OMNI52 Flux

Architektur & GitOps-Toolkit

Pull statt Push

Flux zieht den deklarierten Zustand aus Git, OCI-Registries oder S3-Buckets und wendet ihn kontinuierlich an. Quelle = Wahrheit: was nicht im Repo steht, gehoert nicht in den Cluster. Jede Ressource traegt ihr eigenes `interval`; Aenderungen laufen ueber Git-Push, nicht ueber `kubectl`.

Controller & CRDs

source-controller: GitRepository, OCIRepository, HelmRepository, HelmChart, Bucket.
kustomize-controller: Kustomization (inkl. SOPS-Decryption).
helm-controller: HelmRelease.
notification-controller: Provider, Alert, Receiver (Webhooks).
image-reflector-/image-automation-controller: ImageRepository, ImagePolicy, ImageUpdateAutomation.

```
flux check                # Prereqs + Controller-Status
flux get all -A           # Status aller Flux-Ressourcen
kubectl get fluxcd -A     # alle Flux-CRs (Resource-Kategorie)
```

Bootstrap

flux bootstrap

Installiert die Controller nach `flux-system`, pusht die eigenen Manifeste unter `--path` ins Repo und synct sich ab dann selbst. Idempotent: erneuter Lauf mit gleichen Argumenten ist der Upgrade-Pfad. Danach laeuft jede Aenderung am Cluster – inkl. Flux-Upgrades – per Git-Push.

```
export GITHUB_TOKEN=<pat>
flux bootstrap github \
  --owner=example-org --repository=fleet \
  --branch=main --path=clusters/prod \
  --token-auth
```

Repo-Layout pro Cluster

`--path=clusters/<name>` gibt jedem Cluster einen eigenen Einstiegspunkt im selben Repo. Infrastruktur (CRDs, Operatoren, Ingress) und Apps als getrennte Kustomizations verketten – `dependsOn` erzwingt die Reihenfolge.

Sources

GitRepository & OCIRepository

`url` + `interval` + `ref`: Git per `branch`, `tag`, `semver` oder `commit`; OCI per `tag`, `semver` oder `digest`. Auth via `secretRef`, tokenlos via `provider` (z.B. `github`, `azure`). OCIRepository liefert Manifeste als OCI-Artefakt inkl. Signatur-Pruefung (`verify`: cosign/notation).

HelmRepository & Bucket

HelmRepository: klassischer HTTP/S-Index oder `type: oci` fuer Charts aus OCI-Registries. Bucket: S3-kompatibler Objektspeicher als Quelle. Jede Source produziert ein versioniertes Artefakt, auf das Kustomization/HelmRelease per `sourceRef` zeigen.

```
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata: {name: platform, namespace: flux-system}
spec:
  interval: 1m
  url: https://github.com/example-org/platform
  ref: {branch: main}   # oder tag/semver/commit
  secretRef: {name: git-auth}
---
apiVersion: source.toolkit.fluxcd.io/v1
kind: OCIRepository
metadata: {name: manifests, namespace: flux-system}
spec:
  interval: 5m
  url: oci://ghcr.io/example-org/manifests
  ref: {semver: "≥1.0.0 <2.0.0"}
```

Kustomization

Spec-Kern

`path`: Verzeichnis im Source-Artefakt. `prune: true`: Garbage-Collection geloeschter Manifeste. `sourceRef`: GitRepository, OCIRepository oder Bucket. `retryInterval` steuert Fehler-Retries getrennt vom regulaeren `interval`; `serviceAccountName` erzwingt Impersonation (RBAC-Grenze), `targetNamespace` zwingt alle Objekte in einen Namespace.

```
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata: {name: apps, namespace: flux-system}
spec:
  interval: 10m
  retryInterval: 2m
  path: ./apps/production
  prune: true             # Garbage Collection
  sourceRef: {kind: GitRepository, name: platform}
  dependsOn:
    - name: infrastructure
  wait: true              # Health aller Objekte
  timeout: 5m
  postBuild:
    substituteFrom:
      - kind: ConfigMap
        name: cluster-vars
```

Health & Reihenfolge

`wait: true` prueft die Health aller applizierten Objekte, `healthChecks` gezielt einzelne (Deployment, HelmRelease, …). `dependsOn` gated nachgelagerte Kustomizations, bis die Abhaengigkeit Ready ist – Standard-Muster: infrastructure vor apps.

postBuild-Substitution

`postBuild.substituteFrom` ersetzt `${var}`-Platzhalter aus ConfigMaps/-Secrets – cluster-spezifische Werte (Domain, Zone, Replicas) ohne Overlay-Wildwuchs. `optional: true` toleriert fehlende Quellen.

