cloudogu

// THE PERFECT GITOPS PROCESS: REPOS, FOLDERS, STAGES, PATTERNS

Johannes Schnatterer, Cloudogu GmbH

@schnatterer@floss.social

@jschnatterer

Version: 202302281634-ca16fe3

Agenda

1. Design basics
2. Example + demo
3. More examples

GitOps process design basics

Preamble

- **Chronology**:
  - Step 1: Chose an operator
  - Step 2: **Design process/repos** ⬅️ focus of this talk
- **Use case**:
  - Deploying infra
  - **Deploying apps** ⬅️ focus of this talk
- **Responsibility**: platform/infra teams, cluster admins ↔️ app teams
- **Conway's law**: No standard for structures (intentionally)

🤯 GitOps Chasm

Infra

- repos
- folders
- ~~branches~~
- clusters
- namespaces
- operator instances
- operator-specific config

↔️

Mapping?

🤔

🌍 Real-world

- company/departments
- teams
- projects
- applications
- microservices
- customers
- tenants
- **stages/environments**
- etc.

No standard but emerging patterns

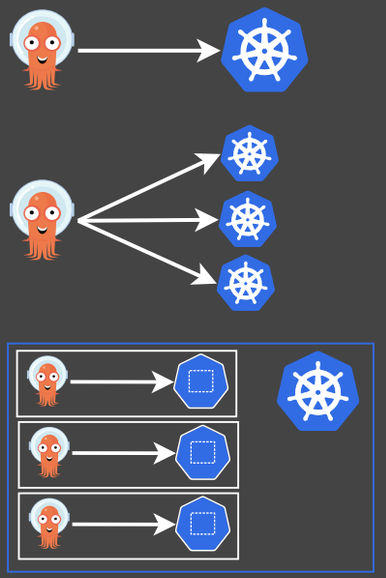
AKA strategies, models, approaches, best practices

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

GitOps Operator deployment patterns

How many GitOps operators to deploy, relating to Kubernetes clusters?

- *Standalone*: 1 Controller : 1 Cluster
- *Hub and Spoke*: 1 Controller : n Clusters
- Namespaced: n Controllers : 1 Cluster



Repository patterns

How many GitOps repos?

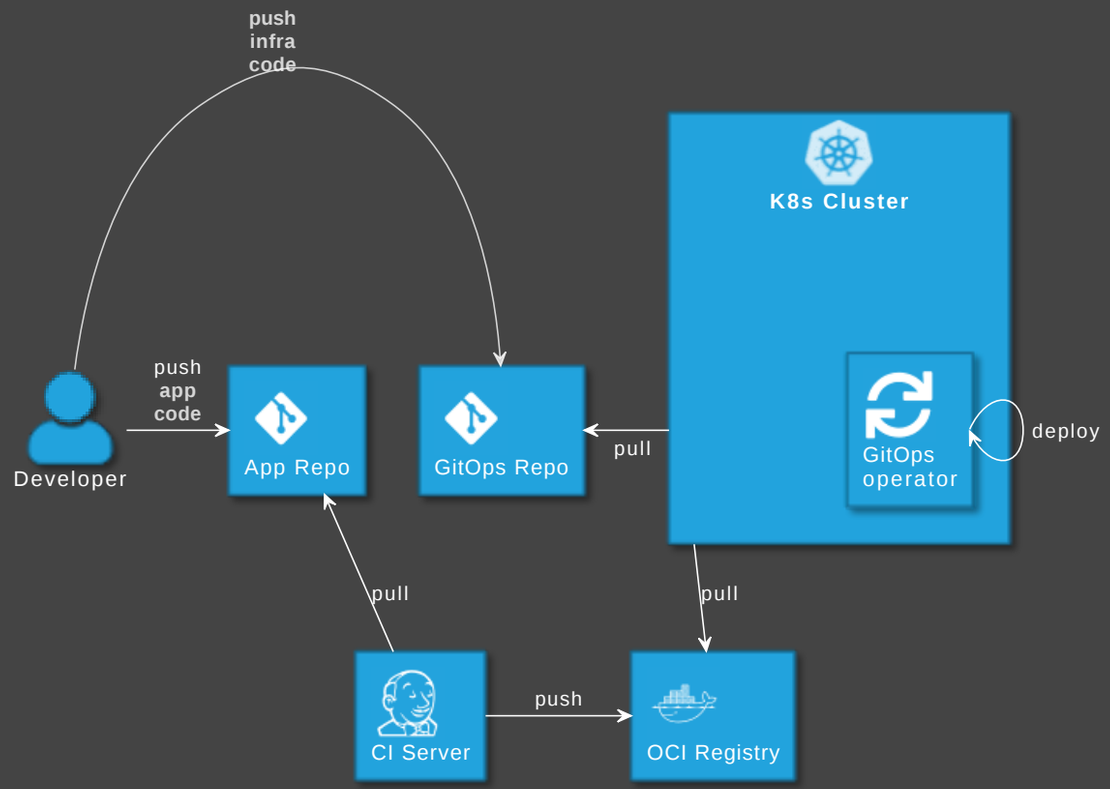
- *Monorepo* (opposite polyrepo)
- *Repo per Team* (Tenant)
- *Repo per App*
  - Config replication
  - Repo pointer
- *Repo per stage/environment* 🕒

Can be mixed 🏗️

Repository types

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

Separating GitOps repo from app repo



GitOps tools: Put infra in separate repo! See

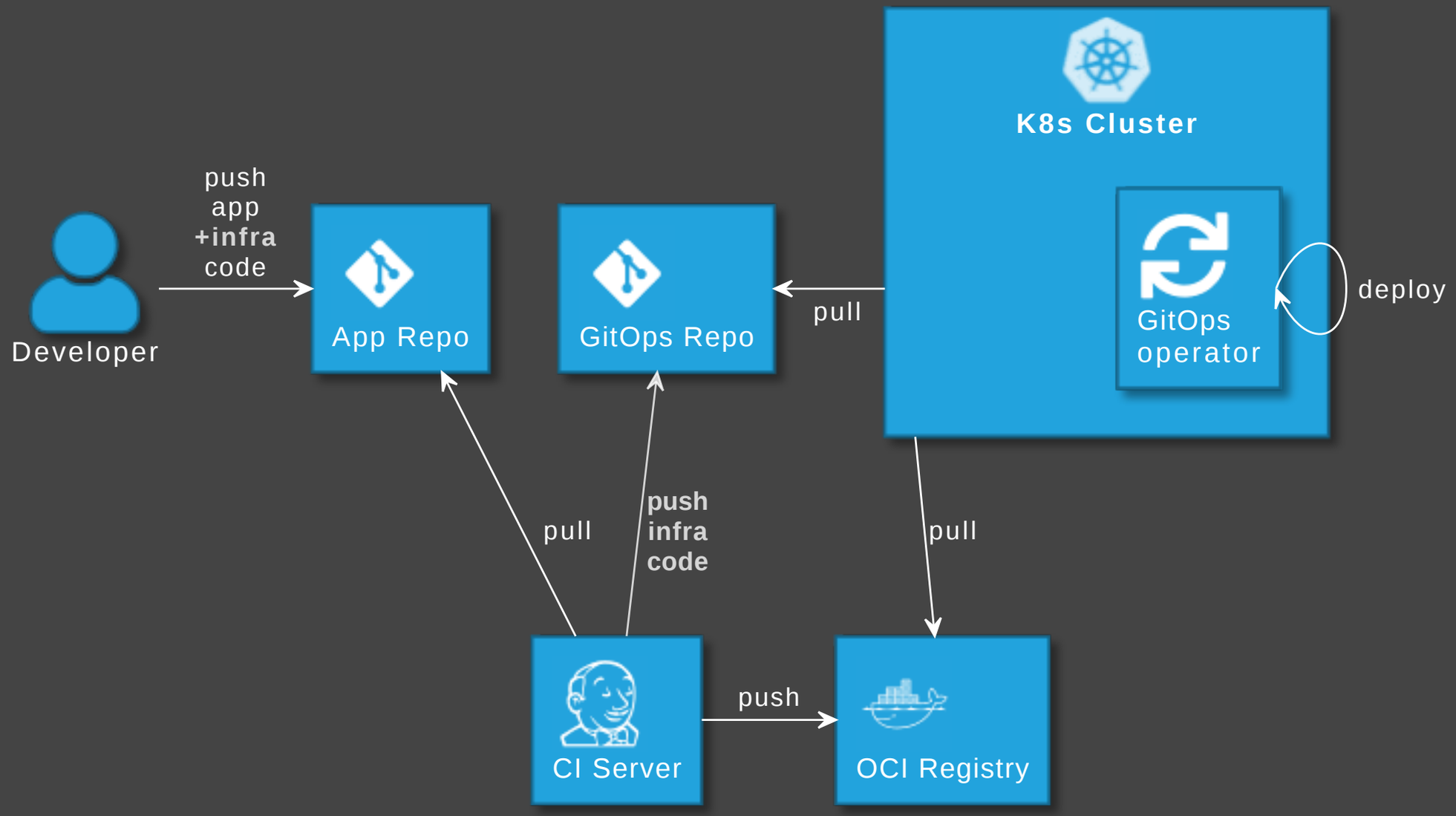
argo-cd.readthedocs.io/en/release-2.6/user-guide/best_practices

