GitOpsCon EU 2025 (Virtual)

Let’s Talk GitOps Reliability

Kingdon Barrett

Navteca

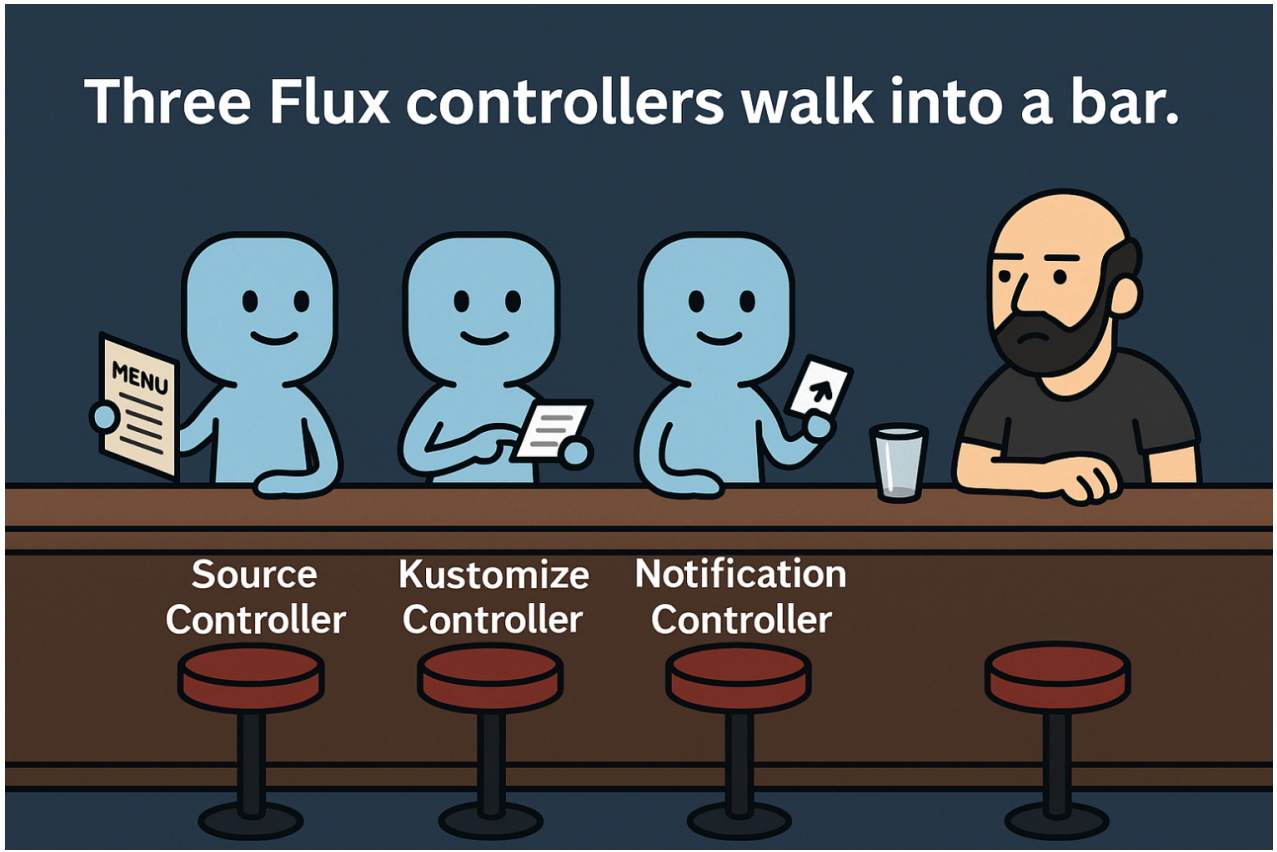
2025-05-28

Push, Wait, … Nothing?

Agenda

- Story
- Joke
- Anecdote
- Talk content
- Image Update Automation

Joke


Three Flux controllers walk into a bar.
MENU
Source Controller
Kustomize Controller
Notification Controller

Slides

Slides are Markdown, with Rabbit-Shocker (Ruby)

- Sorry - Lots of slides, I might go too fast (https://github.com/kingdonb/gitopscon2025-slides) or run out of time!

