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

Johannes Schnatterer, Cloudogu GmbH

@schnatterer@floss.social

@jschnatterer

Version: 202303021327-a4b0478

## Agenda

1 GitOps process 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

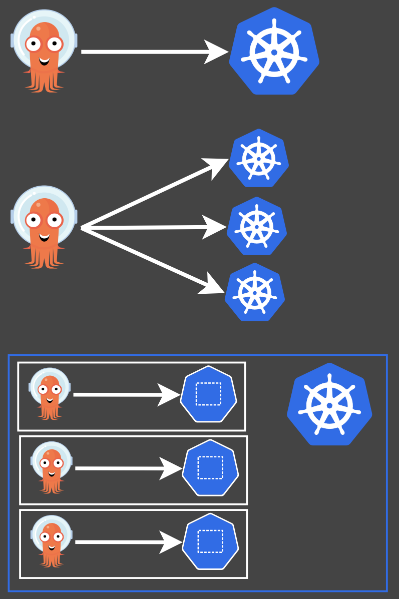
AKA strategies, models, approaches, best practices

- 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 Operator : 1 Cluster
- *Hub and Spoke*: 1 Operator : n Clusters
- Namespaced: n Operators : 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

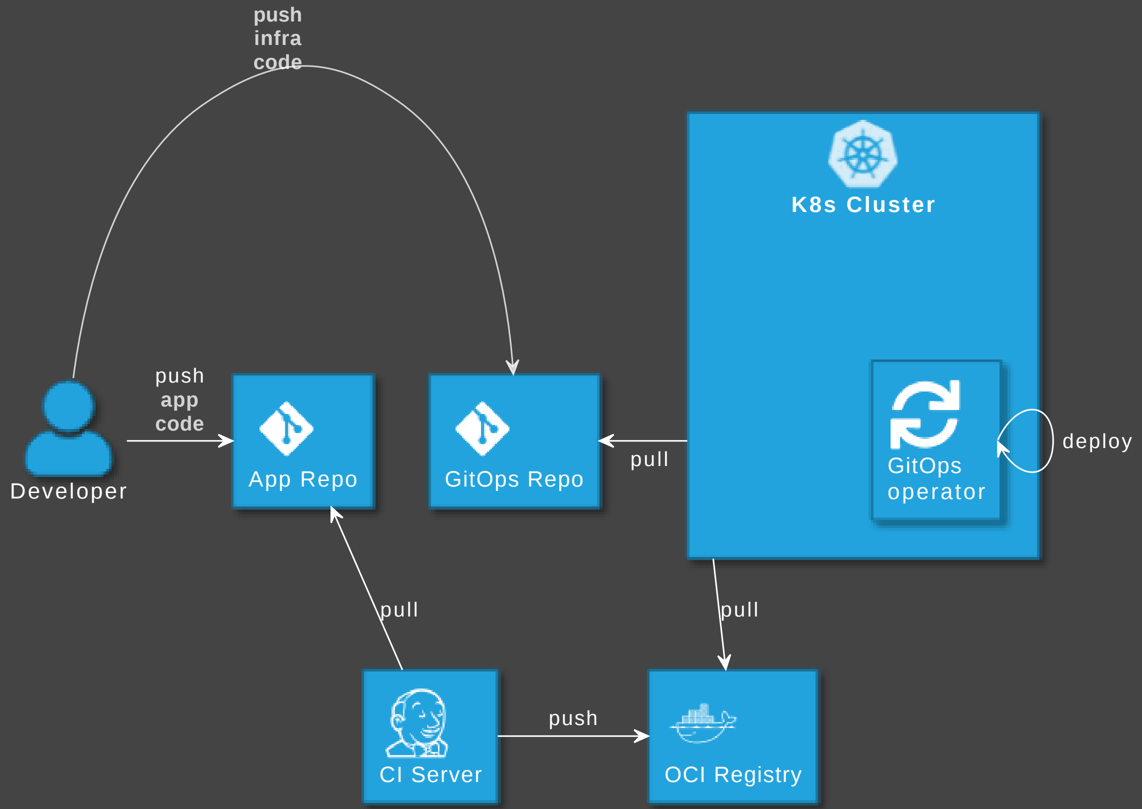
| | GitOps repo | App repo |
|---|---|---|
| Content | IaC/Manifests/YAMLs | Application source code |
| Synonyms | • Config repo | • Source code repo |
| | • Infra repo | • Source repo |
| | • Payload repo | |

Example

```
gitops-repo
├── app1
│   ├── deployment.yaml
│   └── service.yaml
└── app2
    ├── deployment.yaml
    └── service.yaml
```

```
app-repo
├── src
├── test
├── CI.pipeline
├── Dockerfile
├── package.json
└── pom.xml
```

## Separating GitOps repo from app repo



GitOps tools: Put infra in separate repo! See

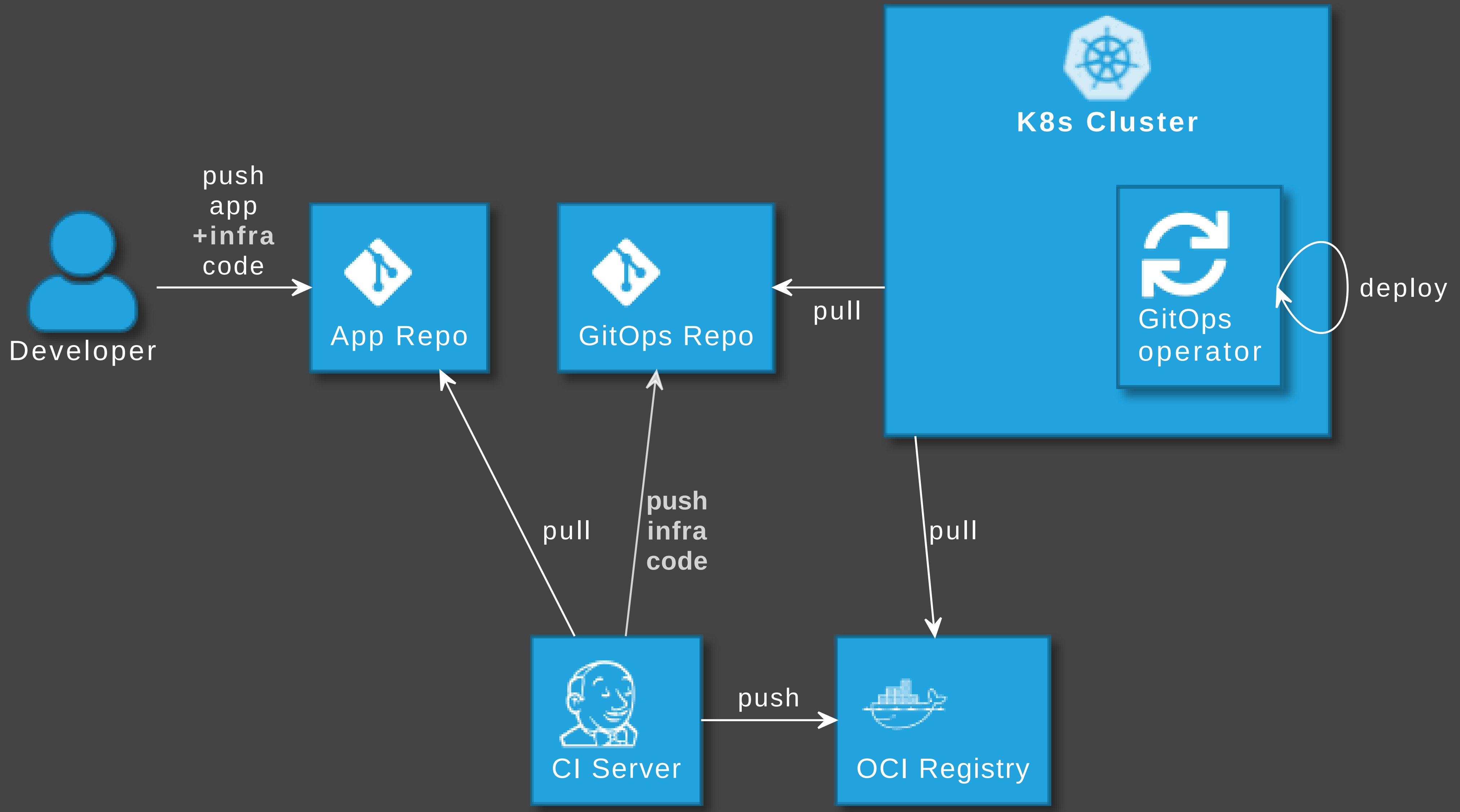
argo-cd.readthedocs.io/en/release-2.6/user-guide/best_practices

