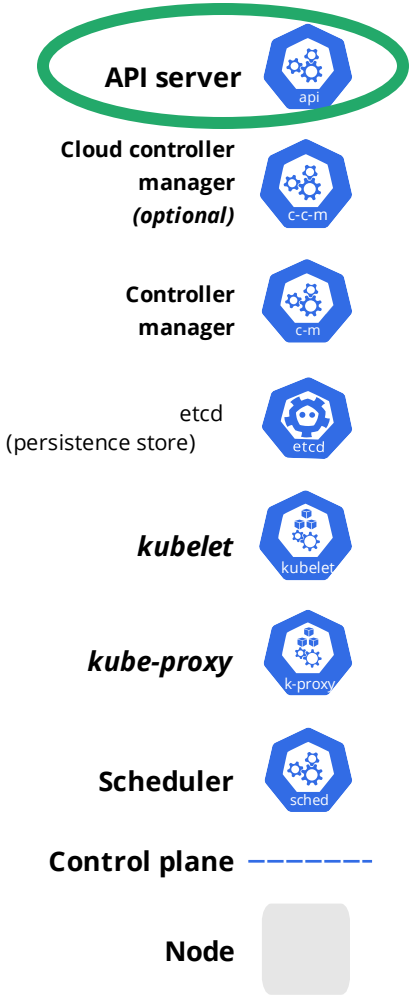
Kubernetes Automatisiert die Bereitstellung, Skalierung und Verwaltung von containersierten Anwendung

Cloud Native

Kubernetes ist ursprünglich von Google entwickelt worden und jetzt wird es von Cloud Native Foundation gepflegt.

Wichtige Kubernetes Komponente





API-Server (kube-apiserver) - Control Plane

Hauptaufgabe

- Stellt die zentrale Schnittstelle für alle Interaktionen mit dem Kubernetes-Cluster bereit.

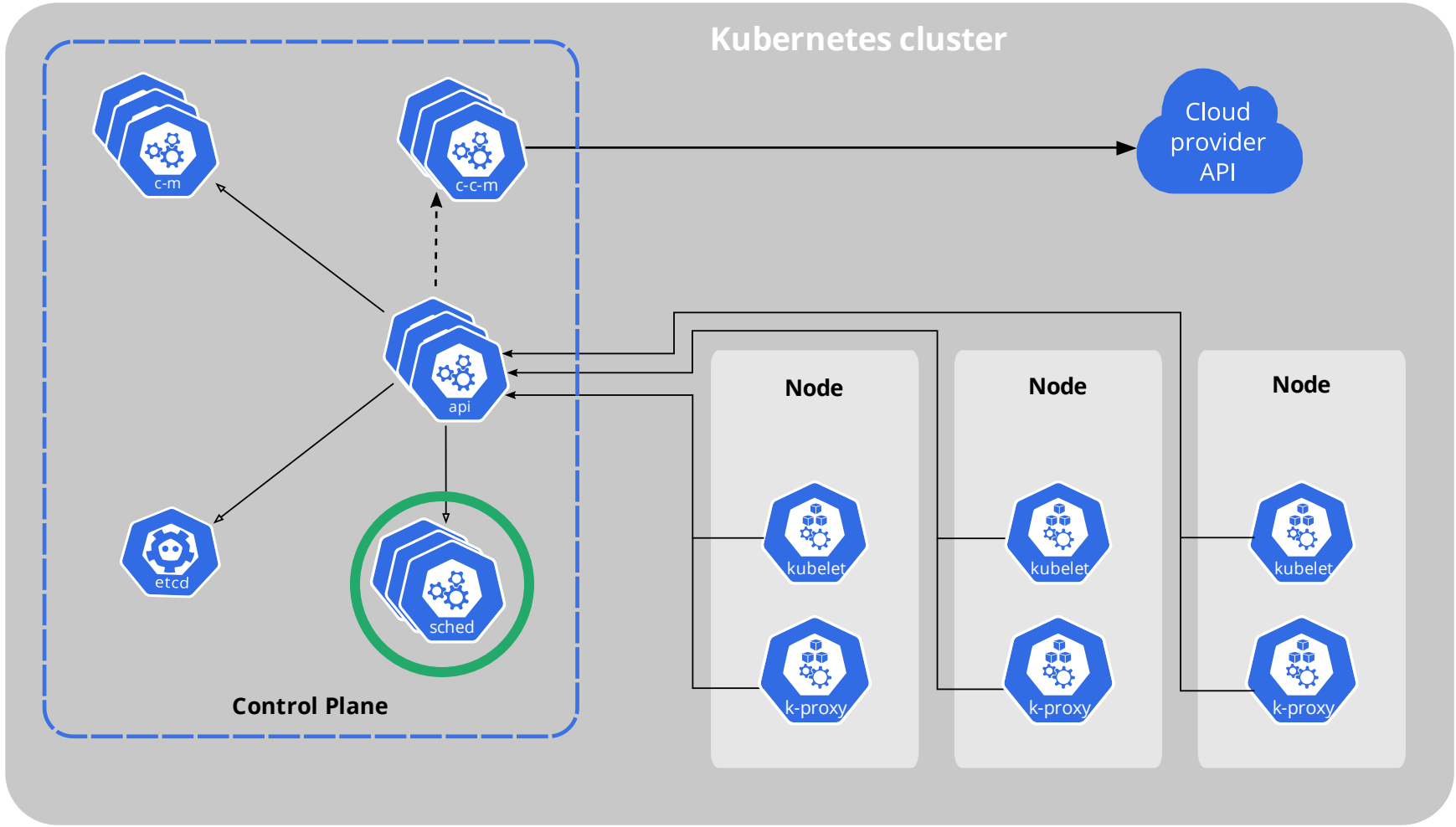
Beispiel

- Wenn ein Benutzer einen Pod bereitstellt (`kubectl apply`), wird diese Anfrage an den API-Server gesendet, der sie überprüft und verarbeitet.

Funktionalität

- Verarbeitet **REST-API-Anfragen** von Benutzern, CLI-Tools (z. B. `kubectl`), und anderen Komponenten.
- Authentifiziert und autorisiert Anfragen basierend auf RBAC (Role-Based Access Control).
- Validiert und speichert Konfigurationen in der **etcd**-Datenbank

Wichtige Kubernetes Komponente



API server

Cloud controller manager *(optional)*

Controller manager

etcd (persistence store)

kubelet

kube-proxy

Scheduler

Control plane

Node

Scheduler (kube-scheduler) - Control Plane

Hauptaufgabe

- Weist neue Pods den passenden Worker Nodes zu.