Disadvantages

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

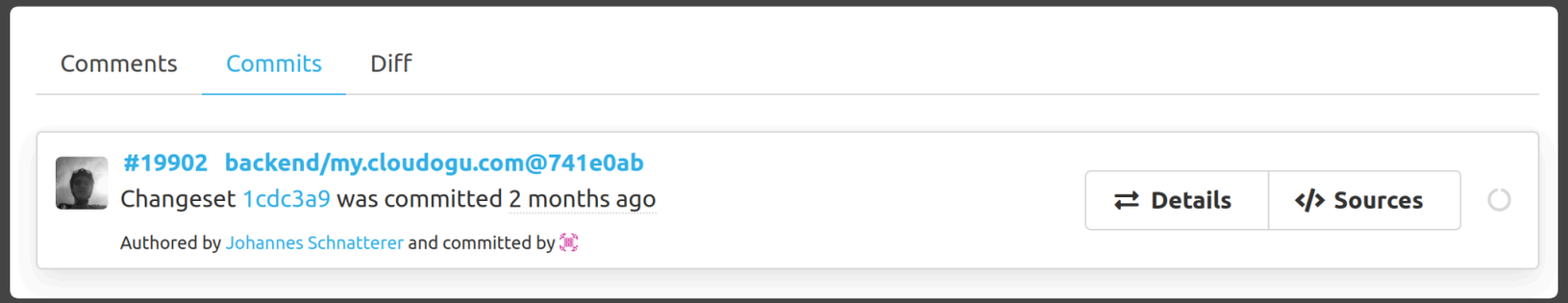
How to avoid those?

Config replication



Advantages

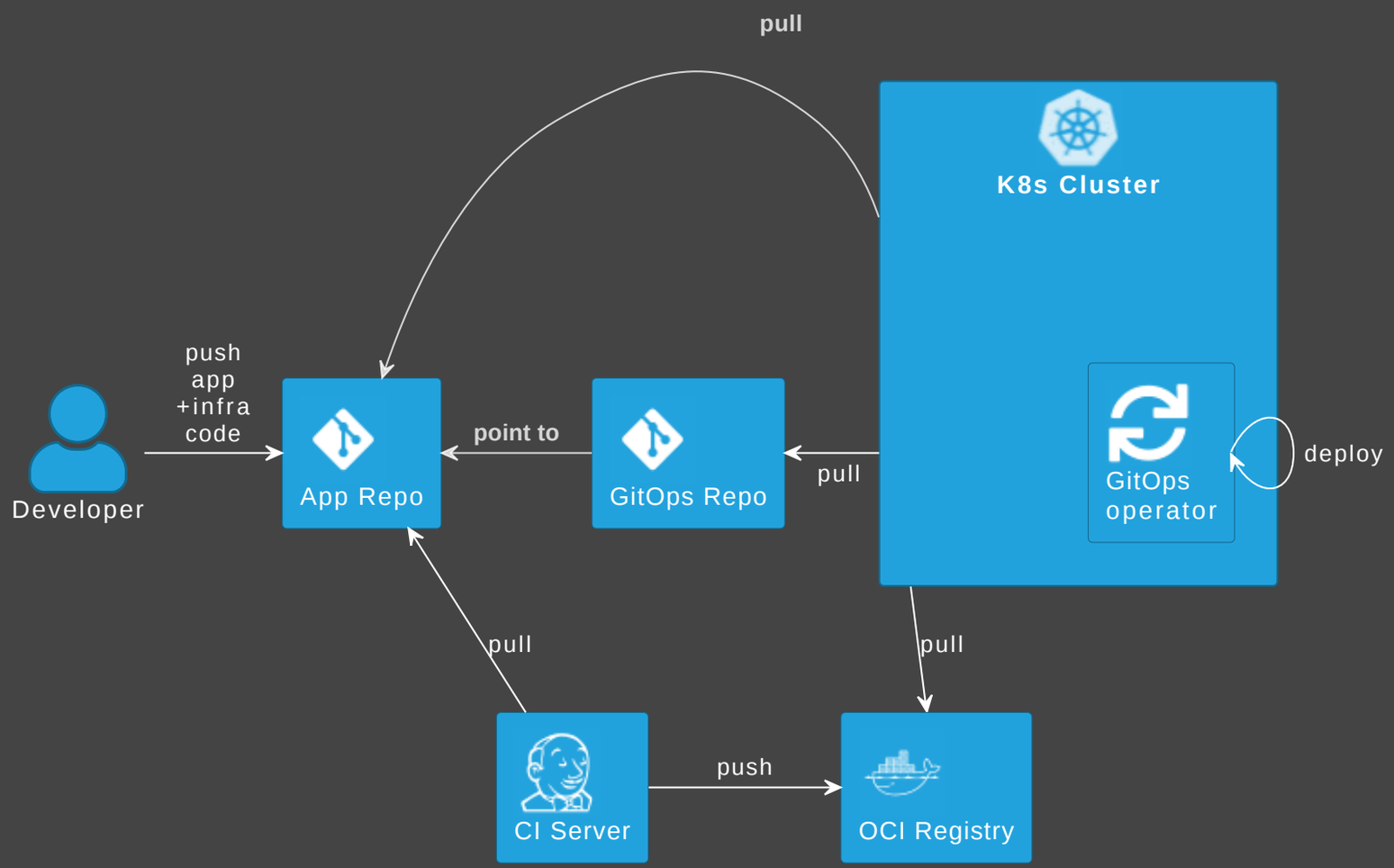
- Single repo for development: higher efficiency
- Shift left: static code analysis + policy check on CI server, e.g. yamlint, kubeval, helm lint, conftest
- Automated staging (e.g. PR creation) 🕒
- Simplify review by adding info to PRs



Disadvantages

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

Alternative: Repo pointer



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

Release promotion patterns

How to model environments AKA stages?

