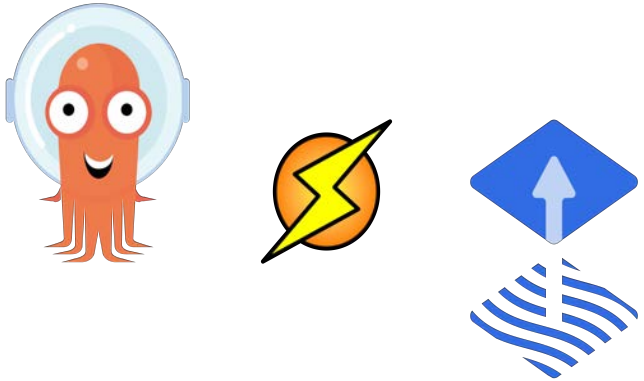


wescale

2023.01.26

ArgoCD or Flux ?

— A GitOps comparison

00

It’s not about the “best”

— Disclaimer

- We do not advocate for a frontal opposition between Flux & ArgoCD
- We target a broad understanding of the philosophy which drive both tools
- Both come with their own assets and tradeoffs
- We worked with both solutions. The vision we bear is subjective but aims to be objective
- Don’t hesitate to contact us on the Discord channel :)

00

On what ground ?

— Our comparison criteria

- In order to be the most relevant, we decided to have a wide array of criteria

Criteria	Description
Model	How the tool is thinking the GitOps paradigm
User Experience	Day1, Day2. How easy is it to stay aligned with the promises
Benchmarking	CPU/RAM Requirements, Scalability, elasticity, …

00

Two approaches which need to be clarified

— Why comparing Flux with ArgoCD ?

- Two majors implementations of GitOps in a Kube native context. In terms of
  - Community
  - Recognition
  - Adoption
- It’s a question we often see with our clients
- We haven’t seen yet a comparison that satisfied us
- And frankly, who never wondered ?



00

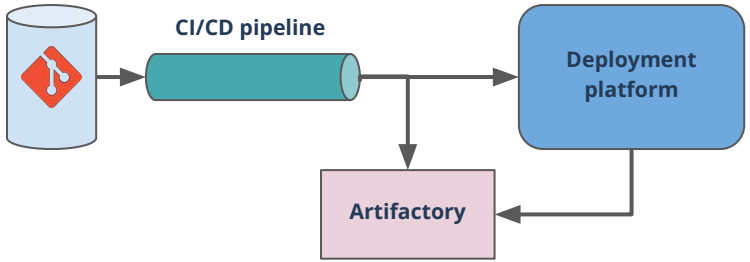
Prologue

A kind reminder: a CI AND a CD

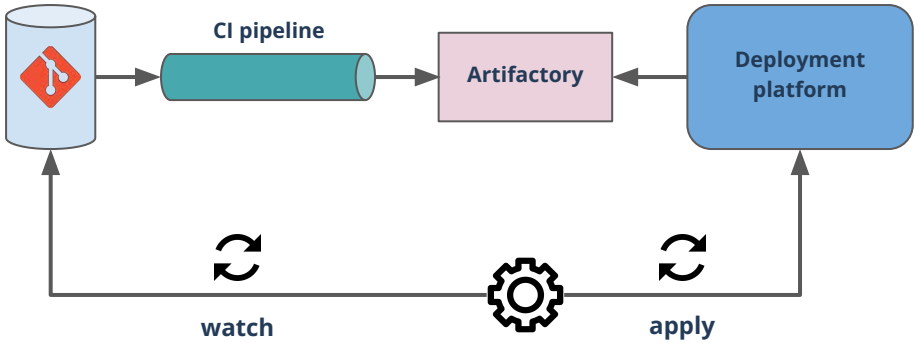
— GitOps in a nutshell

- GitOps is a continuous Deployment pattern which favors “pull” approach over the classical “push” one
- Not theoretically bounded to k8s but the latter is by design adapted to GitOps

CI/CD



CI **and** CD



00

Ok, so ?

— GitOps, what for ?

- No additional deployment tool to install in our pipeline
- No secret to store either
- Git as our source of truth for our platform infrastructure
- Capacity to remove access to the platform
- Deployment is purely declarative
- Possibility to use the Git provider permission to ease collaboration between dev & ops

01

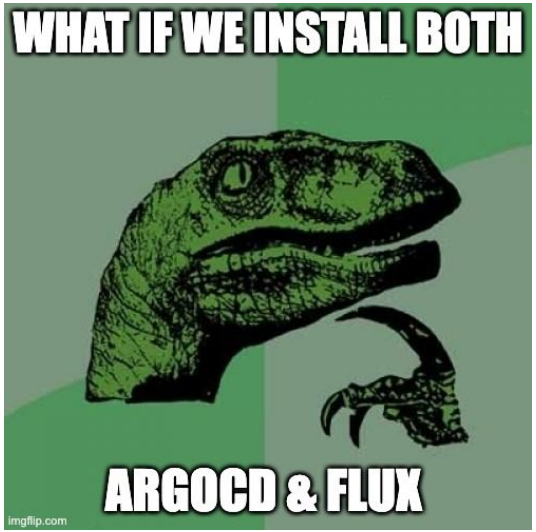
An overview

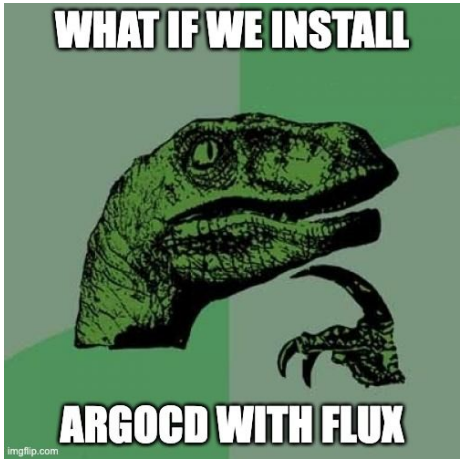
01 Few words — Flux and ArgoCD in a nutshell

	Flux	ArgoCD
One line presentation	Open and extensible continuous delivery solution for Kubernetes	Declarative continuous deployment for Kubernetes.
First commit datetime	FluxV1: 2016-07-07T12:32:02Z FluxV2: 2020-04-24T09:38:22Z	2018-02-15T00:53:07Z
Github stars	4.3k ⭐	11.8k ⭐
CNCF	Graduated 🎉 since November 2022	Graduated 🎉 since December 2022
Sponsor	WeaveWork, then donated to the CNCF	Intuit, then donated to the CNCF
Integration	Flagger, Weavework GitOps	ArgoProj

Not a good idea...

— Last but not least





02

Model

02

What is this all about ?

