

[zurück](#)

Kubernetes - Grundlagen (Pods, Nodes, Deployments, Services, Ingress)

Kubernetes (oft „K8s“ genannt) ist ein System zur Orchestrierung von Containern. Es verwaltet automatisch:

- Deployment (Bereitstellung) von Containern
- Skalierung (mehr Container starten)
- Self-Healing (Neustart bei Fehlern)
- Updates ohne Ausfall
- Netzwerk zwischen Containern
- Storage für Container

Kubernetes ist der Standard in modernen Cloud-, DevOps- und CI/CD-Umgebungen.

1. Warum Kubernetes?

Docker allein startet Container - Kubernetes betreibt ganze Systeme.

Vorteile:

- automatische Skalierung
- Ausfallsicherheit
- Rollout / Rollback von Updates
- Lastverteilung
- Cluster über viele Server hinweg
- Self-Healing: Container werden automatisch ersetzt

Perfekt für:

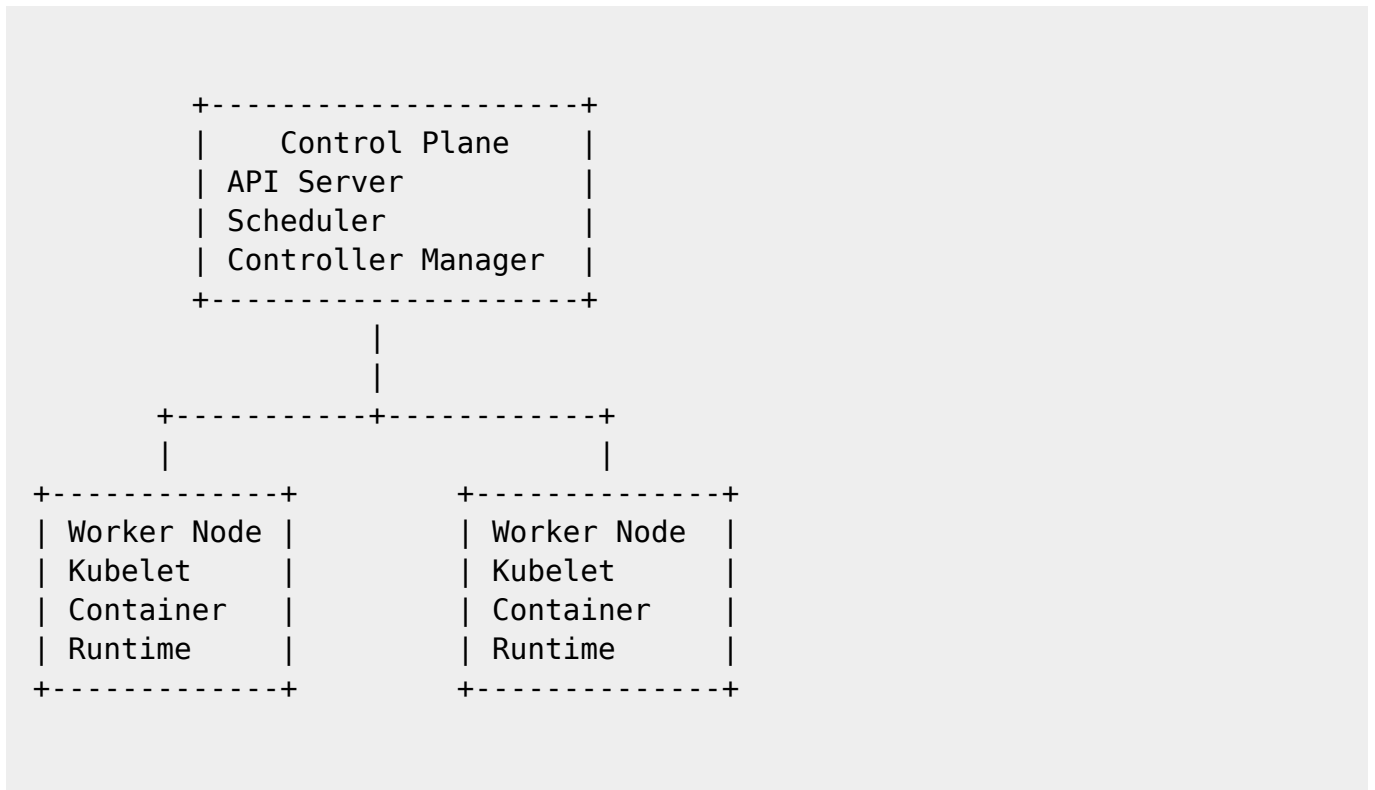
- große Webplattformen
 - Microservices
 - APIs
 - Clouddienste
 - Enterprise-Infrastruktur
-

2. Kubernetes Architektur - Überblick

Kubernetes besteht aus zwei Bereichen:

- **Control Plane** – steuert das Cluster
- **Worker Nodes** – führen Container aus

ASCII-Übersicht:



3. Die wichtigsten Kubernetes-Objekte

Kubernetes arbeitet mit sogenannten *Ressourcen* oder *Objekten*.

Die wichtigsten:

- **Pod**
- **Deployment**
- **Service**
- **Ingress**
- **ConfigMap**
- **Secret**
- **PersistentVolume (PV)**
- **PersistentVolumeClaim (PVC)**

4. Pod

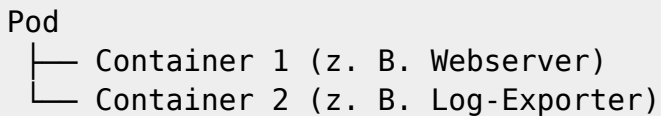
Der **Pod** ist die kleinste Einheit in Kubernetes.

