// GITOPS REPO STRUCTURES AND PATTERNS

Johannes Schnatterer, Cloudogu GmbH

@schnatterer@floss.social in/jschnatterer @jschnatterer

Version: 202311091814-b1f6084

Categories of patterns

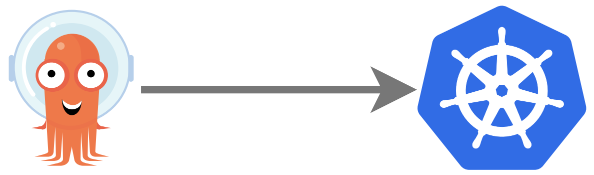
AKA strategies, models, approaches, best practices

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

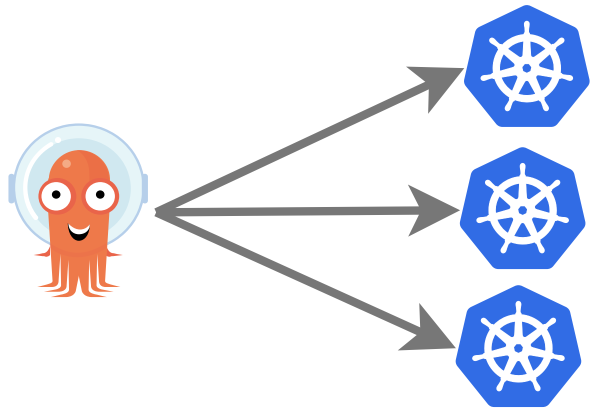
GitOps Operator deployment patterns

How many GitOps operators per cluster?

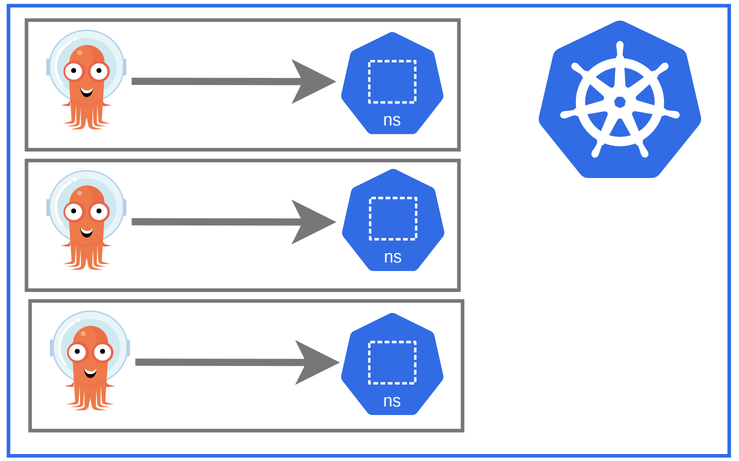
Instance per Cluster



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

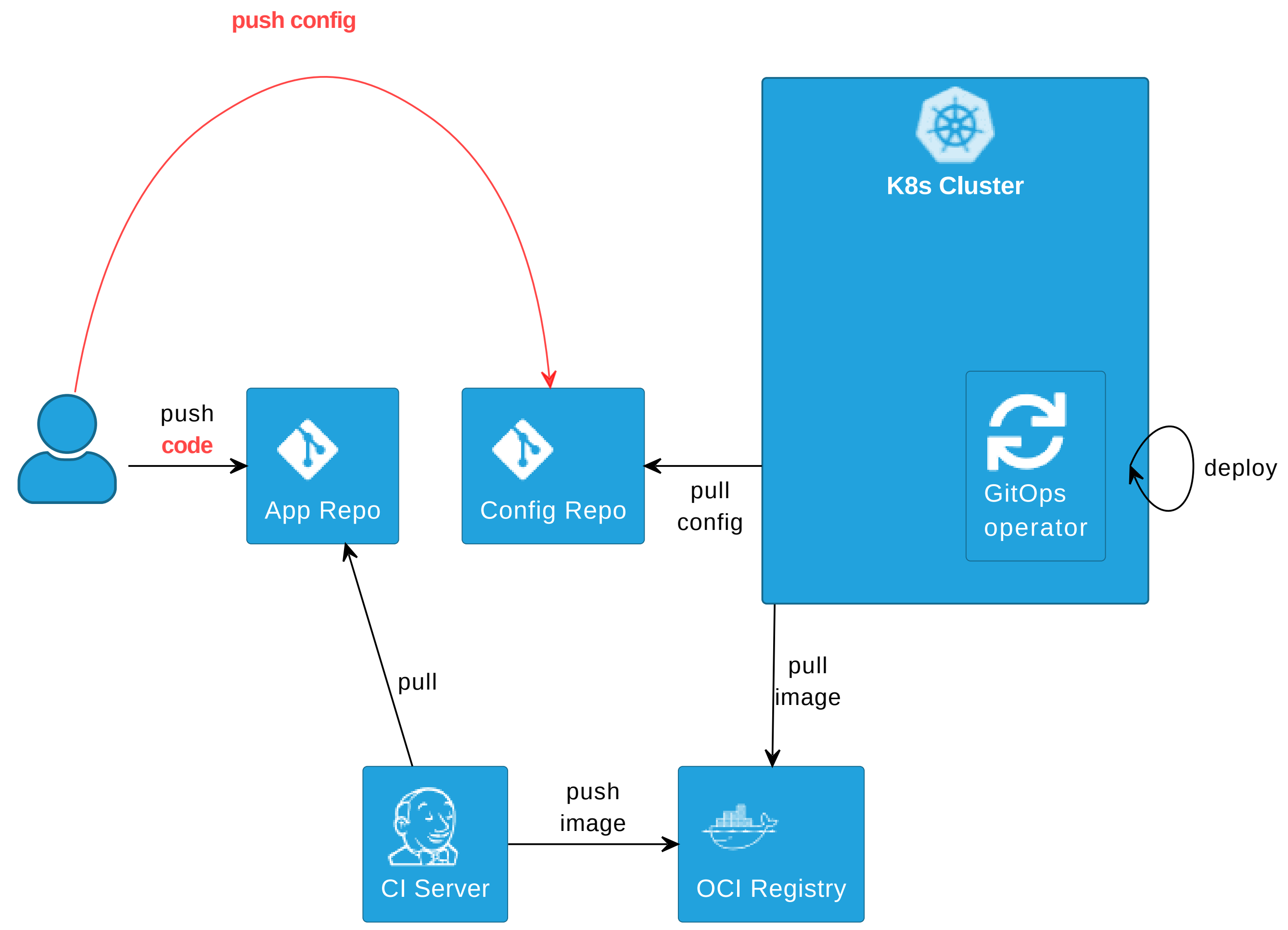
	Config repo	App repo
Content	Config/Manifests/YAMLs (IaC)	Application source code
Synonyms	• GitOps repo • Infra repo • Environment repo • Payload repo	• Source code repo • Source repo

Example

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

```
app-repo
├── src
├── test
├── Dockerfile
├── package.json
├── pom.xml
└── some-ci.yaml
```

