Helm Troubleshooting Runbook Pack

A senior engineer's incident runbook for Helm v3 releases that won't install, upgrade, or roll back. DevOps AI ToolKit · devopsaitoolkit.com

Most Helm incidents fall into a handful of shapes: a release is stuck `pending`, an upgrade fails because a resource "already exists", a template refuses to render, or values didn't merge the way you expected. Helm v3 stores every release as a Kubernetes Secret (`sh.helm.release.v1.<name>.v<rev>`) in the release namespace — so the release history *is* recoverable data, not a black box. Work top-to-bottom: read the current release state first, understand *why* it's in that state, and only then mutate. Read-only commands never change a release; treat `upgrade`, `rollback`, and `uninstall` as the change events they are.

> Examples use namespace `prod`, release `web`, and registry `registry.example.com`. Run the read-only checks first; take a change record before any `upgrade` / `rollback` / `uninstall`, and ideally have a second engineer confirm.

1. First-Response Triage

Before touching anything, capture the release's current state. These four commands are entirely read-only.

```
# What state is the release in, and did the last op succeed?
helm status web -n prod

# Full revision history — look at the STATUS column for the last good revision
helm history web -n prod

# List ALL releases including failed/uninstalling/pending ones (-a)
helm list -a -n prod

# What did Helm actually apply to the cluster last?
helm get manifest web -n prod | head -40
```

- `helm status` shows `STATUS: deployed` (healthy), or `failed` / `pending-upgrade` / `pending-install` / `pending-rollback` / `uninstalling`.
- In `helm history`, note the highest revision whose status is `deployed` or `superseded` — that is your rollback target.
- `helm list -a` reveals releases a plain `helm list` hides (anything not `deployed`).
- `helm get manifest` prints the last-applied rendered YAML. Diff it against the cluster when reality and Helm disagree.

2. "another operation is in progress" & Stuck Pending States

`Error: UPGRADE FAILED: another operation (install/upgrade/rollback) is in progress` means a previous command died mid-flight (client killed, CI timeout, controller crash) and left the release in a `pending-*` status. Helm refuses to start a new op until that clears.

```
helm status web -n prod          # confirm STATUS is pending-upgrade / pending-install / pending-rollback
helm history web -n prod         # find the last DEPLOYED / SUPERSEDED revision
```

```
# Preferred fix: roll back to the last good revision (this clears the pending lock)
helm rollback web 7 -n prod

# If the STUCK revision was a first install (pending-install, no prior good rev),
# there is nothing to roll back to — remove the half-release and reinstall:
helm uninstall web -n prod
```

- Do **not** blindly `kubectl delete secret sh.helm.release.v1.web.vN` unless you understand it drops that revision from history; rollback is safer and reversible.
- A truly wedged `pending-install` with no usable history can only be cleared by `uninstall` (see §3) — there is no deployed state to preserve.
- Confirm no controller/CI job is *still* running the op before you intervene, or you'll race it.

3. "has no deployed releases" & Failed First Install

`Error: UPGRADE FAILED: "web" has no deployed releases` almost always follows a *failed first install*. When `helm install` fails, the release exists in history with status `failed` but was never `deployed`; a later `helm upgrade --install` then can't find a deployed revision to upgrade from.

```
helm list -a -n prod             # you'll see web with STATUS failed
helm history web -n prod         # every revision is failed — no deployed anchor
```

```
# There is no good revision to keep — uninstall the failed release, fix the root cause, reinstall
helm uninstall web -n prod
helm install web ./web -n prod --create-namespace

# Investigate WHY the first install failed before reinstalling (image pull, RBAC, quota):
kubectl get events -n prod --sort-by=.lastTimestamp | tail -20
```

- Since Helm 3.2 you can sometimes recover with `helm upgrade --install ... --force`, but a clean `uninstall` / `install` is more predictable for a never-deployed release.
- Fix the underlying cause first (bad image tag, missing secret, quota) or the reinstall fails identically.

---

4. Template Rendering Errors

Rendering errors are pure client-side Go-template failures — nothing has touched the cluster yet. Render locally with `--debug` to see the offending YAML.

```
helm template web ./web -n prod --debug -f values.prod.yaml
# Same render path an install uses, but prints to stdout instead of applying
```

- `nil pointer evaluating interface {}.foo` → you referenced `.Values.image.tag` but `image` (or `tag`) is undefined. Guard it: `{{ .Values.image.tag | default "latest" }}` or gate with `{{- if .Values.image }}`.
- `function "toYaml" not defined` (or `include`, `tpl`) → almost always a typo or a helper used outside a chart context; check spelling and that the chart's `_helpers.tpl` is present.
- Broken indentation / `error converting YAML to JSON` → you used `toYaml` without re-indenting. Use `nindent`: `{{ toYaml .Values.resources | nindent 10 }}`, matching the parent key's depth.
- `--debug` prints the partially rendered document with a `---` marker near the failure — read the YAML it *did* produce to locate the line.

