OMNI52

Kubernetes Core

v1.34 – 2026.06

Architektur

Control Plane

kube-apiserver: REST-Front, einzige etcd-Schreibstelle, Admission-Plugins.
etcd: konsistenter KV-Store (Raft), 3/5/7 Nodes, fsync-latenz ist der Bottleneck.
kube-scheduler: Bind-Phase Pod → Node, Filter + Score.
controller-manager: Reconcile-Loops (Deployment, ReplicaSet, Endpoints, Node, …).
cloud-controller-manager: provider-spezifisch (LB, Routes, Node-Lifecycle).

Node

kubelet: Pod-Spec → CRI-Calls, Probes, Volume-Mounts, Status-Report.
kube-proxy: Service-VIP → Pod-IP (iptables/ipvs/nftables).
CRI (containerd, CRI-O), CNI (Cilium, Calico), CSI (Treiber pro Storage).

```
kubectl version --short
kubectl get --raw='/readyz?verbose'
kubectl get componentstatuses     # deprecated, sieh healthz
```

Workloads

Pod & Wahl der Workload-Resource

Deployment: stateless, rolling, austauschbare Replicas.
StatefulSet: ordinale Namen (`web-0…N`), stabile Netz-IDs, geordneter Start/Stop, eigene PVCs.
DaemonSet: ein Pod je Node (CNI, Log-Shipper, Node-Exporter).
Job / CronJob: einmalig/periodisch, `completions` + `parallelism`, `backoffLimit`.
Pod direkt nur für Debug/ephemeral.

Deployment: Rolling Update

`strategy.rollingUpdate.maxUnavailable` + `maxSurge` steuern Tempo. Default 25 % / 25 %. Für Singletons: `strategy.type: Recreate`. Pause: `kubectl rollout pause`, dann mehrere Patches, dann `resume`.

```
apiVersion: apps/v1
kind: Deployment
metadata: {name: api}
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate: {maxUnavailable: 0, maxSurge: 1}
  selector: {matchLabels: {app: api}}
  template:
    metadata: {labels: {app: api}}
    spec:
      containers:
      - name: api
        image: ghcr.io/acme/api:1.4.2
        resources:
          requests: {cpu: 100m, memory: 128Mi}
          limits:   {memory: 256Mi}
        readinessProbe:
          httpGet: {path: /readyz, port: 8080}
          periodSeconds: 5
```

StatefulSet: Eigenheiten

`serviceName` = Headless-Service (`ClusterIP: None`), DNS pro Pod: `web-0.svc.ns.svc`. `volumeClaimTemplates` legt PVC pro Replica an (bleibt bei Pod-Replace). `podManagementPolicy: Parallel` bricht Ordnung auf, schneller bei zustandslosen Workloads, die StatefulSet nur wegen stabiler DNS brauchen.

Job-Pattern

`completions: N, parallelism: M` – N Tasks, M parallel.
`activeDeadlineSeconds` hartes Timeout (Pod-Kill).
`ttlSecondsAfterFinished` – Aufräumen ohne Cron.
CronJob `concurrencyPolicy: Forbid|Replace|Allow`, `startingDeadlineSeconds` gegen Backlog nach Cluster-Pause.

Networking

Service-Typen

ClusterIP (default): VIP im Cluster.
NodePort: + Port 30000–32767 auf jedem Node.
LoadBalancer: + externer LB via cloud-controller.
ExternalName: DNS-CNAME, kein Proxy.
Headless (`clusterIP: None`): A-Records pro Endpoint, für StatefulSets + Client-side LB.

```
apiVersion: v1
kind: Service
metadata: {name: api}
spec:
  selector: {app: api}
  ports:
  - {name: http, port: 80, targetPort: 8080}
  internalTrafficPolicy: Cluster   # Local = nur lokale Node-Pods
```

EndpointSlice (seit 1.21)

Löst `Endpoints` ab. Slices à 100 Endpoints, skaliert auf >1000 Pods. Topology-Hints (`service.kubernetes.io/topology-mode: Auto`) reduzieren Cross-Zone-Traffic.

Ingress vs. Gateway API

Ingress ist eingefroren – HTTP/L7-only, vendor-Annotations explodieren.
Gateway API (GA seit 1.31) trennt Rollen: Infrastructure (`GatewayClass`, `Gateway`) vs. App-Team (`HTTPRoute`, `GRPCRoute`, `TLSRoute`). Cross-Namespace via `ReferenceGrant`. Für Service Mesh: `kubernetes-cheatsheet.de` → `istio-cheatsheet.de`.