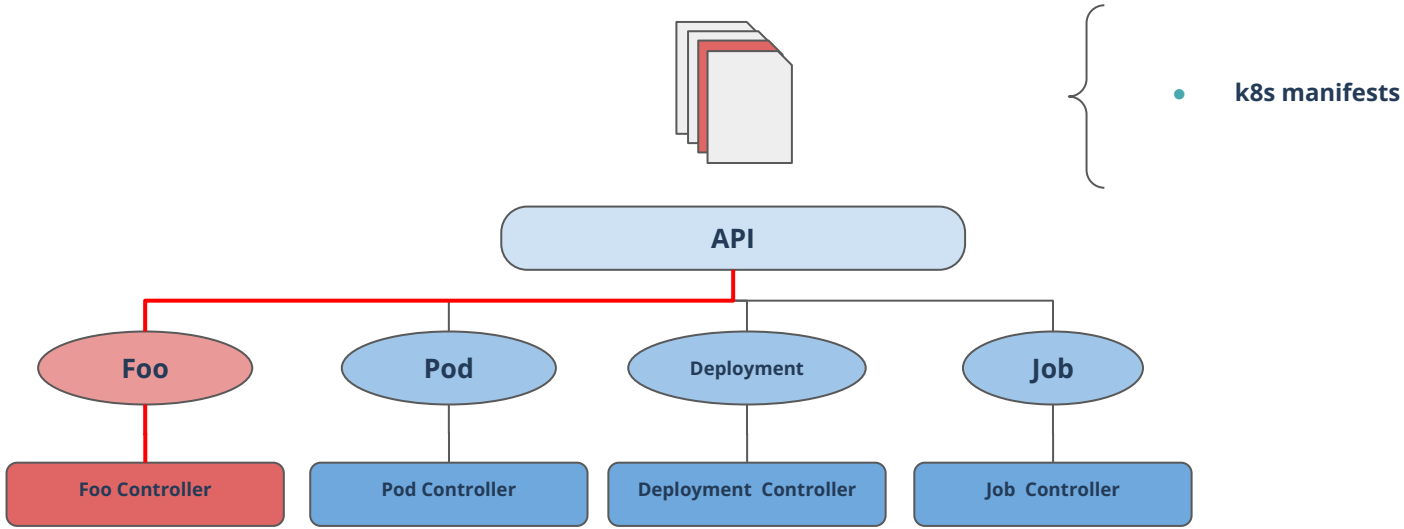
— GitOps model

- Flux & ArgoCD are both GitOps implementations
- Both comes with their own conceptions on how to understand GitOps
- Understanding the model i.e. the manipulated entities and their relationships is an important first step in comparing the tools.
- Given the maturity of the both tools, we think that model differences will weigh a lot in the decision choice between Flux & ArgoCD

Extending the Kubernetes API

— Operator pattern, kind reminder

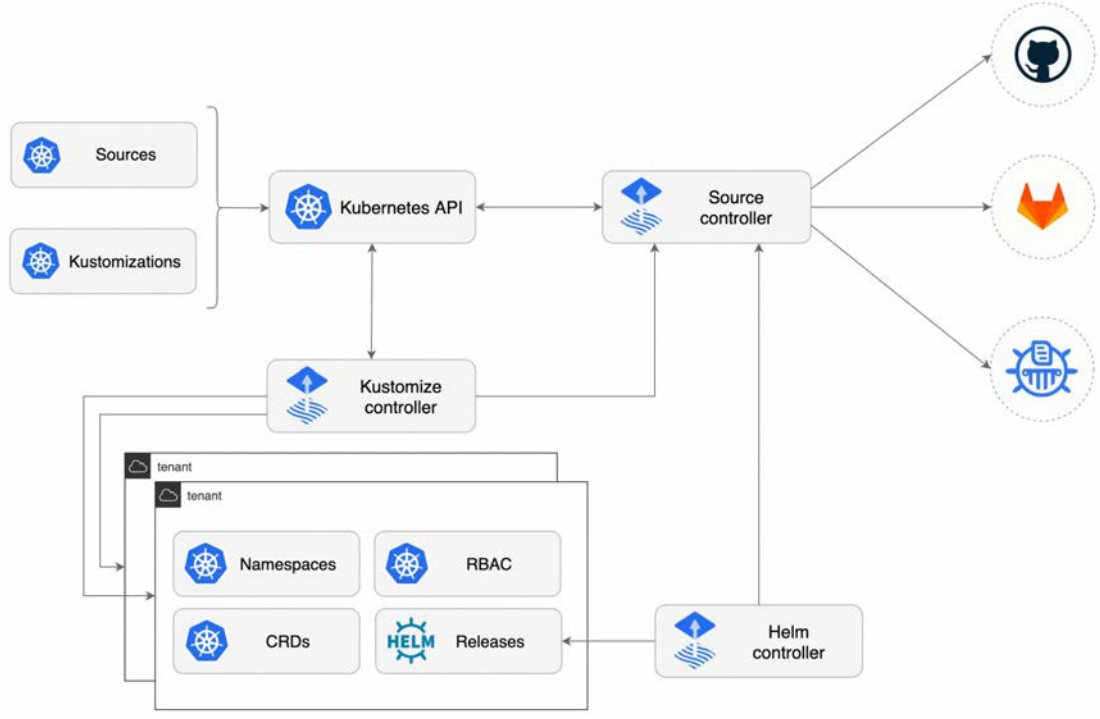
- *"An operator is a client of the Kubernetes API that acts as a controller for a Custom Resource."*
- We extend the k8s model by defining a CRD and its associated controller → k8s now understands your logic



FLUX

The GitOps Tool Kit (gotk)

— Flux entity model



02

Where to find artifacts

— Sources

- Represents an artifact storage where Flux would synchronize from
- Several types
  - **Git Repositories**
    Allow to define to flux your source of truth repositories. Central type to the GitOps paradigm

    Specialized source type to reference artifact sources needed for our Pod, Deployment, Job, …
  - Buckets
  - OCI Repositories
  - HelmCharts & Repositories

Extending the Kubernetes API

— Source examples

- Extracted from the official documentation

```yaml
apiVersion:
source.toolkit.fluxcd.io/v1beta2
kind: GitRepository
metadata:
 name: podinfo
 namespace: default
spec:
 interval: 5m0s
 url:
https://github.com/stefanprodan/pod
info
 ref:
   branch: master
```

```yaml
apiVersion:
source.toolkit.fluxcd.io/v1bet
a2
kind: HelmRepository
metadata:
 name: podinfo
 namespace: default
spec:
 interval: 5m0s
 url:
https://stefanprodan.github.io
/podinfo
```

```yaml
apiVersion:
source.toolkit.fluxcd.io/v1beta2
kind: HelmChart
metadata:
 name: podinfo
 namespace: default
spec:
 interval: 5m0s
 chart: podinfo
 reconcileStrategy: ChartVersion
 sourceRef:
   kind: HelmRepository
   name: podinfo
 version: '5.*'
```