Repo Separation



Recommendation: Keep config separate from code

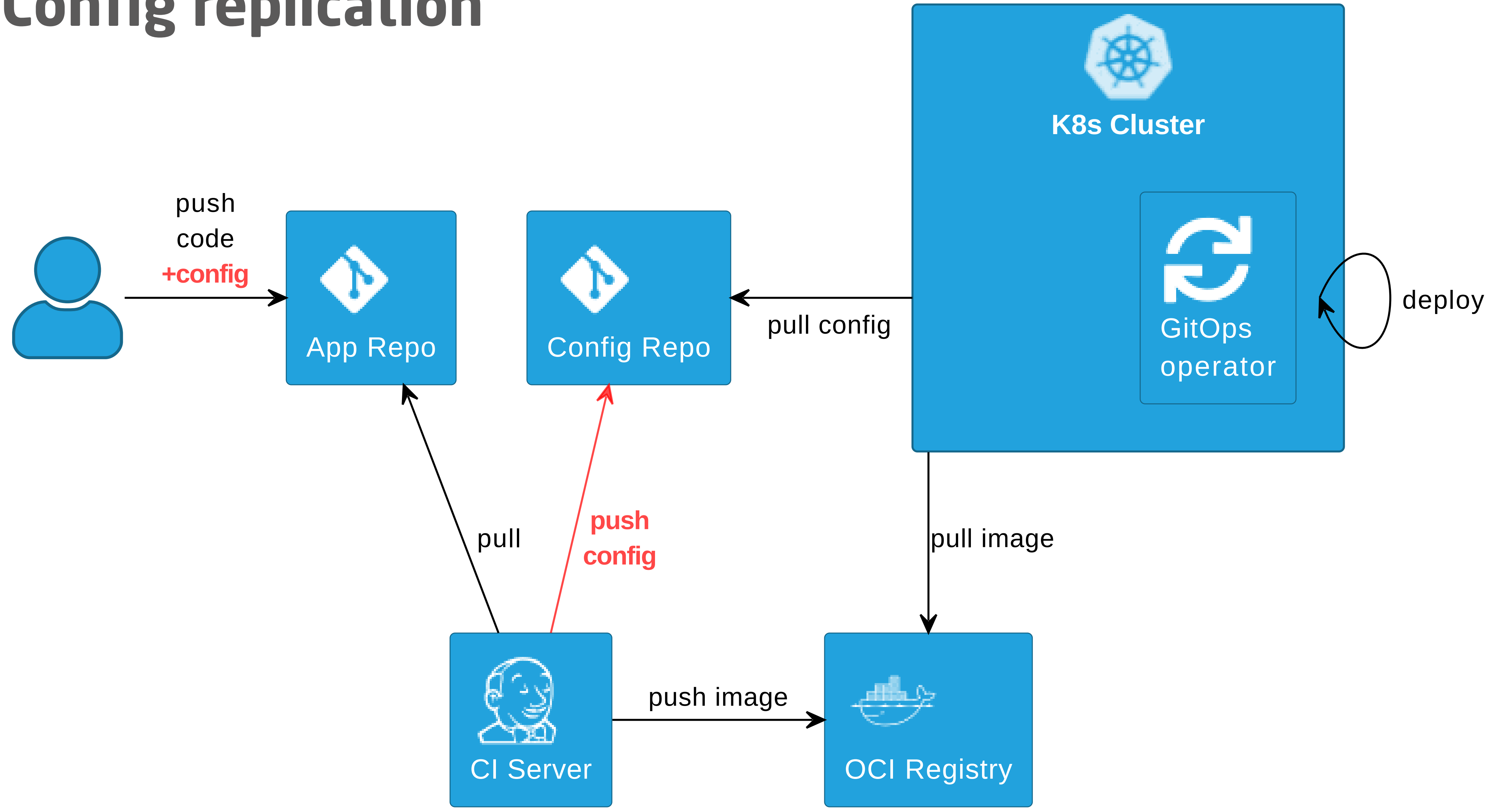
argo-cd.readthedocs.io/en/release-2.8/user-guide/best_practices

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

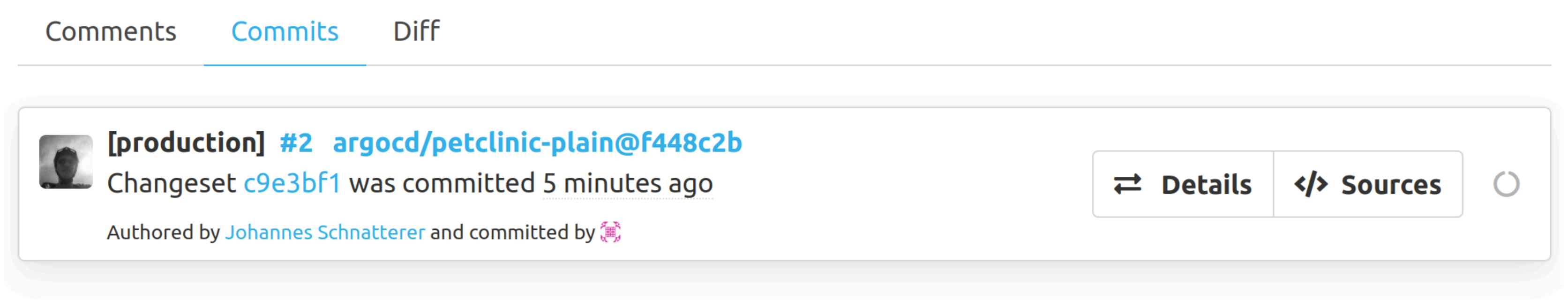
How to avoid those?

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

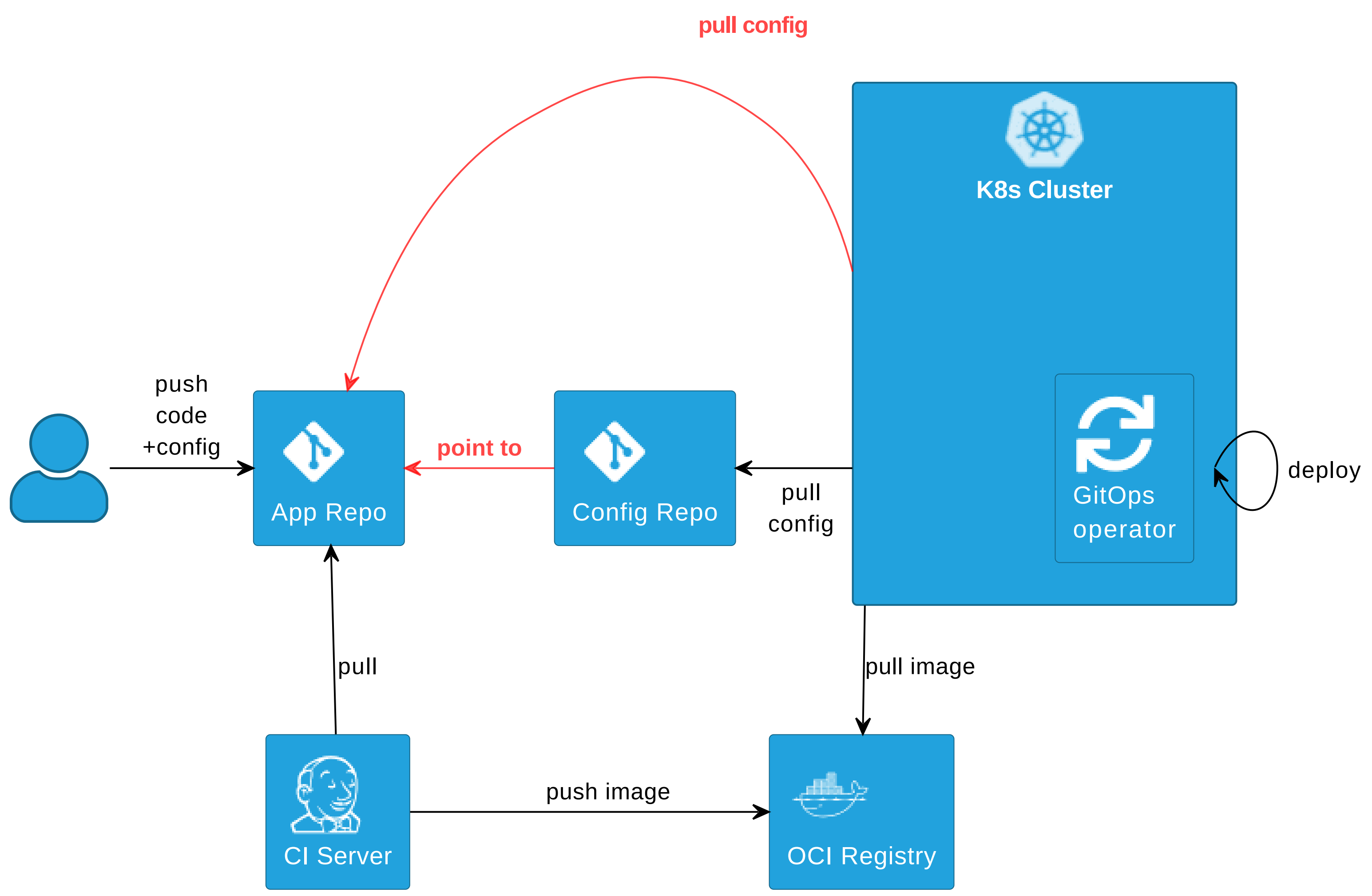


[production] #2 argocd/petclinic-plain@f448c2b
Changeset c9e3bf1 was committed 5 minutes ago
Authored by Johannes Schnatterer and committed by

