O'REILLY®
Velocity Conference

GitOps 101

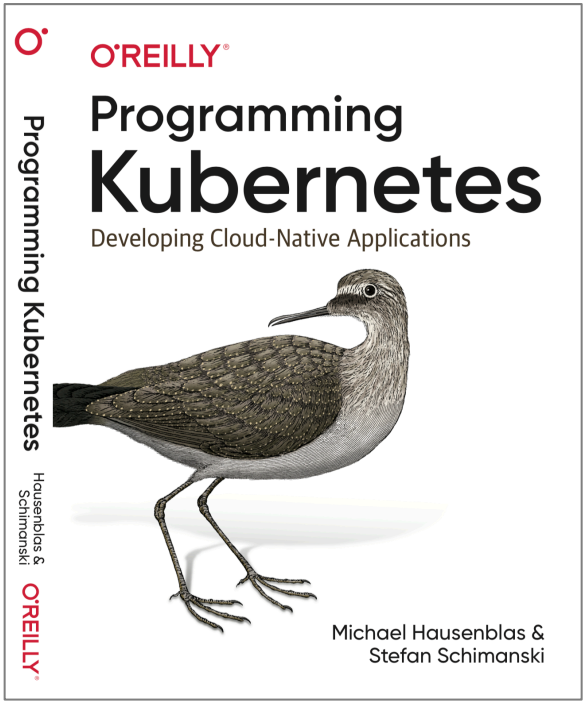
2019-11-05, Michael Hausenblas

velocityconf.com
#VelocityConf

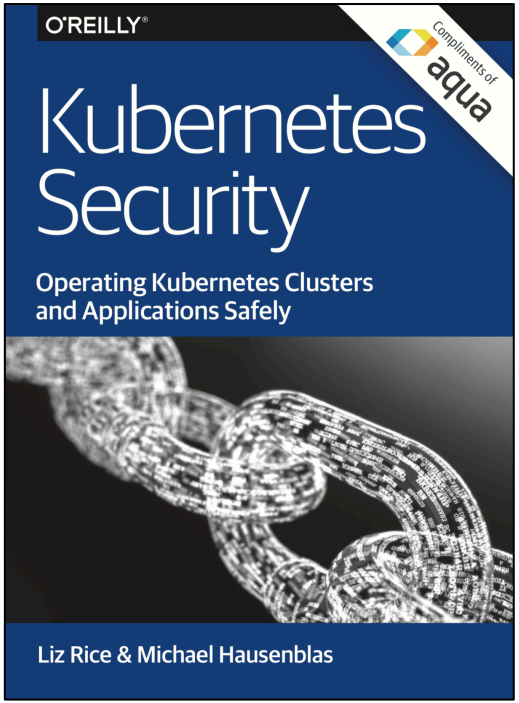
Who am I?

- Developer Advocate in the AWS container service team
- Previous roles at Red Hat, Mesosphere, MapR, applied research
- Find me via …
  - Slack: AWS dev, Kubernetes, CNCF, Weaveworks
  - Twitter: @mhausenblas
  - Mail: hausenbl@amazon.com

Books



programming-kubernetes.info



kubernetes-security.info

Note: book signing Thu 7 Nov from 10:30 to 11:30

Who are you?

admin

developer

architect

SRE

QA

PM

Agenda

- Kubernetes and Git 101
- GitOps motivation & model
- *BREAK 10:30am to 11am*
- GitOps in action
- Progressive delivery
- Challenges

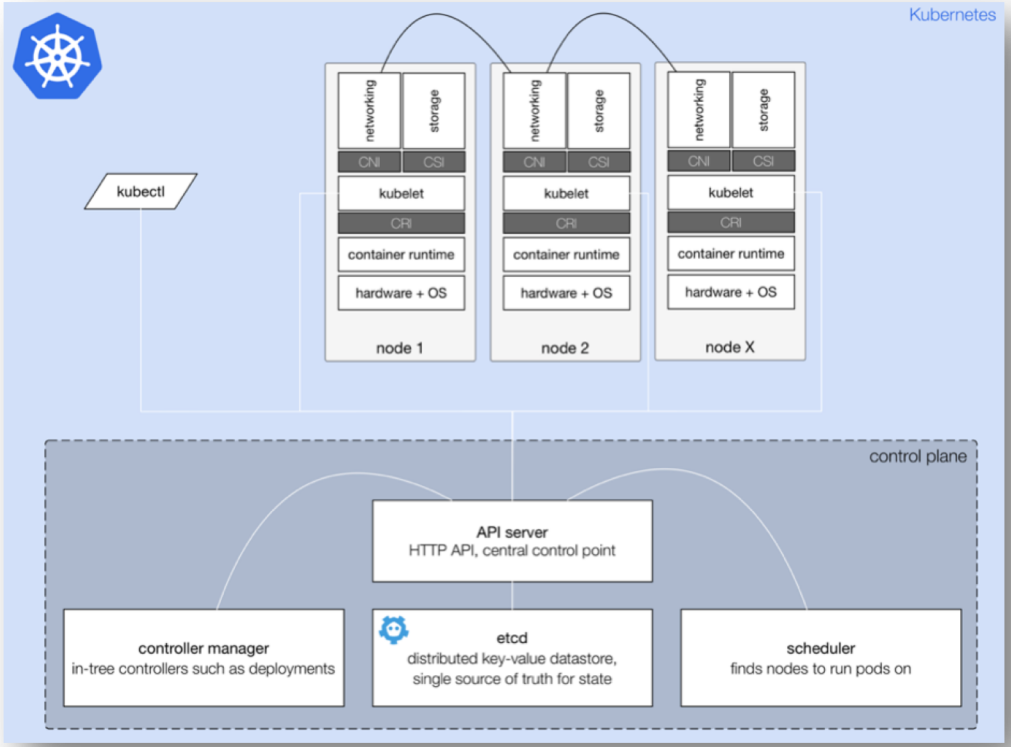
Learning goals

- Being able to explain what GitOps is
- Know about the benefit of GitOps
- Having some hands-on experience with an example GitOps set up
- Being able to decide if GitOps is the right choice for team/project

Kubernetes 101

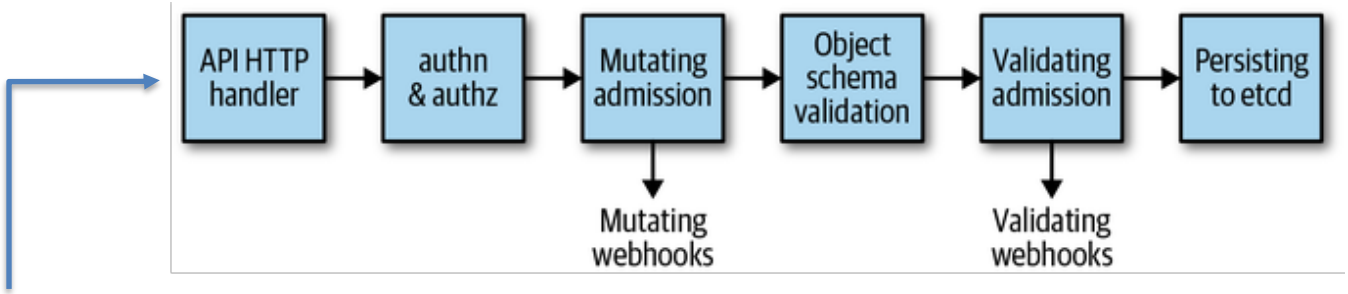
Architecture

Overview



Architecture

API Server




```
kubectl get deploys
```

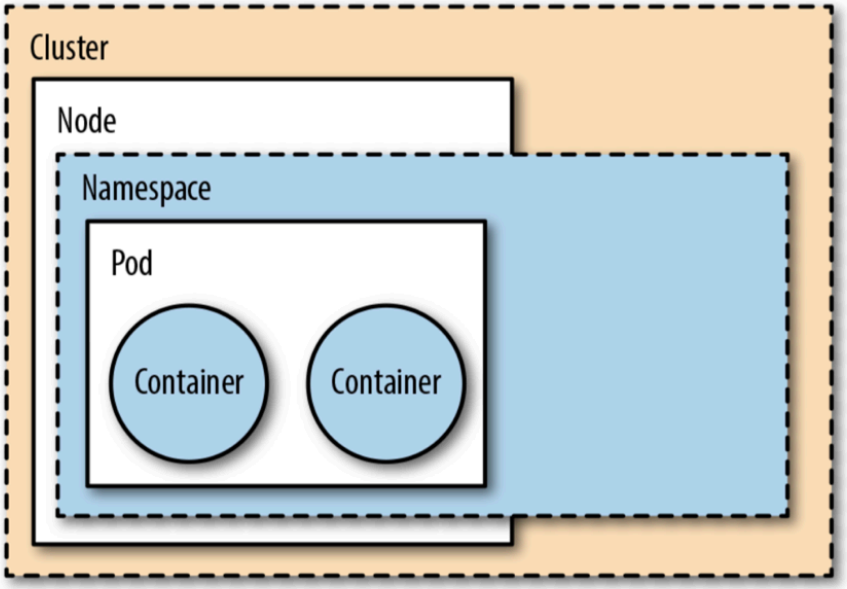
Kubernetes resources

- Core and custom resources
  - spec: kubectl/HTTP API (via params and/or YAML/JSON)
  - status: kubectl/HTTP API (YAML/JSON)
- Namespaces
- Labels and selectors, annotations
- Pods
- Deployments
- Services