## 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

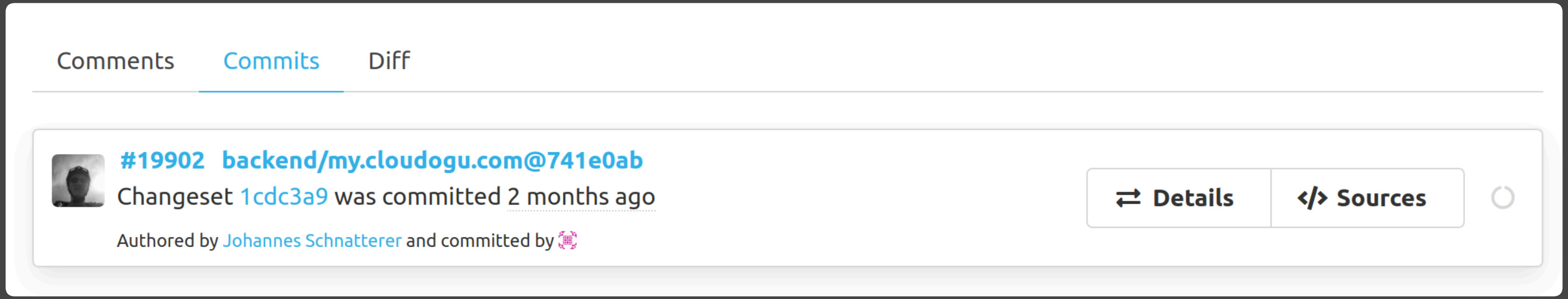
# How to avoid those?

# 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, security scanners
- Automated staging (e.g. PR creation) 🕒
- Simplify review by adding info to PRs



#19902 backend/my.cloudogu.com@741e0ab
Changeset 1cdc3a9 was committed 2 months ago
Authored by Johannes Schnatterer and committed by

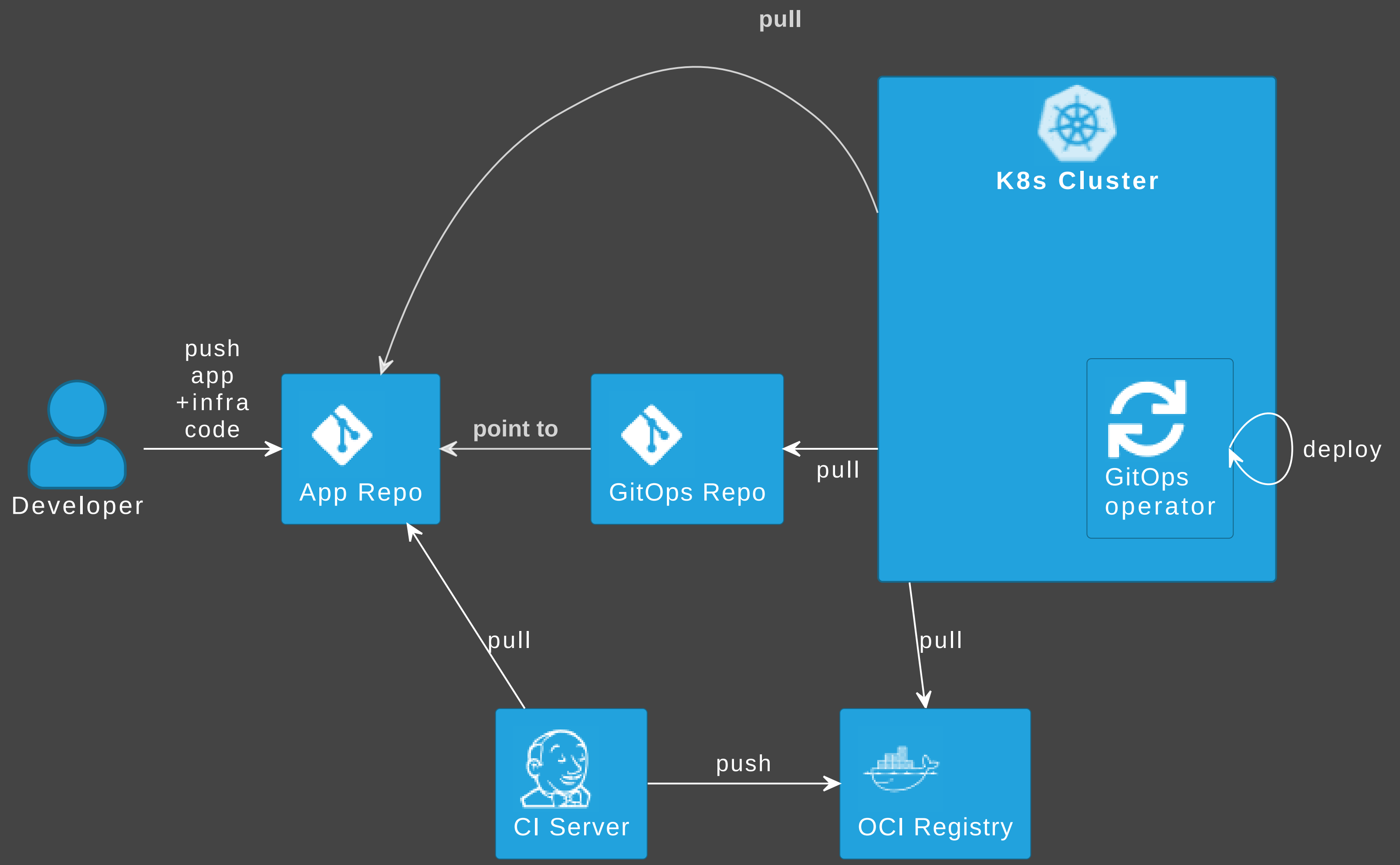
## Disadvantages

- 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/Directory per environment*
- *Branch per environment* (anti-pattern)
- *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 (unidrectional)
- 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

AKA Templating, Patching, Overlay, Rendering?