---

5. Values & Precedence

Confusing output usually means values merged differently than expected. Precedence is defined and stable: each `-f/--values` file merges left-to-right (later files win), and every `--set` overrides all `-f` files.

```
# Values YOU supplied on top of chart defaults
helm get values web -n prod

# The full COMPUTED value set (defaults + overrides) that was actually rendered
helm get values web -n prod -a

# Reproduce a merge locally to see the winner for a given key
helm template web ./web -f values.a.yaml -f values.b.yaml --set image.tag=1.4.2 --debug
```

- Order matters: `-f base.yaml -f prod.yaml` lets `prod.yaml` override `base.yaml`; swap them and the result flips.
- `--set image.tag=1.4.2` beats any `image.tag` in every `-f` file. `--set-string` forces string typing (e.g. a numeric-looking tag).
- If the chart ships a `values.schema.json`, Helm validates the merged values against it and fails with `values don't meet the specifications of the schema(s)` — read the JSON-schema path in the error; it names the exact key.

---

6. Resource Conflicts

`Error: rendered manifests contain a resource that already exists. Unable to continue with install: <Kind> "<name>" in namespace "prod" exists and cannot be imported into the current release: invalid ownership metadata` means the object exists but lacks Helm's ownership labels/annotations (it was created by `kubectl`, another release, or a prior failed run).

```
# Inspect the conflicting object's ownership metadata
kubectl get service web -n prod -o jsonpath='{.metadata.annotations}{"\n"}{.metadata.labels}{"\n"}'
```

- Helm adopts a pre-existing resource only if it already carries `app.kubernetes.io/managed-by: Helm` **and** annotations `meta.helm.sh/release-name: web` and `meta.helm.sh/release-namespace: prod`.

```
# To let Helm adopt an existing object, stamp the ownership metadata to match the release:
kubectl label   service web -n prod app.kubernetes.io/managed-by=Helm --overwrite
kubectl annotate service web -n prod \
  meta.helm.sh/release-name=web meta.helm.sh/release-namespace=prod --overwrite
```

- If the object genuinely belongs to something else, don't adopt it — rename the resource in your chart or delete the stray object instead.

---

7. Chart Dependencies

Dependency problems surface at build time, before any cluster call. Subcharts declared in `Chart.yaml` must be materialized into `charts/` (as `.tgz`) with the lock file.

```
helm dependency list ./web          # shows each dependency's status: ok / missing
helm dependency update ./web        # refresh repos, resolve versions, write Chart.lock, fill charts/
helm dependency build ./web         # rebuild charts/ from an EXISTING Chart.lock (CI-friendly, reproducible)
```

```
# Chart.yaml
dependencies:
  - name: redis
    version: "18.x.x"          # semver range: matches highest 18.* release
    repository: "https://charts.example.com"
    condition: redis.enabled    # subchart included only when this value is true
```

- `Error: no repository definition for https://…` or `repo not found` → add it first: `helm repo add example https://charts.example.com && helm repo update`.
- `can't get a valid version for repositories` → your `version:` constraint matches nothing; loosen the range or run `helm repo update` for fresh index data.
- Commit `Chart.lock` and use `helm dependency build` in CI so every environment resolves the identical subchart versions.

---

8. Hooks & Wait Failures

`--wait` blocks until resources report Ready; `--atomic` implies `--wait` and **auto-rolls-back** on failure. `timed out waiting for the condition` means something never became Ready within `--timeout` (default `5m0s`).

```
# Reproduce with a longer window and watch what never goes Ready
helm upgrade web ./web -n prod --atomic --timeout 10m --debug
kubectl get pods,jobs -n prod            # find the Pod/Job stuck non-Ready
kubectl describe pod <stuck-pod> -n prod # events: ImagePullBackOff, CrashLoop, unschedulable
```

```
# A hook Job: annotations control ordering and cleanup
metadata:
  annotations:
    "helm.sh/hook": pre-upgrade
    "helm.sh/hook-weight": "0"          # lower weights run first
    "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded
```

- A failing `pre-upgrade` / `post-install` hook Job aborts the whole release — inspect it: `kubectl logs job/<hook-job> -n prod`.
- With `--atomic`, a timeout triggers an automatic rollback, so `helm status` may read `deployed` on the *old* revision — check `helm history` to see the failed one.
- Without `--wait`, Helm reports success once objects are *created*, not Ready — so "helm said OK but pods are crashing" usually means no `--wait`.