Disadvantages

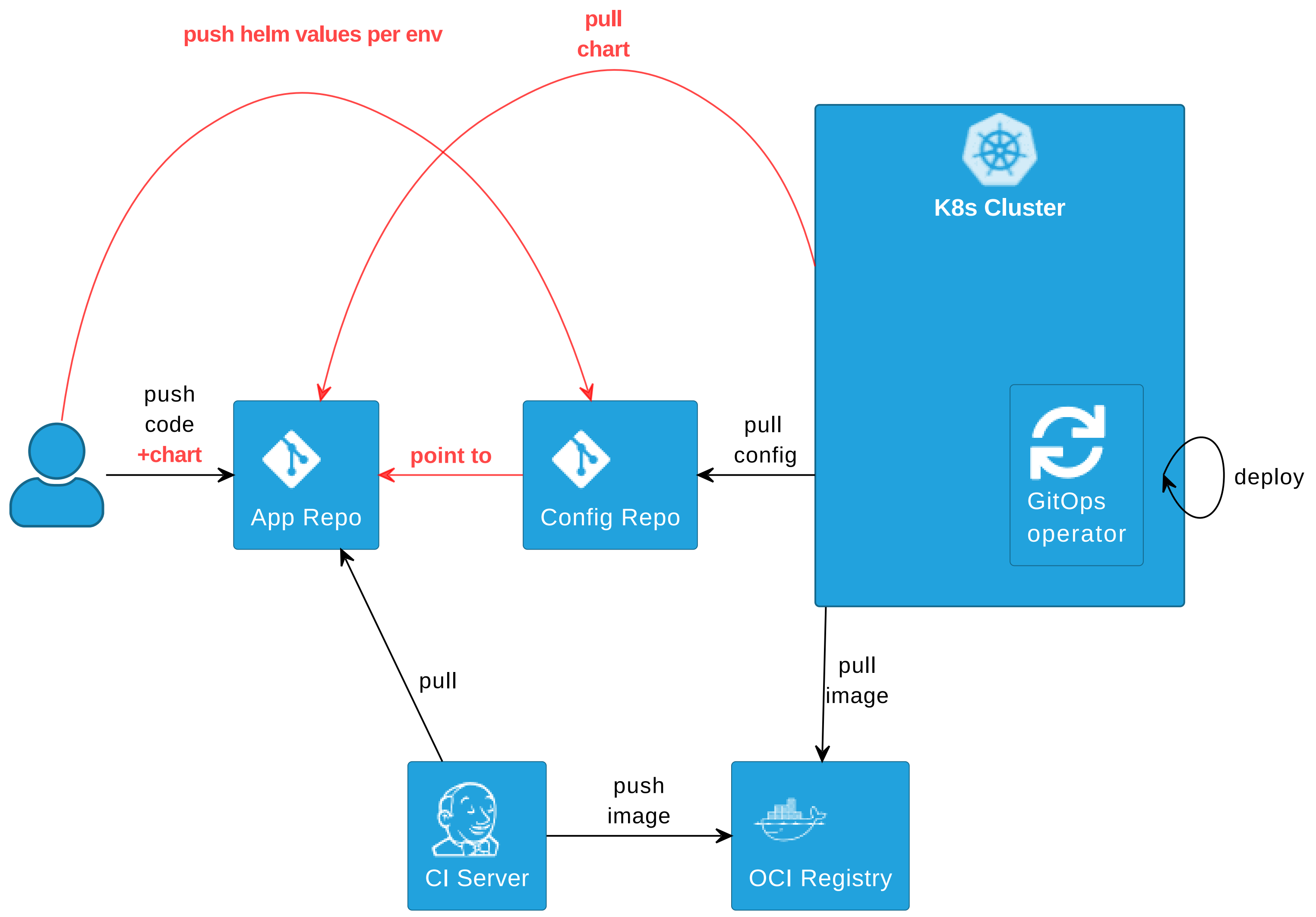
- Complexity in CI pipelines
  ➡️ Recommendation: Use a plugin or library, e.g.
  cloudogu/gitops-build-lib
- Redundant config (app repo + config repo)

