

Kubernetes manifests management & operation in Mercari

@b4b4r07 (Apr 22, 2019) / Kubernetes meetup tokyo #18

BABAROT / @b4b4r07

Mercari, Inc.
SRE, Microservices Platform



Blog / tellme.tokyo

mercari.com

MERCARI

What are you looking for?

Sell now Sign in

Women Men Toys Kids Home Vintage Beauty Electronics Sports Handmade Other

The Selling App.

Sell or buy. Almost anything.

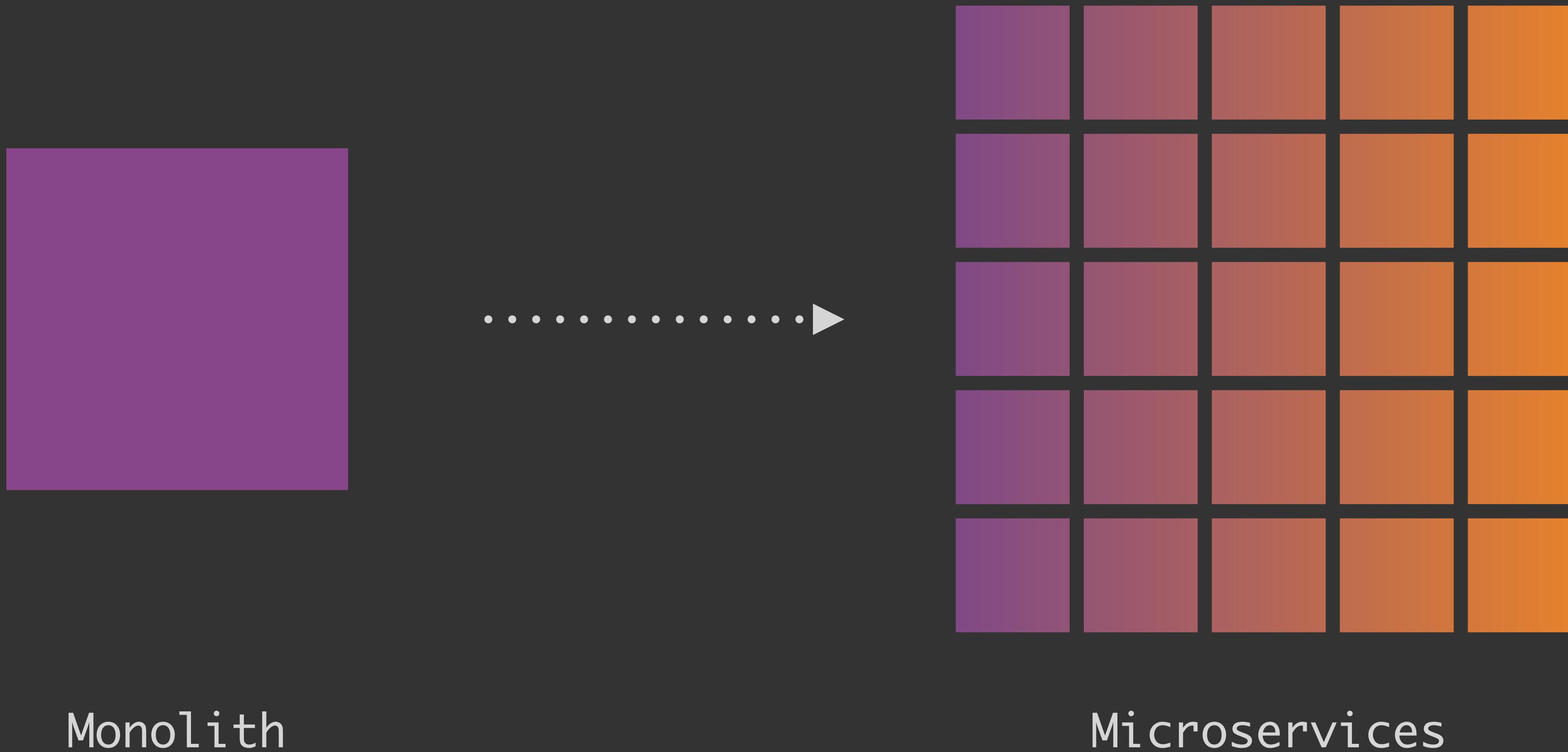
Sell now

Sell it.
List in minutes. Take a few photos. Add a description. Set your price.

Ship it.
No meetups. Printable shipping label emailed to seller.