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

## 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
```

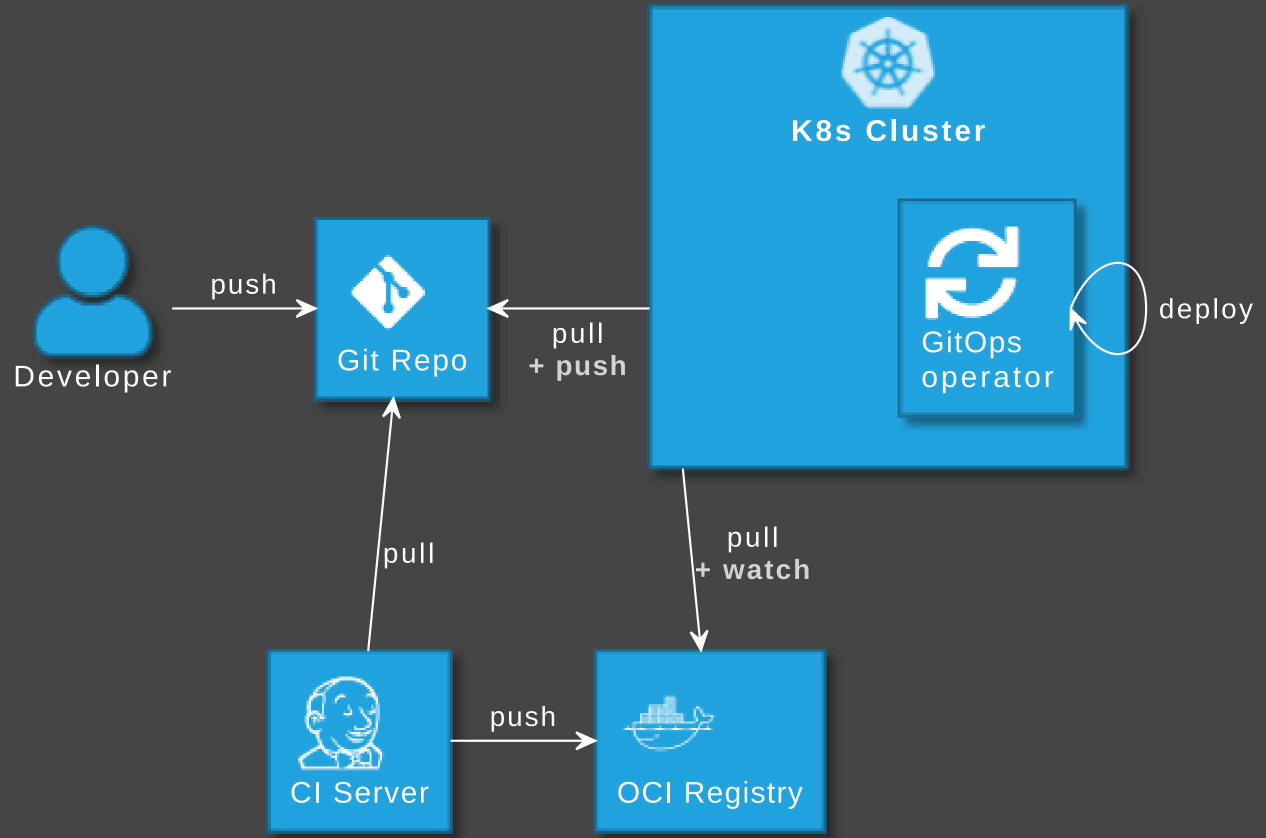
e.g. Preview Envs

## 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

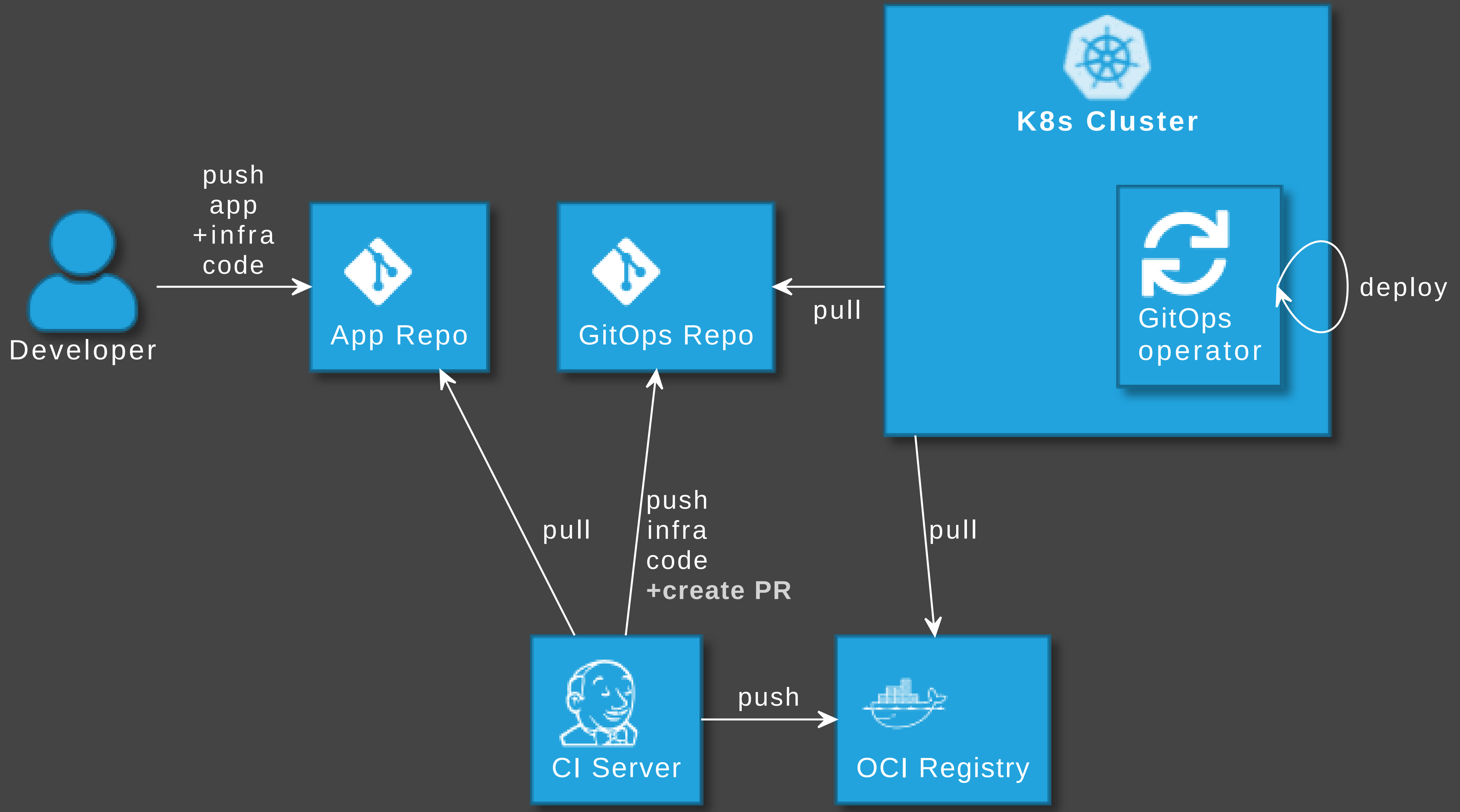
## Image updater



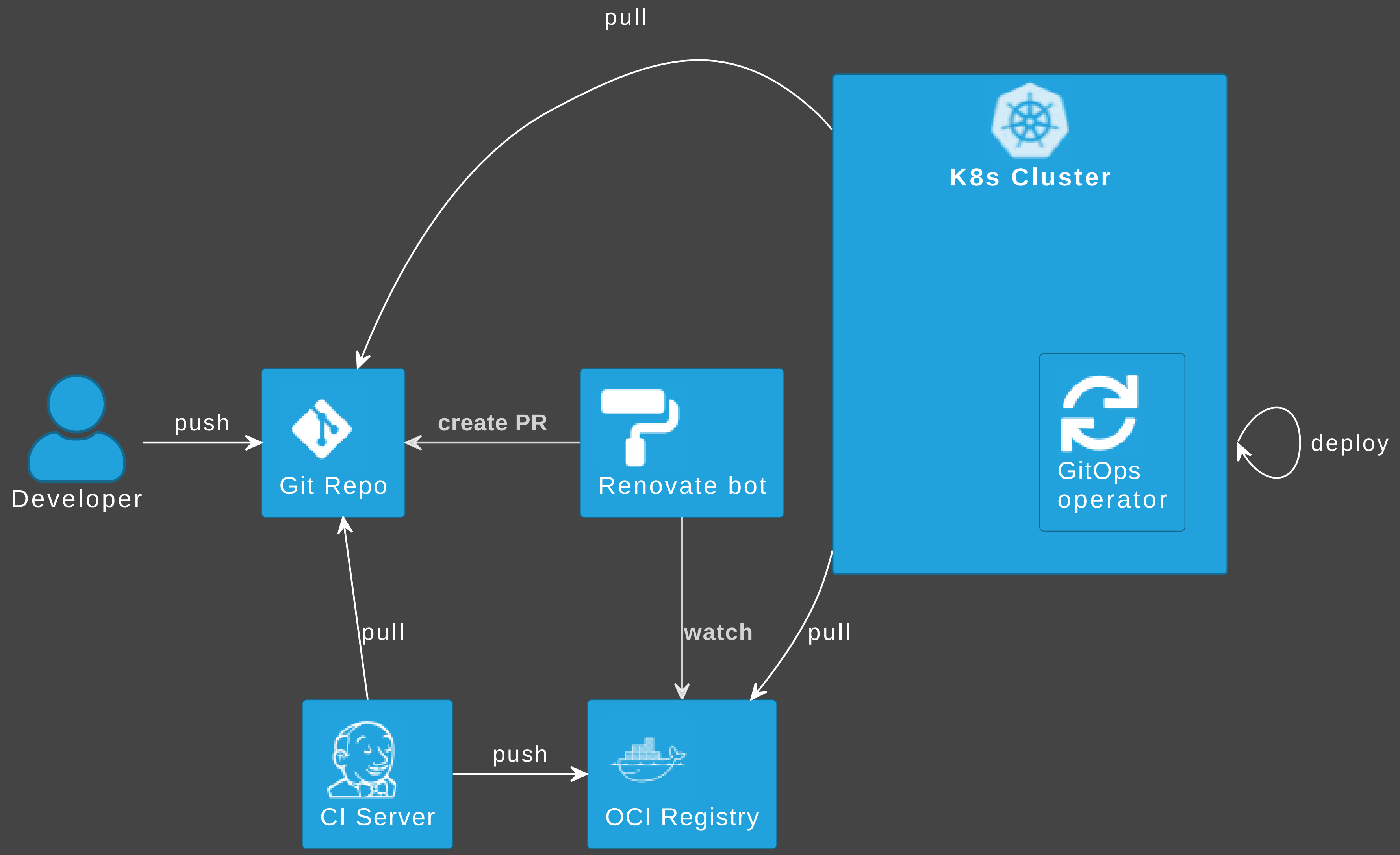