Avoid Redundancy: Repo pointer



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

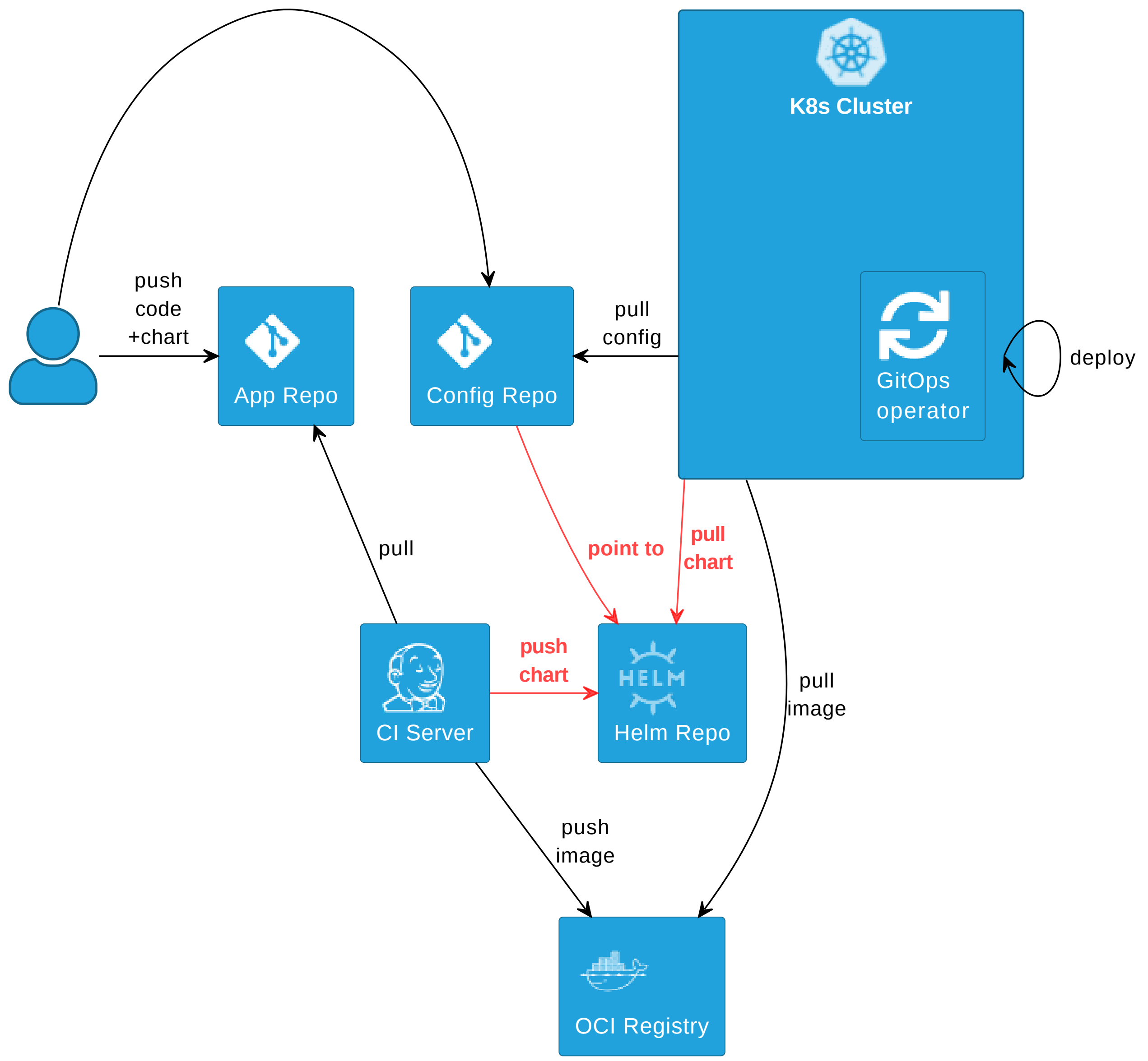
Middle ground: Config Split



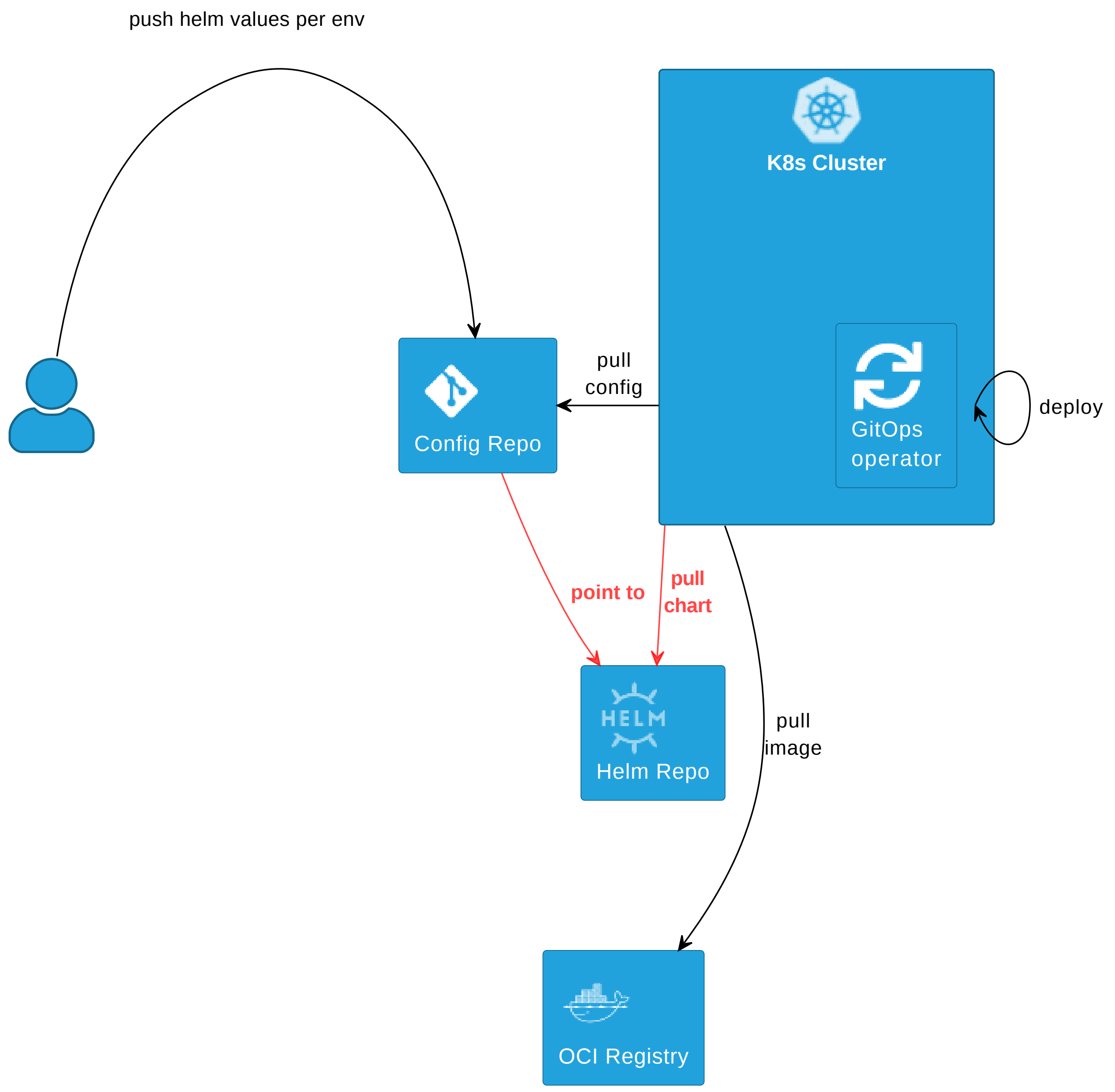
👉 HELM example

💡 Also works with K

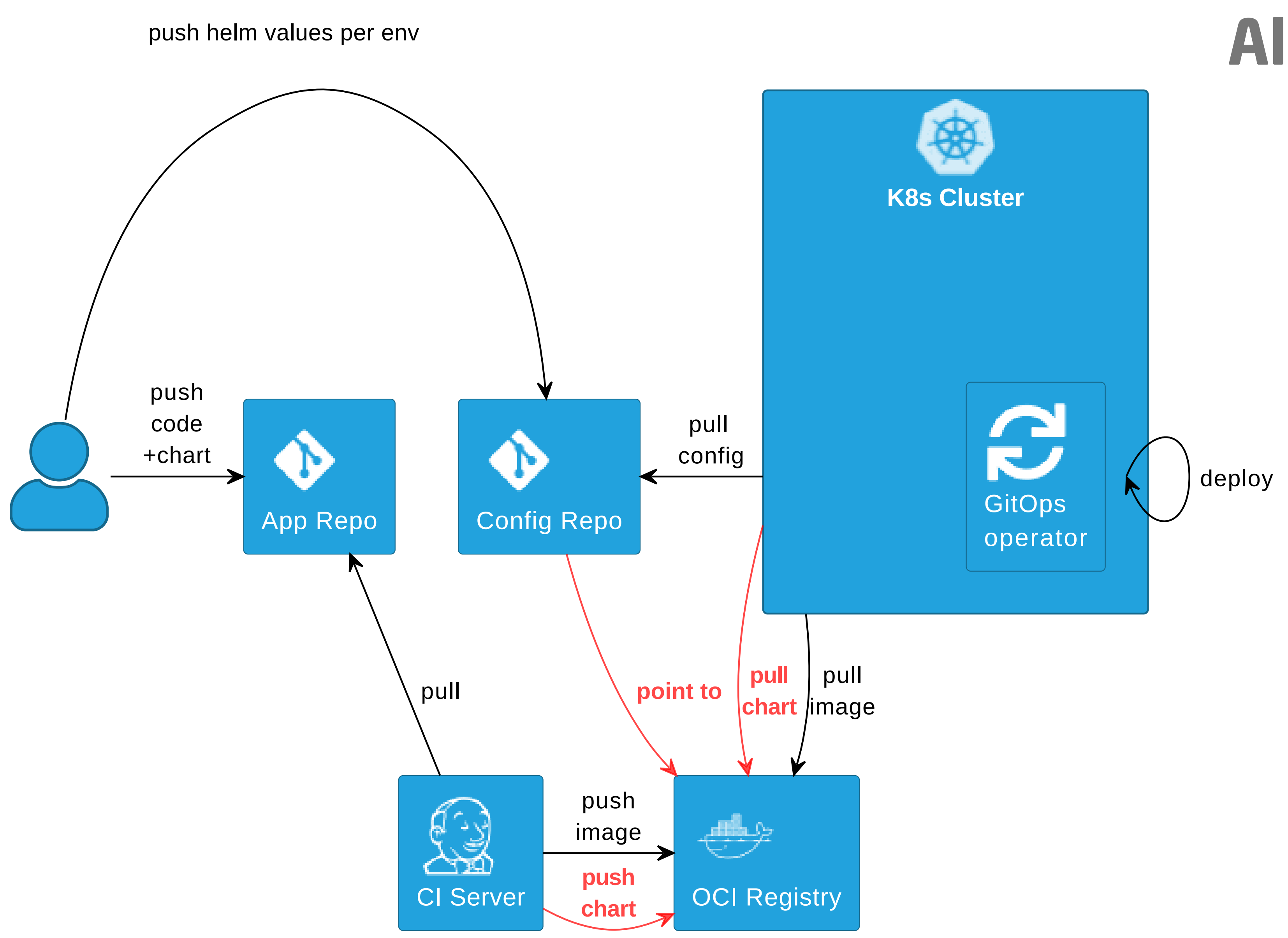
Alternative: Helm repo



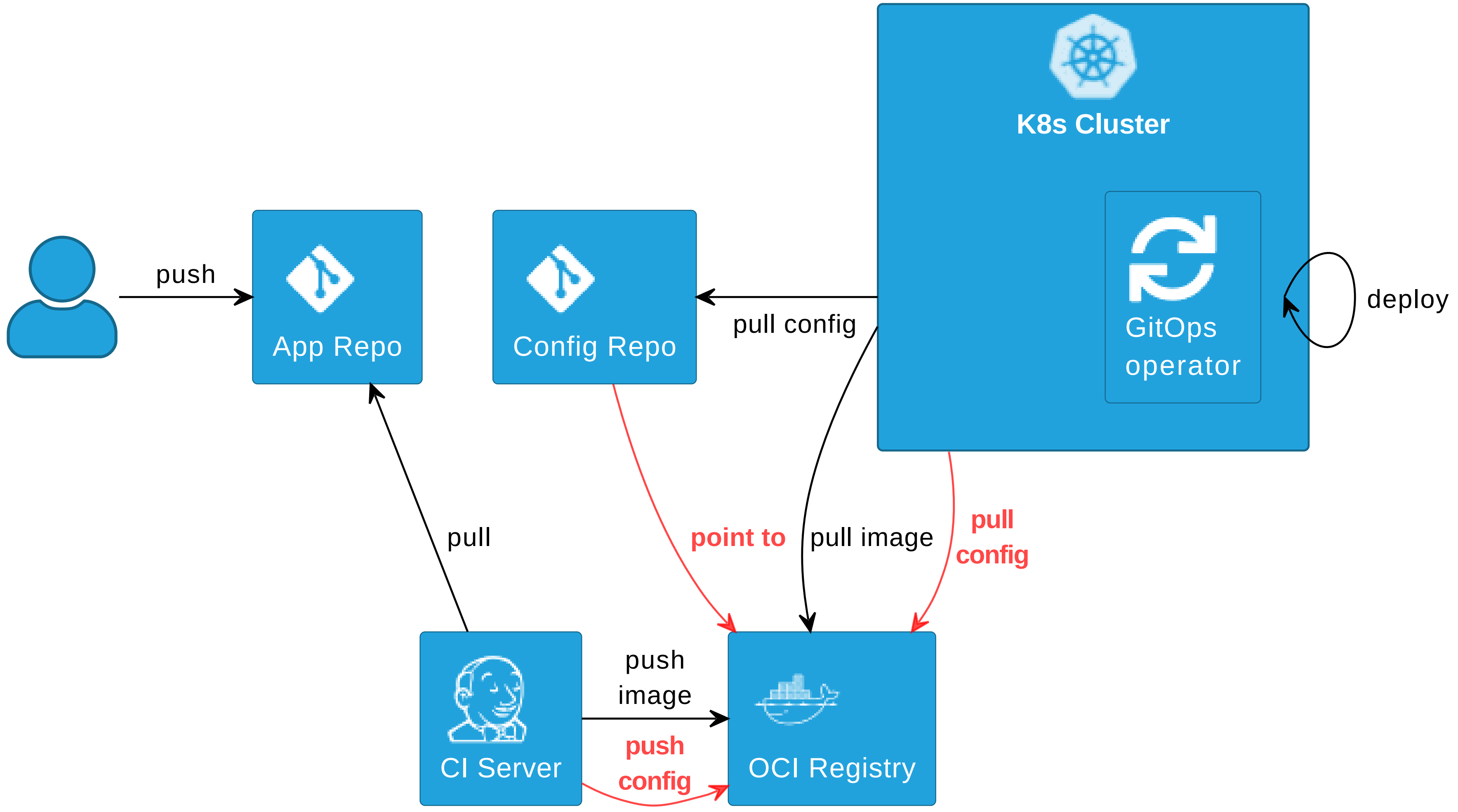
💡 Same pattern for 3rd-party apps



Alternative 2: Helm in OCI



Alternative 3: OCI artifacts



CUE

fluxcd.io/flux/cheatsheets/oci-artifacts

Promotion patterns

How to model environments AKA stages?

