ADMIN:DOCS

Kubernetes: Eine umfassende Dokumentation zu Container-Orchestrierung

Eine umfassende Einführung in Kubernetes: Von der Installation über praktische Implementierung bis hin zu fortgeschrittenen Konzepten der Container-Orchestrierung.

Kategorie: DevOps **Veröffentlicht:** 08.11.2024 **Zuletzt geprüft:** 09.05.2025

Die moderne Softwareentwicklung steht vor der Herausforderung, komplexe Anwendungen effizient zu verwalten und zu skalieren. Kubernetes (oft als K8s abgekürzt) hat sich als leistungsfähige Lösung für diese Anforderungen etabliert. Als Open-Source-Plattform ermöglicht Kubernetes die automatisierte Verwaltung, Bereitstellung und Skalierung von containerisierten Anwendungen.

Mit seinen umfangreichen Funktionen wie automatischer Lastverteilung, Self-Healing und flexibler Ressourcenverwaltung bietet Kubernetes die notwendigen Werkzeuge für den Betrieb moderner Cloud-nativer Anwendungen. Die Plattform wurde ursprünglich von Google entwickelt und 2014 der Open-Source-Community übergeben, wo sie sich seitdem kontinuierlich weiterentwickelt.

> **Wichtiger Hinweis:** Dieser Artikel baut auf den Grundlagen aus unserem Leitfaden DevOps-Grundlagen: Moderne Softwareentwicklung auf. Grundlegende DevOps-Kenntnisse werden vorausgesetzt:
>
> - Grundlegendes Verständnis von Container-Technologien
> - Erfahrung mit Linux-Systemen und der Kommandozeile

Quelle: https://admindocs.de/de/devops/kubernetes-eine-umfassende-dokumentation-zu-container-orchestrierung

- Grundlegendes Verständnis in DevOps-Praktiken

Warum Kubernetes?

In modernen Microservice-Architekturen stehen Entwicklungsteams vor komplexen Herausforderungen:

- Verwaltung hunderter oder tausender Container
- Automatische Skalierung bei Last
- Selbstheilung bei Ausfällen
- Rolling Updates ohne Downtime
- Load Balancing zwischen Services
- Service Discovery in dynamischen Umgebungen

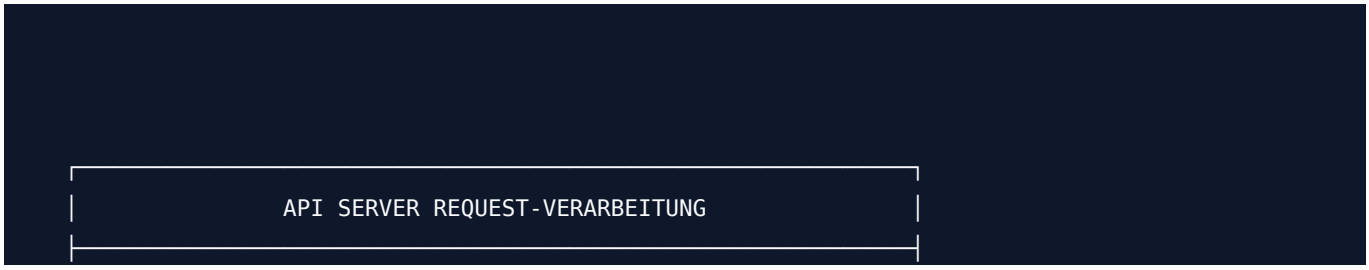
Kubernetes löst diese Herausforderungen durch ein ausgeklügeltes System von Abstraktionen und Automatisierungen.

Architektur und Komponenten

Control Plane (Master Node)

Die Control Plane bildet das „Gehirn" des Kubernetes-Clusters und besteht aus mehreren kritischen Komponenten:

API Server (kube-apiserver):





Der API Server ist der zentrale Kommunikationshub und erfüllt folgende Aufgaben:

- REST-Schnittstelle für alle Cluster-Operationen
- Authentifizierung und Autorisierung von Anfragen
- Validierung von Objektkonfigurationen
- Persistenz in etcd

```
apiVersion: v1
kind: Pod
metadata:
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.1.10
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
```

```
    - --enable-bootstrap-token-auth=true
    image: k8s.gcr.io/kube-apiserver:v1.24.0
    name: kube-apiserver
```

etcd:

Der verteilte Key-Value Store speichert den gesamten Cluster-Status:

- Hochverfügbare Datenspeicherung
- Konsistente Datenhaltung durch Raft-Konsensus
- Versionierte Speicherung aller Kubernetes-Objekte

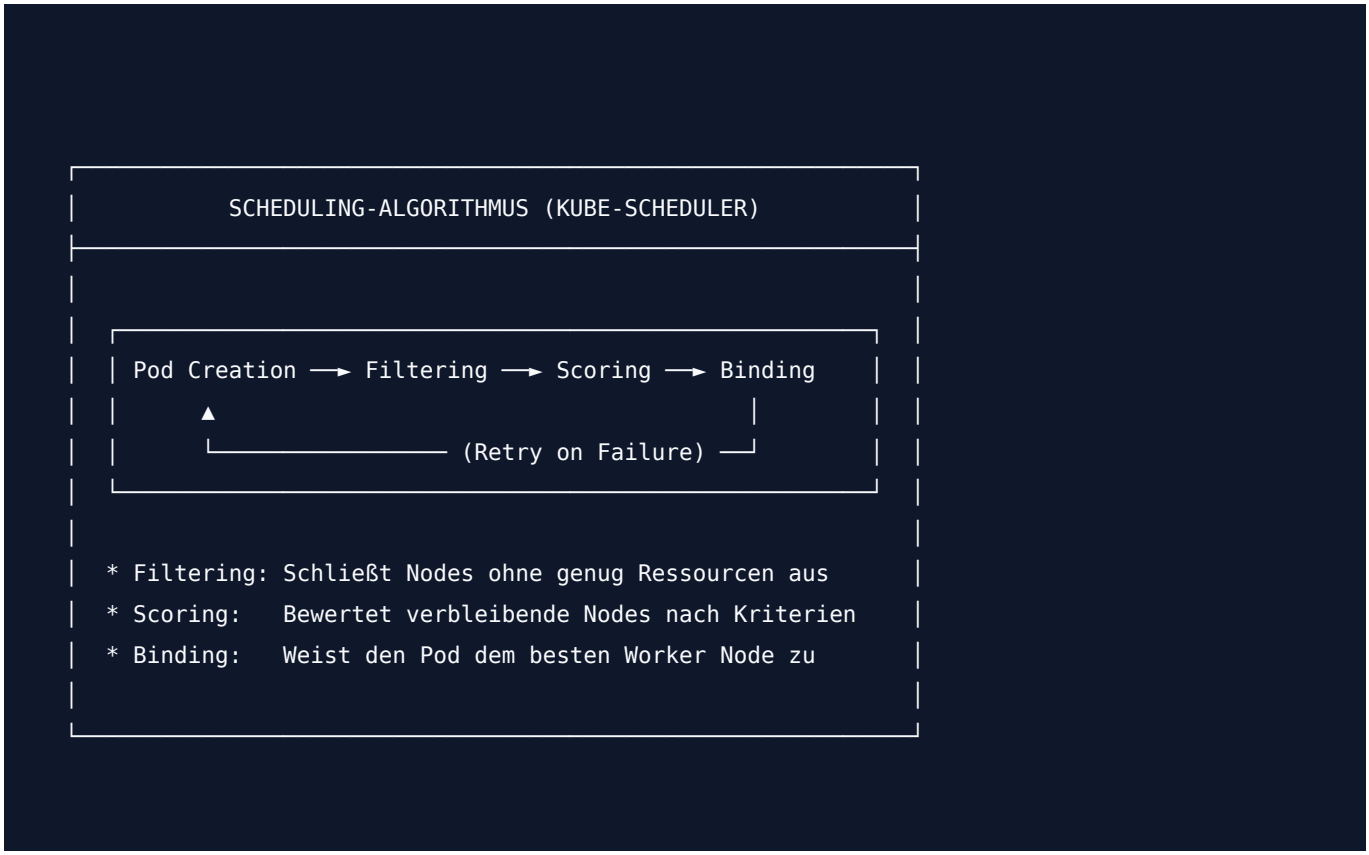
```
apiVersion: v1
kind: Pod
metadata:
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://192.168.1.10:2379
    - --data-dir=/var/lib/etcd
    - --initial-cluster-state=new
    - --initial-cluster-token=etcd-cluster
    image: k8s.gcr.io/etcd:3.5.3-0
    name: etcd
```

Scheduler (kube-scheduler):

Der Scheduler ist verantwortlich für die intelligente Platzierung von Pods:

- Ressourcenanalyse der Worker Nodes
- Berücksichtigung von Affinität/Anti-Affinität

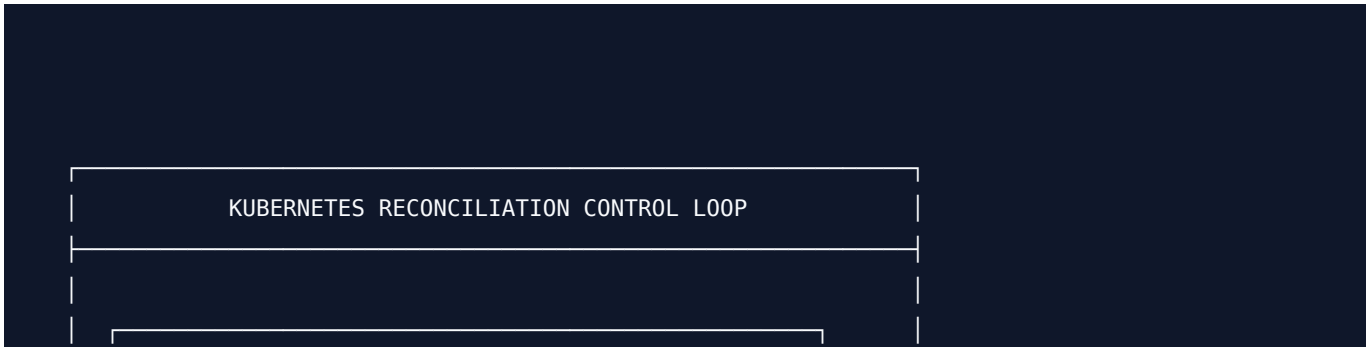
- Priorisierung basierend auf definierten Policies

