GitOps is Likely More Than You Think it is

Kubecon
November 20, 2020

Cornelia Davis
CTO, Weaveworks
@cdavisafc • cornelia@weave.works

Me?

Developer (wasn't Ops)

Web architectures for ~15 years

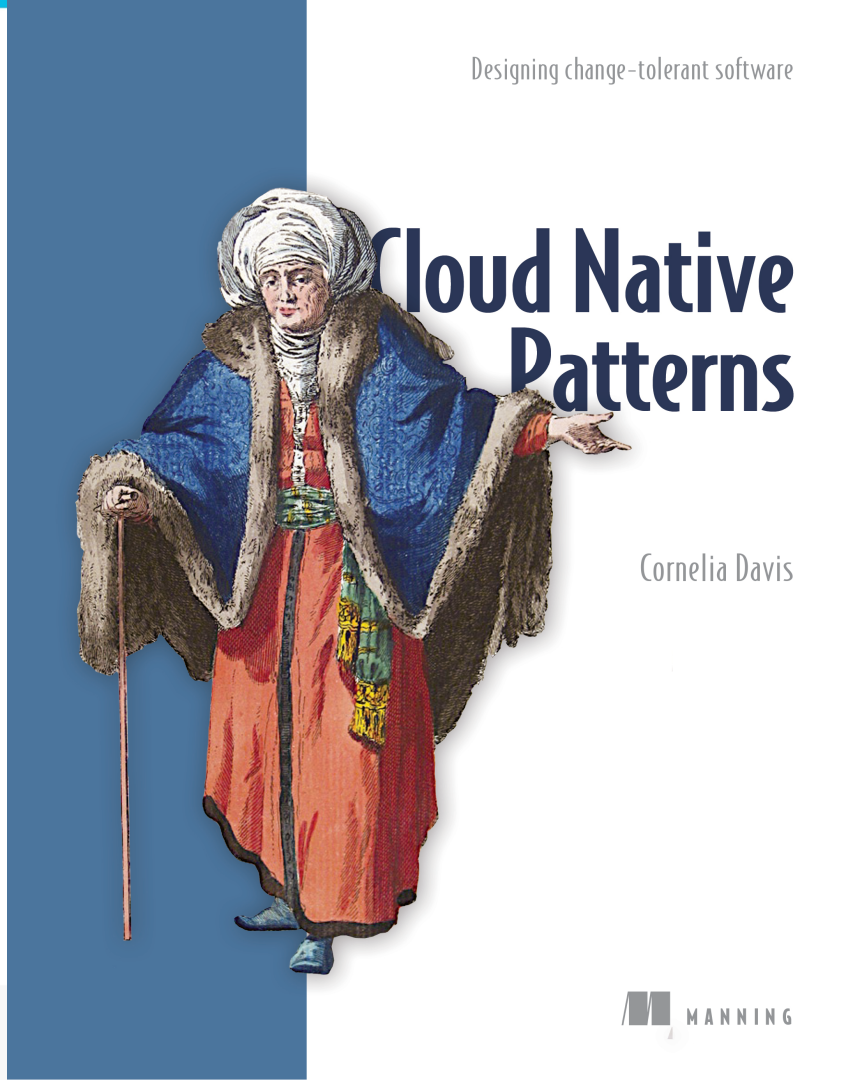
Cloud-native for nearly a decade

Cloud Foundry for 8+ years

Kubernetes for nearly 4

Discount code 35% off!: ctwkubecloud20

Go here → http://mng.bz/4BlV



GitOps - Cloud Native Agility and Reliability

Solution

GitOps is a set of modern best practices for deploying and managing cloud native infrastructure and applications.

Based on our experience operating a full cloud native stack

GitOps manages the whole stack:

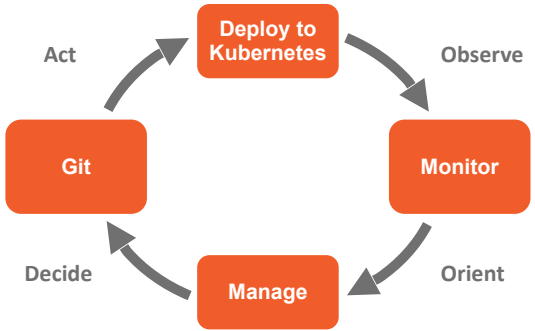
- Cluster and application versioned configuration
- Security and policy enforcement
- Monitoring and observability
- Continuous Deployment of workloads

Benefits

- **Complete platform**: Single platform for infrastructure, core components and applications.
- **Productivity**: Dramatically increase deployments and faster feedback and control loop,
- **Reliability**: Enables cluster and application operator model with standardised tooling.
- **Compliance and Security**: Enforces standard security policy and an audit trail
- **Multi-cloud and on-premise**: Deploy a complete cluster from git with all applications.

Vision

- All application deployments, application operations and cluster management operations under one platform with a common workflow.

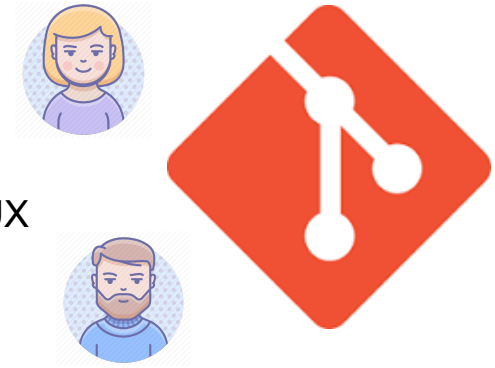


So then, how do you GitOps?

