// GITOPS REPO STRUCTURES AND PATTERNS

Johannes Schnatterer, Cloudogu GmbH

@schnatterer@floss.social in/jschnatterer @jschnatterer

Version: 202311081800-ad019bf

Categories of patterns

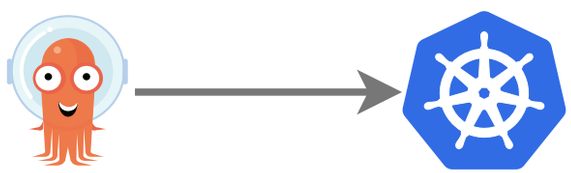
AKA strategies, models, approaches, best practices

- **Operator deployment**: GitOps operators ↔ Clusters/Namespaces
- **Repository**: How many repos?
- **Promotion**: How to model environments/stages?
- **Wiring**: Bootstrapping operator, linking repos and folders

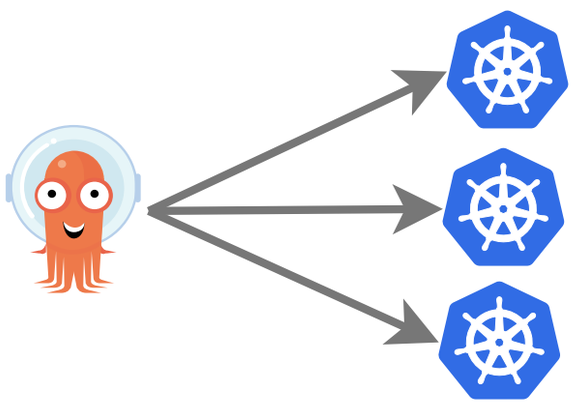
GitOps Operator deployment patterns

How many GitOps operators per cluster?

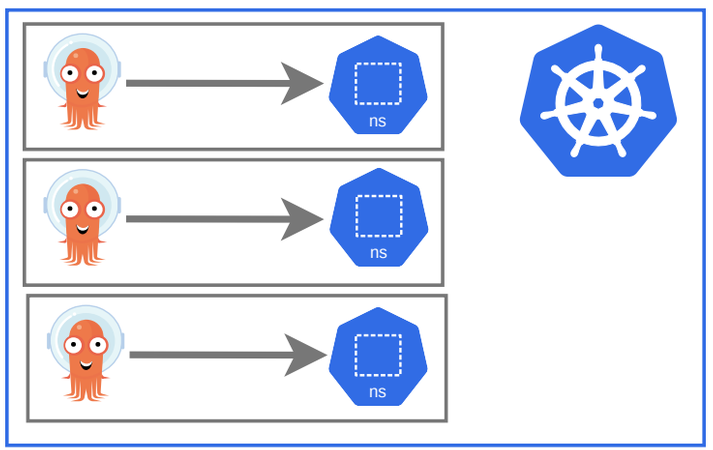
Instance per Cluster



Hub and Spoke



Instance per Namespace



Repository patterns

How many config repos?

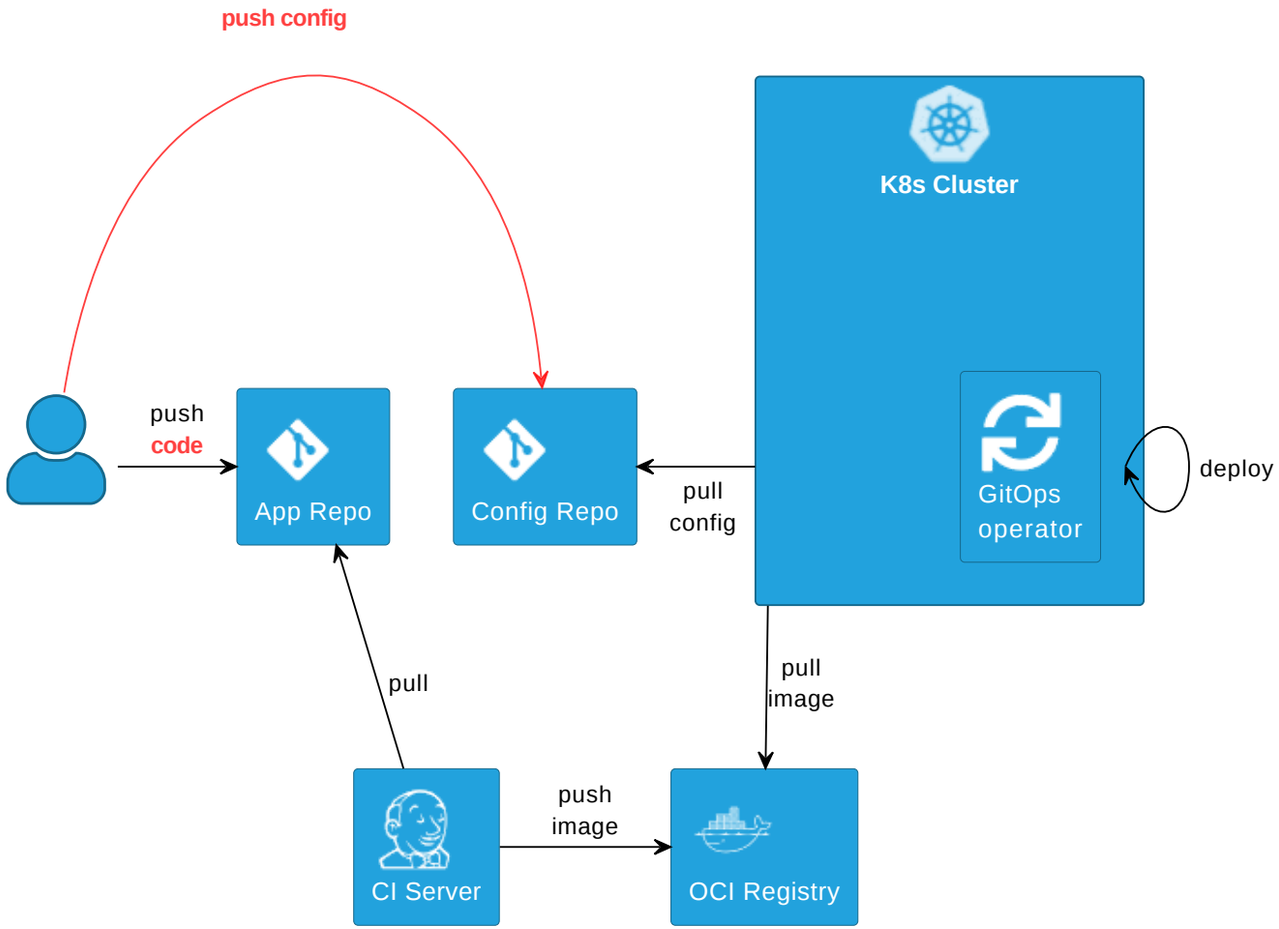
- **Monorepo** (opposite: polyrepo)
- **Repo per Team** / Tenant
- **Repo per App**
  - Repo Separation
  - Config replication
  - Repo pointer
  - Config Split
- **Repo per environment** 🕒

💡 Can be mixed 🏗️

Repository types

	Config repo	App repo
Content	Config/Manifests/YAMLs (IaC)	Application source code
Synonyms	• GitOps repo • Infra repo • Environment repo • Payload repo	• Source code repo • Source repo
Example		

Repo Separation



Recommendation: Keep config separate from code

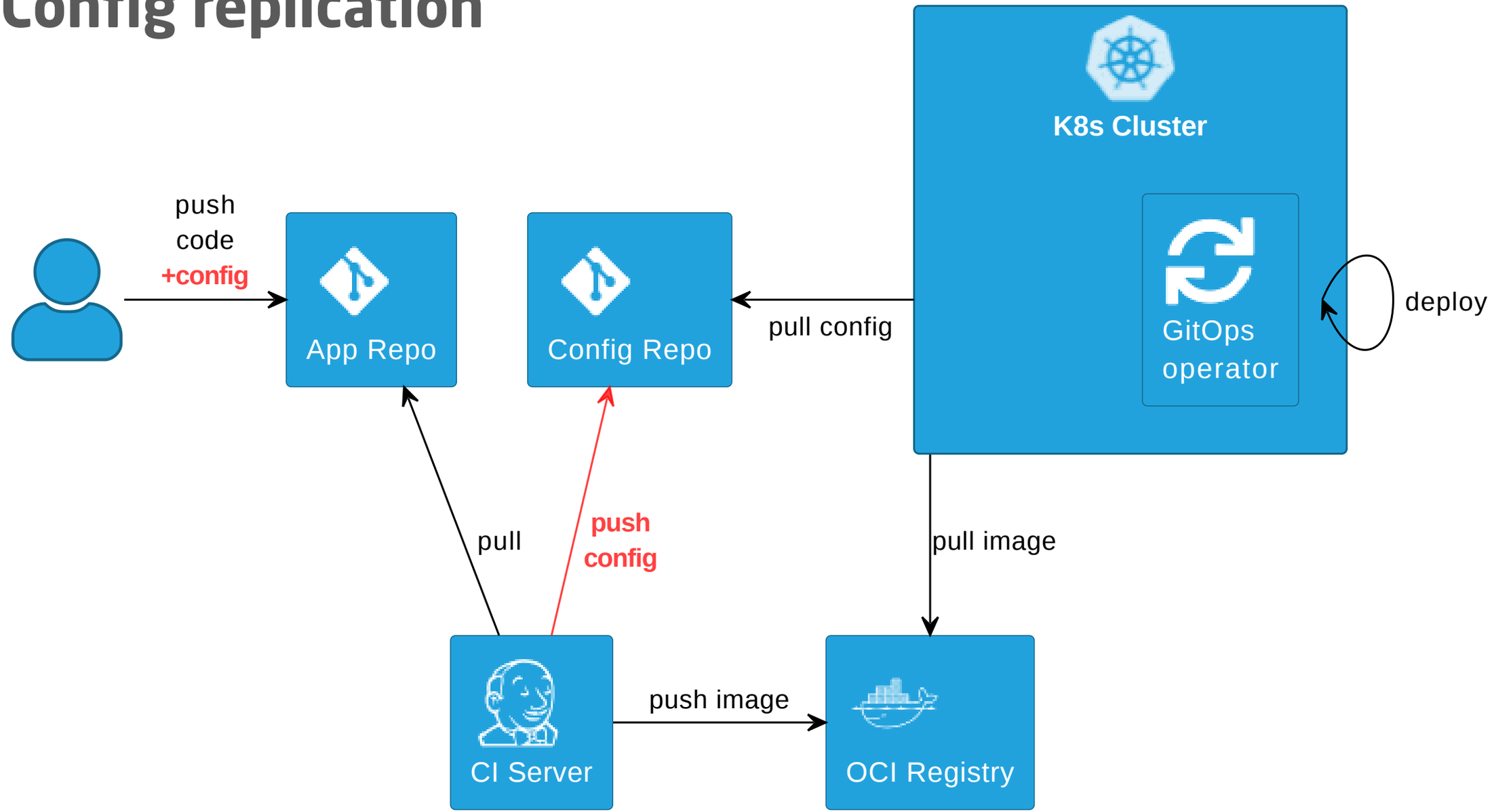
argo-cd.readthedocs.io/en/release-2.8/user-guide/best_practices