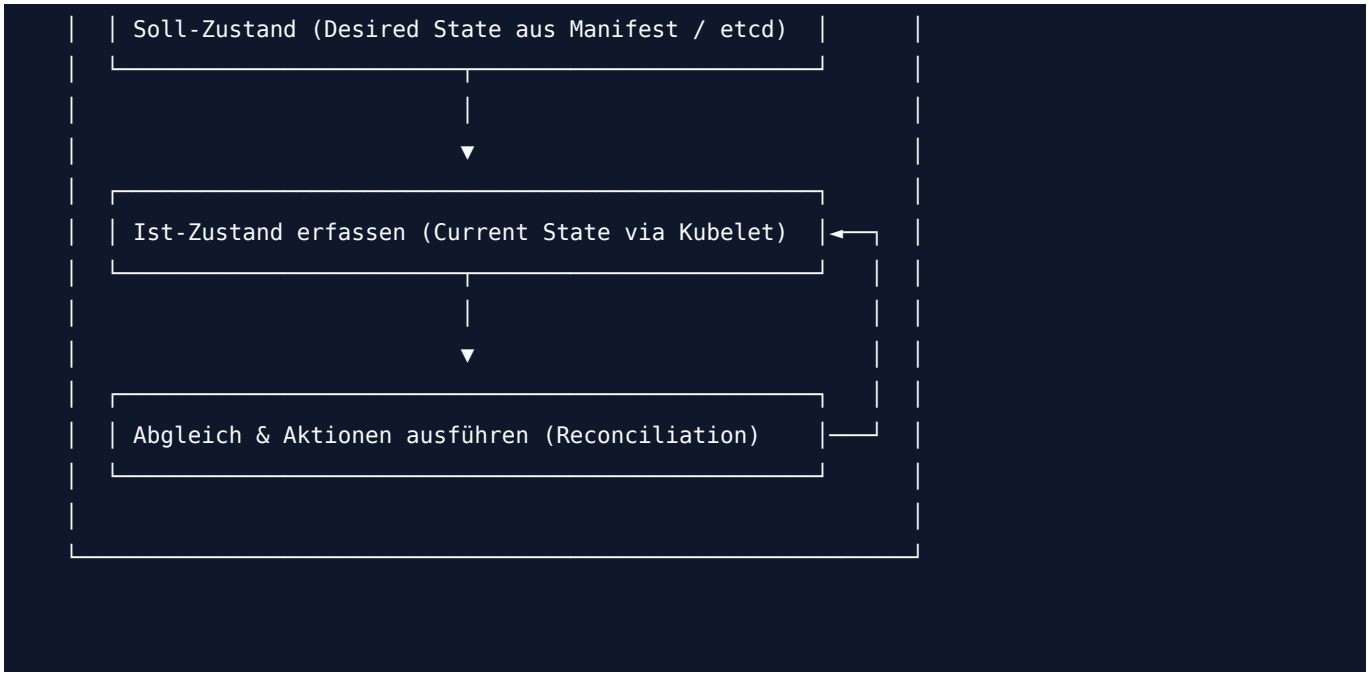


Controller Manager (kube-controller-manager):

Der Controller Manager implementiert die Kernlogik des Kubernetes Control Loops:

- Node Controller: Überwacht den Zustand der Nodes
- Replication Controller: Stellt sicher, dass die gewünschte Anzahl von Pods läuft
- Endpoints Controller: Verknüpft Services mit Pods
- Service Account & Token Controller: Verwaltet Zugriffstoken






```yaml
apiVersion: v1
kind: Pod
metadata:
  name: kube-controller-manager
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-controller-manager
    - --allocate-node-cidrs=true
    - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --bind-address=127.0.0.1
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --cluster-cidr=10.244.0.0/16
    - --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
    image: k8s.gcr.io/kube-controller-manager:v1.24.0
    name: kube-controller-manager
```

Worker Node Komponenten

Kubelet:

Der Kubelet ist der primäre Node-Agent:

- Pod-Lifecycle-Management
- Container Health Checks
- Volume-Mounting
- Node-Status-Reporting

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
address: "0.0.0.0"
port: 10250
serializeImagePulls: true
evictionHard:
  memory.available: "100Mi"
  nodefs.available: "10%"
  nodefs.inodesFree: "5%"
```

Container Runtime:

Die Container Runtime (z.B. containerd) ist verantwortlich für:

- Container-Ausführung
- Image-Management
- Container-Isolation

```
version = 2
[plugins."io.containerd.grpc.v1.cri"]
  sandbox_image = "k8s.gcr.io/pause:3.6"
  [plugins."io.containerd.grpc.v1.cri".containerd]
    default_runtime_name = "runc"
```

```
      [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
        runtime_type = "io.containerd.runc.v2"
```

Kube-proxy:

Kube-proxy implementiert das Kubernetes Service-Konzept:

- Service-Load-Balancing
- IP-Tables-Regeln
- Session Affinity

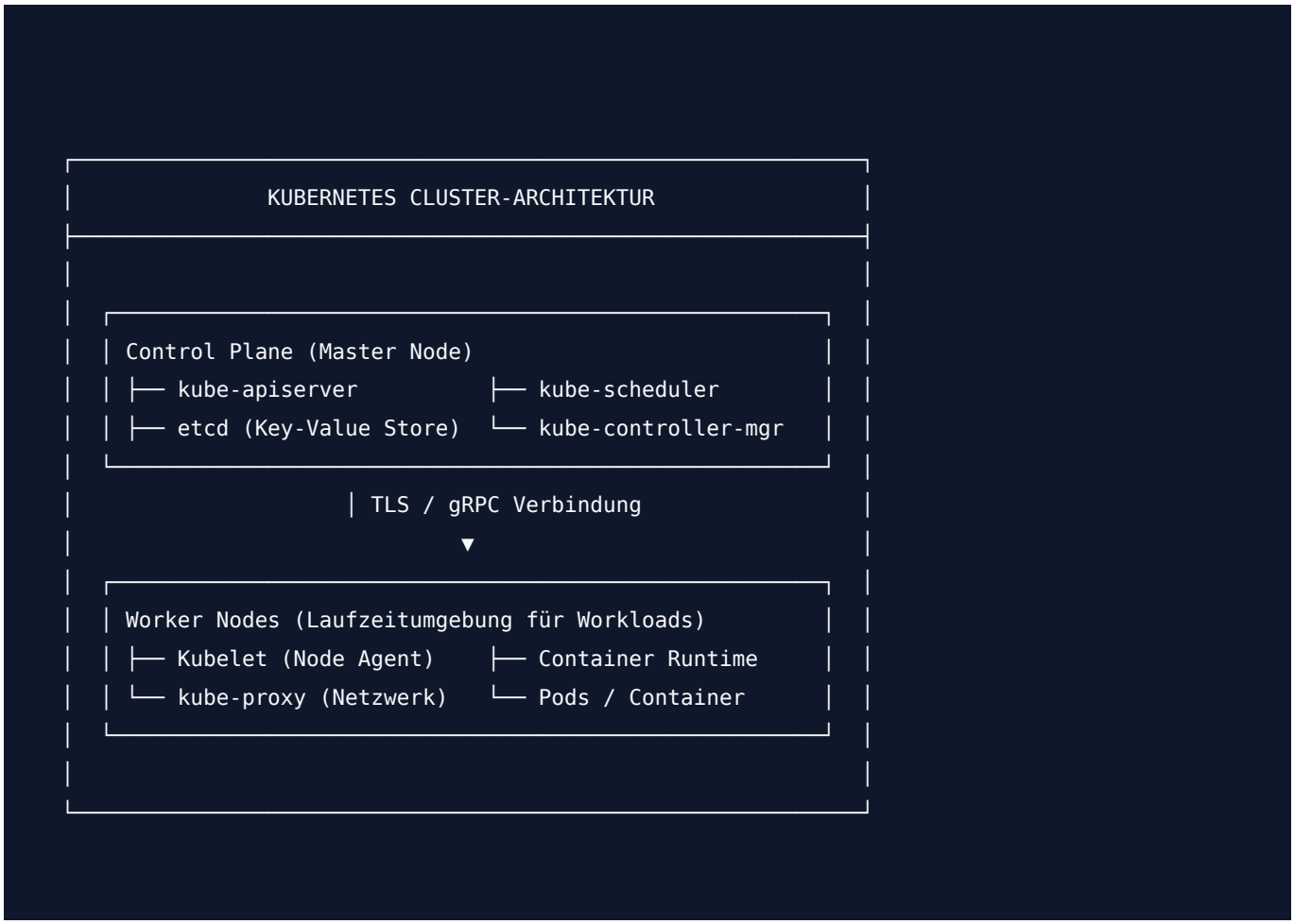
```
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
bindAddress: 0.0.0.0
clientConnection:
  acceptContentTypes: ""
  burst: 10
  contentType: application/vnd.kubernetes.protobuf
  kubeconfig: /var/lib/kube-proxy/kubeconfig.conf
  qps: 5
clusterCIDR: 10.244.0.0/16
mode: "ipvs"
```

Installation und Setup

> **Information:** In diesem Artikel verwenden wir Ubuntu/Debian als Beispiel-Distribution. Die grundlegenden Konzepte sind auf allen Linux-Systemen gleich, aber die Installation von Paketen und einige Konfigurationspfade können sich je nach Distribution unterscheiden.

Voraussetzungen

Bevor wir mit der eigentlichen Installation beginnen, ist es wichtig zu verstehen, welche Anforderungen ein Kubernetes-Cluster an die Hardware und Software stellt.



Hardwareanforderungen:

Control Plane Node (Master):

- **CPU:** Minimum 2 Cores, Empfohlen 4 Cores, Produktiv 8+ Cores
- **RAM:** Minimum 2 GB, Empfohlen 4 GB, Produktiv 16+ GB
- **Festplatte:** Minimum 50 GB, Empfohlen 100 GB, Produktiv 250+ GB (SSD empfohlen)

Worker Nodes:

- **CPU:** Minimum 1 Core, Empfohlen 2 Cores, Produktiv 4+ Cores
- **RAM:** Minimum 1 GB, Empfohlen 2 GB, Produktiv 8+ GB

- **Festplatte:** Minimum 20 GB, Empfohlen 50 GB, Produktiv 100+ GB

> **Wichtige Hinweise zur Planung:**
>
> - Plane die Cluster-Größe entsprechend deiner Workloads
> - Berücksichtige Hochverfügbarkeitsanforderungen
> - Beachte Network-Policy-Anforderungen
> - Plane Storage-Ressourcen sorgfältig

Softwareanforderungen:

Betriebssystem:

```
# Überprüfung der Ubuntu-Version
lsb_release -a

# Mindestanforderungen:
# - Ubuntu 20.04 LTS oder neuer
# - Linux-Kernel 3.10 oder höher
uname -r
```

Systemkonfiguration:

```
# Überprüfung der erforderlichen Kernel-Module
lsmod | grep -e br_netfilter -e overlay