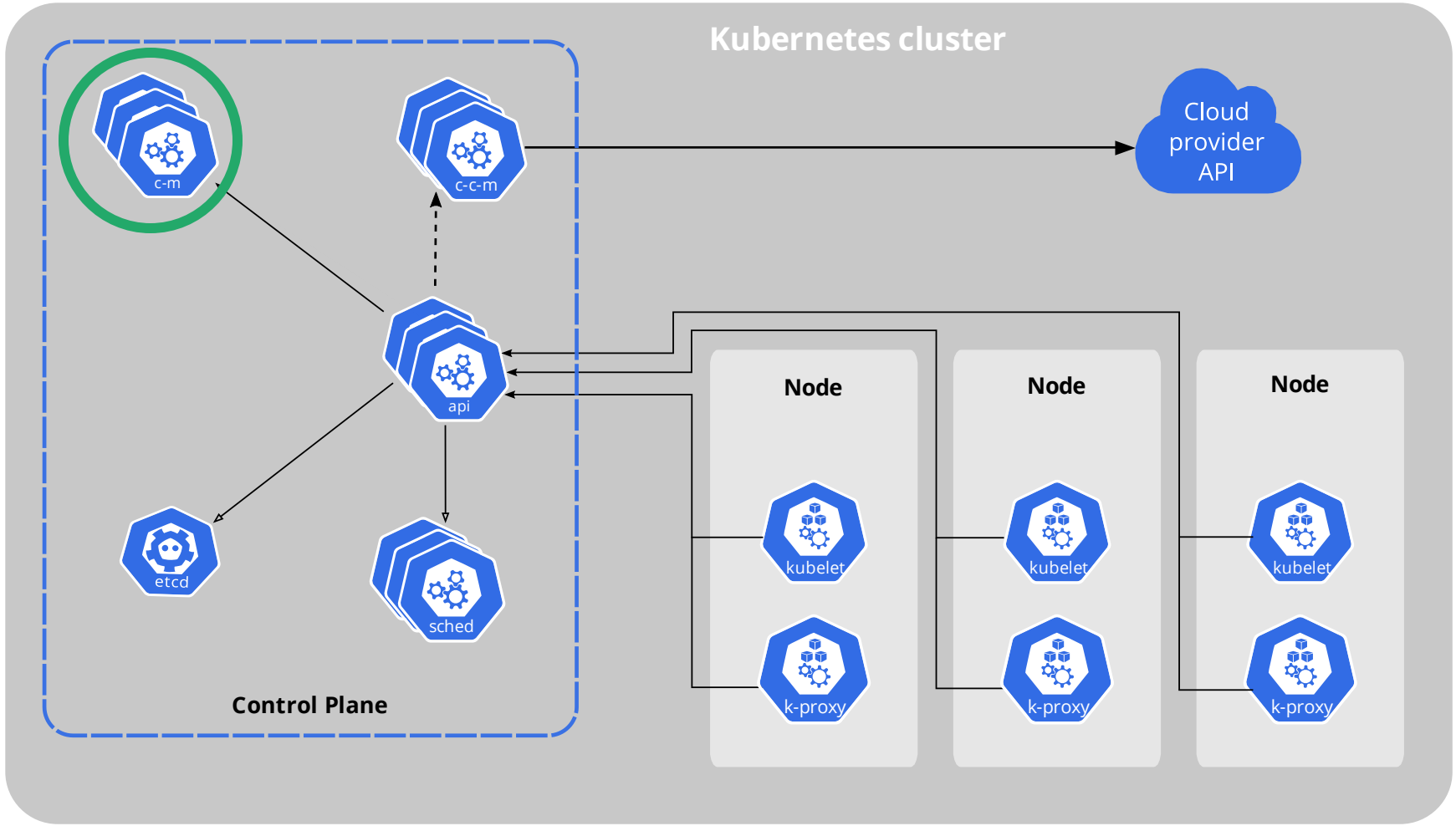
Beispiel

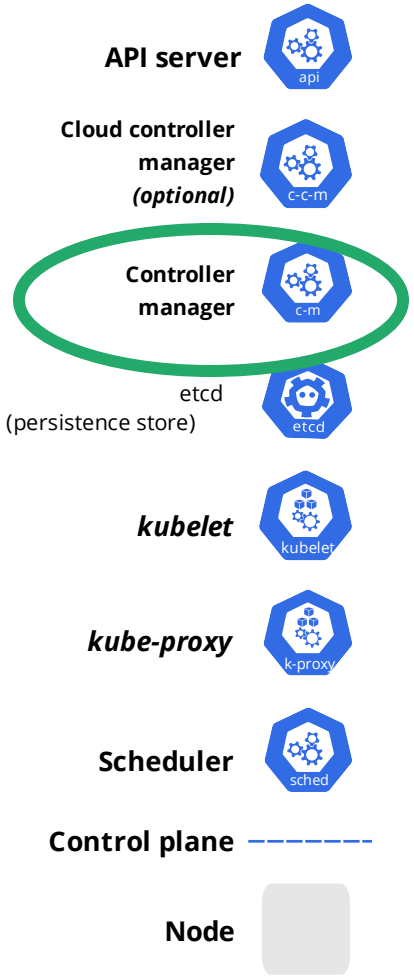
- Wenn ein Deployment 3 Replicas eines Pods erstellt, bestimmt der Scheduler, auf welchen Nodes diese Pods gestartet werden.

Funktionalität

- Analysiert die **Anforderungen der Pods** (z. B. CPU, Speicher).
- Überprüft den Status und die Kapazität der verfügbaren Nodes.
- Wählt die beste Node basierend auf vordefinierten Regeln und Constraints (z. B. Anti-Affinity).

Wichtige Kubernetes Komponente





Controller-Manager (kube-controller-manager) - Control Plane

Hauptaufgabe

- Führt verschiedene Controller aus, die den Cluster überwachen und Aktionen auslösen, um den gewünschten Zustand zu erreichen

Beispiel

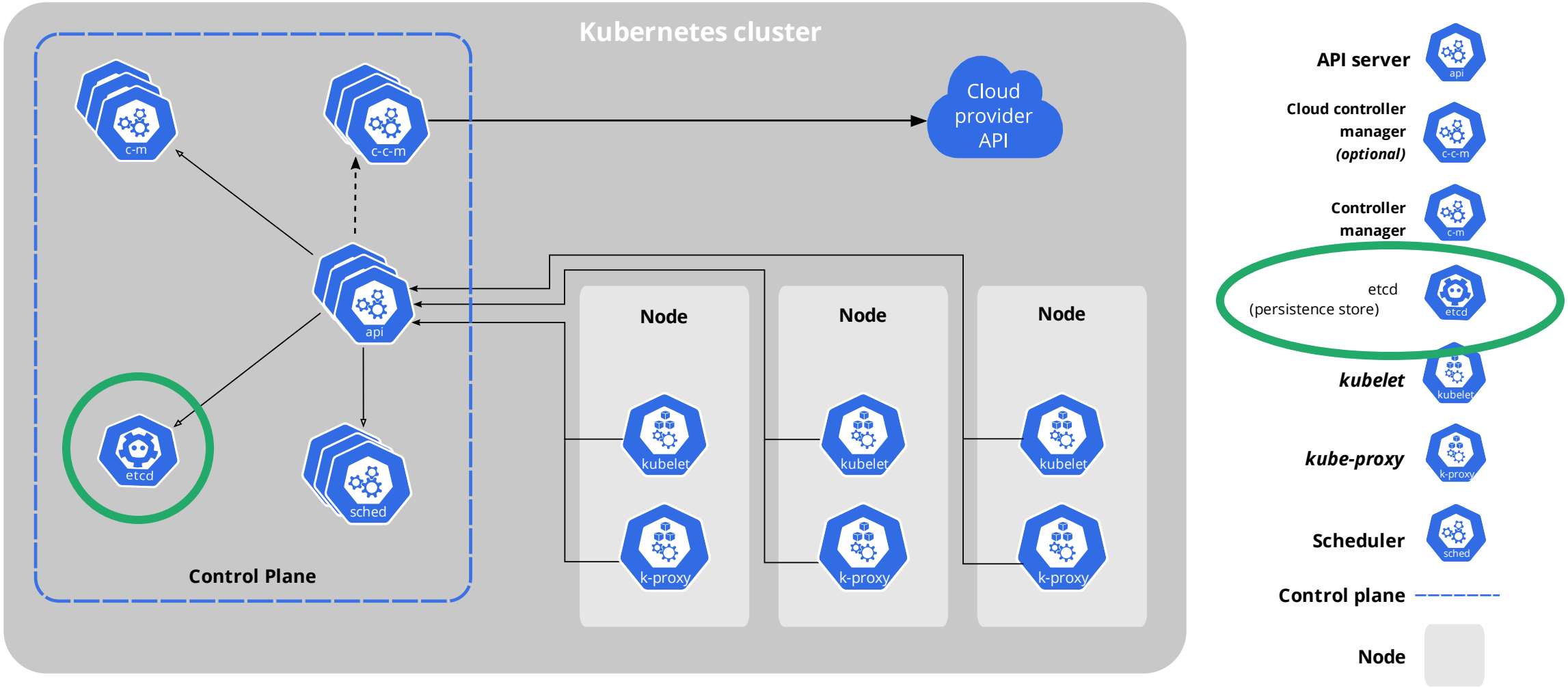
- Wenn ein Node ausfällt, erkennt der Node Controller dies und plant die Pods dieses Nodes auf andere Nodes um.

Funktionalität

Node Controller: Überwacht Nodes und markiert sie als "nicht verfügbar", wenn sie nicht mehr erreichbar sind.

- **Replication Controller:** Stellt sicher, dass die gewünschte Anzahl von Replikaten eines Pods läuft.
- **Endpoint Controller:** Verknüpft Services mit Pods.
- **Job Controller:** Verarbeitet kurzlebige Jobs.

Wichtige Kuberntes Komponente



etcd (kube-etcd) - Control Plane

Hauptaufgabe

- Verteilter, konsistenter Key-Value-Speicher, der die gesamte Cluster-Konfiguration und den Zustand speichert.

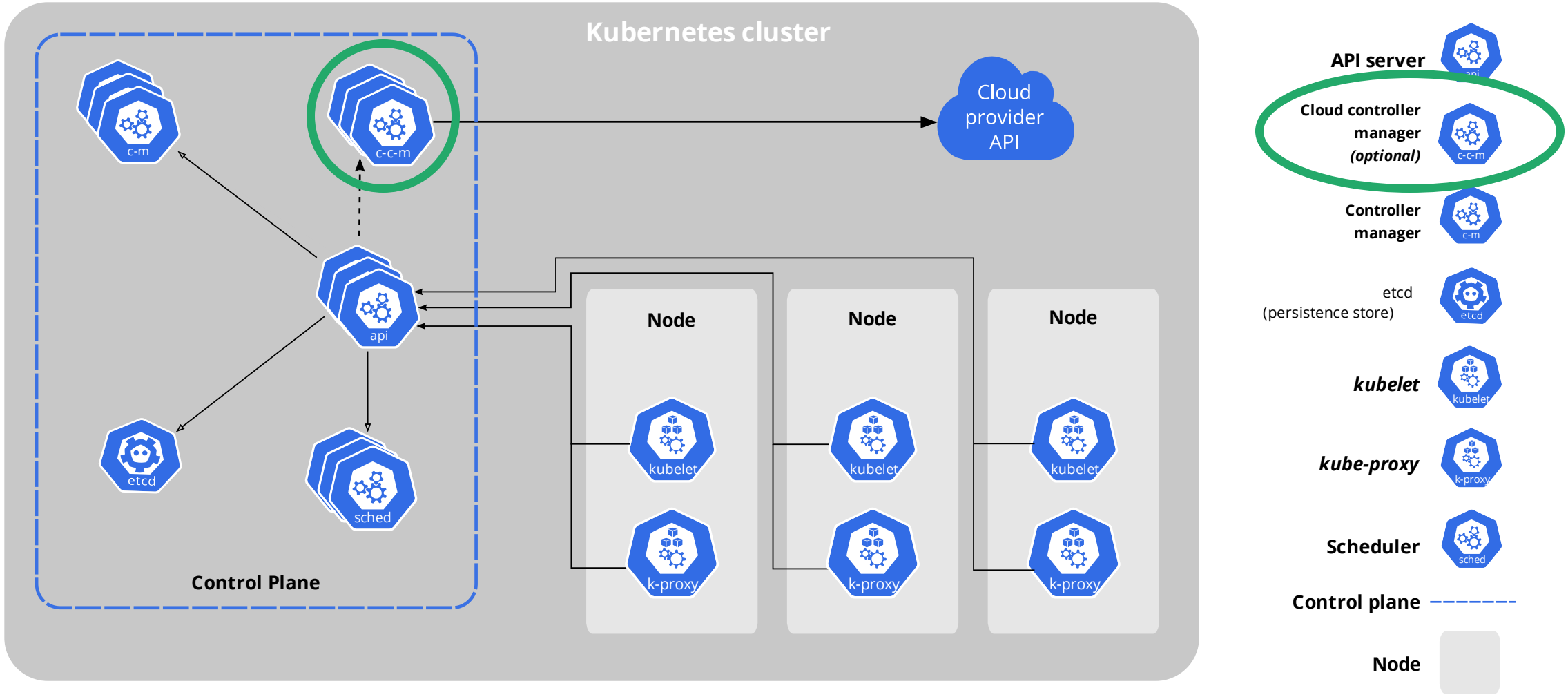
Beispiel

- Wenn der API-Server Anfragen verarbeitet, liest und schreibt er Daten in **etcd**, um die Cluster-Zustände zu synchronisieren.

Funktionalität

- Speichert alle Cluster-Daten, z. B. Pods, Deployments, Services und ConfigMaps.
- Unterstützt Snapshots und Backups, um die Cluster-Konsistenz sicherzustellen.