Store code/config

Automation

Runtime Environment

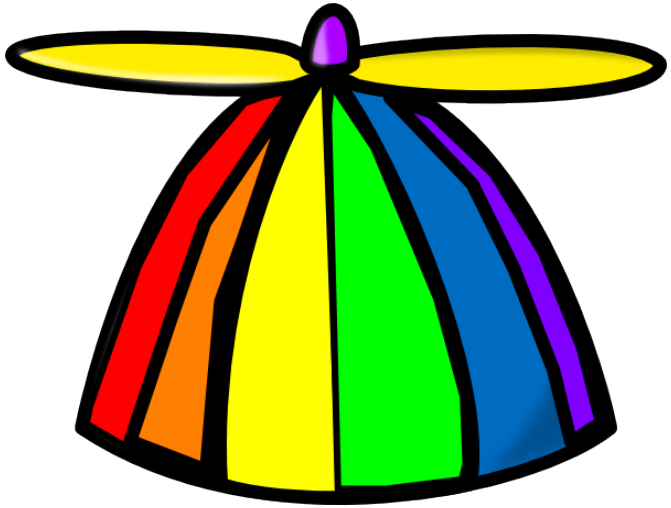


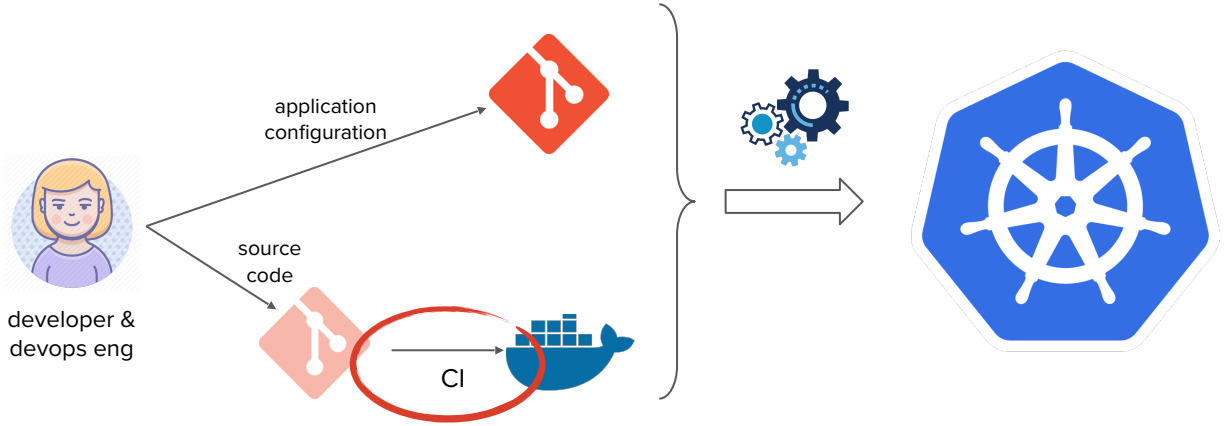
UX

Yes, but…

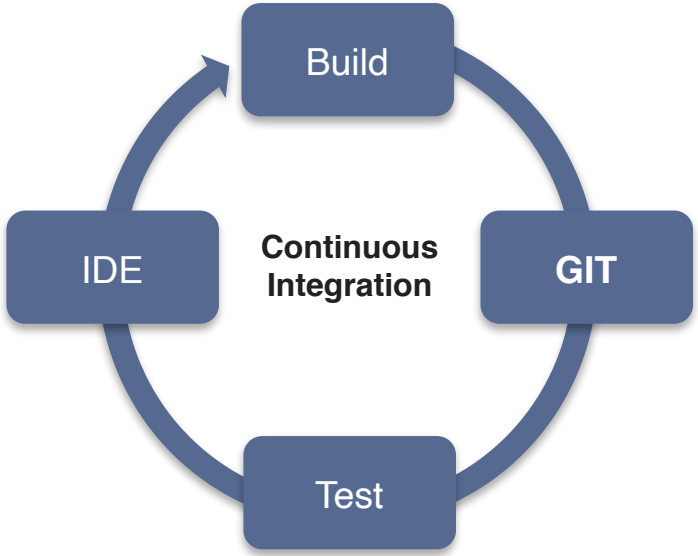
… details matter

How we do this relates directly to how many of the benefits we enjoy



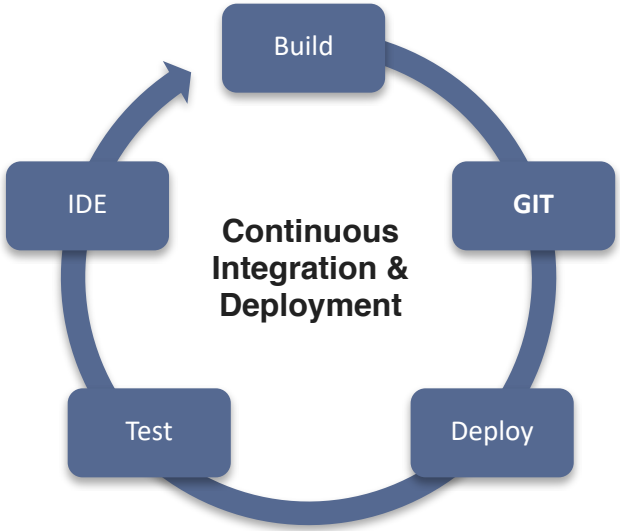

application
configuration
source
code
developer &
devops eng
CI

Continuous Integration as Development Model



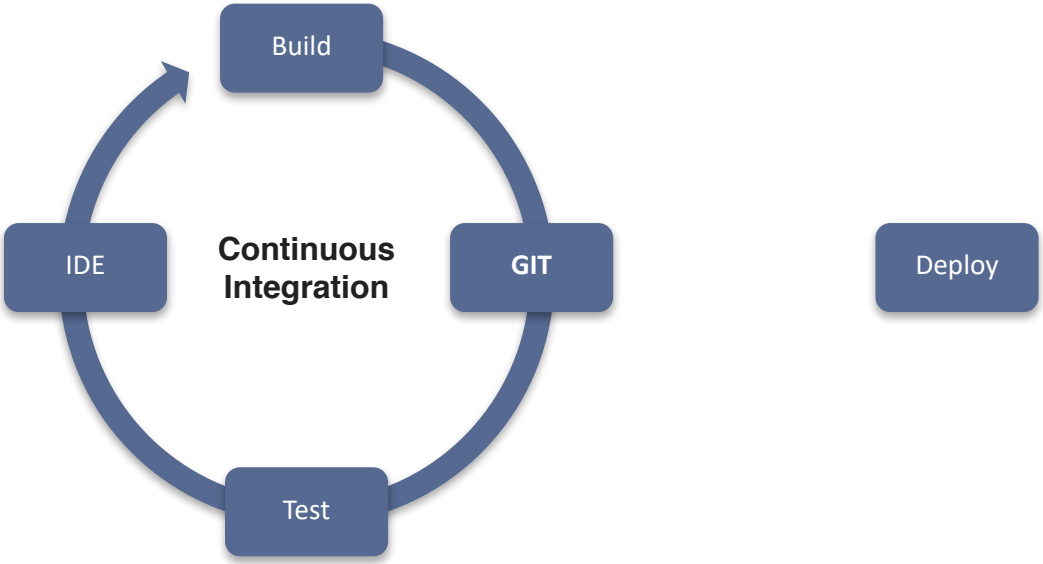
- Iterative Process of Software Build & Test
- Developers Control the Flow and Process
- Delivers Higher Quality in Shorter Times
- Well Known Tools and Methodologies
- Well Supported Solutions: both Commercial & Open Source

BUT Continuous Integration is NOT Continuous Delivery/Deployment



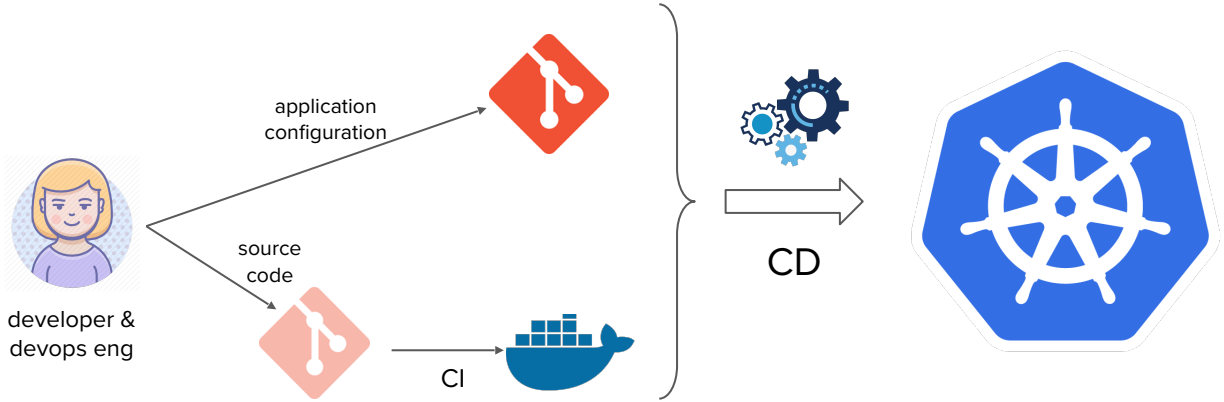
Application Deployment Should be Separate from CI

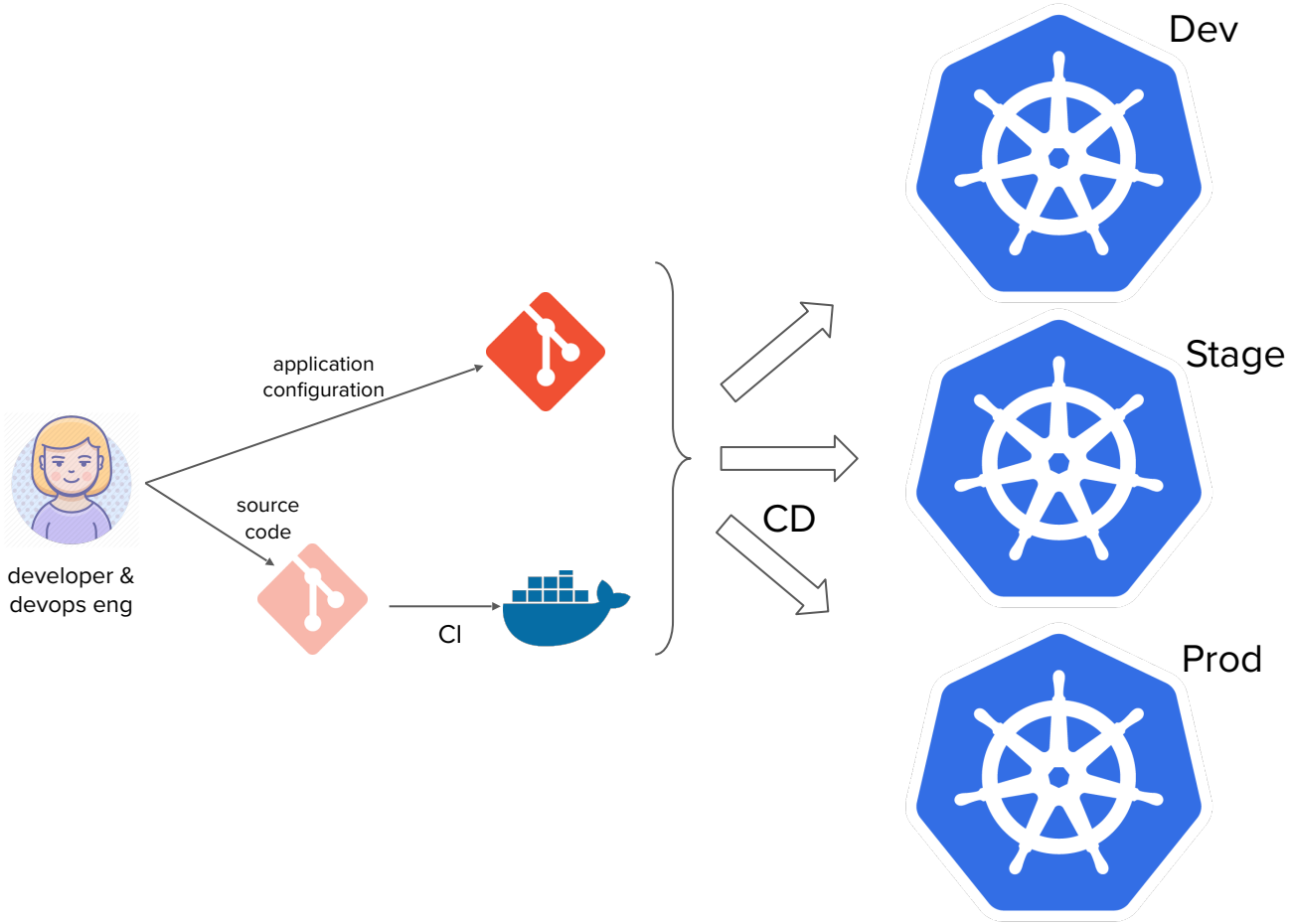
- Separation of Concerns: Developers Release, Operators Deploy
- Many deployment environments
- Recreating a deployment shouldn't require a new build
- …

