
Giant Swarm
GitOps 101
Lorenzo Soligo - Linux Day Milano - 22.10.2022
KUBERNETES

Agenda

	**git blame lorenzo.md**	*Get to know me*
	**Manual deployment**	*Ouch, he's sick today!*
	**CI/CD**	*Look at my super readable Jenkins pipeline ;)*
	**GitOps**	*Whooopsie, can't hammer the cluster today*
	**The future of GitOps**	*Infrastructure, blackjack, open questions, 42*

git blame lorenzo.md

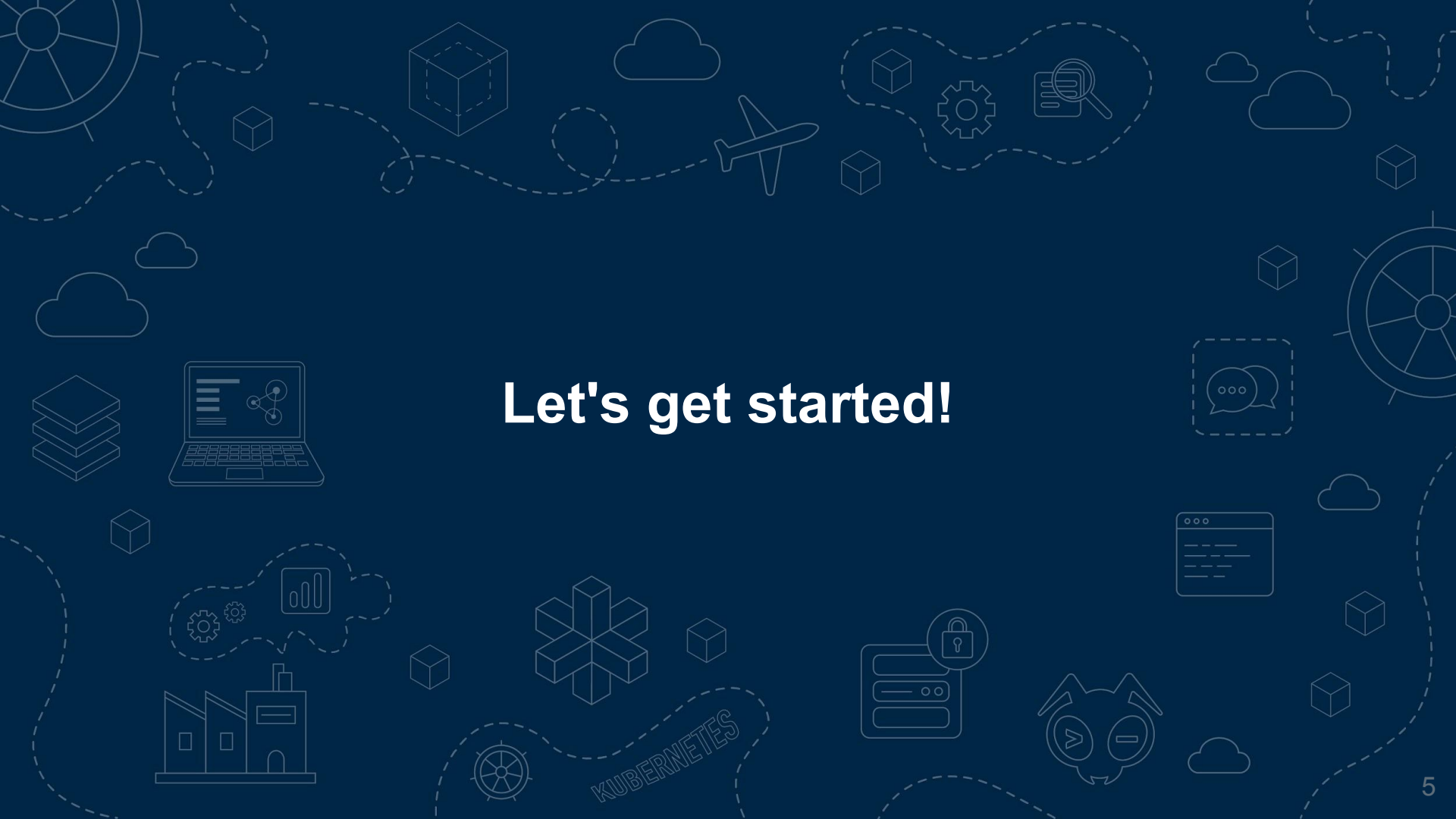
- I studied at this University (and feel old just saying this)
- I work at Giant Swarm, we do a lot of fun Kubernetes stuff
- I enjoy teaching and learning about cloud technologies, sooo… let's give a talk!
- Most people in this room are way more skilled than I am :)

(don't actually blame me)

1. Because of time constraints and my ignorance, I might be touching very briefly (or not at all) many technologies, challenges, …

2. This presentation heavily relies on sarcasm and many of the horror stories you will hear about have actually been experienced/seen first-person by your lovely host

Please bear with me :)



Let's get started!

The first steps: manual deployment

Manual deployment of code and infrastructure

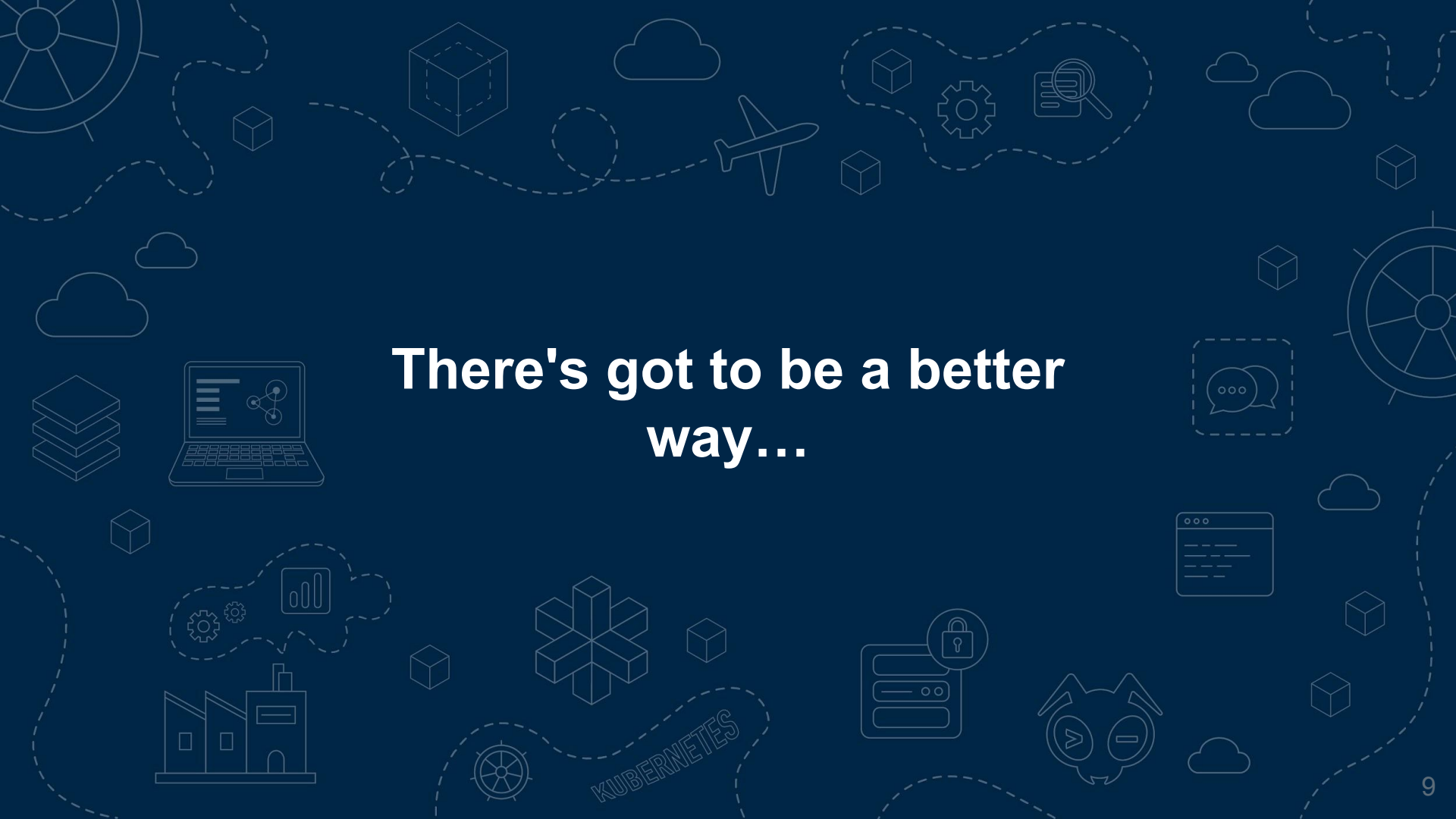
- Installation, upgrade, deletion, … all done by hand:
  - Create a VM -> ask the VMware guy
  - Upgrade an app -> pray that everything works
  - Double a VM in size -> re-ask the VMware guy
  - Update a security group -> fill in 5 forms to the network team
  - Install an app -> follow the docs
  - …