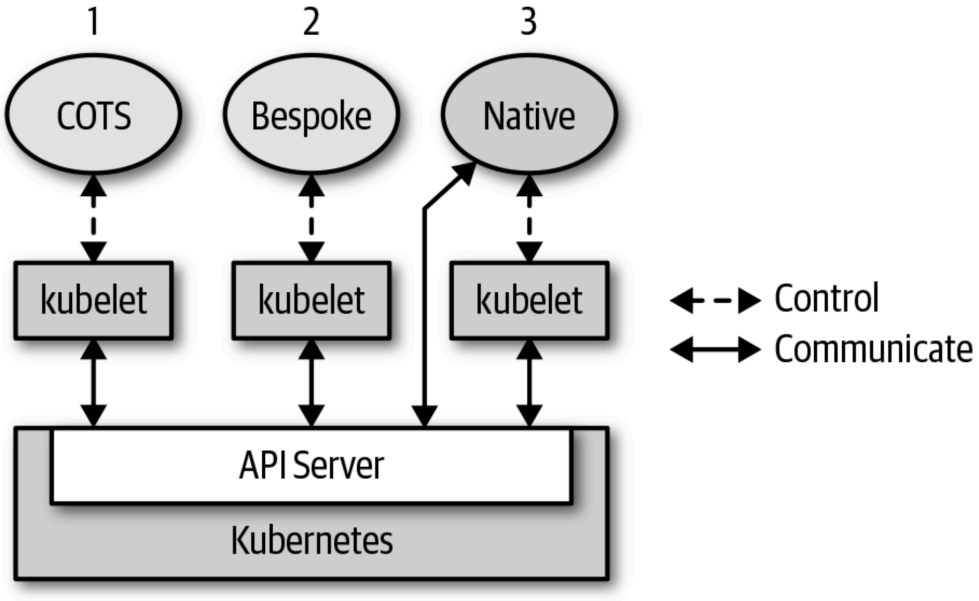
Kubernetes resources

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: podinfo
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: podinfo
    spec:
      containers:
      - name: podinfo
        image: quay.io/stefanprodan/podinfo:3.0.0
        ports:
        - containerPort: 3000
```

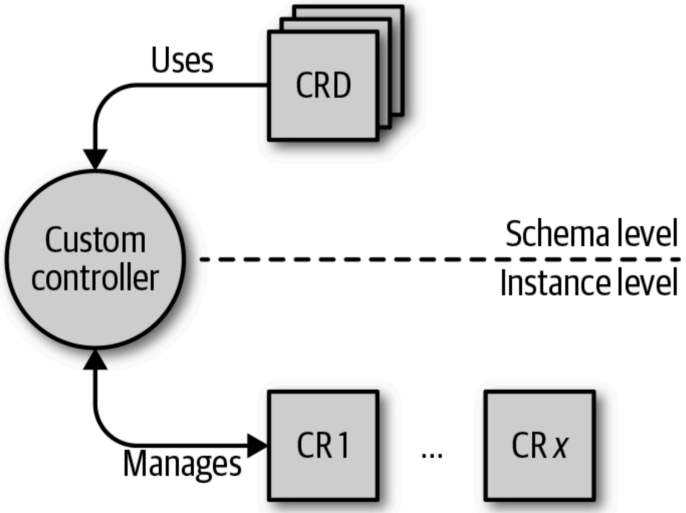
Boundaries



Running on Kubernetes?



Operators

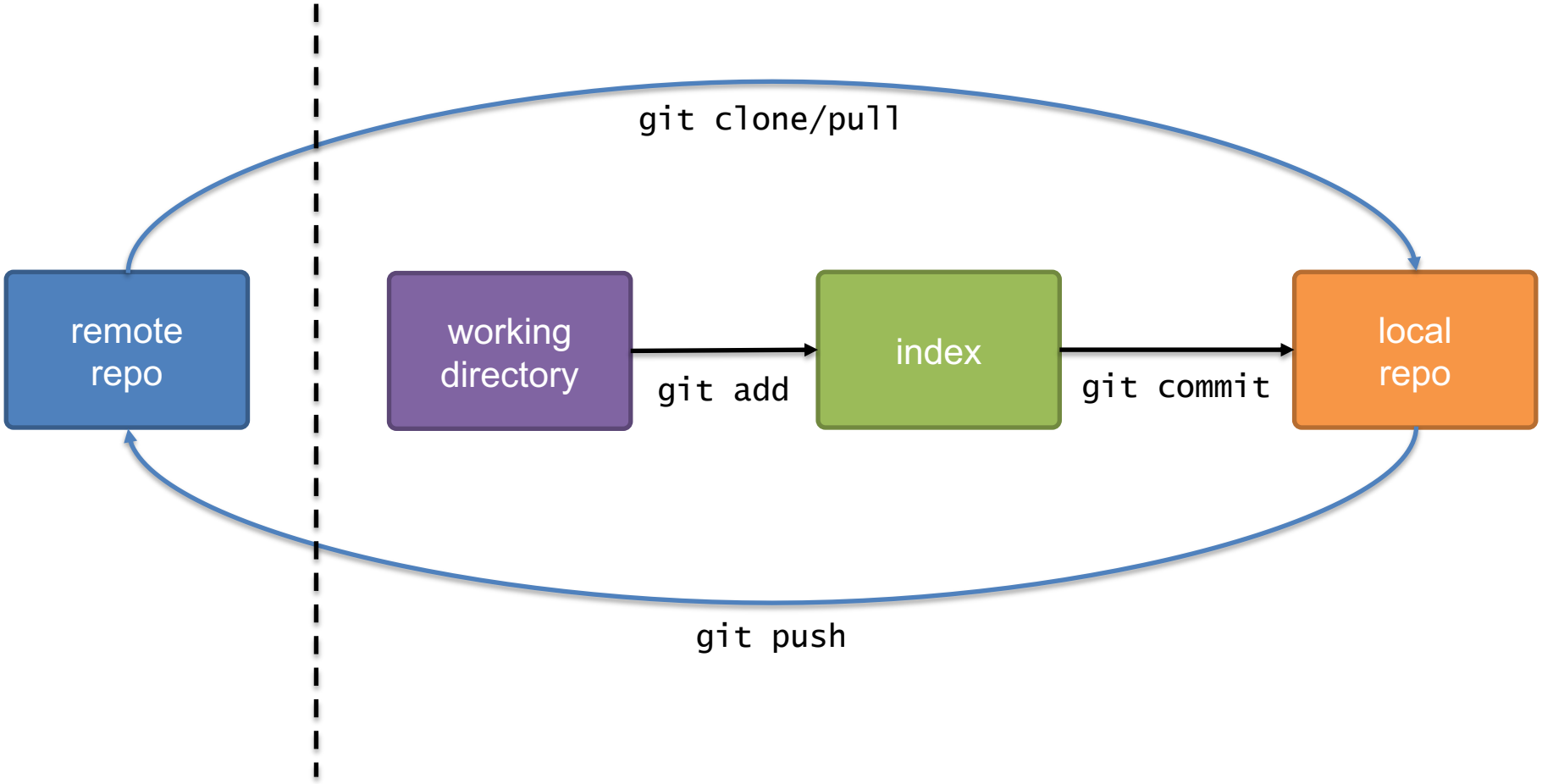


operator = *custom resource + custom controller*

A short deployment history

- Shell scripts, Make files
- Ansible, Chef, Puppet, etc.
- Helm, ksonnet, kustomize, etc.
- GitOps

Git 101


git clone/pull
remote repo
working directory
git add
index
git commit
local repo
git push

What is GitOps?

What is GitOps?

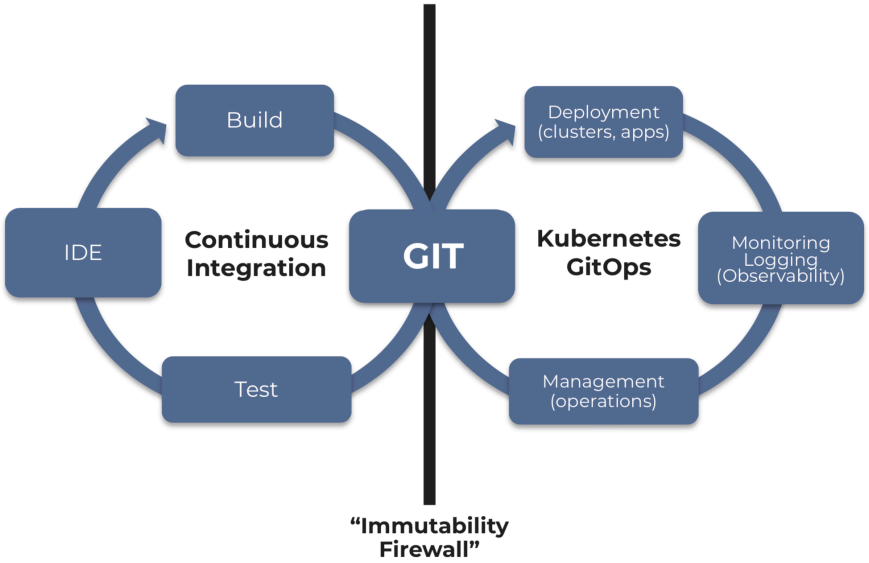
GitOps is a standardized workflow for how to deploy, configure, monitor, update and manage infrastructure-as-code

Core idea is having a Git repository that contains declarative descriptions of the infrastructure desired in the production environment and an automated process to make production environment match the described state in the repository