Ein Pod enthält:

- einen oder mehrere Container
- gemeinsame IP-Adresse
- gemeinsames Volume

Die Container in einem Pod sind eng gekoppelt.

Schema:



Pods sind **flüchtig** - sie werden ständig ersetzt.
Deshalb nutzt man **Deployments**, nicht einzelne Pods.

5. Deployment

Ein Deployment steuert, wie viele Pods laufen und wie sie aktualisiert werden.

Funktionen:

- Skalieren (mehr/weniger Pods)
- automatische Neustarts
- Rolling Updates (ohne Downtime)
- Rollbacks bei Fehlern

Beispiel YAML:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
```

```
template:  
  spec:  
    containers:  
      - name: app  
        image: nginx
```

Dieses Deployment startet **3 Pods** mit Nginx.

6. Service

Pods haben dynamische IPs – ein Service sorgt für **stabile Erreichbarkeit**.

Arten:

- **ClusterIP** → nur intern erreichbar
- **NodePort** → von außen über hohen Port erreichbar
- **LoadBalancer** → Cloud-Loadbalancer automatisch
- **ExternalName** → Alias zu externem DNS-Namen

Schema:

```
Clients → Service → verteilt Traffic → mehrere Pods
```

Service = Kubernetes Loadbalancer.

7. Ingress

Ingress ist ein **Reverse Proxy / Loadbalancer auf Layer 7** für HTTP/HTTPS.

Beispiel:

- <https://api.example.com> → Backend 1
- <https://app.example.com> → Backend 2

Ingress-Controller (du kennst das ☐):

- Traefik

- Nginx Ingress
- HAProxy
- Istio Gateway

Schema:

```
Client → Ingress → Service → Pods
```

Ingress ermöglicht:

- Routing nach Hostname/URL
- TLS-Zertifikate
- Load Balancing
- Middlewares

8. ConfigMaps & Secrets

ConfigMap

Konfigurationen (Text).

```
env: APP_MODE=production
```

Secret

Passwörter / Zertifikate (Base64-kodiert, nicht verschlüsselt!)

```
env: DB_PASSWORD=*****
```

9. Persistenter Speicher (Storage)

Pods sind flüchtig → Daten würden verloren gehen.
Daher nutzt man PV und PVC.

PersistentVolume (PV)

das eigentliche Storage-Backend
z. B. NFS, iSCSI, Ceph, lokal

PersistentVolumeClaim (PVC)

Anfrage eines Pods nach Speicher

Schema:

```
PVC (Pod) → PV → Storage (NFS/SSD/Ceph)
```

10. Kubernetes Netzwerk

Jeder Pod bekommt:

- eigene IP (intern)
- kann mit allen Pods kommunizieren (by default)

CNI-Plugins regeln das Netzwerk:

- Calico
- Flannel
- Cilium (besser, modern)
- Weave

11. Skalierung

Kubernetes kann automatisch skalieren:

Horizontal Pod Autoscaler (HPA)

z. B. starte 10 zusätzliche Pods wenn CPU > 70%

Vertical Autoscaler

passt CPU/RAM an

Cluster Autoscaler

startet neue Nodes in der Cloud

12. Self-Healing

Kubernetes überwacht seine Pods:

Wenn:

- Pod crasht
- Node ausfällt
- Container hängt

Dann:

- Pod wird automatisch ersetzt
 - Deployment sorgt für korrekte Anzahl
 - Load Balancer leitet Traffic auf gesunde Pods
-

13. Kubernetes vs Docker Compose

| Funktion | Docker Compose | Kubernetes |
|------------|----------------|------------------|
| Deployment | einfach | komplex, mächtig |
| Skalierung | manuell | automatisch |

| Funktion | Docker Compose | Kubernetes |
|-----------------|-----------------------|-----------------------------|
| Self-Healing | nein | ja |
| Updates | manuell | rolling updates |
| Netzwerk | einfach | Cluster-weite Kommunikation |
| Betrieb | Einzelserver | mehrere Nodes |

Kurz: **Compose = kleine Projekte** **Kubernetes = Großprojekte / Enterprise**

14. Beispiele aus der Praxis

Microservices

Viele kleine Dienste:

- Auth-Service
- Payment-Service
- User-Service

Cloud-Apps

Kubernetes ist Grundlage von:

- Google Cloud
- Azure AKS
- AWS EKS

Home-Lab

Mini-Kubernetes mit:

- k3s
- MicroK8s
- kind

Zusammenfassung

- Kubernetes orchestriert Container automatisiert
 - Pod = kleinste Einheit

- * Deployment = steuert Pods, Updates, Skalierung
- * Service = Load Balancer für Pods
- * Ingress = HTTP/HTTPS Reverse Proxy
- * PV/PVC = persistenter Speicher
- * Kubernetes = Standard für moderne Cloud-Anwendungen

From:

<http://wiki.nctl.de/dokuwiki/> - ☐ Veni. Vidi. sudo rm -rf / vici.

Permanent link:

<http://wiki.nctl.de/dokuwiki/doku.php?id=it-themen:grundlagen:netzwerkdienste:kubernetes&rev=1764852751>

Last update: **04.12.2025 13:52**