# Falls nicht geladen, Module aktivieren:
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
```

```
overlay
EOF

modprobe br_netfilter
modprobe overlay
```

Netzwerkkonfiguration:

```
# Kernel-Parameter für Kubernetes
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# Parameter aktivieren
sysctl --system

# Überprüfung
sysctl net.bridge.bridge-nf-call-iptables
sysctl net.bridge.bridge-nf-call-ip6tables
sysctl net.ipv4.ip_forward
```

Installation der Grundkomponenten

Systemvorbereitung:

```
# System aktualisieren
sudo apt update
sudo apt upgrade -y
```

```
# Notwendige Basispakete installieren
sudo apt install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release \
    software-properties-common
```

> ⚠ **Wichtig:** Nach dem Systemupdate solltest du prüfen, ob ein Neustart erforderlich ist und stelle sicher, dass swap deaktiviert ist:

```
# Swap deaktivieren
sudo swapoff -a

# Swap-Eintrag aus /etc/fstab entfernen
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab

# Prüfen, ob ein Neustart erforderlich ist
if [ -f /var/run/reboot-required ]; then
    echo 'Neustart erforderlich!'
fi
```

Container Runtime Installation:

Kubernetes benötigt eine Container Runtime. Wir verwenden containerd, da es der aktuelle Standard ist:

```
# Docker Repository hinzufügen
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-
keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

# containerd installieren
sudo apt update
sudo apt install -y containerd.io
```

Konfiguration von containerd:

```
# Standard-Konfiguration erstellen
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml

# SystemdCgroup aktivieren
sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml

# containerd neu starten
sudo systemctl restart containerd
sudo systemctl enable containerd
```

> **□□ Best Practices für containerd:**
>
> - Konfiguriere SystemdCgroup
> - Setze angemessene Resource Limits
> - Aktiviere Log Rotation

Kubernetes-Komponenten Installation

```
# Kubernetes Repository hinzufügen
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o
/usr/share/keyrings/kubernetes-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list

# Kubernetes-Komponenten installieren
sudo apt update
sudo apt install -y kubelet kubeadm kubectl

# Versionen pinnen um ungewollte Updates zu verhindern
sudo apt-mark hold kubelet kubeadm kubectl
```

> ⚠ **Wichtig:** Überprüfe die installierten Versionen:

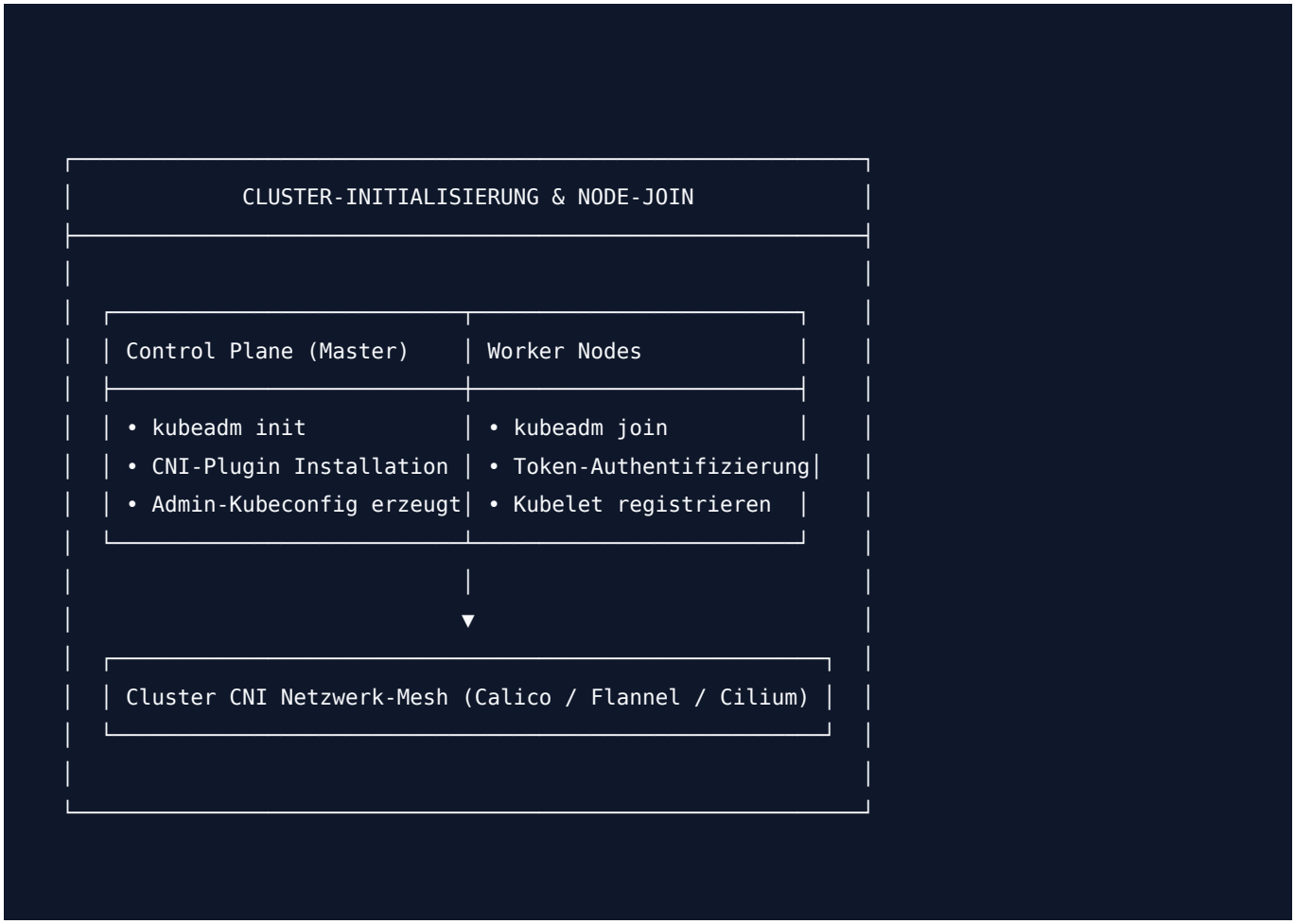
```
kubectl version --client
kubeadm version
kubelet --version
```

> **□□ Typische Fallstricke:**
>
> - Vergessen der Kernel-Module Aktivierung
> - Inkorrekte Netzwerk-Konfiguration

- Unzureichende Ressourcen

Cluster-Initialisierung

Control Plane Setup:



⚠ **Vor der Initialisierung prüfen:**

- Alle Voraussetzungen erfüllt
- Ausreichend Ressourcen verfügbar
- Netzwerk-Ports erreichbar
- Hostname und DNS korrekt konfiguriert

```
# Cluster initialisieren
sudo kubeadm init \
    --pod-network-cidr=192.168.0.0/16 \
    --kubernetes-version=$(kubeadm version -o short) \
    --control-plane-endpoint="$(hostname -I | awk '{print $1}')" \
    --upload-certs

# kubeconfig für den aktuellen Benutzer einrichten
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

> **Wichtige Hinweise zur Cluster-Initialisierung:**
>
> - Notiere dir den `kubeadm join`-Befehl, der am Ende der Initialisierung ausgegeben wird
> - Sichere die Zertifikate und Token-Informationen
> - Der Control Plane Node ist standardmäßig nicht für Workloads vorgesehen

Netzwerk-Plugin Installation

Wir verwenden Calico als CNI-Plugin aufgrund seiner Flexibilität und fortgeschrittenen Netzwerkfunktionen:

```
# Calico Operator installieren
kubectl create -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/tigera-
operator.yaml
```

```
# Calico Custom Resources erstellen
kubectl create -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/custom-
resources.yaml
```

Network Policy Konfiguration:

```
# Beispiel einer grundlegenden Network Policy
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

> ⚠ **Typische Fallstricke bei der Netzwerkkonfiguration:**
>
> - Überlappende Pod-CIDR-Bereiche
> - Fehlende Network Policies
> - Inkorrekte MTU-Einstellungen
> - DNS-Probleme

Überprüfung der Installation:

```
# Cluster-Status überprüfen
kubectl get nodes
kubectl get pods --all-namespaces
kubectl cluster-info

# CNI-Status überprüfen
kubectl get pods -n calico-system
```

Worker Node Integration



Worker Node Vorbereitung:

```
# Systemaktualisierung und Basispaketinstallation
sudo apt update && sudo apt upgrade -y
sudo apt install -y apt-transport-https ca-certificates curl gnupg lsb-release

# Container Runtime (containerd) Installation
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-
keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update
sudo apt install -y containerd.io
```

> **Wichtige Vorbereitungsschritte:**
>
> - Hostname eindeutig setzen
> - Firewall-Regeln anpassen
> - Ressourcen überprüfen
> - SELinux/AppArmor konfigurieren

Worker Node Konfiguration:

```
# containerd konfigurieren
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml
sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml
sudo systemctl restart containerd
sudo systemctl enable containerd
```

```
# Kubernetes-Komponenten installieren
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o
/usr/share/keyrings/kubernetes-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list

sudo apt update
sudo apt install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Worker Node Join:

```
# Token auf dem Control Plane Node generieren
kubeadm token create --print-join-command

# Auf dem Worker Node ausführen
sudo kubeadm join <control-plane-ip>:6443 \
  --token <token> \
  --discovery-token-ca-cert-hash sha256:<hash>
```

> **□□ Wichtige Hinweise:**
>
> - Der Join-Befehl muss mit Root-Rechten ausgeführt werden
> - Token sind standardmäßig 24 Stunden gültig
> - Bei abgelaufenem Token kann ein neuer erstellt werden: `kubeadm token create --print-join-command`

Node Labels und Taints:

```
# Node Labeling für spezifische Workloads
kubectl label node worker-1 workload=production
kubectl label node worker-2 workload=staging

# Taint für dedizierte Workloads
kubectl taint nodes worker-1 dedicated=production:NoSchedule
```

> **Best Practices für Node Management:**
>
> - Implementiere systematische Naming-Konventionen
> - Nutze Labels für Workload-Segregation
> - Plane Node-Kapazitäten sorgfältig
> - Implementiere Node Auto-Scaling

Validierung und Troubleshooting:

```
# Node-Status überprüfen
kubectl get nodes -o wide

# Node-Details anzeigen
kubectl describe node worker-1

# System-Komponenten Status
kubectl get pods -n kube-system
```

Erweiterte Netzwerkkonfiguration

Netzwerk-Policies:

```
# Beispiel einer grundlegenden Network Policy
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Service-Netzwerk Konfiguration:

```
# Überprüfung der Service-CIDR-Konfiguration
kubectl cluster-info dump | grep -m 1 service-cluster-ip-range