This doesn't scale!

- In the end we always rely on the same old people doing the same old things
- There is no way to *really* be sure of what's being done
- Code from Git (hoping it's used) to environments takes ages
- Super hard to validate environments

In general: no standardization, slow process, tracking almost impossible, it's a mess!

There's got to be a better way…



The power of automation: CI/CD

The fundamentals

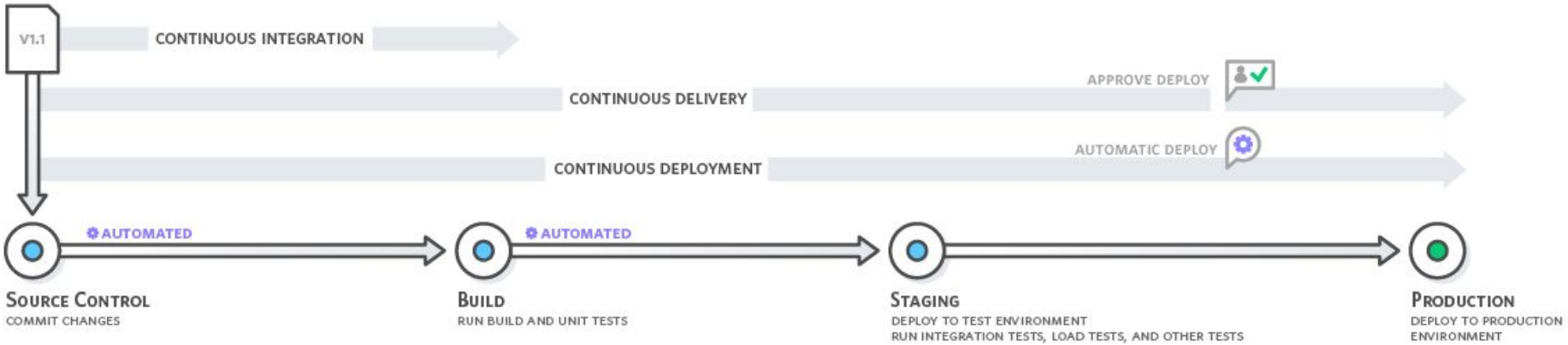
- **Git**: *distributed version control system*
  - Cooperate on code, enforce checks on it, …

- **Continuous Integration (CI)**: *[...] the practice of merging all developers' working copies to a shared mainline several times a day*
  - In other words: merge on *main* frequently, run tests, build (e.g. Docker image)

- **Continuous Delivery (CD)**: *[...] expands upon continuous integration by deploying all code changes to a testing environment [...]*
  - In other words: take CI and then actually install the software in a lower environment

Credits to AWS for definitions and image

Let's visualize

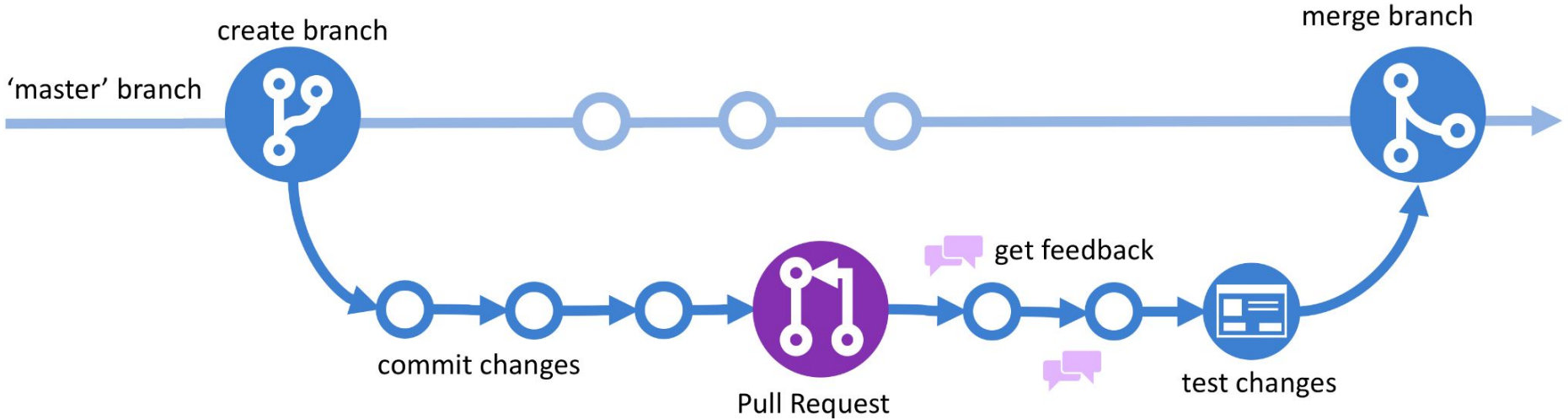


Credits to AWS for definitions and image

The basic workflow

- I work on a feature/fix/…
- When I believe the code is ready, I open a Pull Request/Merge Request to merge the *branch* I worked on to *main*
- Some automatic tests are run and my colleagues can leave comments on my code, ask for changes, and finally approve it
- My branch is merged into *main*
- Tests (hopefully) pass, a new image is created and pushed to some registry
- In "mature" cases, the image is automatically pushed to some dev/test environment

GitHub Flow



http://buildazure.com

This is getting better!