HelmRelease

Chart-Referenz

`chart.spec` + `sourceRef` auf HelmRepository/GitRepository – oder `chartRef` direkt auf OCIRepository/HelmChart. `version` nimmt SemVer-Ranges (`6.x`, `≥0.2.0 <1.0.0`): Flux waehlt automatisch die hoechste passende Version – kontrolliertes Auto-Update pro Range.

```
apiVersion: helm.toolkit.fluxcd.io/v2
kind: HelmRelease
metadata: {name: podinfo, namespace: apps}
spec:
  interval: 30m
  chart:
    spec:
      chart: podinfo
      version: "≥6.0.0 <7.0.0"  # SemVer-Range
      sourceRef:
        kind: HelmRepository
        name: podinfo
        namespace: flux-system
  install:
    remediation: {retries: 3}
  upgrade:
    remediation: {retries: 3, strategy: rollback}
  driftDetection: {mode: enabled}
  valuesFrom:
    - kind: Secret
      name: podinfo-values
  values:
    replicaCount: 2
```

Remediation & Drift

`install.remediation.retries` / `upgrade.remediation.retries`: automatische Wiederholung fehlgeschlagener Releases; `strategy`: `rollback` oder `uninstall`. `driftDetection.mode: enabled` erkennt und korrigiert manuelle Aenderungen am Release-Inventar (`warn` meldet nur). `test.enable` fuehrt Helm-Tests nach Install/Upgrade aus.

Reconciliation & Betrieb

Intervalle & Trigger

Jede Ressource reconciled periodisch (`interval`); `flux reconcile … --with-source` stoesst sofort an, inklusive frischem Source-Fetch. Push statt Polling: Receiver (notification-controller) nimmt Webhooks von GitHub/GitLab an; Provider/Alert melden Events an Chat- oder Webhook-Endpunkte.

suspend / resume

`flux suspend` haelt die Reconciliation an (Incident-Freeze, Wartungsfenster), `flux resume` nimmt sie wieder auf. Suspendierte Objekte driften still – Freeze immer zeitlich begrenzen.

```
flux reconcile kustomization apps --with-source
flux reconcile source git platform
flux suspend kustomization apps
flux resume kustomization apps
flux get kustomizations --watch
```

Image Automation

Drei CRDs

ImageRepository scannt Registry-Tags, ImagePolicy waehlt den Ziel-Tag (`semver.range`, alphabetisch/numerisch, `filterTags` mit Pattern/Extract), ImageUpdateAutomation schreibt den neuen Tag per Commit zurueck ins Repo (Strategie: Setters-Marker im Manifest).

```
apiVersion: image.toolkit.fluxcd.io/v1
kind: ImagePolicy
metadata: {name: podinfo, namespace: flux-system}
spec:
  imageRepositoryRef: {name: podinfo}
  policy:
    semver: {range: 6.x}
# Setters-Marker im Ziel-Manifest:
#  image: ghcr.io/example-org/podinfo:6.9.0 \
#    # {"$imagepolicy": "flux-system:podinfo"}
```

Multi-Tenancy

Lockdown: Flags am Controller

`--no-cross-namespace-refs=true` auf kustomize-, helm-, notification- und image-Controllern: Tenants koennen keine Sources ausserhalb ihres Namespaces referenzieren. `--default-service-account=default`: Kustomizations/HelmReleases ohne `serviceAccountName` laufen mit dem unprivilegierten Default-Account des Tenant-Namespaces – Rechte kommen ausschliesslich per RBAC.

```
# clusters/prod/flux-system/kustomization.yaml
patches:
  - patch: |
      - op: add
        path: /spec/template/spec/containers/0/args/-
        value: --no-cross-namespace-refs=true
    target:
      kind: Deployment
      name:
      "(kustomize-controller|helm-controller|notification-controll
```