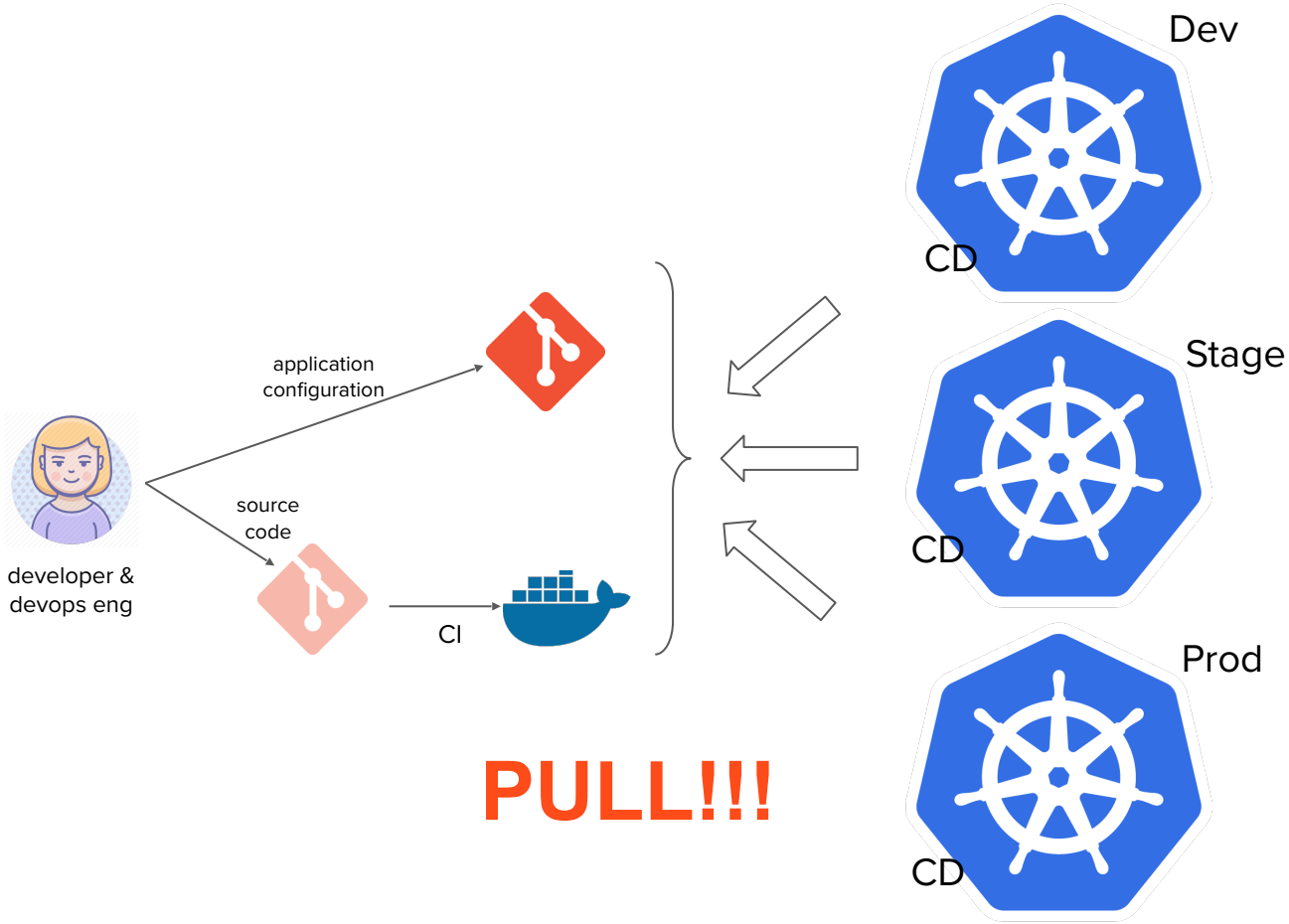

Build
IDE
Continuous Integration
GIT
Test
Deploy

There is no CI/CD

There are CI and CD!

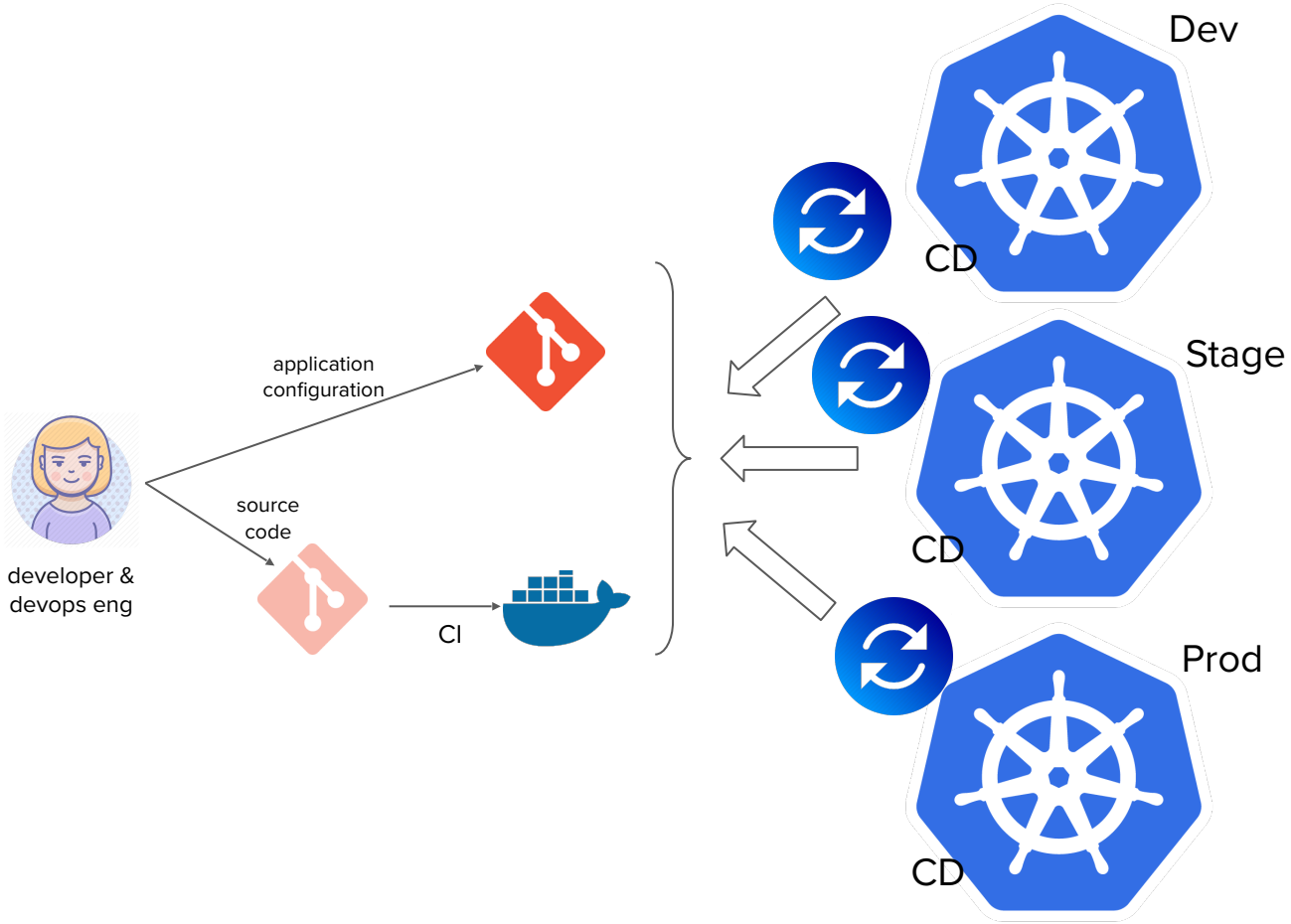

application
configuration
source
code
developer &
devops eng
CI
CD

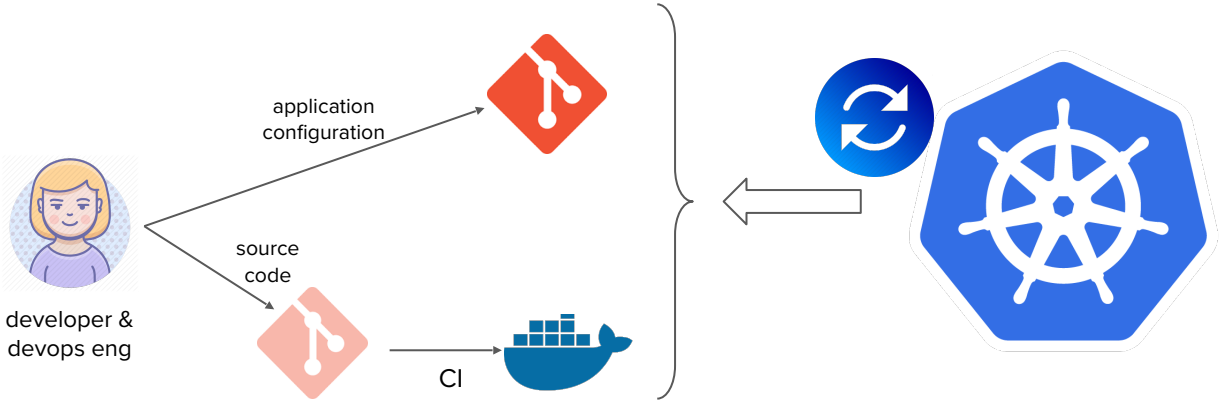

Dev
Stage
Prod
application
configuration
source
code
developer &
devops eng
CI
CD


Dev
CD
Stage
CD
Prod
CD
application
configuration
source
code
CI
developer &
devops eng
PULL!!!

But then, how do we know when to pull?

We don’t have to!


Dev
CD
Stage
CD
Prod
CD
application
configuration
source
code
CI
developer &
devops eng


application
configuration
source
code
developer &
devops eng
CI

So, this enables a bunch of additional interesting patterns

Drift detection and remediation

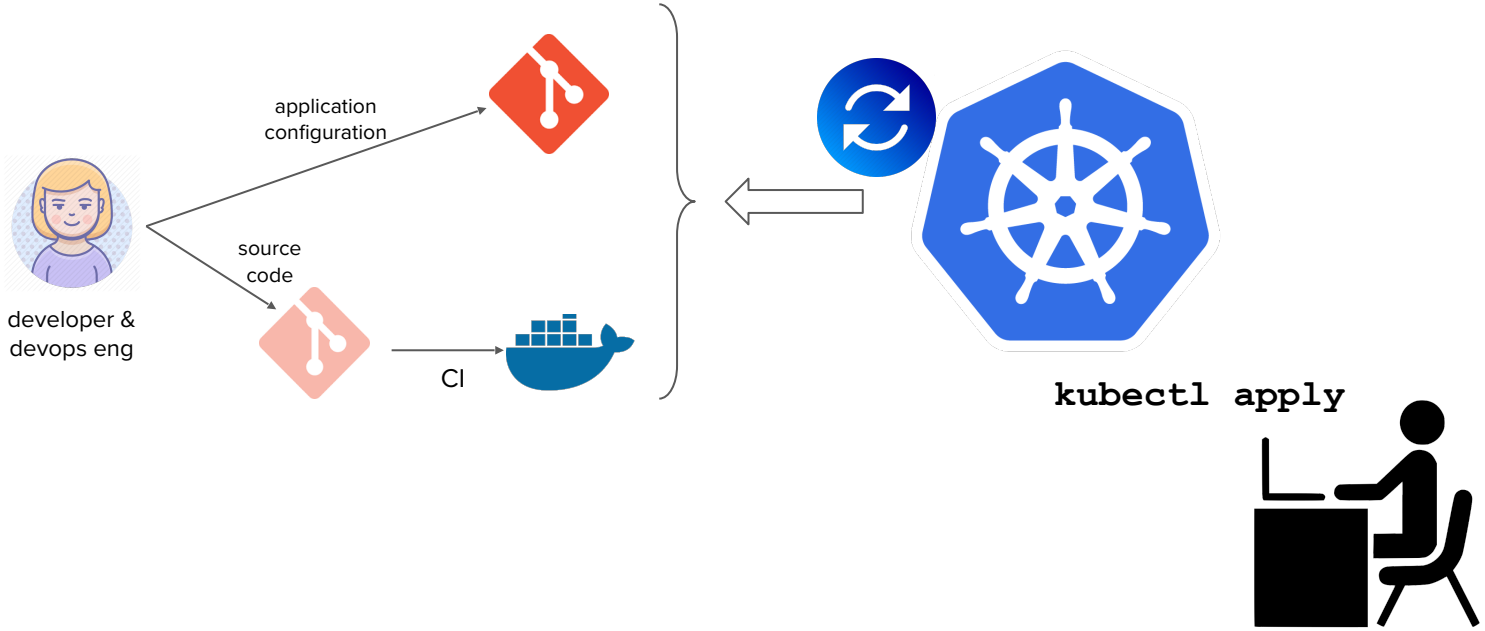


Image Update Automation


```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: posts
  labels:
    app: posts
spec:
  replicas: 2
  selector:
    matchLabels:
      app: posts
  template:
    spec:
      containers:
      - name: posts
        image: myimage:0.0.1
```