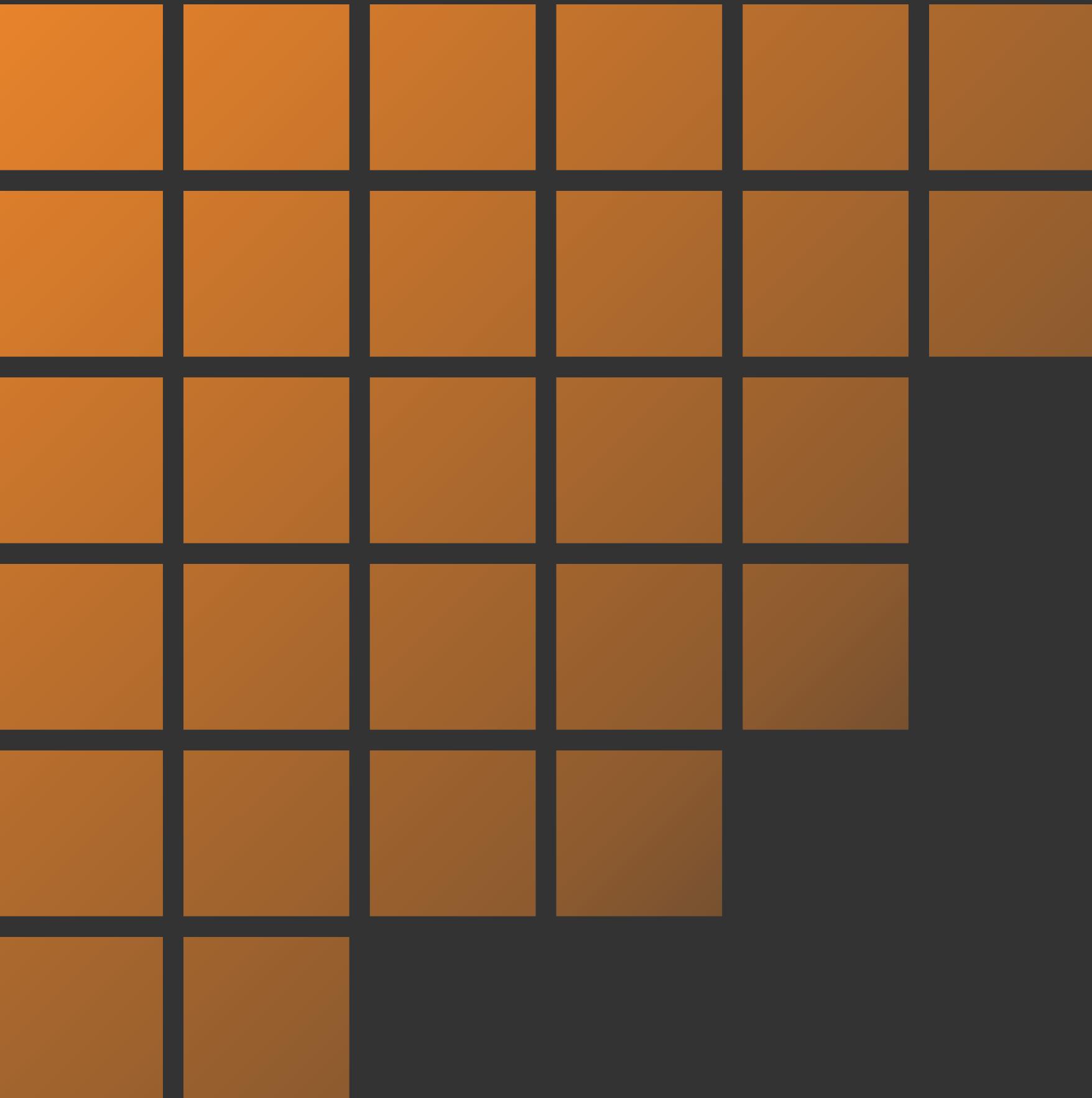
Get paid.
Listings are free. Flat 10% selling fee charged when sale completes.