- If tests don't pass, the code does not even get to *main*
- Test & build steps are coded → standardized!
- (If well structured) encourages best practices, e.g. code review
- Easy promotion between environments
- Faster delivery of code, even when fixing business-critical code/bugs

Let's play a game

- I have a CI/CD pipeline that gets triggered when some new branch is merged into *main*. The image is pushed to a test environment and, one week later, to production.
- The app isn't updated frequently, let's say once every two months on average.
- One day, soon after the production pipeline runs, the application slows down.
- As a man of culture, I log into the production environment and double CPU and RAM for the application, as well as the size of the VM it is running on.
- **All good, right?**

The drawbacks of CI/CD

- Code is deployed and forgotten about until the next pipeline
- The smart engineer can still log into an environment and hammer stuff around
  - … which is reverted as soon as the pipeline is triggered again :)
- Many pipelines get super complex, don't really follow best practices, become unreadable

Let's introduce GitOps!

Much ado about… what?

Fundamentals of the GitOps approach:

- A Git repository represents the **desired state** you want to have…
- …as the **single source of truth**…
- …and the state of the **system is constantly reconciled** to match the desired state

Which means…

- As soon as the smart engineer changes something in a system managed via GitOps, the change is reverted (he probably didn't need those extra GBs of RAM anyway 😬)
- There is only one way to push to a system: merging new changes in Git
- If something breaks, we can just revert the commit in Git 🎉
- We have a clear history of changes
- It can be used as a disaster recovery tool if we represent entire systems (e.g. clusters + apps) as resources in Git and rely on eventual consistency

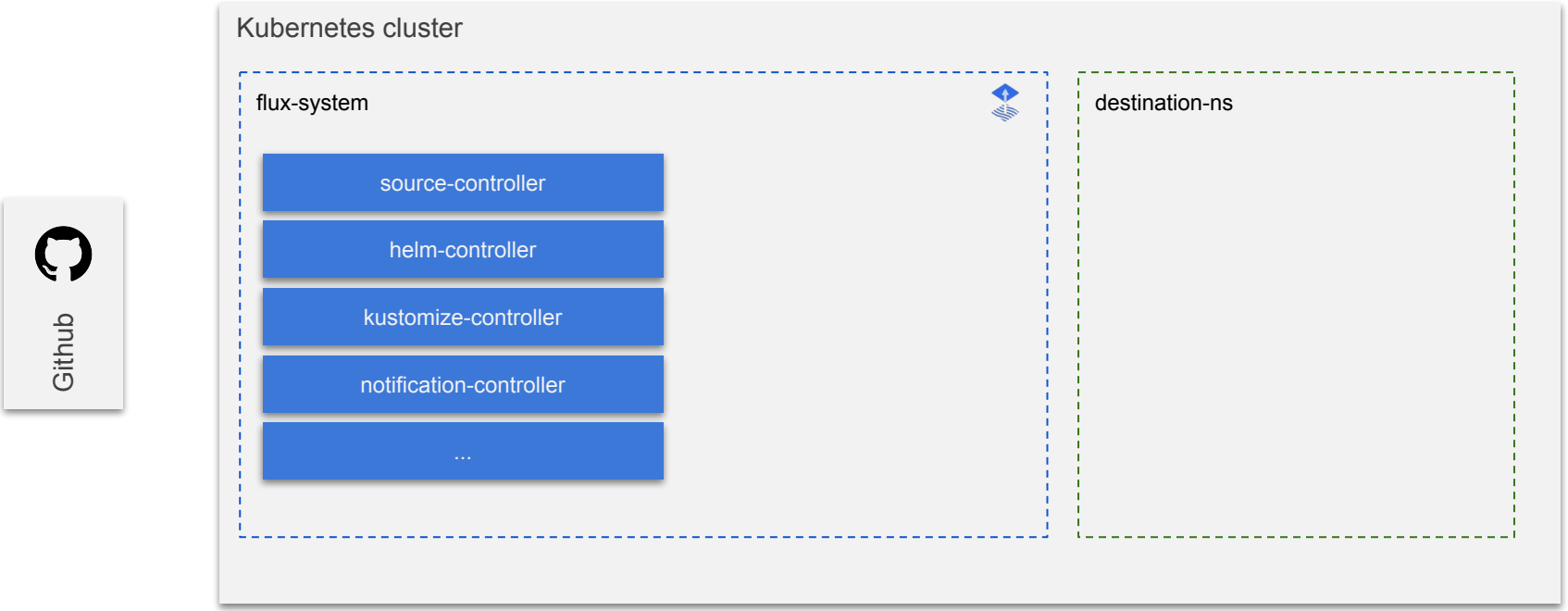
Tools you've likely heard of

- Fundamentally two leaders: **Argo CD** and **Flux CD**
- Both are nice tools, each one has pros and cons
- If you're looking for something "barebone" that's very Kubernetes oriented and *just works*™, Flux is nice
- If you're looking for something closer to a suite of services with a nice UI as well, you might want to consider Argo

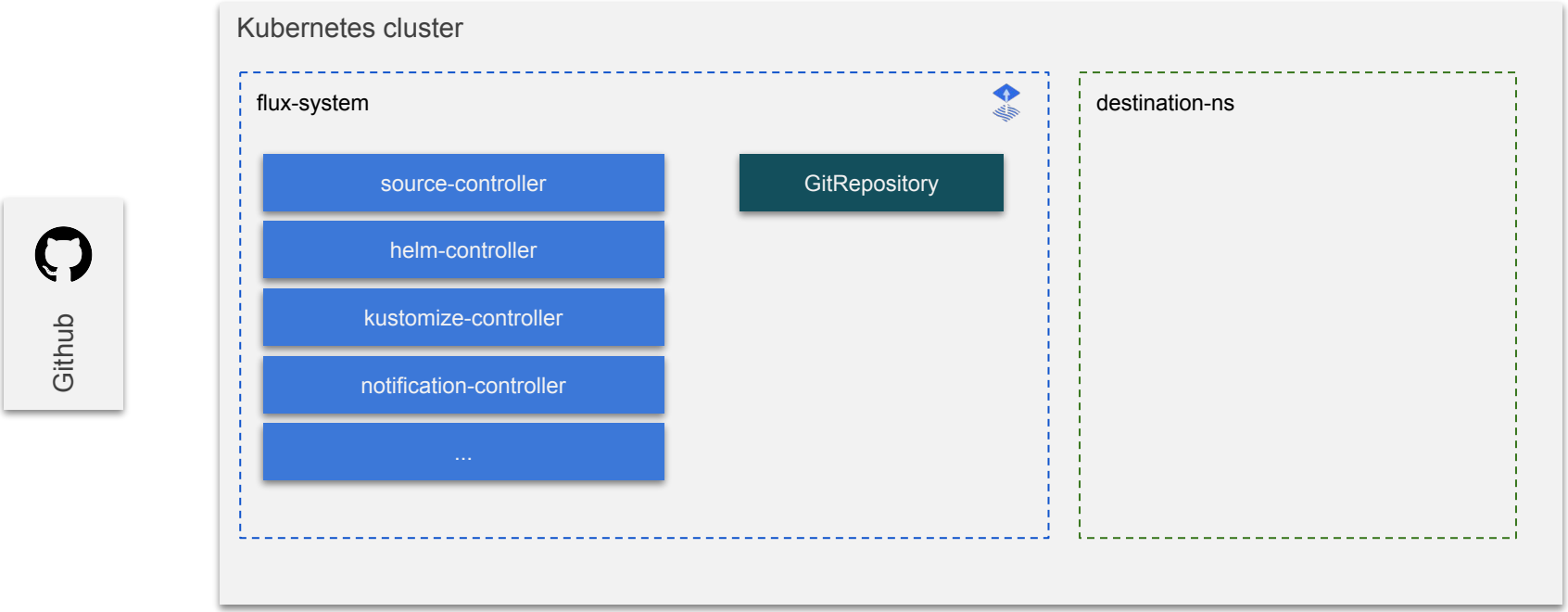
Reminder: this approach is very recent (~2017). Tools are quickly evolving.


