Declarative Delivery on Kubernetes

GitOps • CD Workflows • Helm • Kustomize • Operators • Controllers

- Audience: Advanced / Intermediate (beginner-accessible)
- Focus: Theory-first, vendor-neutral patterns & architecture

Agenda

1. Declarative GitOps (Argo CD, Flux CD)
2. Continuous Deployment Workflows with Kubernetes
3. Helm (package management for K8s)
4. Kustomize (overlay configurations)
5. Operators & CRDs (Custom Resource Definitions)
6. Extending Kubernetes with Controllers

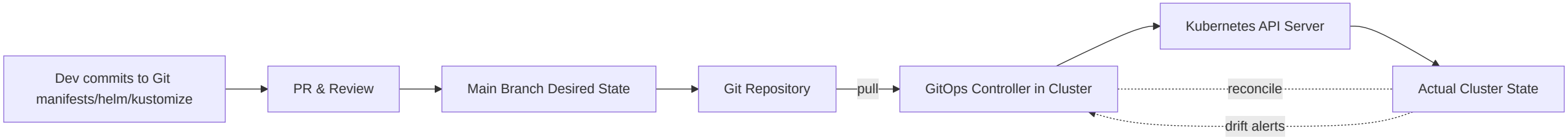
1 Declarative GitOps

Core Principles

- Single source of truth: Git holds the desired state of clusters/apps
- Pull-based reconciliation: In-cluster agents compare desired vs. actual and reconcile continuously
- Change via PRs: Reviews, audits, rollbacks through Git history
- Drift detection: Agents alert and correct out-of-band changes

> GitOps = declarative config + reconciliation loop + version-controlled ops.

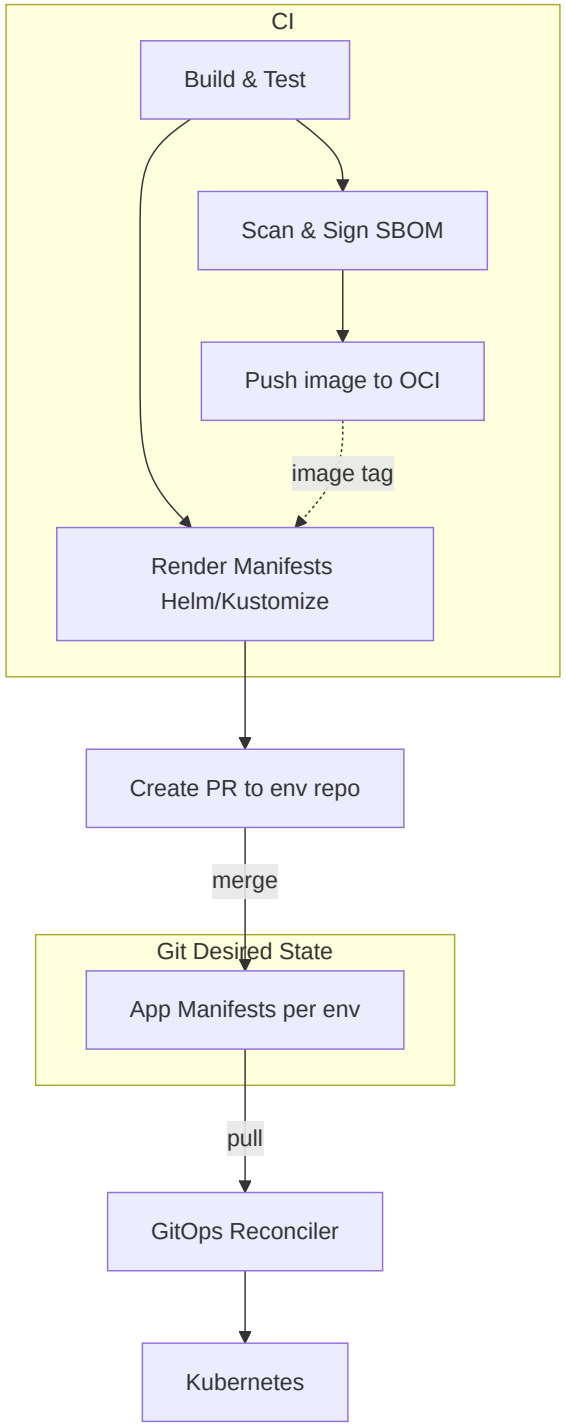
Aspect	Argo CD	Flux CD
Model	App-of-Apps, Application CRD	GitRepository/Kustomization/HelmRelease CRDs
UX	Rich UI, diffing, health/status	CLI/CRD-focused, Git-native ops
Sources	Git, Helm repos, OCI	Git, Helm, OCI, Buckets
Policies	Sync windows, waves, hooks	Depends-on graph, reconciliation intervals
Progressive Delivery	Argo Rollouts (add-on)	Flagger (add-on)