02 Extending the Kubernetes API — Kustomization

- A wrapper entity around **Kustomize**

- **Kustomize** is an embedded tool with Kubernetes which enables configuration management through patches
  - No yaml copy/paste
  - No specific yaml version per branch (dev, staging, prod)
  - One base in a parse file, several patches to complete and adapt the base

- Kustomization is a Flux representation of a Kustomize *action*
  - Make sense in a semantic way "*I kustomize ⇔ A kustomization*"

- In the flux sens, a synchronisation is the result of at least one Git source and one Kustomization
  - Those two entities represent the big bang of the Flux GitOps approach
  - No {Source; Kustomization}, no synchronisation in the cluster

Extending the Kubernetes API

— Kustomization example

- Extracted from the official documentation

```yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
 name: podinfo
 namespace: default
spec:
 interval: 10m
 targetNamespace: default
 sourceRef:
   kind: GitRepository
   name: podinfo
 path: "./kustomize"
 prune: true # remove stale resources from cluster
```

- In the *path* spec folder, we expect to find a *kustomization.yaml* file (the base) which indicates to Flux which k8s entities to consider
- If absent it is automatically generated and Flux recursively consider all valid Yaml file in the same folder

02

Extending the Kubernetes API

— HelmRelease

- A wrapper entity around **an helm release**
- **Helm** is known to be a kind of Kubernetes package manager.
- It is a de-facto standard in the Kubernetes community
- You manipulate **YAML** through a templating language to adapt the content according a context
- Quite often compared with **Kustomize** which favor patches
- The genius idea of flux is to reconcile both approaches as a **Flux Kustomization** can target a **Flux Helm Release** to adapt its spec through patches and which in turn will render the release with the patched spec.
- It’s not Kustomize VS Helm, It’s Kustomize with Helm

Extending the Kubernetes API

HelmRelease example

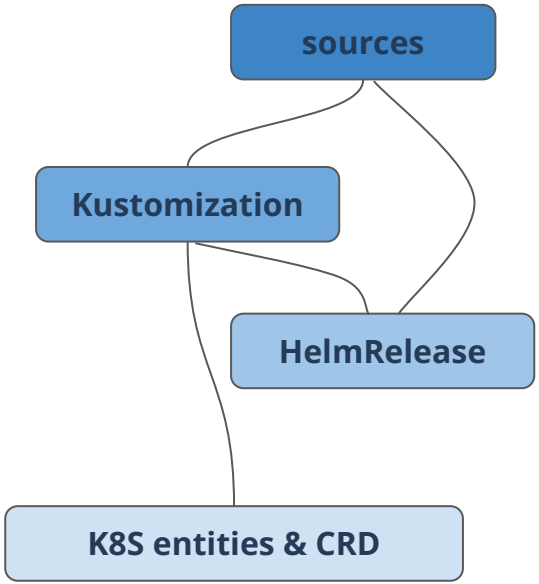
- Extracted from the official documentation

```yaml
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
 name: backend
 namespace: default
spec:
 interval: 5m
 chart:
   spec:
     chart: podinfo
     version: ">=4.0.0 <5.0.0"
     sourceRef:
       kind: HelmRepository
       name: podinfo
       namespace: default
     interval: 1m
 upgrade:
   remediation:
     remediateLastFailure: true
 test:
   enable: true
 values:
   service:
     grpcService: backend
   resources:
     requests:
       cpu: 100m
       memory: 64Mi
```

Extending the Kubernetes API

— A “*by design*” graphical representation

- What we saw implies a hierarchy between CRDs.



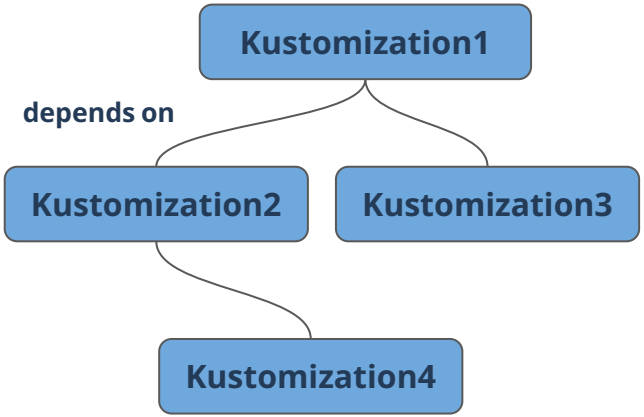
Interesting point: k8s entities and CRDs can very well be Flux entities.
For instance a GitRepository source and associated Kustomization.
The graph can grow indefinitely and should keep the property of a **D.A.G** (Directed Acyclic Graph).
“should” because it isn’t enforced, and if cycles are created, the Flux reconciliation flow will fail on the Kustomization creating the cycle.

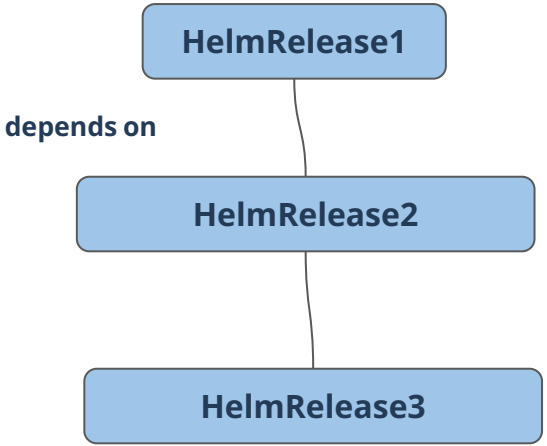
02

It depends...on !

— A “logical” graphical representation

- Somes Flux CRDs offers a “dependsOn” specification to reference other “same kind” entities to tell Flux that it should install it first. In other words, Flux enables orchestration.