What is GitOps?

- An operation model
- Derived from operation knowledge
- Technology agnostic
- A set of principles
- A way to speed up your team

GitOps: an operation model



Providing …

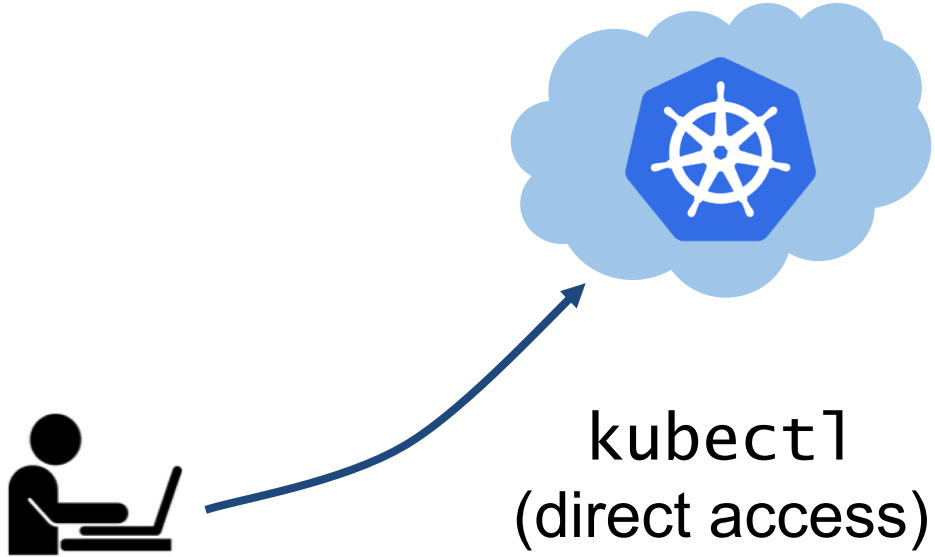
- A single source of truth for the desired system’s state
- Separation of concerns between development and deployment process
- Transparency and auditability
- Risk reduction (rollbacks)

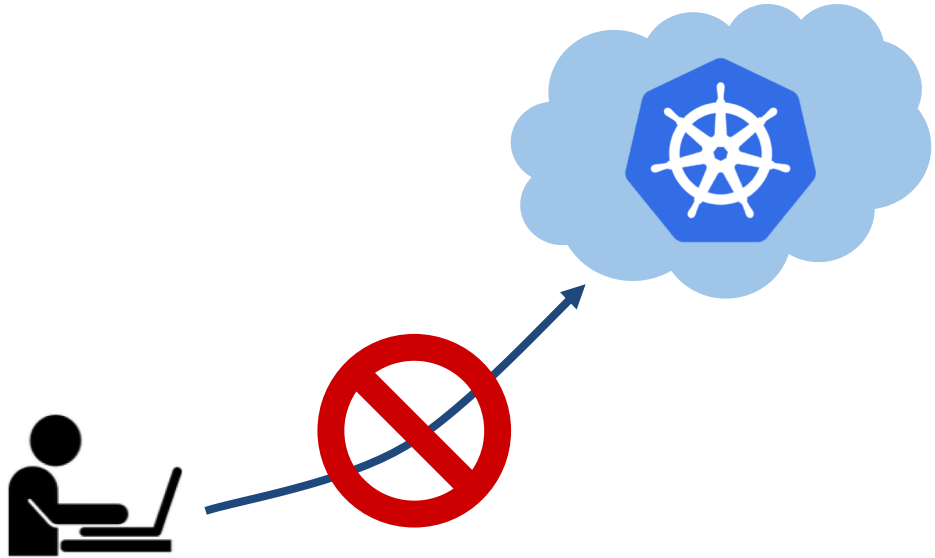
Why should we care?

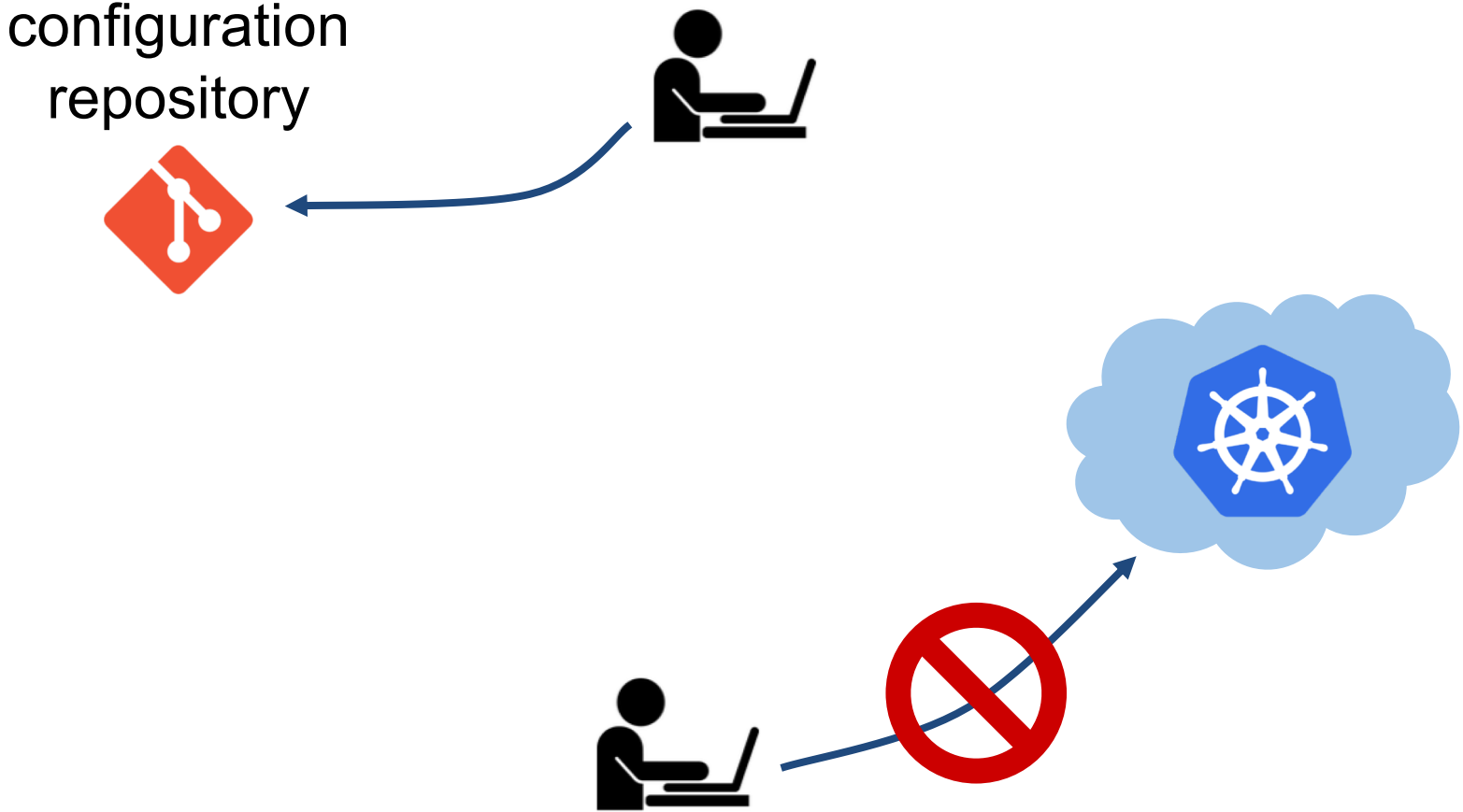
- Auditing and attribution
- Separation of concerns
- No crossing security boundary
- Process & constraints enforcement
- Great software ↔ human collaboration point
- Easy to validate for correctness (policies)
- System can self heal

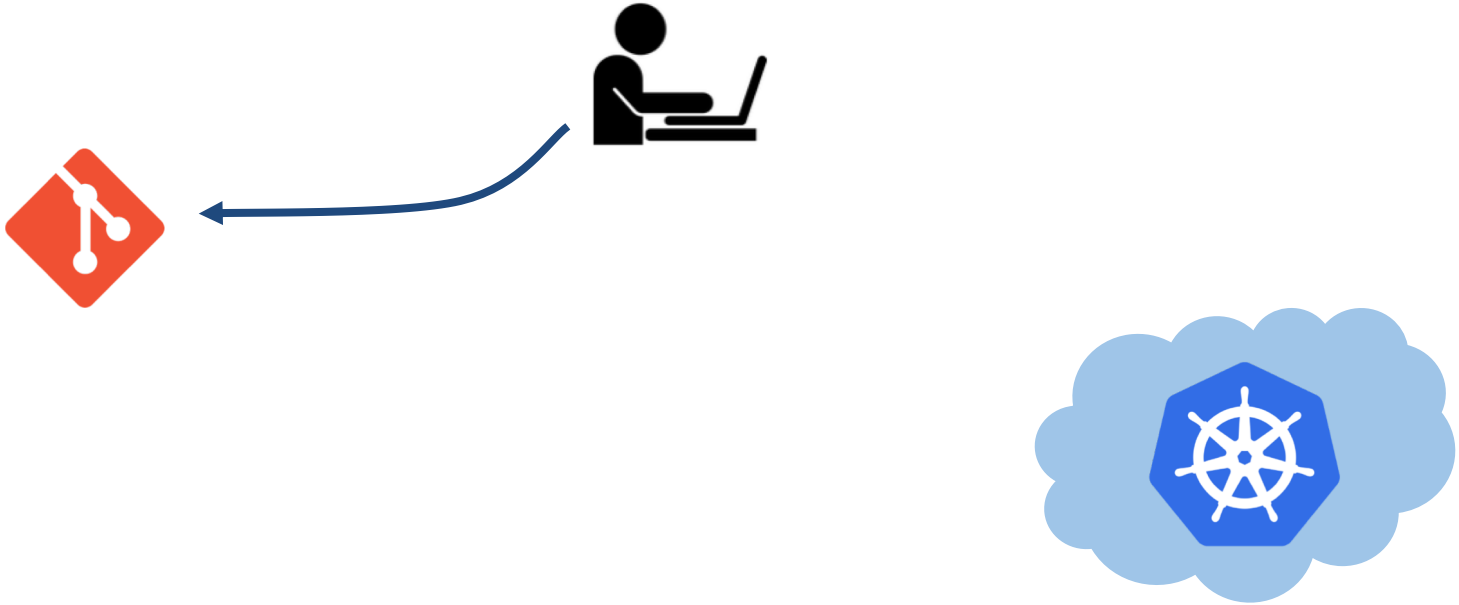
The GitOps Model

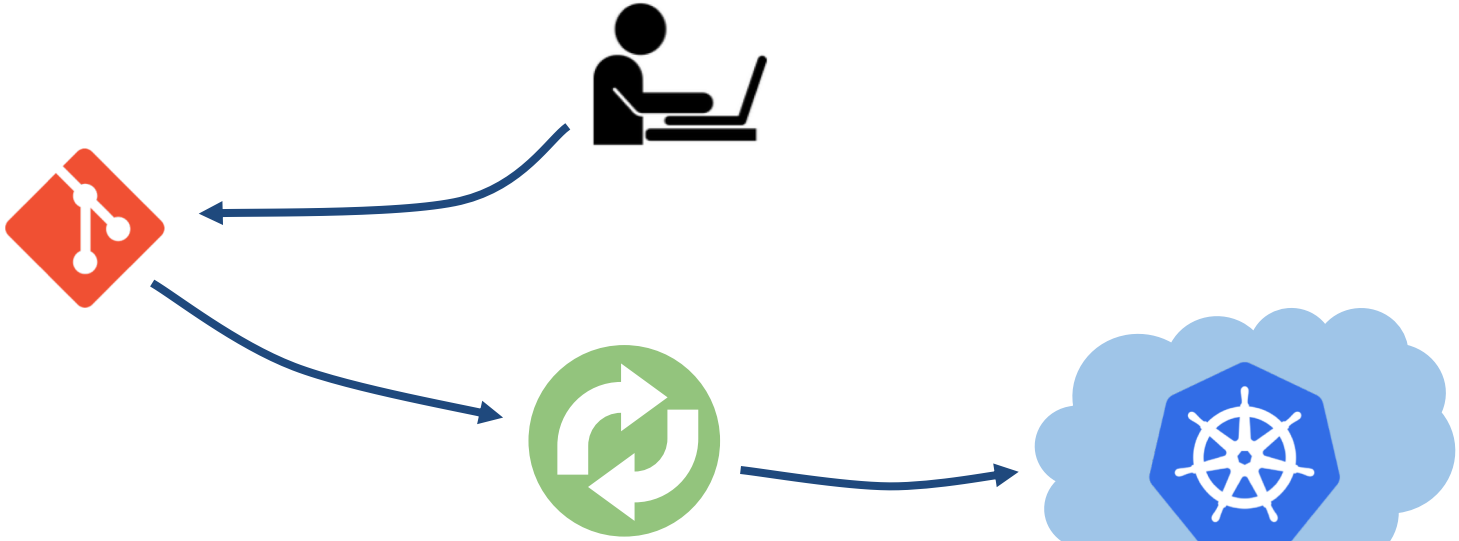
Kubernetes cluster


kubectl
(direct access)




configuration
repository




deployment
agent

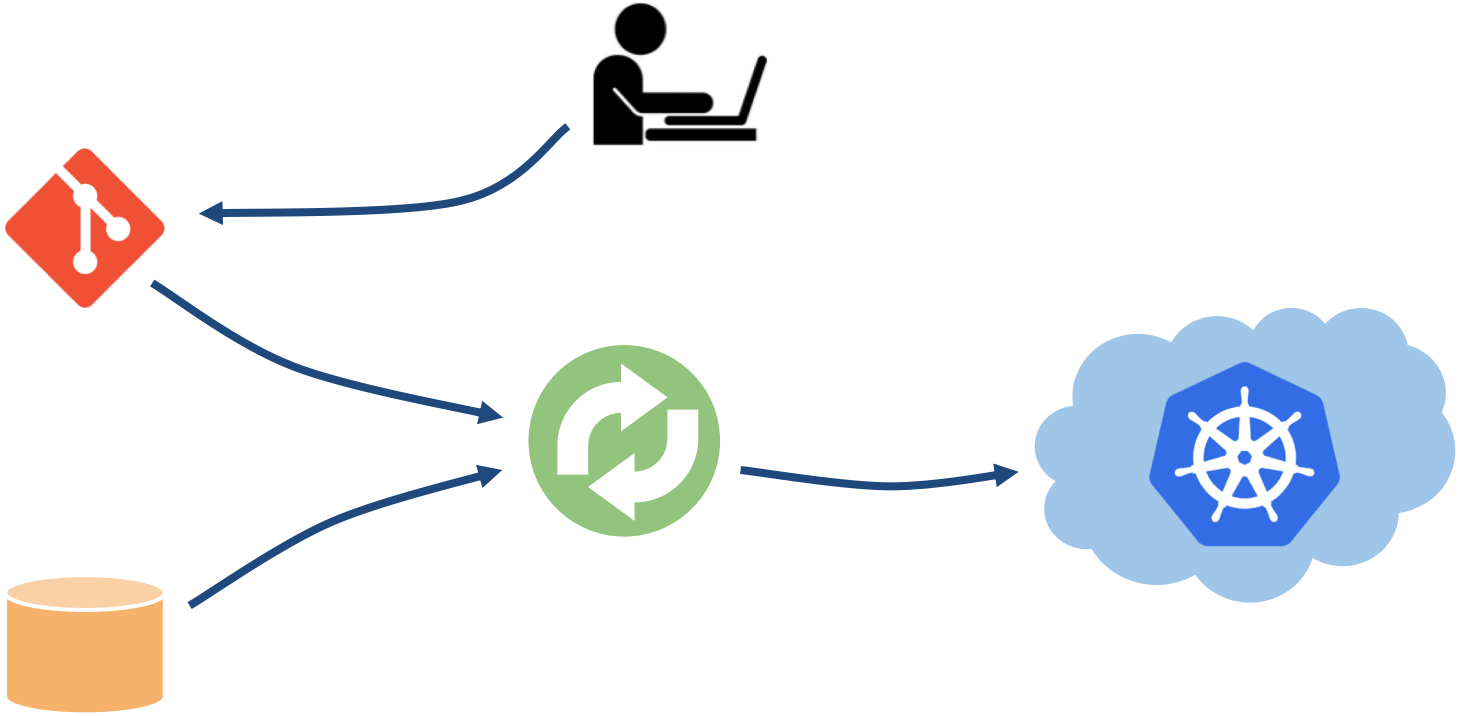
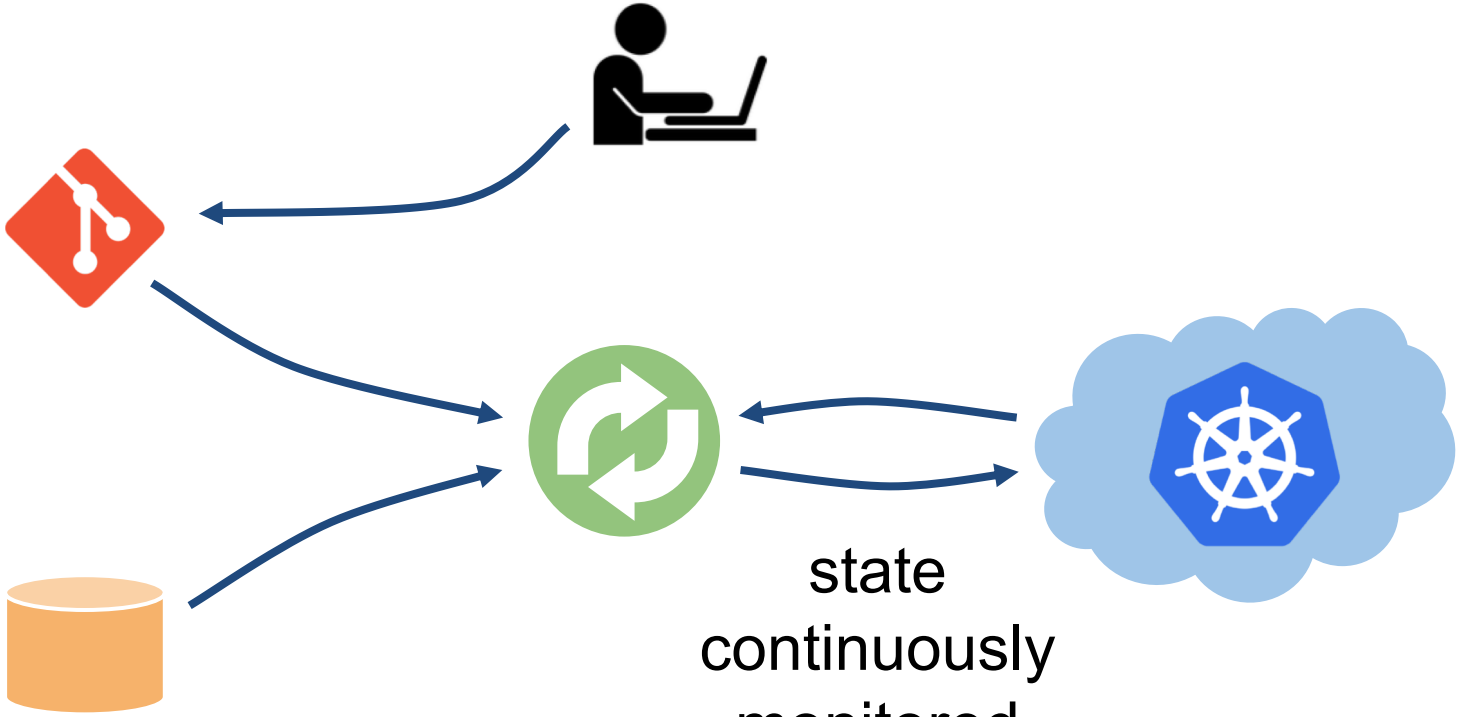
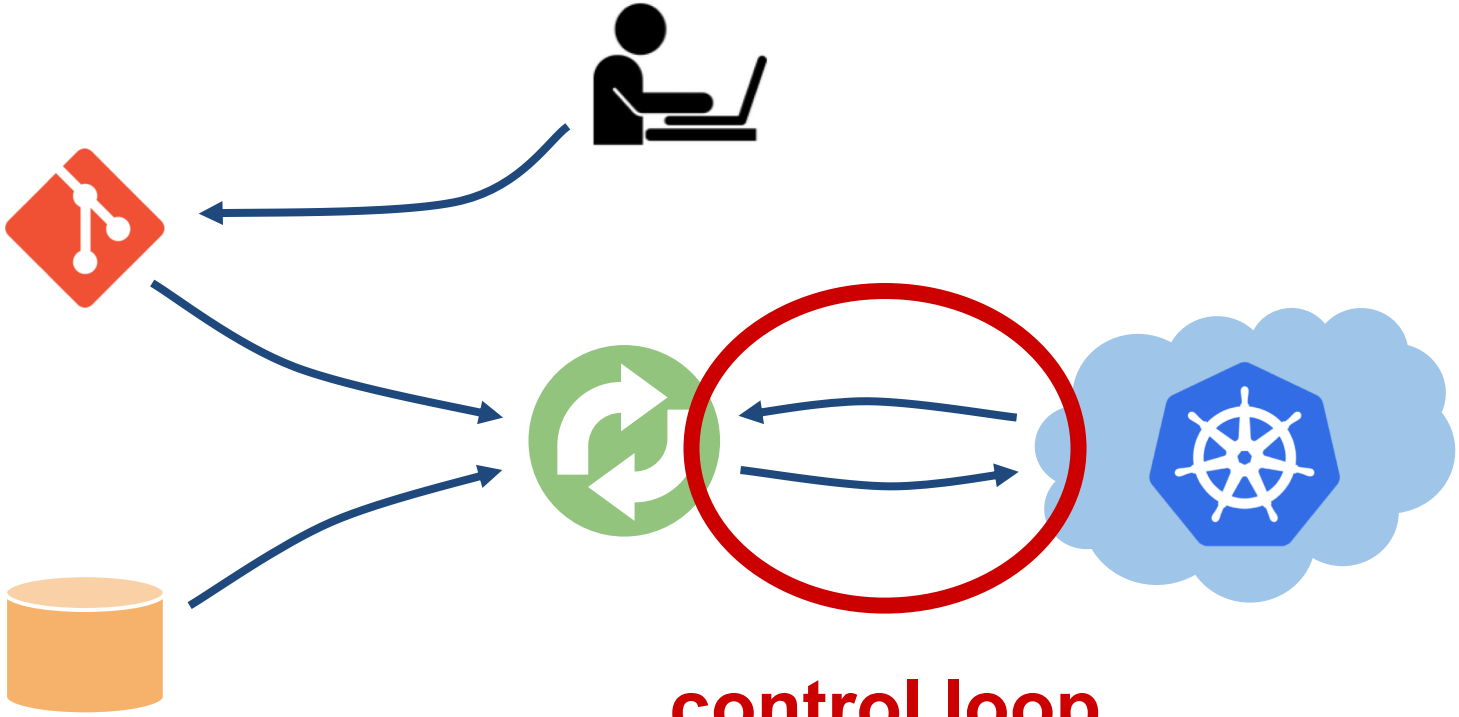


image
repository


state
continuously
monitored