Welcome to clone & follow along during the talk, read ahead, etc.

Agenda (pt. 2)

- Flux 2.6
- Roadmap (fluxcd.io/roadmap)
- What is Flux
- What resources are in Flux?
- What is a good Flux experience like

Different ways to arrive at a good experience

Agenda (pt. 3)

- So, you followed my advice, and something went wrong
- Now what?
- How not to have a bad experience

The Setup

- Just “Git Push” and your changes go in
- Well there’s a little more to it than that,
- We might be waiting for some time…

Two options

Two options

- Minimal installation vs. Tricked Out
- **RICE**:
  Race Inspired Cosmetic Enhancement

RICE

Two options

- It’s not “ricing” it’s just properly configured
- **RICE**:
  Race Inspired Cosmetic Enhancement
- Goal is for devs to be able to move fast

Protip

- How do you know if your change went in?
- `flux get kustomizations`
- `flux get sources git`
- `flux get ...` install a UI
- Give up (┛ಠ_ಠ)┛彡┻━┻

Protip

- Give up (┛ಠ_ಠ)┛彡┻━┻
- Or, just set up Notifications! fluxcd.io/flux/monitoring/alerts/

Instant Feedback

Instant Feedback

- Know right away when something went wrong - shorten the feedback loop
- **Key:** instant
- Delays impose cognitive overhead
- If you thought polling was wasteful before, …

Instant Feedback

- If you thought polling was wasteful before, try polling by hand!
- `flux reconcile ...` is an anti-pattern
- Delays, context switches impose **cognitive overhead**,
- leads to overload, eventually burnout
- Notifications & Receivers **help devs**

Alerts

- We all need a little help sometimes
- We won't cover setting up alerts today
- youtu.be/zrSUjAXF0xA - KubeCon Talk about Flux Obsv. by Somtochi in 2023
- (But do set up alerts though!)
  20+ supported providers
- Teams, Mattermost, Discord, Telegram, …
  many others (get notified where you are!)

Shorten Feedback Loops

- Notifications & Receivers help devs to **iterate fast** with no extra steps!
- **Surprise and Delight** devs with good DX
- Have the joyful experience, don’t skip this
- Avoid pain/misery and **inevitable RSI**

Two options

1. If you don’t care about performance

- Maybe you are running Flux in Kind
- On your laptop, this is good
- (This is not the way in Production)
- or in any permanent environment
- which devs are using actively

Two options

1. Don’t care about perf (only speed)

- is that a contradiction?
  …how bad can performance actually be?

We can sacrifice some:

- CPU
- RAM
- Bandwidth - to avoid configuration hassle

Two options

1. ~~Don't care about perf (only speed)~~
   these things do have a cost,
   Using Cloud, it adds up quickly!
2. Let’s do this right

- First, some concrete examples:

YAML becomes stale

A problem that Flux solves:

```
  spec:
    template:
      spec:
        containers:
        - name: sintache
          image: img.hephy.pro/examples/sinatra-mustache:b22.20220712.042140
```

When you write this image tag in your YAML, how does it get updated?

YAML becomes stale

Flux solution:

```yaml
apiVersion: pkg.crossplane.io/v1
kind: Provider
metadata:
  name: provider-aws
spec:
  package: host/repo/provider-aws:v0.52.6 # {"$imagepolicy": "flux-system:provider-aws"}
  controllerConfigRef:
    name: provider-aws-config
```

Add an ImagePolicy marker
(lets Flux know where to update the image tag.)

ImagePolicy definition

```yaml
apiVersion: image.toolkit.fluxcd.io/v1beta2
kind: ImagePolicy
metadata:
  name: provider-aws
  namespace: flux-system
spec:
  imageRepositoryRef:
    name: provider-aws
  policy:
    semver:
      range: 0.x # (Among other sorting strategies: Alphabetical, Numerical)
```

Flux knows SemVer.
Recommend SemVer for production targets.