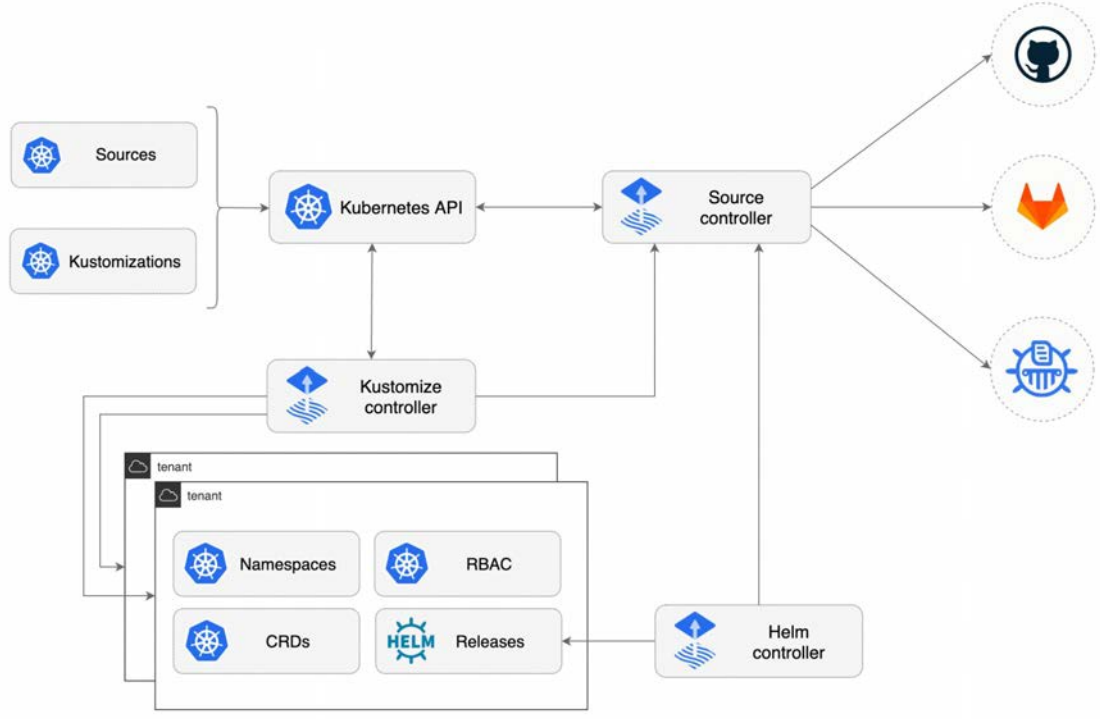




02

Listen carefully, 1, 2, 3.... start !

— Wrap up

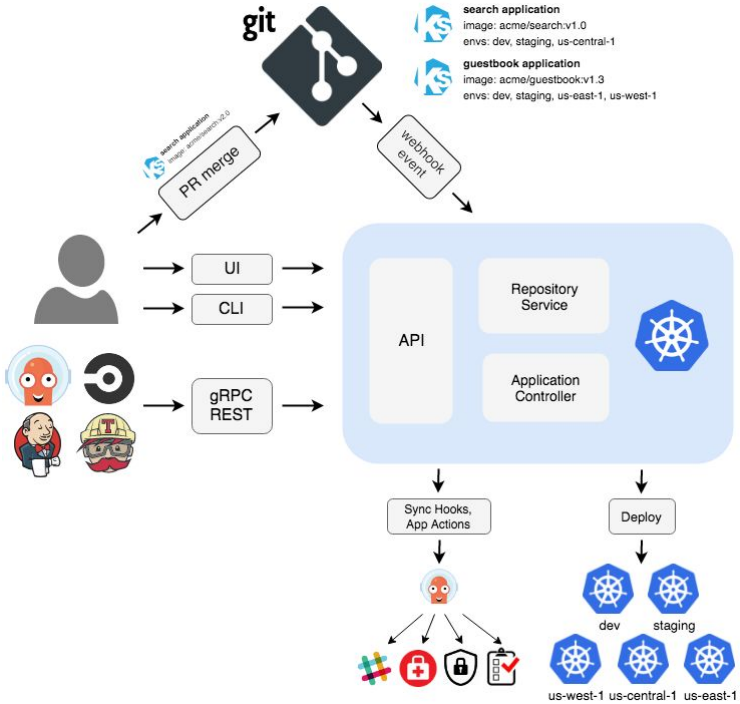


ARGO CD

02

A Domain Driven Approach of the GitOps

— ArgoCD entity model



02

The base entity of ArgoCD

— Application

- **A meta entity to describe your *Application* in the sense of ArgoCD**
  - Contrary to Flux it does not really stick to the k8s model.
  - It bring a new first class entity like the Deployment (it's the ArgoCD's Deployment)
- **Includes the source and the target destination**
  - Sources by default can be defined as {plain YAML files, Kustomize, Helm, Jsonnet}
  - Can be augmented with a system of plugin
- **All-in-one**: Handles raw manifests, helm charts, kustomize and other tools.
  - Contrary to Flux it does not segregate tools per CRDs
  - The cost is an extra complexity in the configuration, and an extra resource consumption by the associated controller
- **Multi-Cluster**

  App destination can be on the current cluster, or target another one but is visible to the ArgoCD installation by living in the same cluster as the one where Argo is installed.
- **"no accidents" defaults**:
  - no resources pruning / cascading by default
    - this means children resources can still exist on the cluster after removing an Application
  - manual apply by default (automatic apply can be configured)

02 YAML

— Application examples

- Extracted from the official documentation
- Different source configuration parameter depending the source type

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL:
https://github.com/argoproj/argocd-example-apps.gi
t
    targetRevision: HEAD
    path: guestbook
  destination:
    server: https://kubernetes.default.svc
    namespace: guestbook
```