control loop

GitOps Principles

1. The entire system is described **declaratively**
2. The canonical desired system state is **versioned** (Git)
3. Approved changes to the desired state are **automatically applied** to the system
4. Software agents ensure **correctness** and **alert** on divergence

1 The entire system is described **declaratively**

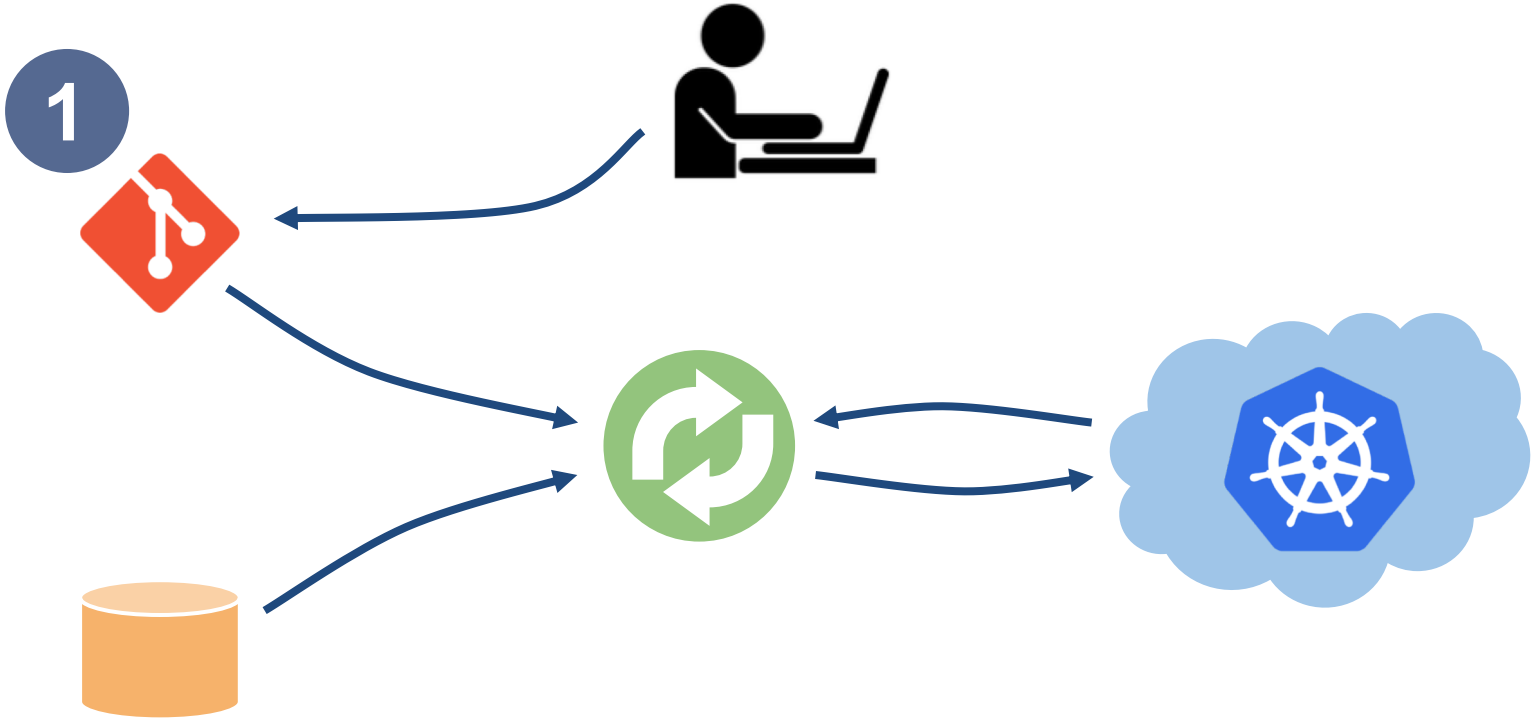
Beyond code, data ⇒

Implementation independent

Easy to abstract in simple ways

Easy to validate for correctness

Easy to generate & manipulate from code



The entire system is described **declaratively**

2 The canonical desired system state is **versioned** (Git)

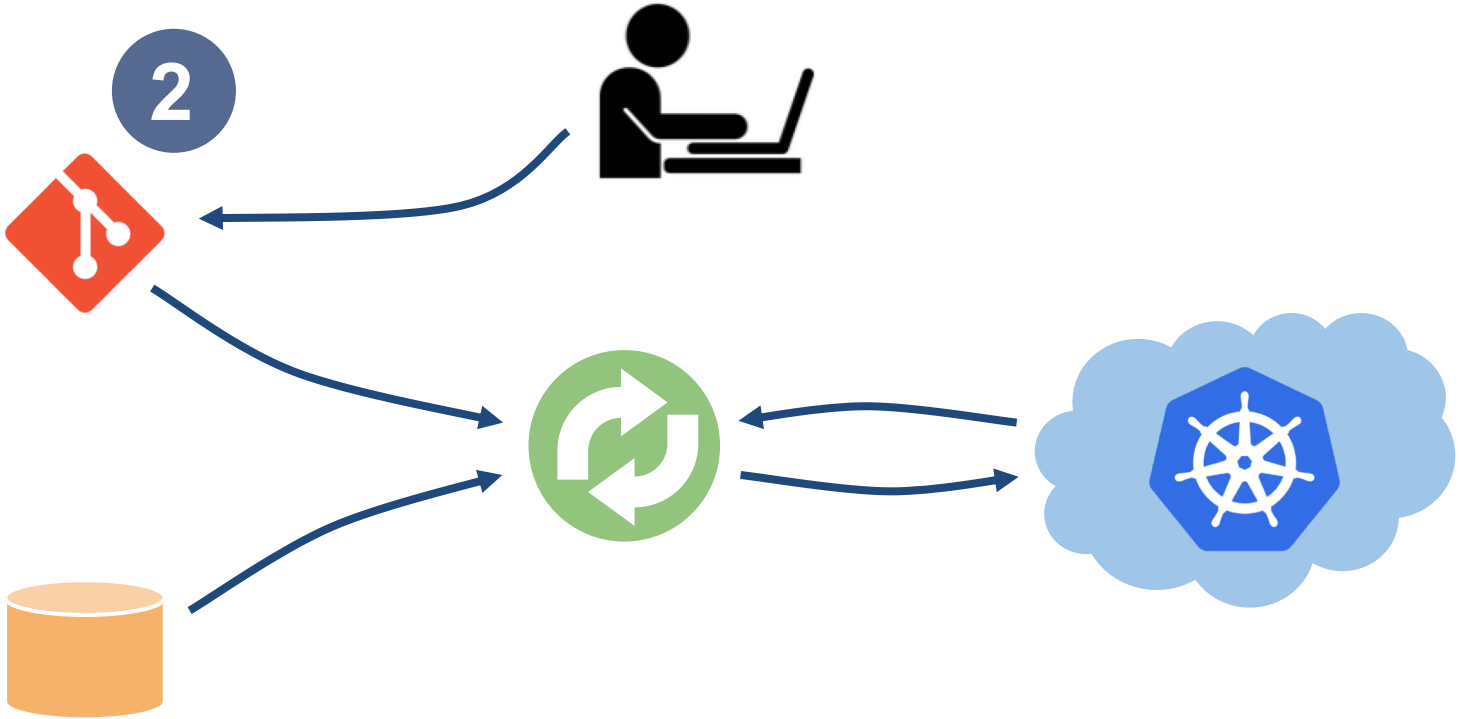
Canonical Source of Truth (DRY)

With declarative definition, trivialises rollbacks

Excellent security guarantees for auditing

Sophisticated approval processes

Great software ↔ human collaboration point



The desired system state is **versioned**

3 Approved changes to the desired state are **automatically applied** to the system

Significant velocity gains

Privileged operators don't cross security boundaries

Separates the **What** and the **How**