Wichtige Kubernetes Komponente



Control Plane-Komponenten - Zusammenfassung

API-Server

- Zentrale Schnittstelle für alle Anfragen an den Cluster.
- Validiert und verarbeitet Anfragen (z. B. Pod-Erstellung).
- Speichert Daten im etcd.

Scheduler

- Entscheidet, auf welcher Node neue Pods ausgeführt werden.
- Berücksichtigt Ressourcenanforderungen (z. B. CPU, Speicher) und Constraints.

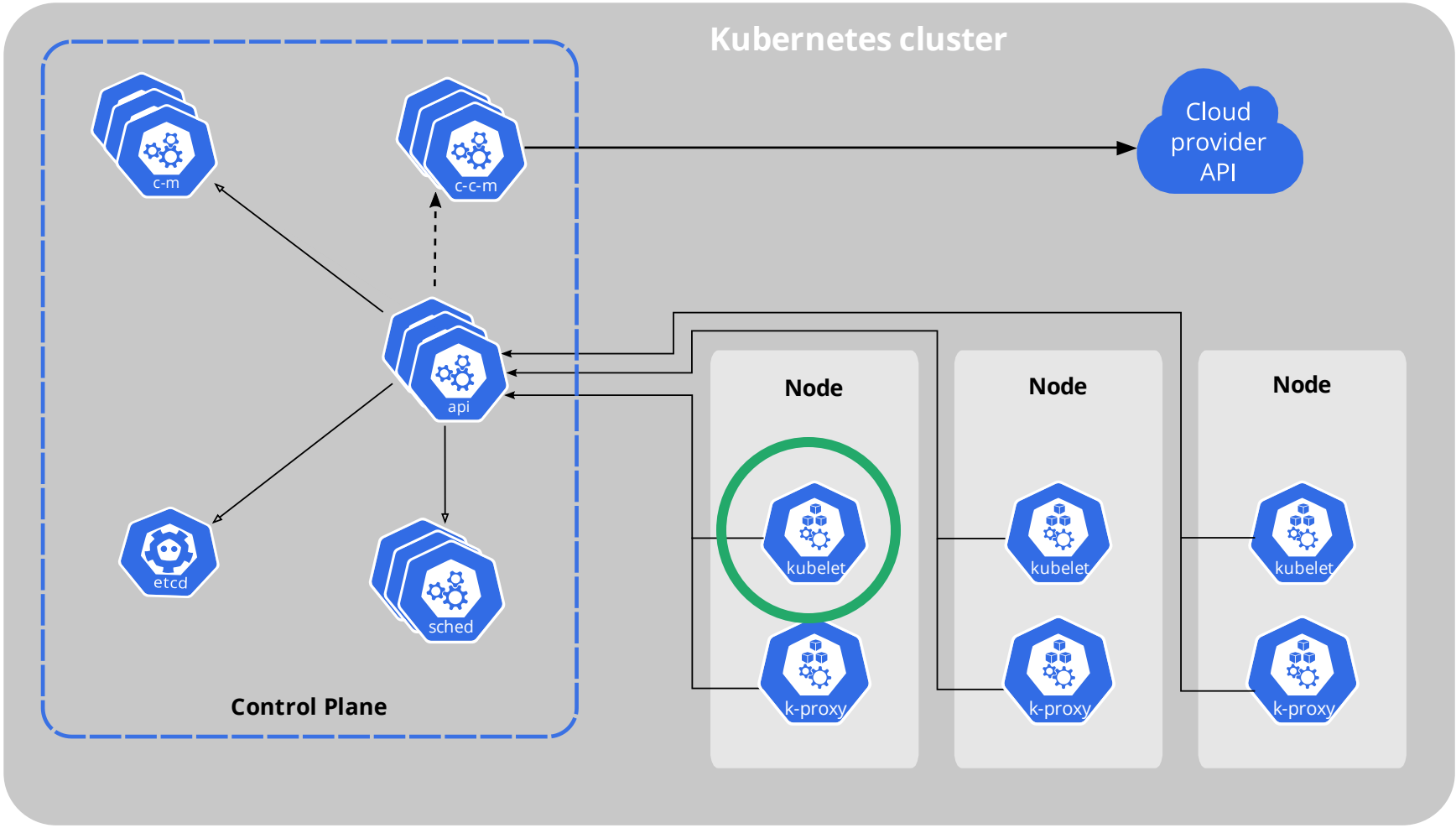
Controller-Manager

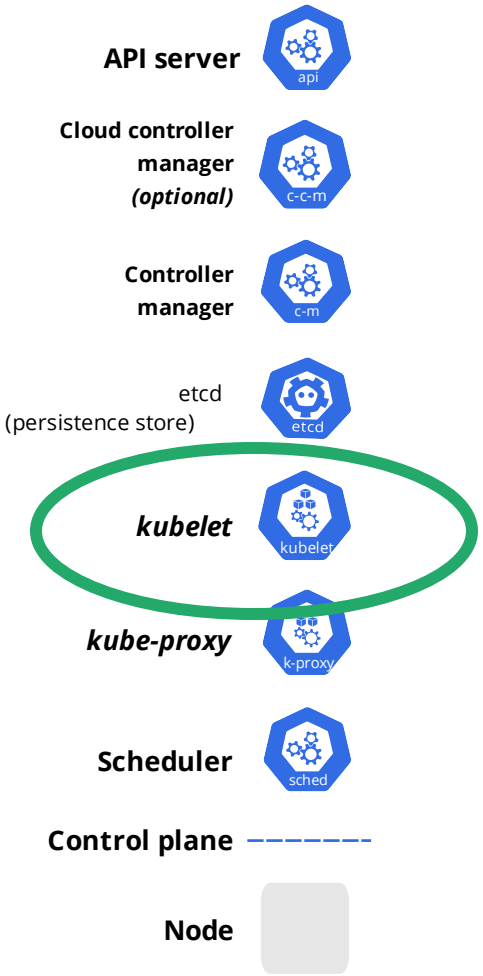
- Führt verschiedene Controller aus, um den gewünschten Zustand des Clusters zu gewährleisten.
- Beispiele: Node Controller, Replication Controller, Endpoint Controller.

Etcd

- Verteilter, konsistenter Key-Value-Speicher für den Cluster-Zustand.
- Speichert Konfigurationsdaten, Deployments, Services und mehr.

Wichtige Kuberntes Komponente





Kube-Proxy - Worker Node

Hauptaufgabe

- Kubelet ist der **Agent**, der auf jeder Worker Node läuft. Es sorgt dafür, dass die Container, die einem Pod zugeordnet sind, entsprechend der Spezifikation ausgeführt werden.

Beispiel

- Wenn ein Pod erstellt wird, zieht kubelet das entsprechende Container-Image von der Registry, startet die Container und überwacht sie.

Funktionen:

Kommunikation mit der Control Plane:

- Empfängt Anweisungen vom API-Server, z. B. zur Erstellung, Aktualisierung oder Löschung eines Pods.
- Meldet regelmäßig den Zustand der Node (z. B. verfügbare Ressourcen wie CPU und Speicher) an den API-Server.

Pod-Management:

- Startet, überwacht und beendet Container innerhalb eines Pods.
- Arbeitet eng mit der Container-Laufzeitumgebung (z. B. Docker, containerd, CRI-O) zusammen.

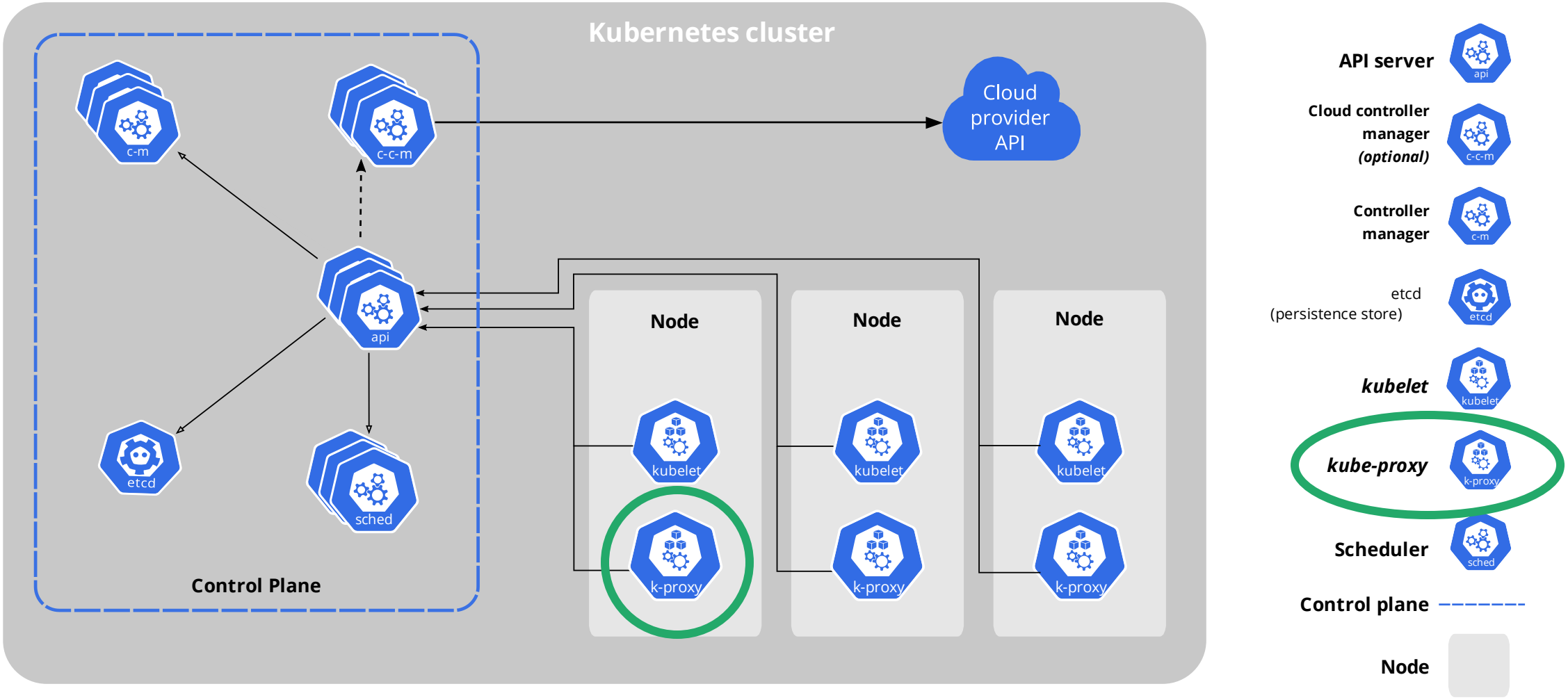
Statusberichte:

- Überwacht den Zustand der laufenden Pods und Container.
- Sendet Statusinformationen (z. B. „Pod läuft“ oder „Pod abgestürzt“) an den API-Server.