Giant Swarm
GitOps flow with FluxCD
Totally stolen from the GitOps team at Giant Swarm ❤
KUBERNETES

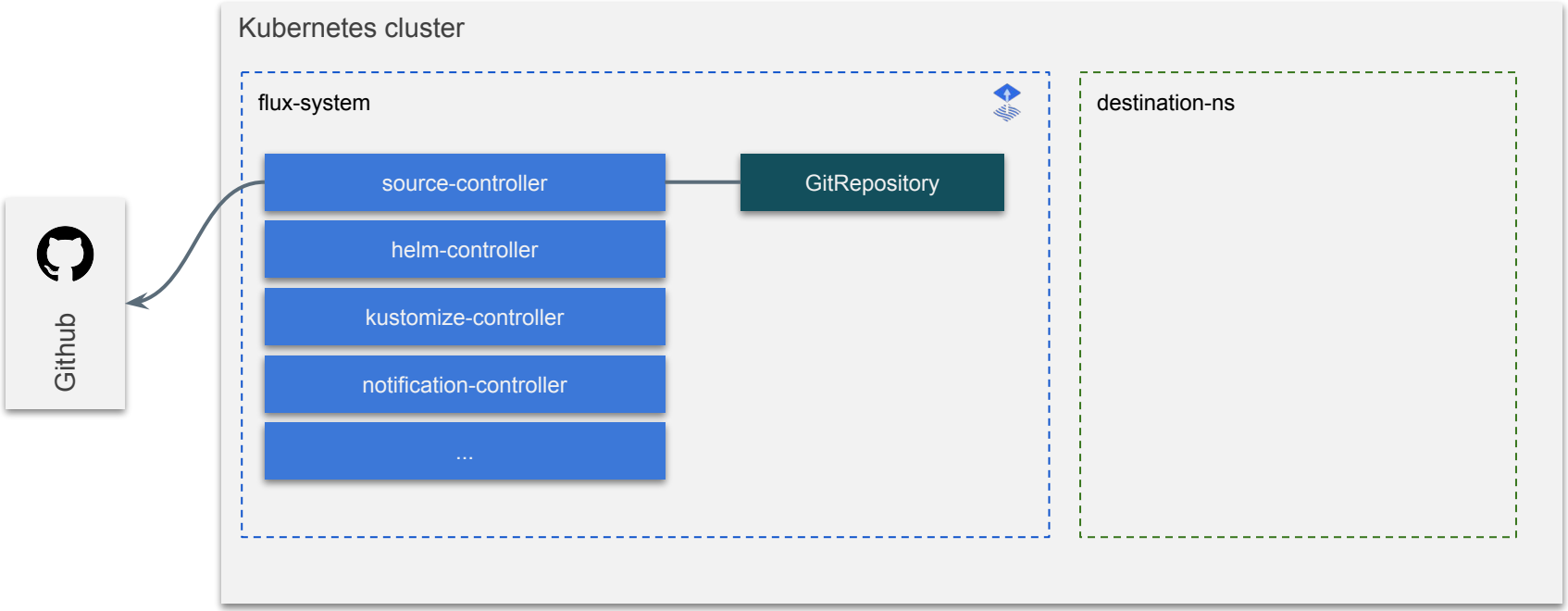
GitOps flow with FluxCD - overview



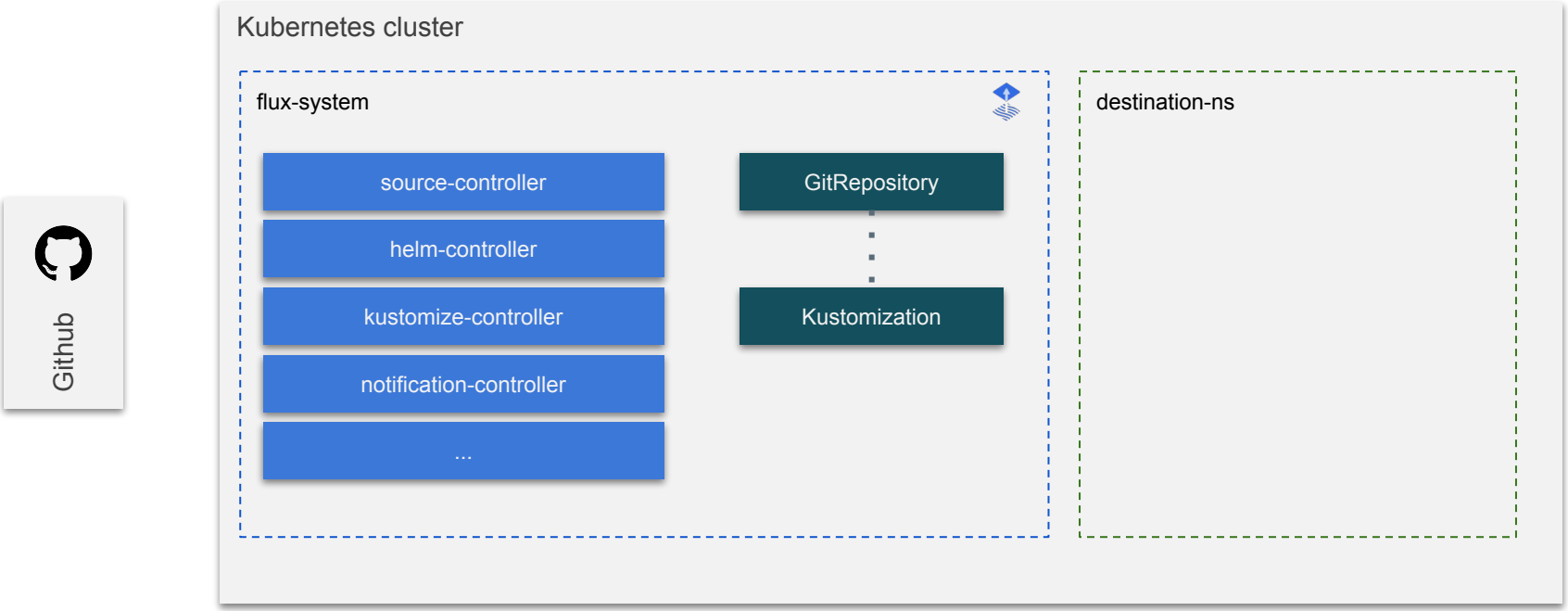
GitOps flow with FluxCD - overview



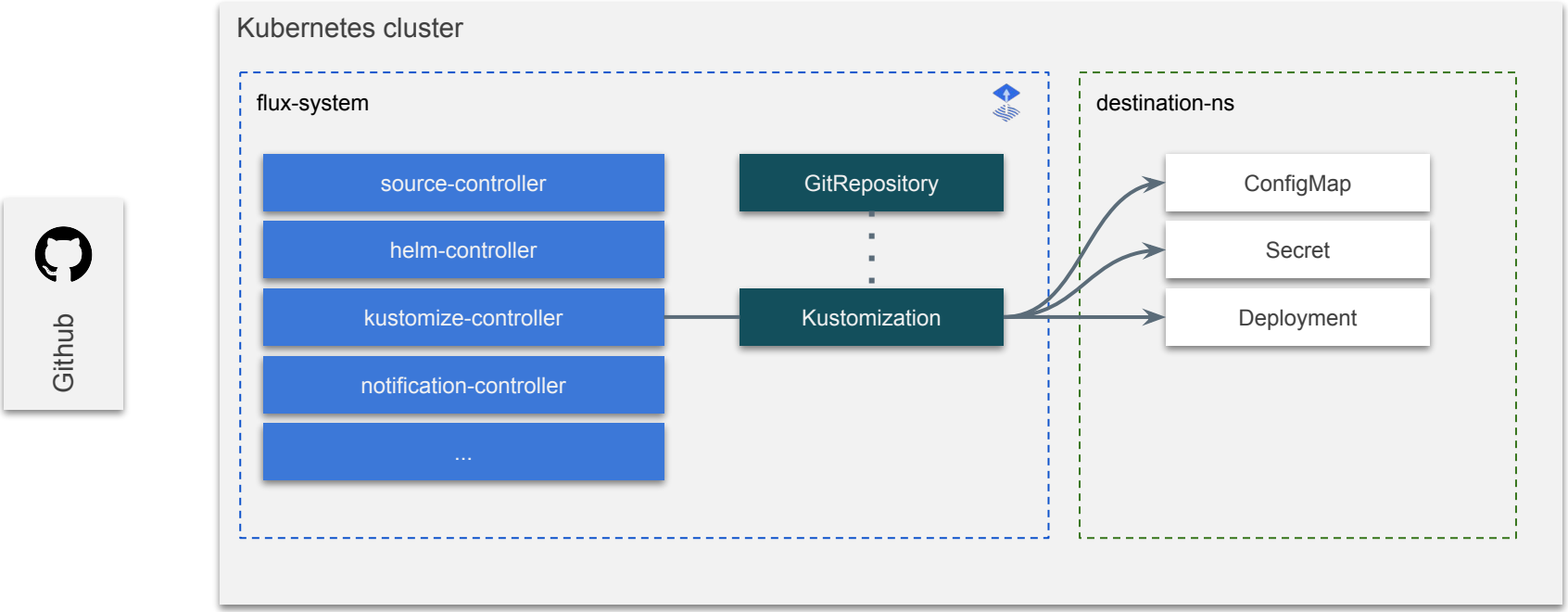
GitOps flow with FluxCD - overview