Disadvantages

- Separated maintenance & versioning of app and infra code
- Review spans across multiple repos
- Local dev more difficult
- No static code analysis on config repo

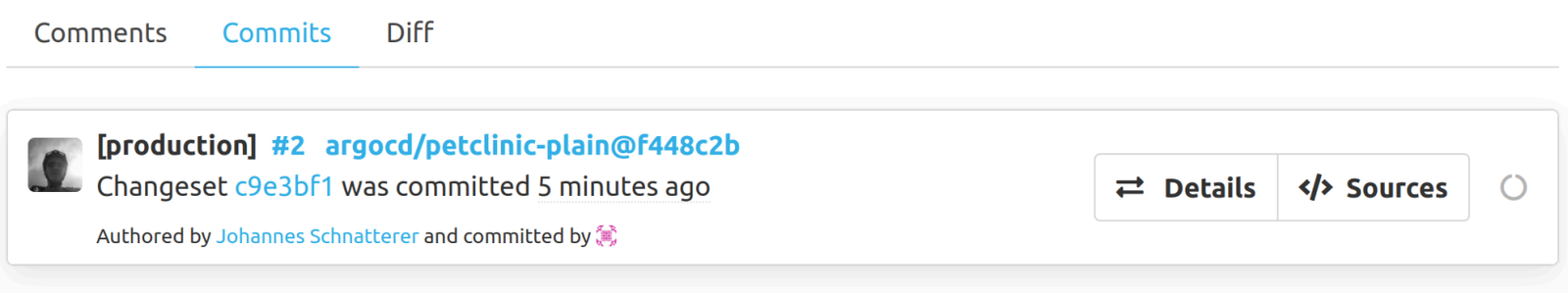
How to avoid those?

Config replication



Advantages

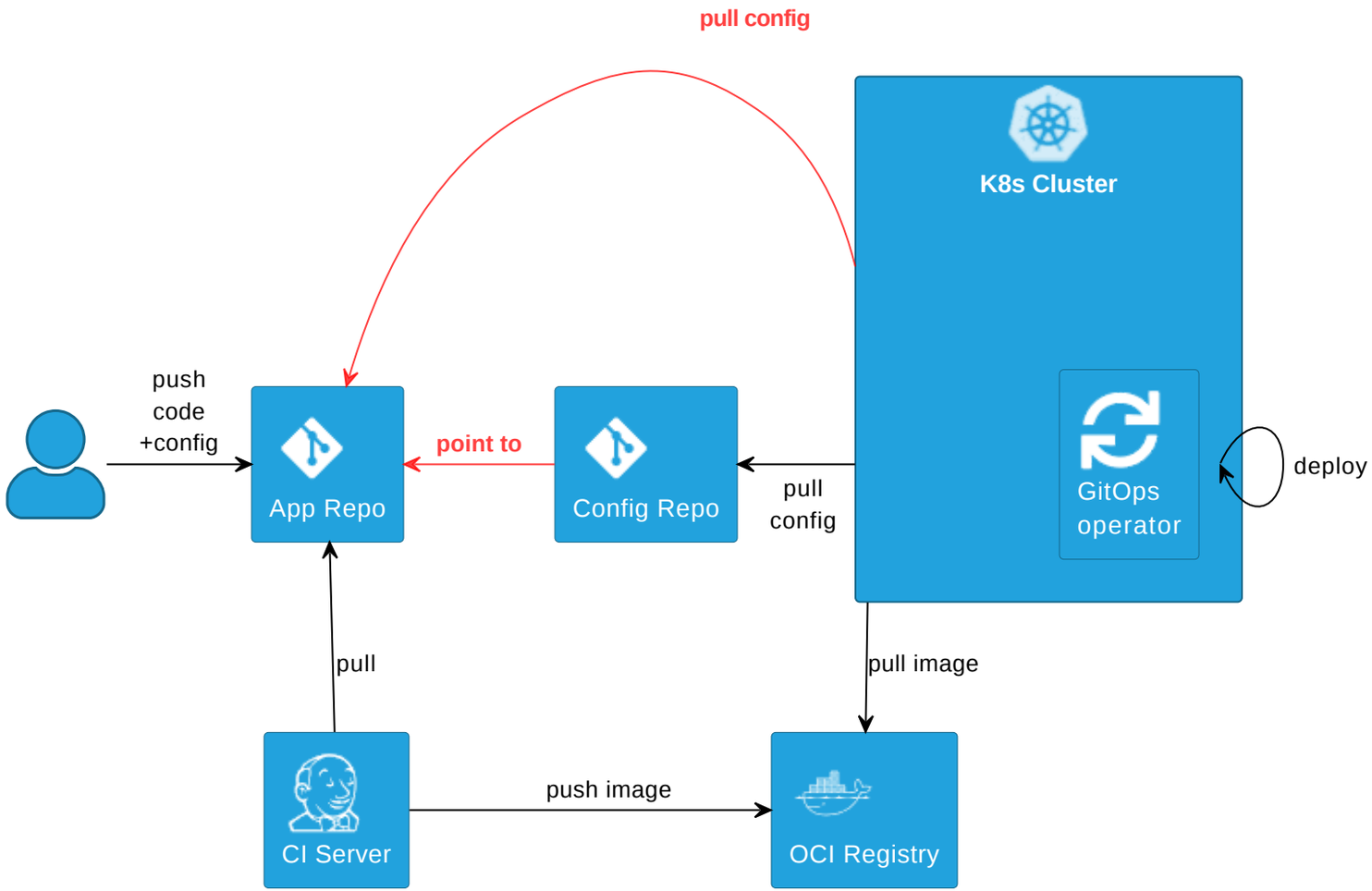
- Single repo for development: higher efficiency
- Shift left: static code analysis + policy check on CI server,
  e.g. yamlint, kubeconform, helm lint, conftest, security scanners
- Automate config update (image tag + PR creation) 🕓
- Simplify review by adding info to PRs



Disadvantages

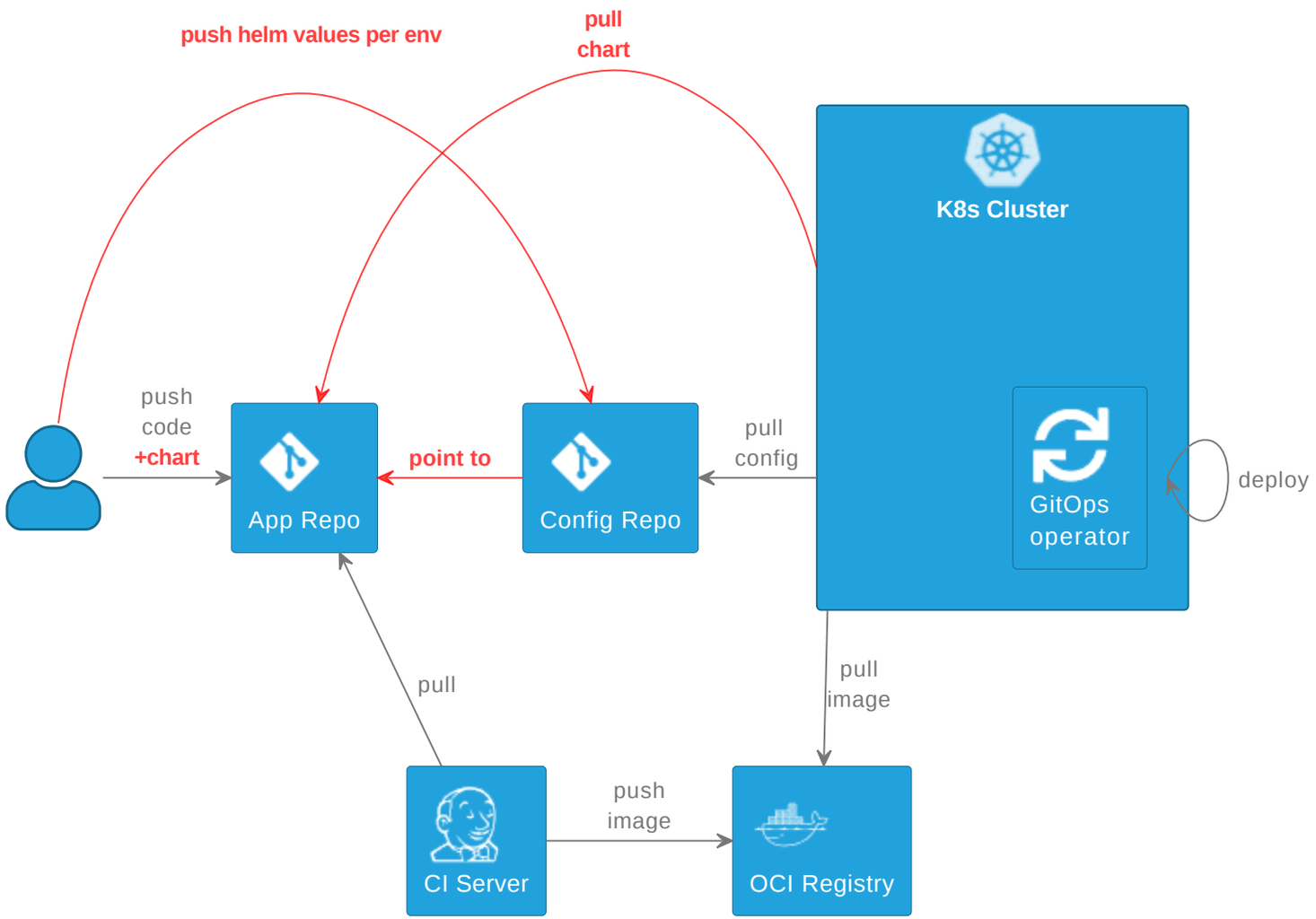
- Complexity in CI pipelines
  - ➡️ Recommendation: Use a plugin or library, e.g.
  - cloudogu/gitops-build-lib
- Redundant config (app repo + config repo)