Log-Sammlung und Debugging:

- Sammelt Logs von Containern und stellt sie für das Debugging bereit.

Wichtige Kuberntes Komponente



Kube-Proxy - Worker Node

Hauptaufgabe

- Kube-Proxy ist der **Netzwerk-Proxy**, der das Kubernetes-Netzwerk implementiert. Es stellt sicher, dass **Dienste (Services)** korrekt mit den zugehörigen Pods kommunizieren können.

Beispiel

- Ein Benutzer greift auf die URL `http://my-service:80` zu. Kube-Proxy leitet diese Anfrage an einen passenden Pod (z. B. `10.244.0.5:8080`) weiter.

Funktionen:

Service Discovery:

- Implementiert das **Cluster-IP-basierte Routing**: Übersetzt Service-Namen (z. B. `backend-service`) in IP-Adressen und Ports der Pods.

Lastverteilung:

- Verteilt Anfragen an den Service auf die entsprechenden Pods (Load Balancing).

Netzwerkregeln:

- Nutzt **iptables** oder **ipvs**, um Netzwerkregeln für die Weiterleitung von Anfragen zu erstellen.
- Sorgt dafür, dass Anfragen an einen Service (z. B. `http://backend-service:80`) an einen der zugehörigen Pods weitergeleitet werden.

Externe Zugriffe:

- Unterstützt NodePort und LoadBalancer-Services, um Anfragen von außerhalb des Clusters weiterzuleiten.

Control Plane-Komponenten - Zusammenfassung

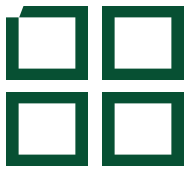
Kubelet:

- Sorgt dafür, dass die Container des Pods auf der Node ausgeführt werden.
- Überwacht den Pod und meldet dessen Status an die Control Plane.

Kube-Proxy:

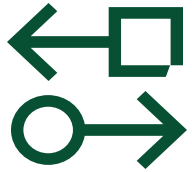
- Ermöglicht die Kommunikation zwischen Services und Pods.
- Stellt sicher, dass Netzwerkanfragen korrekt weitergeleitet werden.

Warum Kubernetes Verwenden



Skalierbarkeit

Bewältigung von Anwendungen mit hohen Traffic-Anforderungen.



Portabilität

Workloads in verschiedenen Umgebungen betreiben.

Self-Healing

Automatische Neustarts von fehlerhaften Containern.

Deklarative Konfigurationen

Definition des gewünschten Zustands mit YAML oder JSON.

Praktischer Teil

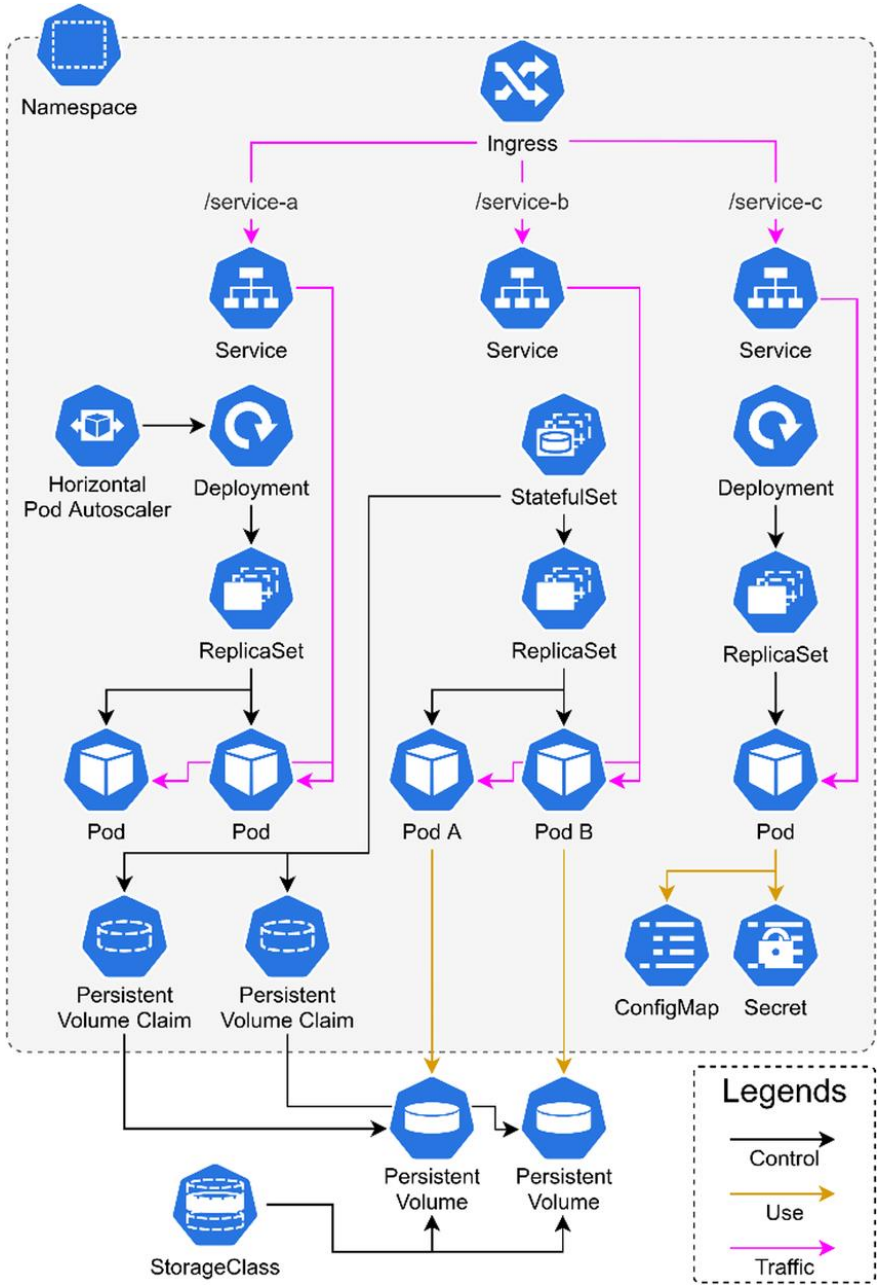
"Kubernetes ist eine Plattform, die Ihnen hilft, genau das zu tun, was Sie in der Cloud benötigen: Anwendungen skalieren, automatisieren und verwalten – unabhängig davon, wo sie laufen."

2

```
git clone https://github.com/tom9eiger/kubernetes-workshop.git
```

Kubernetes – Wichtigste Ressourcen

1. **Namespace**: Logische Trennung.
2. **Pod**: Basiseinheit.
3. **ReplicaSet**: Skalierbarkeit.
4. **Deployment**: Rollouts und Updates.
5. **DaemonSet**: Pods auf jedem Node.
6. **StatefulSet**: Stateful-Anwendungen.
7. **Service**: Netzwerkzugriff und Kommunikation.



Namespace – Logische Trennung von Ressourcen

- **Definition:** Kubernetes-Namespace bietet eine logische Trennung innerhalb eines Clusters.
- **Funktionen:**
  - Gruppiert Ressourcen wie Pods, Deployments und Services.
  - Ermöglicht Multi-Tenancy und Isolierung.
- **Standard-Namespace:**
  - `default`: Standard für Ressourcen.
  - `kube-system`: Für Kubernetes-Systemkomponenten.
  - `kube-public`: Öffentlich zugängliche Ressourcen.

```
# Liste aller Namespaces anzeigen
kubectl get namespaces

# Ressourcen in einem bestimmten Namespace
kubectl get pods -n <namespace>

# Namespace erstellen
kubectl create namespace <namespace-name>

# Namespace wechseln
kubectl config set-context --current --namespace=<namespace-name>
```

Pod – Die Basiseinheit in Kubernetes

- **Definition:** Der Pod ist die kleinste deploybare Einheit in Kubernetes.
- **Funktionen:**
  - Läuft auf einem Node.
  - Kann einen oder mehrere Container enthalten.
- **Anwendungsfall:** Einzelne Anwendungen oder Services.

```yaml
# Example Pod
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: nginx
      image: nginx
```

ReplicaSet – Für Skalierbarkeit und Redundanz

- **Definition:** Stellt sicher, dass eine bestimmte Anzahl identischer Pods läuft.
- **Funktionen:**
  - Unterstützt Skalierung (Replikate hinzufügen/entfernen).
  - Ersetzt fehlerhafte Pods automatisch.
- **Anwendungsfall:** Statische Workloads mit konstanter Skalierung.

```yaml
# Example ReplicaSet
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: example-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
    app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: nginx
          image: nginx
```

Deployment – Für Skalierbarkeit und Redundanz

- **Definition:** Abstraktion über ReplicaSet, bietet Rollouts und Versionsverwaltung.
- **Funktionen:**
  - Unterstützt **Rolling Updates** und **Rollbacks**.
  - Einfaches Skalieren.
- **Anwendungsfall:** Häufig für Webanwendungen oder APIs.

```yaml
# Example Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: nginx
          image: nginx
```

DaemonSet – Pods auf jedem Node

- **Definition:** Stellt sicher, dass ein Pod auf jedem Node läuft.
- **Funktionen:**
  - Ideal für Logging, Monitoring oder Netzwerkanwendungen.
- **Anwendungsfall:** Node-spezifische Dienste wie **Fluentd**, **Prometheus Node Exporter**.

```yaml
# Example DaemonSet
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: example-daemonset
spec:
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: fluentd
          image: fluentd
```

StatefulSet – Pods auf jedem Node

- **Definition:** Verwalten von Pods mit persistenter Identität und Speicher.
- **Funktionen:**
  - Statische Namen für Pods.
  - Geordnete Starts und Stops.
  - Unterstützt PersistentVolumeClaims (PVCs).
- **Anwendungsfall:** Datenbanken, Kafka, Redis.