ImageRepository

```yaml
apiVersion: image.toolkit.fluxcd.io/v1beta2
kind: ImageRepository
metadata:
  name: provider-aws
  namespace: flux-system
spec:
  image: xpkg.upbound.io/crossplane-contrib/provider-aws # Scans & mirrors Image Tag lists from here
  interval: 30m
```

Interval

30 minutes is fine when it’s externally sourced

```
spec:
  interval: 30m
```

Interval

Active Dev should not wait longer than 5m

```
spec:
  interval: 2m
```

Interval

Faster the better - but polling has limits!

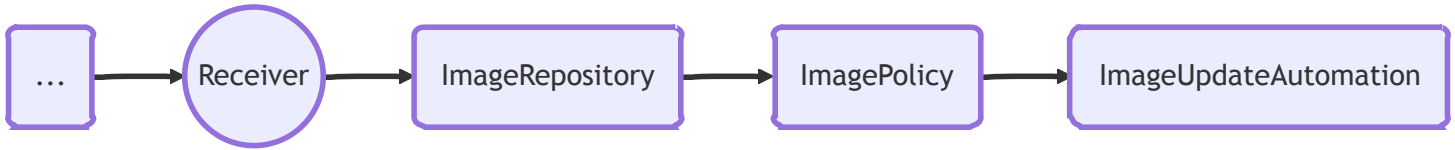
```
spec:
  interval: 10s
```

- Protip: don’t do this when perf matters (cost)
- Set up a `Receiver` instead!

Subscription

Subscription?

Receiver’s Role



ImageUpdateAutomation

```yaml
apiVersion: image.toolkit.fluxcd.io/v1beta2
kind: ImageUpdateAutomation
metadata:
  name: xplane-updater
  namespace: flux-system
spec:
  interval: 30m
  sourceRef:
    kind: GitRepository
    name: image-auto-gitrepo # This repository's secretRef deploy key has write access
  git:
    checkout:
      ref:
        branch: main
    commit:
      author:
        email: fluxcdbot@git.your.company.com
        name: fluxcdbot
      messageTemplate: |
        Crossplane Providers: Automated image update

        Automation name: {{ .AutomationObject }}
```

ImageUpdateAutomation (pt. 2)

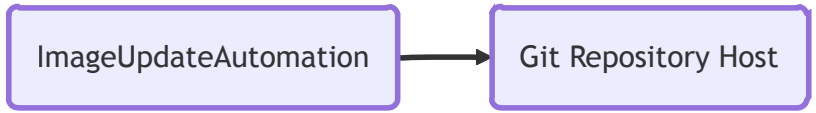
```
...
      messageTemplate: |
        Crossplane Providers: Automated image update

        Automation name: {{ .AutomationObject }}

        ```
        Files:
        {{ range $filename, $_ := .Changed.FileChanges -}}
        - {{ $filename }}
        {{ end -}}

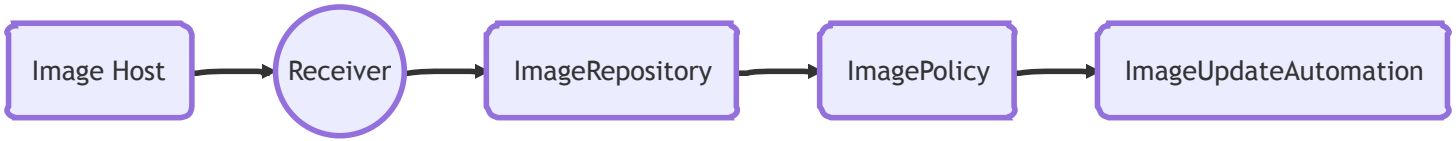
        Objects:
        {{ range $resource, $changes := .Changed.Objects -}}
        - {{ $resource.Kind }} {{ $resource.Name }}
          Changes:
        {{- range $_, $change := $changes }}
            - {{ $change.OldValue }} -> {{ $change.NewValue }}
        {{ end -}}
        {{ end -}}
        ```
    push:
      branch: xplane-updater
      options:
        merge_request.create: ""   # special options for GitLab
        merge_request.target: main
  update:
    path: ./deploy/staging/crossplane-providers
    strategy: Setters
```