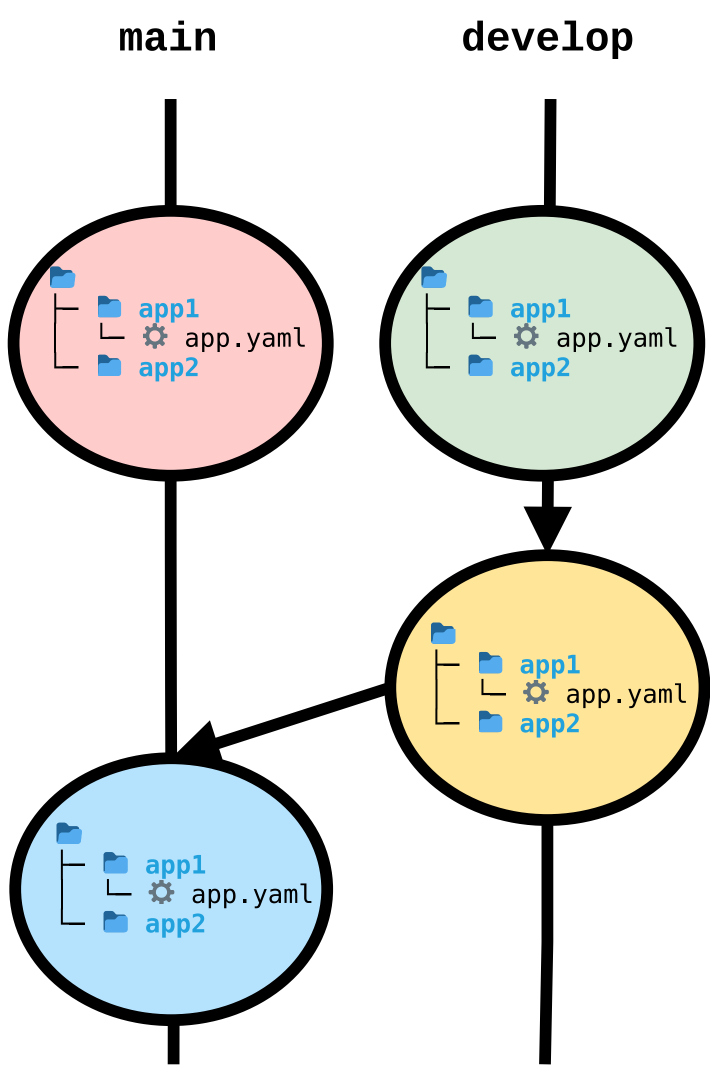
- **Branch per environment**
- **Folder/Directory per environment**
- **Repo per environment** (edge case)
- 🔥 **Preview environments**

AKA Env per (folder | branch | repo)

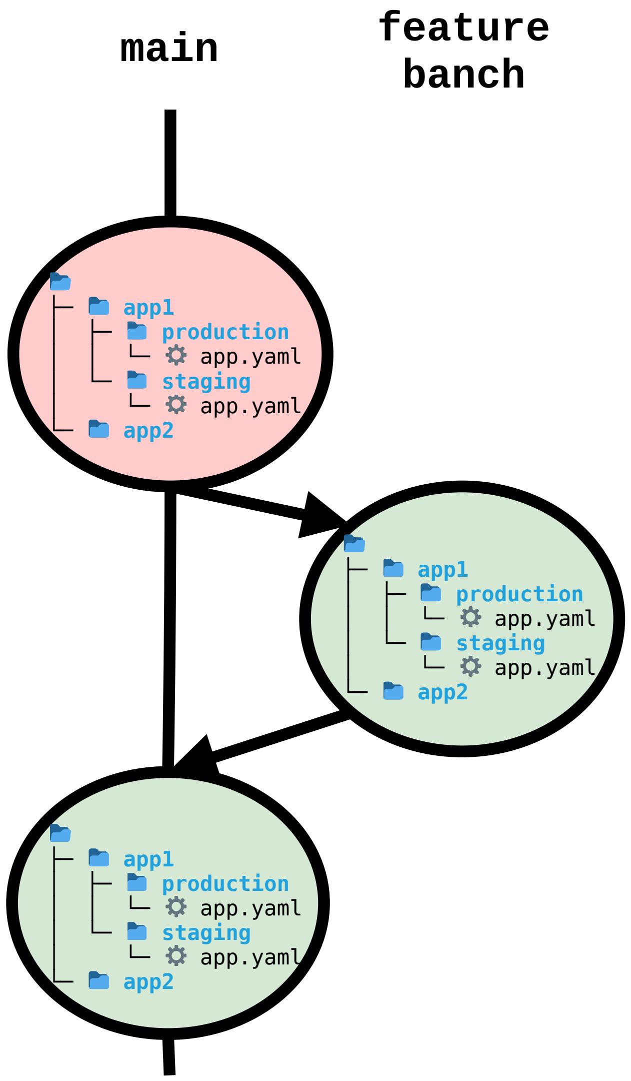
Branch vs folder per Environment

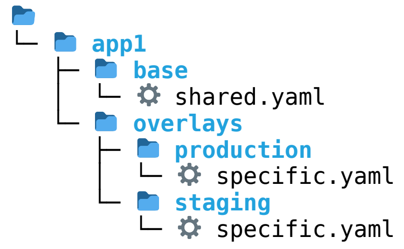
	Branch per env	Folder per env
envs	permanent branches	trunk-based folders
mapping example	`develop` ➡ staging `master` ➡ production	`staging` ➡ Staging `production` ➡ Production
promotion	merge	copy (+merge short-lived branches)