Secrets: SOPS & age

Verschluesselt ins Repo

Secrets liegen SOPS-verschluesselt im Git; der kustomize-controller entschluesselt beim Apply (`decryption.provider: sops`). Private Key als Secret `sops-age` in `flux-system` (Key-Datei muss auf `.agekey` enden). Nur `data`/`stringData` verschluesseln – die Manifest-Struktur bleibt diff-bar.

```
age-keygen -o age.agekey
cat age.agekey | kubectl create secret generic \
  sops-age --namespace=flux-system \
  --from-file=age.agekey=/dev/stdin
sops --age=<public-key> --encrypt \
  --encrypted-regex '^(data|stringData)$' \
  --in-place basic-auth.yaml
# Kustomization-Ausschnitt:
spec:
  decryption:
    provider: sops
    secretRef: {name: sops-age}
```

Diagnose (flux CLI)

Vom Symptom zur Quelle

`flux trace` zeigt fuer ein beliebiges Cluster-Objekt, welche Kustomization/HelmRelease und welche Source es verwaltet – der schnellste Einstieg, wenn unklar ist, woher ein Objekt kommt.

Status & Events

`flux get` pro Kind oder `all`, `--watch` fuer Live-Status. `flux events` filtert Kubernetes-Events auf Flux-Ressourcen (`--for Kind/name`). `flux logs` aggregiert die Controller-Logs (`--level=error`).

```
flux check
flux get all -A
flux trace -n apps deployment my-app
flux events --for Kustomization/apps
flux logs --level=error --all-namespaces
```

Anti-Patterns

Was du nicht tun solltest

kubectl apply am Repo vorbei: die naechste Reconciliation ueberschreibt den Hotfix still. Aenderungen gehoeren ins Repo; Notfall-Stopp ist `flux suspend`.
prune: false als Dauerzustand: geloeschte Manifeste bleiben als Waisen im Cluster stehen – Drift, den niemand mehr sieht.
Eine Mega-Kustomization: CRDs, Operatoren und deren Custom Resources in einem Apply racen gegeneinander. Aufteilen + `dependsOn`/`healthChecks`.
latest-Tags automatisieren wollen: ImagePolicy braucht sortierbare Tags (SemVer, numerisch) – ein mutables `latest` ist nicht selektierbar.
Klartext-Secrets im Repo: SOPS/age gehoert vor das erste Secret-Commit ins Setup, nicht danach.
suspend vergessen: suspendierte Ressourcen driften unbemerkt weiter – `flux get` zeigt den Suspended-Status, nach Wartung `resume`.
Multi-Tenant ohne Lockdown: ohne `--no-cross-namespace-refs` und Default-ServiceAccount greift jeder Tenant auf fremde Sources zu.

flux-cheatsheet.de

Flux Cheatsheet von OMNI52

Weiterfuehren, nicht selbst neu erfinden

GitOps-Rollout & Flux-Betrieb

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

Repo-Struktur und Multi-Tenancy-Design mit sauberen RBAC-Grenzen, Bootstrap mit SOPS/age-Secret-Management von Tag eins, HelmRelease-Remediation statt naechtlicher Rollback-Einsaetze, Image-Automation mit signierten Artefakten. Output landet als GitOps-Repo bei euch im Cluster: Kustomizations, Runbooks, Diagnose-Skripte. Euer Team operiert weiter, nicht wir.

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

Verwandte Cheatsheets

Ebenfalls von OMNI52: argocd-cheatsheet.de (GitOps mit Argo CD) · helm-cheatsheet.de (Helm Charts) · terraform-cheatsheet.de (Infrastruktur as Code) · rke2-cheatsheet.de (RKE2-Distribution).

Lizenz & Weiterverteilung

Trademarks