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

AKA Env per (folder | branch | repo)

Why not use branches for environments?

Idea:

- Develop ➡️ Staging
- Main ➡️ Production

- Drifts/conflicts because of merge direction develop ➡️ main (unidirectional)
- Promoting specific changes only: Copy vs cherry pick
- DRY - resources shared by multiple environments, e.g. HELM K
- Scalability: More envs, more chaos

➡️ Branches more complicated than folders. Don't.

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.

Folder per environment

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

- Create *short-lived* branches and PRs
- 💡 Use folders to design envs (instead of *long-lived* branches per env)
- Merge promotes release, triggers deployment

Implementing release promotion

Tools for separating config

- Kustomize
  - plain `kustomize.yaml`
  - ≠ Flux CRD `Kustomization`
- Helm
  - CRD (`Application`, `HelmRelease`)
  - *Umbrella Chart*

Global envs vs. env per app

```
Global Environments
├── production
│   ├── app1
│   │   └── deployment.yaml
│   └── app2
│       └── deployment.yaml
└── staging
    ├── app1
    │   └── deployment.yaml
    └── app2
        └── deployment.yaml
```

```
Environment per app
├── app1
│   ├── production
│   │   └── deployment.yaml
│   └── staging
│       └── deployment.yaml
└── app2
    └── deployment.yaml
```

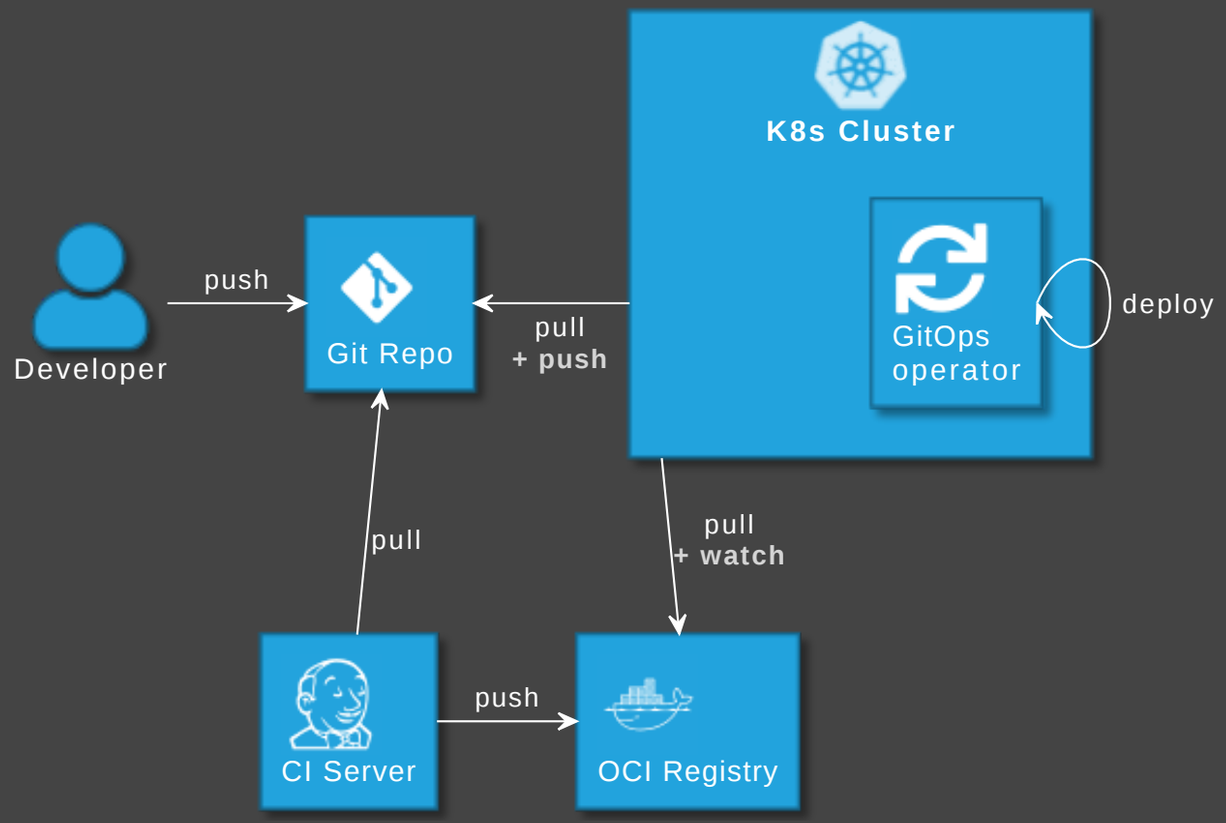
e.g. Preview Envs

Branch and PR creation

Who bumps versions in GitOps 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