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)	?	✓ 
references	1	1, 2, 3, 4, … Branches = `anti-pattern`

```
📁
└ 📁 app1
  ├ 📁 base
  │ └ ⚙ shared.yaml
  └ 📁 overlays
    ├ 📁 production
    │ └ ⚙ specific.yaml
    └ 📁 staging
      └ ⚙ specific.yaml
```

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` ≈

Challenges with preview envs

- Resource consumption ➡️ cluster autoscaler
- Dependent systems
- Test data
- Dynamic namespaces: Authorization; SealedSecrets

Implementing promotion

Configuration Management tools

Tools for separating config of envs, keeping them DRY

- Kustomize
  - plain `kustomization.yaml` ➡️ "agnostic"
  - ≠ 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

```
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

Config update

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

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

Promotion via CI Server

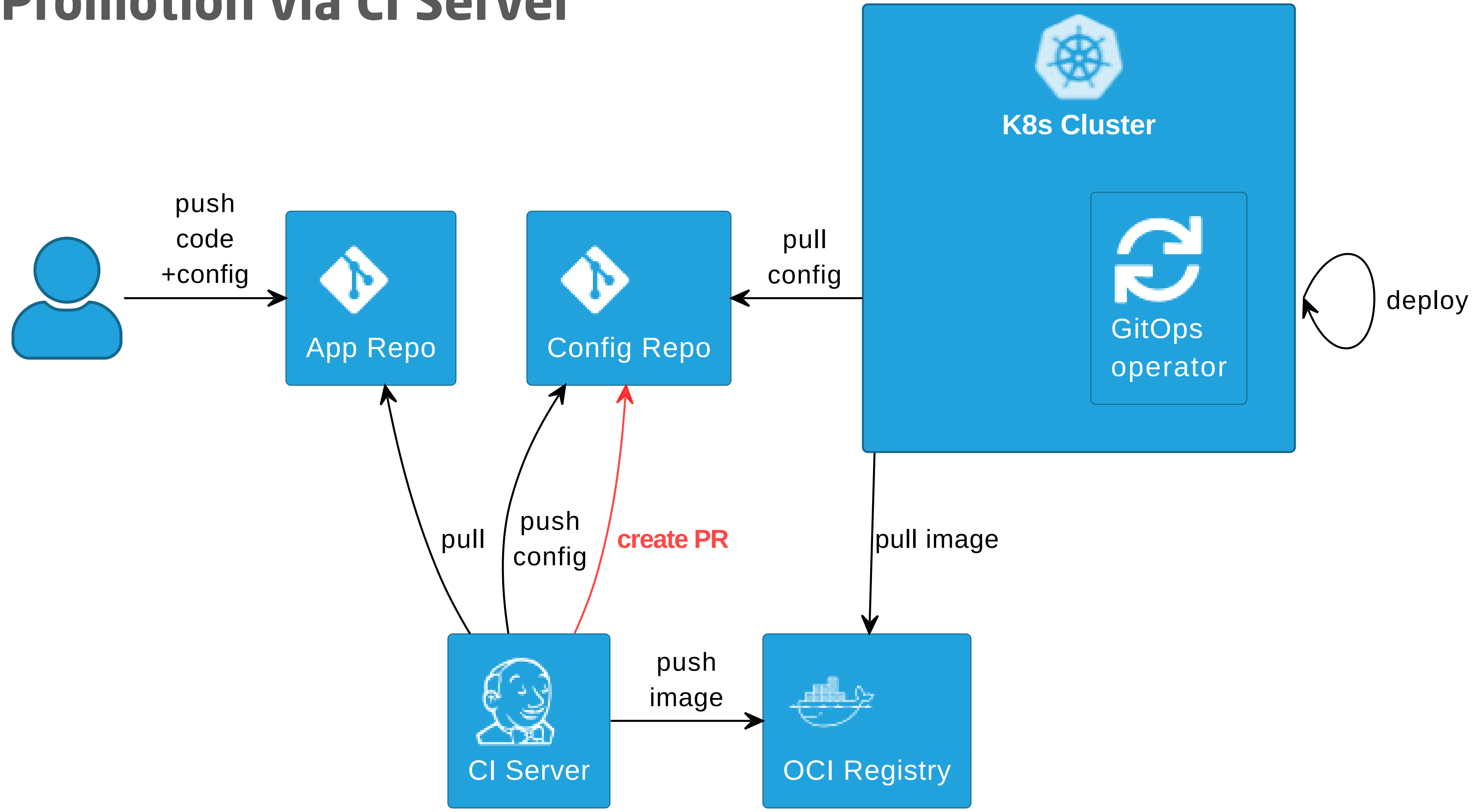
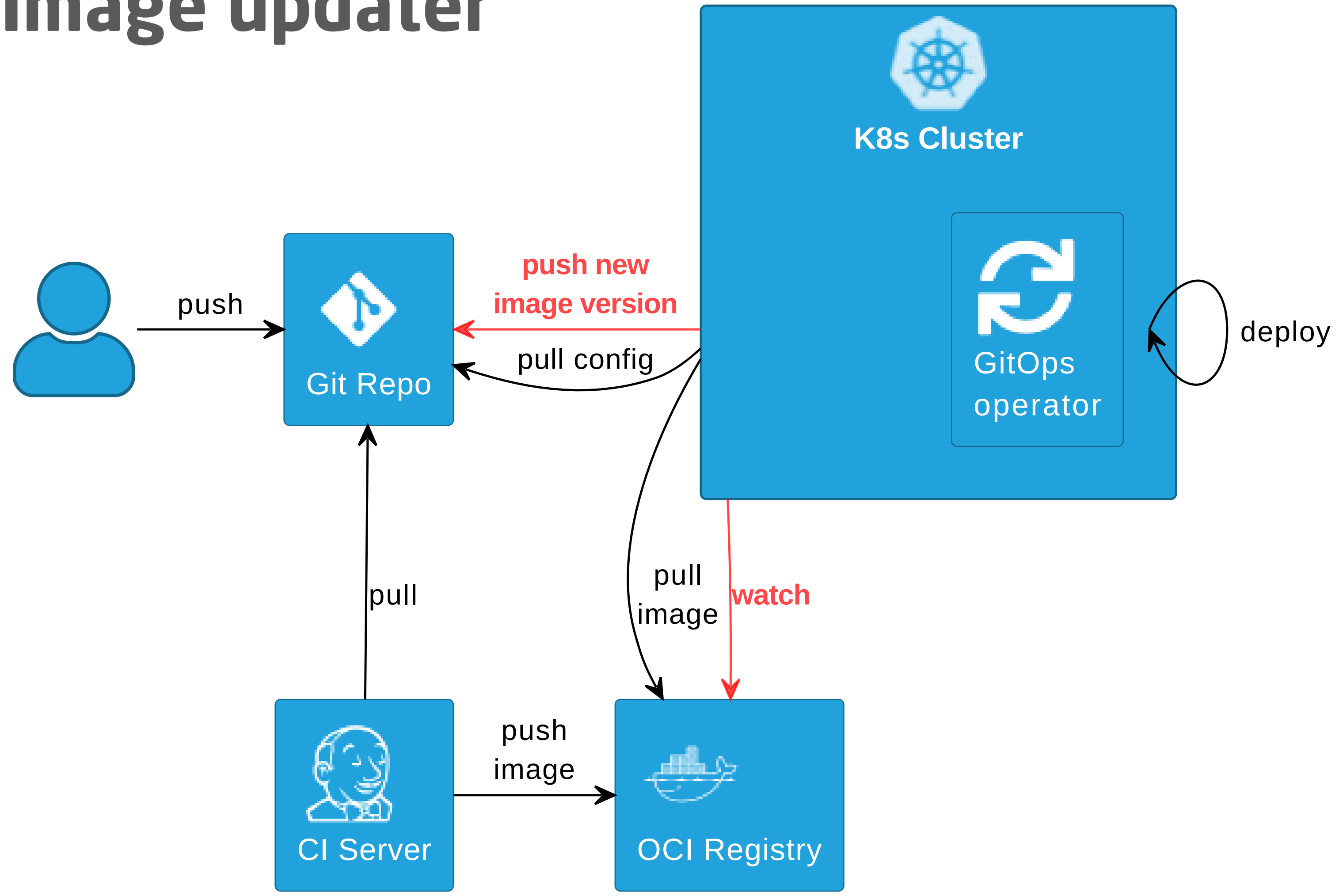


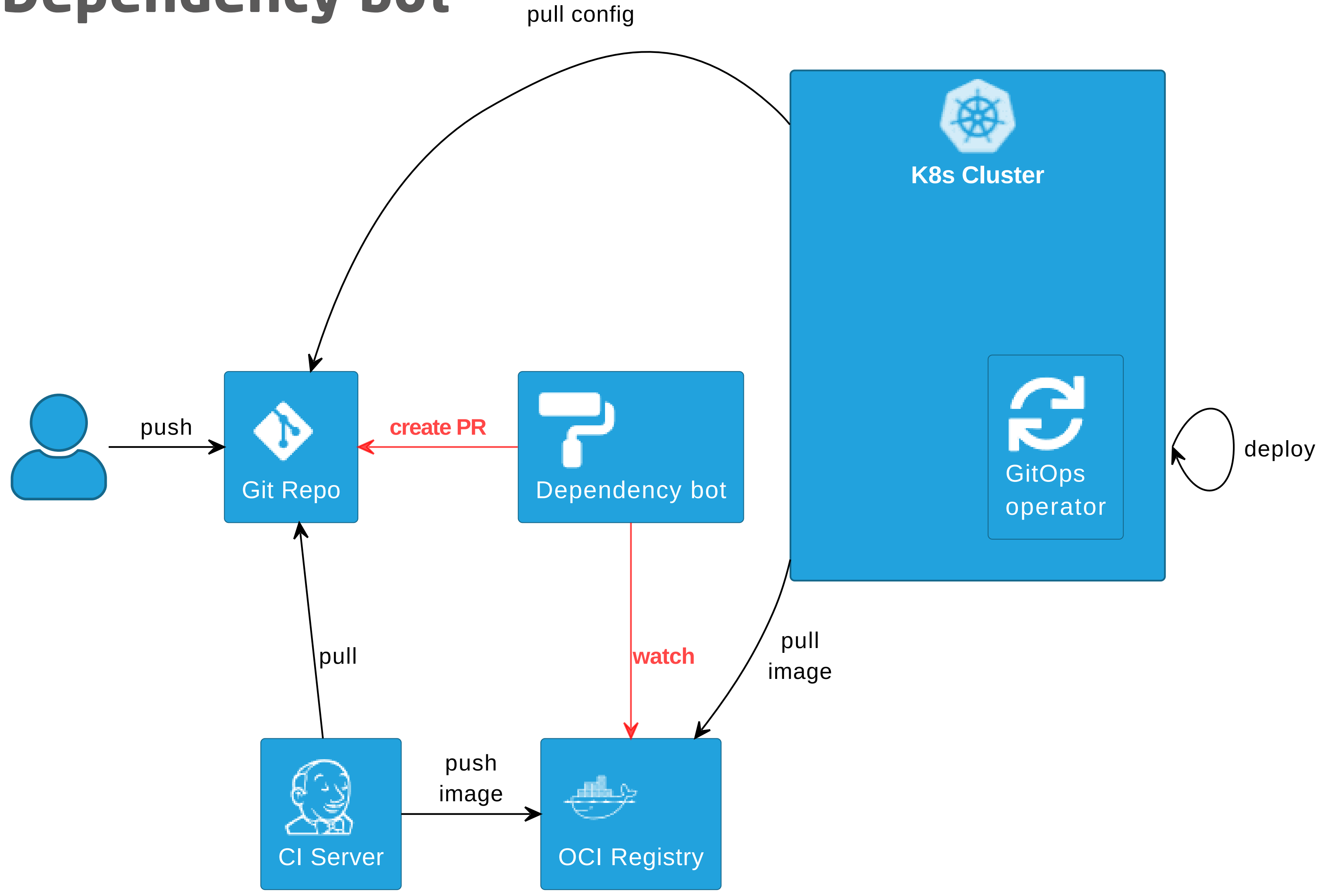
Image updater



github.com/argoproj-labs/argocd-image-updater

fluxcd.io/docs/guides/image-update

Dependency bot



e.g. github.com/renovatebot/renovate

Pull Requests

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

But: avoid cargo cult

ℹ PRs not mentioned in principles

Wiring patterns

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

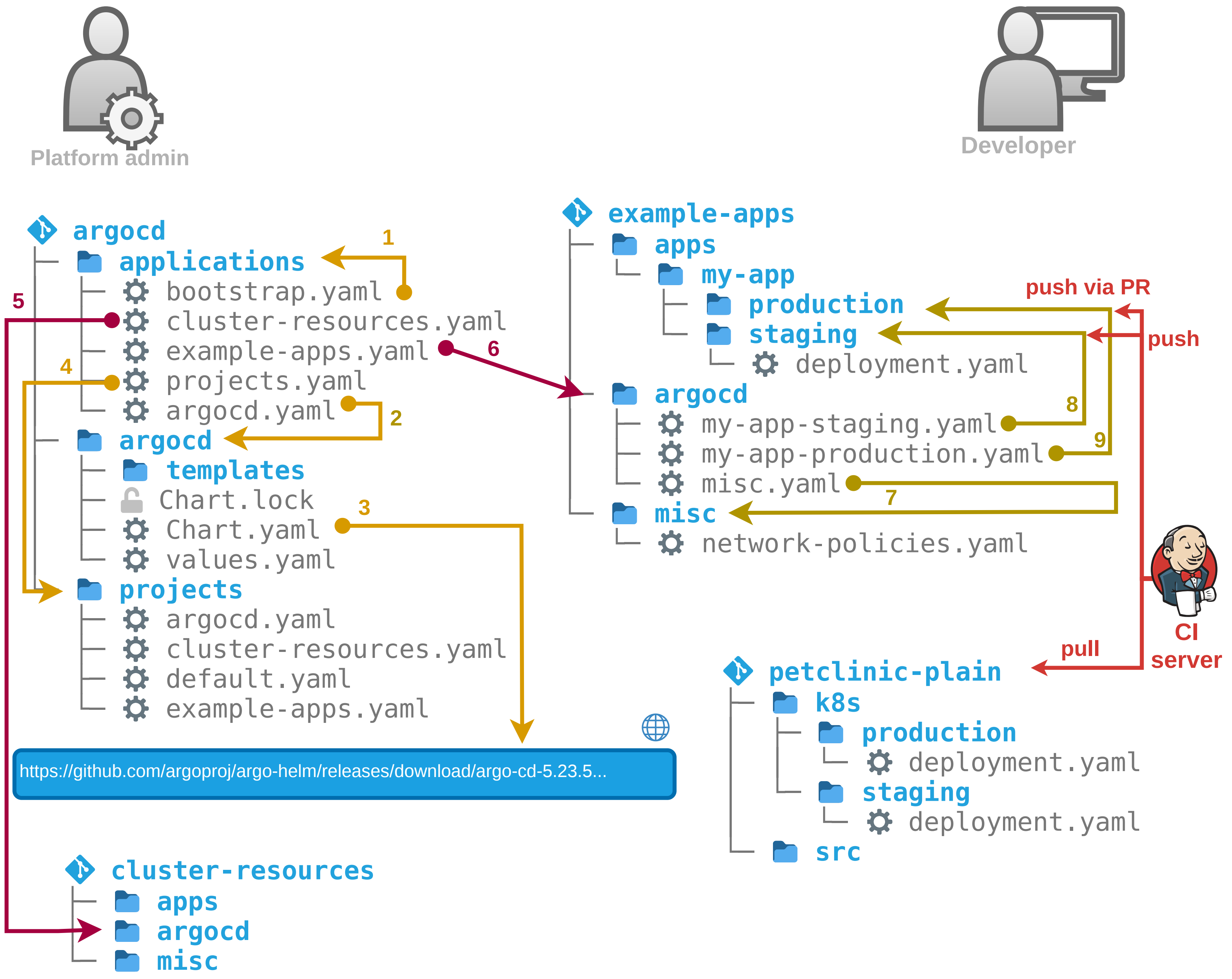
- Bootstrapping: `kubectl`, operator-specific CLI
- Linking/Grouping:
  - Operator-specific CRDs
    - `Kustomization`