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: example-statefulset
spec:
  serviceName: "example-service"
  replicas: 2
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
        - name: redis
          image: redis
          volumeMounts:
            - name: data
              mountPath: /data
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 1Gi
```

Ende Part 1

bechtle

Willkommen zum **Kubernetes** ***Basic Kurs – Part 2***

Durchgeführt von Thomas Geiger.

Mägenwil, Thomas Geiger

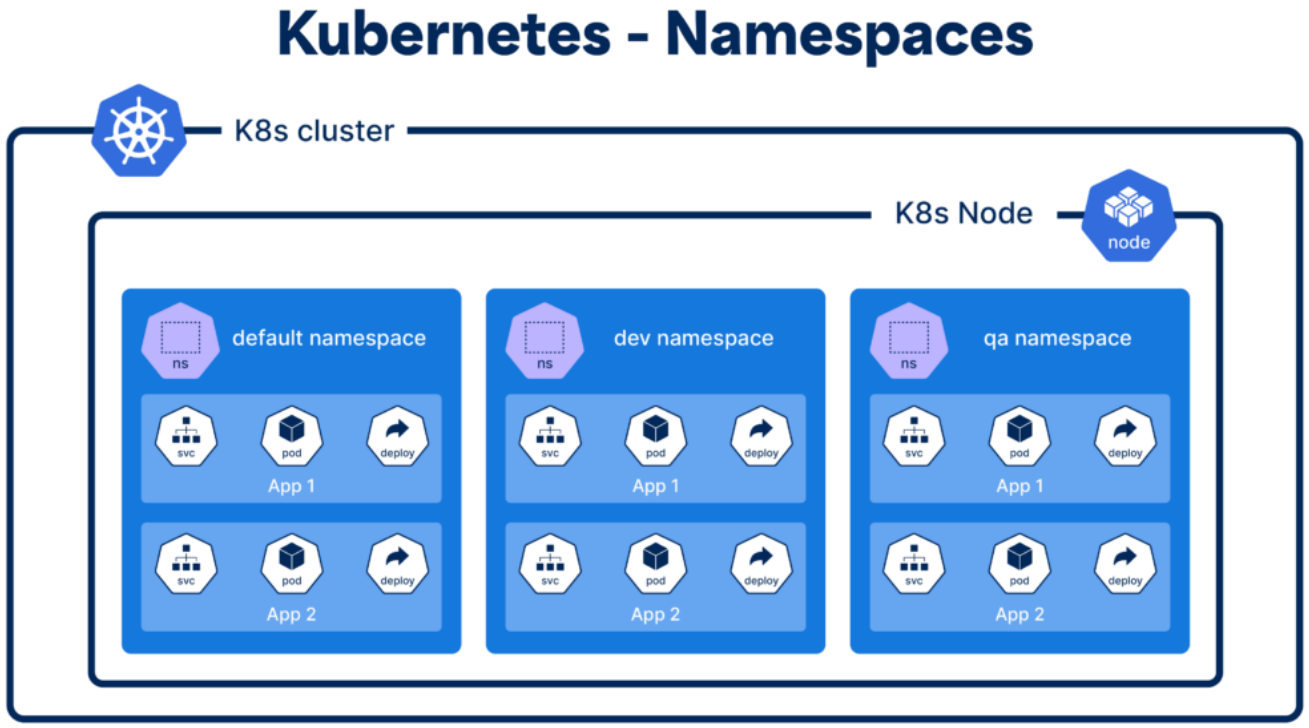
31.01.2025

Agenda

1. **Theorie Teil 1: Namespace in Kubernetes**
2. **Praktischer Teil 1: Namespace Kubernetes**
3. **Theorie Teil 2: DNS in Kubernetes**
4. **Praktischer Teil 2: DNS in Kubernetes**
5. **Theorie Teil 3: Networking in Kubernetes**
6. **Praktischer Teil 3: Networking in Kubernetes**

Kubernetes – Einführung Namespace

- **Für was brauchen wir Namespaces:**
  - Zur logischen Trennung von Ressourcen innerhalb eines Clusters.
- **Anwendungsfälle:**
  - Multi-Tenancy (z. B. `dev`, `prod`).
  - Organisation nach Teams oder Projekten.
- **Standard-Namespaces:**
  - `Default`: Allgemeine Nutzung.
  - `kube-system`: Kubernetes-Systemressourcen.
  - `kube-public`: Öffentlich zugängliche Ressourcen.



Kubernetes – Vorteile von Namespaces

Isolierung von Ressourcen (z. B. dev vs. prod).

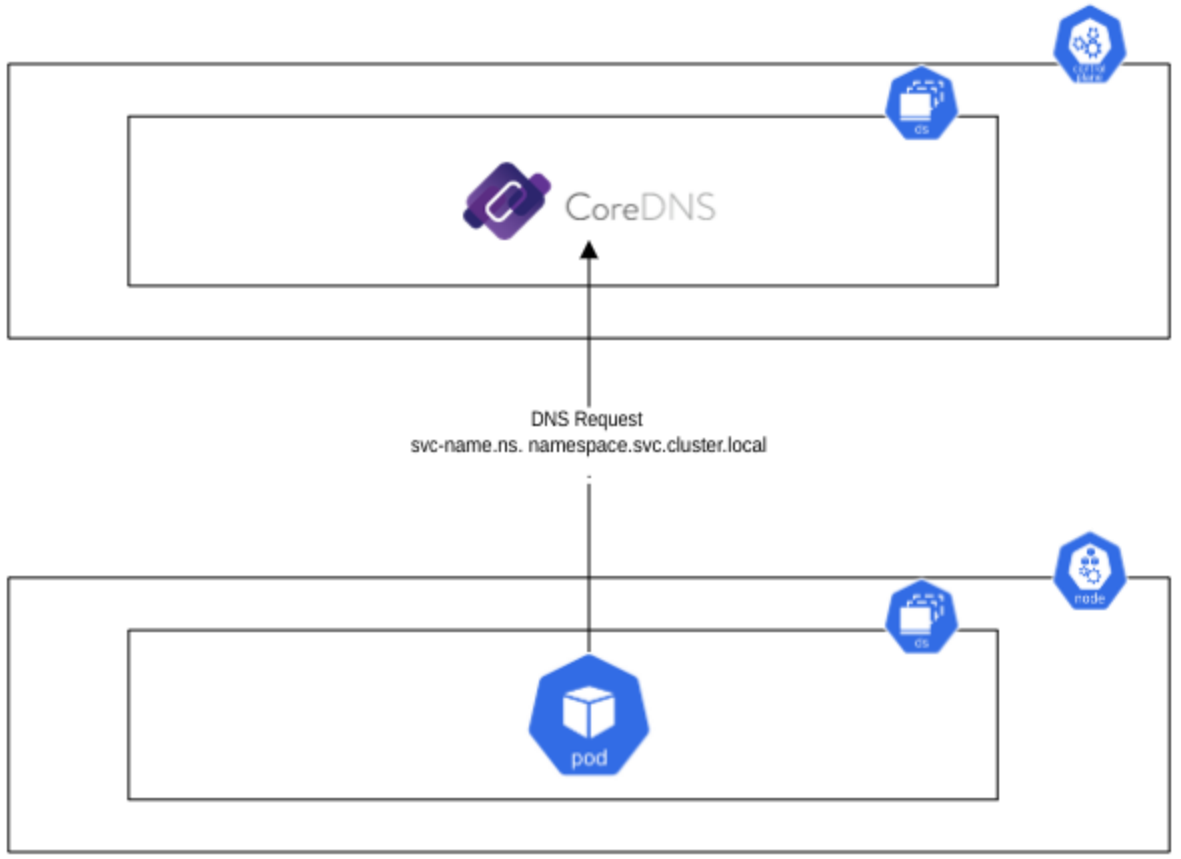
Anwendung von Role-Based Access Control (RBAC).

Ressourcen - management durch Quotas.

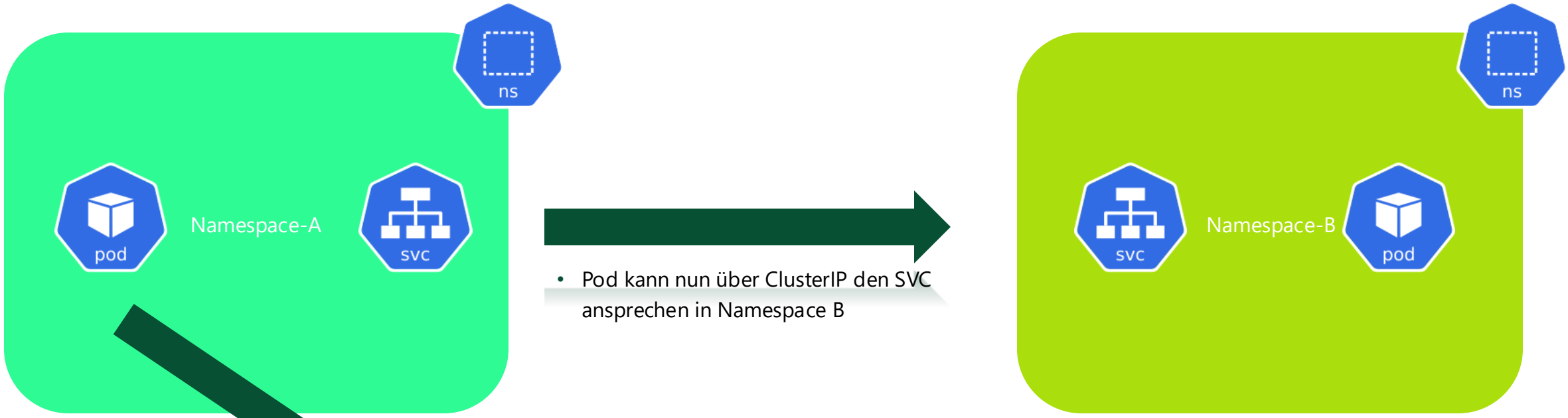
Praktische Übung Teil1: Namespace

Kubernetes – Was macht der CoreDNS

- **Für was brauchen wir DNS in Kubernetes**
  - DNS (Domain Name System) wird in Kubernetes für die Service-Erkennung verwendet.
  - **CoreDNS** ist der Standard-DNS-Dienst in Kubernetes.
- **Auflösefunktionen:**
  - **Services:** Übersetzt Servicenamen in ClusterIPs.
  - **Pods:** (Optional) Übersetzt Pod-Namen in Pod-IPs.
- **Wie funktioniert CoreDNS:**
  - Läuft als **Deployment** im Namespace kube-system.
  - Überwacht die Kubernetes-API auf Ressourcenänderungen.
  - Aktualisiert DNS-Einträge dynamisch für Services und Pods.

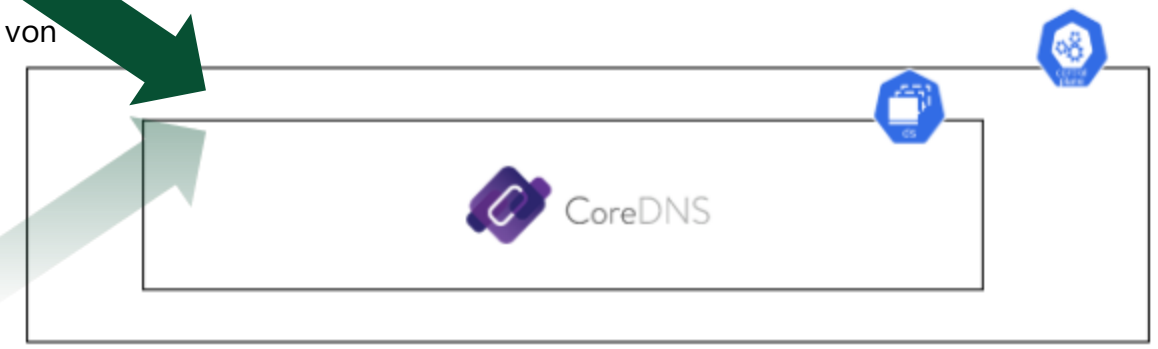


Kubernetes – DNS-Flow in Kubernetes



- Pod kann nun über ClusterIP den SVC ansprechen in Namespace B

- Pod fragt CoreDNS and für ClusterIP von svc-a.namespace-b.svc.cluster.local



Kubernetes – DNS-Namenskonvetionen

- **Service-DNS:**
  - `<service-name>.<namespace>.svc.cluster.local`
  - Beispiel: `nginx.dev.svc.cluster.local`
- **Pod-DNS (Optional):**
  - `<pod-ip>.<namespace>.pod.cluster.local`
- **ExternalName-Services:**
  - Leiten zu externen Domains weiter (z. B. example.com).

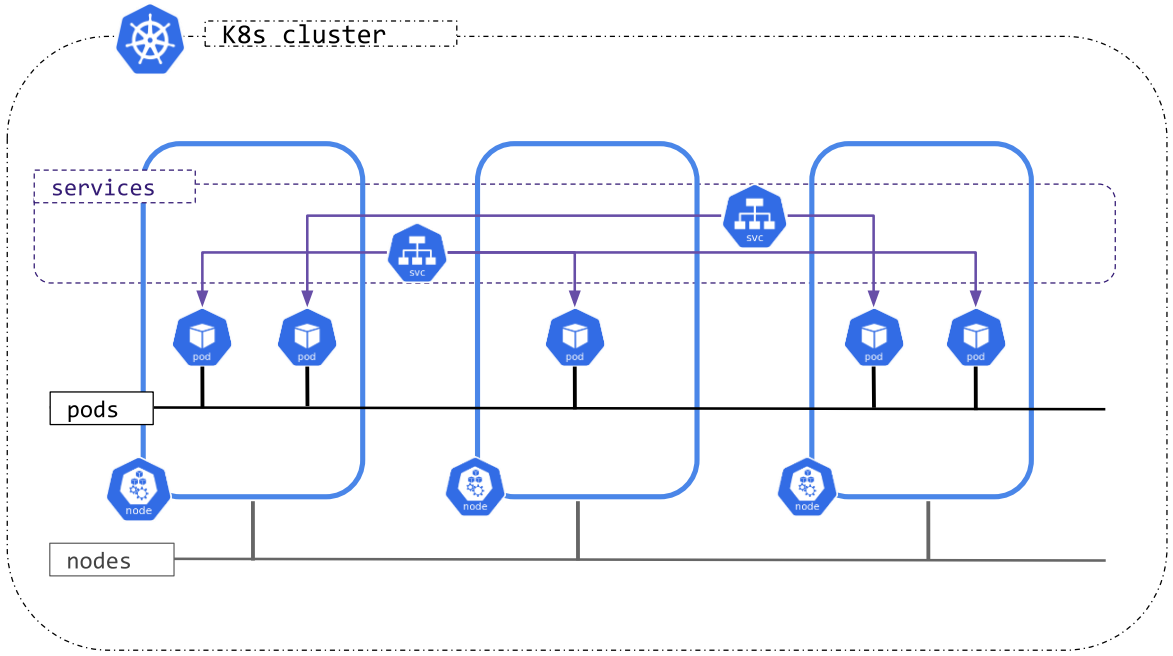
DNS-Name	Beschreibung	Beispiel
`<service-name>.<namespace>.svc.cluster.local`	Interne DNS für Kubernetes-Services.	`nginx.dev.svc.cluster.local`
`<service-name>`	Kurzform für Services im selben Namespace.	`nginx`
`<pod-ip>.<namespace>.pod.cluster.local`	Optional: DNS-Einträge für Pods (Pod-IP-Auflösung).	`10-244-0-5.dev.pod.cluster.local`
`kubernetes.default.svc.cluster.local`	DNS für die Kubernetes-API.	`kubernetes.default.svc.cluster.local`
`<external-service-name>`	Externe DNS-Auflösung mit ExternalName-Services.	`example.com`
`.<namespace>.svc.cluster.local`	DNS-Einträge für alle Services in einem Namespace.	`.dev.svc.cluster.local`