NetworkPolicy (Default-Deny)

Allowlist-Modell – ohne Policy: alles offen. Pattern: default-deny pro Namespace, dann Ingress/Egress explizit öffnen. `podSelector: {}` = alle Pods, `policyTypes: [Ingress, Egress]` ohne `egress:`-Block = total dicht.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: {name: default-deny, namespace: prod}
spec:
  podSelector: {}
  policyTypes: [Ingress, Egress]
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata: {name: allow-api, namespace: prod}
spec:
  podSelector: {matchLabels: {app: api}}
  ingress:
  - from:
    - namespaceSelector: {matchLabels:
      {kubernetes.io/metadata.name: ingress}}
    ports: [{port: 8080}]
```

Storage

PV / PVC / StorageClass

StorageClass: Treiber + Parameter (`provisioner`, `reclaimPolicy: Delete|Retain`, `volumeBindingMode: WaitForFirstConsumer` → Topology-Aware).
PVC: Anforderung der App (`accessModes`, `storage: 20Gi`). Dynamic Provisioning erzeugt PV.
PV: Cluster-Resource, an PVC gebunden.

AccessModes & Erweitern

`ReadWriteOnce` (RWO, Node-exklusiv), `ReadOnlyMany` (ROX), `ReadWriteMany` (RWX, NFS/CephFS), `ReadWriteOncePod` (RWOP, ein Pod).
Online-Expand: StorageClass mit `allowVolumeExpansion: true`, PVC patchen (nicht schrumpfen). Treiber muss `ControllerExpandVolume` + `NodeExpandVolume` können.

VolumeMode & Snapshots

`volumeMode: Filesystem` (Default) oder `Block` (Raw-Device, DB-Storage).
VolumeSnapshot / `VolumeSnapshotClass` (CSI-Feature) für Point-in-Time-Restore. Snapshot → neues PVC via `dataSource`.

Security

RBAC

Role / ClusterRole: `rules[].{apiGroups, resources, verbs}`.
RoleBinding / ClusterRoleBinding: bindet Subject (User, Group, ServiceAccount) an Role.
Aggregation: `aggregationRule.clusterRoleSelectors` sammelt mehrere ClusterRoles – `view`, `edit`, `admin` sind so gebaut.

```
kubectl auth can-i delete pods --as=alice -n prod
kubectl auth can-i --list --as=system:serviceaccount:prod:api
```

ServiceAccount-Token (seit 1.24)

Projected, bounded: kurzlebig (1h), `audience`-bound, auto-rotiert.
Klassische `Secret`-Tokens werden nicht mehr automatisch angelegt – explizit via `kubernetes.io/service-account-token` oder besser: TokenRequest-API + `projected` Volume.

Pod Security Admission

Ersetzt PodSecurityPolicy. Label am Namespace: `pod-security.kubernetes.io/{enforce|audit|warn}: {privileged|baseline|restricted}`. `restricted` verlangt `runAsNonRoot`, dropped `ALL` Caps, `seccompProfile: RuntimeDefault`.

SecurityContext (Pflicht-Set)

`runAsNonRoot: true`, `runAsUser: 1000`, `readOnlyRootFilesystem: true`, `allowPrivilegeEscalation: false`, `capabilities.drop: [ALL]`, `seccompProfile.type: RuntimeDefault`.

Secrets at Rest

etcd-Klartext per Default. KMS-Provider (`EncryptionConfiguration`) + externer KMS (Vault, Cloud-KMS) für Envelope-Encryption. Alternativ: External Secrets Operator (kein Klartext im etcd).

Scheduling

Requests / Limits / QoS

Guaranteed: requests = limits, kein OOM unter Druck.
Burstable: requests < limits.
BestEffort: keine requests, erste Opfer beim Eviction.
CPU-Limits drosseln (CFS-throttle, Latenz-Spitzen). Memory-Limit-Hit ⇒ OOMKill.
Faustregel: Memory-Limit ja, CPU-Limit nur wenn Multi-Tenant-Schutz nötig.

Affinity / Anti-Affinity

`podAntiAffinity.requiredDuringSchedulingIgnoredDuringExecution` mit `topologyKey: kubernetes.io/hostname` verteilt Replicas auf Nodes. `preferred…` ist Hint, kein harter Constraint. `topologyKey: topology.kubernetes.io/zone` = Zone-Spread.

Taints / Tolerations

Taint am Node: `kubectl taint node n1 gpu=true:NoSchedule`.
Toleration im Pod erlaubt Schedule auf getainted Node – erlaubt, nicht erzwingt (dafür nodeSelector oder Affinity).

Topology Spread Constraints

`maxSkew` + `topologyKey` + `whenUnsatisfiable: DoNotSchedule|ScheduleAnyway`. Modernerer Ersatz für viele Anti-Affinity-Regeln, skaliert besser bei >50 Replicas.

PriorityClass & Preemption

`PriorityClass` mit `value` (höher = wichtiger) und `preemptionPolicy: PreemptLowerPriority|Never`. `system-cluster-critical` / `system-node-critical` reserviert. Eigene Klassen für SLO-kritische Workloads anlegen.

Autoscaling & Resource-Schutz

HPA

metrics.k8s.io (CPU/Mem) oder custom/external (Prometheus-Adapter, KEDA). `minReplicas≥2` für Verfügbarkeit. Stabilisierung: `behavior.scaleDown.stabilizationWindowSeconds` (Default 300 s) gegen Flapping.

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata: {name: api}
spec:
  scaleTargetRef: {kind: Deployment, name: api, apiVersion:
     apps/v1}
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource: {name: cpu, target: {type: Utilization,
     averageUtilization: 70}}
```

VPA

recommender: schlägt requests vor (sicher).
updater + admission: setzt requests live – Pod-Recreate.
Achtung: VPA `Auto` + HPA auf CPU/Mem kollidieren. Kombi nur via custom metrics oder VPA nur für Memory.