```yaml
spec:
  source:
    repoURL:
https://argoproj.github.io/argo-helm
    chart: argo
```

02

Enhancing source possibilities

— Config Management Plugin

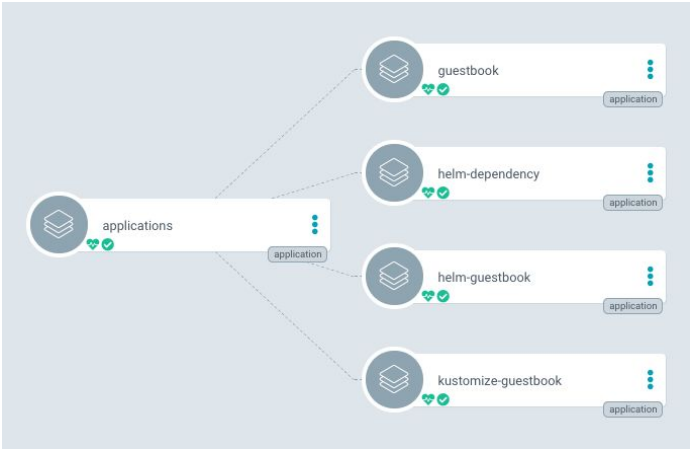
- Enhance the Application capabilities such as understanding additional config management tools as source type
- Community driven
- e.g: **kustomized-helm**
  - Allows you to define sources mixing Helm Charts and Kustomize overlays
  - The chart is rendered, then kustomize overlays are applied
  - It's the reverse of the Flux approach which applies Kustomize overlays on HelmRelase CRD wich then render the chart

02

Reflecting Applications

— Application of Applications

- An ArgoCD Application can itself point to sources containing Application definitions
  - You can create a DAG of Applications…
  - …which root application centrally control the sync behavior of leaf applications



02 Application Sync lifecycle

— Synchronization Hooks

- To specify specific behavior during the Application sync it is possible to bind process executions at different **phases** of the synchronization
  - Presync
  - Sync
  - PostSync
  - SyncFail
- Processes are generally represented by K8S jobs that are annotated to be taken into account as hooks

02 Sync lifecycle

— Synchronization priority

- Previous synchronization **phases** are organized by **waves** which can be represented by a weight attached to a resource to be sync. The lighter the weight, the higher the priority.
- Weaker than a true depends on relationship but still valuable
- Managed through annotations
- Strongly reminds Workflow management system at smaller scale

02

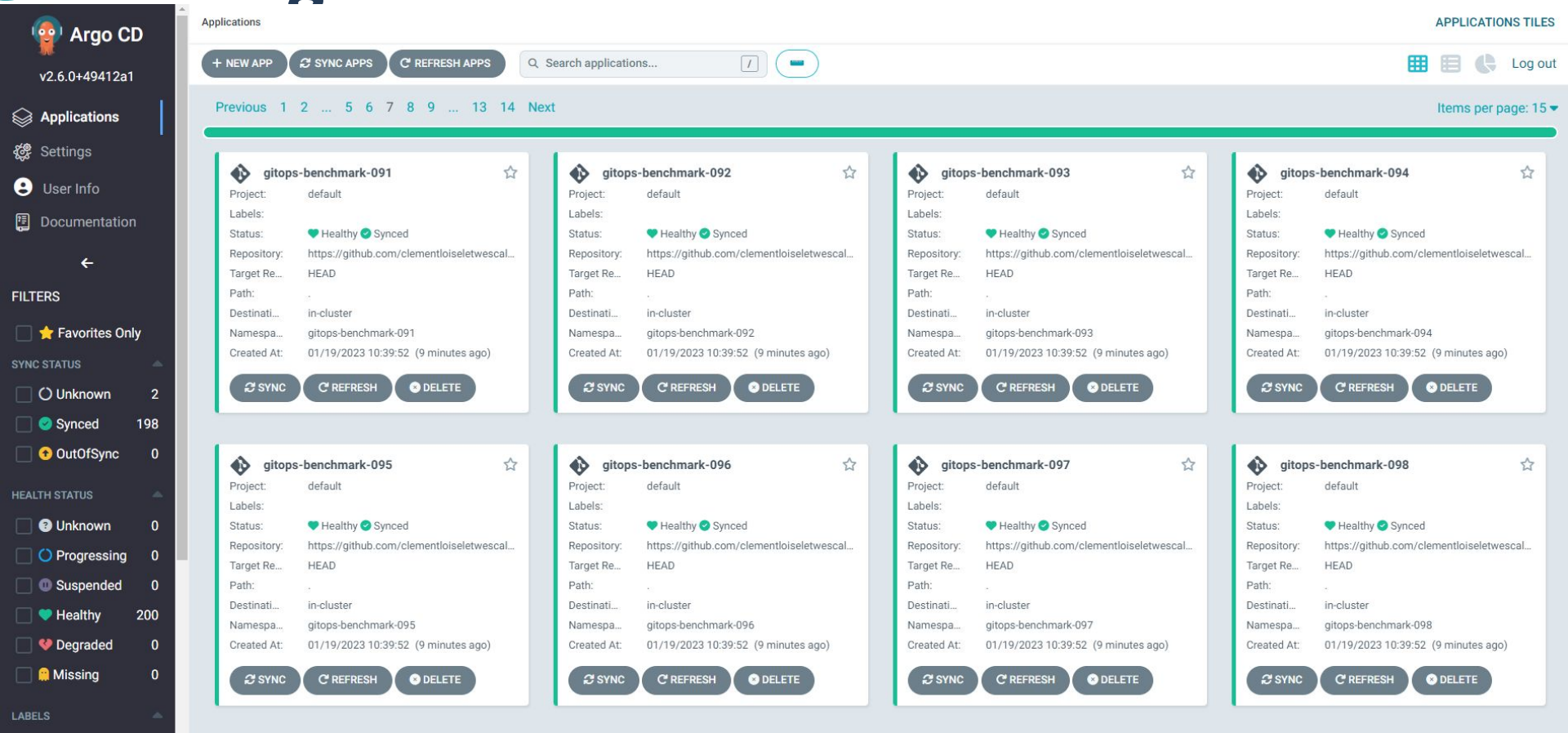
Last but not least

— ApplicationProject

- Represent a logical grouping of Applications
- It’s the ArgoCD’s namespace on steroid
- All Applications belong to an ApplicationProject. By default the *default*
- Defines common rules for the managed Applications
  - *what* may be deployed
  - *where* apps may be deployed
  - *restrict* what kind of object may or may not be deployed
  - defining roles through RBAC applied to all applications in the project
  - *many more*

02 ArgoCD

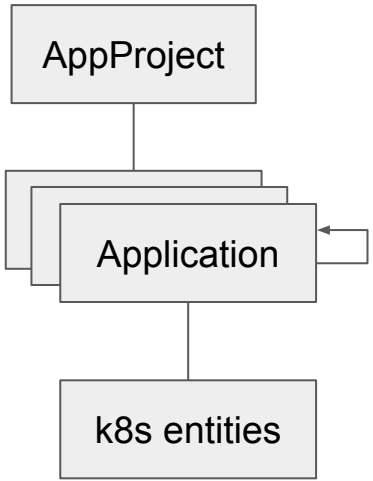
— ArgoCD's Web UI



02

ArgoCD's CRD

— Dependency model



- Applications belongs to a Project, *default* by default
- Applications will directly create k8s entities
  - helm will be installed with `helm template | kubectl apply -f .`
- Application can handle workload on another cluster

WRAP UP

02

What to understand

— Wrap up

	Flux	ArgoCD
model complexity	simple model	rich model
GitOps approach	thin extra layer to Kubernetes	promotes its own GitOps entities
Relation to k8s	stick to what k8s provides. Kind of k8s GitOps extension	richer interface to the cost of greater complexity
Enriching model	Operator pattern	Plugins
Sync orchestration	Strong sync orchestration	Sync orchestration by waves
Extra sync actions	Init containers	Sync phases with hooks
Main CRDs	Sources + Kustomization	AppProject + Application
Need for an UI ?	K9S is enough	Most likely bc of the rich parameter features

03

multi-{clustering, tenancy}

03 Advanced modeling

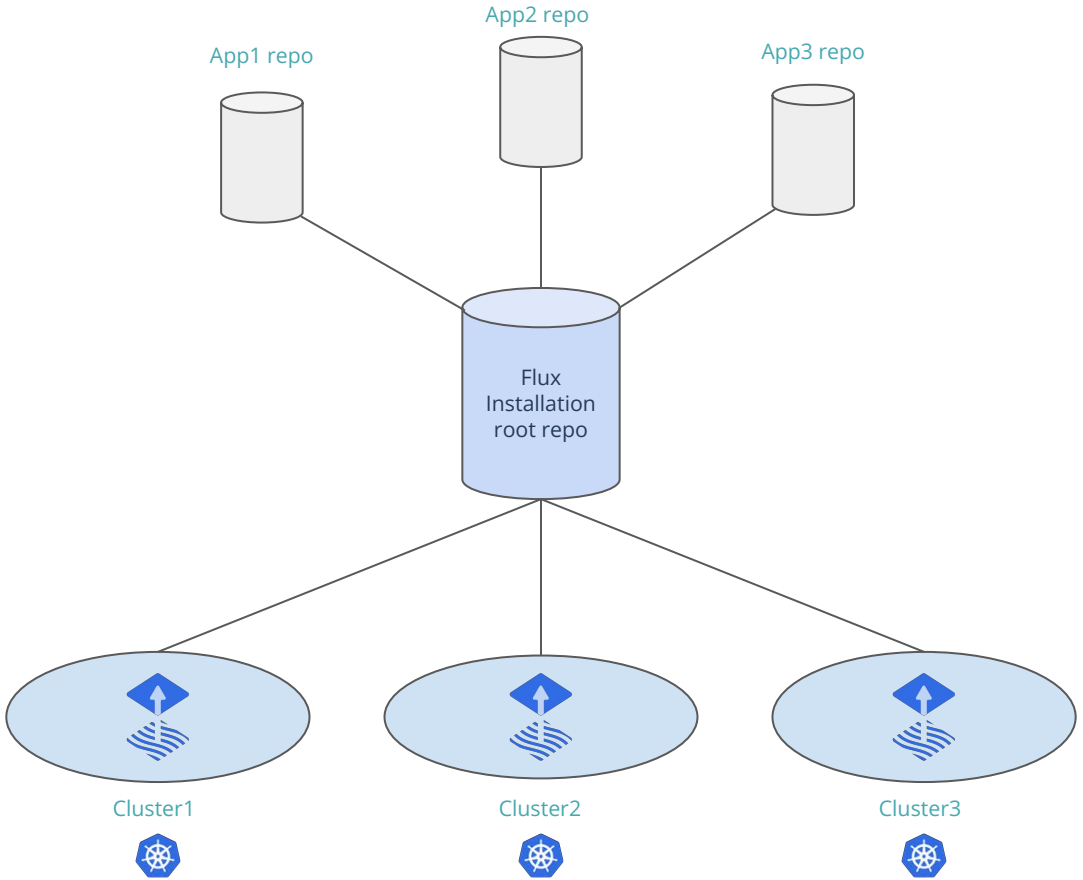
— GitOps & Multi-clustering

- **Multic-clustering**
  - Capacity to manage several k8s cluster infrastructures through a single control plane