---

9. Rollback & Recovery

Every revision is retained (bounded by `--history-max`, default 10), so rollback is your fastest recovery path.

```
helm history web -n prod                # pick the last DEPLOYED/SUPERSEDED revision
helm rollback web 7 -n prod --wait      # roll back to revision 7 and wait for Ready
helm rollback web -n prod               # omit the number to go to the immediately previous revision
```

- `Error: create: failed to create: Secret "sh.helm.release.v1.web.vN" is invalid: data: Too long` (or `request entity too large`) → the release Secret exceeds etcd's ~1 MB object limit, usually from too many revisions or huge rendered manifests. Trim history:

```
helm upgrade web ./web -n prod --history-max 3   # keep only the last 3 revisions going forward
kubectl get secret -n prod -l owner=helm,name=web  # inspect stored release revisions
```

- If the driver storage is bloated, reducing `--history-max` prunes old revision Secrets on the next op. Never hand-delete the *current* deployed revision Secret.

---

10. Debugging the Deployed Resources

When Helm reports success but the app is broken, the problem is in the live objects, not Helm. Use Helm to identify what it manages, then `kubectl` to inspect it.

```
# Everything Helm applied for this release
helm get manifest web -n prod | kubectl get -f - -n prod

# Server-side render/validate WITHOUT persisting a release (catches admission/CRD issues)
helm upgrade web ./web -n prod --dry-run=server

kubectl rollout status deploy/web -n prod
kubectl get events -n prod --sort-by=.lastTimestamp | tail -20
```

- `Error: unable to build kubernetes objects … no matches for kind "<Kind>" in version "<group>/<v>"` → a CRD isn't installed. Install the

operator/CRD chart first, or use `--skip-crds` / `crds/` ordering correctly.

- `--dry-run=client` renders locally; `--dry-run=server` sends it to the API server for real admission/validation (webhooks, CRD schemas) without saving a release — prefer it for pre-flight checks.

Decision Tree

```
Helm command failed?
 ├─ STATUS is pending-* (another operation in progress)? ── yes → helm rollback to last good rev (§2); if pending-install with no history → uninstall
 ├─ "has no deployed releases"?                          ── yes → failed first install: uninstall, fix cause, reinstall (§3)
 ├─ Error during TEMPLATE render (nil pointer / toYaml)? ── yes → helm template --debug, fix chart (§4)
 ├─ "resource already exists / invalid ownership"?       ── yes → adopt with meta.helm.sh annotations, or remove stray object (§6)
 ├─ dependency/repo/version error?                       ── yes → helm repo add + helm dependency update/build (§7)
 ├─ "timed out waiting for the condition"?               ── yes → describe the non-Ready pod/job/hook (§8)
 └─ install "succeeded" but app broken?                  ── yes → helm get manifest | kubectl + --dry-run=server (§10)
```

Copy/Paste One-Shot Triage

```
NS=prod REL=web
echo "== status ==";     helm status "$REL" -n "$NS" | sed -n '1,12p'
echo "== history ==";    helm history "$REL" -n "$NS"
echo "== all releases =="; helm list -a -n "$NS"
echo "== user values =="; helm get values "$REL" -n "$NS"
echo "== recent events =="; kubectl get events -n "$NS" --sort-by=.lastTimestamp | tail -15
echo "== not-ready pods =="; kubectl get pods -n "$NS" --field-selector=status.phase!=Running
echo "== release secrets =="; kubectl get secret -n "$NS" -l owner=helm,name="$REL"
```

Incident Notes Template

```
INCIDENT: Helm release failure
Started (UTC):        ____    Detected by: ____    Sev: ____
Release / namespace:  web / prod          Chart + version: ____
Command that failed:  (helm upgrade / install / rollback ... )
Release STATUS:       [ ] pending-upgrade [ ] pending-install [ ] failed [ ] deployed
Last good revision:   ____ (from helm history)
Failure class:        [ ] op-in-progress [ ] no-deployed-releases [ ] template
                      [ ] values/schema [ ] resource-conflict [ ] dependency
                      [ ] wait/hook timeout [ ] storage-too-large [ ] CRD/live-object
Evidence:             (error text / kubectl describe / hook job logs) ____
Action taken:         (rollback to rev __ / uninstall+reinstall / metadata adopt + who + when)
Result:               [ ] recovered @ ____   [ ] escalated to ____
Root cause:           ____
Follow-ups:           (--atomic in CI, --history-max, schema, chart guard) ____
```