Avoid Redundancy: Repo pointer



e.g. fluxcd.io/flux/guides/repository-structure

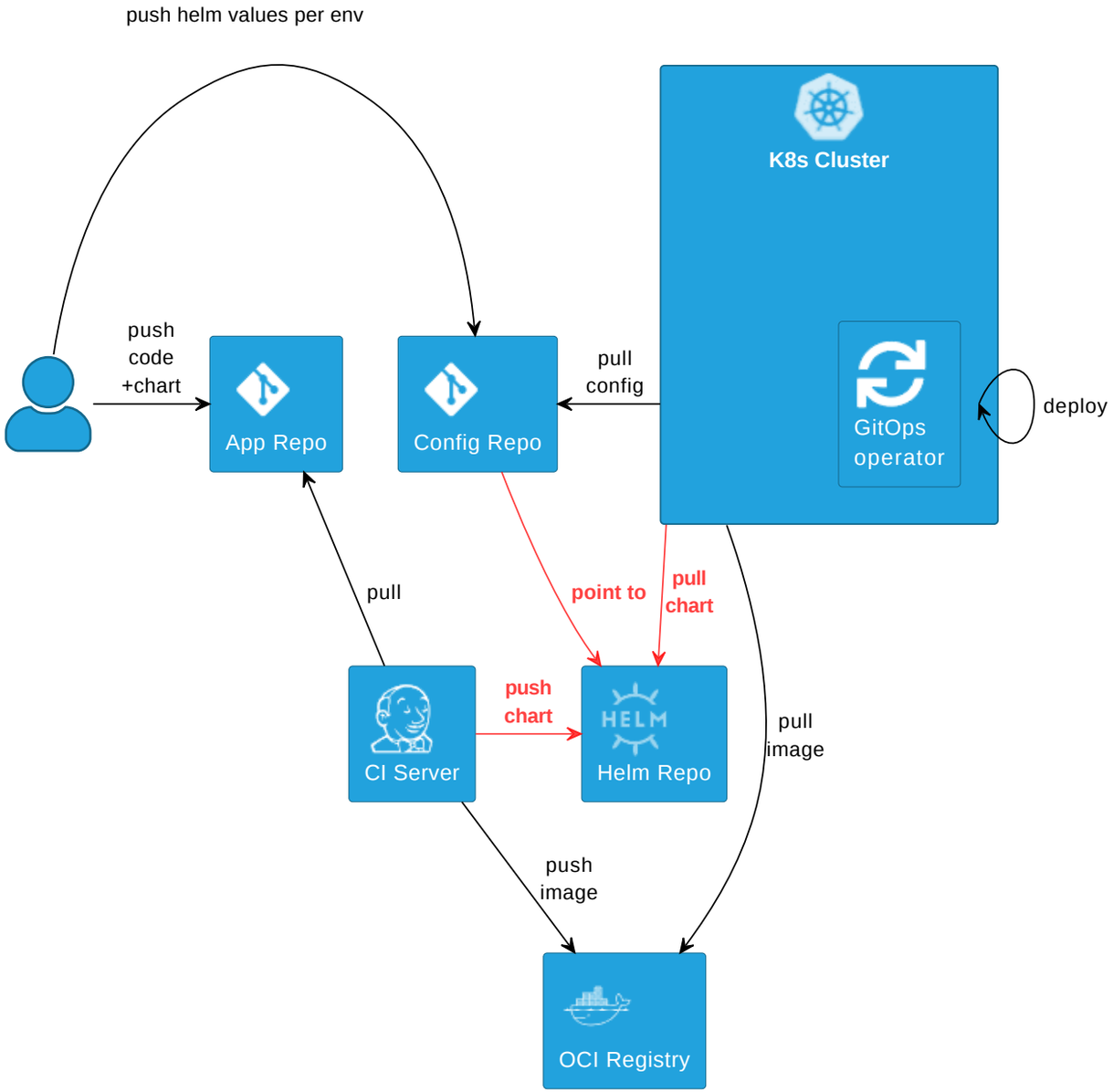
Middle ground: Config Split

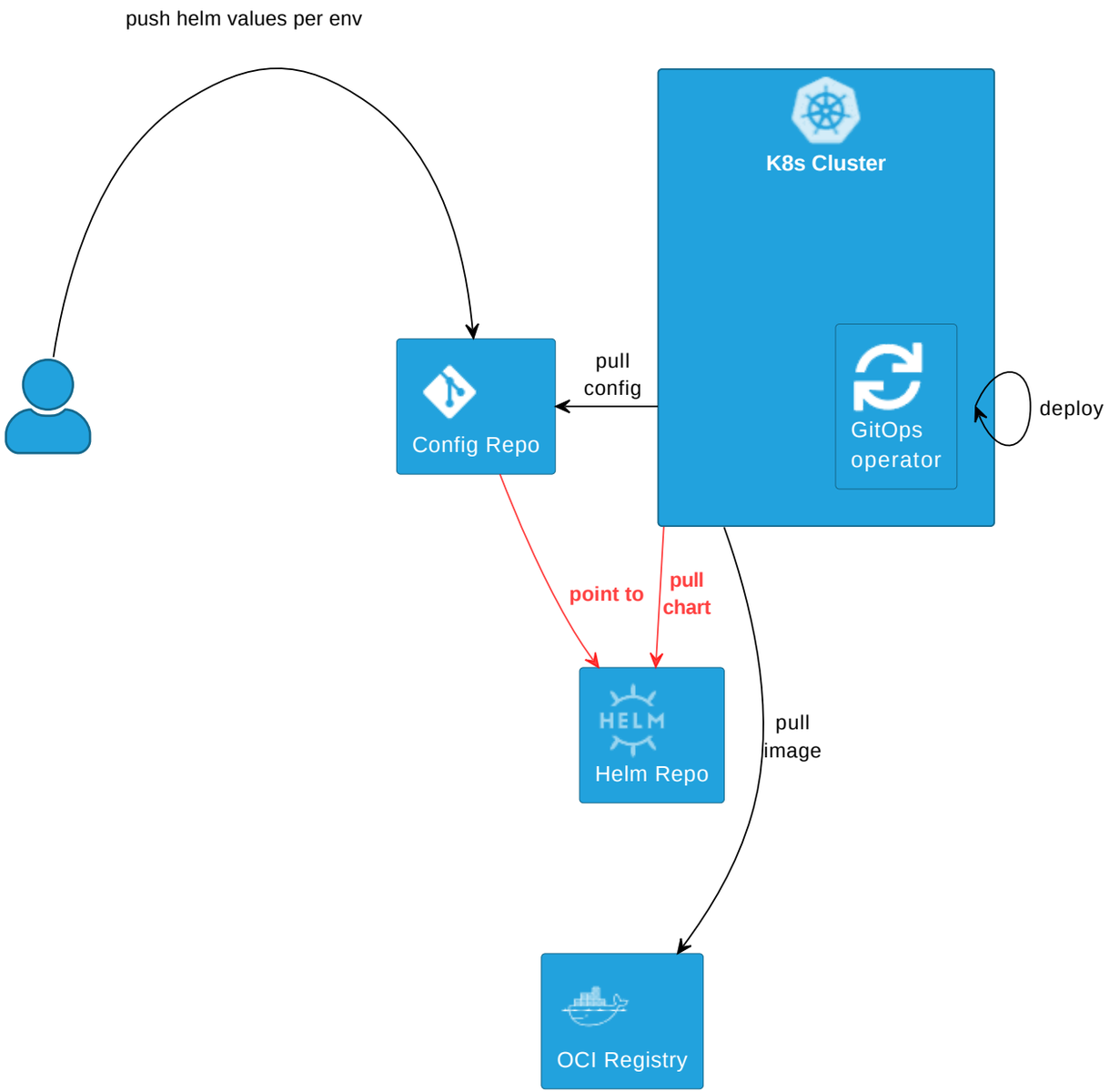


👉 HELM example

💡 Also works with K

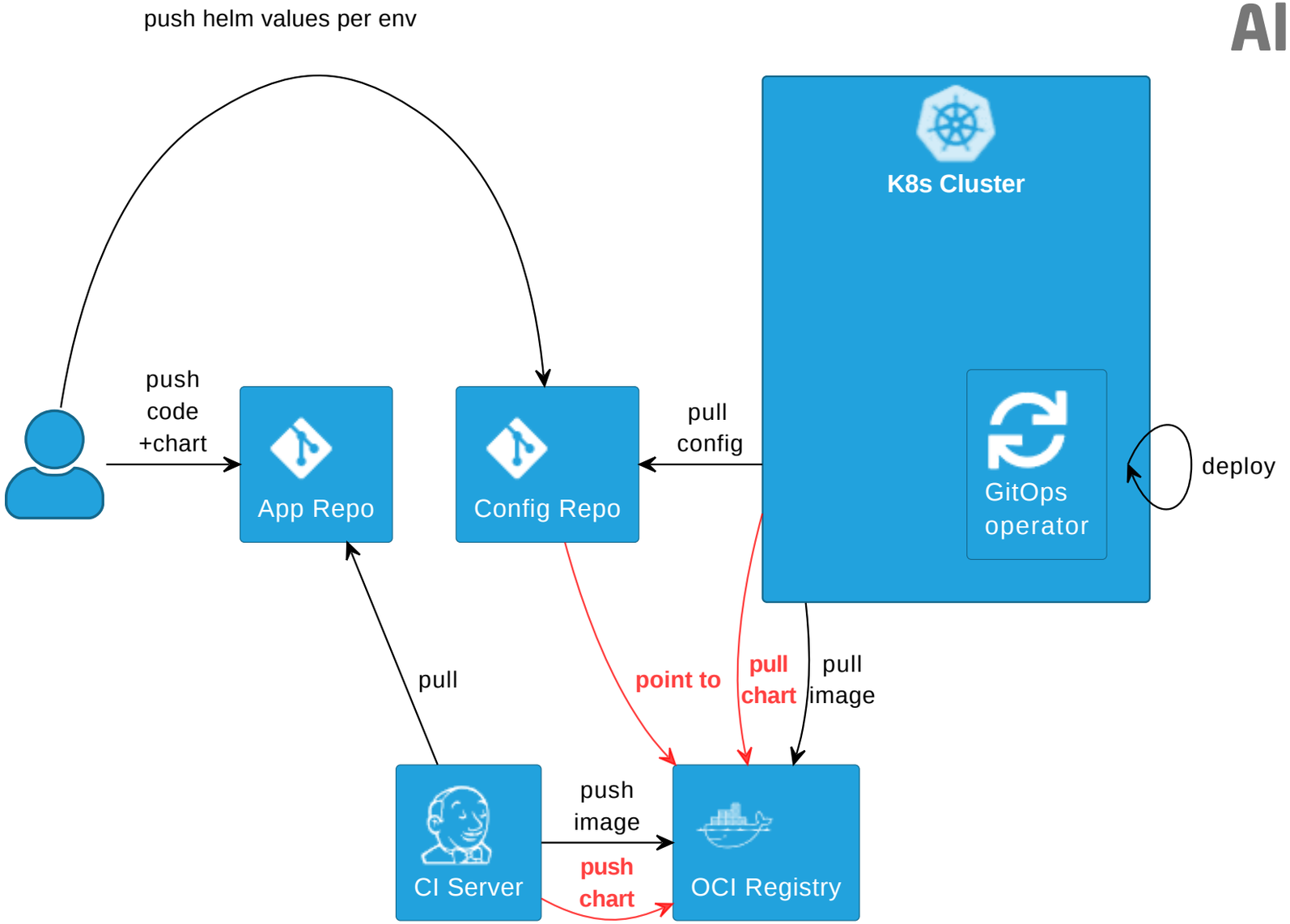
Alternative: Helm repo



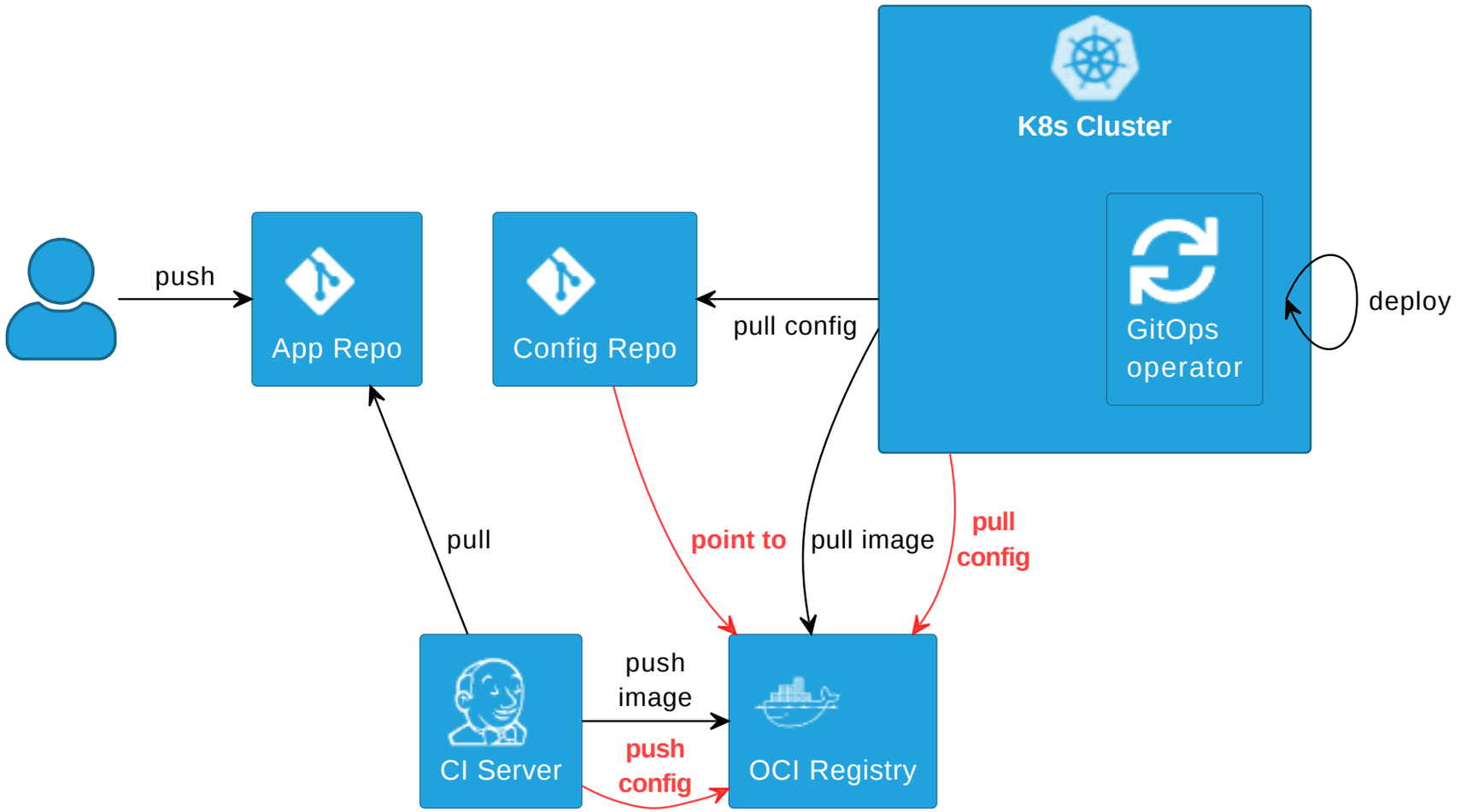


💡 Same pattern for 3rd-party apps

Alternative 2: Helm in OCI



Alternative 3: OCI artifacts



CUE

fluxcd.io/flux/cheatsheets/oci-artifacts

Promotion patterns

How to model environments AKA stages?

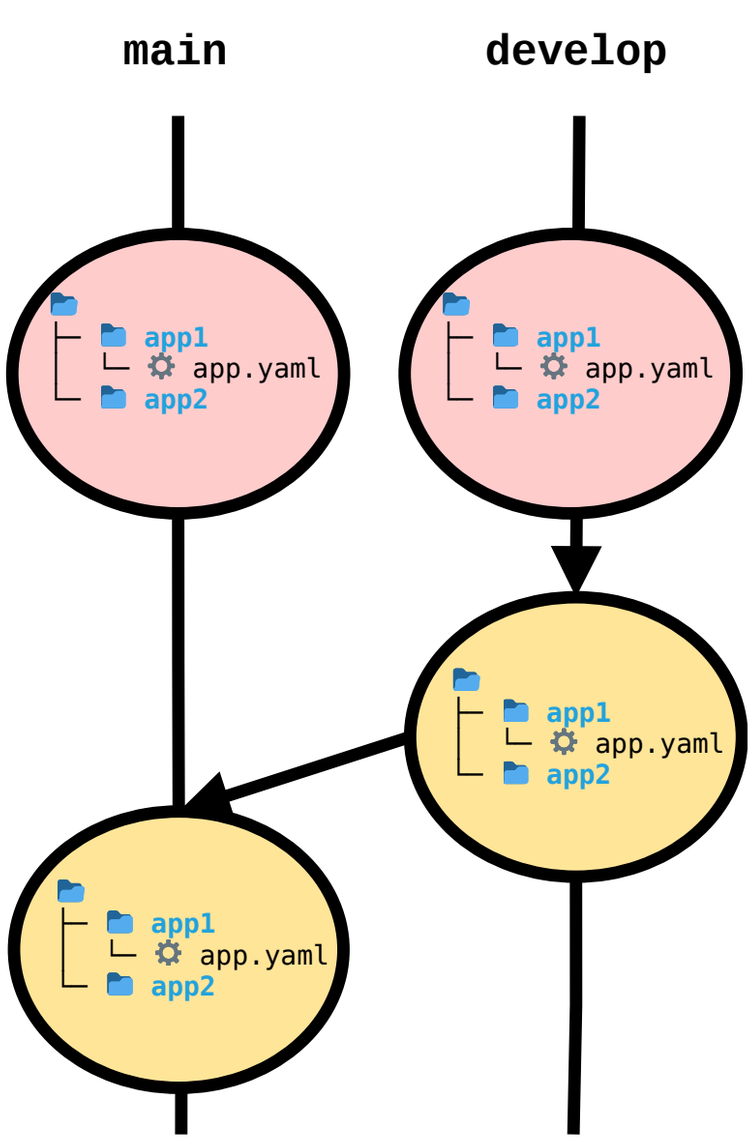
- **Branch per environment**
- **Folder/Directory per environment**
- **Repo per environment** (edge case)
- 🔥 **Preview environments**

AKA Env per (folder | branch | repo)

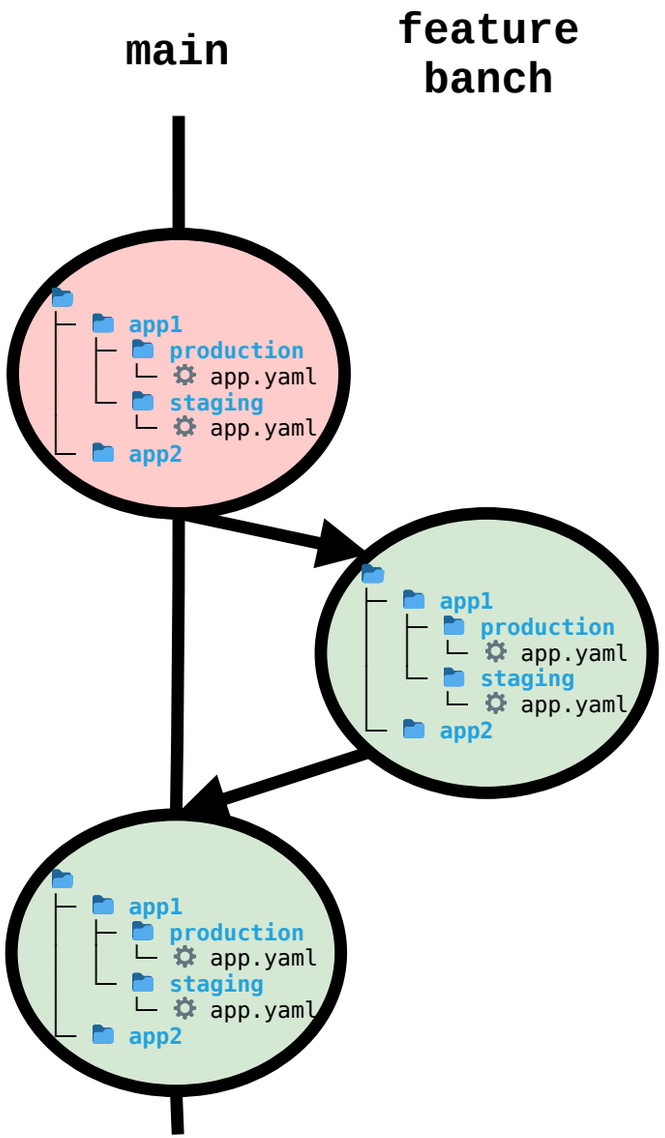
Branch vs folder per Environment

	Branch per env	Folder per env
envs	permanent branches	trunk-based folders
mapping example	`develop` → staging `master` → production	`staging` → Staging `production` → Production
promotion	merge	copy (+merge)

Branch per env



Folder per env



	Branch per env	Folder per env
pros	• Forces PRs • Feels natural for devs	• Avoids conflicts/drift • Copy vs cherry pick • Scales with envs
CM tool support (DRY) HELM K	?	app1 base shared.yaml overlays production specific.yaml staging specific.yaml
references	1	1, 2, 3, 4, …

Repo per environment

Why would you want to use one repo per env?

- Access to folders more difficult to constrain than repos
- Organizational constraints, e.g.
  - "devs are not allowed to acces prod"
  - security team needs to approve releases

➡️ Repos more complicated than folders. Use only when really necessary.

