OMNI52

Rancher

2.x – 2026.06

Architektur

Management-Plane

Rancher Server: Multi-Cluster-Management, via Helm auf einem eigenen lokalen Cluster (RKE2/K3s) deployt. HA: 3 Replicas, zustandslos – State liegt im etcd des lokalen Clusters, nicht im Pod.
`local`: der Cluster, auf dem Rancher selbst läuft.
Downstream: die verwalteten Cluster (imported/provisioned).

Agents im Downstream

`cattle-cluster-agent` (Deployment, `cattle-system`): öffnet den Tunnel ausgehend zu Rancher (remotedialer/websocket). Rancher braucht keinen Inbound-Zugang zum Downstream – kubectl, Metrics, Health laufen durch diesen Tunnel zurück.
`cattle-node-agent` (DaemonSet): Node-Operationen, Upgrade-Handling auf importierten Clustern.

API-Frameworks

Norman – legacy v3-API (`management.cattle.io`, `/v3`): Cluster, User, Project, RoleTemplate.
Steve – generischer K8s-Proxy (`/v1`), treibt Cluster Explorer/Dashboard, spricht jede CRD durch.
Downstream-Zugriff wird über `/k8s/clusters/<id>` authentifiziert geproxyt.

Cluster-Management

Imported vs. Provisioned

Imported: bestehender Cluster, Registration-Manifest anwenden, `cattle-cluster-agent` übernimmt. Rancher verwaltet Zugriff/RBAC, aber keinen Node-Lifecycle.
Provisioned (Provisioning v2): Rancher erzeugt Nodes und Cluster über Cluster API (CAPI). CRD `provisioning.cattle.io/v1 Cluster`. RKE2/K3s als Distribution. RKE1 (`rke.cattle.io`) ist deprecated.

Machine-Pools & Registrierung

`machinePools`: Node-Gruppen mit Rolle (`controlPlane`/`etcd`/`worker`) + Machine-Config je Infra-Provider (Amazon EC2, vSphere, Harvester, …).
Import: `kubectl apply -f <registrationURL>`.
Custom-Nodes: `docker run ... rancher/rancher-agent` mit Node-Rollen-Flags.

```yaml
apiVersion: provisioning.cattle.io/v1
kind: Cluster
metadata: {name: prod-eu, namespace: fleet-default}
spec:
  kubernetesVersion: v1.31.5+rke2r1
  rkeConfig:
    machinePools:
    - name: cp
      controlPlaneRole: true
      etcdRole: true
      quantity: 3
      machineConfigRef:
        kind: HarvesterConfig
        name: cp-medium
    - name: worker
      workerRole: true
      quantity: 4
      machineConfigRef: {kind: HarvesterConfig, name:
     worker-large}
```

Auth & RBAC

Rollen-Ebenen

Global: Rancher-weite Rechte (User anlegen, Cluster erstellen).
Cluster: `cluster-owner`, `cluster-member`.
Project: `project-owner`, `project-member`, `read-only`.
RoleTemplate (`management.cattle.io`) definiert die aggregierten Regelsätze, custom Rollen davon abgeleitet.

Auth-Provider

Local, Active Directory, LDAP (OpenLDAP/FreeIPA), SAML (Keycloak, Okta, ADFS, PingID, Shibboleth), OIDC (generisch/Keycloak), GitHub, Google, Azure AD/Entra.
Ein Provider aktiv zur Zeit; Gruppen-Mapping treibt die Rollen-Zuweisung.

Mapping auf Kubernetes-RBAC

Rancher-Rollen sind keine K8s-Objekte – der Management-Controller projiziert sie als `ClusterRoleBinding`/`RoleBinding` in den Downstream. Zugriff läuft über den Proxy als `impersonate`-User; native K8s-RBAC greift zusätzlich. Gruppen aus dem Auth-Provider → Rancher-Rolle → K8s-Binding.

Projects & Namespaces

Project-Abstraktion

Ein Project bündelt Namespaces (Rancher-Konstrukt, im Management-Cluster gespeichert – nicht im Downstream-API). Namespace-Zuordnung via Annotation `field.cattle.io/projectId`. RBAC und Quotas hängen am Project, nicht am einzelnen Namespace.

Resource-Quotas & PSA

Project-Quota: Gesamtbudget, das Rancher auf die Namespaces verteilt (Project-Limit + Namespace-Default).
Pod Security Admission: Rancher liefert PSA Configuration Templates (Ersatz für die entfernten PSP-Templates), clusterweit oder pro Namespace via `pod-security.kubernetes.io/*`-Labels.