GitOps operator can update image version in Git

- 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

GitOpsSets ❓

## Wiring

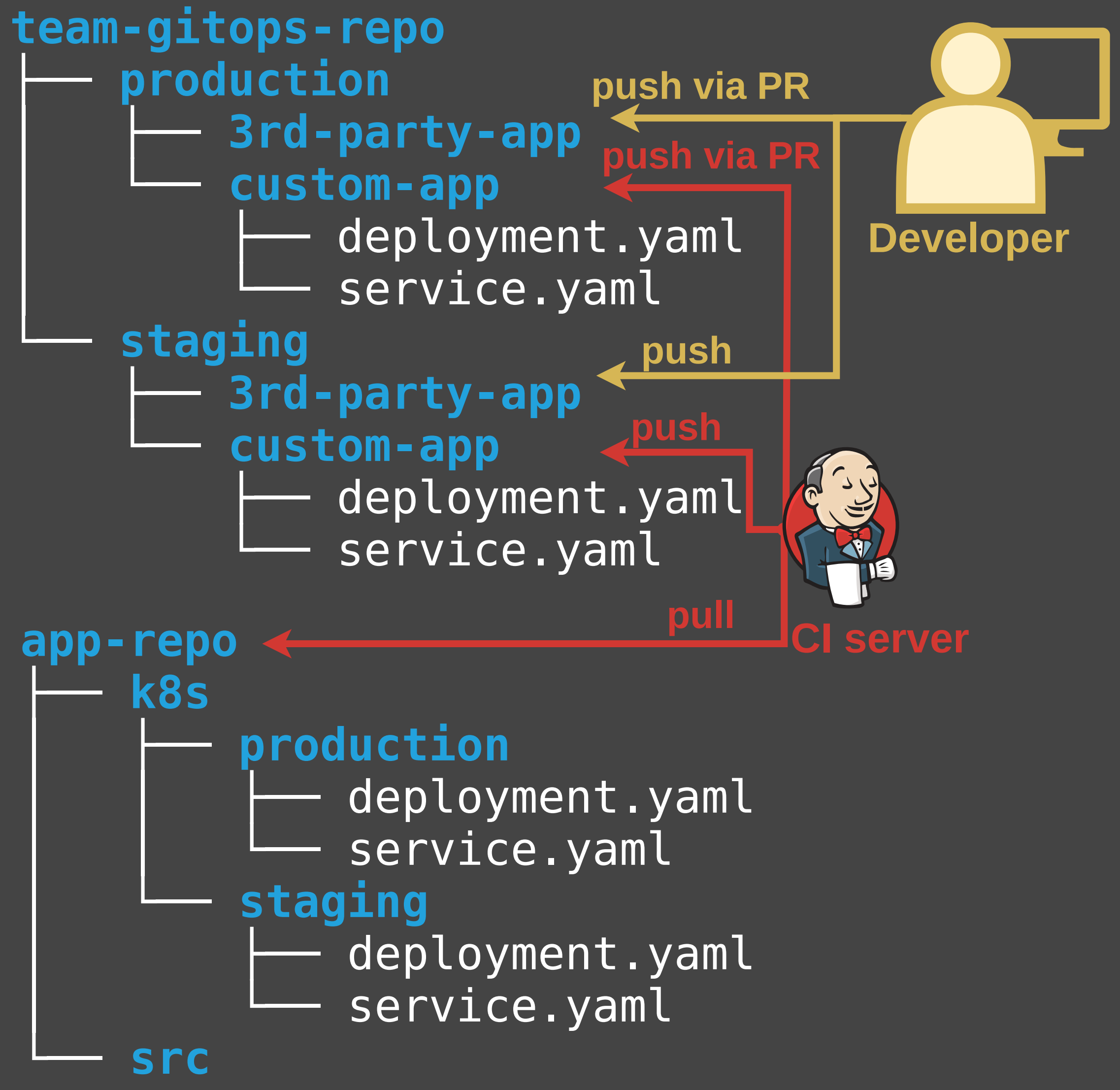
Wiring up operator, repos, folders, envs, etc.

- 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

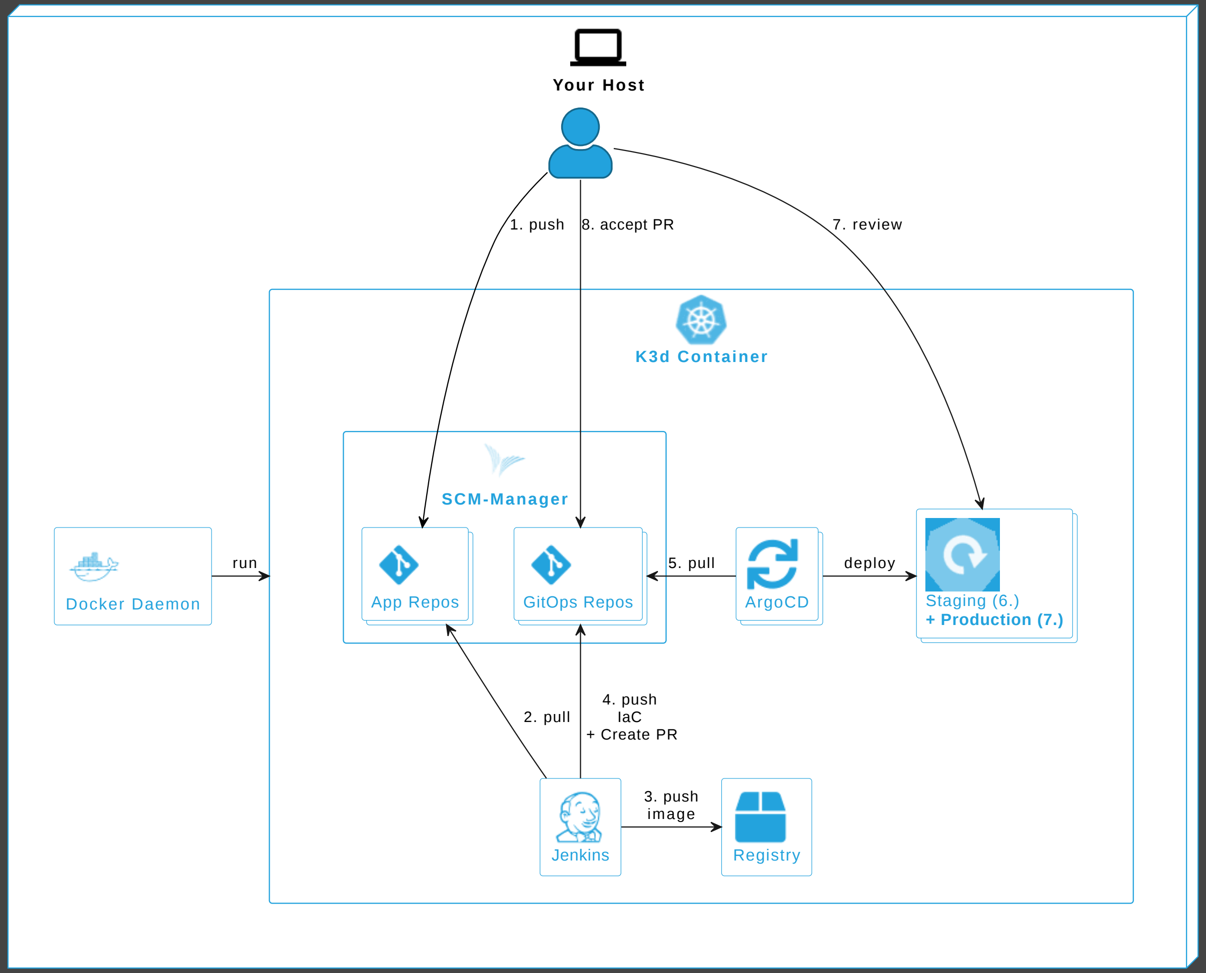