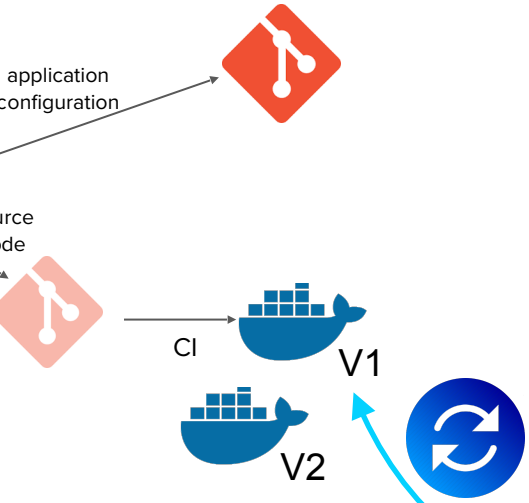


Image Update Automation


```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: posts
  labels:
    app: posts
spec:
  replicas: 2
  selector:
    matchLabels:
      app: posts
  template:
    spec:
      containers:
      - name: posts
        image: myimage:0.0.2
```

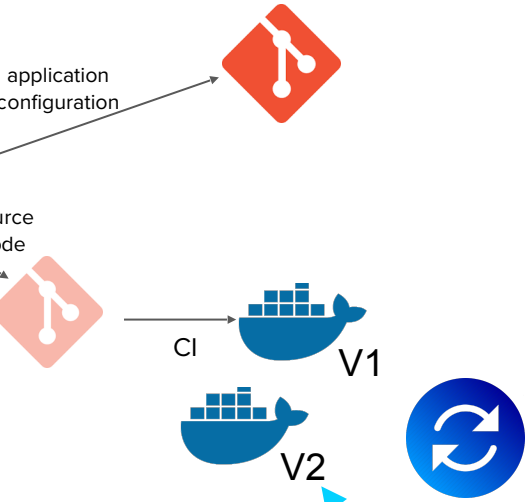
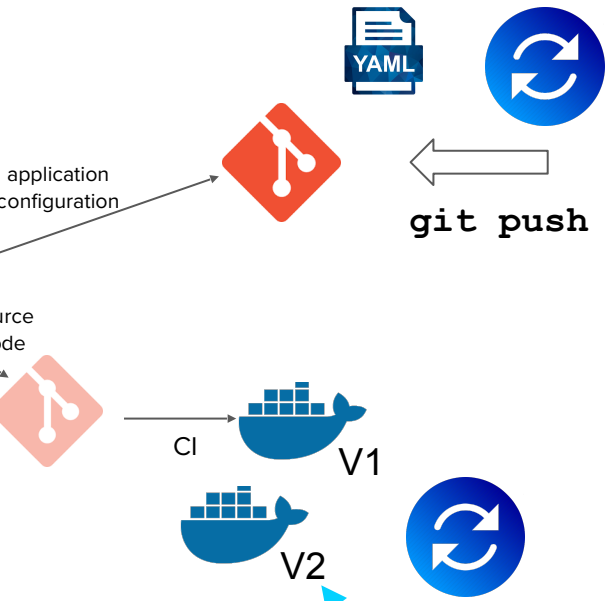
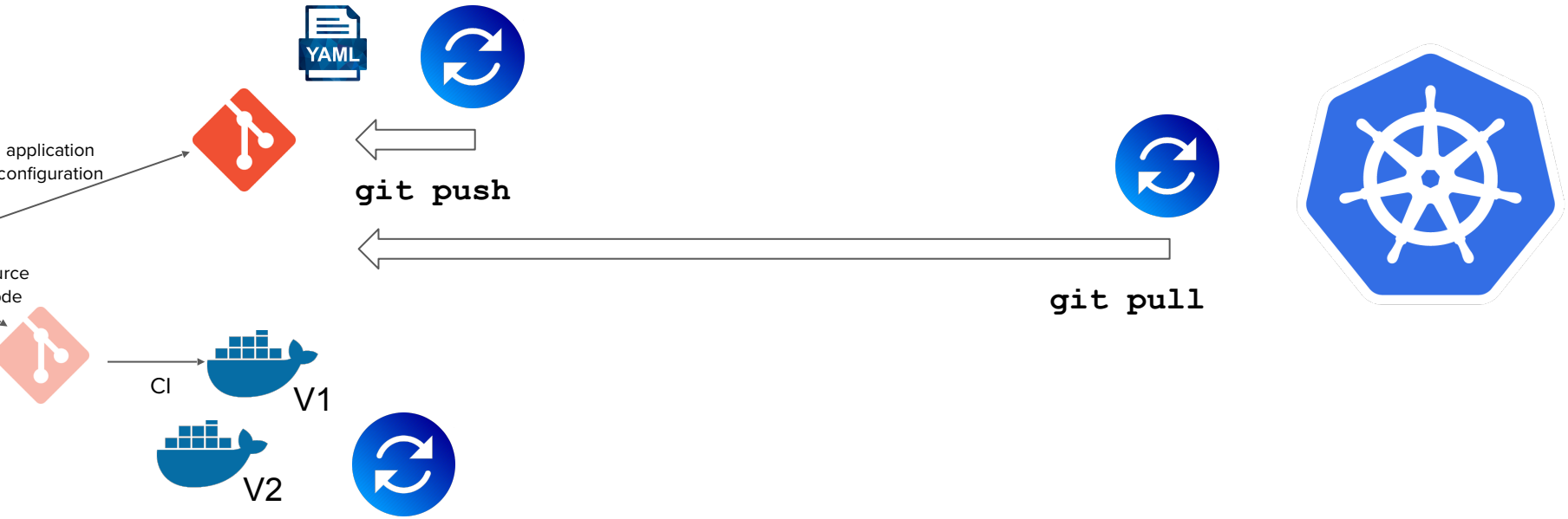


Image Update Automation

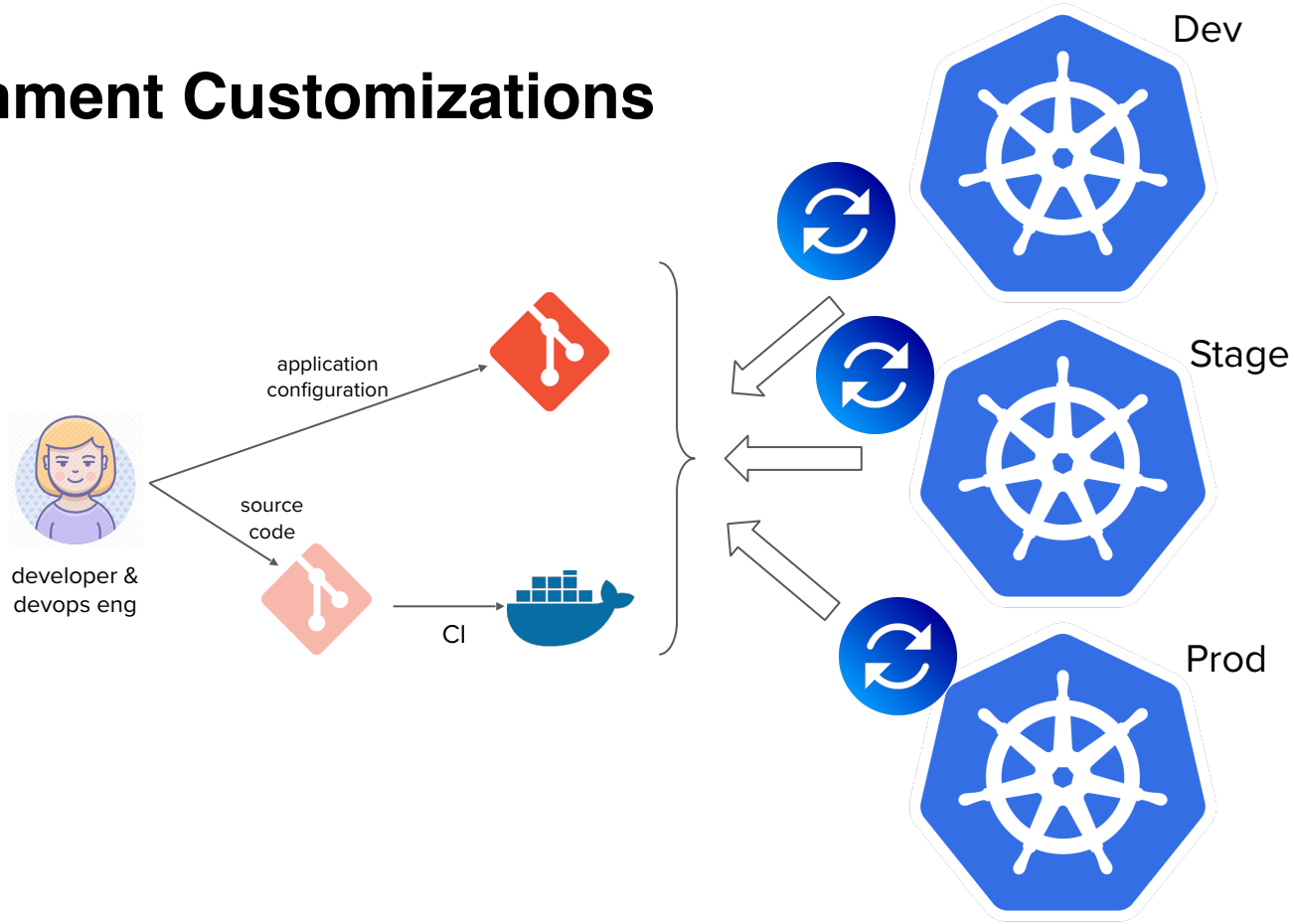



```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: posts
  labels:
    app: posts
spec:
  replicas: 2
  selector:
    matchLabels:
      app: posts
  template:
    spec:
      containers:
      - name: posts
        image: myimage:0.0.2
```