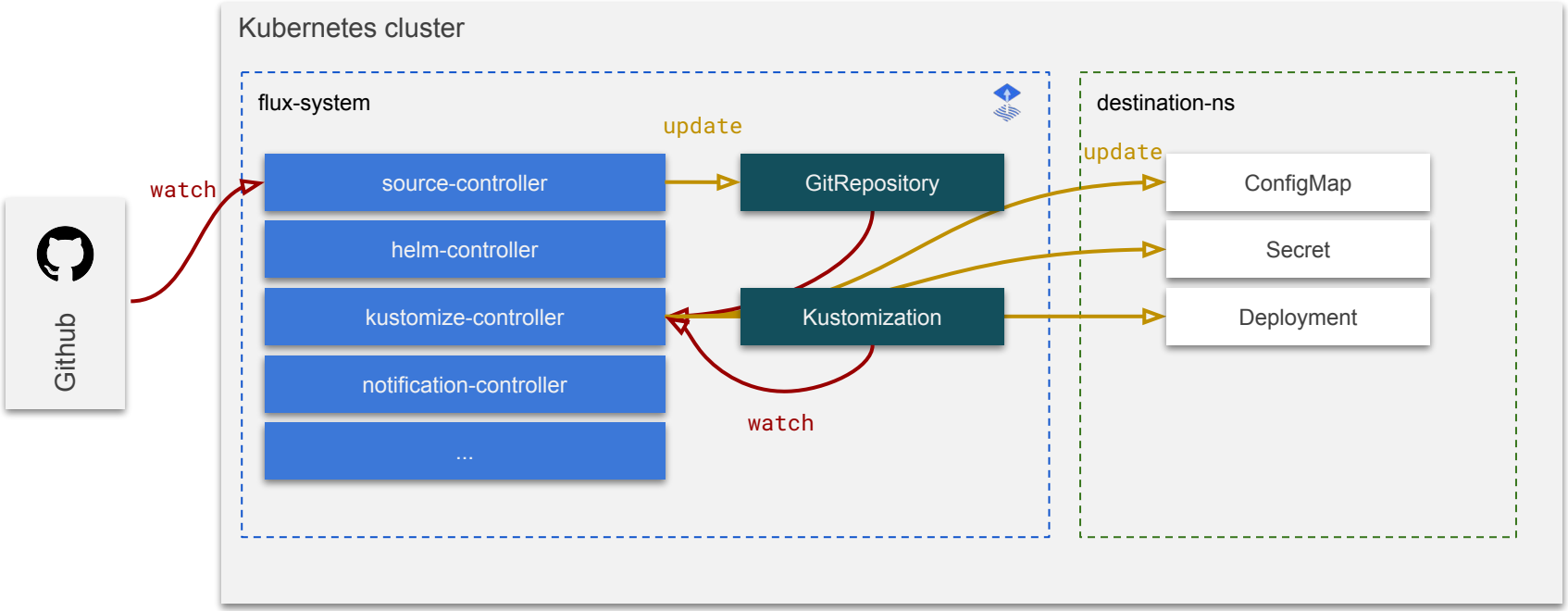
GitOps flow with FluxCD - overview



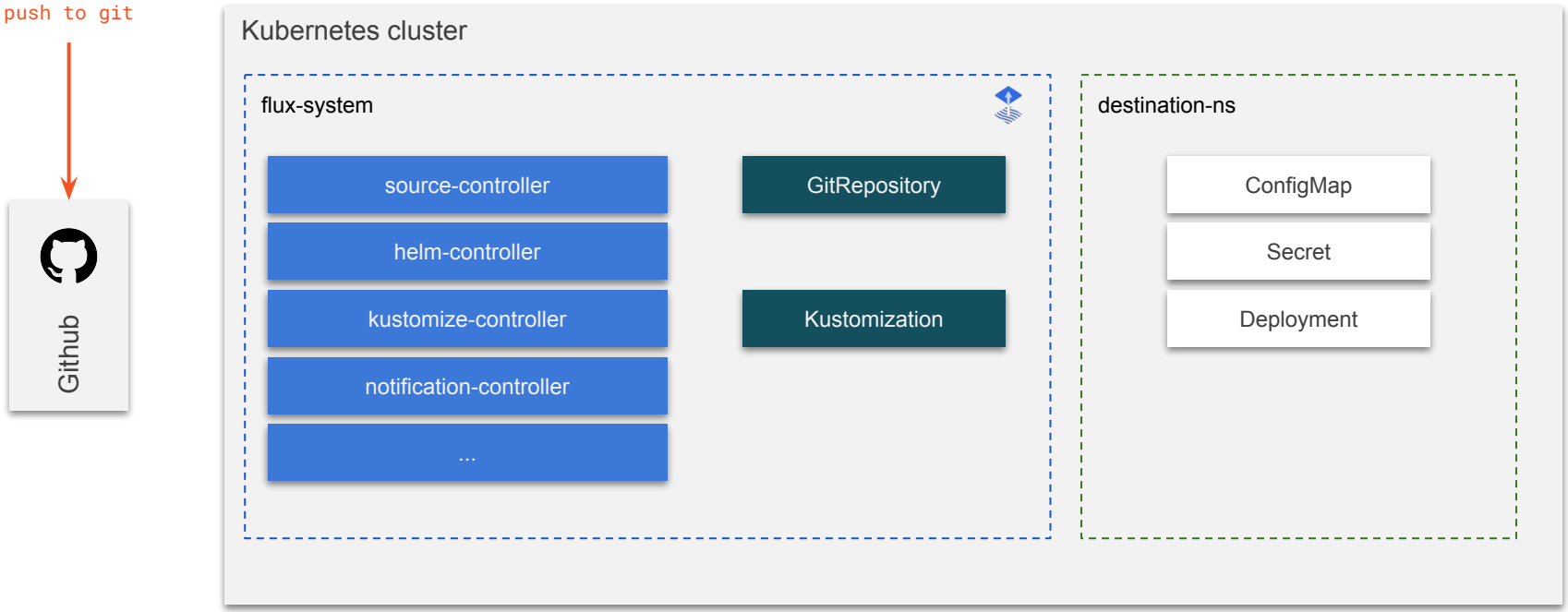
GitOps flow with FluxCD - overview



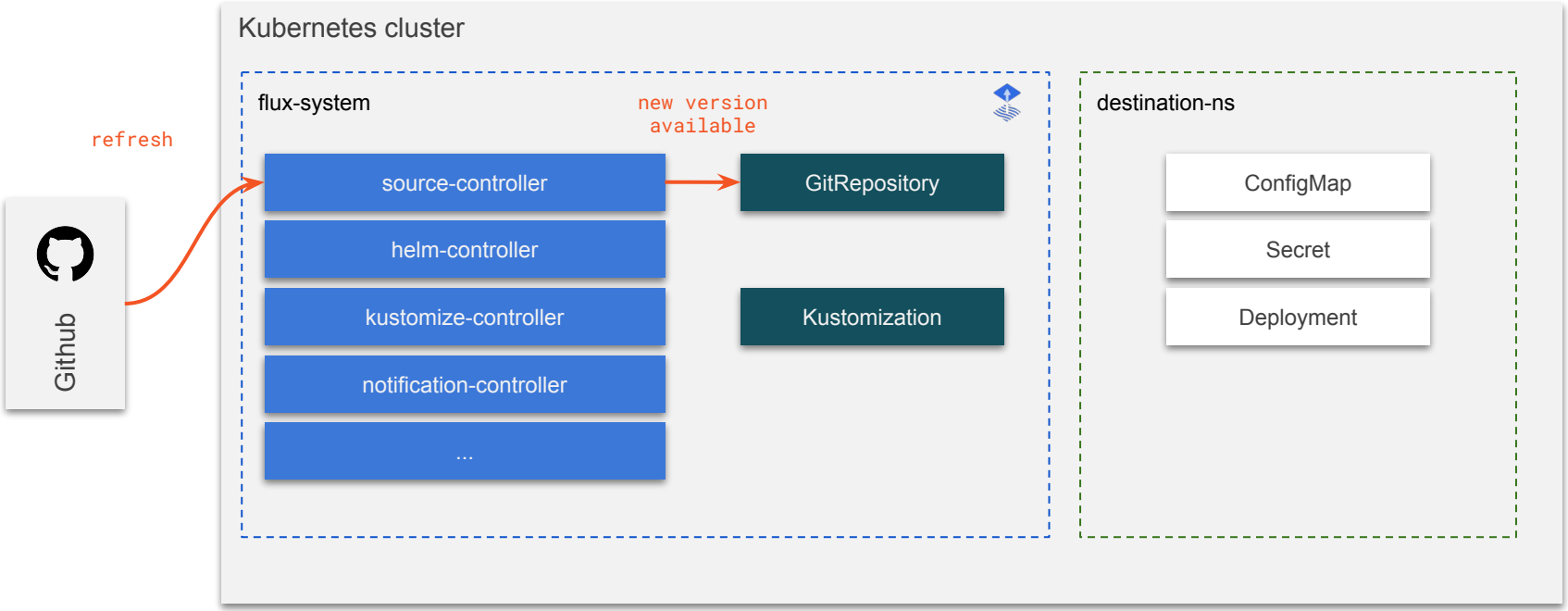
GitOps flow with FluxCD - overview



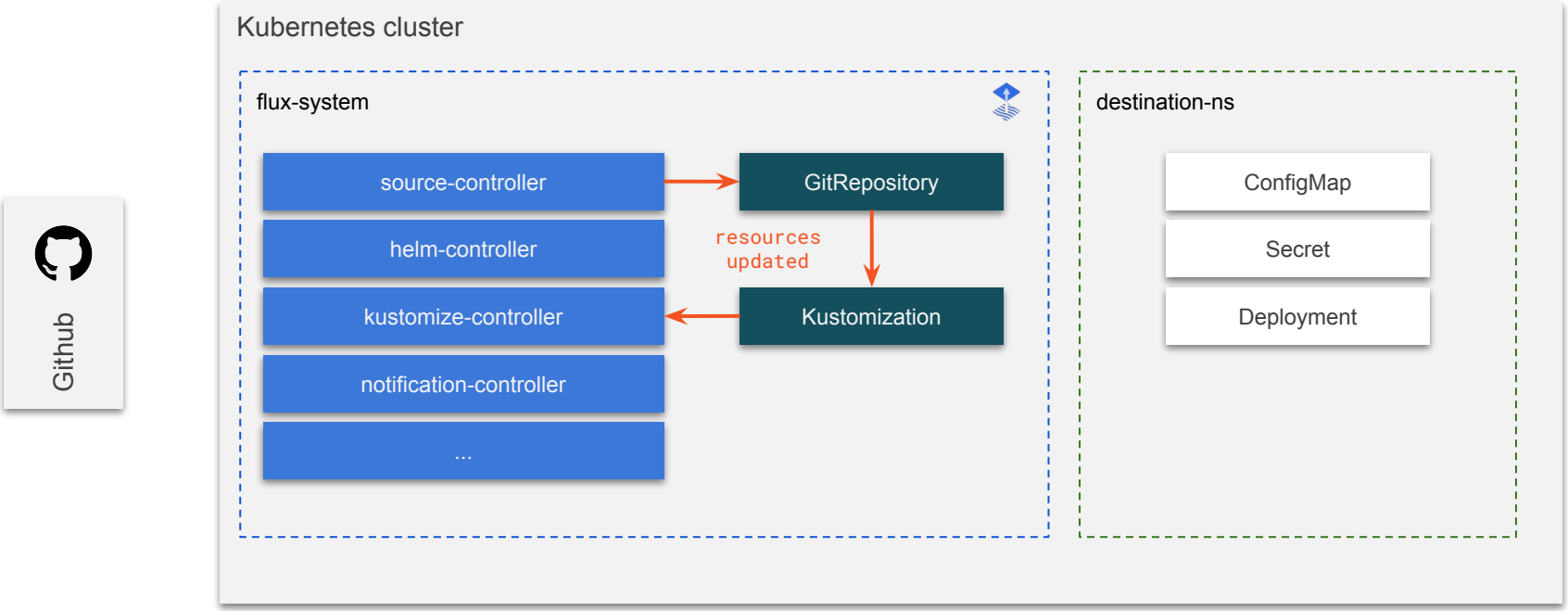
GitOps flow with FluxCD - step-by-step



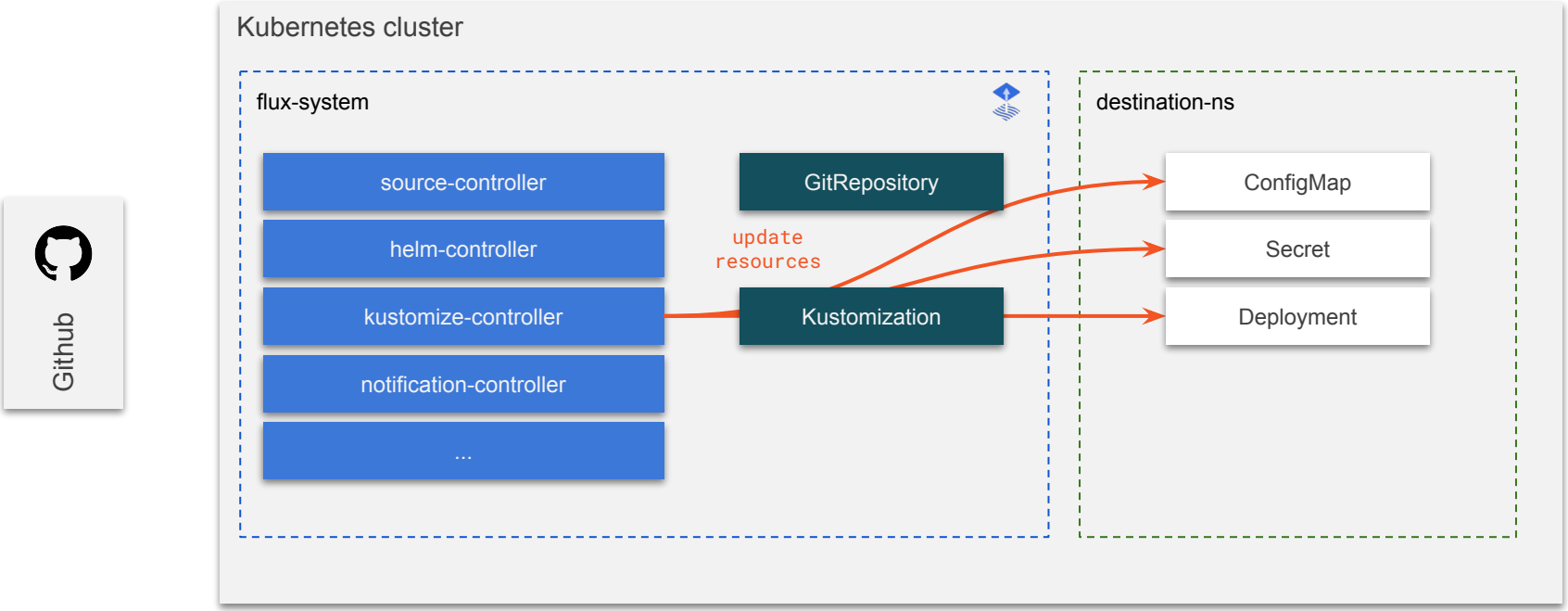
