KubeCon | CloudNativeCon

Europe 2020

Virtual

Flux Deep Dive

The road to “Flux v2” and Progressive Delivery

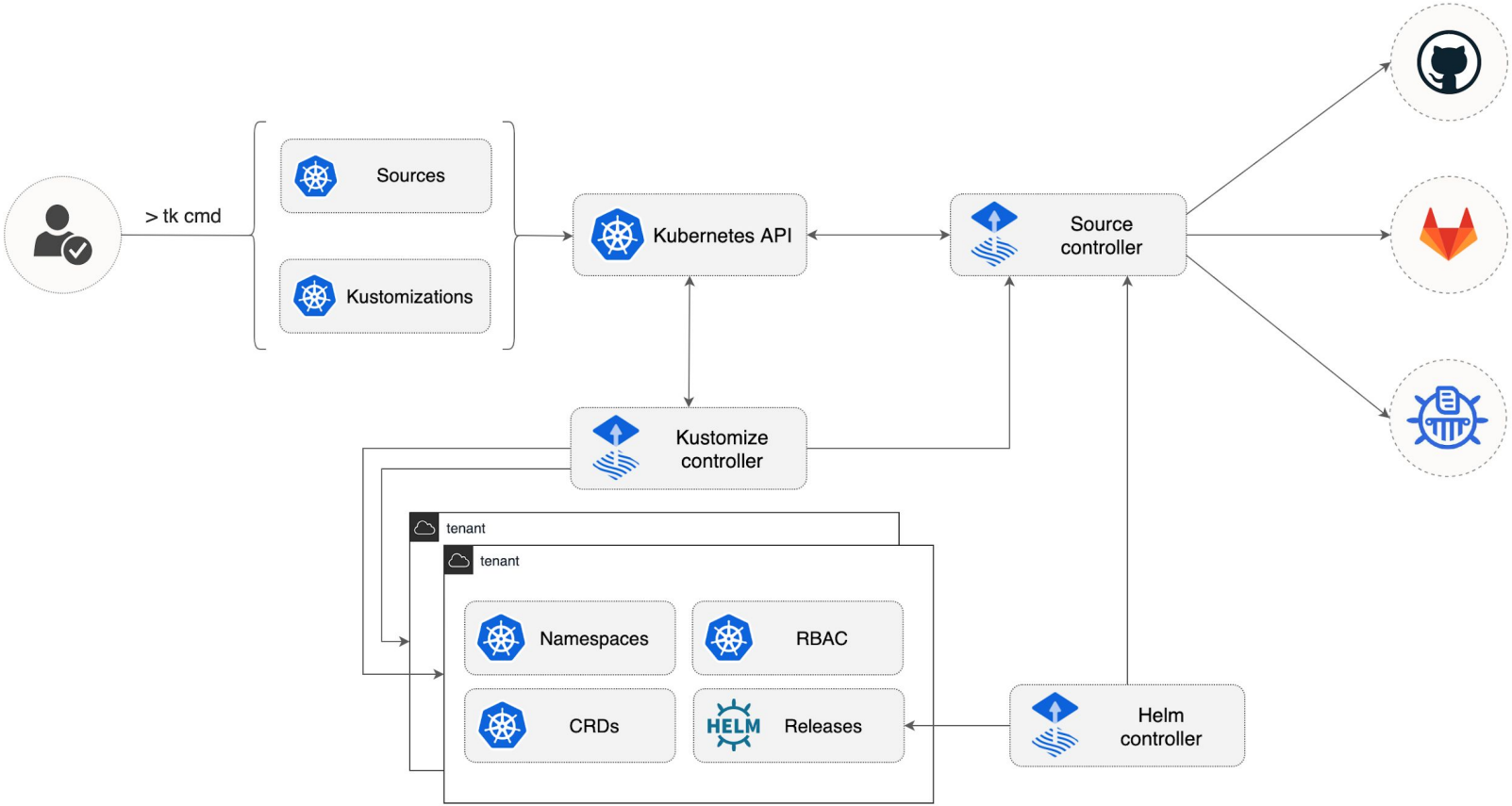
Stefan Prodan & Hidde Beydals

Flux most wanted

Top feature requests

1. Multi-repository support
2. Operational insight through health checks, events and alerts
3. Multi-tenancy capabilities

“Flux v2” overview



Introducing the GitOps Toolkit

The GitOps Toolkit is a set of **composable APIs** and **specialized tools** that can be used to build a Continuous Delivery platform on top of Kubernetes.