- **From a Gitops perspective**
  - Capacity to synchronize applications on several cluster while using a single tool installation

03

A distributed control plane

Flux multi-clustering approach



- Application infrastructure repositories (Deployments, services, ...) and sources (java, php, python, ...)

- Contain a flux installation sources per cluster
- Each installation sources kickstart syncing referencing others {Source + Kustomization}
- {Source + Kustomization} may concern others repository dedicated to your application Deployments

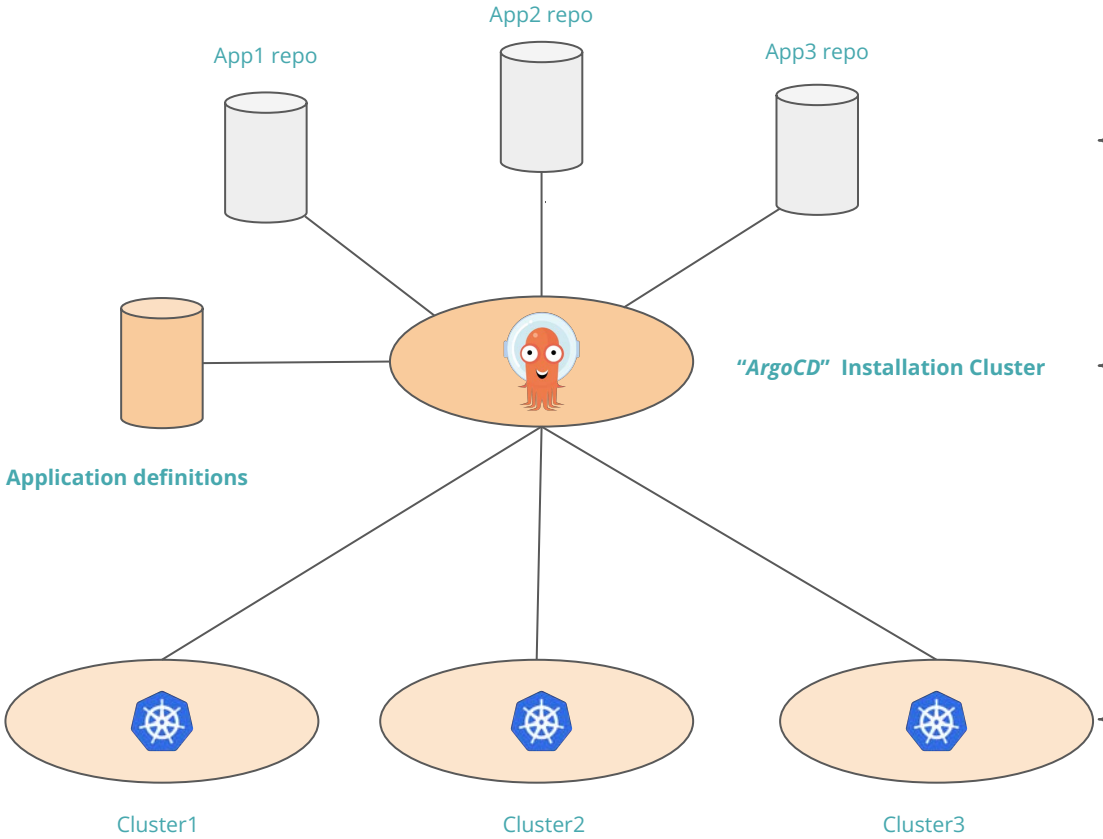
- Each cluster has a dedicated flux installation
- Each installation is synchronized pointing the same Flux installation Git repository while specifying dedicated folder in it

wescale

03

A distributed control plane

_ ArgoCD multi-clustering approach



- Application infrastructure repositories (Deployments, services, ...) and sources (java, php, python, ...)

- ArgoCD installed in a unique cluster
- Optional access to other clusters
- Monitoring a root repository to define and store Application kind resources

- Each cluster is registered from the "ArgoCD" installation cluster (optional)
- No ArgoCD installation

wescale

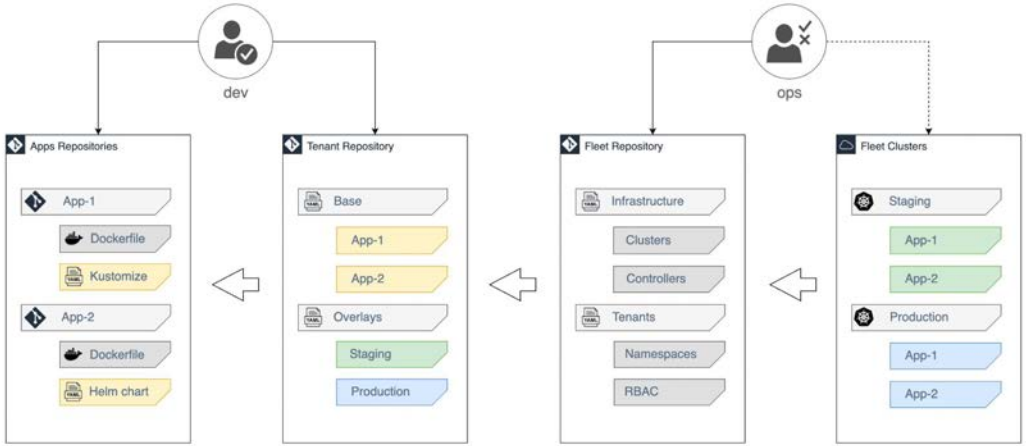
03 Advanced modeling

— GitOps & Multi-tenancy

- **A tenant**
  - Segregation quantum. A team, an organisation, an environment,...it really depends of your context
- **Multi-tenant**
  - Capacity of an entity to embed multiple tenant without interacting with one anothers
  - Capacity of an entity to handle multiple tenants with differents rights on different objects
- **From a Gitops perspective**
  - Tenant bounded to a namespace

03 Multi-tenancy — Flux approach

- Extracted from the official documentation. A possible approach



- **Tenant Squad**
  - Admin of the namespace**s** and the Git repository assigned by platform admin
  - Manages application deployment & release (`GitRepositories`, `Kustomizations`, `HelmRepositories`, `HelmReleases`)
- **Platform Admin**
  - Maintain platform Git repository
  - Manage cluster wide resources (add-ons, CRDs, controllers, etc.)
  - Onboard Tenant CRDs (`Kustomizations`, `GitRepository`)
  - Manage Tenants namespaces & RBAC

03

Multi-tenancy

— ArgoCD approach

- Use of AppProject to restrict the rights of the created ressources
  - restrict the destination cluster+namespaces
  - restrict the kind of object created
  - restrict the repo used as source

- integrated RBAC & SSO to restrict the rights of the users on the actions on the UI

03

User experience

03 Let's get party started

— Our casting

The user team

We develop applications and rely on the Gitops paradigm to deploy it (and by the way, it works on my machine ^^).

The admin team

We install GitOps tools on the k8s cluster and are accountable for its behavior (and by the way, your application fails ^^)

wescale

03 Admin point of vue
— Day 0 : Organisation

- Before the installation and usage, we must **first discuss how the git repositories should be handled**
    - Do we want a multi repo ? if so, what scope should each repository have ?
    - How should the repositories be structured ? one folder per environment, per namespace ?
    - Who is responsible for the code in the repositories ?