    - `Application`
  - Nesting: *App of Apps*
    (same principle with `Kustomization`)
  - Templating: `ApplicationSets` - folders, lists, config files

Example + Demo

GitOps playground

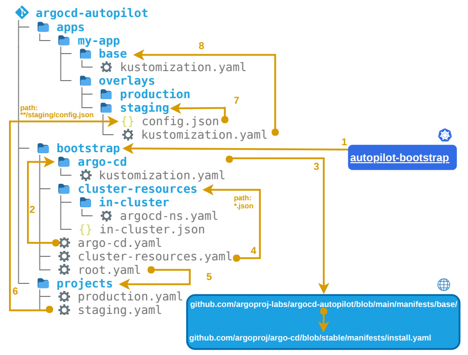
- **Repo pattern:** Per team per app
- **Operator pattern:** Standalone (Hub and Spoke)
- **Operator:**
- **Boostrapping:** `Helm`, `kubectl`
- **Linking:** `Application`
- **Features:**
  - Operate ArgoCD with GitOps
  - Solution for cluster resources
  - Config update + replication via CI
  - Mixed repo patterns
  - Env per app pattern
- **Source:** cloudogu/gitops-playground

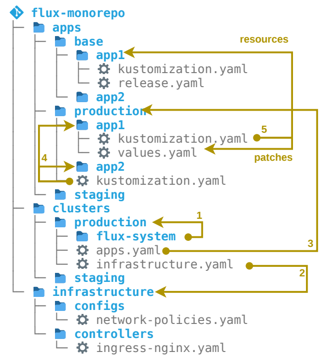
```
COMMIT='8e21bd4'
bash <(curl -s \
  "https://raw.githubusercontent.com/cloudogu/gitops-playground/$COMMIT/scripts/init-cluster.sh)" \
  --bind-ingress-port=80  \
  && sleep 2 && docker run --rm -it --pull=always -u $(id -u) \
    -v ~/.config/k3d/kubeconfig-gitops-playground.yaml:/home/.kube/config \
    --net=host \
    ghcr.io/cloudogu/gitops-playground:$COMMIT --yes --argocd --base-url=http://local.gd -x
```

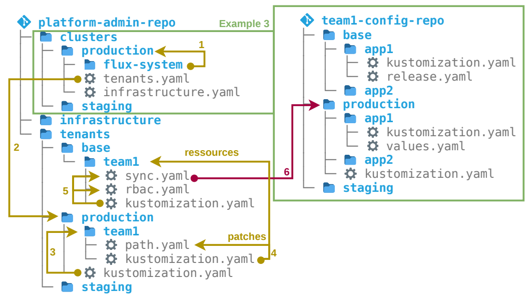

Platform admin
Developer
argocd
applications
bootstrap.yaml
cluster-resources.yaml
example-apps.yaml
projects.yaml
argocd.yaml
argocd
templates
Chart.lock
Chart.yaml
values.yaml
projects
argocd.yaml
cluster-resources.yaml
default.yaml
example-apps.yaml
1
2
3
4
5
6
https://github.com/argoproj/argo-helm/releases/download/argo-cd-5.23.5...
cluster-resources
apps
argocd
misc
example-apps
apps
my-app
production
staging
deployment.yaml
argocd