These tools are built with Kubernetes controller-runtime libraries and they can be **dynamically configured** with Kubernetes **custom resources** either by cluster admins or by other automated tools.

The GitOps Toolkit components interact with each other via **Kubernetes events** and are responsible for the reconciliation of their designated API objects.

GitOps Toolkit website: https://toolkit.fluxcd.io

tk: the GitOps Toolkit CLI

The GitOps toolkit CLI utility allows cluster admins to configure the toolkit and assemble CD pipelines without having to write tomes of YAML

- Seamlessly integrates with Git providers like GitHub and GitLab
- Deploy keys provisioning for Git sources (SSH and token based auth)
- Install/upgrade/check/uninstall operations for the toolkit components
- Scaffolding and CRUD operations for the toolkit custom resources

tk documentation: https://toolkit.fluxcd.io/cmd/tk/

Toolkit component driven “Flux v2”

Flux v1 is a monolithic do-it-all operator, the GitOps Toolkit separates the functionalities into **specialized controllers**.

Flux v2 will be a curated set of configuration for the GitOps Toolkit which you can simply consume using the `tk` command. Limiting the use of features and/or adding extensions on top of Flux has never been this easy.

```sh
# Flux v2
tk install --namespace=flux \
--components=source-controller,kustomize-controller

# Flux Helm v2
tk install --namespace=flux \
--components=source-controller,kustomize-controller,helm-controller
```

Source Controller

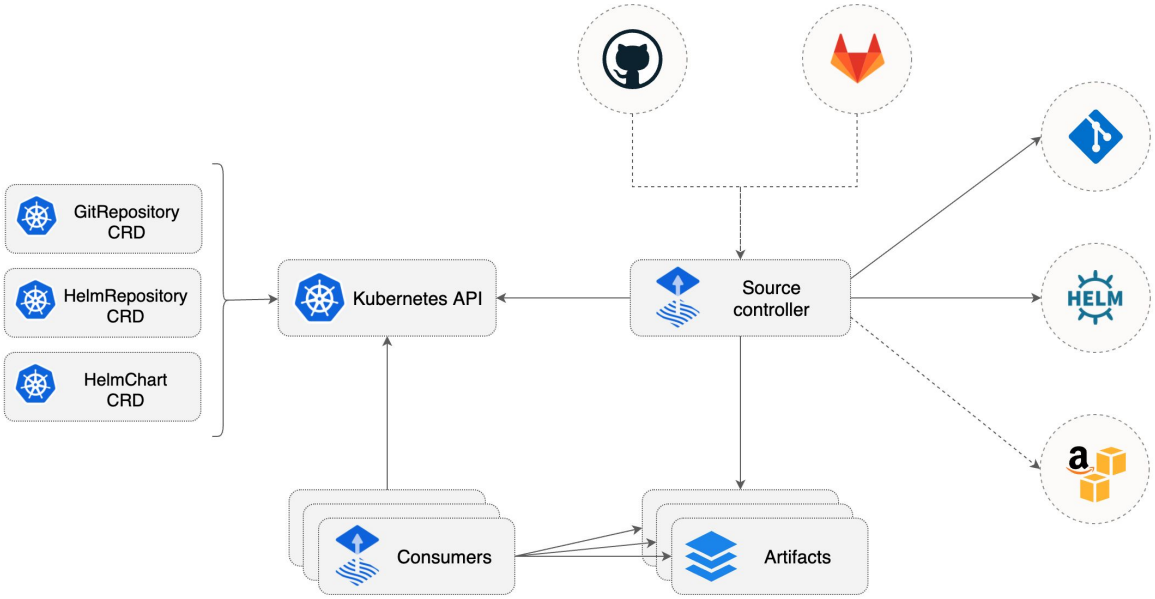
The main role of the **source management** component is to provide a common interface for artifacts acquisition. The source API defines a set of Kubernetes objects that cluster admins and various automated operators can interact with to offload the **Git and Helm repositories operations** to a dedicated controller.