`.<namespace>.pod.cluster.local`	DNS-Einträge für alle Pods in einem Namespace.	`.dev.pod.cluster.local`
`custom.domain.local`	Benutzerdefinierte Domains (mit CoreDNS konfiguriert).	`internal.domain.local`

Praktische Übung Teil 2: DNS in Kubernetes

Kubernetes – Einfürung Netzwerk in k8s

Kubernetes-Netzwerke ermöglichen die Kommunikation zwischen Containern, Services und externen Systemen.

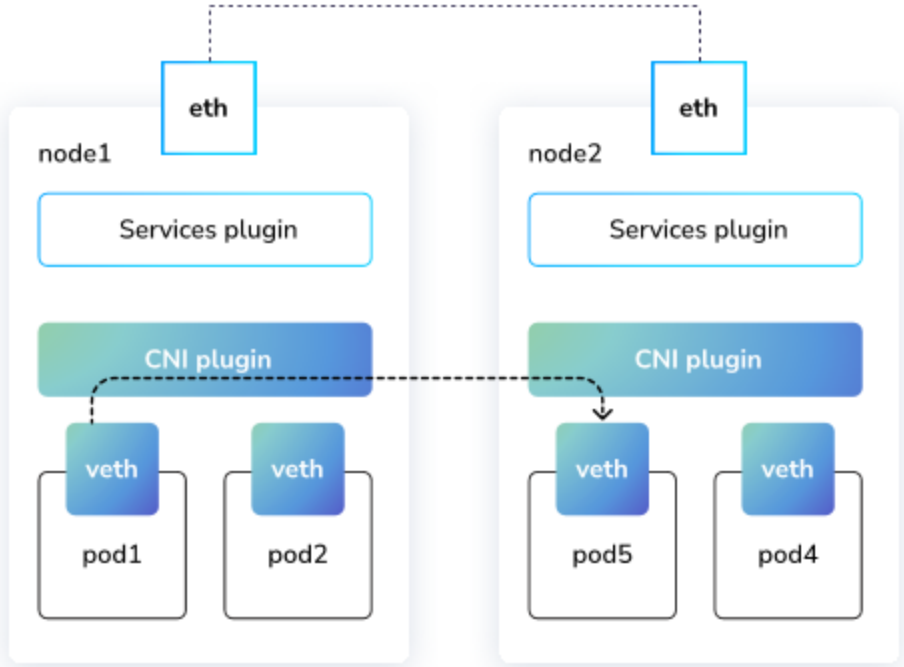
- **Pod-Netzwerk:**
  - Ermöglicht die Kommunikation zwischen Pods im Cluster.
- **Service-Netzwerk:**
  - Verwendet stabile IP-Adressen und DNS für die Service-Erkennung.
  - Lenkt Traffic zu den richtigen Pods.
- **Node-Netzwerk:**
  - Verbindet Nodes miteinander und ermöglicht externe Zugriffe.
  - Unterstützt Speichernetzwerke und Management.



Kubernetes – Pod-Netzwerk in Kubernetes

Bereitgestellt durch das **CNI** (Container Network Interface).

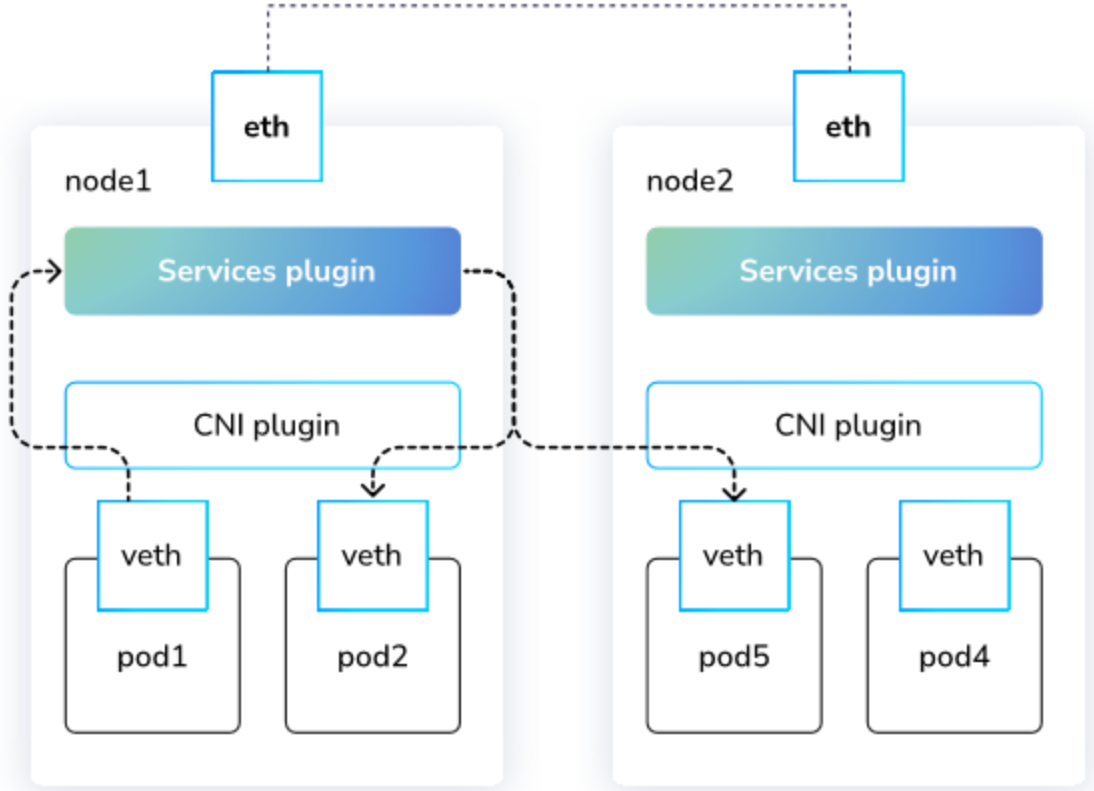
- **Funktion:**
  - Stellt transparente Konnektivität zwischen allen Containern und Nodes her.
- **Verbindungsarten:**
  - Über Bridges oder physische Schnittstellen.
- **IP-Adressverwaltung (IPAM):**
  - CNI verwaltet Pod-IP-Adressen, oft über einen einfachen DHCP-Mechanismus.
- **Netzwerkrichtlinien (Network Policies):**
  - Kontrolle des **Ingress Traffic** (eingehend).
  - Kontrolle des **Egress Traffic** (ausgehend).



Kubernetes – Service-Netzwerk

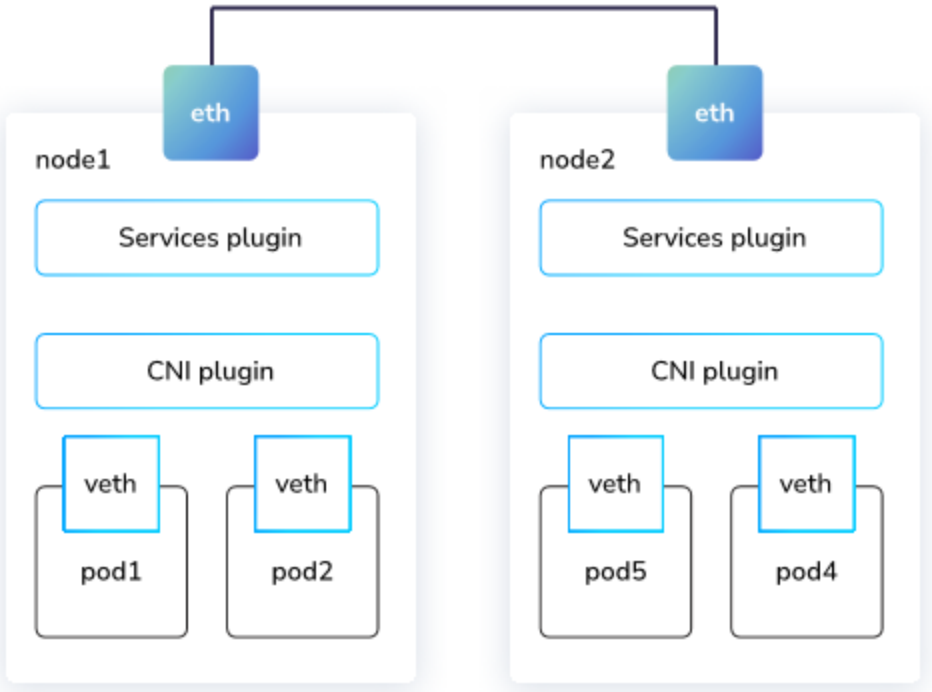
Bereigestellt durch **kube-proxy**

- **Service Discovery:**
  - Ermöglicht die Erkennung von Services über DNS-Namen oder stabile Cluster-IPs.
- **Verkehrslenkung:**
  - Lenkt Traffic zu den richtigen Pods.
  - Verwaltet durch **kube-proxy** oder moderne CNI-Plugins.
- **NAT-Regeln:**
  - Verwendet NAT (Network Address Translation) über **iptables** oder **IPVS**.
- **Externe Verbindungen:**
  - Kann mit Ingress oder Gateways für externe Zugriffe kombiniert werden.



Kubernetes – Node-Netzwerk

- **Verwaltung:**
  - Wird vom Rechenzentrum oder Cloud-Anbieter verwaltet.
- **Mehrere Netzwerke:**
  - Node-to-Node-Netzwerk: Kommunikation zwischen Nodes im Cluster.
  - Storage-Netzwerk: Verbindung zu externen Speicherlösungen.
  - Externes Netzwerk: Ermöglicht Zugriff auf das Internet.
  - Management-Netzwerk: Verwaltung der Nodes (z. B. für Monitoring oder SSH).



Praktische Übung Teil 3: Netzwerk in Kubernetes

bechtle

Willkommen zum **Kubernetes** ***Basic Kurs – Part 3***

Durchgeführt von Thomas Geiger.

Mägenwil, Thomas Geiger

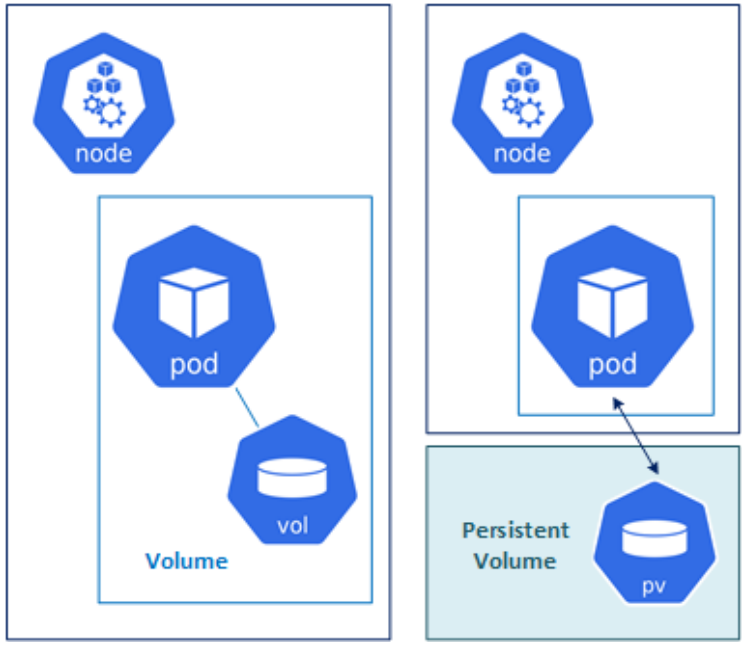
31.01.2025

Agenda

1. **Theorie Teil 1: Kubernetes Storage**
2. **Praktischer Teil 1: Kubernetes Storage**
3. **Theorie Teil 2: Deployment Methods in Kubernetes**
4. **Praktischer Teil 2: Deployment Method in Kubernetes**

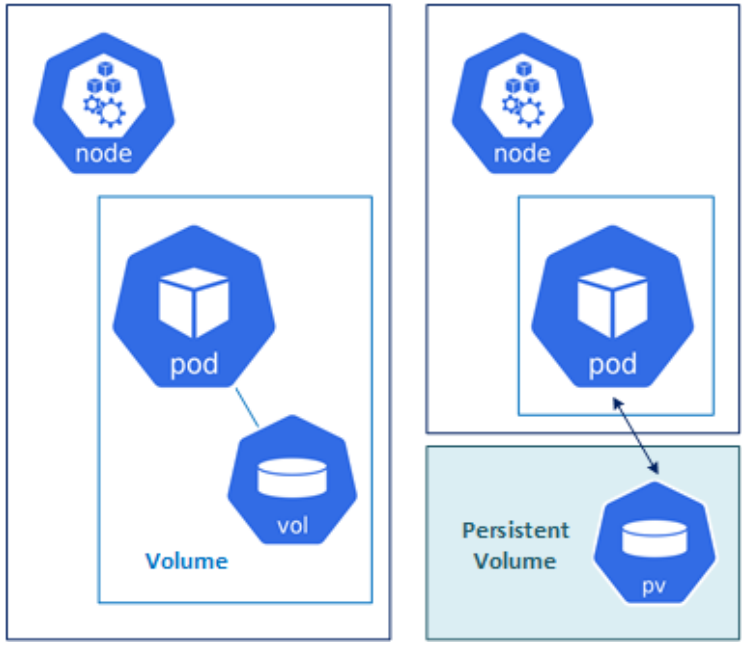
Kubernetes – Storage im Überblick

- Kubernetes-Container sind von Natur aus flüchtig – alle im Container gespeicherten Daten gehen beim Neustart oder Löschen eines Pods verloren.
- Persistenter Speicher ist erforderlich, um Daten dauerhaft zu speichern, auch wenn Pods neu gestartet werden.
- **Anwendungsfälle:**
  - Datenbanken (z. B. MySQL, MongoDB)
  - Dateisysteme (z. B. Logs, hochgeladene Dateien)
  - Zustandsbehaftete Anwendungen, die Daten zwischen Neustarts beibehalten müssen.