my-app-staging.yaml
my-app-production.yaml
misc.yaml
misc
network-policies.yaml
7
8
9
push via PR
push
pull
CI server
petclinic-plain
k8s
production
deployment.yaml
staging
deployment.yaml
src

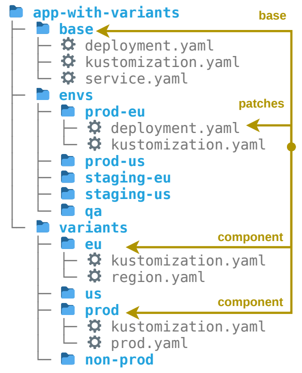
More examples + further reading

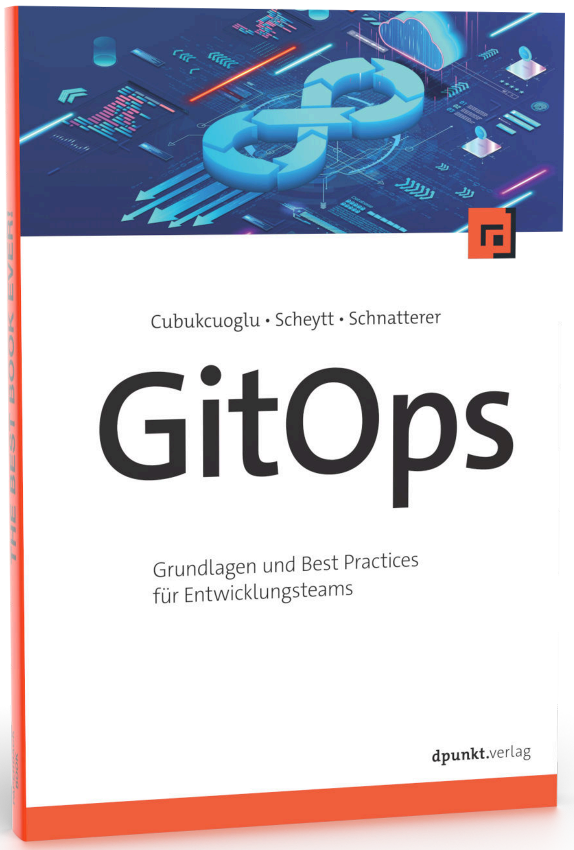
cloudogu/gitops-patterns










Cubukcuoglu • Scheytt • Schnatterer
GitOps
Grundlagen und Best Practices für Entwicklungsteams
dpunkt.verlag

How to get started?

- Chronology:
  - Step 1: Chose an operator
  - Step 2: Design process/repos/promotion
  - Step 3: Wire everything
- Keep in mind:
  - **Conway's law**: no standard, find the structure for *your* org
  - **Responsibility**: platform/infra teams, cluster admins ↔️ app teams
  - **Use case**: deploying apps vs infra

➡️ Use **Patterns/examples** as inspiration

Johannes Schnatterer, Cloudogu GmbH

Slides

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

@schnatterer@floss.social in/jschnatterer @jschnatterer

Wir entwickeln einen open source GitOps-Stack für K8s

Sag uns wie wir GitOps für dich leichter machen können

Survey

Image sources

- implementation https://unsplash.com/photos/selective-focus-photography-blue-and-black-makita-power-drill-KlbyOnxseY8
- Demo https://unsplash.com/photos/assorted-color-hot-air-balloons-during-daytime-DuBNA1QMpPA
- coloured-parchment-paper background by brgfx on Freepik https://www.freepik.com/free-vector/coloured-parchment-paper-designs_1078492.htm
- Question mark https://pixabay.com/illustrations/question-mark-question-response-1020165/