Fleet (GitOps)

Fleet-Modell

Continuous Delivery über viele Cluster, mit Rancher gebündelt (`cattle-fleet-system`). Manager auf `local`, Agents im Downstream.
`GitRepo` (`fleet.cattle.io`) → rendert Bundles → BundleDeployment je Ziel-Cluster. Targeting per `clusterSelector`/`clusterGroup`.

```yaml
apiVersion: fleet.cattle.io/v1alpha1
kind: GitRepo
metadata: {name: platform, namespace: fleet-default}
spec:
  repo: https://git.example.com/platform
  branch: main
  paths: [infra/, apps/]
  targets:
  - name: prod
    clusterSelector:
      matchLabels: {env: prod}
```

Targeting-Regeln

Labels auf dem Fleet-`Cluster`-Objekt treiben die Auswahl. Ohne `targets` greift das Default-Ziel (nur `local`). `fleet.yaml` im Repo steuert Overlays, Helm-Values und `dependsOn` zwischen Bundles.

Observability

rancher-monitoring

Auf Basis kube-prometheus-stack (Prometheus Operator, Prometheus, Alertmanager, Grafana, node-exporter), pro Cluster aus Apps/Charts installiert, Namespace `cattle-monitoring-system`. Dashboards + `ServiceMonitor`/`PrometheusRule` out-of-the-box.

rancher-logging & Alerting

Logging (Banzai Logging Operator: Fluent Bit + Fluentd), CRDs `ClusterFlow`/`ClusterOutput`/`Flow`/`Output`, Namespace `cattle-logging-system`.
Alerting: Alertmanager + `PrometheusRule`, Rancher-Alerting-Drivers für Teams/Slack/PagerDuty.

Backup & Restore

rancher-backup Operator

Sichert den Rancher-Server-State (Management-CRDs, User, Cluster, Rollen) – nicht Downstream-Workloads. CRDs `Backup`/`Restore` (`resources.cattle.io`), Ziel S3 oder PVC. Kernszenario: DR/Migration des Rancher-Servers.

```yaml
apiVersion: resources.cattle.io/v1
kind: Backup
metadata: {name: rancher-nightly}
spec:
  resourceSetName: rancher-resource-set
  schedule: "0 2 * * *"
  retentionCount: 7
  storageLocation:
    s3:
      bucketName: rancher-backups
      endpoint: s3.eu-central-1.amazonaws.com
      credentialSecretName: s3-creds
      credentialSecretNamespace: cattle-resources-system
```

etcd-Snapshots (Downstream)

Getrennt vom rancher-backup: RKE2/K3s sichern ihr eigenes etcd. `etcd.snapshotScheduleCron` + `snapshotRetention` im Cluster-Spec, S3-Ziel optional. Restore über Rancher-UI oder `rke2 server ---cluster-reset ---cluster-reset-restore-path`.

Ops & Diagnose

cattle-cluster-agent (Fehlerquelle #1)

Cluster `Unavailable`? Fast immer der Agent-Tunnel:
server-url falsch/nicht erreichbar, CA-Mismatch (Agent braucht Rancher CA), Proxy/DNS, abgelaufenes Token.

```bash
kubectl -n cattle-system logs -l app=cattle-cluster-agent
kubectl -n cattle-system get deploy cattle-cluster-agent
# Registration/Server-URL prüfen:
kubectl -n cattle-system get settings.management.cattle.io \
  server-url -o jsonpath='{.value}'
```

Stuck-States & Zerts

Updating/Unavailable: Agent-Disconnect, Cert-Expiry oder etcd/controlplane not ready – `describe cluster`, Agent-Logs.
Cert-Rotation: RKE2/K3s über `rke2 certificate rotate` (Node-Neustart) oder UI "Rotate Certificates". Interne Zerts ~1 Jahr gültig, auto-Rotation beim Restart innerhalb 90 d vor Ablauf.
kubectl: Kubeconfig aus der UI proxyt über Rancher (`/k8s/clusters/<id>`), authentifiziert per Rancher-Token.

Anti-Patterns

Was du nicht tun solltest

Rancher-Server + Prod-Workloads im selben Cluster: der **`local`**-Cluster ist Management-Plane, kein Workload-Cluster – Blast-Radius und Upgrade-Kopplung.
Keine etcd-Backups: rancher-backup sichert nur den Server, nicht die Downstream-Cluster – beide brauchen einen Plan.
Version-Skew: Rancher-Version zu Downstream-K8s außerhalb der Support-Matrix koppelt Upgrades hart.
Self-signed Certs ohne Rotation: Agent-Tunnel bricht bei Ablauf, Cluster geht **`Unavailable`**.
Cluster-Zugriff nur über UI: kein GitOps – Drift. Fleet/**`GitRepo`** als Source of Truth.

rancher-cheatsheet.de

Rancher Cheatsheet von OMNI52

Weiterführen, nicht selbst neu erfinden

Multi-Cluster-Betrieb & Hardening

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

Management-Cluster-Design, RBAC-/Auth-Provider-Integration, Fleet-GitOps-Rollout, Backup-/DR-Konzept für Server und Downstream-etcd, Agent-Konnektivität ohne Produktions-Riss. 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

Verwandte Sheets: kubernetes-cheatsheet.de (Kubernetes Core) · rke2-cheatsheet.de (RKE2-Distribution) · k8s-cheatsheet.de (Kubernetes-Kurzref) · service-mesh-cheatsheet.de (Service Mesh im Vergleich).

Lizenz & Weiterverteilung

Trademarks