🔥 Preview environments

AKA (ephemeral | dynamic | pull request | test | temporary) environments

- An environment that is created with a pull request
- and deleted on merge/close

 `ApplicationSet`, using the `PullRequest` generator

 GitOpsSets ❓

Implementing promotion

Pull Requests

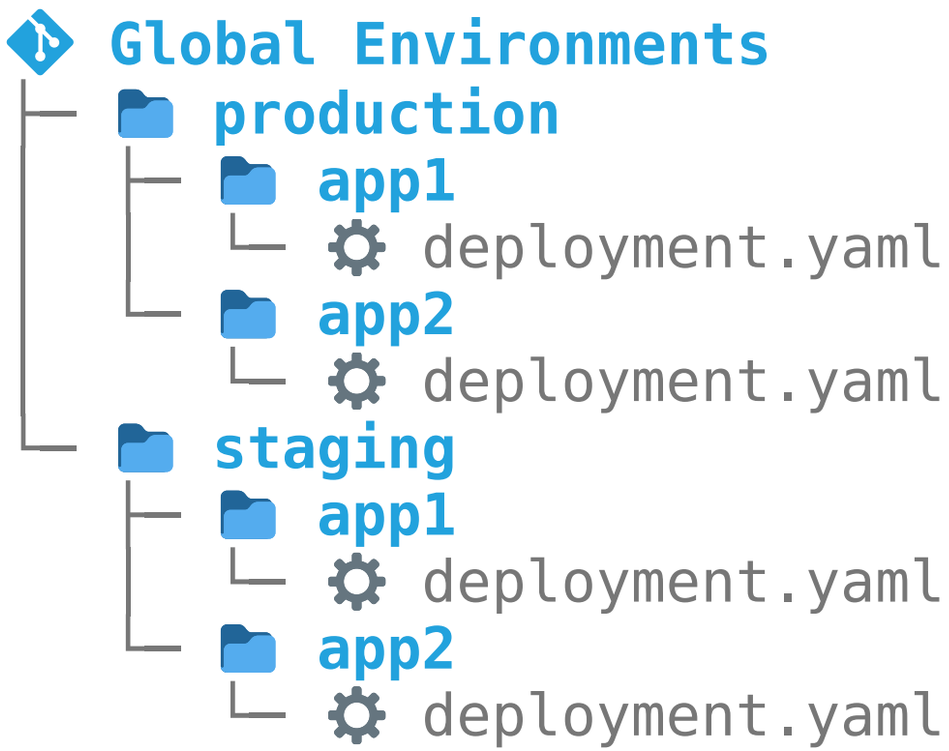
> GitOps - Operations by Pull Request
> 🌐 weave.works/blog/gitops-operations-by-pull-request

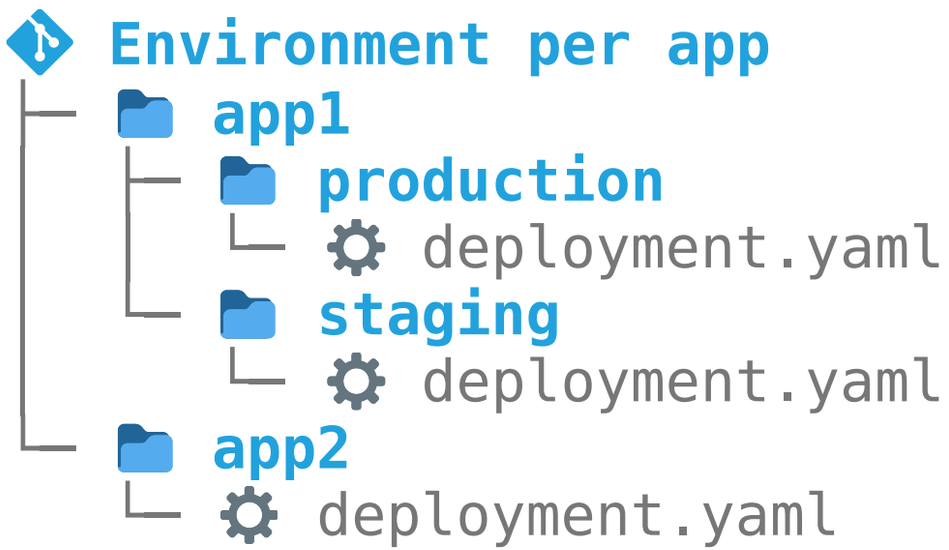
Configuration Management

AKA Templating, Patching, Overlaying, Rendering
➡️ Tools for separating config of envs, keeping them DRY

- Kustomize
  - plain `kustomize.yaml`
  - ≠ Flux CRD `Kustomization`
  - `kustomize build` / `kubectl kustomize` via CI server
- Helm
  - CRD (`Application`, `HelmRelease`)
  - *Umbrella Chart*
  - `helm template` via CI server

Global envs vs. env per app





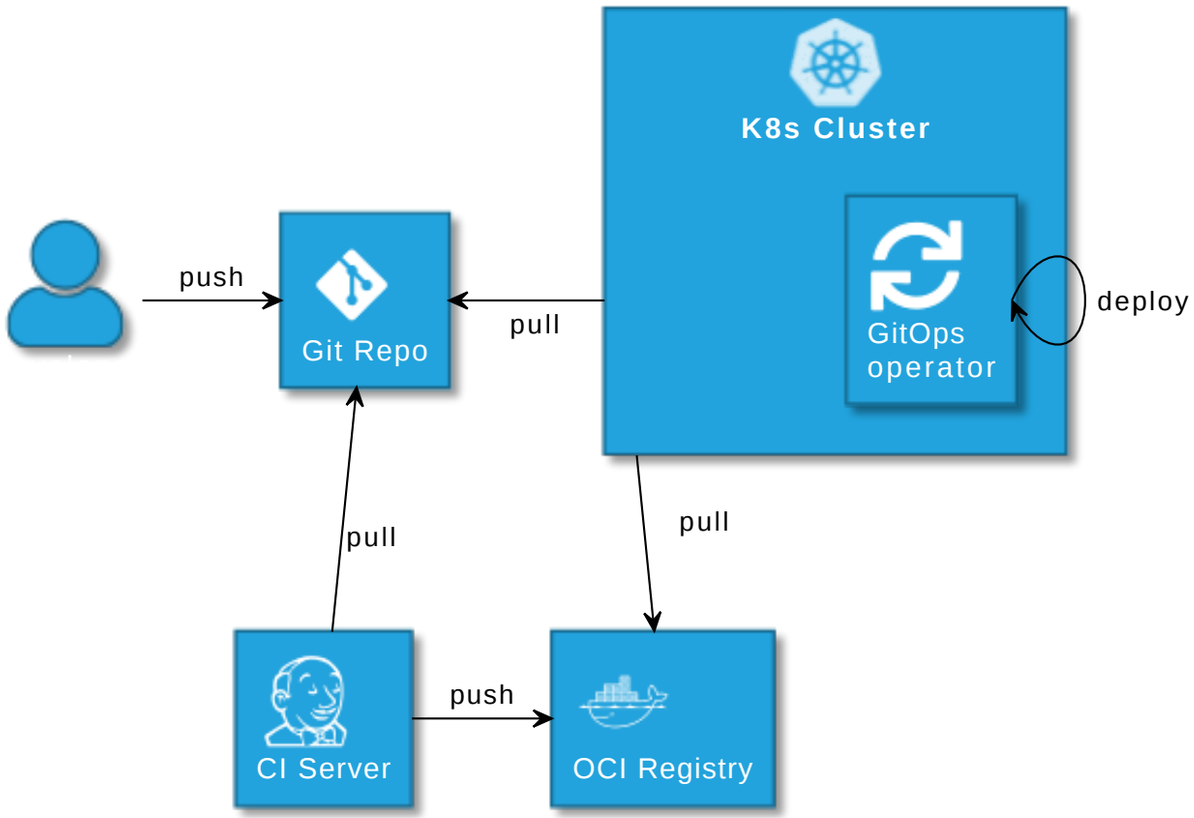
e.g. Preview Envs

Config update

Who updates image version in config repo, creates branch and PR?

- **Manual**: Human pushes branch and create PR 😈
- **Image Updater**: Operator pushes branch, create PR manually
- **CI Server**: Build job pushes branch, creates PR
- **Dependency Bot**: Bot pushes branch, creates PR