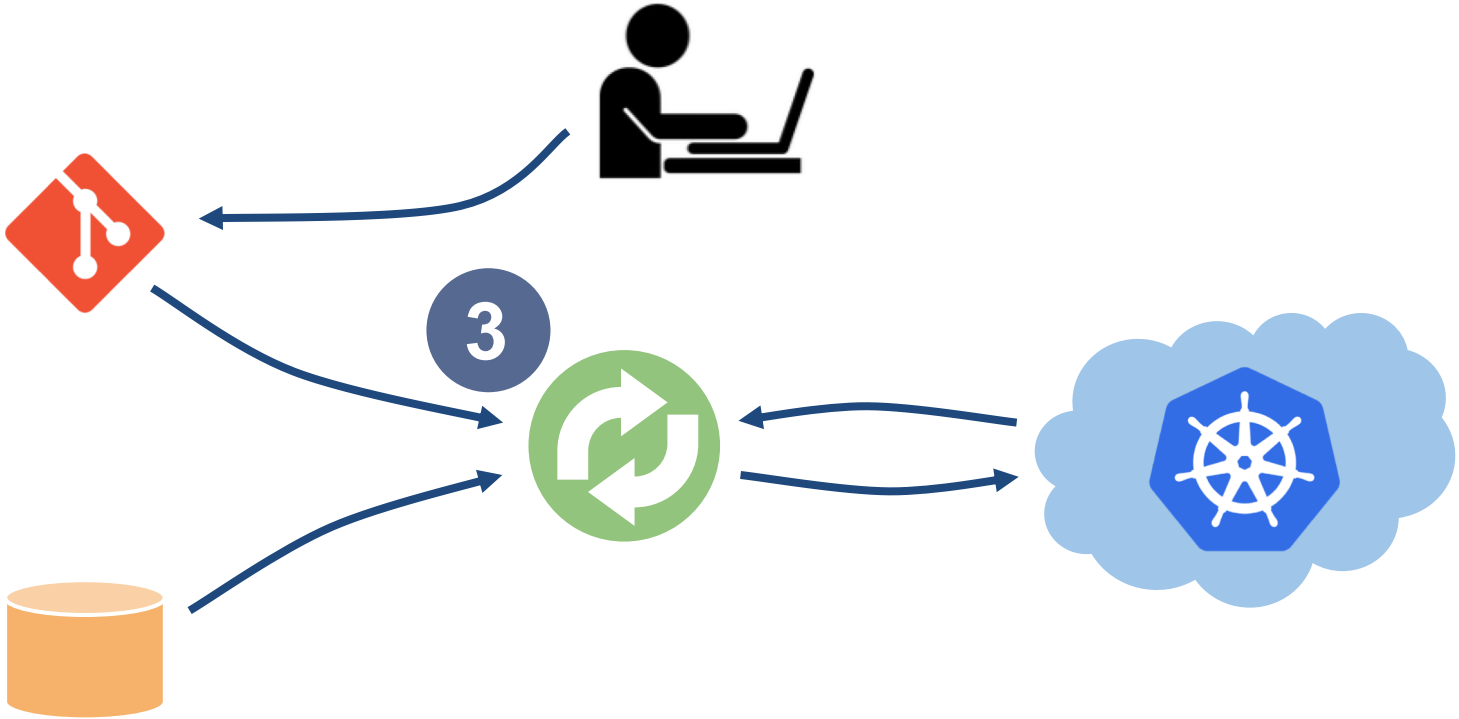


image repository

Approved changes to the desired state are **automatically applied** to the system

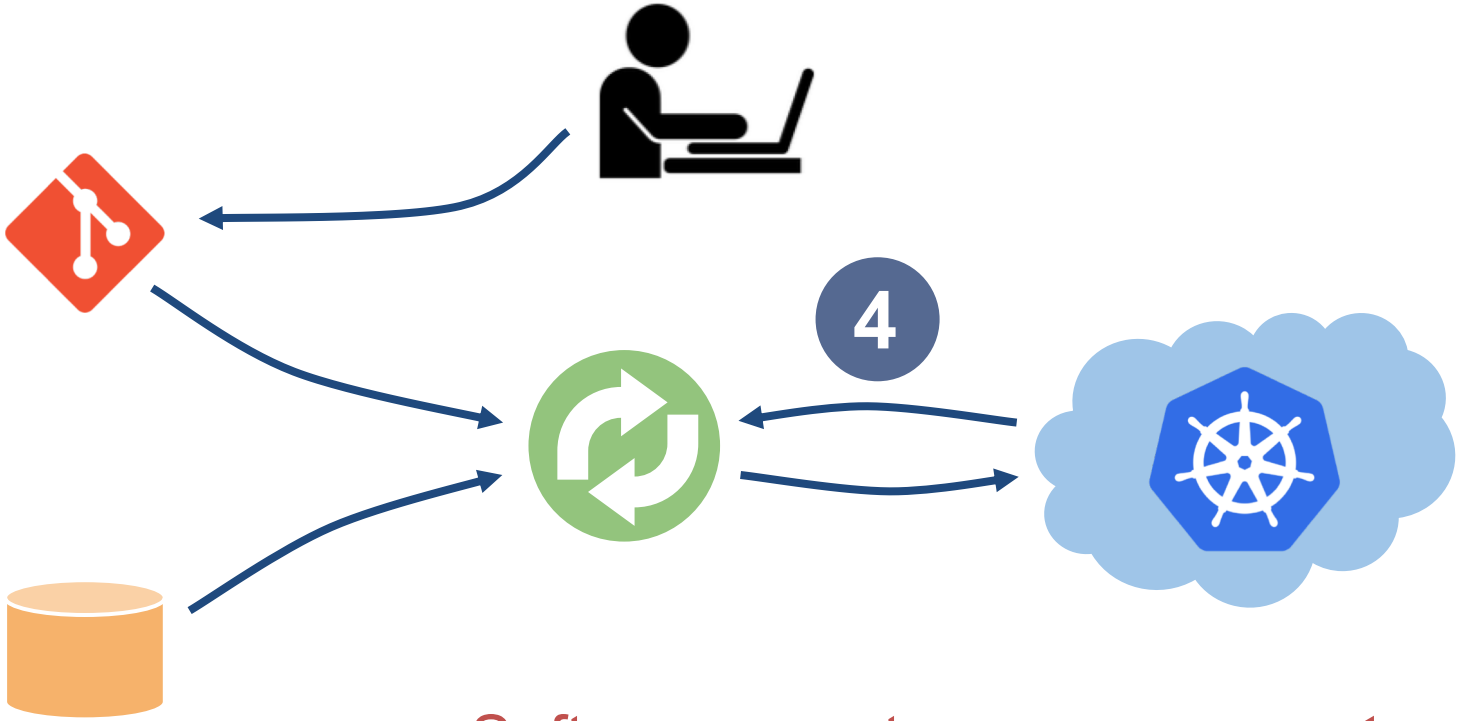
4 Software agents ensure **correctness** and **alert** on divergence

Continuously checking that desired state is met

System can self-heal

Recovers from errors without intervention (layer 8 issues)

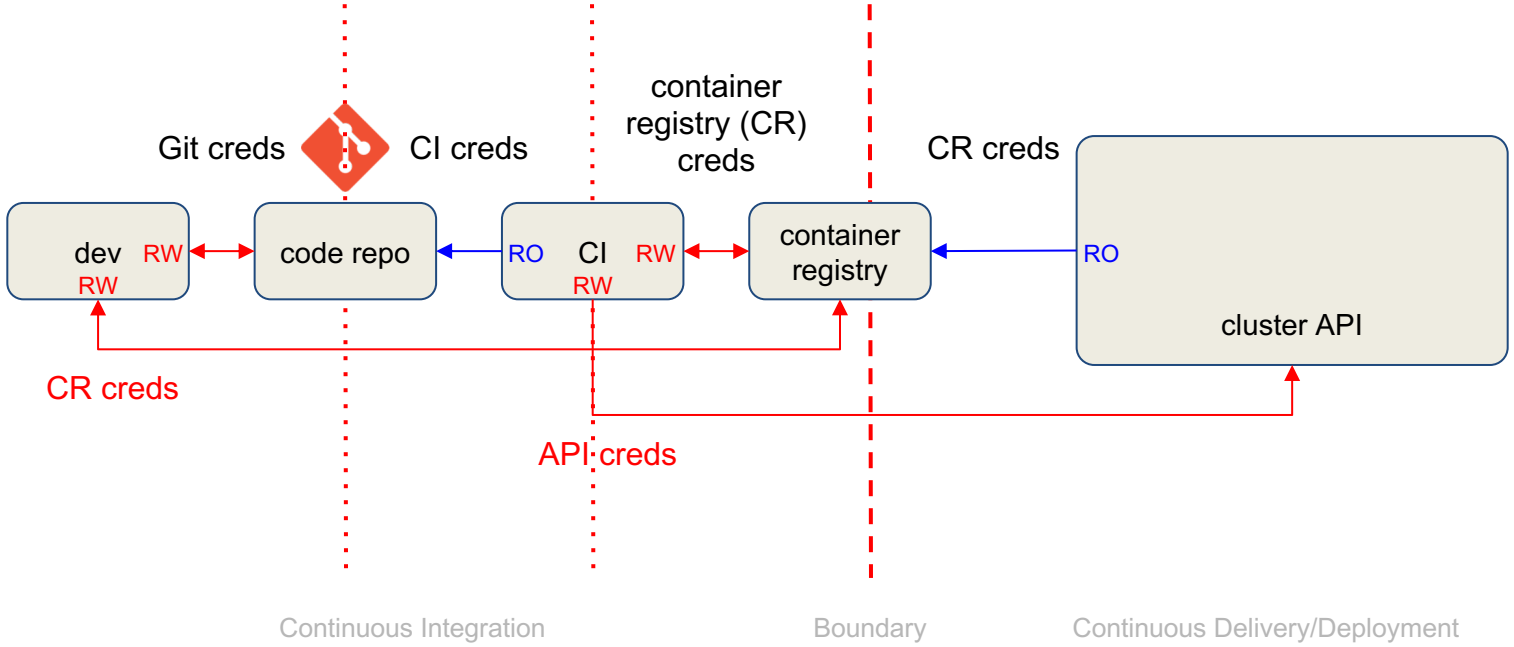
It’s the control loop for your operations



Software agents ensure **correctness** and **alert** on divergence

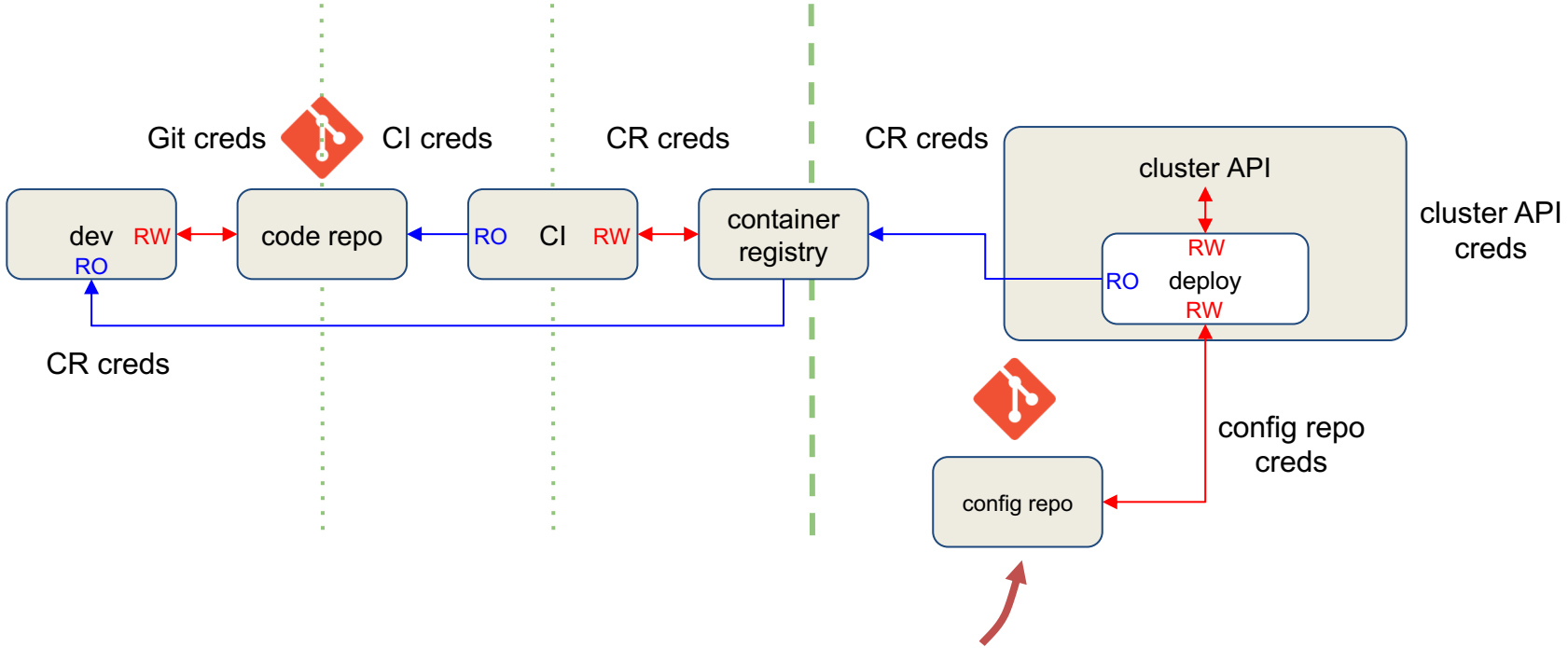
Typical CI/CD pipeline

Shares credentials cross several logical security boundaries.



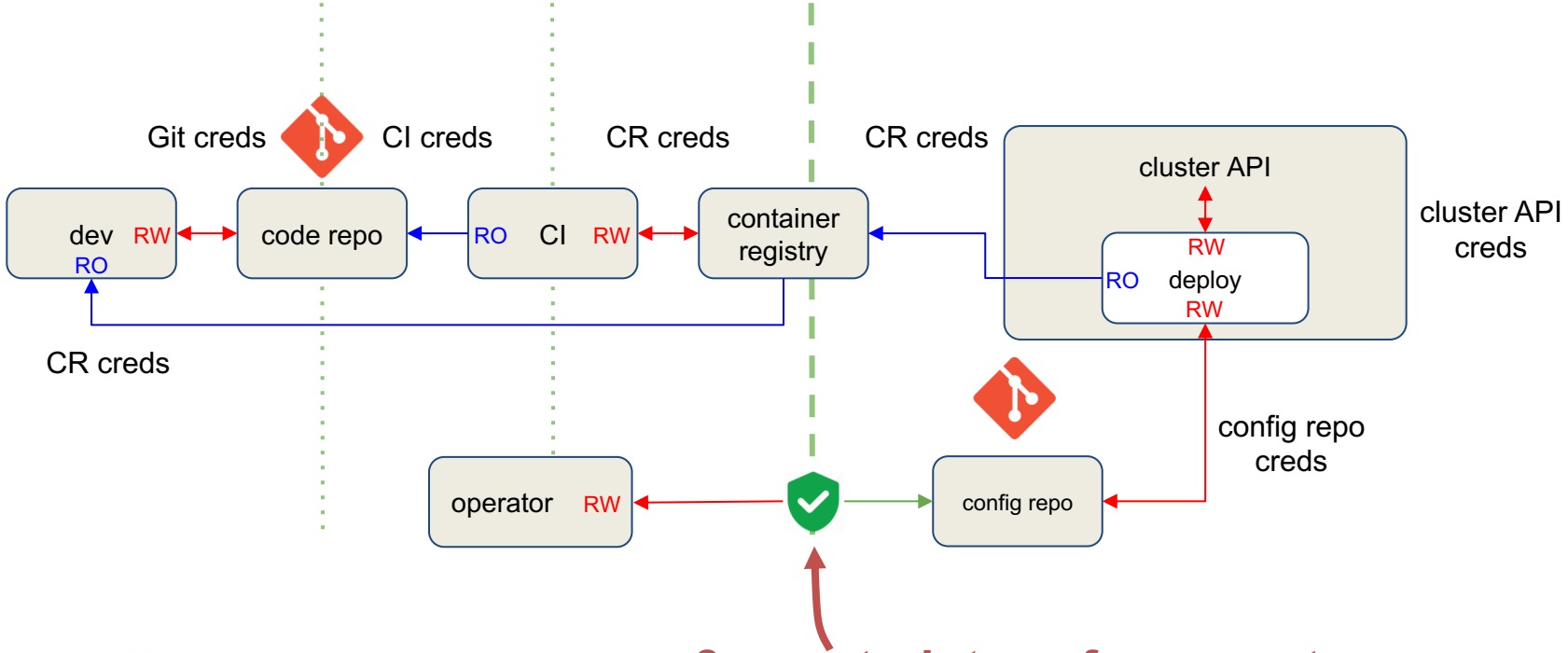
GitOps pipeline

Credentials are never shared across a logical security boundary.



GitOps pipeline

Credentials are never shared across a logical security boundary.



Example workflow

- One Git branch for development:
  - All developers can push changes there for testing purposes
- One Git branch for staging:
  - Project lead or devops manager has ability to push/merge here for testing the complete staged application
