Helm 4: Stand Juli 2026

Versionen, Support-Fenster

Helm 4 ist die aktuelle Major-Version (4.0.0 im November 2025, aktuell 4.2.x). Charts mit **apiVersion: v2** laufen unveraendert; bestehende Helm-3-Releases ueber-nimmt Helm 4 ohne Migrationsschritt.
Helm 3 laeuft aus: letzter (limitierter) Feature-Release am 09.09.2026 (nur K8s-Client-Updates), Security-Fixes bis 10.02.2027, danach EOL.

Breaking Changes 3 auf 4

--atomic → **--rollback-on-failure**, **--force** → **--force-replace** (alte Flags funktionieren mit Deprecation-Warnung).
--post-renderer nimmt einen Plugin-Namen, kein Executable mehr.
helm registry login akzeptiert nur noch Domains, keine URLs.
--wait nutzt kstatus: RBAC braucht zusaetzlich das **watch**-Verb, sonst schlaegt Warten sofort fehl.

Neu in Helm 4

Server-Side Apply als Default fuer neue Releases (Details unter Release-Lifecycle).
Plugin-System neu: optionale WebAssembly-Runtime, Typen CLI / Getter / Post-Renderer; Subprocess-Plugins laufen weiter (deprecated).
Multi-Dokument-Values, Custom-Template-Funktionen via Plugins, Content-basiertes Chart-Caching, schnellere Dependency-Aufloesung.

Chart-Anatomie

Verzeichnis-Layout

Chart.yaml (Metadaten, Pflicht) · **values.yaml** (Defaults) · **values.schema.json** (optionales JSON-Schema) · **templates/** (Manifeste + **_helpers.tpl**) · **charts/** (gevendorte Dependencies) · **crds/** (CRDs, siehe Anti-Patterns).
charts/, **crds/**, **templates/** sind fuer Helm reserviert.

Chart.yaml-Kernfelder

apiVersion: v2 – Standard fuer Helm 3 und 4 (**v1** = Helm 2; Charts v3 in Entwicklung auf Basis der stabilen Helm-4-SDK).
version: SemVer des Charts – treibt Paketname, Registry-Tag und Range-Aufloesung.
appVersion: Version der paketierten App, informativ.
type: **application** oder **library** (Library-Charts rendern selbst nichts, liefern Template-Bausteine).

```yaml
apiVersion: v2
name: shop-api
version: 1.4.0        # Chart-SemVer
appVersion: "2.8.1"   # App-Version, informativ
type: application
dependencies:
  - name: postgresql
    version: "⩾16.0.0 <17.0.0"
    repository: oci://registry.example.com/charts
    condition: postgresql.enabled
```

Templating-Patterns

include, required, tpl

include rendert ein Named Template zu einem String – der Output laesst sich pipen (**| nindent 4**), anders als bei der **template**-Action.
required bricht das Rendering mit eigener Fehlermeldung ab, wenn der Values-Eintrag leer ist.
tpl evaluiert Strings aus values als Template – fuer konfigurierbare Templates und **.Files.Get**-Inhalte.

Named Templates (_helpers.tpl)

define-Namen sind global ueber Chart + Subcharts – immer mit Chart-Namen prefixen: **shop.labels** statt **labels**. Ablage konventionell in **templates/_helpers.tpl** (Underscore-Dateien rendern kein Manifest).

```
{{- define "shop.labels" -}}
app.kubernetes.io/name: {{ .Chart.Name }}
app.kubernetes.io/version: {{ .Chart.AppVersion }}
{{- end }}
# Verwendung: Output pipen, nicht 'template'
metadata:
  labels: {{- include "shop.labels" . | nindent 4 }}
host: {{ required "host fehlt!" .Values.host }}
conf: {{ tpl (.Files.Get "conf/app.conf") . }}
```

Checksum-Trick

ConfigMap-Aenderung rollt das Deployment nicht automatisch – Checksum-Annotation im Pod-Template erzwingt den Rollout:

```
annotations:
  checksum/config: {{ include (print $.Template.BasePath
    "/configmap.yaml") . | sha256sum }}
```

Values-Schichtung & Secrets

Praezedenz

Chart-**values.yaml** < Parent-Chart-Overrides < **-f**-Dateien (rechts gewinnt) < **--set** / **--set-string** / **--set-file** / **--set-json**.
Subchart-Values unter dem Subchart-Namen scopen; **global:** ist in allen Charts sichtbar. **--reset-values** verwirft beim Upgrade gemerkte Overrides.