Image Update Automation



Environment Customizations



Environment Customizations

```
apiVersion:
kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ./mydeployment.yaml
```

application configuration

CI

```
...
spec:
  replicas: 2
```

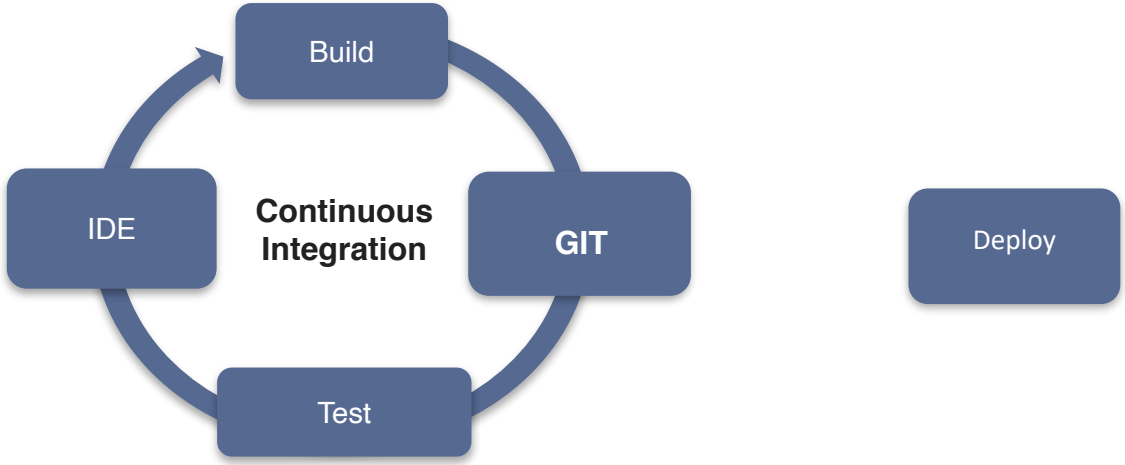
Dev

```
...
spec:
  replicas: 20
```

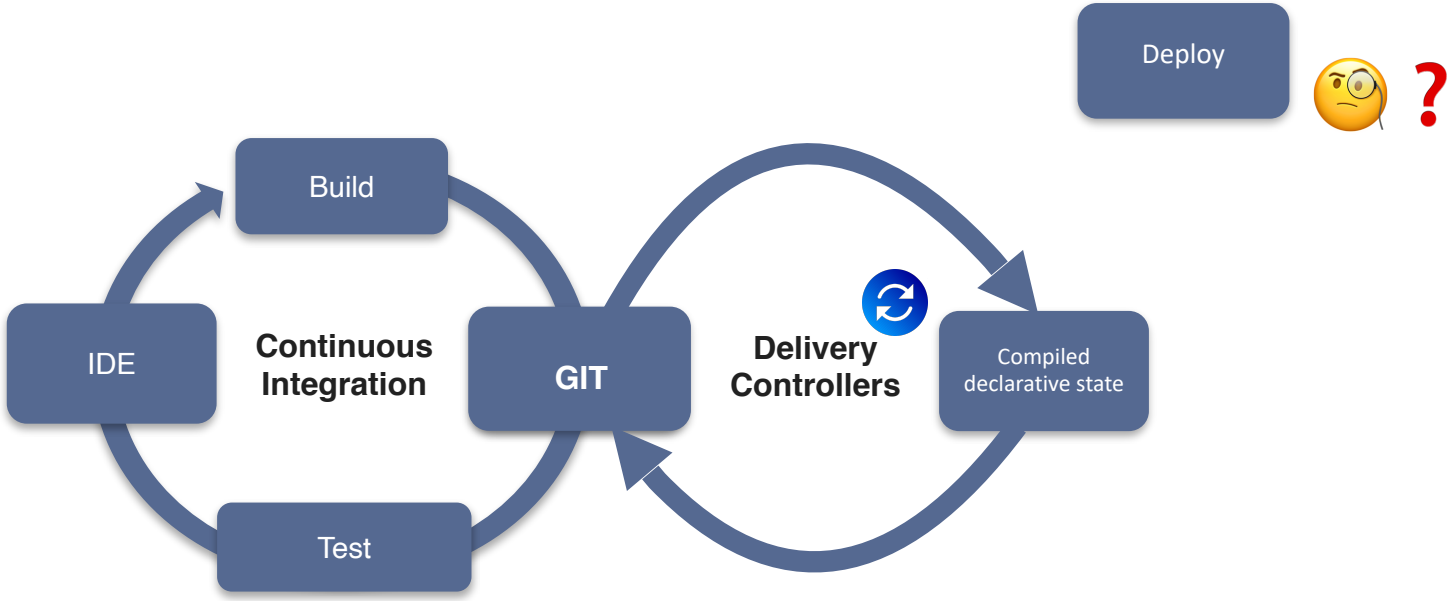
Stage

```
...
spec:
  replicas: 10
```

Prod



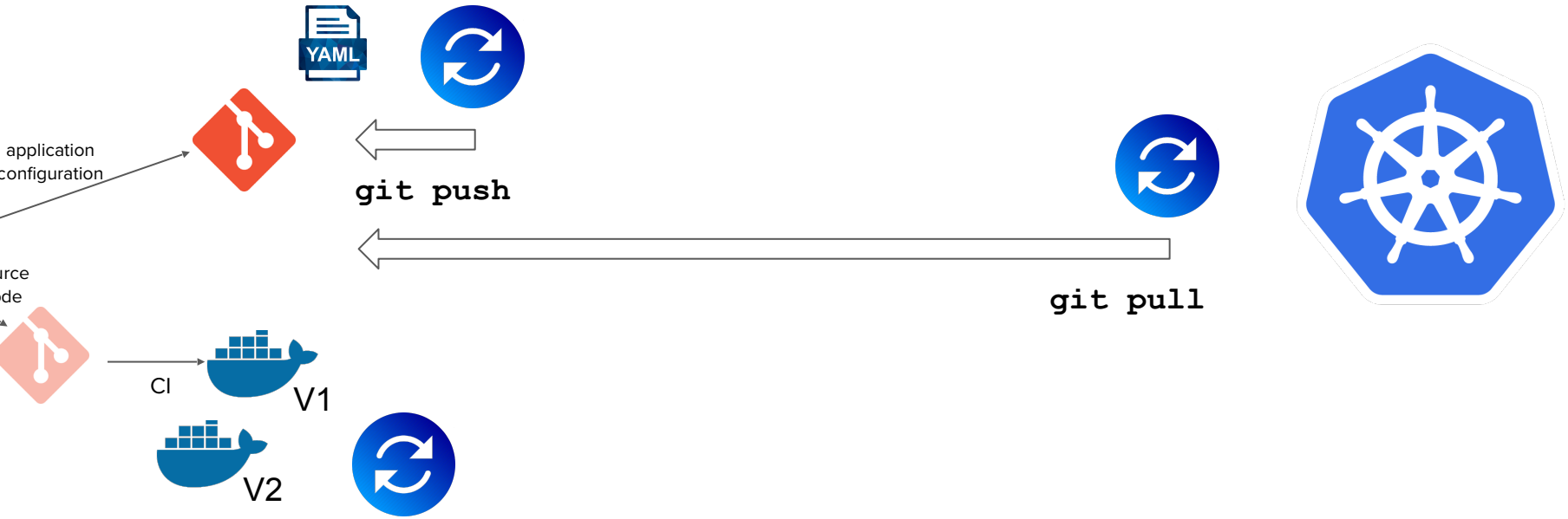
Deploy


Deploy
Build
IDE
Continuous Integration
GIT
Delivery Controllers
Compiled declarative state
Test