- One Git branch for production:
  - The system operations manager is the only person to have push/merge authorization here

Example workflow

Developer writes code, compiles, and builds an image using the standard CI process. Developer pushes the image(s), and updates the manifest in the GitOps development branch. Developer pushes the new manifest into GitOps, and the development cluster now runs the new images.

Once the testing is complete, the developer enters a pull request from the developer GitOps to the staging GitOps.
The pull request is reviewed, and if accepted, is merged into the staging GitOps. The staging cluster is updated with the new application.

Once the staging and integration is complete, the DevOps manager enters a pull request from the staging GitOps to the production GitOps. The pull request is reviewed, and if accepted, is merged into the production GitOps. The production cluster is updated with the new application.

GitOps in Action!

Tooling

- CNCF Flux
- ArgoCD
- Gitkube
- Tekton
- JenkinsX

Flux

github.com/fluxcd/flux

Trainings environment:

301.sh/velocity-gitops-101

Trainings repo:

github.com/mhausenblas/gitops101

So what actually happened?

1. We installed the Flux agent in our cluster
2. We added the agent key to our repository, so that it can read and write the configuration
3. We configured the agent to watch to our repository
4. The agent noticed some manifests in the repository, and automatically applied them
5. Kubernetes deployed the manifests

Progressive delivery

What is progressive delivery?

Progressive Delivery is Continuous Delivery with fine-grained control over the blast radius, requiring:

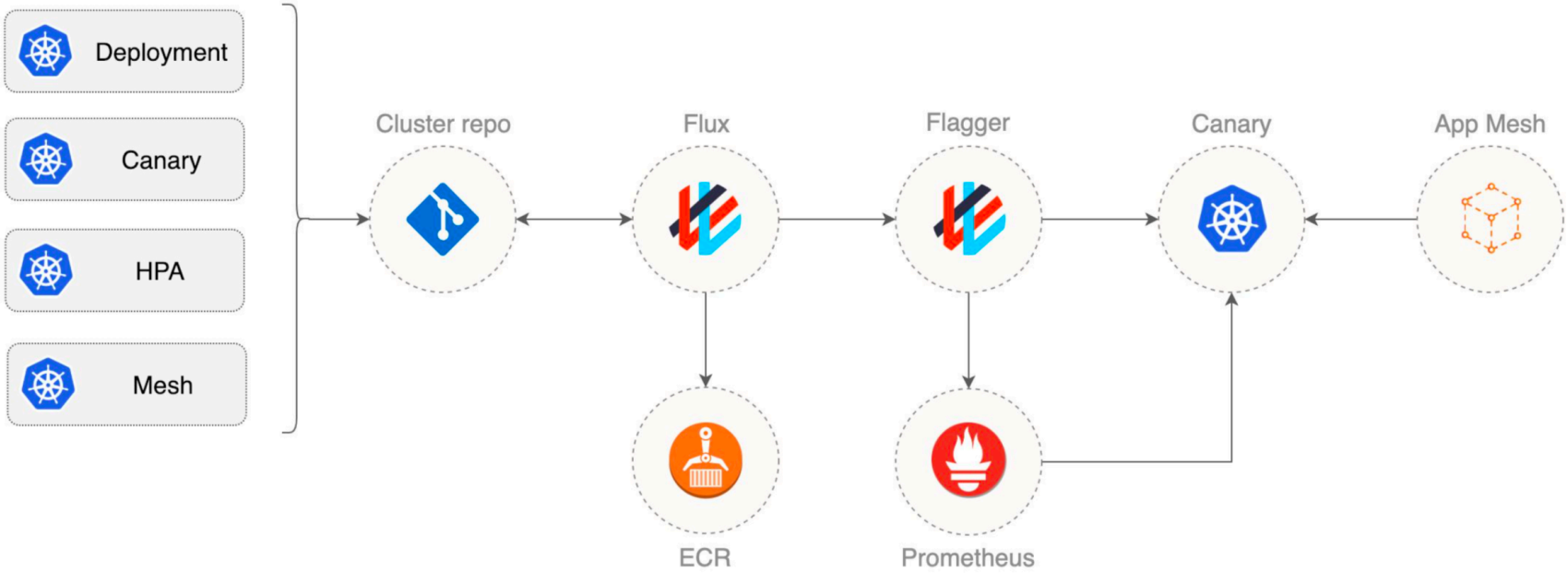
- CI pipeline that produces immutable build artifacts
- CD pipeline designed for desired state reconciliation
- Smart routing for user facing apps and service to service communication
- Observability (performance stats + business metrics)
- Fail fast mentality

Flagger

Flagger is a Kubernetes operator that automates the promotion of canary deployments using App Mesh, Istio, Gloo or NGINX routing for traffic shifting and Prometheus metrics for canary analysis.

Flagger implements a control loop that gradually shifts traffic to the canary while measuring key performance indicators. Based on the KPIs analysis a canary is promoted or aborted.