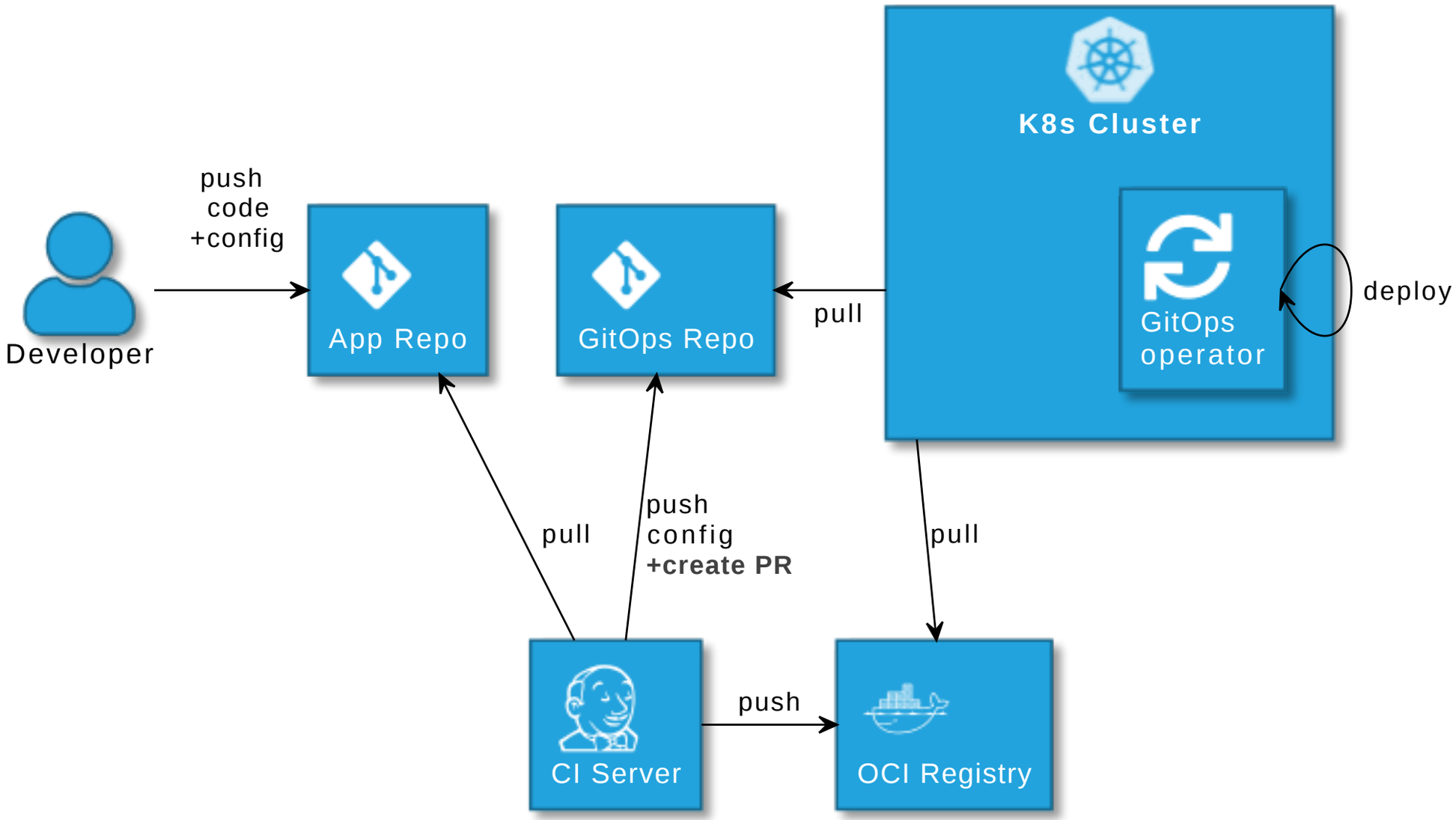
Image updater



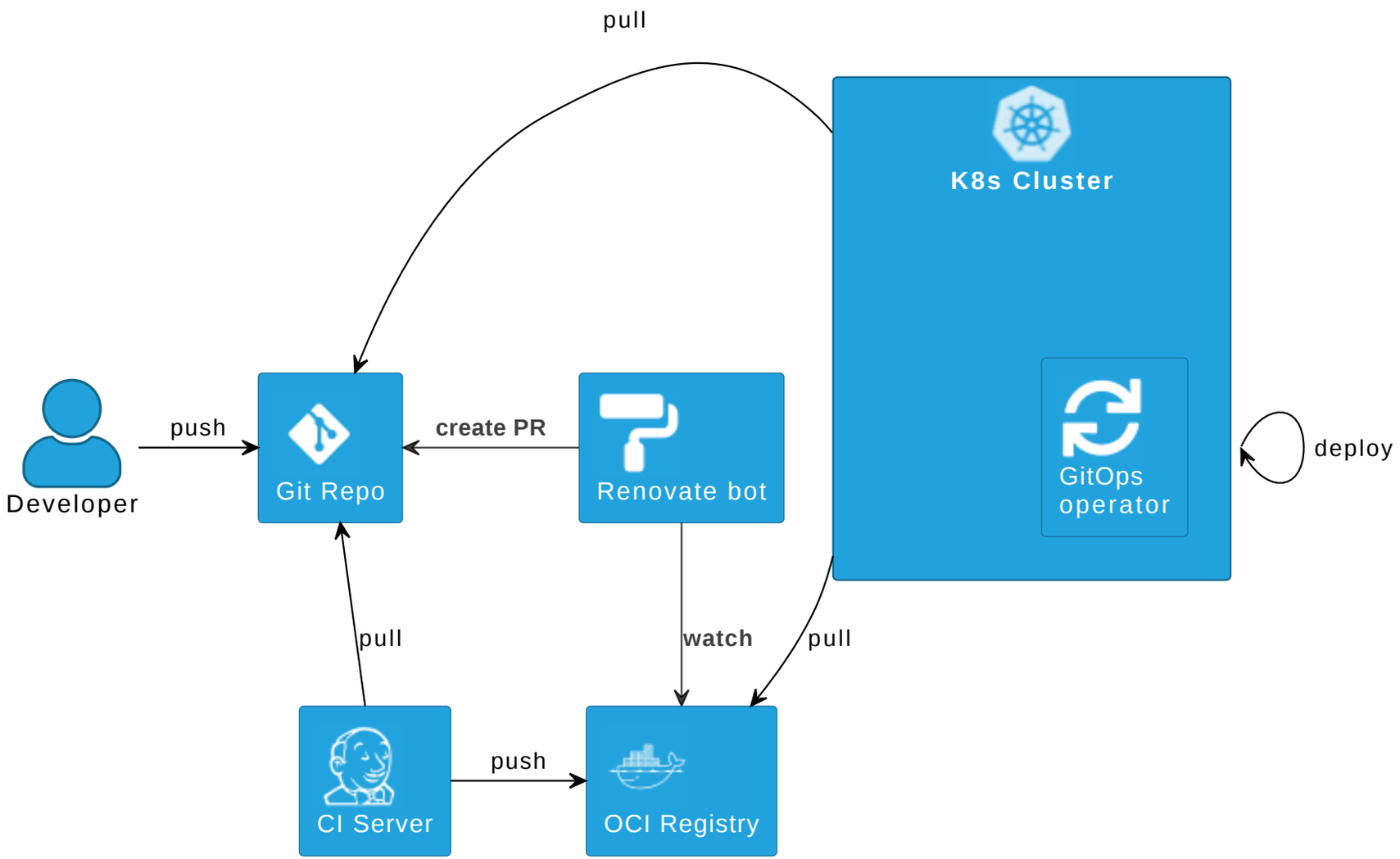
GitOps operator can update image version in Git

- github.com/argoproj-labs/argocd-image-updater
- fluxcd.io/docs/guides/image-update

Promotion via CI Server



Promotion via dependency bot



e.g. github.com/renovatebot/renovate

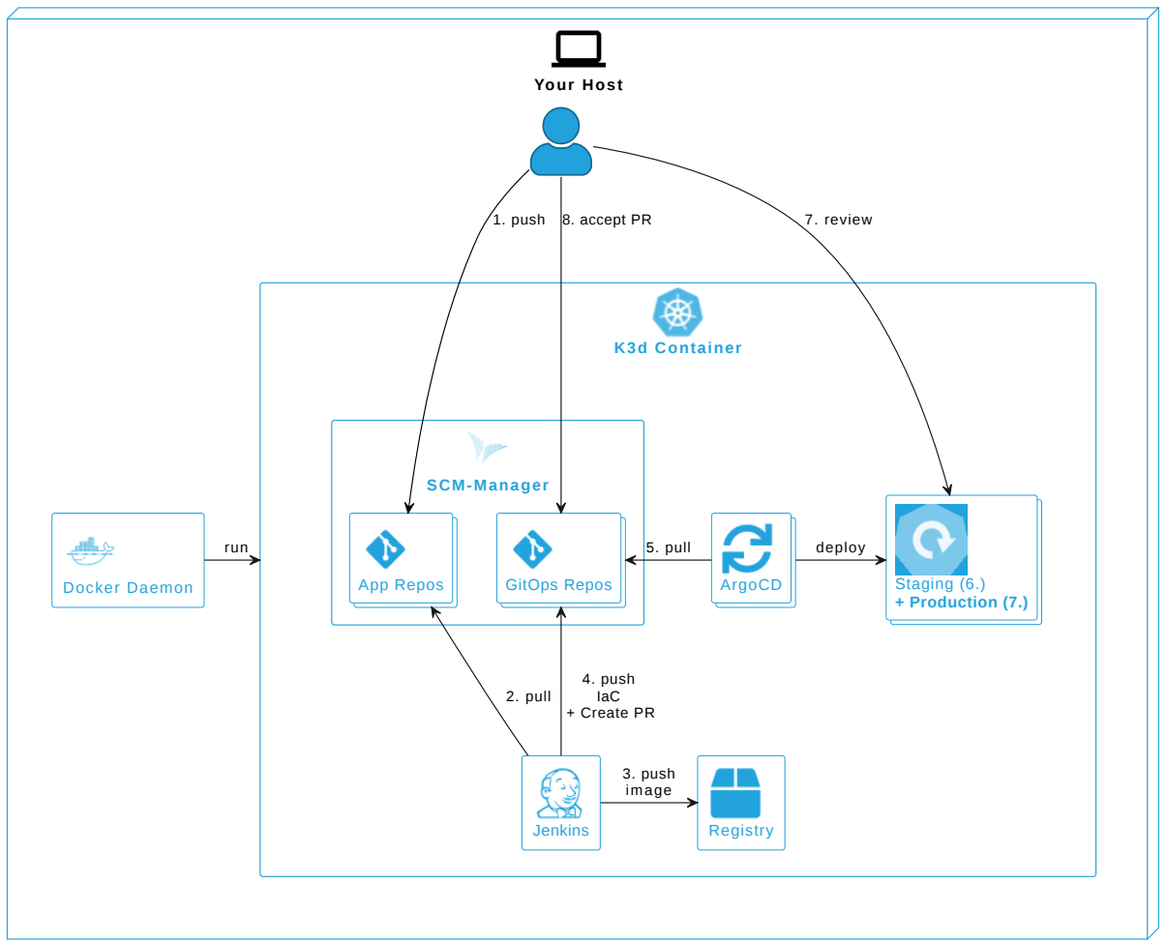
Wiring

Wiring up operator, repos, folders, envs, etc.

- Bootstrapping: `kubectl`, operator-specific CLI
- Linking/Grouping:
  - Operator-specific CRDs
    - `Kustomization`
    - `Application`
  - Nesting: *App of Apps*
    (same principle with `Kustomization`)
  - Templating: `ApplicationSets` - folders, lists, config files

