Kontexte & kubeconfig

Kontext

`kubectl config get-contexts` zeigt alle, `use-context` wechselt, `set-context --current` setzt den Default-Namespace.
Merge: `KUBECONFIG` nimmt eine Pfad-Liste (`:`-getrennt), erste Definition gewinnt. `config view --flatten` schreibt daraus eine Datei. `--kubeconfig` pinnt pro Aufruf.

Version-Skew

kubectl ist ein Minor +/- um den kube-apiserver supported: v1.36-Client passt zu v1.35–v1.37-Servern. `kubectl version` zeigt beide Seiten – bei Cluster-Flotten mit Versions-Spread den Client bewusst waehlen.

```
kubectl config get-contexts
kubectl config use-context prod-admin
kubectl config set-context --current --namespace=payments
export KUBECONFIG=~/.kube/prod.yaml:~/.kube/lab.yaml
kubectl config view --flatten > ~/.kube/merged.yaml
kubectl version   # Client- vs. Server-Version (Skew)
```

Output-Formate

-o: von wide bis go-template

`-o wide|yaml|json|name` fuer den Alltag.
jsonpath: Feld-Extraktion mit `range`/Filter – Script-Futter statt `grep`-Kaskaden.
custom-columns: eigene Tabellen direkt aus dem API-Objekt.
go-template: volle Template-Logik inkl. `base64decode` fuer Secrets.

Sortieren & Filtern

`--sort-by` nimmt einen JSONPath.
`-l` filtert nach Labels (`app=web,tier≠cache`), `--field-selector` nach Objektfeldern (`status.phase=Running`). `-A` = alle Namespaces.

```
kubectl get pods -o
    jsonpath='{.items[*].spec.containers[*].image}'
kubectl get pods -o jsonpath='{range
    .items[*]}{.metadata.name}{"\t"}{.status.podIP}{"\n"}{end}'
kubectl get nodes -o
    custom-columns='NAME:.metadata.name,TAINT:.spec.taints[*].key'
kubectl get secret db -o go-template='{{index .data "ca.crt"
    | base64decode}}'
kubectl get pods -A --sort-by=.metadata.creationTimestamp
```

Apply, Diff & Server-Side Apply

diff & dry-run

`kubectl diff -f` zeigt den Diff gegen den Live-Stand vor dem Apply (Exit-Code 1 = Unterschiede, CI-tauglich).
`--dry-run=client` validiert nur lokal; `--dry-run=server` laesst Admission-Webhooks + Defaulting am API-Server laufen, ohne zu persistieren – das ehrlichere Preflight.

Server-Side Apply, Field-Ownership

`apply --server-side` (GA seit v1.22) merged am API-Server. Ownership steht pro Feld in `metadata.managedFields`, jeder Writer traegt einen `--field-manager`.
Conflict (HTTP 409), wenn ein anderer Manager das Feld haelt: `--force-conflicts` uebernimmt die Ownership – bewusst einsetzen, das entzieht dem anderen Controller das Feld.

```
kubectl diff -f deploy.yaml
kubectl apply -f deploy.yaml --dry-run=server
kubectl apply --server-side --field-manager=ci -f deploy.yaml
kubectl apply --server-side --force-conflicts -f deploy.yaml
kubectl get deploy web --show-managed-fields -o yaml
```

Patch-Strategien

strategic, merge, json

strategic (Default): kennt Merge-Keys aus dem Schema – patcht z.B. `containers` per `name` statt die Liste zu ersetzen.
–type=merge (RFC 7386): simpel, aber Listen werden komplett ersetzt.
–type=json (RFC 6902): Op-Liste (`add`/`replace`/`remove`) mit praezisen Pfaden.
CRDs: kein strategic-Support – `merge` oder `json` verwenden.

```
kubectl patch deploy web -p '{"spec":{"replicas":4}}'
kubectl patch deploy web --type=merge \
  -p '{"metadata":{"labels":{"team":"pay"}}}'
kubectl patch deploy web --type=json -p \

    '[{"op":"replace","path":"/spec/template/spec/containers/0/im
```

Debug & Ephemeral Containers

Pod-Debug, distroless-tauglich

`kubectl debug -it pod/x --image=...` haengt einen Ephemeral Container an (stable seit v1.25) – Debug-Tools fuer Images ohne Shell. `--target` teilt den Prozess-Namespace des Ziel-Containers.
Ephemeral Containers sind nicht entfernbar; aufraeumen heisst Pod loeschen. Alternativ `--copy-to=dbg` + `--share-processes`: Kopie debuggen, Original bleibt unangetastet.

Node-Debug & Profile

`kubectl debug node/<n>` startet einen Pod auf dem Node, Host-Dateisystem unter `/host`.
`--profile` steuert die Rechte: `general` (Default seit v1.36), `baseline`, `restricted`, `netadmin`, `sysadmin`; `legacy` ist deprecated (Removal geplant fuer v1.39).

```
kubectl debug -it pod/api --image=nicolaka/netshoot
    --target=api
kubectl debug pod/api --copy-to=api-dbg \
  --share-processes -it --image=busybox
kubectl debug node/worker-3 -it --image=busybox
    --profile=sysadmin
chroot /host   # im Node-Debug-Pod
```