Operations:

- Authentication and authenticity validation
- Event-based and on-a-schedule policy driven artifacts acquisition
- Produce immutable artifacts from sources

Overview: https://toolkit.fluxcd.io/components/kustomize/controller/
API Spec: https://toolkit.fluxcd.io/components/kustomize/kustomization/

Source Controller




```yaml
apiVersion: source.fluxcd.io/v1alpha1
kind: GitRepository
metadata:
  name: fleet-infra
  namespace: gitops-system
spec:
  url: ssh://git@github.com/gitopsrun/fleet-infra
  ref:
    semver: ">=1.0.0 <2.0.0"
  secretRef:
    name: fleet-infra-ssh-keys
  verify:
    mode: head
    secretRef:
      name: fleet-infra-pgp-keys
  interval: 1m
```

Specialized Reconcilers

The GitOps Toolkit allows specialized reconcilers to **collaborate** when declaring the **desired state** of a group of clusters:

- *kustomize-controller*
- *helm-controller*
- *image-update-controller (TBA)*
- *fleet-controller (TBA)*
- ...

All these controllers will be using the **Source** API package to acquire artifacts from the *source-controller* and subscribe to “source changes” events.

Reconciliation of Resources from Git

Flux v1

- Limited to reconciling resources from a single Git repository
- “Declarative Git configuration” via arguments in the Flux deployment, cloned and fetched by *fluxd*
- Only supports following the HEAD of Git branches
- Reconciliation can be suspended by downscaling the Flux deployment
- Credentials config via arguments and/or secrets volume mounts in the Flux pod

Toolkit component driven “Flux v2”

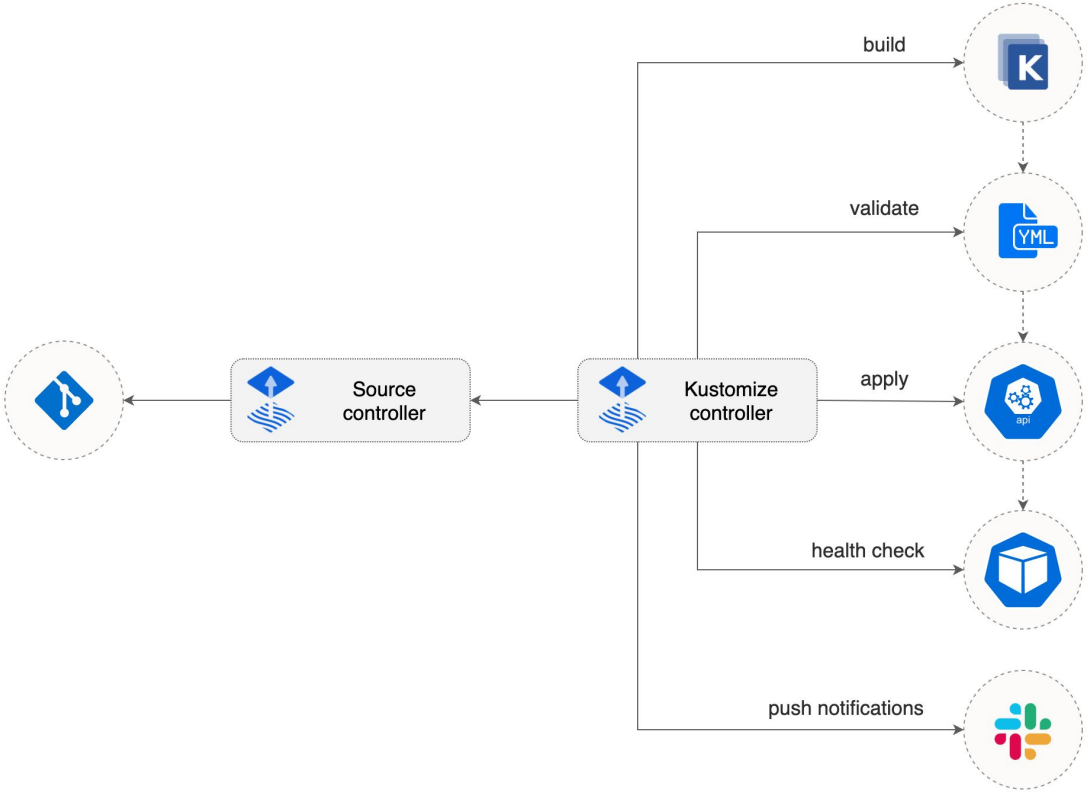
- Can reconcile resources from multiple Git repositories
- Declarative configuration through a **GitRepository** CR, producing an artifact to be reconciled by other controllers
- Supports **Git branches**, pinning on **commits and tags**, and following **SemVer** tag ranges
- Reconciliation can be paused per resource by suspending the GitRepository
- **Credentials config per GitRepository resource** (SSH private key, HTTP/S username/password/token, OpenPGP public keys)

Kustomize Controller

The *kustomize-controller* is a Kubernetes operator, specialized in running continuous delivery pipelines for infrastructure and workloads defined with Kubernetes manifests and assembled with Kustomize.