Outside the Cluster

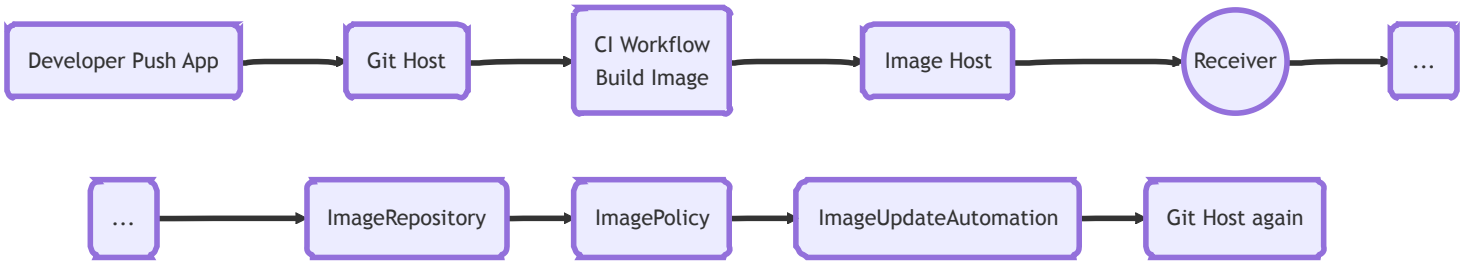


one of
github, gitlab,
gitea, bitbucket,
azure devops…

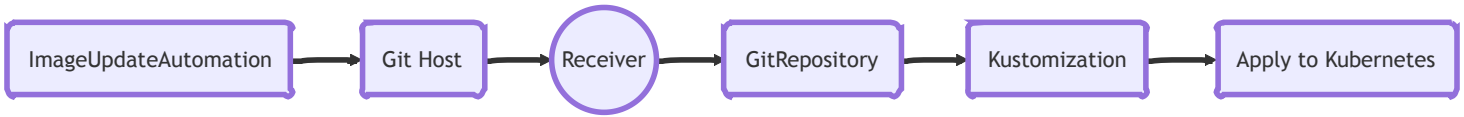
Subscription



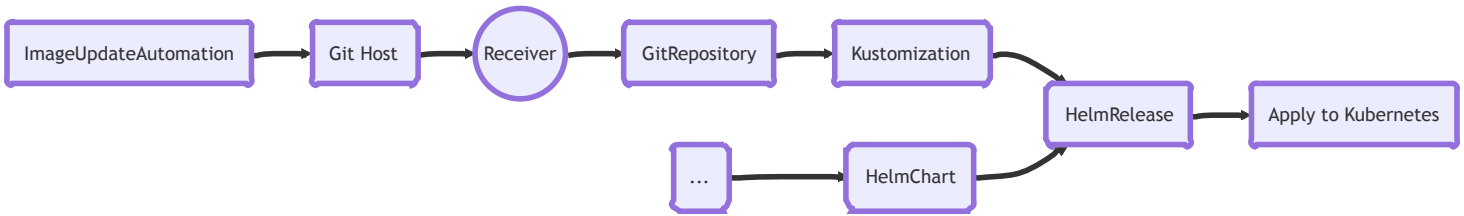
Subscription



Subscription



Subscription



Watchers

There are broadly two types of resources:

1. Connects to outside (`GitRepository`, `OCIRepository`, `Bucket`, `HelmChart`, `HelmRepository`)

- Let’s call them sources

Watchers

- except not all of these are in Source API (`ImageRepository`, ...)
- Outside of Kubernetes, automatic subscription is not possible
- (well, GitLab Agent manages it, but Flux natively does not support this capability)

GitLab user?

- Install GitLab Agent, then have a soda!

Easy, single command to get this working
Subscribe every GitRepository automatically

- …but most of us are on GitHub
- I am on GitLab, I couldn’t get this working
- Something wrong with my ELB, doesn’t support websocket connection upgrade

Watchers

Back to native Flux solutions that are available for practically every Source kind

- Even unsupported Git & Image Repos can
  use `Generic` type of `Receiver`
  or `Generic+HMAC` type for better security

Watchers

Two primary Flux types

1. Source: Connects to outside (Git, …)
2. Applier: Refers to another Flux resource (upstream) and applies sth. to the cluster

Sources and **Appliers** transact via **Artifacts**

- Appliers can be automatically subscribed to their source, supported by Kubernetes

Watchers

1. Connects to outside (`GitRepos`, ...)
2. Connects to another Flux resource
3. Resources with no reconciler

Third kind

- Resources with no reconciler (`Alert`, `ImagePolicy`, ...)
- Roadmap shout out: Flux 2.6 adds long-awaited digest hash pinning [image-reflector-controller#368]
  so you can use `:latest` in `ImageUpdateAutomation`!