# CoreDNS Status überprüfen
kubectl get pods -n kube-system -l k8s-app=kube-dns
```

Load Balancing:

```
# Beispiel eines LoadBalancer Services
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  type: LoadBalancer
```

```
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: example
```

Überprüfung der Installation

```
# Überprüfung der Node-Status
kubectl get nodes -o wide

# Überprüfung der System-Pods
kubectl get pods --all-namespaces

# Cluster-Komponenten Status
kubectl get componentstatuses

# Netzwerk-Funktionalität testen
kubectl run test-pod --image=busybox -- sleep 3600
kubectl exec test-pod -- ping -c 3 8.8.8.8
```

> ⚠ **Wichtig:** Dokumentiere alle durchgeführten Konfigurationsschritte und bewahre wichtige Informationen wie Tokens und Zertifikate sicher auf.

Praktische Implementierung

In diesem Abschnitt lernen wir die praktische Anwendung von Kubernetes kennen.



Deployment-Strategien

Kubernetes bietet verschiedene Strategien für das Deployment von Anwendungen.

> **□□ Best Practices für Deployments:**
>
> - Setze immer Resource Limits
> - Definiere Readiness/Liveness Probes
> - Nutze Labels für bessere Organisation
> - Plane Update-Strategien sorgfältig

Rolling Updates:

Rolling Updates sind die Standardstrategie in Kubernetes und ermöglichen Updates ohne Downtime:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-deployment
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
```

> **Wichtige Hinweise zum Rolling Update:**
>
> - Die Parameter `maxSurge` und `maxUnavailable` bestimmen die Update-Geschwindigkeit
> - Readiness Probes sind kritisch für erfolgreiche Rolling Updates

> - Setze Resource Limits und Requests für vorhersehbares Verhalten

Blue-Green Deployments:

Blue-Green Deployments ermöglichen risikoarme Updates durch parallele Umgebungen:

```
# Blue Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-blue
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      version: blue
  template:
    metadata:
      labels:
        app: myapp
        version: blue
    spec:
      containers:
      - name: myapp
        image: myapp:1.0
---
# Service für Blue-Green Switch
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
    version: blue  # Ändern zu 'green' für Switch
  ports:
  - port: 80
```

```
    targetPort: 8080
```

Canary Deployments:

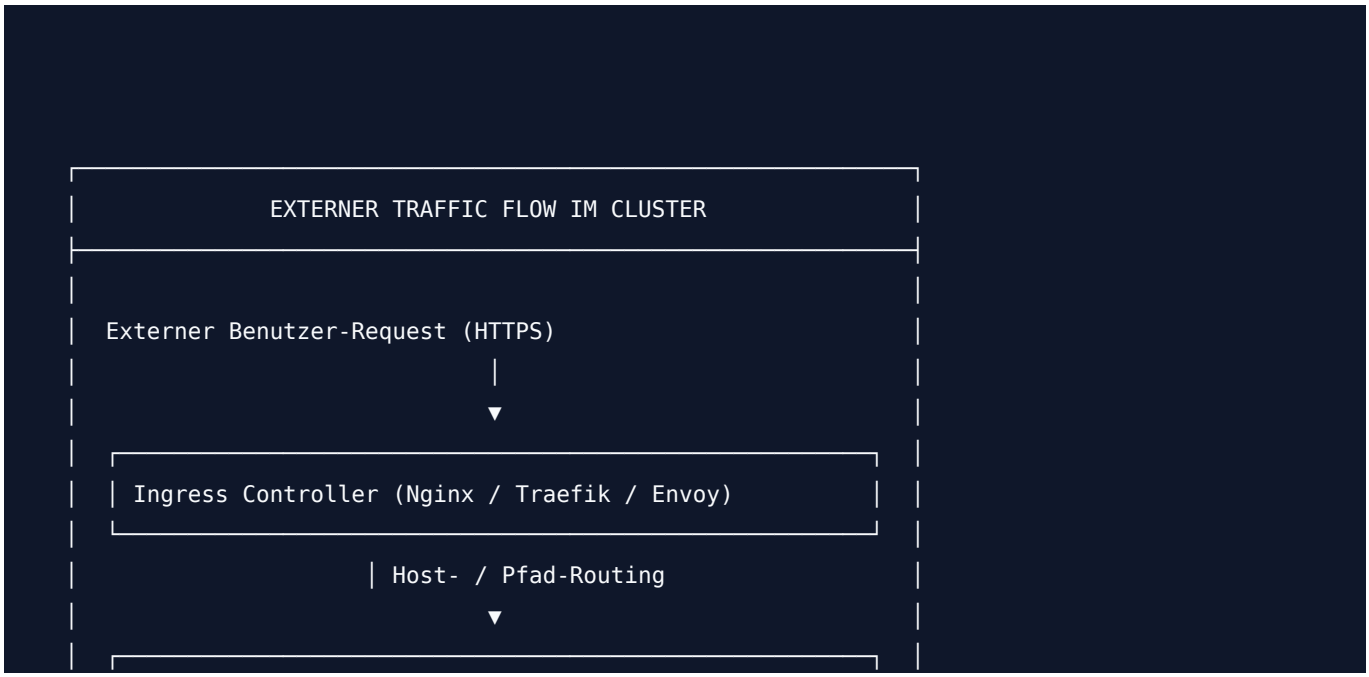
Canary Deployments ermöglichen das schrittweise Testen neuer Versionen mit einem Teil des Traffics:

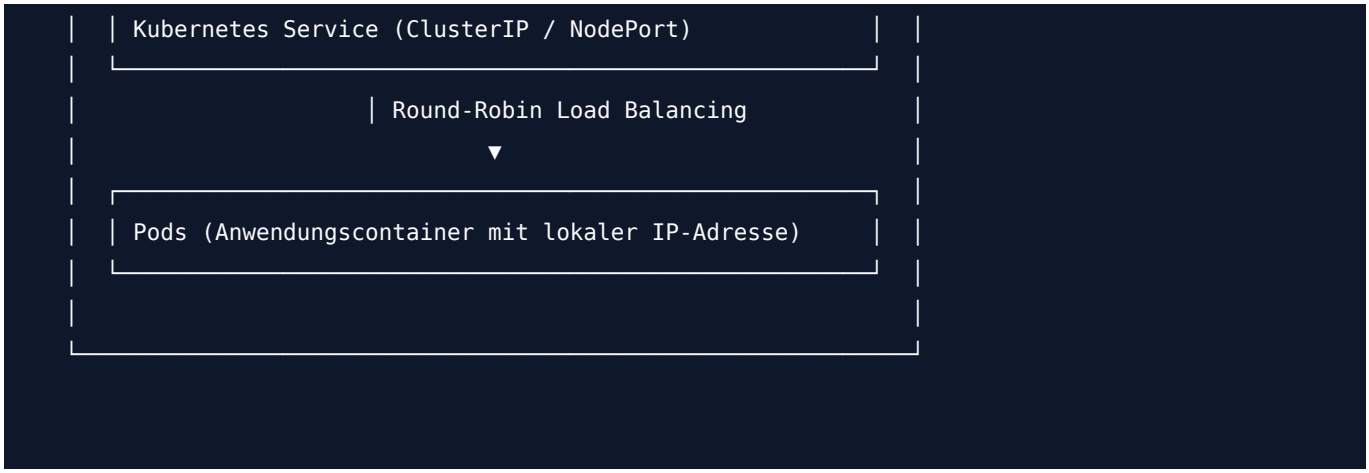
```
# Stable Deployment (90% Traffic)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-stable
spec:
  replicas: 9
  selector:
    matchLabels:
      app: myapp
      version: stable
  template:
    metadata:
      labels:
        app: myapp
        version: stable
    spec:
      containers:
      - name: myapp
        image: myapp:1.0
---
# Canary Deployment (10% Traffic)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-canary
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
      app: myapp
      version: canary
  template:
    metadata:
      labels:
        app: myapp
        version: canary
    spec:
      containers:
      - name: myapp
        image: myapp:2.0
```

> ⚠ **Wichtig für Canary Deployments:**
>
> - Implementiere gründliches Monitoring
> - Plane Rollback-Strategien
> - Nutze Service Mesh für feingranulare Traffic-Kontrolle

Service- und Ingress-Konfiguration





Service Types:

> **Wichtig:** Wähle den Service-Typ basierend auf deinen Anforderungen:
>
> - **ClusterIP:** Intern
> - **NodePort:** Direkt über Node-Ports
> - **LoadBalancer:** Cloud-Provider Integration

```
# ClusterIP (Default)
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
  - port: 80
    targetPort: 8080
---
# NodePort
apiVersion: v1
kind: Service
```

```
  metadata:
    name: frontend-service
  spec:
    type: NodePort
    selector:
      app: frontend
    ports:
    - port: 80
      targetPort: 80
      nodePort: 30080
  ---
  # LoadBalancer
  apiVersion: v1
  kind: Service
  metadata:
    name: public-service
  spec:
    type: LoadBalancer
    selector:
      app: public
    ports:
    - port: 80
      targetPort: 8080
```

Ingress-Konfiguration:

Ingress ermöglicht Layer-7 Routing und SSL-Terminierung:

```
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: example-ingress
    annotations:
      nginx.ingress.kubernetes.io/rewrite-target: /
  spec:
    rules:
    - host: example.com
```

```
    http:
      paths:
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: api-service
            port:
              number: 80
      - path: /
        pathType: Prefix
        backend:
          service:
            name: web-service
            port:
              number: 80
```

> **□□ Best Practices:**
>
> - Konfiguriere SSL/TLS
> - Setze Rate Limiting
> - Implementiere Pfad-basiertes Routing
> - Nutze sinnvolle Annotations