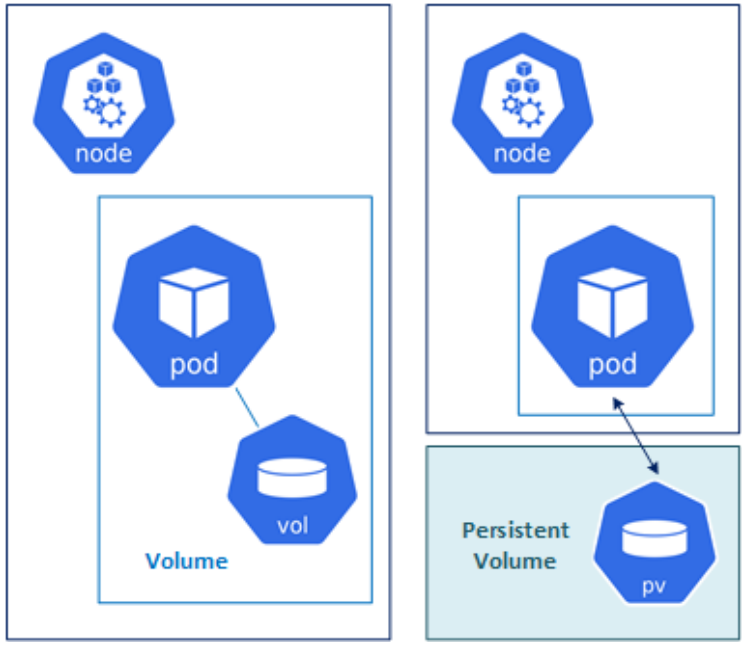
Kubernetes – Kubernetes Volumes

- Ein Volume in Kubernetes ist ein Verzeichnis, das einem Container zum Speichern von Daten bereitgestellt wird.
- **Wichtige Arten von Volumes:**
  - **emptyDir:** Temporärer Speicher, der beim Löschen des Pods ebenfalls gelöscht wird.
  - **hostPath:** Verzeichnis auf dem Host-Node, das dem Container zur Verfügung gestellt wird.
  - **PersistentVolumeClaim (PVC):** Ermöglicht dauerhafte Speicherung über die Lebensdauer des Pods hinaus.



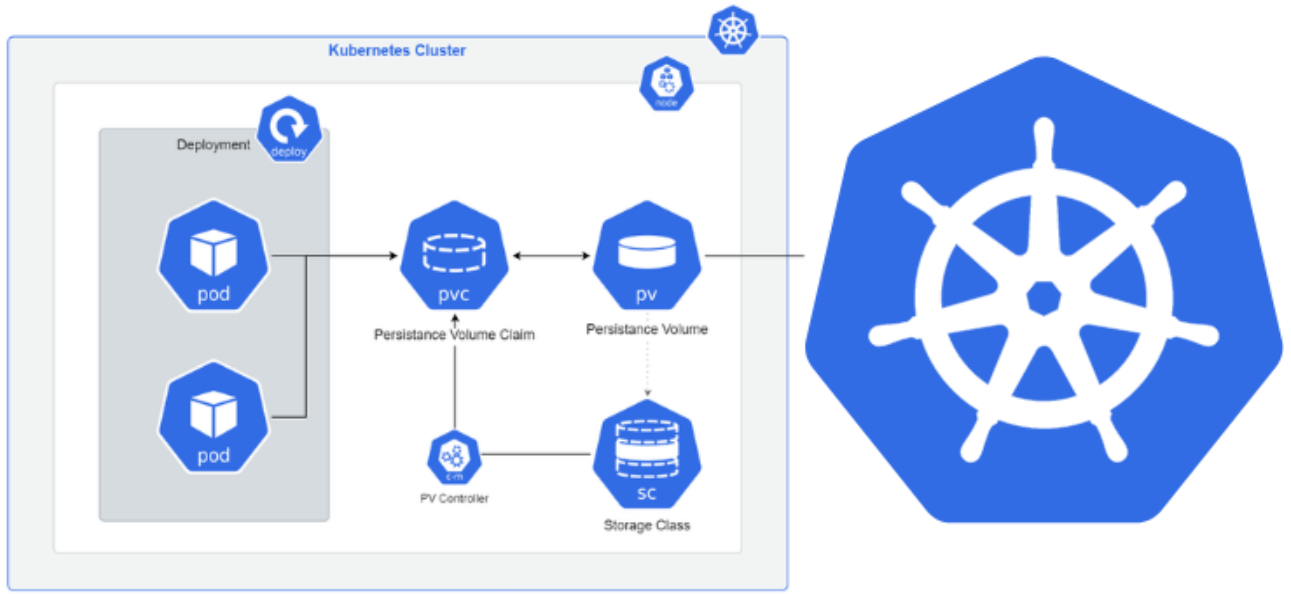
Kubernetes – PersistentVolume (PV) und PersistentVolumeClaim (PVC)

- **PersistentVolume (PV):**
  - Eine Ressource im Cluster, die Speicherplatz bereitstellt.
  - Kann vom Cluster-Administrator vorkonfiguriert werden.
- **PersistentVolumeClaim (PVC):**
  - Eine Anfrage eines Nutzers nach Speicherplatz.
  - Kubernetes ordnet automatisch eine passende PV-Ressource der Anfrage zu.
- **Vorteil**:
  - Die Trennung von PV und PVC ermöglicht flexibles Speicher-Management und Wiederverwendung.



Kubernetes – StorageClass und dynamische Bereitstellung

- Eine **StorageClass** definiert verschiedene Arten von Speicher (z. B. SSD, HDD) mit unterschiedlichen Leistungsmerkmalen.
- Kubernetes unterstützt die automatische Erstellung von PersistentVolumes basierend auf PVC-Anfragen und der StorageClass.



Kubernetes – Zugriffsmodi (Access Modes) PVC

- **ReadWriteOnce (RWO):**
  - Das Volume kann nur von einem Pod auf einem einzigen Node im Lese- und Schreibmodus verwendet werden.
  - Geeignet für lokale Speicherlösungen wie HostPath oder AWS EBS.
- **ReadOnlyMany (ROX):**
  - Das Volume kann von mehreren Pods auf verschiedenen Nodes gleichzeitig im Lesezugriff verwendet werden.
  - Praktisch für Anwendungsfälle wie gemeinsam genutzte Konfigurationsdateien.
- **ReadWriteMany (RWX):**
  - Das Volume kann von mehreren Pods auf verschiedenen Nodes gleichzeitig im Lese- und Schreibmodus verwendet werden.
  - Erfordert ein verteiltes Dateisystem wie NFS, Ceph oder GlusterFS.

Kubernetes – Speicherklassenparameter (StorageClass Parameters)

- **ReclaimPolicy**: Gibt an, was mit dem Volume geschieht, wenn die zugehörige PVC gelöscht wird.
  - `Retain:` PV bleibt bestehen.
  - `Delete:` PV wird zusammen mit der PVC gelöscht.
- **MountOptions**: Optionen, die beim Mounten des Volumes verwendet werden (z. B. `noatime`).
- **Provisioner**: Definiert den Speicheranbieter (z. B. `kubernetes.io/aws-ebs` oder `nfs`).

Praktische Übung Teil 1: Kubernetes Storage

Kubernetes – Warum Deployment-Methoden wichtig sind

- Kubernetes unterstützt mehrere Methoden zum Deployment von Anwendungen.
- Jede Methode hat spezifische Vorteile und eignet sich für unterschiedliche Anwendungsfälle.
- Wichtige Methoden:
  - **Manifeste**: Direkte YAML-Dateien.
  - **Kustomize:** Overlay-basierte Konfiguration.
  - **Helm:** Kubernetes-Paketmanager.

Kube-Proxy - Manifests: Die Basis

Definition

- YAML-Dateien, die Kubernetes-Ressourcen wie Deployments, Services und ConfigMaps beschreiben.

Beispiel:

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
name: my-app
spec:
  replicas: 3
  template:
     metadata:
  labels:
    app: my-app
  spec:
    containers:
      - name: my-app
       image: my-app:1.0
```