Secrets-Strategien

Nie Klartext-Secrets in values.yaml im Git. Gaengige Muster:
helm-secrets-Plugin + SOPS/age: values verschluesselt im Repo, Entschluesselung beim Deploy.
External Secrets Operator / Sealed Secrets: Secret-Lifecycle ausserhalb des Charts, Chart referenziert nur den Namen.
CI-Injection: **--set-file** aus dem Secret-Store der Pipeline, nichts landet im Repo.

```
helm upgrade --install shop ./shop -n shop \
  -f values.yaml -f values-prod.yaml \
  --set image.tag=2.8.1 \
  --set-file tls.cert=./tls.crt \
  --set-json 'resources={"limits":{"cpu":"1"}}'
helm get values shop -n shop   # gemerkte Overrides
```

Release-Lifecycle

install, upgrade, rollback

helm upgrade --install ist der idempotente Standardpfad (CI-tauglich). **helm history** zeigt Revisionen, **helm rollback <rel> <rev>** setzt zurueck (erzeugt neue Revision). **helm get manifest|values|notes** inspiziert den Ist-Stand. **--rollback-on-failure** rollt fehlgeschlagene Upgrades automatisch zurueck.

3-way merge, Server-Side Apply

Helm 3: Three-Way Strategic Merge Patch aus altem Manifest, Live-State und neuem Manifest – out-of-band-Aenderungen bleiben erhalten, soweit nicht im Chart ueberschrieben.
Helm 4: Server-Side Apply (Field-Ownership beim API-Server) als Default fuer neue Releases. Upgrades/Rollbacks latchen auf die bisherige Apply-Methode des Releases (**--server-side=auto**); Umstellung explizit per **--server-side=true|false**.

```
helm upgrade --install shop ./shop -n shop \
  --rollback-on-failure --wait --timeout 5m
helm history shop -n shop
helm rollback shop 3 -n shop
helm get manifest shop -n shop --revision 3
helm uninstall shop -n shop --keep-history
```

Hooks & Tests

Hook-Mechanik

Annotation **helm.sh/hook** macht ein Template zum Hook: **pre-install**, **post-install**, **pre-upgrade**, **post-upgrade**, **pre-delete**, **post-delete**, **pre-rollback**, **post-rollback**, **test**.
Reihenfolge via **helm.sh/hook-weight** (String!, aufsteigend). Job/Pod-Hooks blockieren bis Completion – Fehler = Release fehlgeschlagen.

Delete-Policies, Ownership

Hook-Ressourcen gehoeren nicht zum Release – **helm uninstall** raeumt sie nicht ab. **helm.sh/hook-delete-policy**: **before-hook-creation** (Default), **hook-succeeded**, **hook-failed**.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Release.Name }}-db-migrate"
  annotations:
    "helm.sh/hook": pre-install,pre-upgrade
    "helm.sh/hook-weight": "-5"
    "helm.sh/hook-delete-policy":
     before-hook-creation,hook-succeeded
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: migrate
          image: registry.example.com/shop/migrate:2.8.1
```

Chart-Tests

Tests sind Hooks mit **helm.sh/hook: test**, konventionell unter **templates/tests/**. Nach Deploy ausfuehren: **helm test <release>** – der Test-Pod muss erfolgreich terminieren (Smoke-Test gegen den Service).

Dependencies

dependencies-Feld

Subcharts deklarativ in **Chart.yaml**; **helm dependency update** loest Ranges auf, laedt **.tgz** nach **charts/** und schreibt **Chart.lock** (**dependency build** baut reproduzierbar aus dem Lock).
condition (z.B. **redis.enabled**) schaltet Subcharts, tags buendeln mehrere, alias instanziiert dasselbe Chart mehrfach.

```
dependencies:
  - name: redis
    version: "21.x.x"
    repository: oci://registry.example.com/charts
    condition: redis.enabled
  - name: postgresql
    version: "⩾16.0.0 <17.0.0"
    repository: "@internal"   # Alias aus 'helm repo add'
    tags: [datastores]
```

```
helm repo add internal https://charts.example.com
helm dependency update ./shop   # Ranges -> Chart.lock
helm dependency build ./shop    # exakt aus Chart.lock
```

OCI-Registries

Chart als OCI-Artefakt

helm push laedt das gepackte Chart in eine OCI-Registry – Referenz mit **oci://**-Prefix, ohne Chart-Namen und Tag (kommen aus **Chart.yaml**). **pull**, **install**, **upgrade**, **show**, **template** akzeptieren **oci://** + **--version**; Pinning per Digest **@sha256:...**. Login in Helm 4 nur mit Domain.