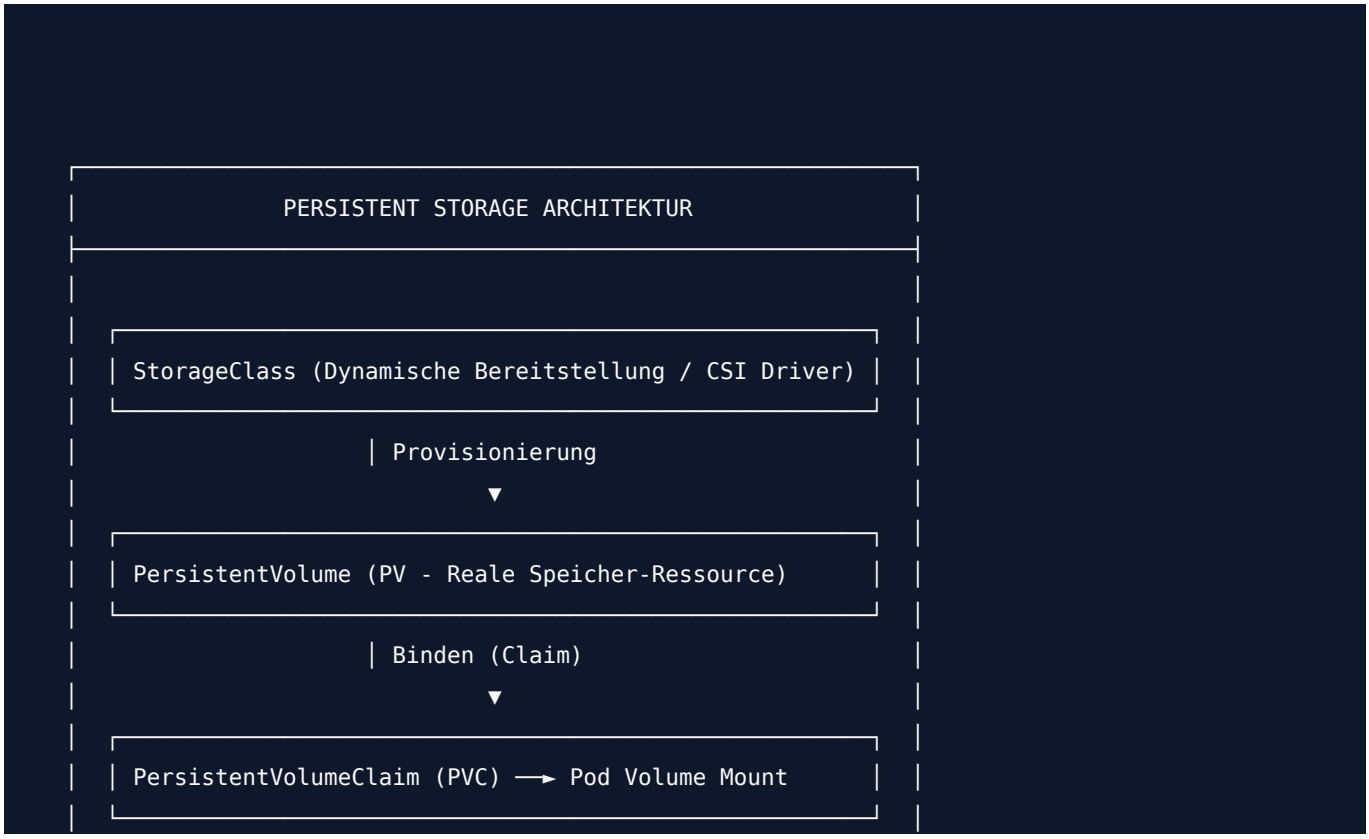
Service Mesh Integration:

```
# Virtual Service (Istio)
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews-route
spec:
```

```
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v1
```

Storage-Management

Persistent Volumes:



```
    |                                                                 |
    └─────────────────────────────────────────────────────────────────┘
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: /mnt/data
---
# Persistent Volume Claim
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: standard
```

Storage Classes:

```
# Definition einer Storage Class für SSD-Storage
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast-storage
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp3
  iopsPerGB: "10"
  encrypted: "true"
reclaimPolicy: Retain
allowVolumeExpansion: true
```

> **□□ Wichtige Hinweise zu Storage Classes:**
>
> - Der Provisioner hängt von deiner Cloud-Plattform oder Storage-Lösung ab
> - Parameter variieren je nach Provisioner
> - Der ReclaimPolicy bestimmt, was mit dem Volume nach dem Löschen des PVC passiert

Volume Snapshots:

```
# Volume Snapshot Class
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: csi-hostpath-snapclass
driver: hostpath.csi.k8s.io
deletionPolicy: Delete
---
```

```
# Volume Snapshot
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: data-snapshot
spec:
  volumeSnapshotClassName: csi-hostpath-snapclass
  source:
    persistentVolumeClaimName: example-pvc
```

Verschlüsselter Storage:

```
# Secret für Verschlüsselungsschlüssel
apiVersion: v1
kind: Secret
metadata:
  name: luks-key
type: Opaque
data:
  key: <base64-encoded-key>
---
# Encrypted PersistentVolume
apiVersion: v1
kind: PersistentVolume
metadata:
  name: encrypted-pv
spec:
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  encryption:
    secretRef:
      name: luks-key
  hostPath:
    path: /mnt/encrypted-data
```

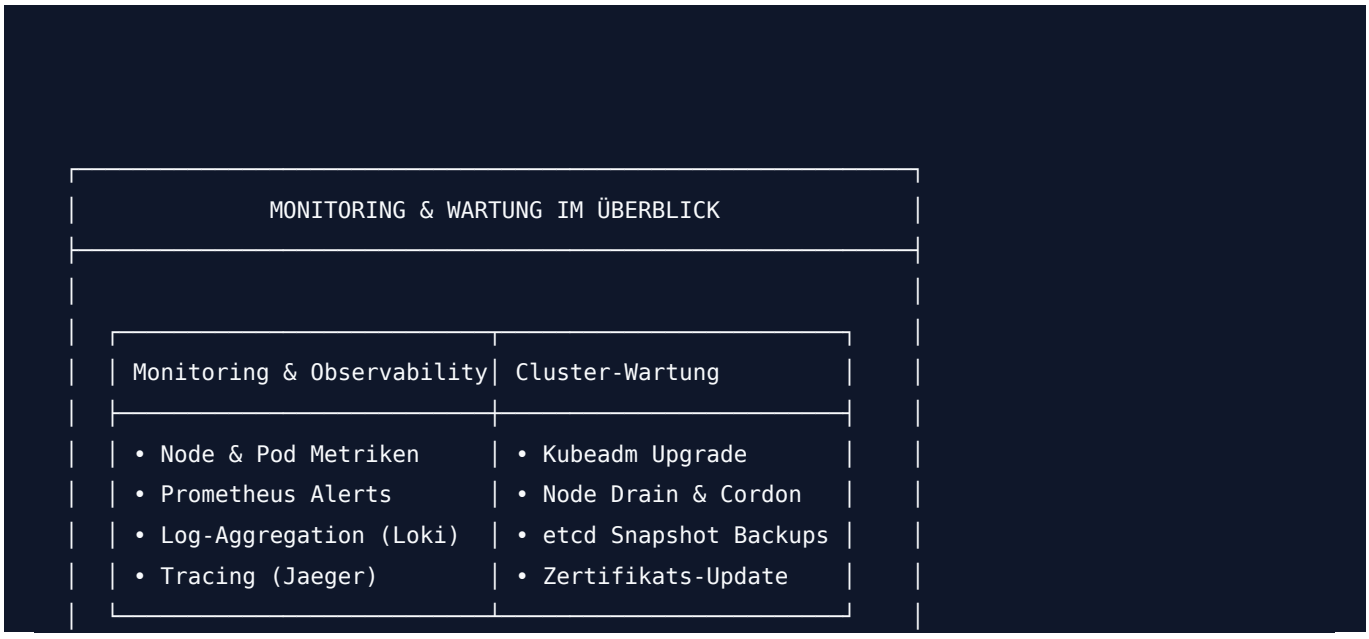
> ⚠ **Wichtige Sicherheitshinweise:**
>
> - Verschlüsselungsschlüssel müssen sicher verwaltet werden
> - Regelmäßige Backup-Strategien implementieren
> - Zugriffsrechte sorgfältig konfigurieren

Best Practices für Storage-Management:

1. **Kapazitätsplanung:** Implementiere proaktives Monitoring, plane Wachstum voraus, setze sinnvolle Quotas.
2. **Backup-Strategien:** Regelmäßige Snapshots, off-site Backups, Disaster Recovery Tests.
3. **Performance-Optimierung:** Wähle passende Storage-Klassen, optimiere Chunk-Größen, implementiere Caching wo sinnvoll.

Monitoring und Wartung

Das Monitoring und die Wartung eines Kubernetes-Clusters sind entscheidend für den stabilen Betrieb.



```
    │                                                         │
    └─────────────────────────────────────────────────────────┘
```

Monitoring-Architektur



Prometheus Installation

```
# Namespace erstellen
apiVersion: v1
kind: Namespace
metadata:
  name: monitoring
```

```
---
# Prometheus Operator Installation
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus-operator
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-operator
  template:
    metadata:
      labels:
        app: prometheus-operator
    spec:
      containers:
      - name: prometheus-operator
        image: quay.io/prometheus-operator/prometheus-operator:v0.59.1
        args:
        - --kubelet-service=kube-system/kubelet
        - --config-reloader-image=jimmidyson/configmap-reload:v0.5.0
```

Wichtige Hinweise zur Prometheus-Installation:

- Stelle sicher, dass genügend Ressourcen verfügbar sind
- Konfiguriere Retention Policies entsprechend deiner Anforderungen
- Plane die Skalierung der Storage-Anforderungen

Monitoring-Konfiguration

Service Monitoring:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: app-monitor
  namespace: monitoring
spec:
  selector:
    matchLabels:
      app: myapp
  endpoints:
  - port: metrics
    interval: 15s
    path: /metrics
```

Alert Rules:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: node-alerts
  namespace: monitoring
spec:
  groups:
  - name: node.rules
    rules:
    - alert: HighCPUUsage
      expr: node_cpu_usage_percentage > 80
      for: 5m
      labels:
        severity: warning
      annotations:
        description: "CPU usage is above 80% for 5 minutes"
```