- Reconciles the cluster state from multiple sources
- Generates manifests with Kustomize from plain yamls or overlays
- Validates manifests against Kubernetes API
- Impersonates service accounts (multi-tenancy RBAC)
- Health assessment of the deployed workloads
- Runs pipelines in a specific order (depends-on relationship)
- Prunes objects removed from source (garbage collection)
- Reports cluster state changes

Kustomize Controller




```yaml
apiVersion: kustomize.fluxcd.io/v1alpha1
kind: Kustomization
metadata:
  name: frontend
spec:
  dependsOn:
    - istio
  sourceRef:
    kind: GitRepository
    name: webapp
  path: "./webapp/frontend/"
  prune: true
  interval: 5m
  healthChecks:
    - kind: Deployment
      name: frontend
      namespace: webapp
  timeout: 2m
```

Kustomize support

Flux v1

- “Declarative configuration” through *.flux.yaml* files in the Git repository
- Manifests are generated via shell exec and then reconciled by *fluxd*
- Reconciliation using the service account of the Flux deployment
- Garbage collection needs a cluster role binding for Flux to query the Kubernetes discovery API
- Support for custom commands and generators executed by *fluxd* in a POSIX shell

Toolkit component driven “Flux v2”

- Declarative configuration through a **Kustomization** CR, consuming the produced artifact from the **GitRepository**
- Generation, server-side validation, and reconciliation is handled by a specialized *kustomize-controller*
- Support for **service account** impersonation
- Garbage collection needs no cluster role binding or access to Kubernetes discovery API
- No support for custom commands

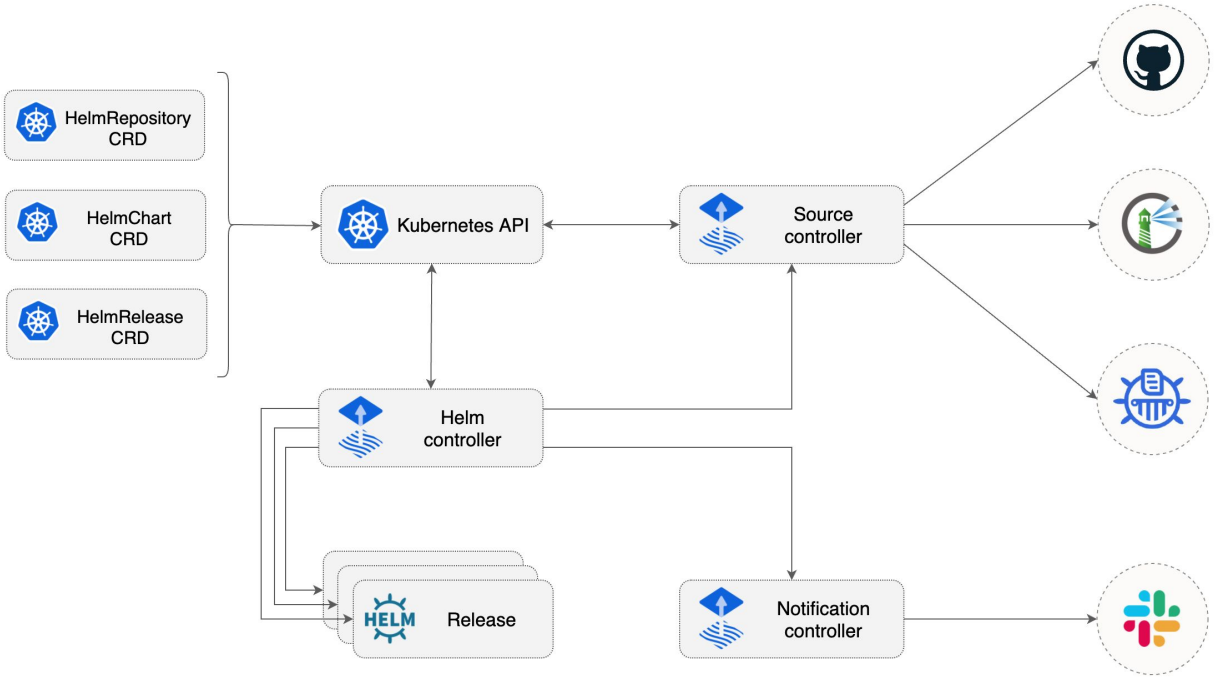
Helm Controller

The *helm-controller* is the GitOps Toolkit's component driven "Flux v2" Helm Operator, and performs Helm actions for **HelmRelease** resources using the **HelmChart** artifacts produced by the *source-controller*.