Flagger


Deployment
Canary
HPA
Mesh
Cluster repo
Flux
Flagger
Canary
App Mesh
ECR
Prometheus

Challenges

Technical challenges

- All your artifacts in VCS such as Git?
- Tooling selection
  - app-focused vs infra-focused
  - pace of ecosystem

Organizational challenges

- Is there a devops mentality in your organization?
- Is your organization ready for “cloud native”?

Recap and Resources

Recap: GitOps CI/CD

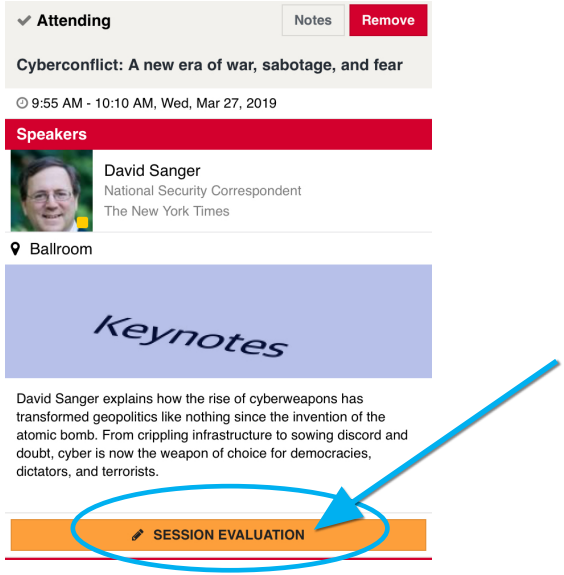
- Git is our single source of truth
- Deployments and rollback are all done via Git
- Auditing built-in
- Having separate pipelines for CI and CD enables better security
- Automated the deployment
- It’s easier to deal with if a deployment goes wrong

Resources

- https://www.gitops.tech
- https://github.com/weaveworks/awesome-gitops
- https://thenewstack.io/what-is-gitops-and-why-it-might-be-the-next-big-thing-for-devops/
- https://www.reddit.com/r/kubernetes/comments/c2wgdz/gitops_in_production_share_your_experiences/
- https://deploy.live/blog/a-year-with-gitops-in-production/

Rate the session

Session page on conference website



O'Reilly Events App