GitOps process example + demo

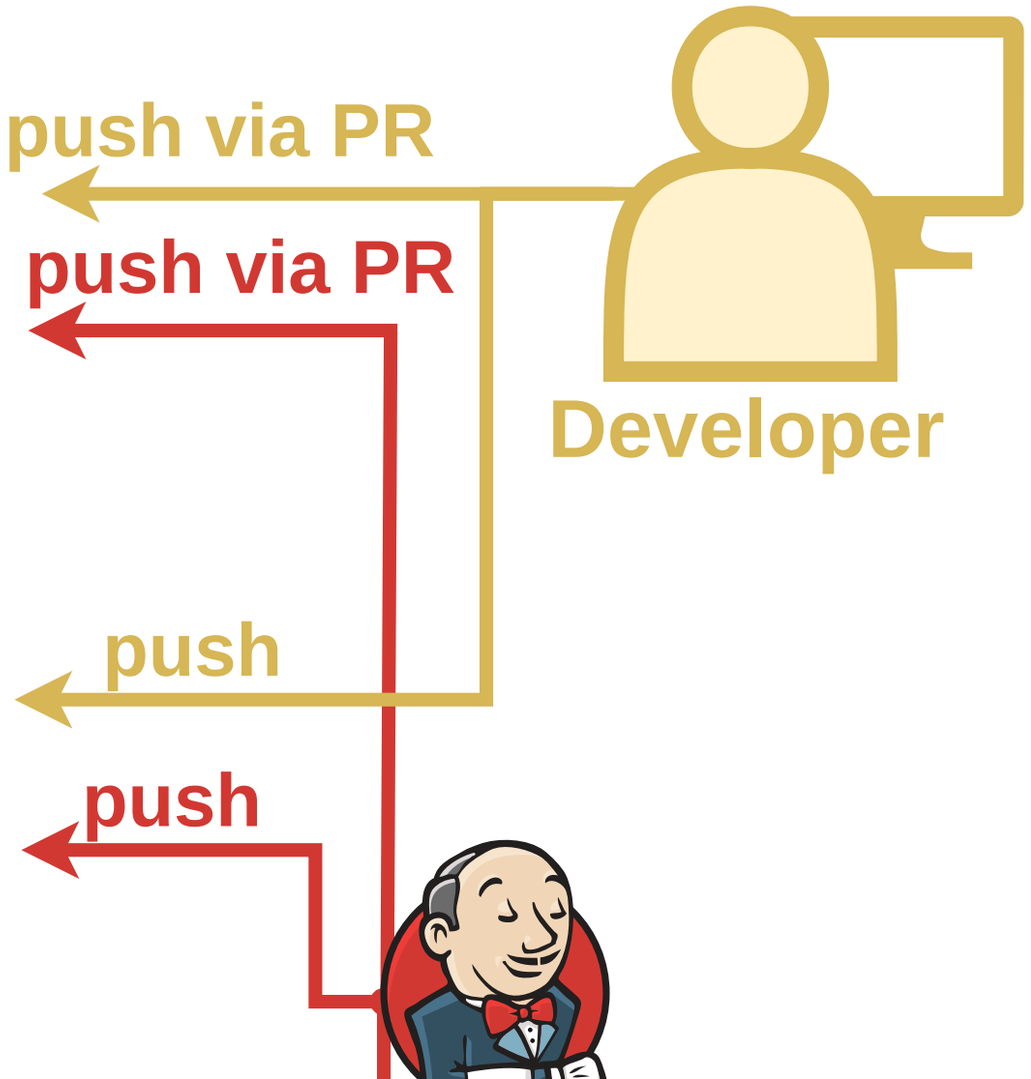
Demo



cloudogu/gitops-playground

Recap: Repos + Promotion

```
team-gitops-repo
    production
        3rd-party-app
        custom-app

    staging
        3rd-party-app
        custom-app
```



pull

CI server

app-repo

k8s

production

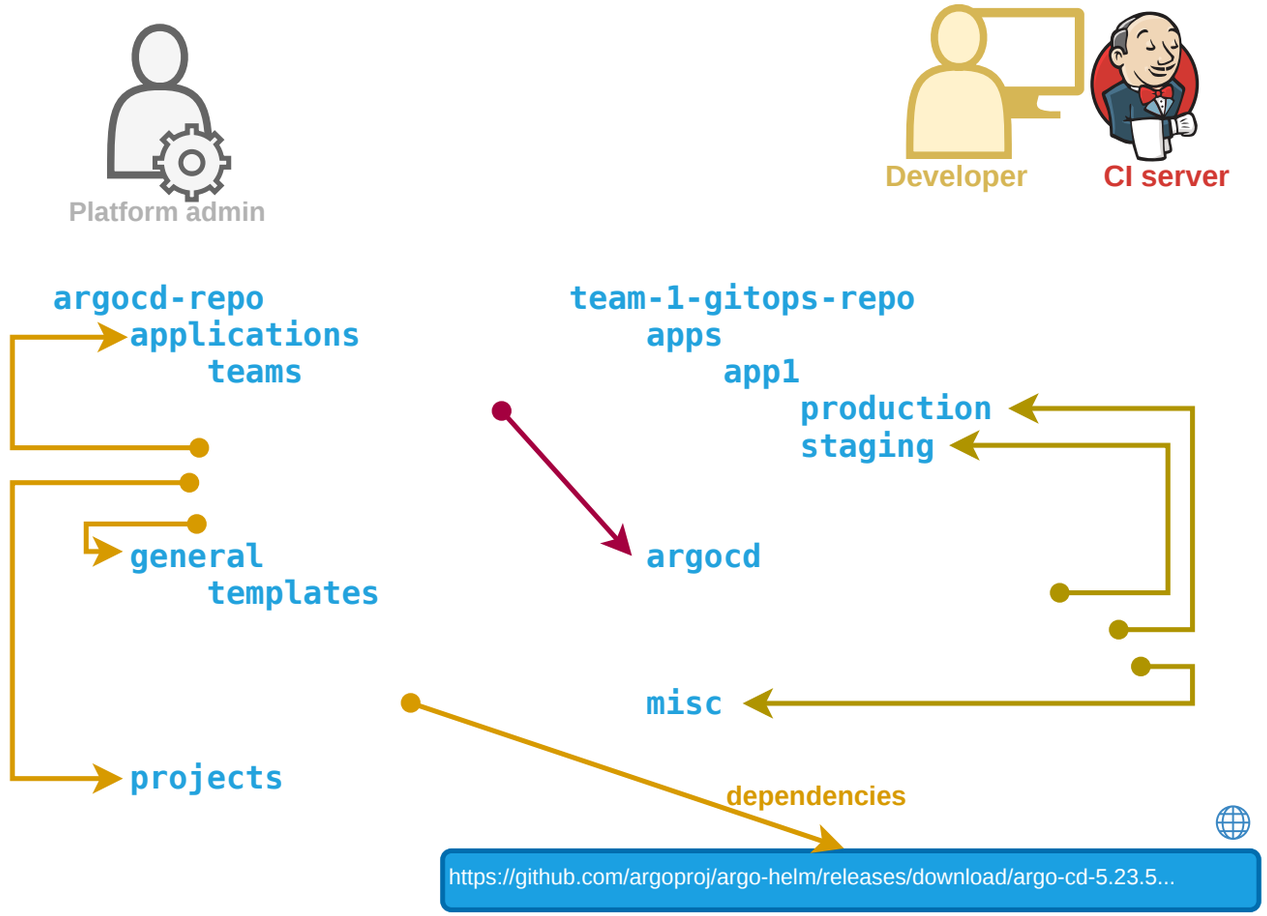
staging

src

More examples

Example 2: GitOps playground

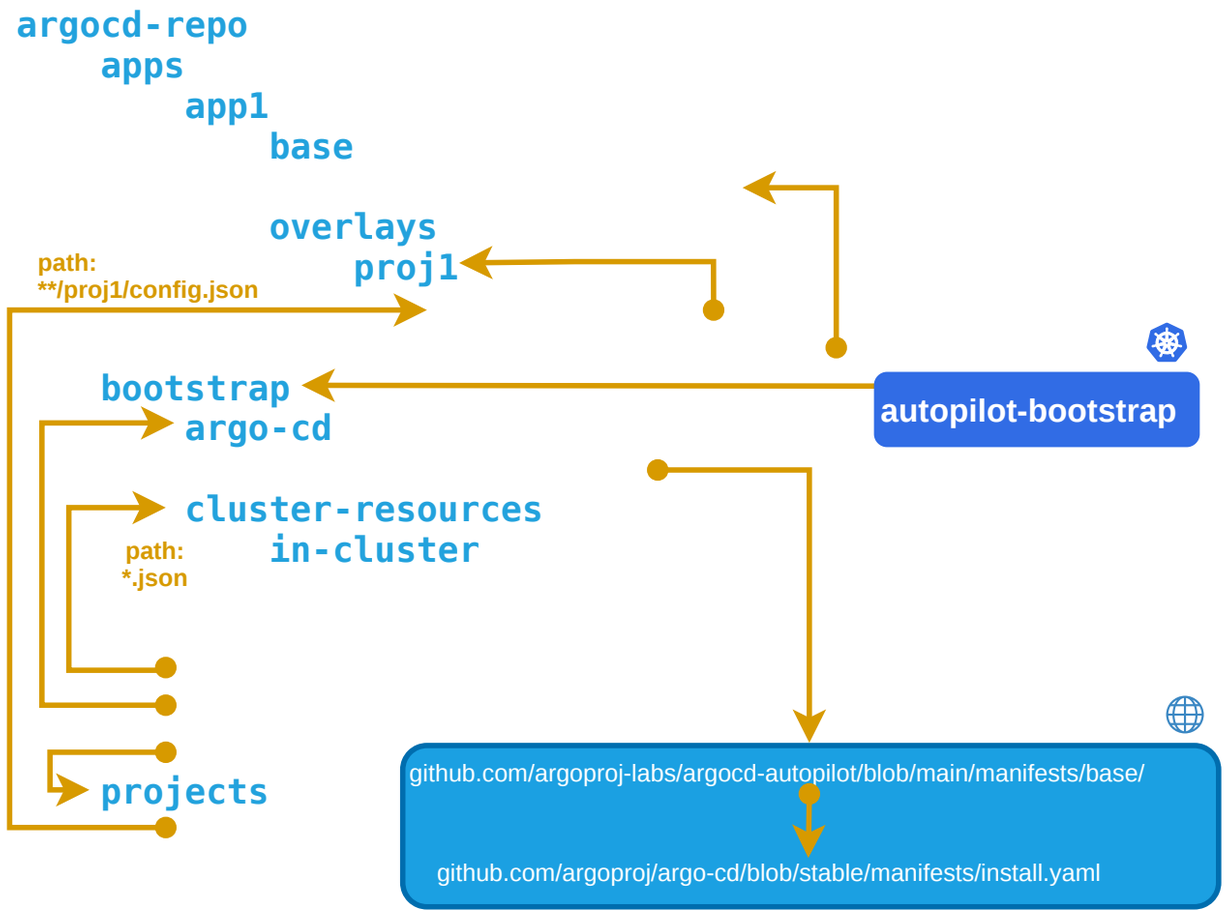
TODO re-number examples, update SVGs for bright mode



- **Repo pattern:** Per team 🏗 per app
- **Operator pattern:** Standalone (Hub and Spoke)
- **Operator:** 🐙 (🚀)
- **Boostrapping:** `Helm`, `kubectl`
- **Linking:** 🐙 `Application`
- **Features:**
  - Env per app
  - Operate ArgoCD with GitOps
  - Automation via CI server
  - Mixed repo patterns
  - Solution for cluster resources
  - ArgoCD **and** Flux examples
- **Source:** cloudogu/gitops-playground

Example 3: ArgoCD autopilot

- **Repo pattern:** Monorepo
- **Operator pattern:**
  Standalone / Hub and Spoke
- **Operator:**
- **Boostrapping:** `argocd-autopilot`
- **Linking:** `Application`, `ApplicationSet`,
- **Features:**
  - Operate ArgoCD with GitOps
  - Solution for cluster resources
  - Opinionated structure and YAML creation via CLI
- **Source:** argoproj-labs/argocd-autopilot