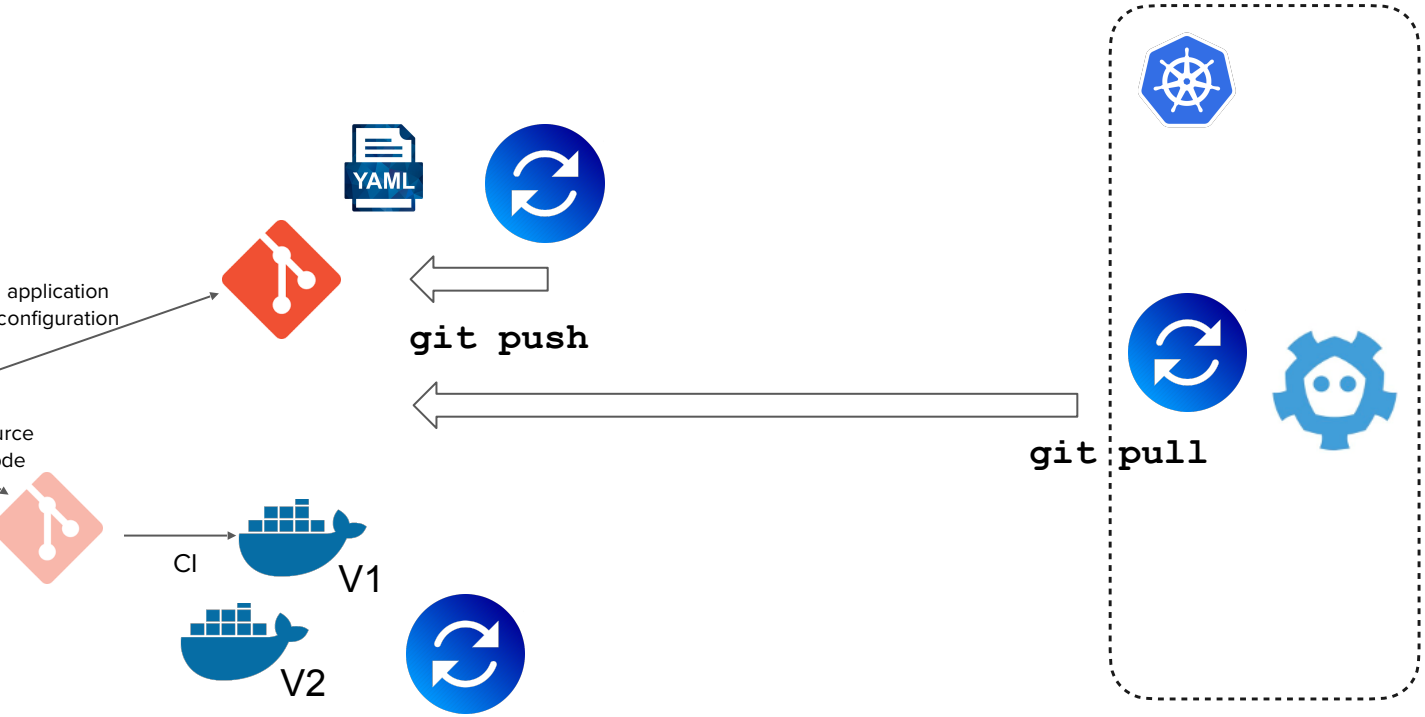
We are getting there…

… let’s look at one more pattern

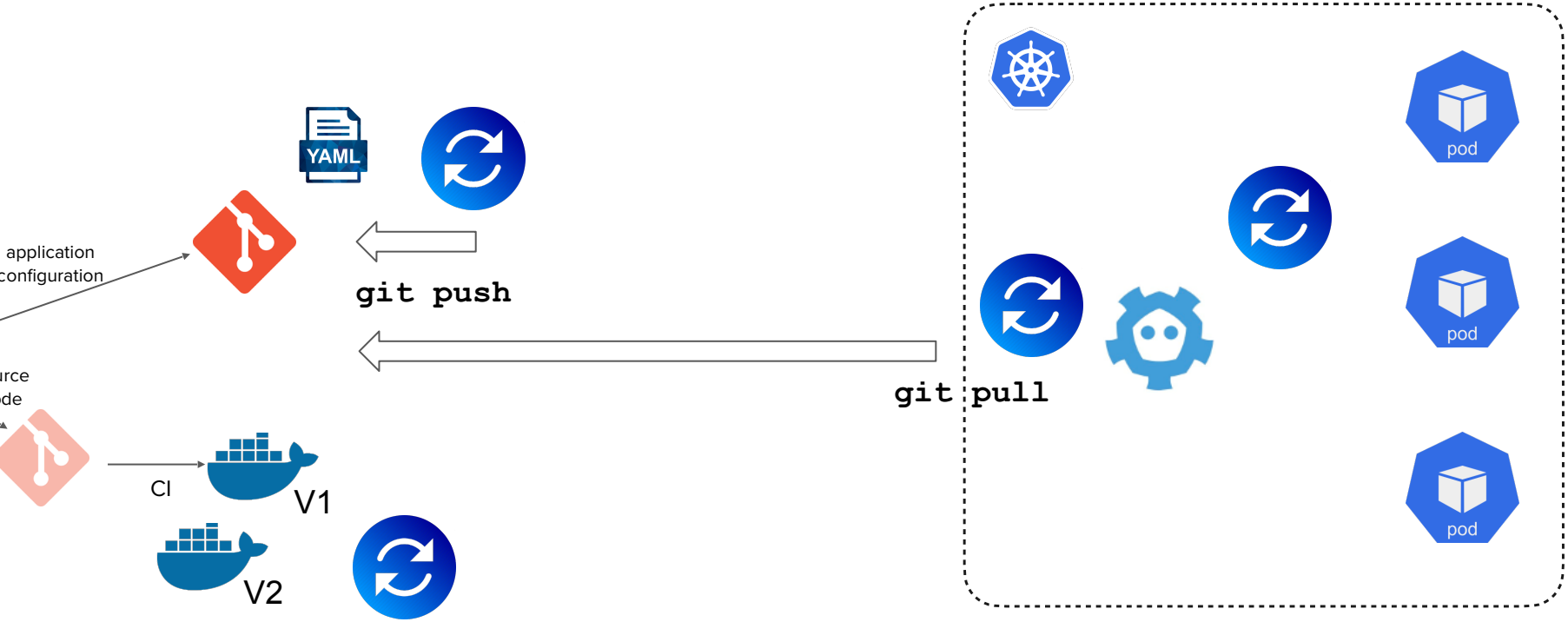
Recall… Image Update Automation



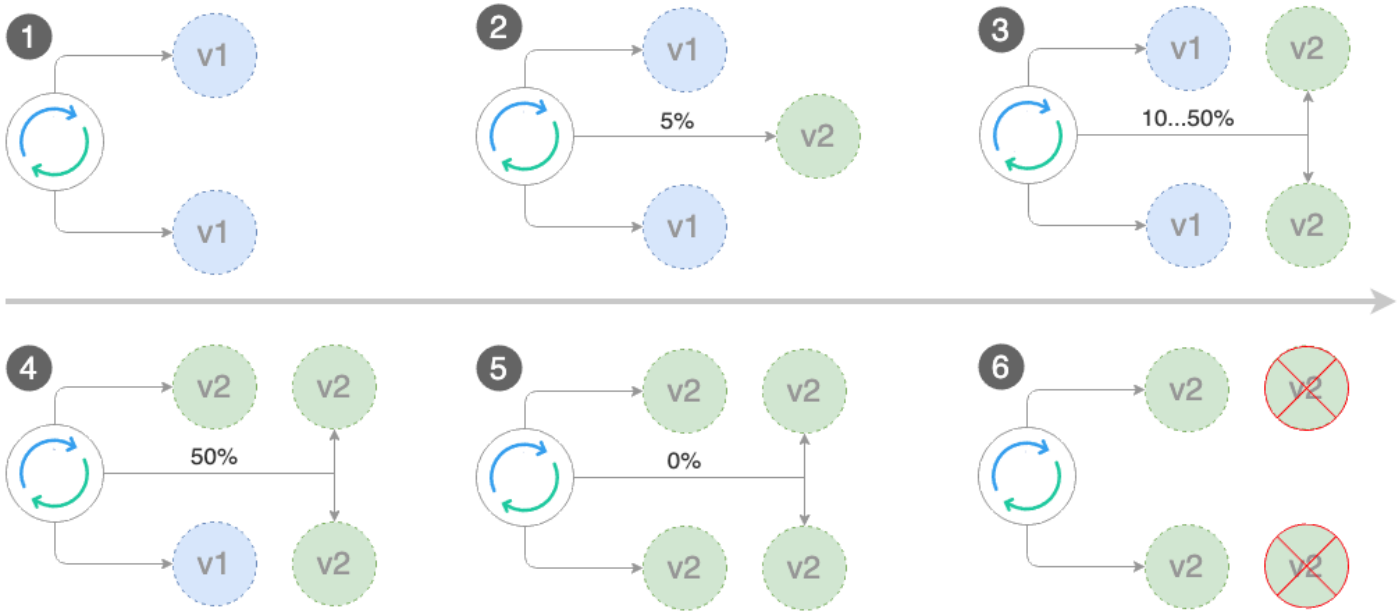
… this is only one half of the equation



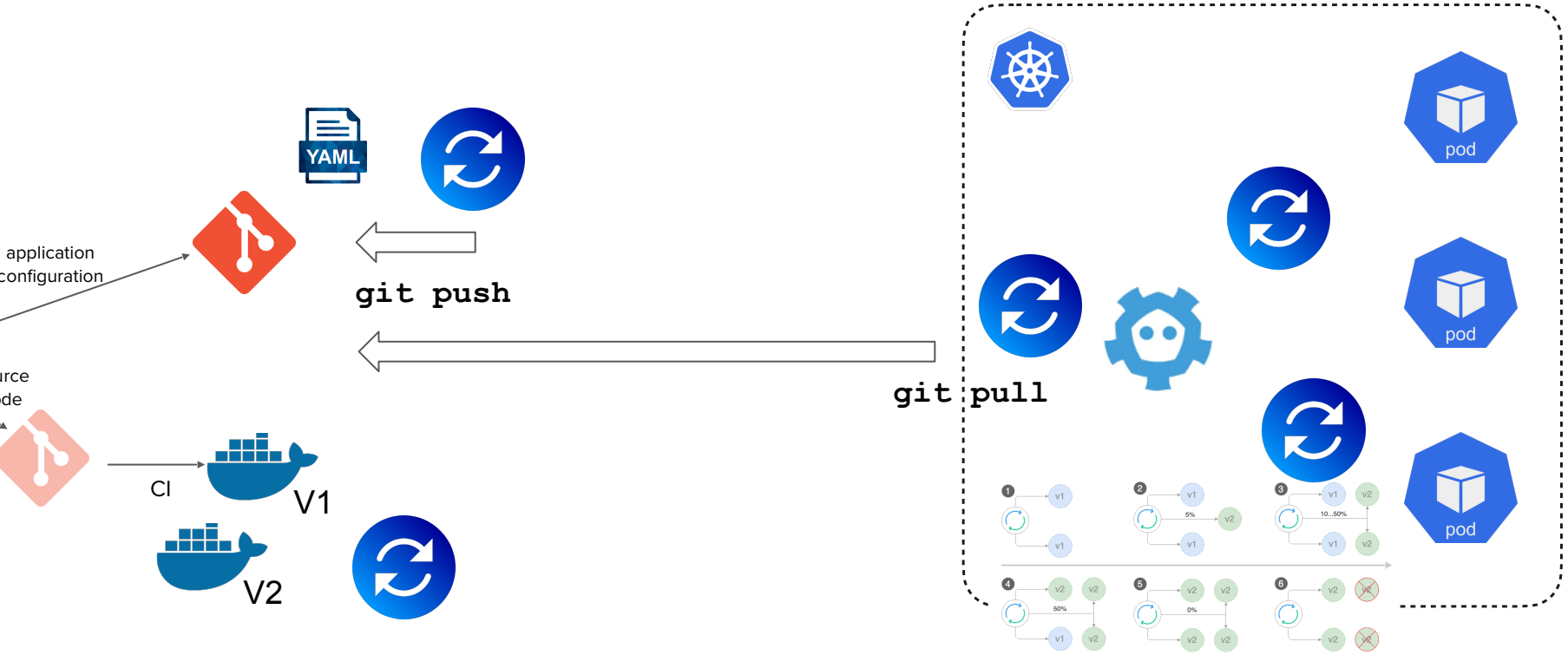
… the other half gets things running



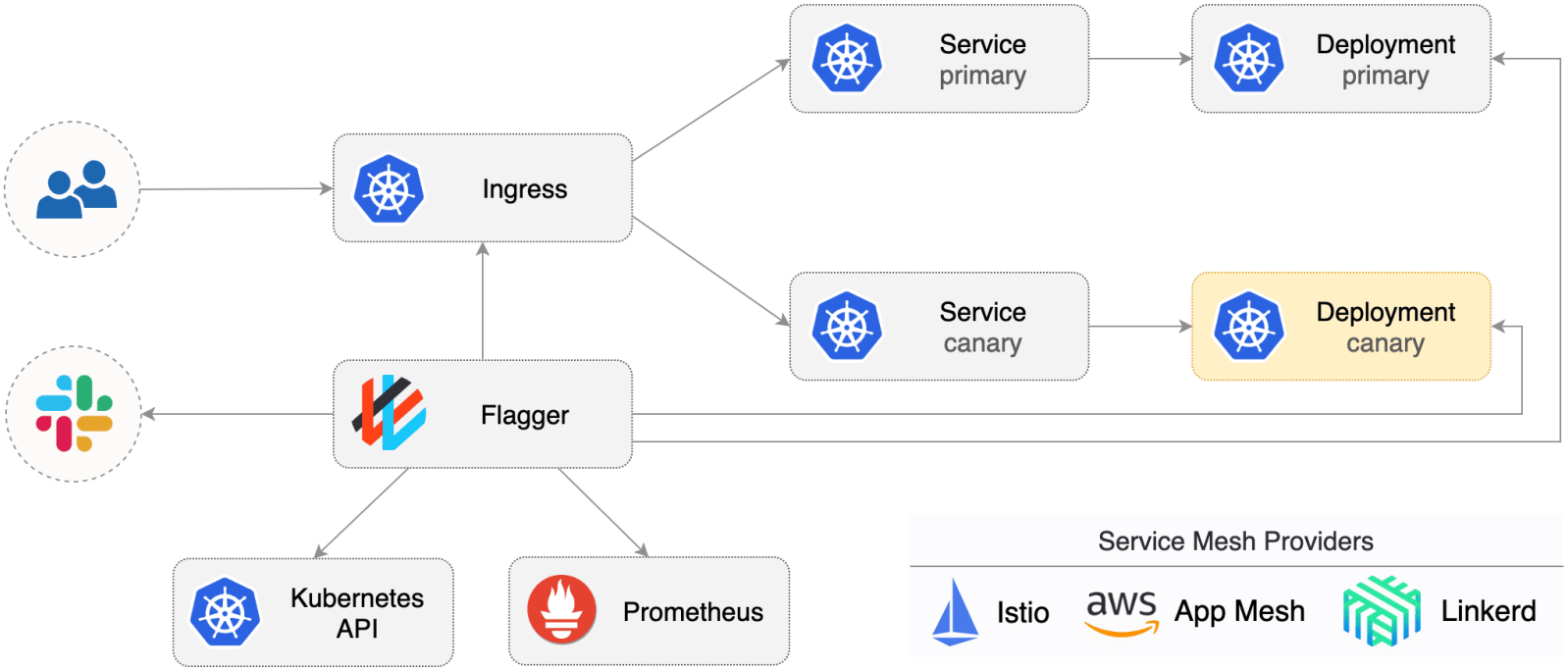
And what if you wanted a different deployment strategy?



… you guessed it



Flagger


Service
primary
Deployment
primary
Ingress
Service
canary