# Example 1: Repo per team and app + CI

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



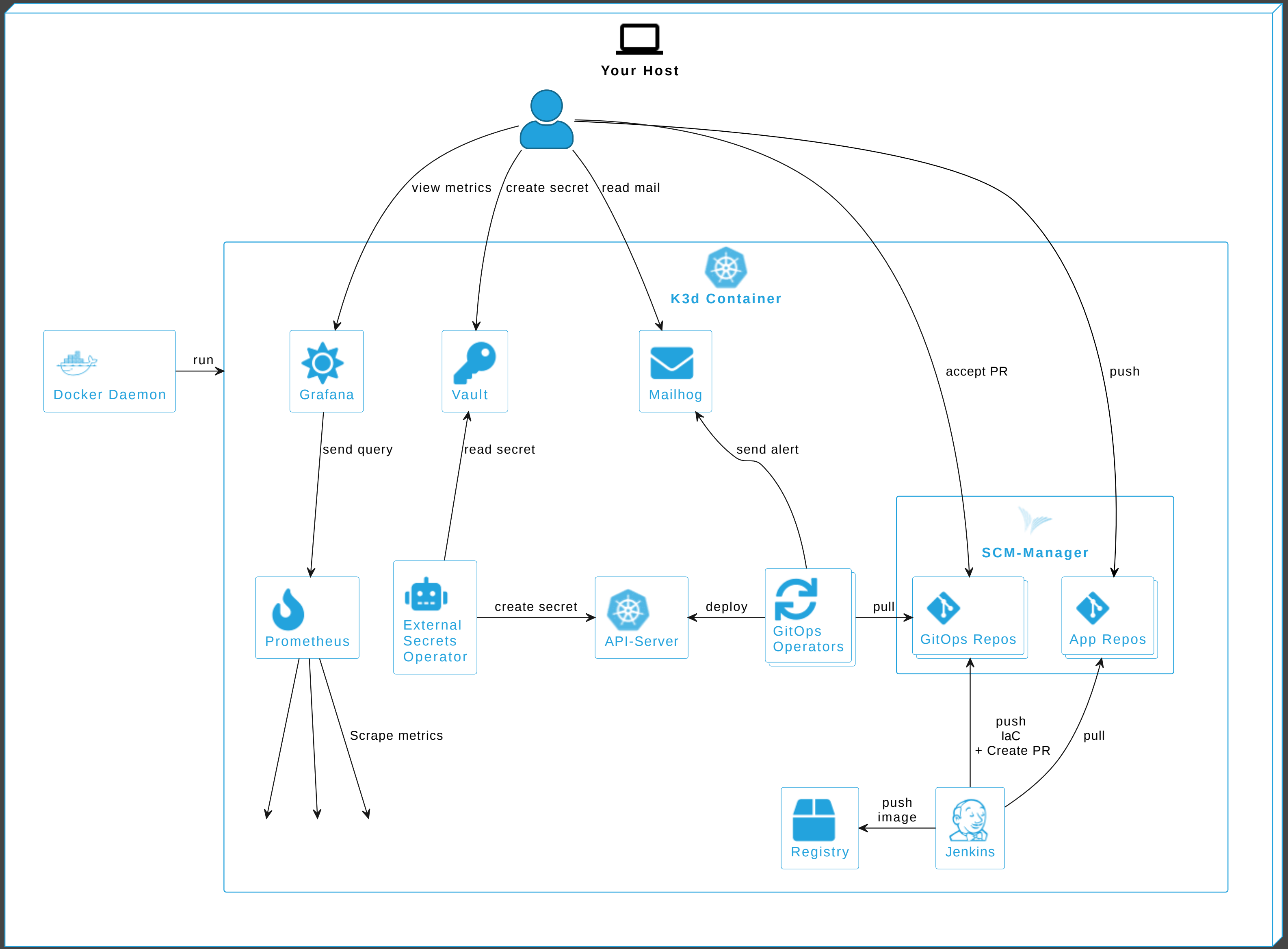
```
team-gitops-repo
├── production
│   ├── 3rd-party-app
│   └── custom-app
│       ├── deployment.yaml
│       └── service.yaml
└── staging
    ├── 3rd-party-app
    └── custom-app
        ├── deployment.yaml
        └── service.yaml