PDB / ResourceQuota / LimitRange

PodDisruptionBudget: `minAvailable` oder `maxUnavailable` – Drains, Upgrades respektieren das.
ResourceQuota: Namespace-Limit für Summen (`requests.cpu`, `count/pods`, `persistentvolumeclaims`).
LimitRange: Default + Min/Max pro Container – erzwingt, dass jeder Pod requests trägt.

Observability & Probes

Probe-Sequenz

startupProbe: blockt liveness/readiness bis App hochfährt (Java, .NET). `failureThreshold · periodSeconds` = Max-Boot-Zeit.
readinessProbe: bestimmt Service-Endpoint-Inclusion. Failt → aus dem LB raus, Pod läuft weiter.
livenessProbe: failt → Container-Restart. Vorsicht: Liveness ist nicht für Dependency-Health (DB unten ⇒ Pod kreist).

Probe-Pflicht

`initialDelaySeconds` ist Legacy – nimm `startupProbe`.
HTTP-Probe ohne Host-Header trifft Default-Backend (falsche 200). `httpHeaders` setzen, wenn die App Host-based-Routing macht.

kubectl-Werkzeuge

`logs --previous` für gerade gecrashte Pods.
`logs -f --tail=100 --since=10m -l app=api`.
`top pod -A --sort-by=memory`.
`events --for pod/api-xxx` (1.30+, ohne Eventlog-Volltext).

Diagnose

kubectl debug

`kubectl debug -it pod/api-xxx --image=busybox --target=api` – ephemeral Container im Pod-Namespace (Process+Network).
`--copy-to=api-debug` – Pod kopieren, Image ersetzen.
`node/n1` – Privilegierter Container auf Node mit Host-PID/-Net.

Häufige Fehlerbilder

CrashLoopBackOff: App stürzt direkt nach Start – `logs --previous` + `describe`.
ImagePullBackOff: Registry, Tag, `imagePullSecret`.
OOMKilled: Memory-Limit zu tief, Reason in `lastState`.
Pending (Scheduling): `describe` sagt warum (Insufficient cpu/memory, no nodes match selector, taints).
Init:0/1: InitContainer hängt – `logs -c <init>`.
ContainerCreating (lange): CSI-mount, image-pull, CNI.

```
kubectl get pod api-xxx -o
    jsonpath='{.status.containerStatuses[*]\
.state}' | jq
kubectl describe node n1 | grep -A5 Allocatable
kubectl get events -A --sort-by=.lastTimestamp | tail -40
```

Anti-Patterns

Was du nicht tun solltest

`image: foo:latest`: nicht reproduzierbar, Rolling Update merkt nichts.
Keine requests: BestEffort, erste Opfer bei Eviction; HPA ohne CPU-request liefert keine Metrik.
CPU-Limit gleich requests bei latenz-kritischen Apps: CFS-throttle erzeugt p99-Spitzen.
`hostNetwork: true` aus Bequemlichkeit: Port-Konflikte, NetworkPolicy umgangen.
Liveness als Health-of-Dependencies: DB unten ⇒ Cluster restartet sich selbst tot.
NodePort als Production-Ingress: kein TLS-Mgmt, kein L7, Ports kollidieren.
kubectl apply ohne `--server-side`: Three-Way-Merge bei verteilten Controllern erzeugt Drift.

kubernetes-cheatsheet.de

Kubernetes Cheatsheet von OMNI52

Weiterführen, nicht selbst neu erfinden

Architektur-Reviews & Hardening

OMNI52™ hilft Plattform-Teams, Kubernetes-Cluster für regulierte Enterprise-Umgebungen sauber aufzustellen.

Cluster-Review, RBAC- und PSA-Härtung, NetworkPolicy-Rollout ohne Produktions-Riss, GitOps-Migration. Output landet als GitOps-Repo bei euch im Cluster: Helm-Charts, Runbooks, Diagnose-Skripte. Euer Team operiert weiter, nicht wir.

OMNI52 GmbH, Ebern · istio-consulting.de · kostenloses 30-min Initial-Assessment

Aus der OMNI52 Cheatsheet-Fabrik

Auch erreichbar unter k8s-cheatsheet.de (301 hierher).

Verwandte Sheets: istio-cheatsheet.de (Service-Mesh-Layer) · service-mesh-cheatsheet.de (Istio + Linkerd + Cilium im Vergleich) · rancher-cheatsheet.de (Cluster-Management).