    - How do we protect the production ? one commit/PR per environment ?

These choices should be discussed before implementing a GitOps solution to prevent refactoring a difficult codebase in the near future.

03

Admin point of view

— Flux Day 1: installation

- **Installation**
  - Bootstrapped with the CLI : `$ flux bootstrap github`
  - Terraform official Flux datasource combined with Kubernetes provider
  - Community Helm chart
  - A git repository is needed to install flux (or highly recommended); we'll call it the root repository.

- **Flux's manifests will be committed to the root repository**

- **You can add all your manifests in the root repository before or after the installation of Flux**
  - On a new cluster, you can point your Flux installation to the existing root repository and all your applications will be installed.

03

Admin point of view

— Flux Day 1: Configuration


- **Simple, if you follow the doc**
  - each CRD does only one thing
  - GitRepo to `git clone` your repository
  - Kustomization to apply some path on this repository
  - HelmRepository to `helm repo add && helm repo update`
  - HelmRelease to deploy your helm release
  - some minor configurations possible (timing, ...)

03

Admin point of view

— Flux Day 2: Update, Debugging, monitoring

- **day to day** : all handled in git - no manual actions required on Flux
  - update → *git commit*
  - revert → *git revert*
  - pause/unpause → with annotations, either in Flux or directly on the cluster.

- **debugging**
  - access to cluster required (CRDs events, flux's log), usage of k9s
  - OR access to a properly configured Grafana
  - OR access to weaveworks flux's UI (early project)

- **monitoring**
  - Grafana → opensource dashboards freely available for Flux)

03

Admin point of view

Flux Day 2: Update, Debugging, monitoring

- **Cluster upgrade**
  - Can be done with the CLI (a little complex)
  - Can be done with Terraform (simple)

- It is not recommended to use a GitOps installation to directly perform an upgrade of the same installation;
  - either use the recommended way
  - or use another gitops installation to perform the upgrade & maintenance of the first one.

03

Admin point of view

— ArgoCD Day 1: installation

- **Installation**
  - Helm
  - Kubectl apply / Kustomize
  - No initial git repository needed
- **Installation type**
  - HA ? non-HA ? core-only ?
- **No initial ArgoCD Application**
- **ArgoCD’s installation manifests do not need to be in one of the monitored repositories**

03

Admin point of view

— ArgoCD Day 1: configuration



- **Complex**
  - 1 Application with a lot of options
  - should it autosync ?
  - should it prune resources that have been removed from git ?
  - should it delete resources when the Application is removed ?
  - Which wave should it be ? (which applications does it depends on ?)
- **But simplified when done with the UI**

03

Admin point of view

ArgoCD Day 2: Update, Debugging, monitoring

- **day to day**
  - handled in git and in the UI, depending on your configuration
- **debugging**
  - the UI contains information about a misconfigured Application or even why a Deployment is failing
- **monitoring**
  - The UI enables you to monitor your gitops deployments easily

03 Admin point of view