app-repo
├── k8s
│   ├── production
│   │   ├── deployment.yaml
│   │   └── service.yaml
│   └── staging
│       ├── deployment.yaml
│       └── service.yaml
└── src
```

# Demo



cloudogu/gitops-playground

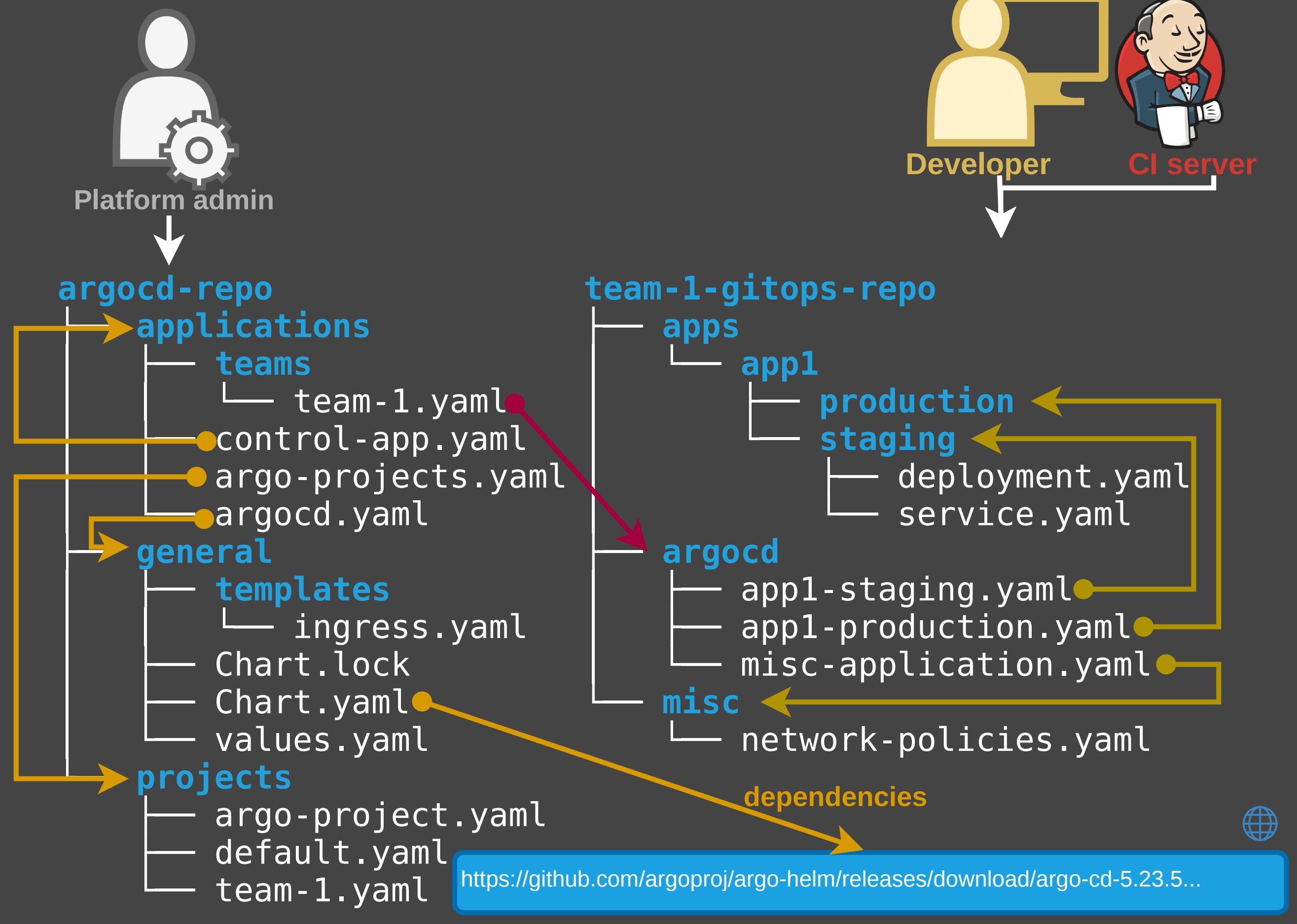
# BTW: More Features to explore



## 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:** Env per app, operate ArgoCD with GitOps
- **Source:** Cloudogu internal, GitOps Playground in the future

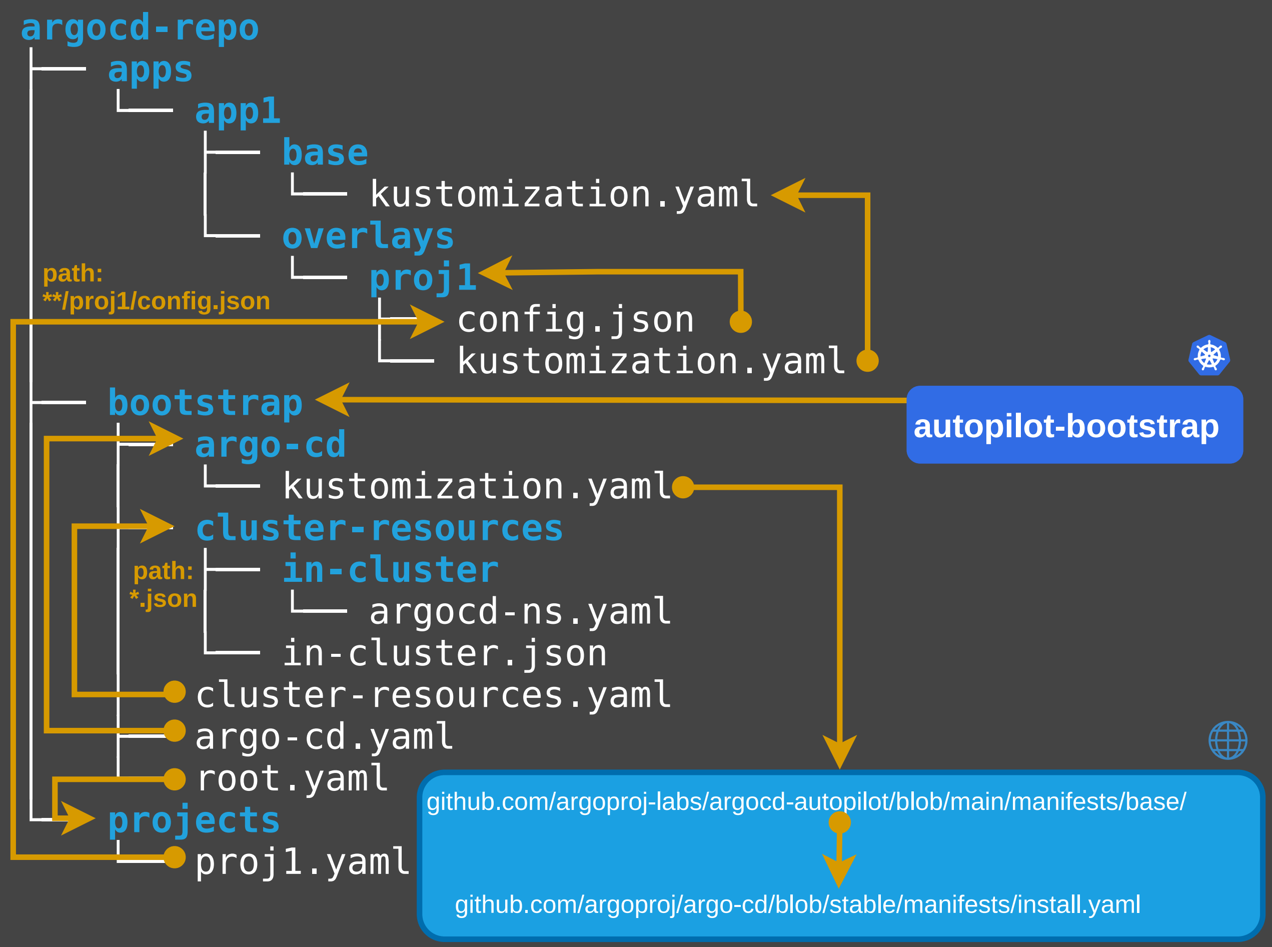

Platform admin
Developer
CI server
dependencies
https://github.com/argoproj/argo-helm/releases/download/argo-cd-5.23.5...