GitOps flow with FluxCD - step-by-step



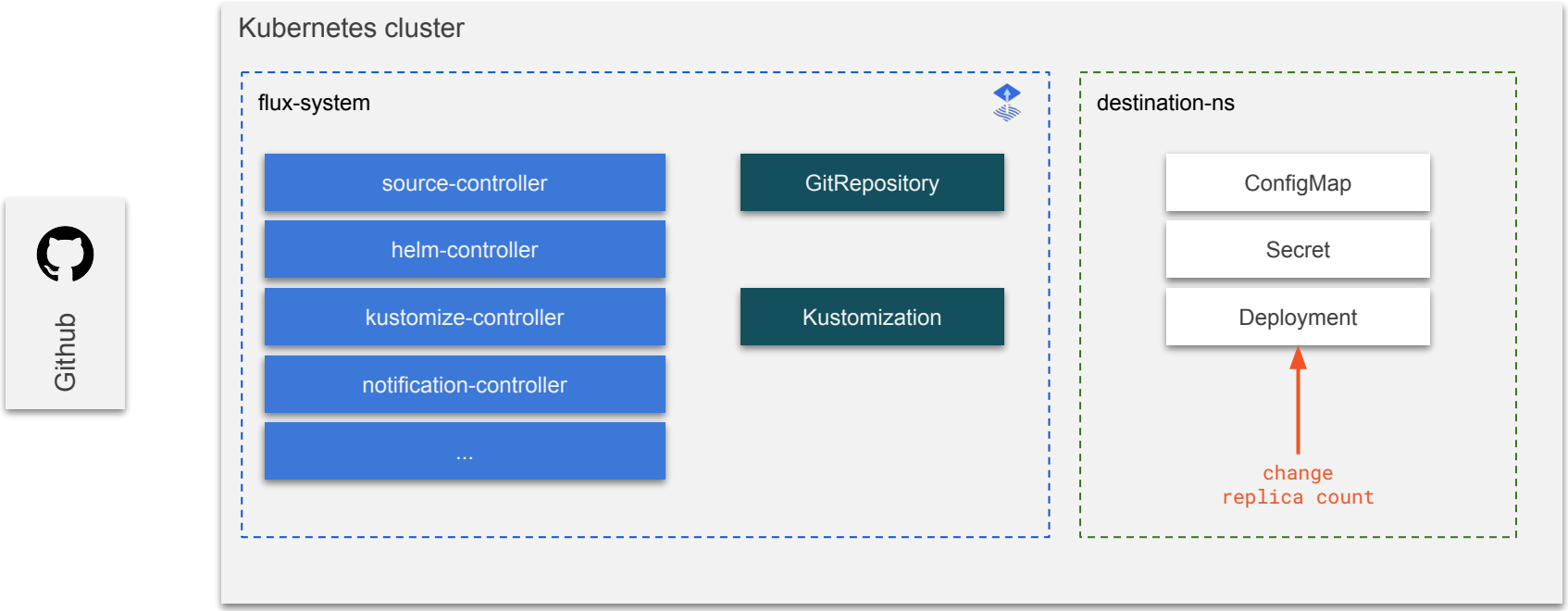
GitOps flow with FluxCD - step-by-step



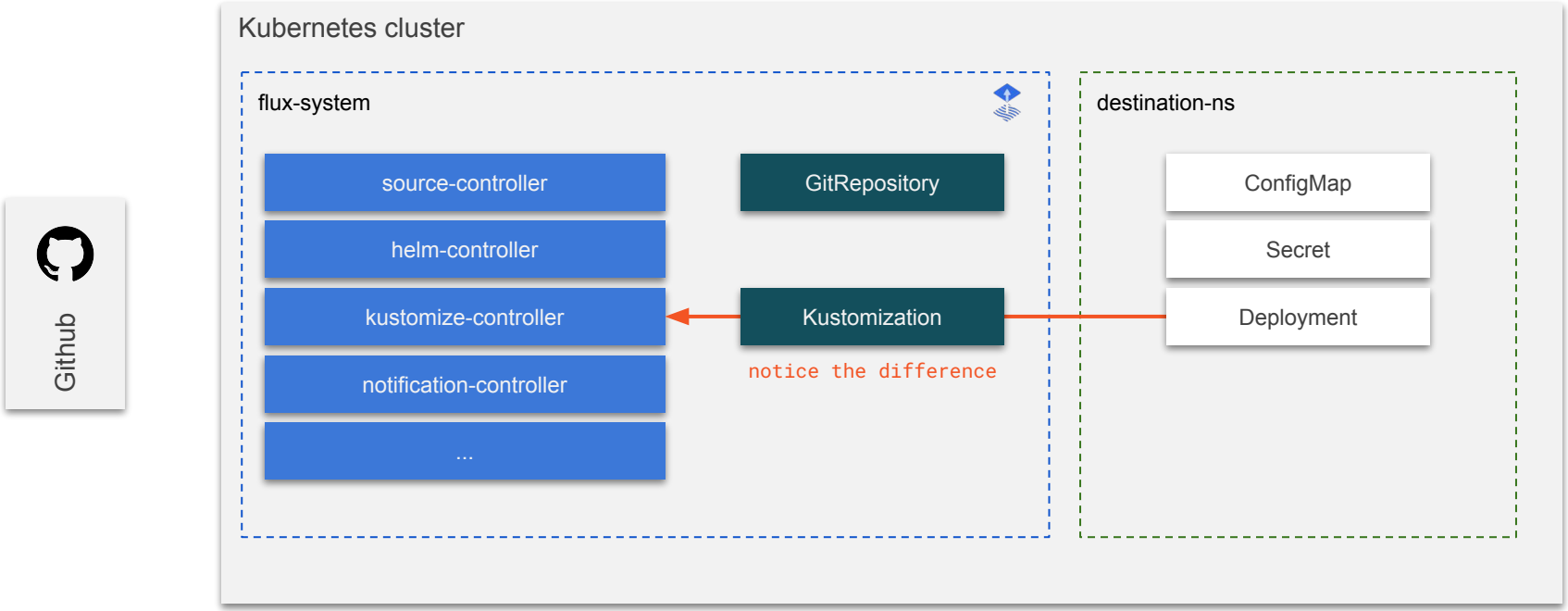
GitOps flow with FluxCD - step-by-step



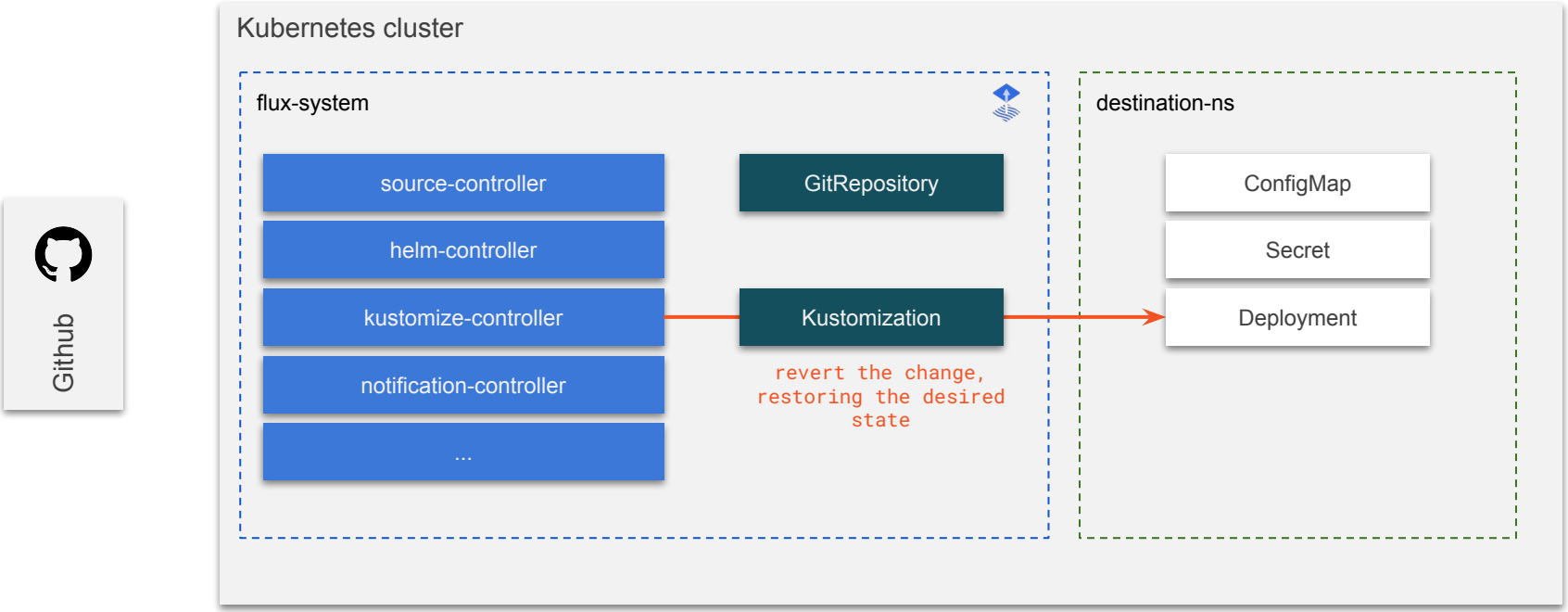
GitOps flow with FluxCD - constant reconciliation



GitOps flow with FluxCD - constant reconciliation