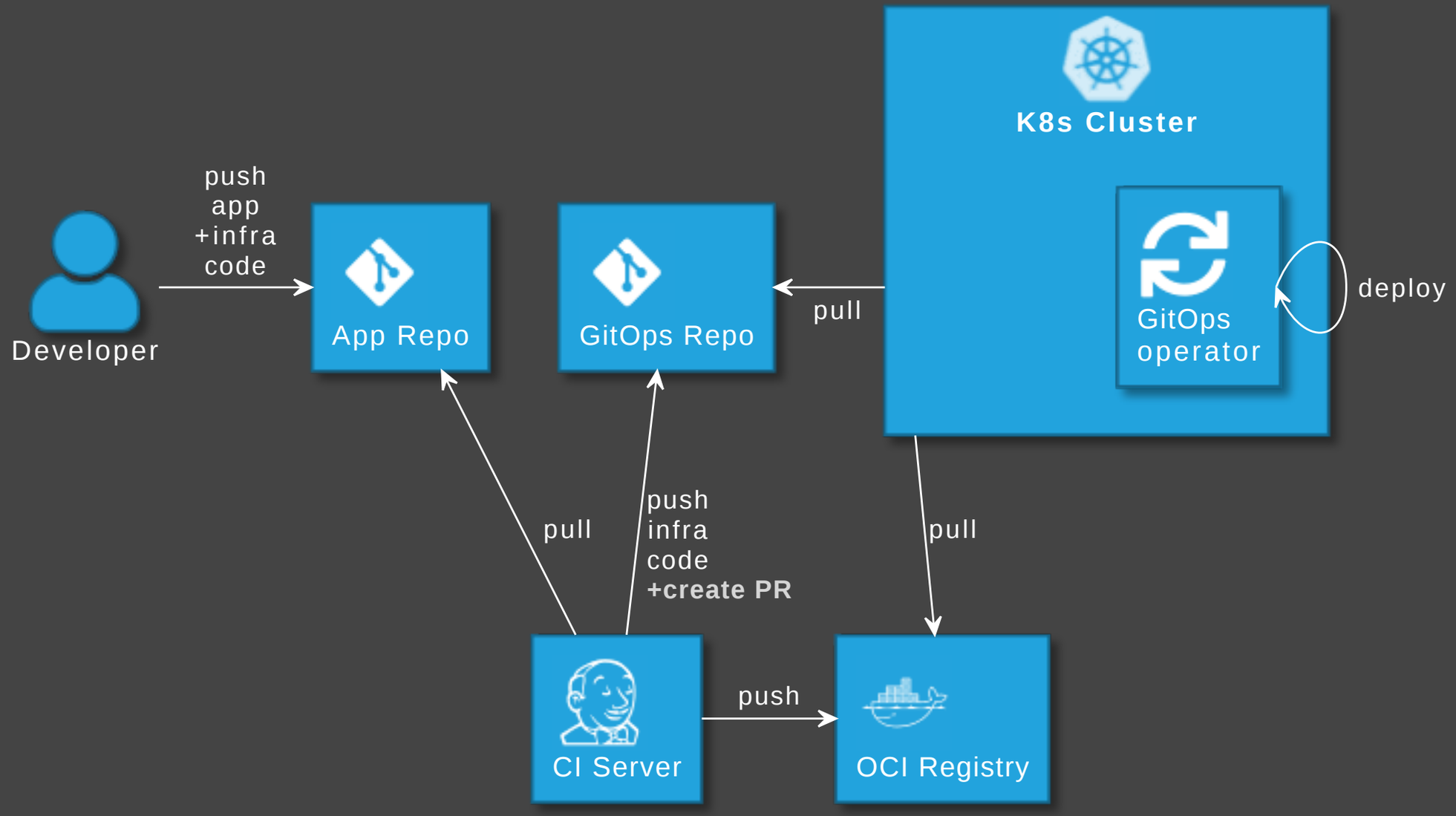
Image updater



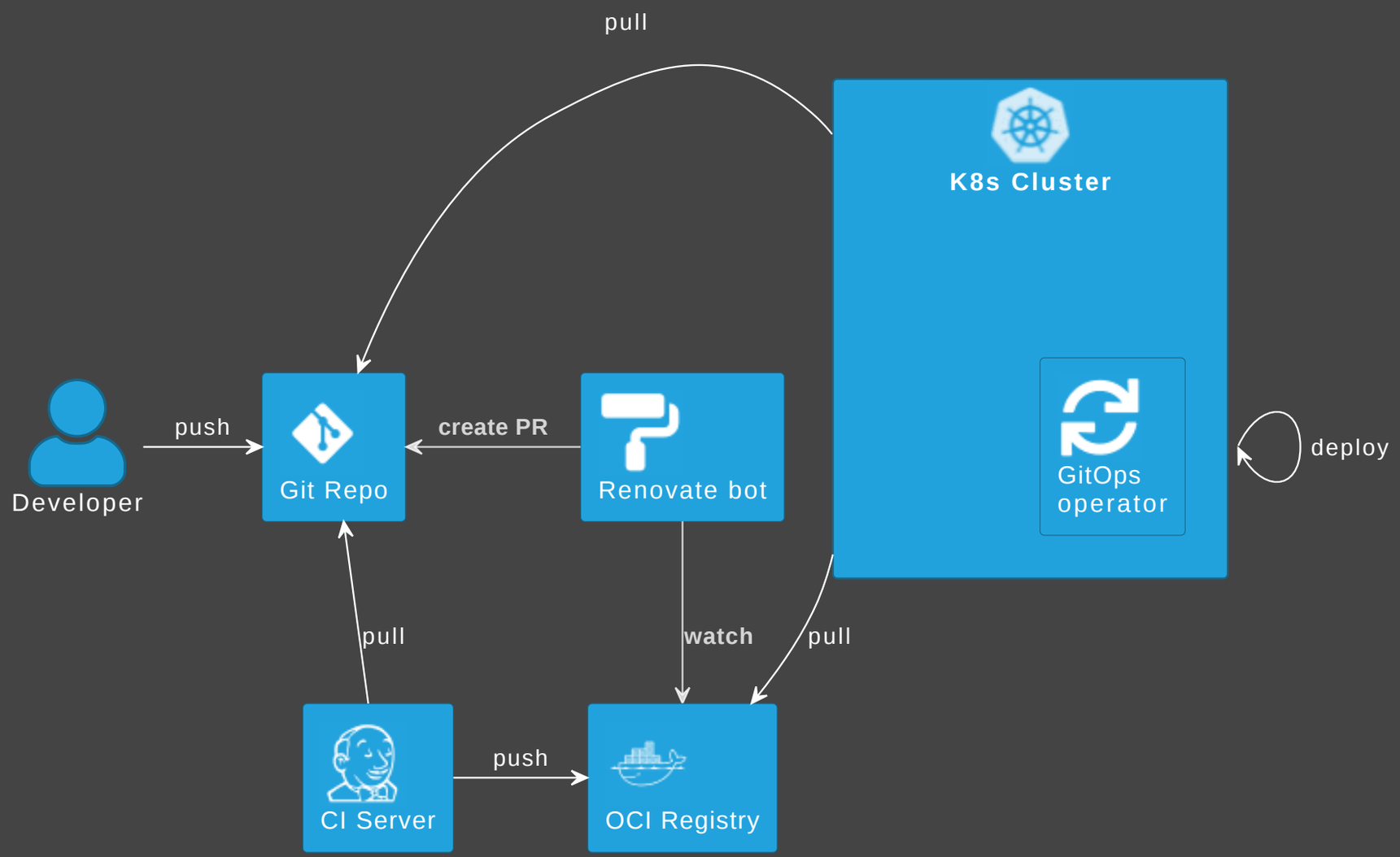
GitOps operator can update image version in Git