Third kind

- (so now `ImagePolicy` **will have** a reconciler!)
- `Alert` is not reconciled
- Most Flux resources **are** reconciled
- All reconcilers have a **sync interval**

Question

Experiment:

So if most subscriptions are automatic and
for those that aren't,
for resources outside the cluster…
I set up `Receivers` and Webhooks

```
spec.interval: 30m
```

Question

Experiment:

Then why not just

```
spec.interval: 30m
```

Question

Experiment:

Then why not just

```
spec.interval: 3000m
```

Talk Content

What could possibly go wrong?

Receivers

How do they work?

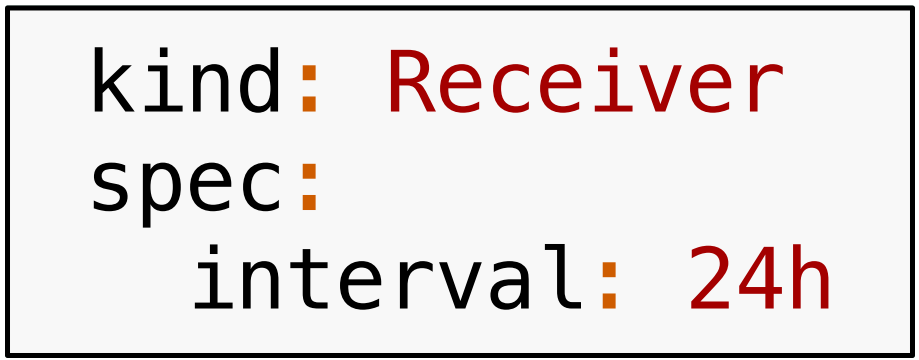
fluxcd.io/flux/guides/webhook-receivers/

Receiver	Type	Supports filtering using Events
Generic webhook	`generic`	❌
Generic webhook with HMAC	`generic-hmac`	❌
GitHub	`github`	✅
Gitea	`github`	✅
GitLab	`gitlab`	✅
Bitbucket server	`bitbucket`	✅
Harbor	`harbor`	❌
DockerHub	`dockerhub`	❌
Quay	`quay`	❌
Nexus	`nexus`	❌
Azure Container Registry	`acr`	❌
Google Container Registry	`gcr`	❌
CDEvents	`cdevents`	✅

Receiver Example

```yaml
apiVersion: notification.toolkit.fluxcd.io/v1
kind: Receiver
metadata:
  name: flux-system
  namespace: flux-system
spec:
  interval: 24h
  type: gitlab
  events:
    - "Push Hook"
    - "Tag Push Hook"
  secretRef:
    name: webhook-token
  resources:
    - kind: GitRepository
      name: flux-system
```

Receiver Example



What does that do?

It: reconciles the Receiver with its `Secret` ref

Receiver Example

```
$ kubectl get receivers
NAMESPACE     NAME          AGE   READY
    STATUS
flux-system   flux-system   60m   True
    Receiver initialized for path: /hook/0484d42e9aabbc852430a12d4c9df4e9abf0fad60c439424b9a7ff46ff9da9e9
flux-system   team-a        60m   True
    Receiver initialized for path: /hook/f70c568a6cd6d8bf7627139108c88e13bc813f952acba48425b0c16dbf2280b7
```

- Hook URL is a hash generated based on your `secretRef` token
- It should be stable until the secret is rotated

What can go wrong?

- `Receiver` Secret might not match!
- Webhook might not be configured at all
- Config can be deleted
- Protip: build Receivers and Webhooks **using IaC** like Terraform, Pulumi, Crossplane for repeatable experience

What can go wrong?

- Protip: use External Secrets controller (now supports 1password! and AWS Secrets Manager, and many others)
- Great combination with Crossplane and other IaC tooling
- Easy secret rotation for compliance purposes

⚠️ Corrected Clarification: Who Should Receivers Target?