- Complete rewrite from scratch
- Offloaded Helm repository and chart reconciliation
- Improved **HelmRelease** API design
- Simplified operations model
- Helm storage drift detection without performing dry-run comparisons

Overview: https://toolkit.fluxcd.io/components/helm/controller/
API Spec: https://toolkit.fluxcd.io/components/helm/helmreleases/

Helm Controller




```yaml
apiVersion: helm.fluxcd.io/v2alpha1
kind: HelmRelease
metadata:
  name: podinfo
spec:
  interval: 5m
  chart:
    name: podinfo
    version: '^4.0.0'
    sourceRef:
      kind: HelmRepository
      name: podinfo
    interval: 1m
  upgrade:
    remediation:
      strategy: rollback
  test:
    enable: true
    ignoreTestFailures: true
  valuesFrom:
  - kind: ConfigMap
    name: podinfo-ingress
  values:
    replicaCount: 2
```

Integration with Helm

Helm Operator v1

- Declarative configuration in a single **HelmRelease** CR
- Chart synchronization embedded in the operator
- Support for fixed SemVer versions from Helm repositories
- Git repository synchronization on a global interval
- Limited observability via the status object of the HelmRelease resource
- Resource heavy, relatively slow

Toolkit component driven *helm-controller*

- Declarative configuration through **HelmRepository**, **HelmChart** and **HelmRelease** CRs
- Extensive release configuration options, and a reconciliation interval per source
- Support for depends-on relationships between **HelmRelease** resources
- Support for **SemVer ranges** for HelmChart resources
- Planned support for charts from **GitRepository** sources
- Better observability via the HelmRelease status object, Kubernetes events, and notifications
- Better performance

Notification Controller

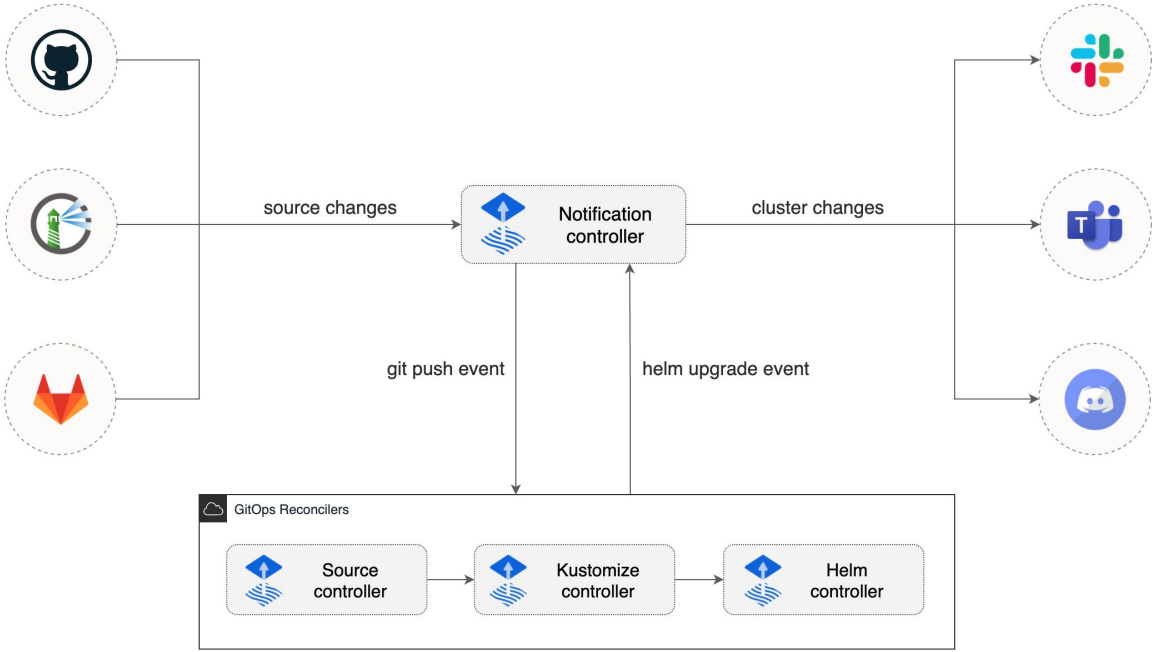
The *notification-controller* is specialized in handling inbound and outbound events.