- 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

🔥 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

 ❓

Wiring

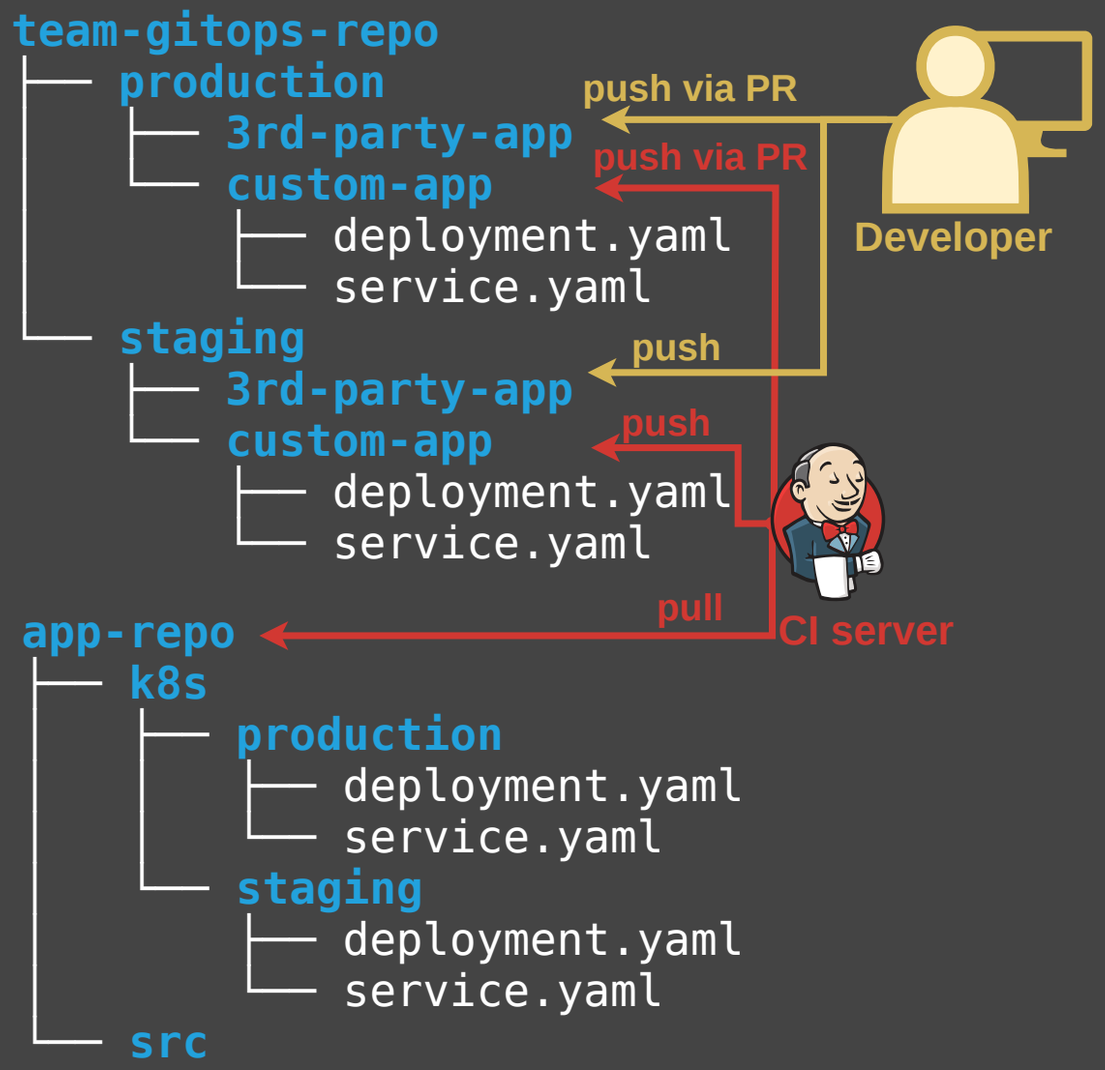
Wiring up operator, repos, folders, envs, etc.

- 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

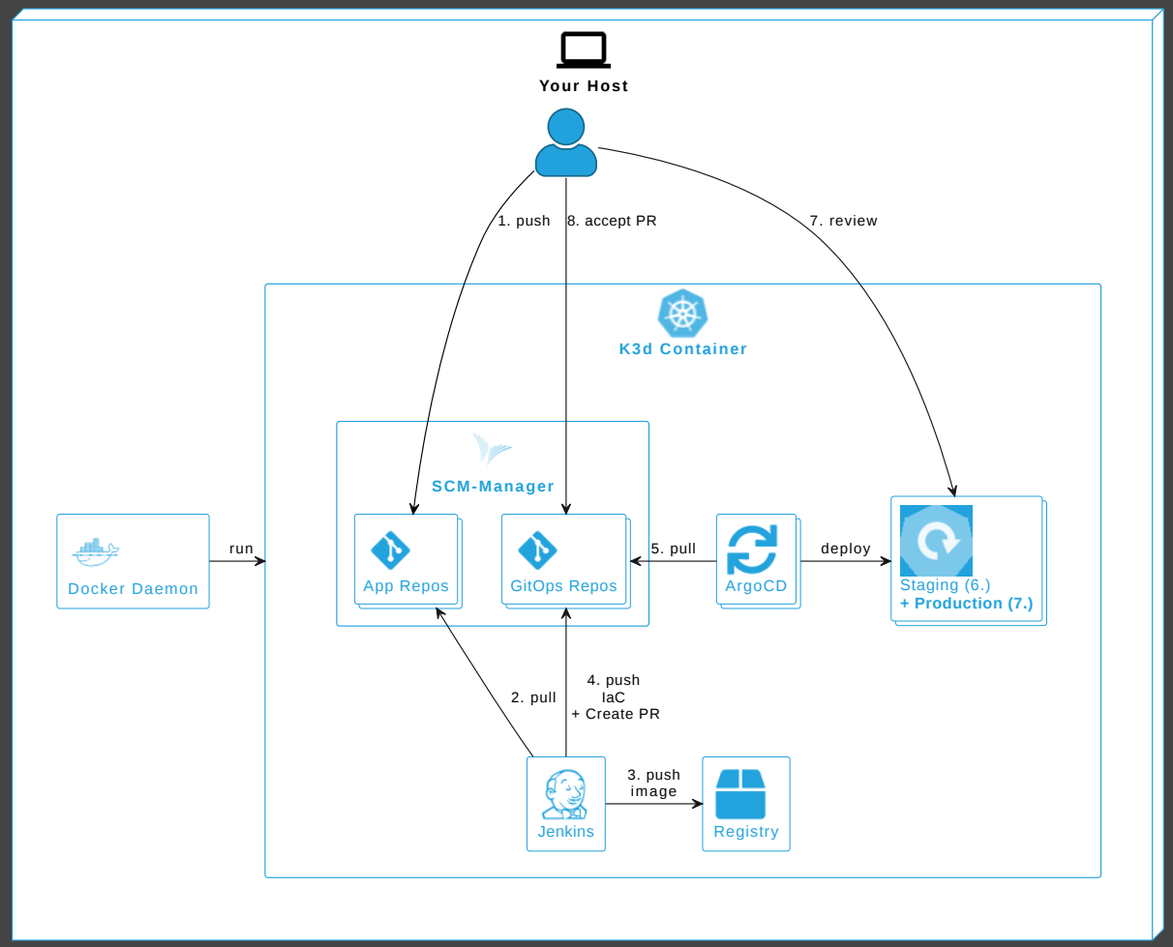
GitOps process example + demo

Example 1: Repo per team and app + CI

- **Repo pattern:**
  Per team/monorepo per app
- **Operator:**
- **Features:**
  - Automation via CI server
  - Mixed repo patterns
  - ArgoCD **and** Flux examples
- **Source:**
  cloudogu/gitops-playground