Dev commits to Git manifests/helm/kustomize
PR & Review
Main Branch Desired State
Git Repository
pull
GitOps Controller in Cluster
Kubernetes API Server
reconcile
Actual Cluster State
drift alerts

2 Continuous Deployment Workflows with Kubernetes

CI vs. CD Separation

- CI: Build, test, scan, sign, publish artifacts + SBOM/provenance
- CD: Reconcile cluster to manifests (GitOps) or push to API (imperative CD)
- Promotion: Environments via branches/folders/tenants; PR-based gates


CI
Build & Test
Scan & Sign SBOM
Push image to OCI
image tag
Render Manifests
Helm/Kustomize
Create PR to env repo
merge
Git Desired State
App Manifests per env
pull
GitOps Reconciler
Kubernetes

Release Strategies (Theory)

- Rolling update: Default; balance surge vs. unavailable
- Blue/Green: Two environments; switch traffic atomically
- Canary: Small % traffic to new version; promote on SLOs
- Shadow: Mirror traffic without serving responses

> Progressive delivery integrates metrics & SLOs into promotion decisions.

3 Helm (Package Management for K8s)

Concepts & Model

- Chart: Package of templates + values + metadata
- Values: Configuration inputs to render manifests
- Release: An installed, versioned chart instance in a cluster
- Repositories: Chart distribution; also supports OCI registries

Helm Best-Practice (Theory)

- Values contract: Documented, minimal; avoid leaking low-level knobs
- Immutability: Pin chart and app versions; track in Git
- Separation: Keep “ops defaults” vs. “app overrides” clear
- Security: Verify provenance/signatures for charts; prefer OCI where possible

4 Kustomize (Overlay Configurations)

Concepts & Model

- Base: A set of plain manifests (no templates)
- Overlay: Patches/transformations (env-specific) applied to the base
- Resources: Composition by reference; declarative transformations

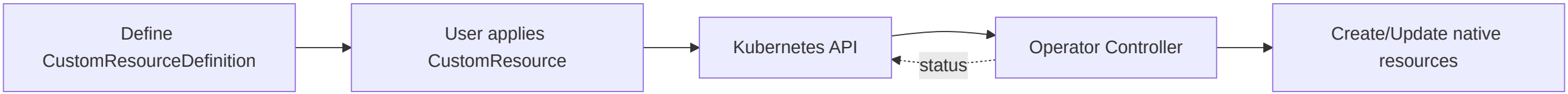
Kustomize Building Blocks

- Common: namePrefix/suffix, labels/annotations, image tags, replicas
- Patches: Strategic merge patches, JSON6902 patches
- Generators: ConfigMap/Secret from literals/files (with content hashing)
- Plugins (optional): For advanced transforms (guard usage)

5 Operators & CRDs

Extending the API

- CRD (CustomResourceDefinition): Add new kinds to the Kubernetes API
- Operator pattern: A controller that encodes human operational knowledge for a CRD (reconcile actual to desired)
- Use cases: Databases (Backups, failover), messaging, ML pipelines, platform abstractions


Define CustomResourceDefinition
User applies CustomResource
Kubernetes API
status
Operator Controller
Create/Update native resources

Operator Maturity (Theory)

- Basic: Installs app components (day-1)
- Lifecycle: Handles upgrades, config drift, scale (day-2)
- Ops-aware: Backups, restore, failover, metrics, self-healing
- Autonomous: Policy-driven decisions with minimal human input

6 Extending Kubernetes with Controllers

Controller Pattern

- Watch → Diff → Act: Observe resources, compute desired actions, apply changes
- Level-based reconciliation: Desired outcomes, not event-by-event scripts
- Idempotency: Re-running actions yields the same end state

Choosing Between Helm, Kustomize, Operators, GitOps

- Helm: Packaging + params for apps you don’t own or reusable modules
- Kustomize: Overlay & composition for your manifests (env differences)
- Operators: Domain automation for stateful/complex systems
- GitOps: Delivery & drift management for everything declarative

> Often combined: Helm chart rendered → committed to Git → deployed by GitOps; Kustomize overlays apply per environment.

Patterns & Anti-Patterns (Summary)

Do:

- Keep desired state in Git; reconcile in-cluster (pull)
- Use immutable tags + SBOMs; automate image updates safely
- Keep values/overlays minimal and documented; validate with policies
- Encode operational runbooks in Operators where justified

Avoid:

- Pushing manifests ad-hoc to API as a standard practice
- Hidden template logic that obscures intent
- Overusing Operators for simple stateless apps
- Mixing env-specific hacks into base manifests

References & Further Reading

- GitOps: Argo CD, Flux CD
- Helm Charts & OCI Artifacts; Chart Provenance
- Kustomize overlays & patch strategies
- Kubebuilder, Operator SDK, controller-runtime
- Admission control & policy (OPA/Gatekeeper, Kyverno)