Deployment
canary
Flagger
Kubernetes
API
Prometheus
Service Mesh Providers
Istio
aws
App Mesh
Linkerd

Release strategies

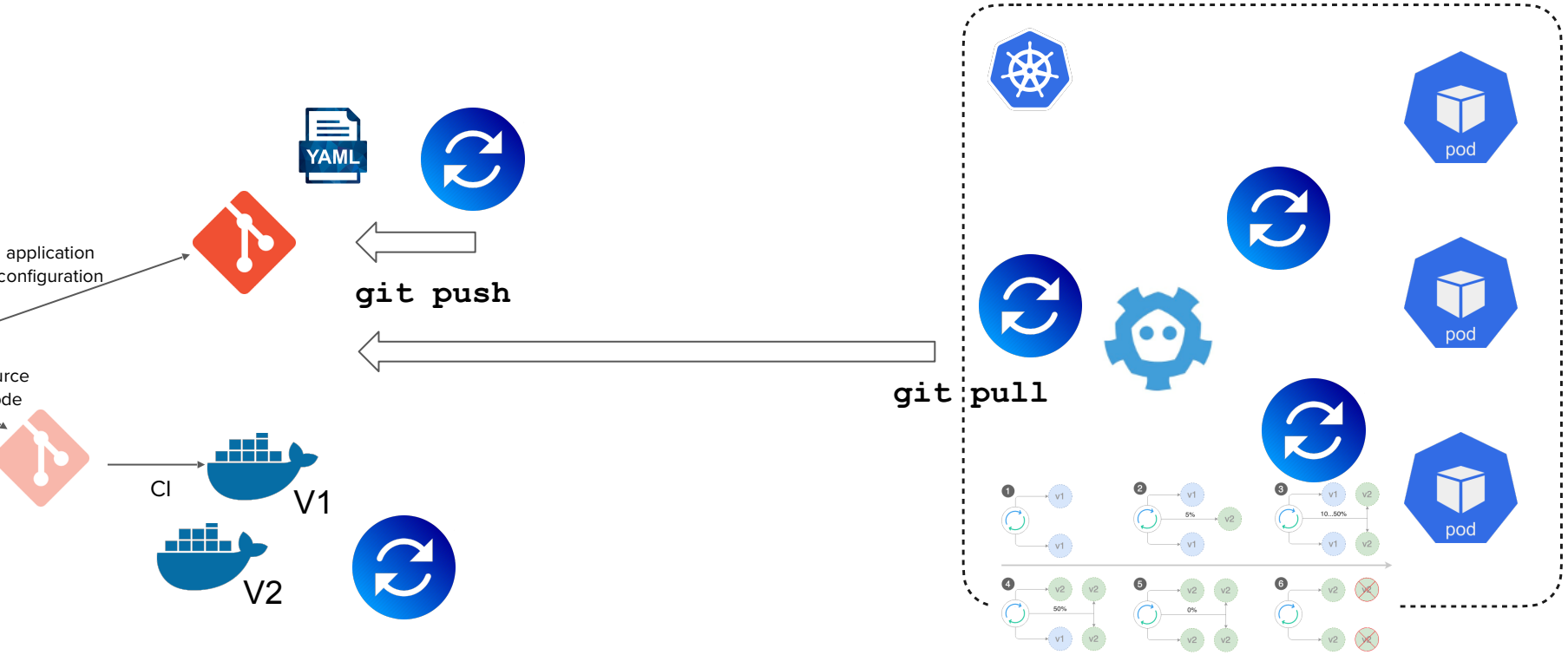
Flagger can run automated application analysis, promotion and rollback for the following deployment strategies:

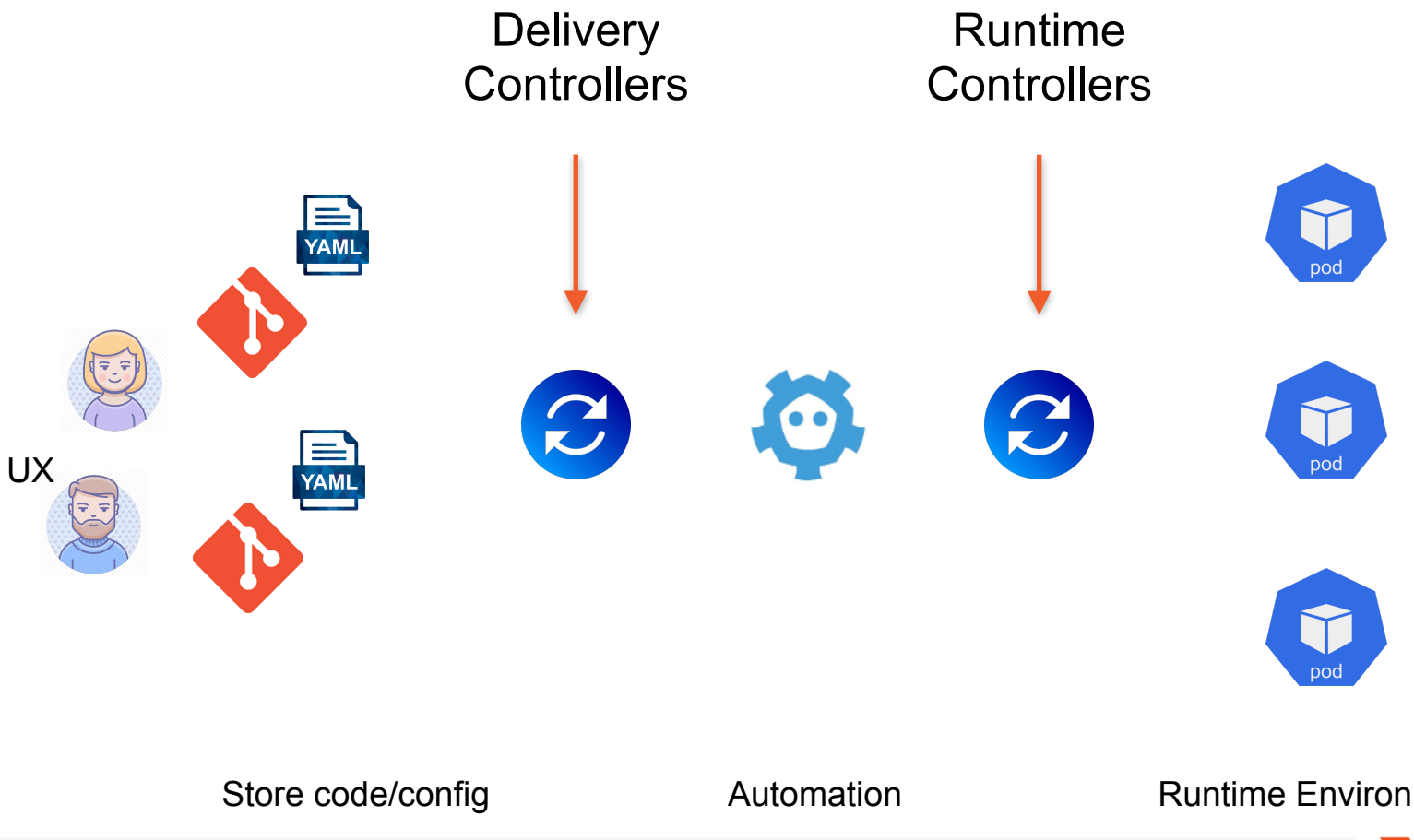
- Canary (progressive traffic shifting)
  - Istio, Linkerd, App Mesh, NGINX, Gloo
- A/B Testing (HTTP headers and cookies traffic routing)
  - Istio, NGINX
- Blue/Green (traffic switch)
  - Kubernetes CNI