Vorteile:

- Einfache, direkte Kontrolle über Ressourcen.
- Keine zusätzlichen Tools erforderlich.
- Gut geeignet für kleine, einfache Setups.

Nachteile:

- Wenig Wiederverwendbarkeit.
- Schwer skalierbar für komplexe Anwendungen.
- Manuelle Pflege bei Änderungen.

Kube-Proxy - Helm: Paketmanager für Kubernetes

Definition

- Helm ist ein Tool, das Ressourcen als Pakete (Charts) verwaltet.

Helm Struktur:

```
my-chart/
  Chart.yaml
  values.yaml
  templates/
```

Helm Command:

```
helm install my-release my-chart --values custom-values.yaml
```

Vorteile:

- Skalierbar für komplexe Anwendungen.
- Integrierte Versionierung und Rollbacks.
- Große Community und viele vorgefertigte Charts.

Nachteile:

- Komplexität bei der Erstellung eigener Charts.
- Abhängigkeit von Helm-Tools und -Versionen.

Kube-Proxy - Kustomize: Ressourcen deklarativ patchen

Definition

- Werkzeug zur Patch-Verwaltung und Konfiguration von YAML-Dateien, ohne Templates.

Ordnerstruktur:

```
base/
  deployment.yaml
overlays/prod/
  kustomization.yaml
  patch.yaml
```

Command:

```
kubectl apply -k overlays/prod
```

Vorteile:

- Keine neue Sprache, basiert auf YAML.
- Einfaches Management von Umgebungen (Dev, Test, Prod).
- Gut kombinierbar mit GitOps.

Nachteile:

- Weniger Flexibilität bei dynamischen Werten.
- Komplex bei sehr großen Projekten.

Kubernetes – Vergleich der Methoden

Kriterium	Manifests	Helm	Kustomize
Einfachheit	Hoch (bei einfachen Projekten)	Mittel (abhängig vom Chart)	Hoch (deklarativ)
Wiederverwendbarkeit	Niedrig	Hoch	Mittel bis Hoch
Umgebungsmanagement	Schwach	Gut (mit values.yaml)	Sehr gut (Overlays)
Abhängigkeiten	Keine	Helm-CLI und -Repo	Keine
Komplexität	Steigt schnell an	Komplex bei großen Charts	Komplex bei großen Overlays

Kubernetes – Anwendungsfälle

- **Manifests**:
  - Einfache, einmalige Deployments.
  - Lernen und Testen von Kubernetes-Grundlagen.
- **Helm**:
  - Große, komplexe Anwendungen mit mehreren Abhängigkeiten.
  - Community-basierte Charts für Standardsoftware (z. B. Nginx, Prometheus).
  - Versionierung und Rollbacks erforderlich.
- **Kustomize:**
  - Projekte mit mehreren Umgebungen (Dev, Test, Prod).
  - Integration in GitOps-Pipelines.
  - Anpassung bestehenden Manifests ohne Templating.

Praktische Übung Teil 2: Deployment Methoden