```
argocd-repo
├── applications
│   ├── teams
│   │   └── team-1.yaml
│   ├── control-app.yaml
│   ├── argo-projects.yaml
│   └── argocd.yaml
├── general
│   ├── templates
│   │   └── ingress.yaml
│   ├── Chart.lock
│   ├── Chart.yaml
│   └── values.yaml
└── projects
    ├── argo-project.yaml
    ├── default.yaml
    └── team-1.yaml
```

```
team-1-gitops-repo
├── apps
│   └── app1
│       ├── production
│       └── staging
│           ├── deployment.yaml
│           └── service.yaml
├── argocd
│   ├── app1-staging.yaml
│   ├── app1-production.yaml
│   └── misc-application.yaml
└── misc
    └── network-policies.yaml
```

# Example 3: ArgoCD autopilot

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



```
argocd-repo
├── apps
│   └── app1
│       ├── base
│       │   └── kustomization.yaml
│       └── overlays
│           └── proj1
│               ├── config.json
│               └── kustomization.yaml
├── bootstrap
│   ├── argo-cd
│   │   └── kustomization.yaml
│   ├── cluster-resources
│   │   ├── in-cluster
│   │   │   └── argocd-ns.yaml
│   │   └── in-cluster.json
│   ├── cluster-resources.yaml
│   ├── argo-cd.yaml
│   └── root.yaml
└── projects
    └── proj1.yaml
```

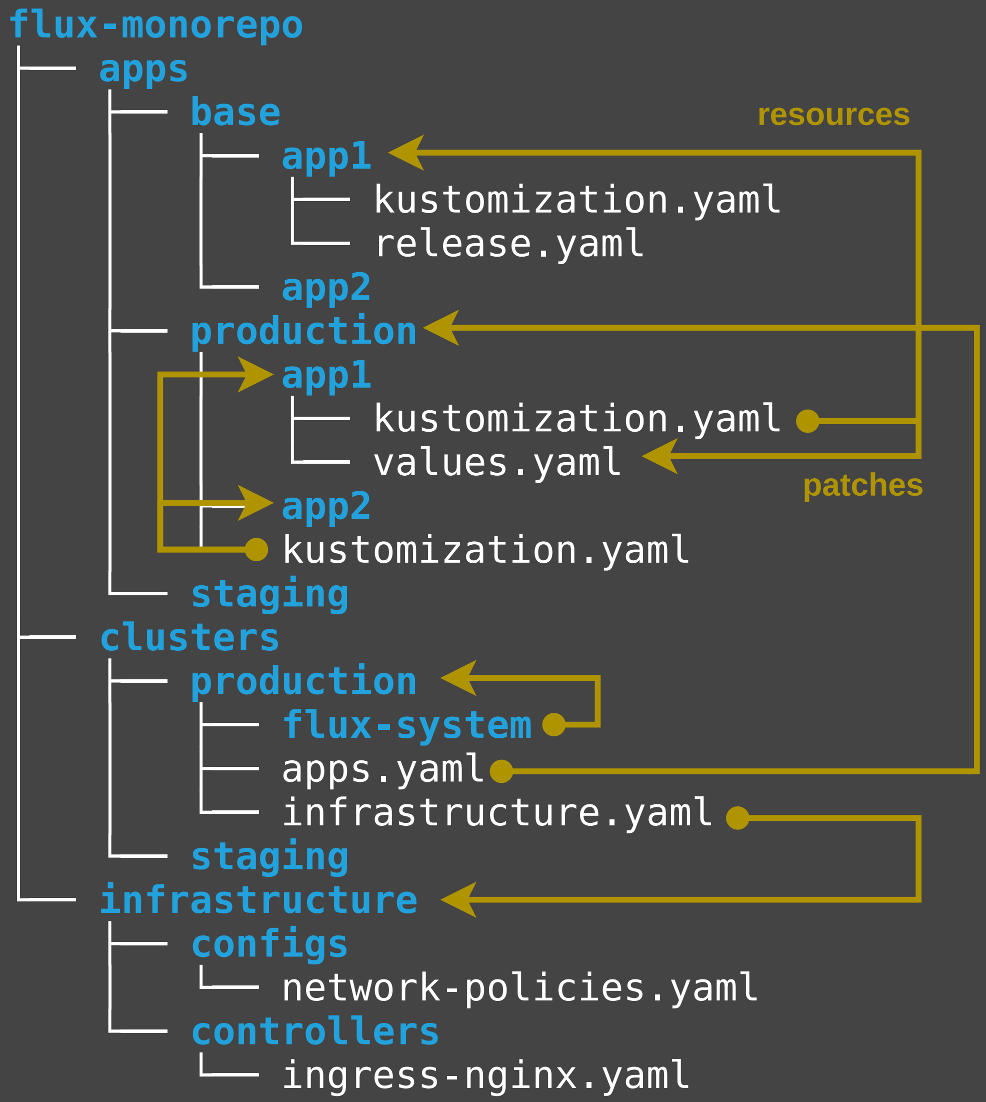
path:
**/proj1/config.json
path:
*.json
autopilot-bootstrap
github.com/argoproj-labs/argocd-autopilot/blob/main/manifests/base/
github.com/argoproj/argo-cd/blob/stable/manifests/install.yaml

# Example 4: Flux Monorepo

- **Repo pattern:** Monorepo
- **Operator pattern:** Standalone
- **Operator:** (?)
- **Boostrapping:** `flux`
- **Linking:** `Kustomization`,
- **Features:**
  - Cross-cutting infra
  - Operate Flux with GitOps
- **Source:**

fluxcd/flux2-kustomize-helm-example#16

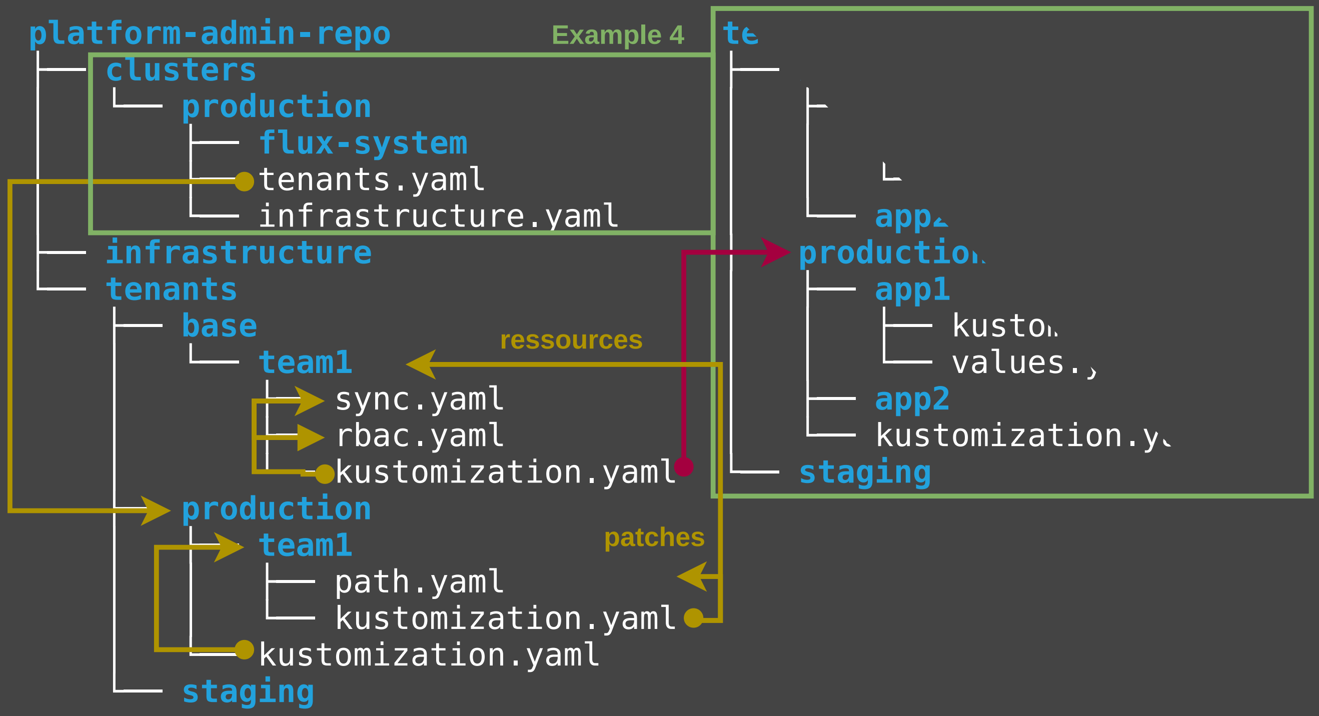
fluxcd.io/flux/guides/repository-structure