The controller handles:

- Events coming from external systems (GitHub, GitLab, Bitbucket, Harbor, Jenkins, etc) and notifies the GitOps Toolkit controllers about source changes
- Events emitted by the GitOps Toolkit controllers, that are dispatched to external systems (Slack, Microsoft Teams, Discord, Rocket) based on event severity and involved objects

Overview: https://toolkit.fluxcd.io/components/notification/controller/
API Spec: https://toolkit.fluxcd.io/components/notification/event/

Notification Controller




```yaml
apiVersion: notification.fluxcd.io/v1alpha1
kind: Receiver
metadata:
  name: webapp
spec:
  type: github
  events:
    - "ping"
    - "push"
  secretRef:
    name: webhook-token
  resources:
    - kind: GitRepository
      name: webapp
    - kind: HelmRepository
      name: webapp
```

Notifications, webhooks and observability

Flux v1

- Emits “custom Flux events” to a webhook endpoint
- RPC endpoint can be configured to a 3rd party OSS solution like *FluxCloud* to be forwarded as notifications to e.g. Slack
- Limited incoming webhook functionalities
- Unstructured logging
- Custom Prometheus metrics

FluxCloud: https://github.com/justinbarrick/fluxcloud

FluxCloud (active fork): https://github.com/topfreegames/fluxcloud

Toolkit component driven “Flux v2”

- Emits **Kubernetes events** for all custom resources part of the Toolkit
- Toolkit components can be configured to POST the events to a *notification-controller* endpoint
- Selective forwarding of POSTed events as notifications using **Provider** and **Alert** CRs
- Extensive support for incoming webhooks for a wide range of platforms
- Structured logging for all components
- Generic / common *controller-runtime* Prometheus metrics

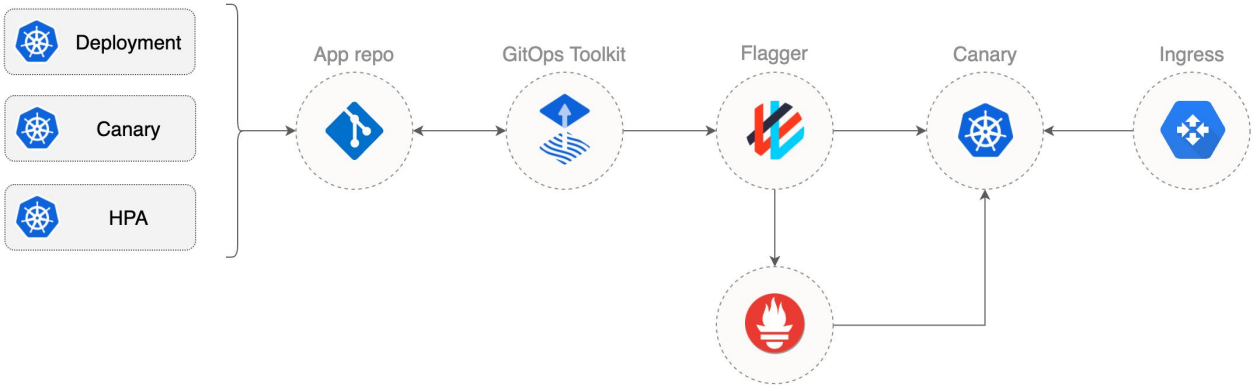
Advanced Deployment Strategies

Given the extensible nature of the GitOps Toolkit, we can **reduce the risk** of introducing a new software version in production by leveraging Flagger's progressive delivery strategies:

- Canary Release (progressive traffic shifting)
  - Applications that expose HTTP or gRPC APIs
- A/B Testing (HTTP headers and cookies traffic routing)
  - User-facing applications that need session affinity
- Blue/Green (traffic mirroring)
  - Idempotent APIs
- Blue/Green (traffic switch)
  - Stateful applications
  - Legacy applications

Flagger and Progressive Delivery

Flagger works out-of-the-box with the GitOps Toolkit.
In the future the toolkit reconcilers will be using Flagger's canary status to drive the release process across interdependent workloads.




```yaml
apiVersion: flagger.app/v1beta1
kind: Canary
metadata:
  name: podinfo
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: podinfo
  service:
    port: 9898
  analysis:
    interval: 5
    threshold: 1
    metrics:
    - name: request-success-rate
      thresholdRange:
        min: 99
      interval: 1m
```

Questions?

Join the GitOps Toolkit discussions on GitHub

https://github.com/fluxcd/toolkit/discussions