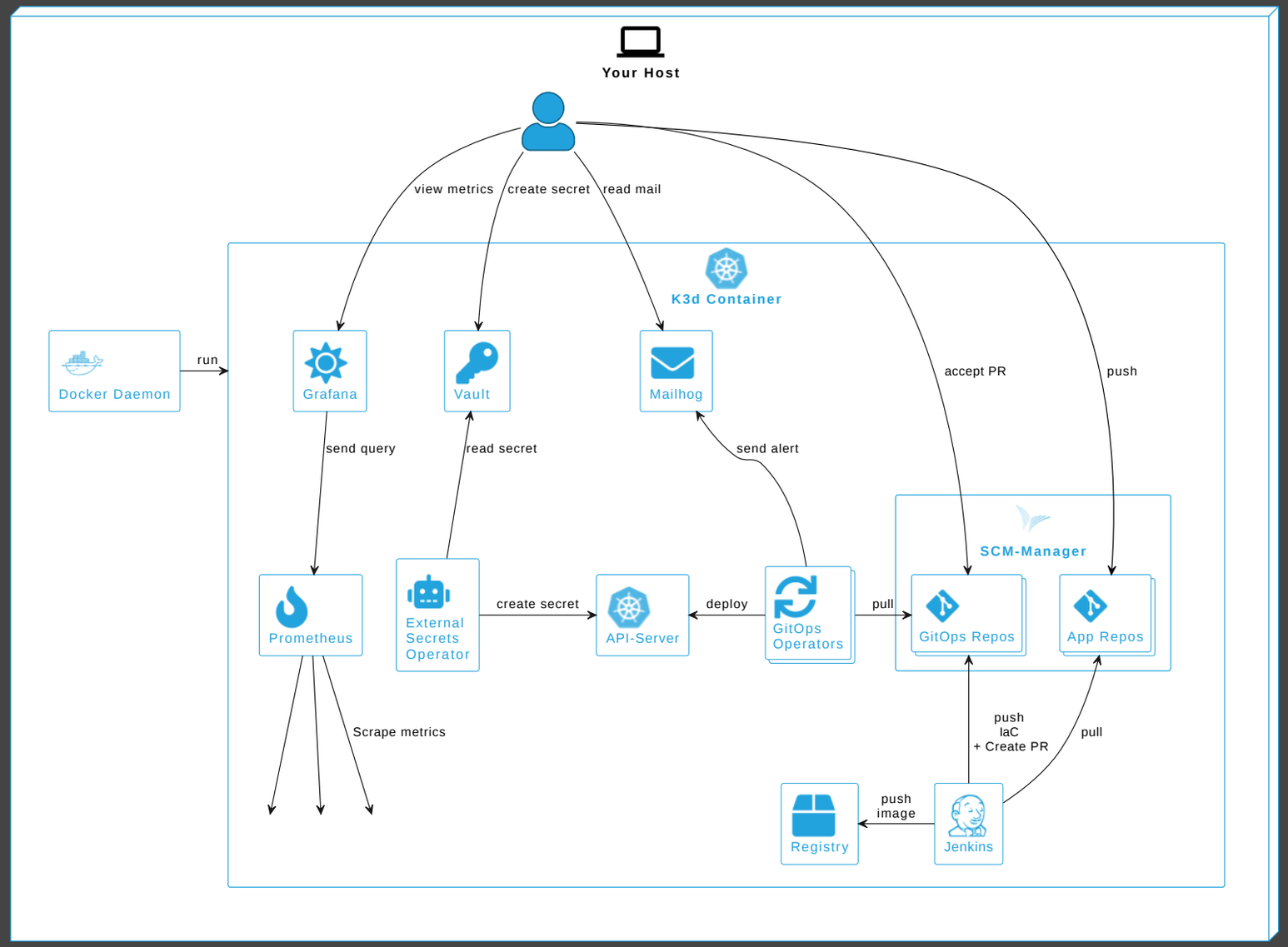


Demo



cloudogu/gitops-playground

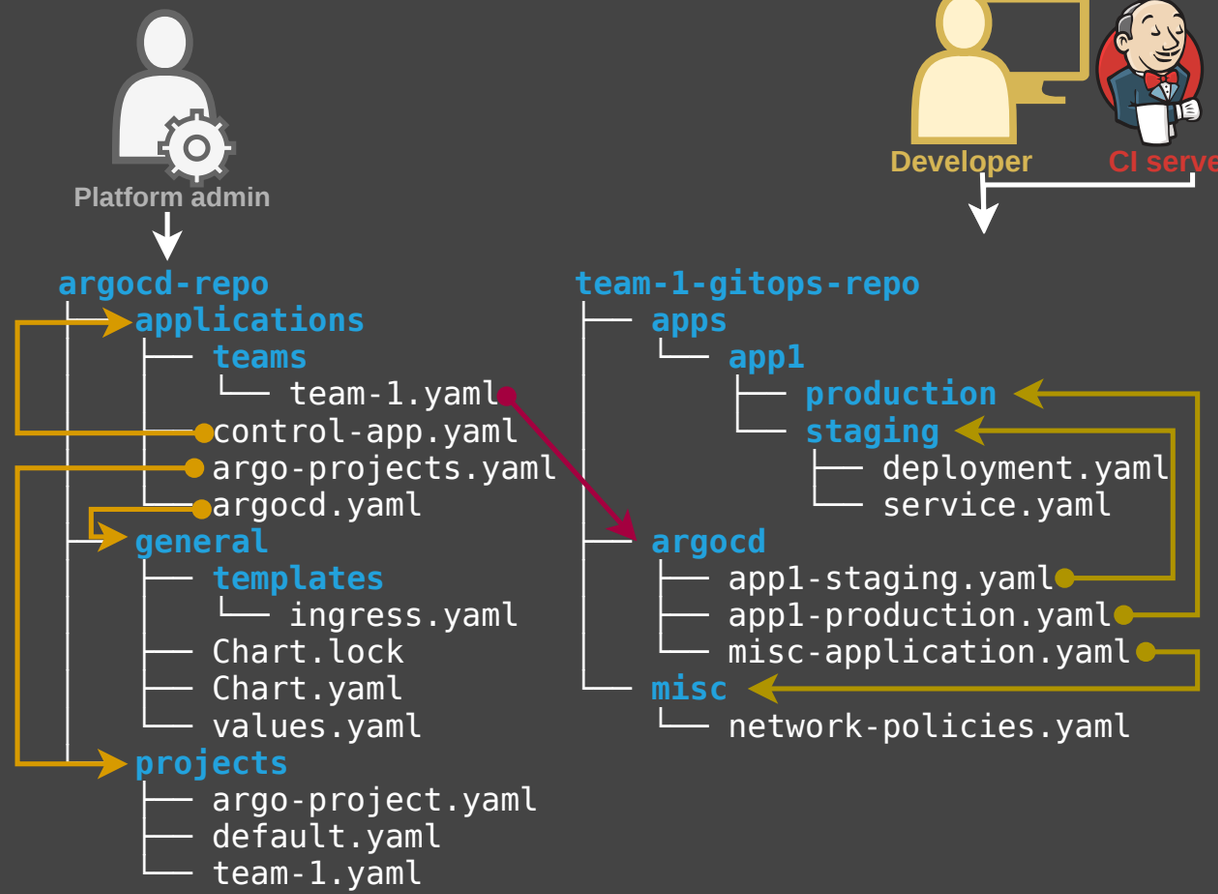
BTW: More Features to explore



More examples

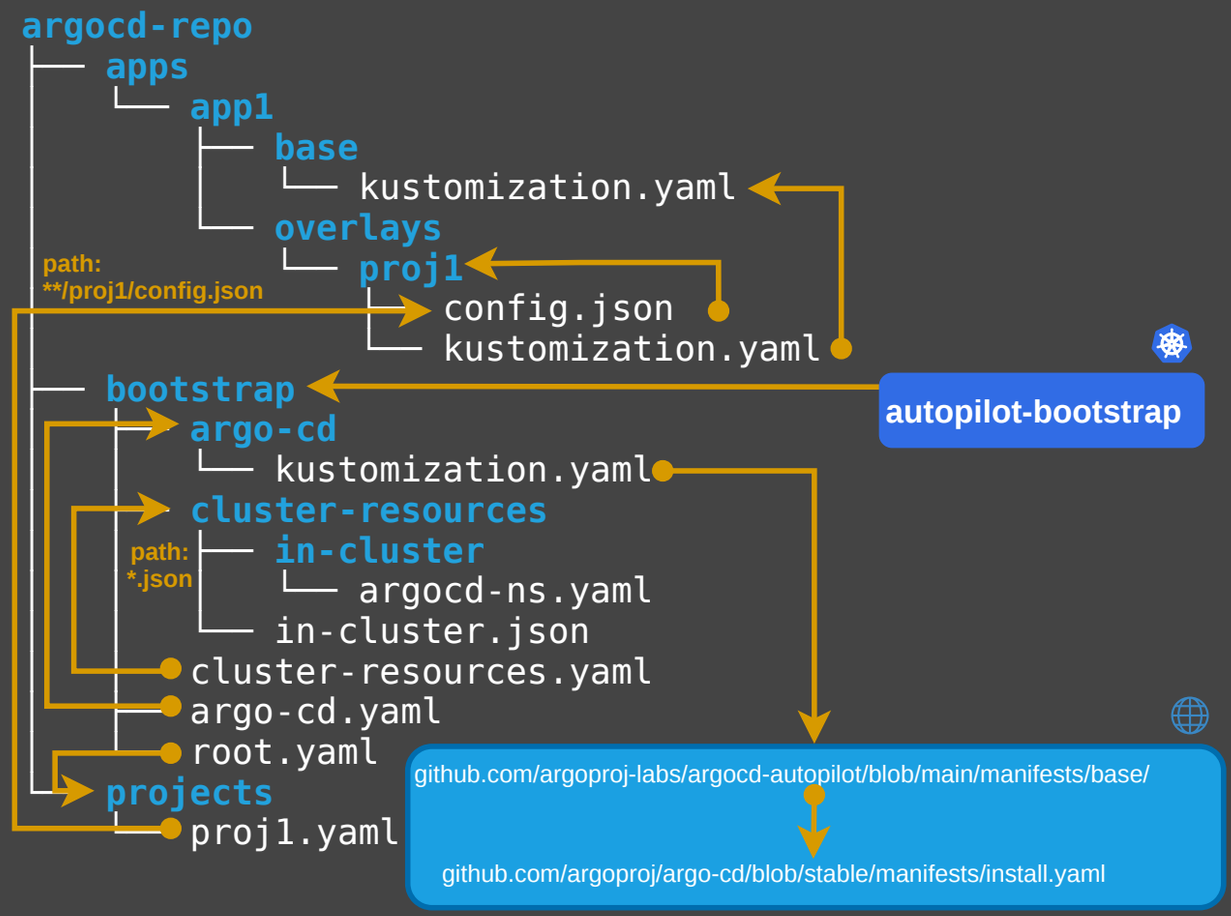
Example 2: Ex 1 with operator

- **Repo pattern:**
  Per team/monorepo per app
- **Operator pattern:** Hub and Spoke
- **Operator:** ()
- **Boostrapping:** `Helm`, `kubectl`
- **Linking:** `Application`
- **Features:** Apps with individual envs, operate ArgoCD with GitOps