— ArgoCD Day 2: Upgrade

- **Cluster upgrade**:
  - using Helm or Kustomize, depending on how you did your initial installation

- Read the release note before upgrading, there may be breaking changes !

03 User experience

— Wrap up

	 ArgoCD	 Flux
platform installation	simple with kubernetes standard tools	advanced with custom tooling, manual actions
configuration	complex lot of options for different usecases	simple does one thing, does it well
usage	simple	simple
safeguards	argocd can be configured to need manual approval before apply	none : a commit to git is a change in the cluster

04

Benchmark

04 Benchmark — Comparison points

- **Volumetry used**
  - Number of final Kubernetes objects created
    - **~ 1400**
  - Number of Deployments created
    - **200** (1 pod per deployment)
  - Number of sources monitored
    - 100 distinct git repositories
    - 100 helm releases from the same chart

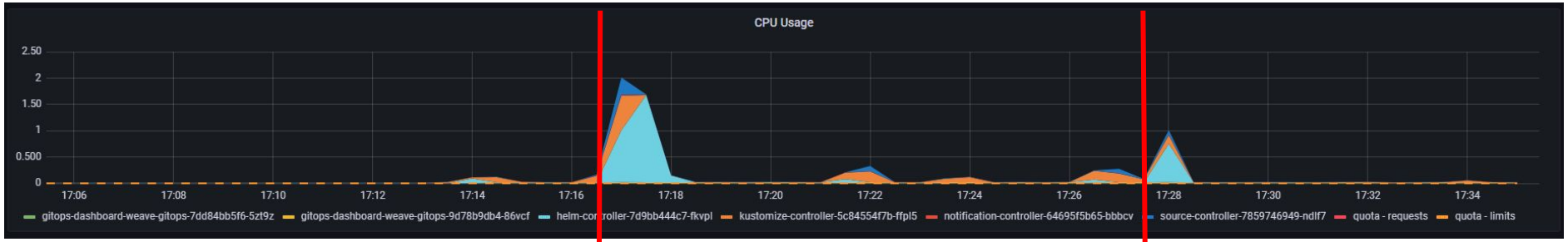
- **Points of comparison**
  - Time to deploy all objects
  - CPU & memory consumed before + during the initial application
  - CPU & memory consumed while monitoring the objects for change
  - CPU & memory consumed while removing the objects.

Only the core components will be considered for the benchmark (no webhooks).

04 Benchmark — Initial setup

	ArgoCD	Flux
cluster	AWS EKS - 5 t3.large	
monitored sources	1 root repository	
installation	non-HA, graphical	standard installation (4 controllers)
version	2.6.0	0.38.3
cpu	<0.1	<0.1
ram	240MiB	300MiB

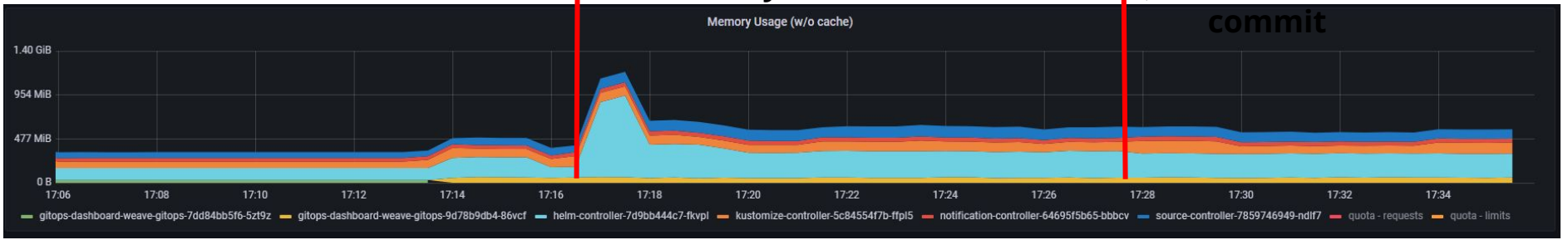
04 Benchmark — Flux Timeline



CPU Usage

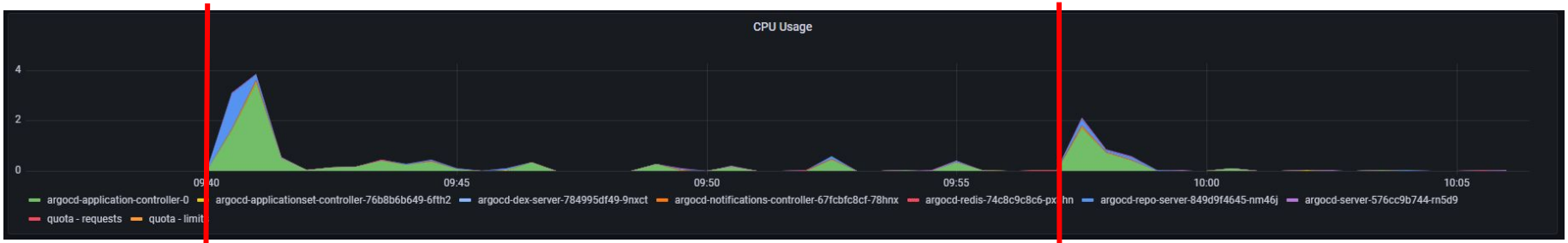
First sync

Removal commit



Memory Usage

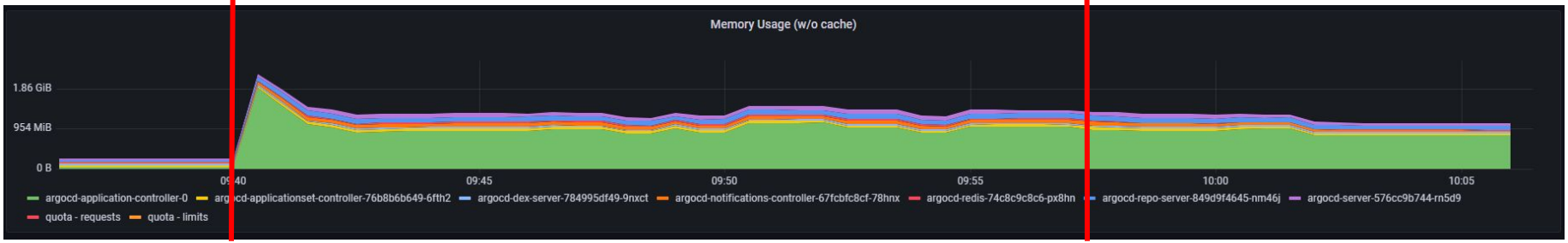
04 Benchmark — ArgoCD's Timeline



CPU Usage

First sync

Removal



Memory Usage

wescale

04 Benchmark

— Deployment: 100 sources, 1000 objects

	ArgoCD	Flux
monitored sources	100 gitrepos + 100 helm repos	
cpu	3.9	2
ram	2.2 GiB	1.2 GiB
time to deploy all objects	~1mn30	~1mn30

04 Benchmark

— Day to day: 100 sources, 1000 objects

	 ArgoCD	 Flux
monitored sources	100 gitrepos + 100 helm repos	
cpu	0.3	0.3
ram	1.4 GiB	500 MiB

04 Benchmark

— Deletion: 100 sources, 1000 objects

	ArgoCD	Flux
monitored sources	100 gitrepos + 100 helm repos	
cpu	2	1
ram	1.4 GiB	500 MiB
time to delete all objects	1mn	30s

04 Benchmark — Results

- Both are comparable in terms of deployment & update speed : lightspeed
- ArgoCD's resources consumption is higher than Flux’s
  - It comes from the architectural choices of ArgoCD
- In our experience, this difference is increased by the number of deployments / users, and should be taken into account when designing your clusters.

05

Conclusion

05 — Conclusion

ArgoCD is best-suited for you if

- You don’t want to use an auto GitOps CD
- You want an interactive UI for your users
- You need to ensure some steps remain manual, e.g. deploying to production
- You want sane defaults preventing accidental removal of components.

Flux is best-suited for you if

- Your users are experienced in kubernetes
- You don’t need a lot of deployment options
- Git & k8s RBAC are enough for you
- Your users understand that adding / modifying / removing something in git will have a direct impact on kubernetes.

Acknowledgments

Bastien Feuillet

wescale

Question Time !