- ✅ **Receivers should target Source resources**, *not Appliers (like Kustomizations or HelmReleases)*.
- 🎯 This is because:
  - Flux Appliers (Kustomize, Helm controllers) **watch their sources**.
  - A reconcile on the source (e.g., `GitRepository`, `OCIRepository`, `HelmRepository`) **triggers** dependent Appliers *automatically*, via Kubernetes watch notifications.
  - **You don't need to hit the Applier.** Doing so risks ordering problems and wasted reconciles.
  - Targeting both Source and Applier with a Receiver can cause **race conditions** and **non-deterministic behavior** (e.g., Applier sees source as not ready, does nothing).
- 🔄 If you *must* reconcile an Applier directly (e.g., you're manually fixing drift, or testing that's fine—but **don't do it from a webhook Receiver.**
- ⚙️ Flux is built on top of Kubernetes controller-runtime. It uses **event-driven reconciliation**, so triggering the Source is sufficient. No polling required if set up properly.

What is this?

Go home GPT, you're drunk!

What can go wrong? Receiver

Lots can go wrong
even with correctly configured Receiver

- **User Error** is the most common error
- **Solar flare-induced errors** are exceedingly rare by comparison

How to detect config-related failure

POST to Probe `Receiver` endpoint periodically with a synthetic payload

- **Wait a second** isn't that just polling with way more steps? **Yeah…** it is that.
- Probably don't need to monitor this, your devs will let you know it's broken
- But what if I am dev? **Monitor Everything**

How to detect ingress failure

Probe `Receiver` endpoint periodically

- **You could** hit the `/hook/...` URL with a GET request, no payload
  `400` response is good news
- This will not trigger the `Receiver`, but it will likely catch the 80% of failure modes

⚠️ But What Can Actually Go Wrong?

Here's your segue into the long-form detective arc — these are the key failure modes to unpack:

1. Receiver misconfiguration

- ❌ Wrong secret or webhook payload not signed correctly → **request rejected**
- ❌ Target set incorrectly (e.g., not pointing at the `GitRepository`) → **reconcile has no effect**

2. Webhook never arrives

- ❌ Git provider can't reach Receiver (firewall, no ingress, etc.)
- ❌ Git provider is misconfigured or webhook is missing entirely

3. Source doesn't update

- ❌ Commit doesn't change the directory being watched (wrong `path`)
- ❌ Branch or revision selector doesn't match the commit
- ❌ Git repository has a transient auth or fetch error

4. Applier doesn't act

- ❌ Depends on polling interval or a secondary reconcile (if Receiver only hit Source)
- ❌ `dependsOn` misconfiguration, or resource ordering problems

5. Kubernetes rejects your change

- ❌ Invalid YAML, CRD not installed, permissions issues
- ❌ You forgot a `git add`, pushed the wrong thing

How to detect failure

- But what if I am **the only** dev? **Monitor Everything** especially then
- It could be **days** until you notice it's broken
- Who knows if you will **remember what** you changed (remember: shorten feedback loops! **avoid burnout**)

What can go wrong? Notifications

- Monitor logs to detect when notifications are failing to fire
- If your notifications are down, **how** do you get notified about that?
- **Curate and use a secondary system**
- CloudWatch alerts don’t flow through Flux

What can go wrong? Notifications

- I'm using **EKS Auto Mode**, it deletes my nodes every day & then Flux spews errors, rendering alerts significantly **less useful**
- How do I get Flux Alerts that are **actually useful**, not noisy?
- That is tough, not enough time today

How do I get useful alerts?

- …but: Karpenter generates K8s events **Monitor those events too**
- You can even route them through Flux Alerts, but this is **advanced wizardry**
- I built **`flux-event-relay`** to send **Karpenter** `NodeClaim` events to Flux
- My **work for hire** (not published as of yet) but if enough people ask me about it I will

Thanks

Questions? Meet me **@KingdonB** on the CNCF Slack, Kubernetes Slack, or find me at **Flux Bug Scrub**

_

Calendar: fluxcd.io/community#meetings

_

Join the Flux Dev meetings
(We're friendly folks, inclusive community!)