Alerting und Benachrichtigungen

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: kubernetes-alerts
  namespace: monitoring
spec:
  groups:
  - name: kubernetes.rules
    rules:
    - alert: KubernetesPodCrashLooping
      expr: rate(kube_pod_container_status_restarts_total[15m]) * 60 * 5 > 0
      for: 15m
      labels:
        severity: warning
      annotations:
        description: "Pod {{ $labels.namespace }}/{{ $labels.pod }} is crash looping"
        summary: "Pod is crash looping"
```

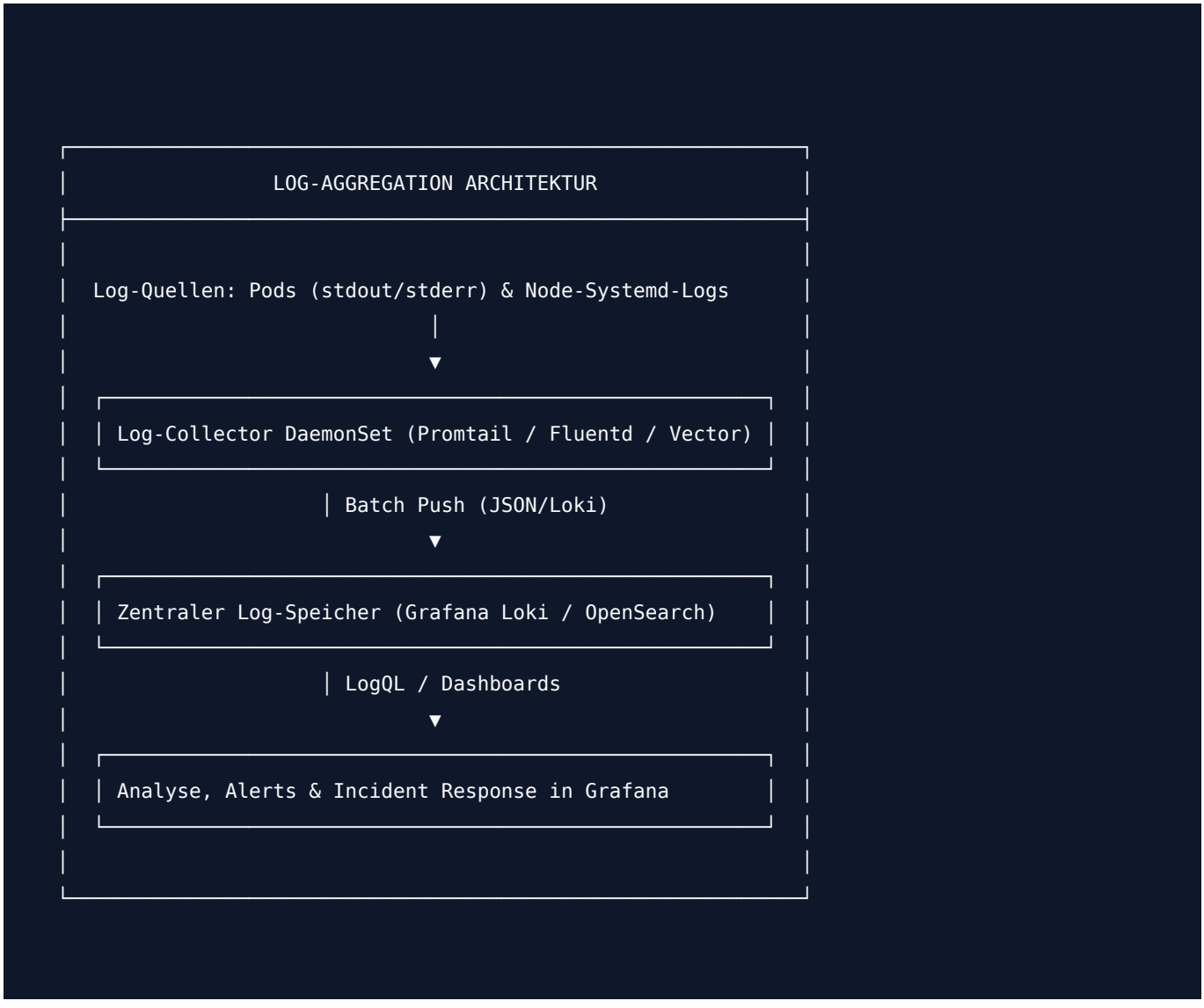
> **Wichtige Hinweise zum Alerting:**
>
> - Definiere sinnvolle Schwellenwerte
> - Vermeide Alert-Fatigue durch zu viele Benachrichtigungen
> - Implementiere Escalation Paths

Resource Quotas Monitoring:

```
apiVersion: v1
kind: ResourceQuota
```

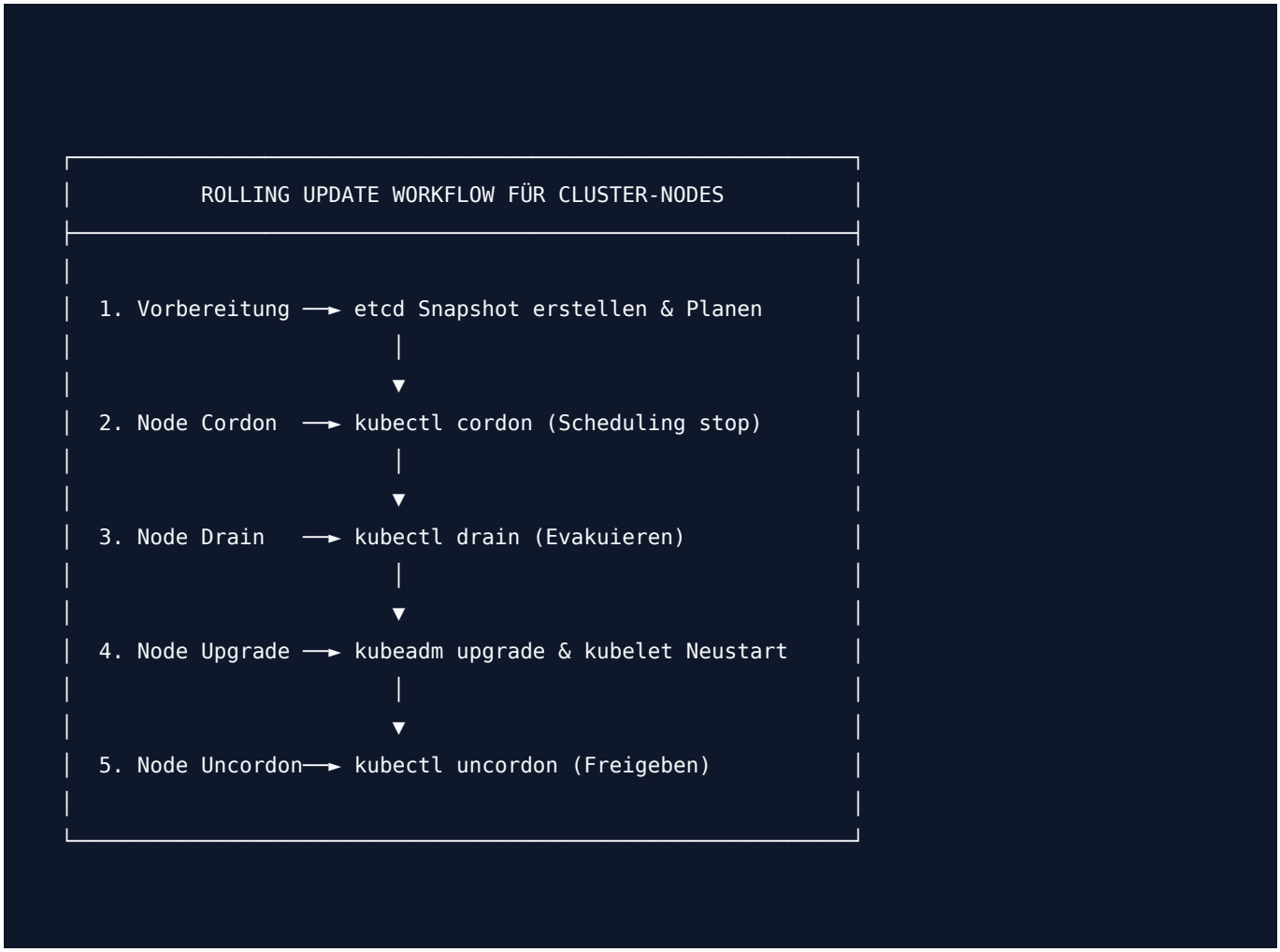
```
  metadata:
    name: compute-resources
  spec:
    hard:
      requests.cpu: "4"
      requests.memory: 8Gi
      limits.cpu: "8"
      limits.memory: 16Gi
```

Log-Aggregation:



Wartungsstrategien

Cluster-Updates:



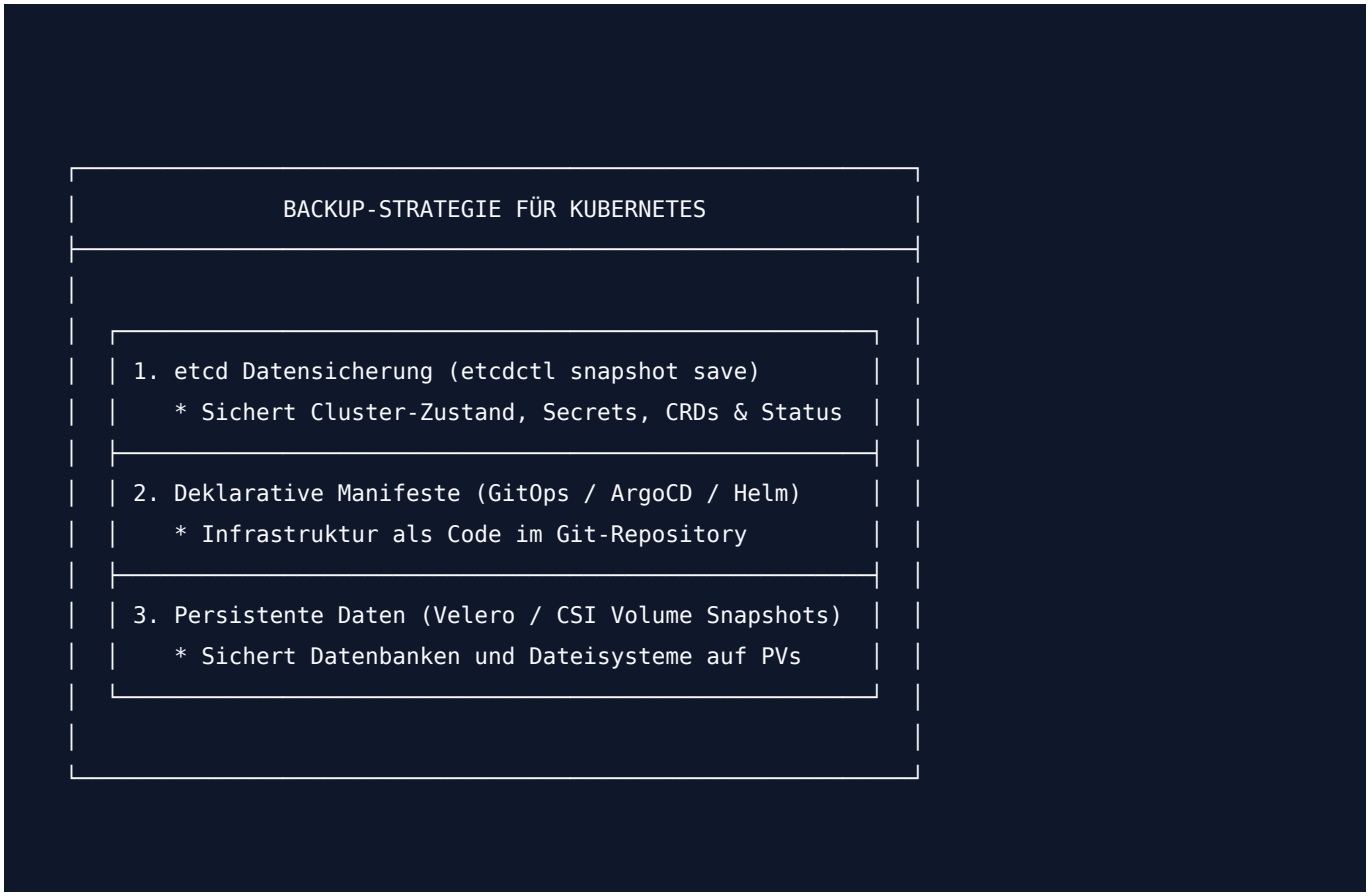
Update-Prozedur für Control Plane:

```
# 1. Backup wichtiger Komponenten
ETCD_BACKUP_DIR="/backup/etcd-$(date +%Y%m%d)"
mkdir -p $ETCD_BACKUP_DIR

# etcd Snapshot erstellen
kubectl exec -n kube-system etcd-master-1 -- etcdctl snapshot save \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  $ETCD_BACKUP_DIR/etcd-snapshot.db
```

> ⚠ **Wichtige Hinweise zum Update-Prozess:**
>
> - Führe Updates immer zuerst in einer Testumgebung durch
> - Plane ausreichende Wartungsfenster ein
> - Halte Rollback-Prozeduren bereit
> - Dokumentiere jeden Schritt sorgfältig

Backup und Recovery




```
#!/bin/bash
# Backup-Script für kritische Cluster-Komponenten
```

```
BACKUP_DIR="/backup/k8s-$(date +%Y%m%d)"
RETENTION_DAYS=30

mkdir -p "${BACKUP_DIR}"/{etcd,manifests,secrets}

# etcd Backup
etcdctl snapshot save "${BACKUP_DIR}/etcd/snapshot.db"

# Kubernetes Manifeste sichern
kubectl get all --all-namespaces -o yaml > \
  "${BACKUP_DIR}/manifests/all-resources.yaml"

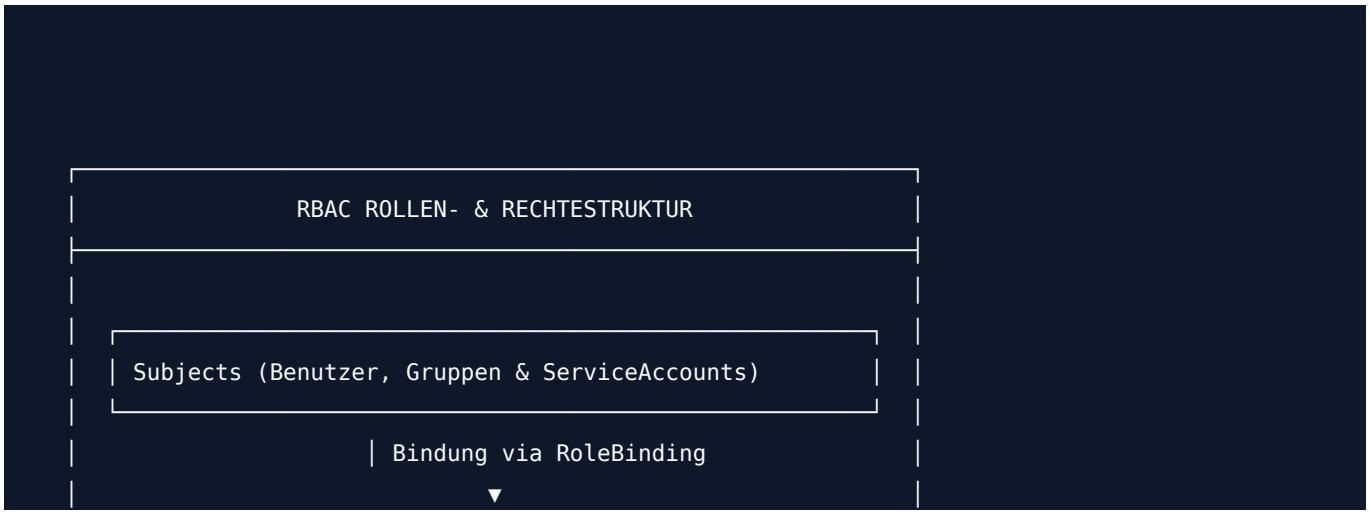
# Secrets sichern (verschlüsselt)
kubectl get secrets --all-namespaces -o yaml | \
  gpg --encrypt > "${BACKUP_DIR}/secrets/all-secrets.yaml.gpg"

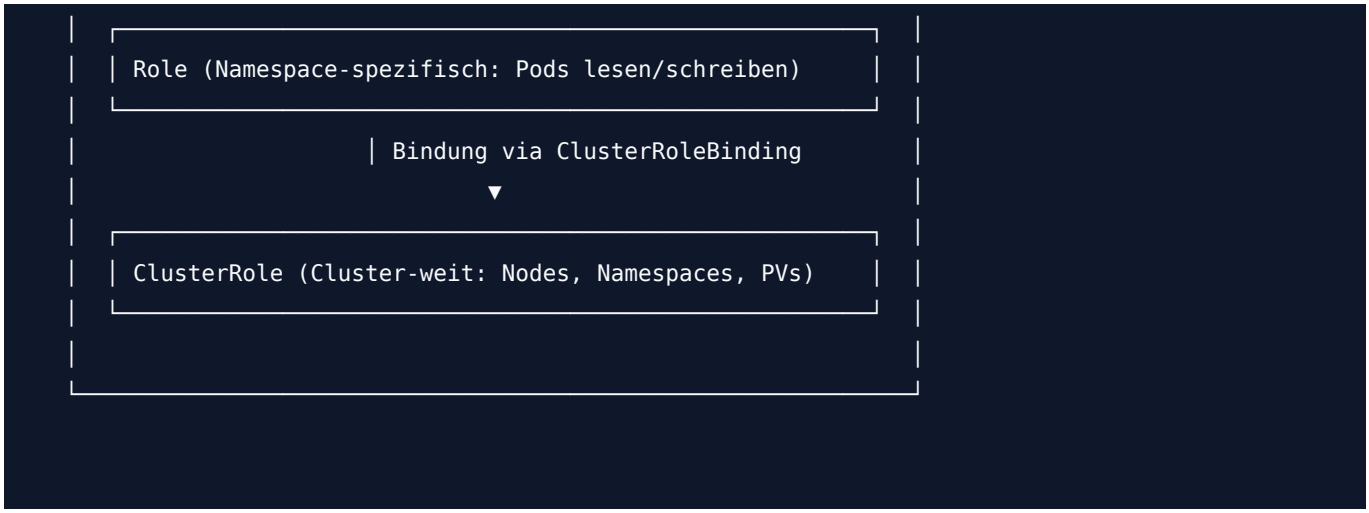
# Alte Backups bereinigen
find /backup -type d -mtime +${RETENTION_DAYS} -exec rm -rf {} +
```

Sicherheit und Best Practices

Die Sicherheit eines Kubernetes-Clusters ist von entscheidender Bedeutung.

RBAC (Role-Based Access Control)






```yaml
# Namespace-spezifische Rolle
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: development
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
---
# Cluster-weite Rolle
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-reader-global
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

> **□□ Wichtige RBAC-Prinzipien:**

- Principle of Least Privilege
- Regelmäßige Audit-Reviews
- Dokumentation aller Rollen und Berechtigungen

Pod Security Policies

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
  runAsUser:
    rule: MustRunAsNonRoot
  fsGroup:
    rule: RunAsAny
  volumes:
  - 'configMap'
  - 'emptyDir'
  - 'persistentVolumeClaim'
```

Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-allow
```

```
spec:
  podSelector:
    matchLabels:
      app: api
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 8080
```

Secrets Management

```
# Secret Definition
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
type: Opaque
data:
  db-password: BASE64_ENCODED_PASSWORD
  api-key: BASE64_ENCODED_API_KEY
```

> **□□ Best Practices für Secrets:**
>
> - Verschlüsselung im Ruhezustand
> - Regelmäßige Rotation
> - Zugriffsbeschränkung

- Audit-Logging

Container Security



Cluster Hardening

```
# Pod Security Policy
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCapabilities:
```

```
    - ALL
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    rule: 'MustRunAsNonRoot'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'MustRunAs'
    ranges:
      - min: 1
        max: 65535
  fsGroup:
    rule: 'MustRunAs'
    ranges:
      - min: 1
        max: 65535
```

Audit Logging

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: RequestResponse
  resources:
  - group: ""
    resources: ["pods", "services"]
- level: Metadata
  resources:
```

```
    - group: ""
      resources: ["configmaps"]
  - level: None
    users: ["system:kube-proxy"]
    resources:
    - group: ""
      resources: ["endpoints"]
```

Automatisierte Sicherheitsprüfungen

```
# Kube-bench Job für automatisierte CIS-Checks
apiVersion: batch/v1
kind: Job
metadata:
  name: kube-bench
spec:
  template:
    spec:
      hostPID: true
      containers:
      - name: kube-bench
        image: aquasec/kube-bench:latest
        command: ["kube-bench", "--benchmark", "cis-1.6"]
        volumeMounts:
        - name: var-lib-kubelet
          mountPath: /var/lib/kubelet
        - name: etc-systemd
          mountPath: /etc/systemd
        - name: etc-kubernetes
          mountPath: /etc/kubernetes
      restartPolicy: Never
      volumes:
      - name: var-lib-kubelet
        hostPath:
          path: "/var/lib/kubelet"
      - name: etc-systemd
```

```
          hostPath:
            path: "/etc/systemd"
        - name: etc-kubernetes
          hostPath:
            path: "/etc/kubernetes"
```

Fortgeschrittene Konzepte

Custom Resources

Custom Resources (CRs) erweitern die Kubernetes API um benutzerdefinierte Ressourcentypen.

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: webapps.mycompany.com
spec:
  group: mycompany.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                image:
                  type: string
                replicas:
                  type: integer
                  minimum: 1
  scope: Namespaced
```