- **Source:** Cloudogu internal, GitOps Playground in the future



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
  - In the future: a lot more automation and YAML creation
- **Source:** argoproj-labs/argocd-autopilot

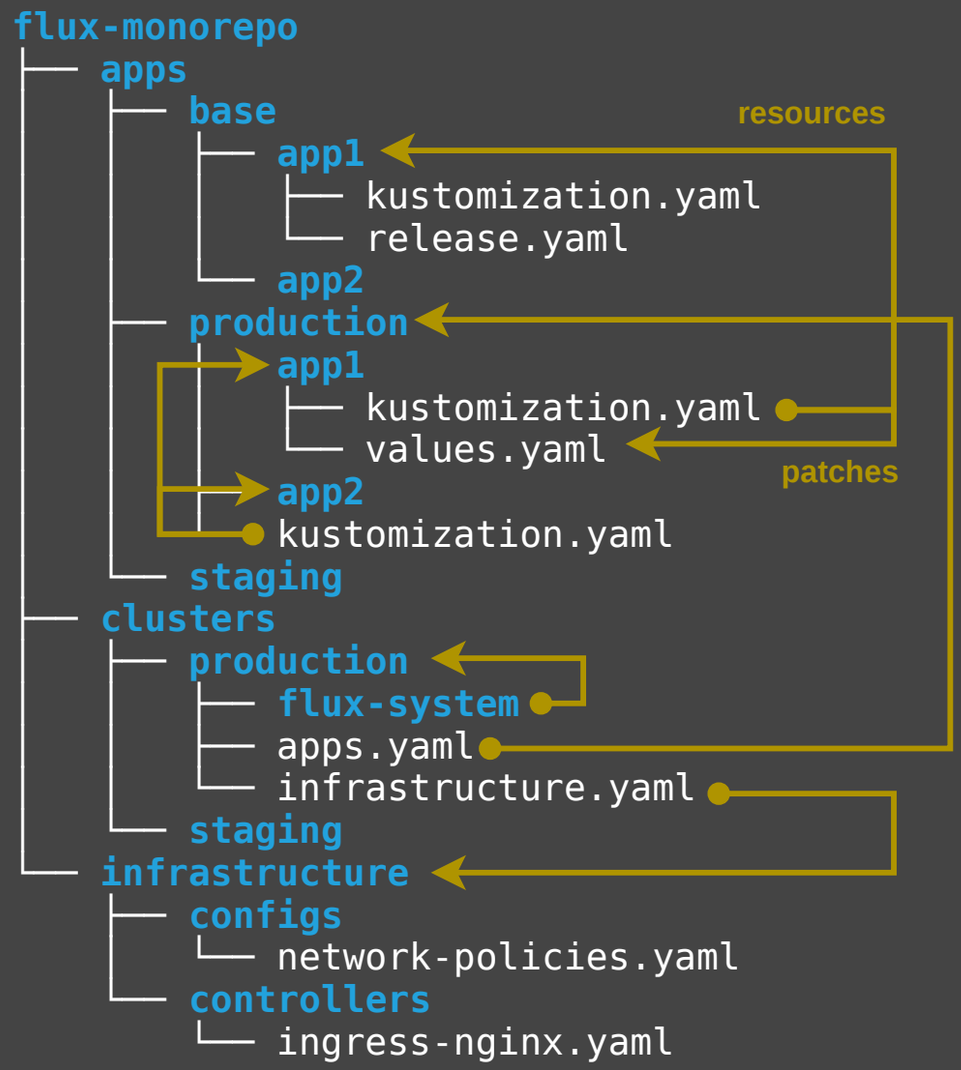


Example 4: Flux Monorepo

- **Repo pattern:** Monorepo
- **Operator pattern:** Standalone
- **Operator:** (?)
- **Boostrapping:** `flux`
- **Linking:** `Kustomization`, K
- **Features:** cross-cutting infra
- **Source:**