```
flux-monorepo
├── apps
│   ├── base
│   │   ├── app1
│   │   │   ├── kustomization.yaml
│   │   │   └── release.yaml
│   │   └── app2
│   ├── production
│   │   ├── app1
│   │   │   ├── kustomization.yaml
│   │   │   └── values.yaml
│   │   ├── app2
│   │   kustomization.yaml
│   └── staging
├── clusters
│   ├── production
│   │   ├── flux-system
│   │   ├── apps.yaml
│   │   └── infrastructure.yaml
│   └── staging
└── infrastructure
    ├── configs
    │   └── network-policies.yaml
    └── controllers
        └── ingress-nginx.yaml
```

# Example 5: Flux repo per team

- **Repo pattern:** Repo per team
- **Operator pattern:** Standalone
- **Operator:** (?)
- **Boostrapping:** `flux`
- **Linking:** `Kustomization`,
- **Features:** Ex 5 with repo for team
- **Source:**
  - fluxcd/flux2-multi-tenancy
  - fluxcd.io/flux/guides/repository-structure



```
platform-admin-repo
├── clusters
│   └── production
│       ├── flux-system
│       ├── tenants.yaml
│       └── infrastructure.yaml
├── infrastructure
└── tenants
    ├── base
    │   └── team1
    │       ├── sync.yaml
    │       ├── rbac.yaml
    │       └── kustomization.yaml
    ├── production
    │   ├── team1
    │   │   ├── path.yaml
    │   │   └── kustomization.yaml
    │   └── kustomization.yaml
    └── staging
```

```
te
├──
│   ├
│   │   └
│   └── app
├── productio
│   ├── app1
│   │   ├── kusto
│   │   └── values.
│   ├── app2
│   └── kustomization.y
└── staging
```

# Example 6: ArgoCD and Flux alternative

- **Repo pattern:** Monorepo
- **Operator pattern:** Standalone
- **Operator:**
- **Boostrapping:** `kubectl`
- **Linking:** `Application`, `ApplicationSet` / `Kustomization`, K
- **Features:**
  - Cross-cutting infra and app(s)
  - ArgoCD **and** Flux examples
- **Source:**

  christianh814/example-kubernetes-go-repo

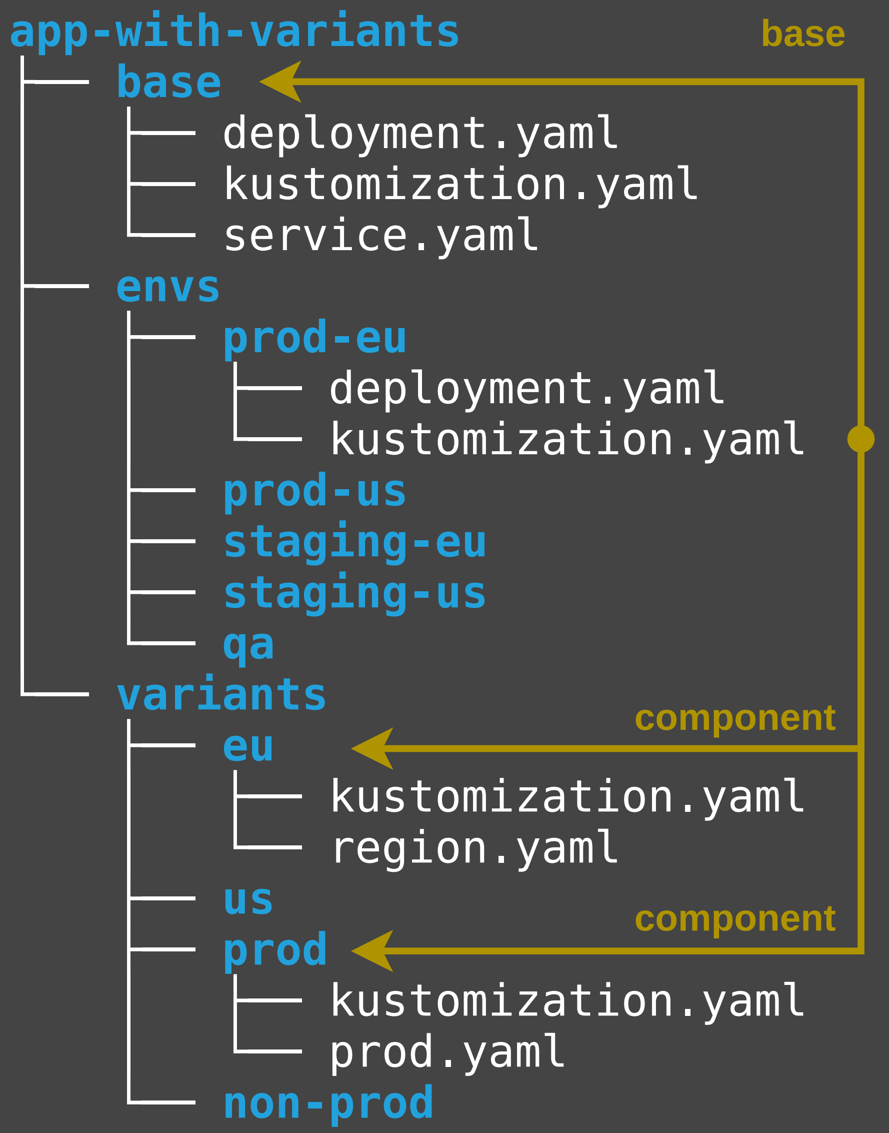
  C. Hernandez - The Path to GitOps

```
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 variations

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

kostis-codefresh/gitops-environment-promotion



```
app-with-variants
├── base
│   ├── deployment.yaml
│   ├── kustomization.yaml
│   └── service.yaml
├── envs
│   ├── prod-eu
│   │   ├── deployment.yaml
│   │   └── kustomization.yaml
│   ├── prod-us
│   ├── staging-eu
│   ├── staging-us
│   └── qa
└── variants
    ├── eu
    │   ├── kustomization.yaml
    │   └── region.yaml
    ├── us
    ├── prod
    │   ├── kustomization.yaml
    │   └── prod.yaml
    └── non-prod
```

## 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 @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/