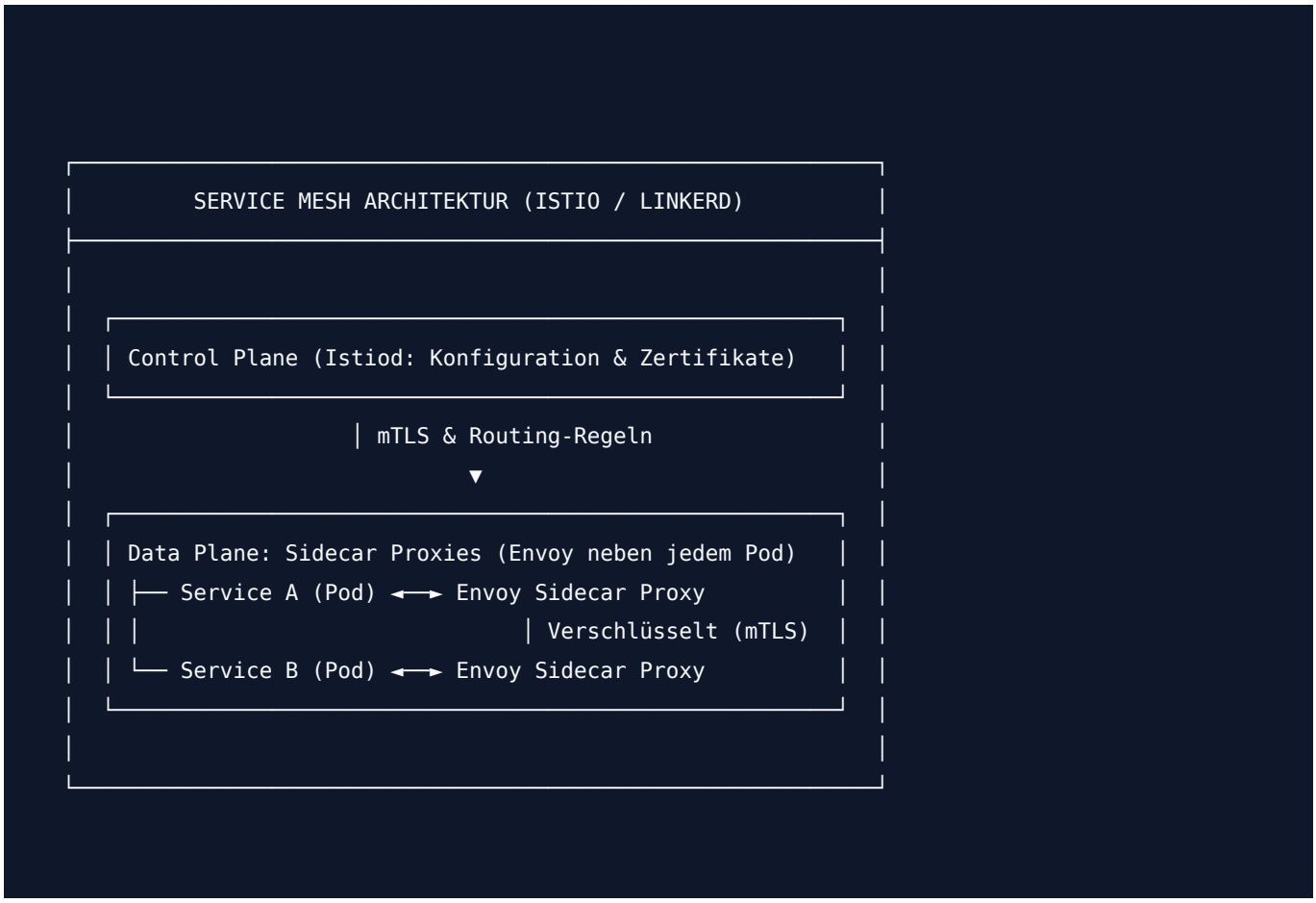
```
  names:
    plural: webapps
    singular: webapp
    kind: WebApp
    shortNames:
    - wa
```

Operators

Operators automatisieren die Verwaltung komplexer Anwendungen in Kubernetes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: mysql-operator
  template:
    metadata:
      labels:
        name: mysql-operator
    spec:
      containers:
        - name: operator
          image: mysql-operator:v1.0
          ports:
          - containerPort: 8080
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
```

Service Mesh



Übung

In dieser Übung werden wir eine Webanwendung mit allen wichtigen Kubernetes-Konzepten deployen.

Übungsziele

- Erstellen eines Deployments mit mehreren Replicas
- Konfiguration eines Services und Ingress
- Implementierung von Monitoring
- Anwendung von Security Policies

Schritt 1: Deployment erstellen

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
```

Schritt 2: Service konfigurieren

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: webapp-service
spec:
  selector:
    app: webapp
  ports:
  - port: 80
    targetPort: 80
  type: ClusterIP
```

Schritt 3: Ingress einrichten

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
spec:
  rules:
  - host: webapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: webapp-service
            port:
              number: 80
```

Schritt 4: Monitoring aktivieren

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: webapp-monitor
spec:
  selector:
    matchLabels:
      app: webapp
  endpoints:
  - port: metrics
```

Schritt 5: Security Policy anwenden

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: webapp-policy
spec:
  podSelector:
    matchLabels:
      app: webapp
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: allowed
```

Überprüfung der Übung

```
# Überprüfe Deployment-Status
kubectl get deployments
kubectl get pods

# Überprüfe Service und Ingress
kubectl get services
kubectl get ingress

# Teste den Zugriff
curl -H "Host: webapp.example.com" http://CLUSTER-IP
```

Befehlsreferenz (Cheatsheet)

Für den schnellen Zugriff im Betriebsalltag fasst die folgende Referenztabelle die wichtigsten Kubernetes-Kommandos zur Cluster-Verwaltung, Fehlerdiagnose und Ressourcenskalierung zusammen:

Befehl / Syntax	Beschreibung & Praxiseinsatz
`kubectl get nodes -o wide`	Listet alle Cluster-Nodes inklusive Status, Version, Kernel und interner IP auf.
`kubectl get pods -A`	Zeigt alle Pods über alle Namespaces hinweg samt Status und Neustart-Zähler an.
`kubectl describe pod <pod-name>`	Detaillierte Fehlerdiagnose mit Events, Container-Status und Exit-Codes.
`kubectl logs -f <pod-name> [-c <container>]`	Streamt Live-Logs eines Pods oder eines spezifischen Sidecar-Containers.

Befehl / Syntax	Beschreibung & Praxiseinsatz
`kubectl exec -it <pod-name> -- /bin/sh`	Öffnet eine interaktive Shell im Container zur Live-Inspektion.
`kubectl apply -f <manifest.yaml>`	Deklaratives Ausrollen oder Aktualisieren von Kubernetes-Ressourcen.
`kubectl rollout status deployment/<name>`	Überwacht den Fortschritt eines Rolling Updates in Echtzeit.
`kubectl rollout undo deployment/<name>`	Führt einen sofortigen Rollback auf die vorherige Revision durch.
`kubectl get services,ingress`	Zeigt aktive Services, ClusterIPs, NodePorts und Ingress-Routen an.
`kubectl get pv,pvc`	Übersicht über PersistentVolumes und deren Bindungsstatus an Claims.
`kubectl top nodes && kubectl top pods`	Zeigt aktuellen CPU- und Speichernutzungsgrad von Nodes und Pods an.
`kubectl cordon <node> && kubectl drain <node>`	Sperrt einen Node für neue Pods und evakuiert bestehende Workloads sicher.
`kubectl uncordon <node>`	Gibt einen gewarteten Worker Node wieder regulär für das Scheduling frei.
`kubeadm init --pod-network-cidr=192.168.0.0/16`	Initialisiert die Kubernetes Control Plane mit CNI-Netzwerkbereich.
`kubeadm join <master>:6443 --token <t> --discovery-token-ca-cert-hash <h>`	Verbindet einen Worker Node mit dem bestehenden Cluster.
`kubeadm upgrade plan`	Prüft verfügbare Versionen und zeigt Upgrade-Schritte für die Control Plane.
`etcdctl snapshot save snapshot.db`	Erstellt einen konsistenten Point-in-Time-Snapshot des etcd-Zustands.

Weiterführende Ressourcen

Die folgenden offiziellen Dokumentationen, Guides und Community-Projekte vertiefen die im Artikel behandelten Konzepte und Best Practices:

Ressource	Beschreibung
Kubernetes Dokumentation	Offizielle und vollständige Dokumentation aller K8s-Komponenten und APIs.
Kubernetes Tutorials	Praxisorientierte Anleitungen für Deployments, Services und Skalierung.
Kubernetes Blog	Aktuelle Ankündigungen, Release-Notes und tiefgehende Architektur-Beiträge.
Interactive Basics Tutorial	Interaktive Einführung in Cluster-Erstellung und Workload-Management.
Minikube Getting Started	Offizieller Leitfaden zum lokalen Betrieb von Single-Node-Clustern.
Kubernetes Training Labs	CNCF-Schulungsmaterialien und interaktive Übungsumgebungen.
Kubernetes GitHub Repository	Quellcode, Issue-Tracker und technische Design-Proposals (KEPs).
Kubernetes Community	Foren, Special Interest Groups (SIGs) und Slack-Channels der Community.
CNCF Kubernetes Project	Governance, Ökosystem-Übersicht und Graduierungsstatus innerhalb der CNCF.
kubectl Cheat Sheet	Offizielle Referenz aller `kubectl`-Optionen und JSONPath-Abfragen.
Helm Paketmanager	Paketverwaltung für Kubernetes zur Templatierung und Verteilung von Apps.
Lens Kubernetes IDE	Leistungsstarke grafische Benutzeroberfläche zur Cluster-Verwaltung.

> **Hinweis:** Die Kubernetes-Landschaft entwickelt sich ständig weiter. Stelle sicher, dass du immer die aktuellste Dokumentation konsultierst.

Fazit

Kubernetes hat sich als De-facto-Standard für Container-Orchestrierung etabliert und bietet eine robuste Plattform für moderne Cloud-native Anwendungen. Die in diesem Artikel behandelten

Konzepte und Praktiken zeigen die Vielseitigkeit und Leistungsfähigkeit von Kubernetes. Von der grundlegenden Installation über die praktische Implementierung bis hin zu fortgeschrittenen Konzepten wie Custom Resources und Service Mesh – Kubernetes bietet die notwendigen Werkzeuge für den Betrieb komplexer Containerumgebungen.

Die kontinuierliche Weiterentwicklung von Kubernetes, insbesondere im Bereich der Automatisierung und Sicherheit, macht die Plattform zukunftssicher. Mit der steigenden Adoption von Cloud-nativen Technologien und der zunehmenden Bedeutung von DevOps-Praktiken wird Kubernetes weiterhin eine zentrale Rolle in der modernen IT-Infrastruktur spielen. Neue Entwicklungen wie GitOps, KubeVirt für VM-Management und die verstärkte Integration von KI-gestützten Operationen werden die Plattform in Zukunft noch leistungsfähiger machen.

Zur Übersicht: Alle DevOps Artikel & Guides

Quelle: https://admindocs.de/de/devops/kubernetes-eine-umfassende-dokumentation-zu-container-orchestrierung