GitOps flow with FluxCD - constant reconciliation



What's the catch?

Not all that glitters is gold

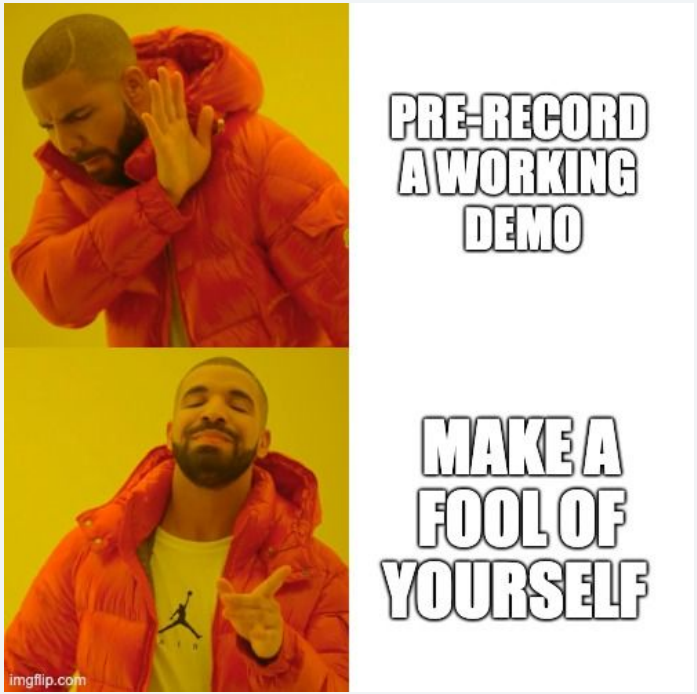
Some things still need to be figured out:

- Automatic app upgrades?
- More and more infrastructure management (K8s Cluster API, Crossplane, …)
- Environment propagation/promotion?
- Observability?
- Easier secret management?

Still "young", lots of stuff to be improved and future plans, community is very active.

Who knows how this will look like in 5 years :)

Demo time!


PRE-RECORD A WORKING DEMO
MAKE A FOOL OF YOURSELF
imgflip.com


My job here is done
But you didn't do anything

Giant Swarm

Lorenzo Soligo
slg.lnz96@icloud.com
@lollones [telegram, github]