Rollout

status, history, undo

`rollout status` blockt bis Ready bzw. progressDeadline – das Gate fuer Pipelines.
`history` listet Revisionen (`--revision=N` zeigt Details; CHANGE-CAUSE kommt aus der Annotation `kubernetes.io/change-cause`).
`undo --to-revision=N` rollt gezielt zurueck. `restart` erzwingt ein neues Rollout (z.B. nach ConfigMap-Aenderung). Gilt fuer Deployments, DaemonSets, StatefulSets.

```
kubectl rollout status deploy/web --timeout=120s
kubectl rollout history deploy/web --revision=3
kubectl rollout undo deploy/web --to-revision=2
kubectl rollout restart deploy/web
kubectl rollout pause deploy/web   # und: resume
```

Auth, Events & Wait

can-i, whoami

`auth can-i` prueft RBAC vor dem Incident; `--as` / `--as-group` testet fremde Identitaeten (ServiceAccounts als `system:serviceaccount:<ns>:<sa>`).
`--list` zeigt alle Rechte im Namespace.
`auth whoami` (GA seit v1.28, SelfSubjectReview-API): UserInfo inkl. Gruppen, wie der API-Server sie nach Authentifizierung sieht.

events, top, wait

`kubectl events --for pod/x --types=Warning`: chronologisch, gefiltert – ergonomischer als `get events` + `--sort-by`.
`top pod|node` braucht den metrics-server.
`wait --for=condition=...`, `--for=delete` oder `--for=jsonpath=...`: deklaratives Warten mit `--timeout` statt sleep-Schleifen.

```
kubectl auth can-i create deploy -n prod
kubectl auth can-i --list -n prod
kubectl auth can-i get secrets --as
    system:serviceaccount:ci:builder
kubectl auth whoami
kubectl events --for pod/api --types=Warning
kubectl top pods --containers -n prod
kubectl wait --for=condition=Ready pod -l app=web
    --timeout=90s
```

Port-Forward, Proxy & cp

port-forward, proxy, cp

`port-forward` tunnelt ueber den API-Server – kein Ingress, kein NodePort noetig; `svc/x` waehlt einen Ready-Pod dahinter.
`proxy` legt die REST-API authentifiziert auf `127.0.0.1:8001` – zum Explorieren und fuer `curl`-Debugging.
`cp` braucht ein `tar`-Binary im Container – bei distroless nicht vorhanden.

```
kubectl port-forward svc/web 8080:80
kubectl port-forward pod/db 5432 --address 127.0.0.1
kubectl proxy --port=8001 &
curl localhost:8001/api/v1/namespaces/prod/pods
kubectl cp prod/api-6d5f7:/var/log/app.log ./app.log
```

Explain, Kustomize & Plugins

explain & API-Discovery

`explain deploy.spec.strategy`: Feld-Doku aus dem OpenAPI-Schema des Clusters – inkl. CRDs. **`--recursive`** zeigt den ganzen Teilbaum, **`--api-version`** pinnt die Version.
`api-resources` / **`api-versions`**: was der Cluster kann, inkl. Shortnames.

Kustomize (-k)

Kustomize ist in kubectl integriert: **`apply|diff|delete -k <dir>`** rendert Base + Overlay und wendet an; **`kubectl kustomize <dir>`** rendert nur. Overlays statt kopierter YAML-Varianten pro Stage.

Plugins & Krew

Jedes Binary **`kubectl-<name>`** im **`PATH`** wird Subcommand; **`kubectl plugin list`** zeigt sie.
Krew (kubernetes-sigs) ist der Plugin-Manager: **`krew search|install|upgrade`**, ueber 200 Plugins im Index.

```
kubectl explain deploy.spec.strategy --recursive
kubectl api-resources --namespaced=false
kubectl kustomize overlays/prod | kubectl diff -f -
kubectl apply -k overlays/prod
kubectl krew install ctx ns tree
```

Anti-Patterns

Was du nicht tun solltest

`edit` in Prod: unversionierter Live-Drift; der naechste Apply/GitOps-Sync ueberschreibt still. Aenderung ins Repo, dann **`apply`**.
`delete --force --grace-period=0`: entfernt nur das API-Objekt, der Prozess kann am Node weiterlaufen – bei StatefulSets Risiko doppelter Identitaeten.
Apply-Modi mischen: client-side und server-side auf demselben Objekt erzeugt managedFields-Konflikte – pro Objekt einen Modus festlegen.
`:latest` + **`rollout restart`** als Deployment- Ersatz: nicht reproduzierbar, kein Rollback-Ziel – immutable Tags.
Blind im falschen Kontext: destruktive Kommandos ohne **`config current-context`**-Check; Prod-Kontexte explizit benennen und im Prompt markieren.
`scale` gegen HPA: der Autoscaler setzt den Wert zurueck – Min/Max am HPA aendern statt manuell zu skalieren.

kubectl-cheatsheet.de

kubectl Cheatsheet von OMNI52

Weiterfuehren, nicht selbst neu erfinden

Kubernetes-Betrieb & Day-2-Operations

OMNI52™ hilft Plattform-Teams, Kubernetes-Betrieb fuer regulierte Enterprise-Umgebungen sauber aufzustellen.

Server-Side-Apply-Strategie mit klarer Field-Ownership zwischen CI, Controllern und Operatoren, RBAC-Audits mit `can-i`-Testmatrix, Debug-Runbooks fuer distroless Workloads, Rollout-Gates in der Pipeline. Output landet als GitOps-Repo bei euch im Cluster: Runbooks, Kustomize-Overlays, 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) · helm-cheatsheet.de (Helm Charts) · kyverno-cheatsheet.de (Policy-as-Code) · rke2-cheatsheet.de (RKE2-Distribution).