```
helm registry login registry.example.com
helm package ./shop   # -> shop-1.4.0.tgz
helm push shop-1.4.0.tgz oci://registry.example.com/charts
```

```
helm install shop \
  oci://registry.example.com/charts/shop --version 1.4.0
# immutable per Digest:
helm install shop \
  oci://registry.example.com/charts/shop@sha256:<digest>
```

Lint, Template & Diff

Validierungs-Pipeline

helm lint --strict: statische Chart-Pruefung.
helm template: Offline-Rendering – in CI gegen Schema-Validatoren wie kubeconform pipen.
helm install --dry-run=server: rendert mit Cluster-Zugriff (**lookup** funktioniert), ohne zu installieren.
helm diff upgrade (Plugin): zeigt den Manifest-Diff gegen den Live-Stand vor dem Upgrade.

```
helm lint ./shop --strict
helm template shop ./shop -f values-prod.yaml \
  | kubeconform -strict -summary
helm install shop ./shop --dry-run=server
helm plugin install \
  https://github.com/databus23/helm-diff
helm diff upgrade shop ./shop -f values-prod.yaml
```

Provenance & Signing

Signieren, Verifizieren

helm package --sign erzeugt neben dem **.tgz** eine Provenance-Datei **.tgz.prov** (PGP-signiert; Keyring im binaeren GnuPG-Format, kein **kbx**).
Pruefung: **helm verify chart.tgz** bzw. Installation mit **--verify** – die **.prov** muss neben dem Chart abrufbar sein.

```
helm package --sign --key 'release@example.com' \
  --keyring ~/.gnupg/secring.gpg ./shop
helm verify shop-1.4.0.tgz
helm install shop shop-1.4.0.tgz --verify \
  --keyring pubring.gpg
```

Anti-Patterns

Was du nicht tun solltest

Chart-Version wiederverwenden: gleiche **version** neu pushen bricht Caches und SemVer-basierte GitOps-Ranges – pro Aenderung bumpen.
Klartext-Secrets in values.yaml im Git: SOPS/helm-secrets, External Secrets oder CI-Injection nutzen.
kubectl edit am Release vorbei: 3-way merge/SSA begrenzen den Schaden, Ownership-Konflikte bleiben – Aenderungen ueber values fahren.
CRDs als Templates: **crds/** wird nur bei der Installation angelegt, nie geupgradet oder geloescht – CRD-Lifecycle separat managen.
App-Logik in Hooks: Hook-Ressourcen gehoeren nicht zum Release, kein Cleanup bei uninstall – delete-policy setzen, Jobs idempotent bauen.
Upgrade ohne Diff/Dry-Run in Prod: **helm diff** bzw. **--dry-run=server** gehoeren vor jedes produktive Upgrade.
–wait ohne watch-RBAC (Helm 4): kstatus braucht **watch** – sonst bricht jede Wait-Operation ab.

helm-cheatsheet.de

Helm Cheatsheet von OMNI52

Weiterfuehren, nicht selbst neu erfinden

Helm-4-Migration & Chart-Hygiene

OMNI52™ hilft Plattform-Teams, Helm in regulierten Enterprise-Umgebungen sauber zu betreiben.

Helm-4-Migration ohne Produktions-Riss (Flag-Renames, Post-Renderer als Plugins, SSA-Latching, kstatus-RBAC), Chart-Reviews gegen die Anti-Patterns dieses Sheets, OCI-Registry- und Signing-Setup, Values-/Secrets-Architektur mit SOPS oder External Secrets. Output landet als GitOps-Repo bei euch im Cluster: Charts, Runbooks, CI-Pipelines. 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) · flux-cheatsheet.de (GitOps mit Flux) · argocd-cheatsheet.de (GitOps mit Argo CD) · kubectl-cheatsheet.de (kubectl Power-Usage).

Lizenz & Weiterverteilung

CC BY-SA 4.0 — du darfst dieses Cheatsheet kopieren, weiterverteilen, ausdrucken und in eigenen Materialien zitieren. Bedingung: Quellenangabe "Helm Cheatsheet — OMNI52 GmbH, helm-cheatsheet.de" bleibt sichtbar, und abgeleitete Werke stehen unter der gleichen Lizenz (Share-Alike).

Nicht erlaubt: Logo, Marken oder den Eindruck zu vermitteln, dass der Inhalt von dir/euch stammt oder dass OMNI52 GmbH die Weiterverwendung sponsort. Volltext der Lizenz: creativecommons.org/licenses/by-sa/4.0/deed.de.

Trademarks

Helm and Kubernetes are registered trademarks of The Linux Foundation in the United States and other countries. Helm is a Cloud Native Computing Foundation (CNCF) graduated project. OMNI52™ is a trademark of OMNI52 GmbH (filed, not yet registered). This cheatsheet is an independent reference work by OMNI52 GmbH and is not affiliated with, endorsed by, or sponsored by The Linux Foundation or the CNCF. Names are used in a nominative / descriptive sense. Code snippets are based on the public Helm documentation.