fluxcd/flux2-kustomize-helm-example#16

fluxcd.io/flux/guides/repository-structure

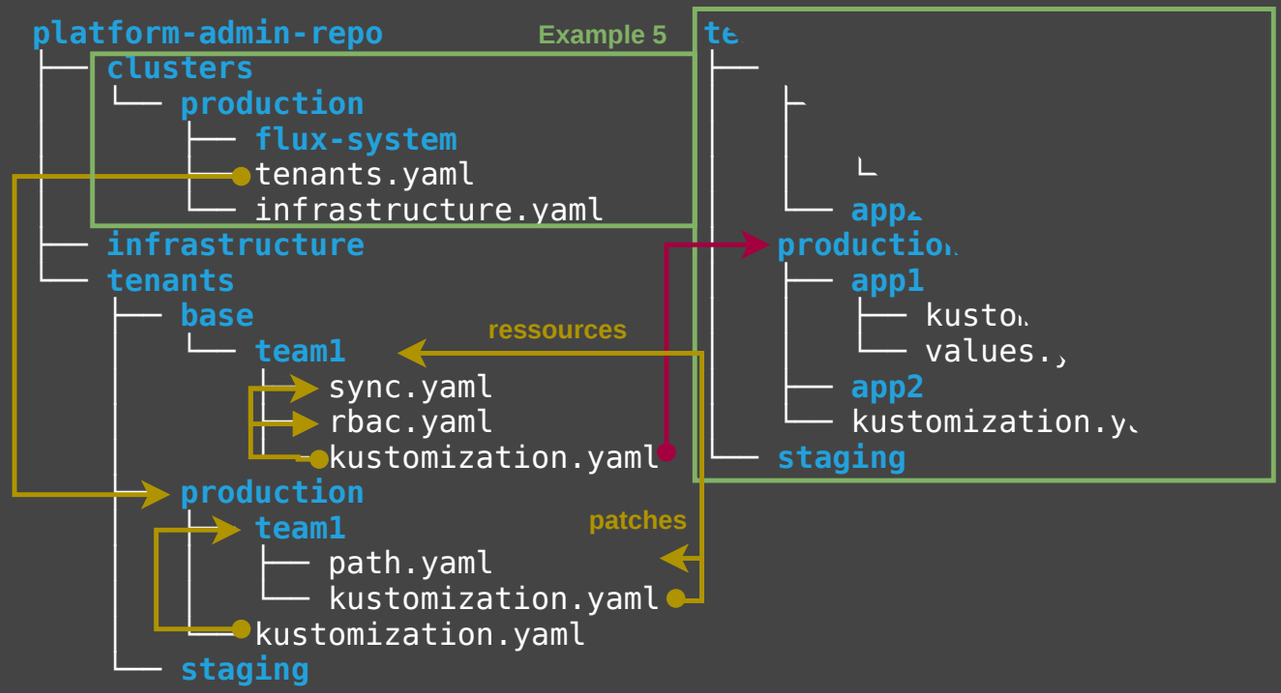


Example 5: Flux repo per team

- **Repo pattern:** Repo per team
- **Operator pattern:** Standalone
- **Operator:** (?)
- **Boostrapping:** `flux`
- **Linking:** `Kustomization`, K
- **Templating/Overlay:** K HELM
- **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 infra and app(s)
  - ArgoCD **and** Flux examples
- **Source:**

christianh814/example-kubernetes-go-repo

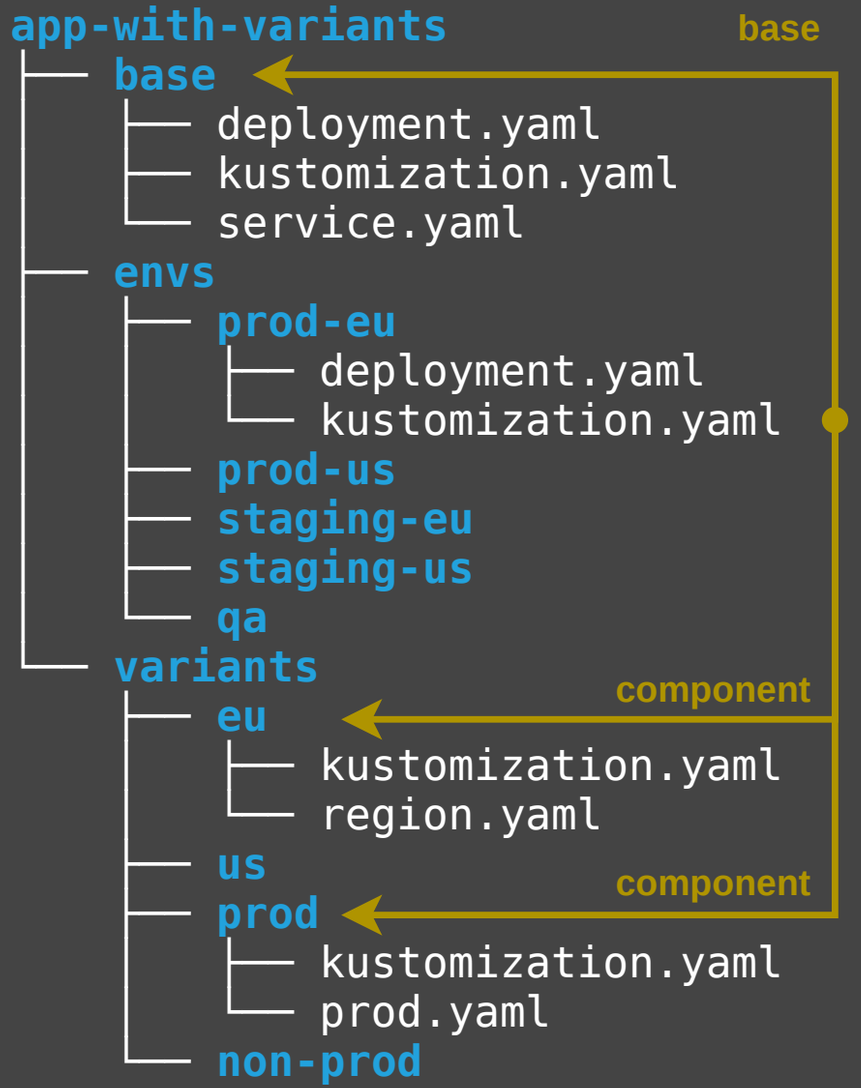
C. Hernandez - The Path to GitOps

```
monorepo
└── cluster-XXXX
    ├── apps
    │   └── myapp
    │       ├── kustomization.yaml
    │       └── myapp-deployment.yaml
    ├── bootstrap
    │   ├── base
    │   │   ├── argocd-ns.yaml
    │   │   └── kustomization.yaml
    │   └── overlays
    │       └── default
    │           └── kustomization.yaml
    ├── cluster-config
    │   ├── gitops-controller
    │   │   └── kustomization.yaml
    │   └── sample-admin-workload
    │       ├── kustomization.yaml
    │       └── sample-admin-config.yaml
    └── components
        ├── applicationsets
        │   ├── apps-appset.yaml
        │   ├── cluster-config-appset.yaml
        │   └── kustomization.yaml
        └── argocdproj
            ├── kustomization.yaml
            └── test-project.yaml
```

Example 7: Environment variants

- **Repo pattern:** *
- **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 scopes (bootstrapping, apps only, etc.)
- Not very much examples on automation (e.g. CI)
- K operator-agnostic choice for promotion

➡️ Use as inspiration, no silver bullet

Johannes Schnatterer, Cloudogu GmbH

☁️ cloudogu.com/gitops

- GitOps Resources
- Community
- Trainings
- Consulting

Slides

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

@schnatterer@floss.social @jschnatterer

Image sources

- coloured-parchment-paper background by brgfx on Freepik
  https://www.freepik.com/free-vector/coloured-parchment-paper-designs_1078492.htm
- Basics:
  https://pixabay.com/illustrations/blackboard-board-school-chalkboard-5639925/
- Example:
  https://unsplash.com/photos/X2PWhiKDQww
- More examples
  https://unsplash.com/photos/XZc4f2XZc84
- Perfect?
  https://pixabay.com/illustrations/question-mark-question-response-1020165/