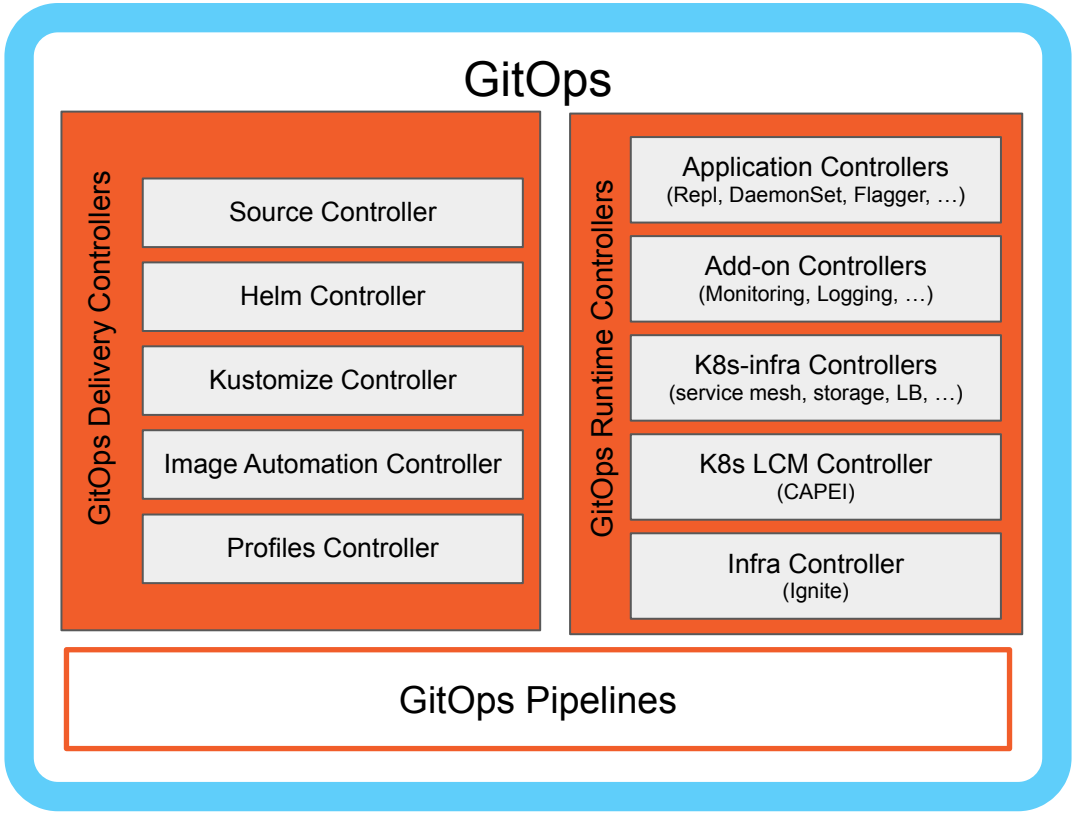
https://github.com/weaveworks/flagger

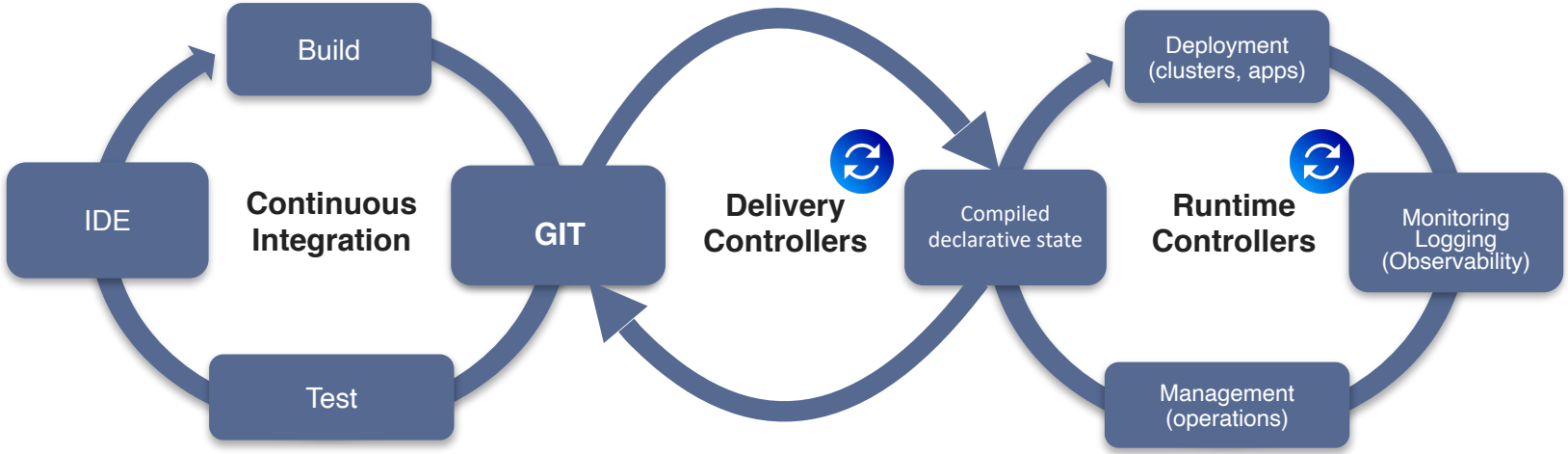
Whew

This is complicated…

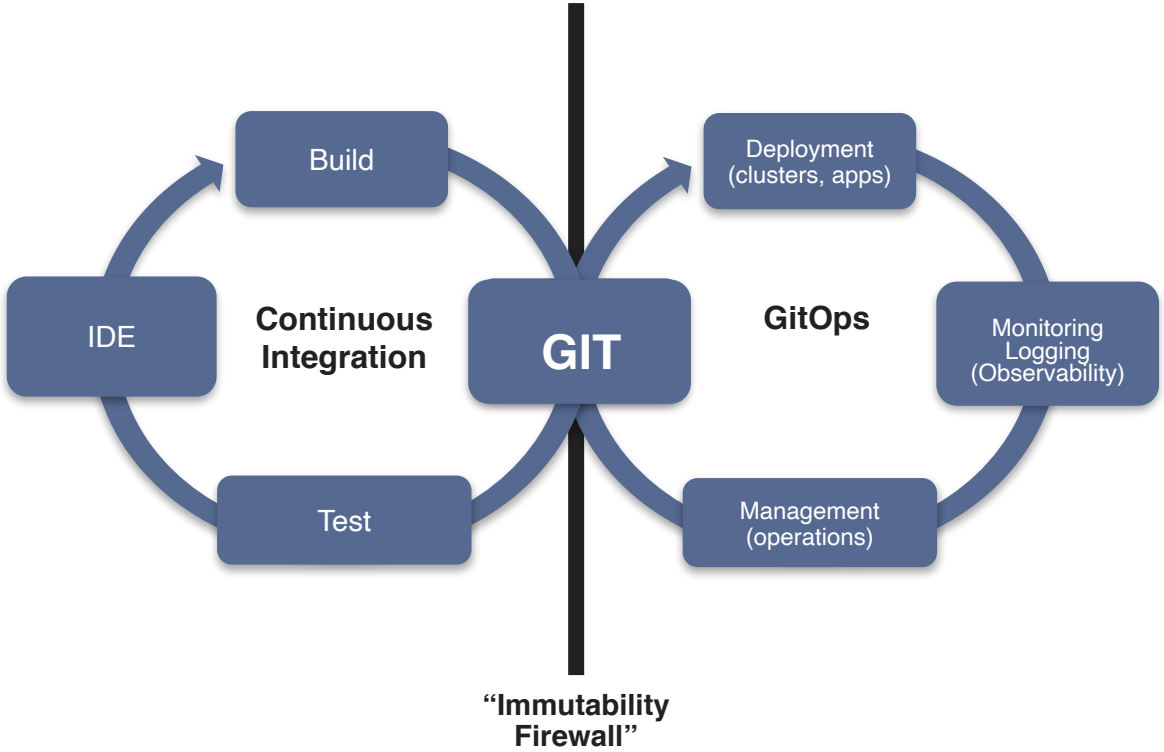



Delivery Controllers
Runtime Controllers
YAML
YAML
UX
pod
pod
pod
Store code/config
Automation
Runtime Environment


GitOps
GitOps Delivery Controllers
Source Controller
Helm Controller
Kustomize Controller
Image Automation Controller
Profiles Controller
GitOps Runtime Controllers
Application Controllers
(Repl, DaemonSet, Flagger, …)
Add-on Controllers
(Monitoring, Logging, …)
K8s-infra Controllers
(service mesh, storage, LB, …)
K8s LCM Controller
(CAPEI)
Infra Controller
(Ignite)
GitOps Pipelines


Build
IDE
Continuous Integration
GIT
Test
Delivery Controllers
Compiled declarative state
Deployment (clusters, apps)
Runtime Controllers
Monitoring Logging (Observability)
Management (operations)

GitOps – An Operating Model for Cloud Native



Unifying Deployment, Monitoring and Management.

Git as the single source of truth of a system's desired state

ALL intended operations are committed by pull request

ALL diffs between intended and observed state with automatic convergence

ALL changes are observable, verifiable and auditable

GitOps
=
Continuous Delivery
+
Continuous Operations

GitOps Principles

The entire system is described **declaratively**

The canonical desired system state is **versioned** in git

Approved changes can be **automatically applied** to the system

Software agents ensure correctness and perform actions on divergence in a closed loop

GitOps Patterns Review

- CD is separate from CI
- Pull configuration
- Drift detection and remediation
- Image Update Automation
- Environment Customization
- Progressive Delivery
- …

THANK YOU!