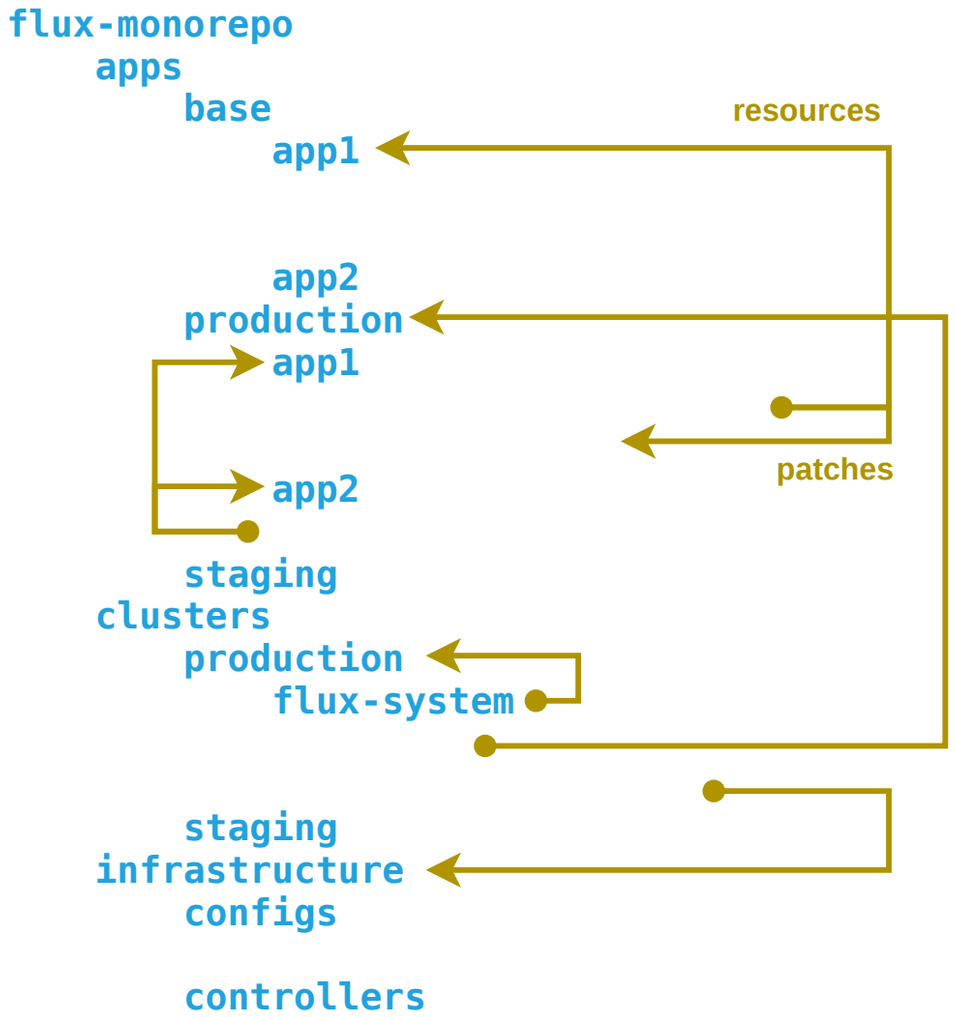


Example 4: Flux Monorepo

- **Repo pattern:** Monorepo
- **Operator pattern:** Standalone
- **Operator:** (?)
- **Boostrapping:** `flux`
- **Linking:** `Kustomization`,
- **Features:**
  - Solution for cluster resources
  - Operate Flux with GitOps
- **Source:**

fluxcd/flux2-kustomize-helm-example#16

fluxcd.io/flux/guides/repository-structure

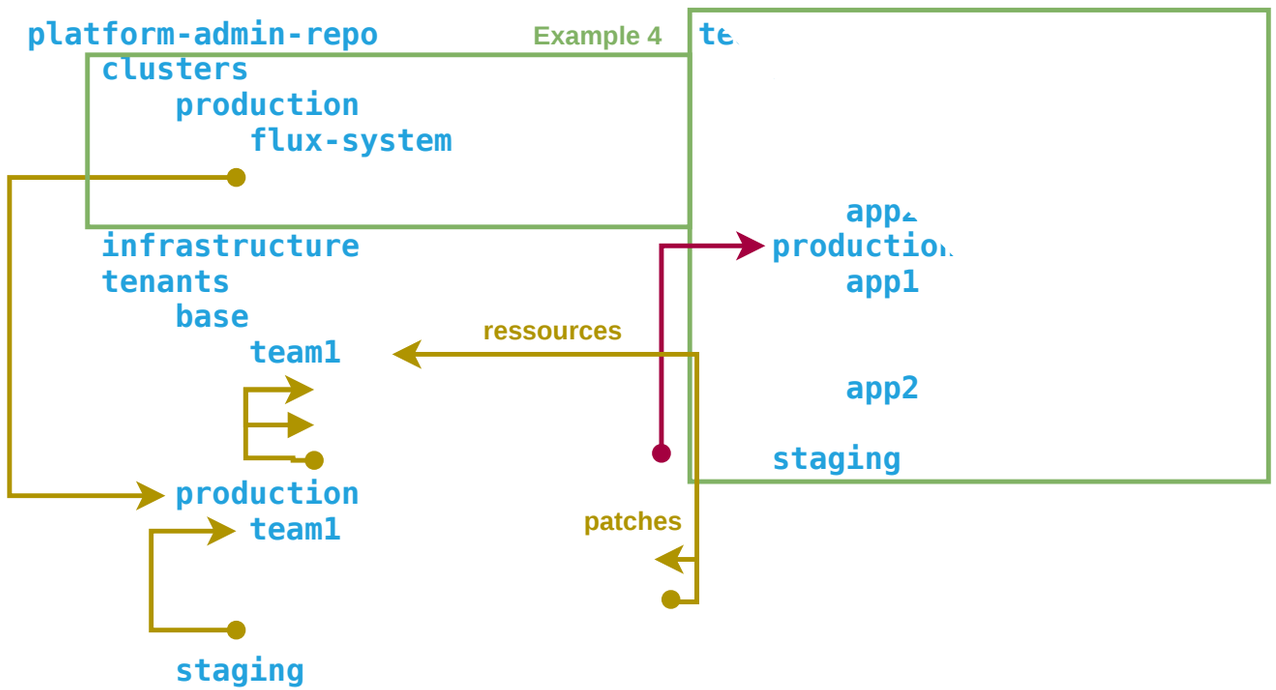


Example 5: Flux repo per team

- **Repo pattern:** Repo per team
- **Operator pattern:** Standalone
- **Operator:** (?)
- **Boostrapping:** `flux`
- **Linking:** `Kustomization`, K
- **Features:** Ex 5 with repo for team
- **Source:**

  fluxcd/flux2-multi-tenancy

  fluxcd.io/flux/guides/repository-structure



Example 6: ArgoCD and Flux alternative

- **Repo pattern:** Monorepo
- **Operator pattern:** Standalone
- **Operator:**
- **Boostrapping:** `kubectl`
- **Linking:** `Application`, `ApplicationSet` / `Kustomization`,
- **Features:**
  - Cross-cutting resources
  - ArgoCD **and** Flux examples
- **Source:**

christianh814/example-kubernetes-go-repo

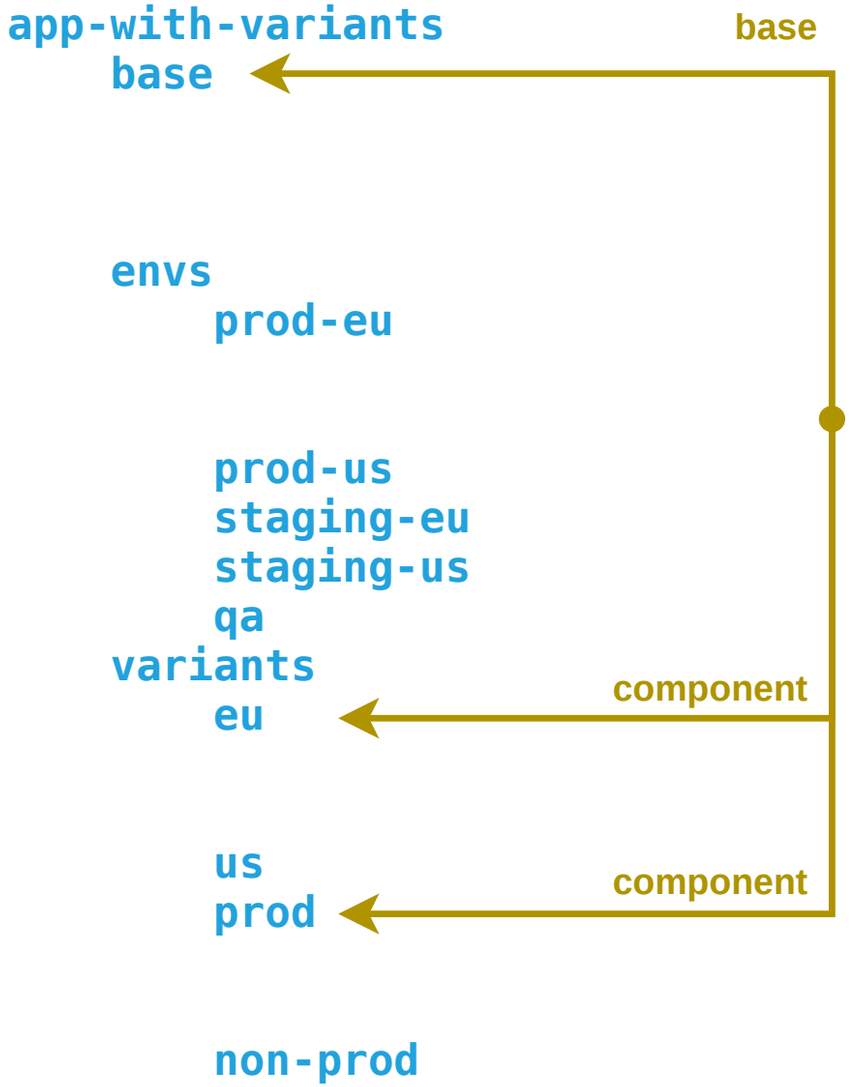
C. Hernandez - The Path to GitOps

```
monorepo
    cluster-XXXX
        apps
            myapp

        bootstrap
            base

            overlays
                default

        cluster-config
            gitops-controller

            sample-admin-workload

        components
            applicationsets

            argocdproj
```

Example 7: Environment variations

- **Operator:** () ()
- **Features:**
  - Env variants for a single app
  - Promotion “via `cp`”
- **Source:**

  kostis-codefresh/gitops-environment-promotion



The perfect GitOps process?

No such thing as the perfect GitOps process

- Patterns exist - for different aspects, inconsistent naming
- Examples exist - different operators + scopes (bootstrapping vs. apps only)

➡️ Use as inspiration

Johannes Schnatterer, Cloudogu GmbH

☁️ cloudogu.com/gitops

- GitOps Resources
- Community
- Trainings
- Consulting

Slides

💪 Join my team: cloudogu.com/join/cloud-engineer

@schnatterer@floss.social in/jschnatterer @jschnatterer

Image sources

- coloured-parchment-paper background by brgfx on Freepik
  https://www.freepik.com/free-vector/coloured-parchment-paper-designs_1078492.htm
- Example:
  https://unsplash.com/photos/X2PWhiKDQww
- More examples
  https://unsplash.com/photos/XZc4f2XZc84
- Perfect?
  https://pixabay.com/illustrations/question-mark-question-response-1020165/