Our current status



Our current status

100+ microservices

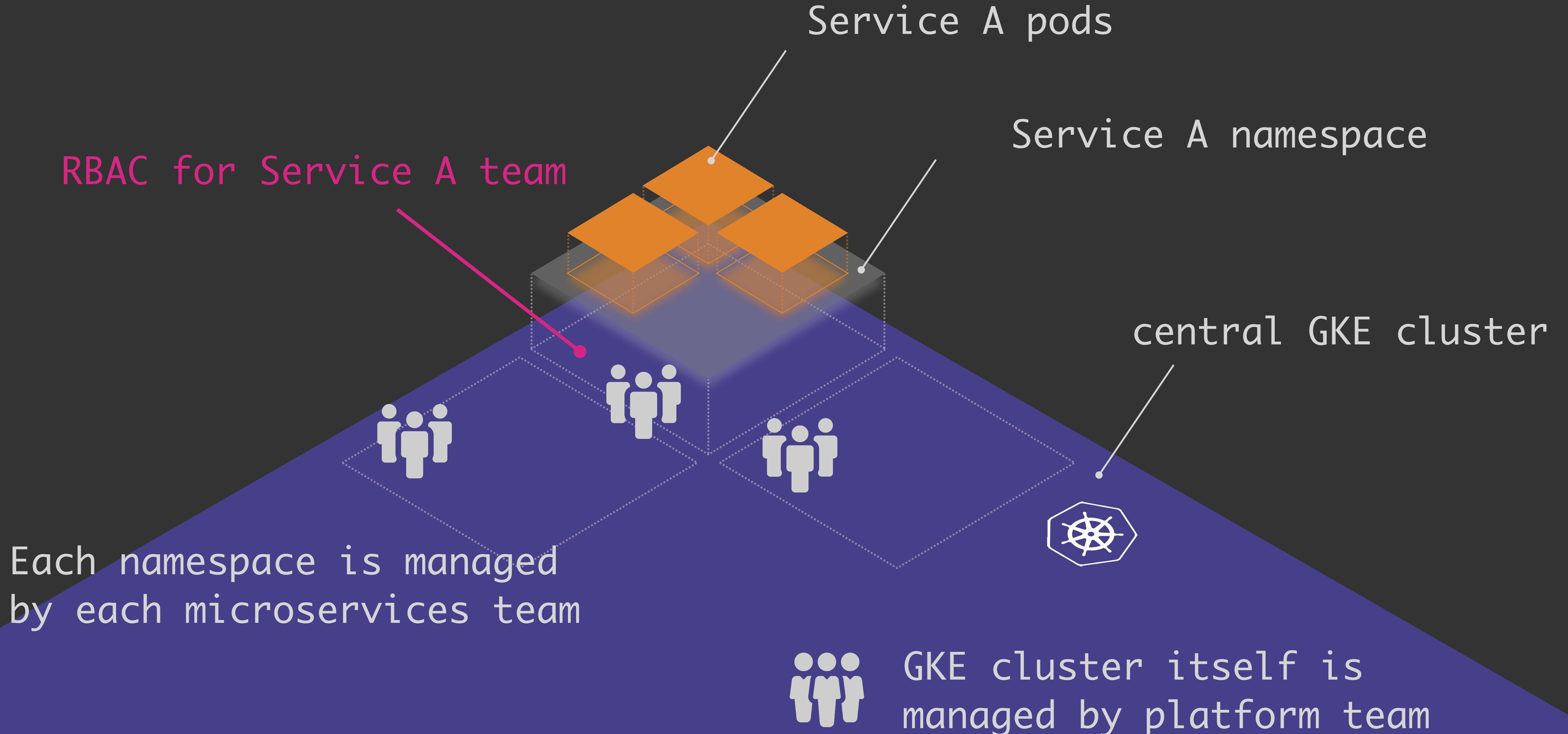


Our current status

200+ contributors



Our current status



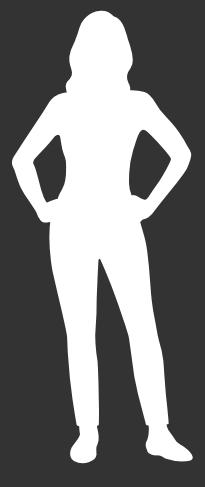
Agenda

- 1. Kubernetes YAML
 - 1. GitHub Pull Requests
 - 2. GitOps
 - 3. Monorepo
 - 4. Directories
- 2. Repository Ecosystem
- 3. Recap

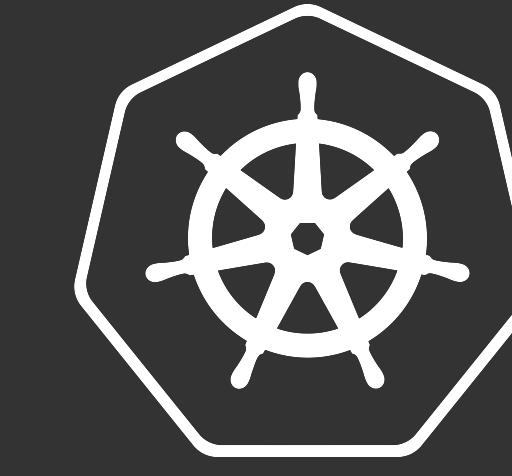
Kubernetes YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  namespace: x-echo-jp-dev
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
```

How do you manage YAML and operate it
to your Kubernetes Clusters?



Write

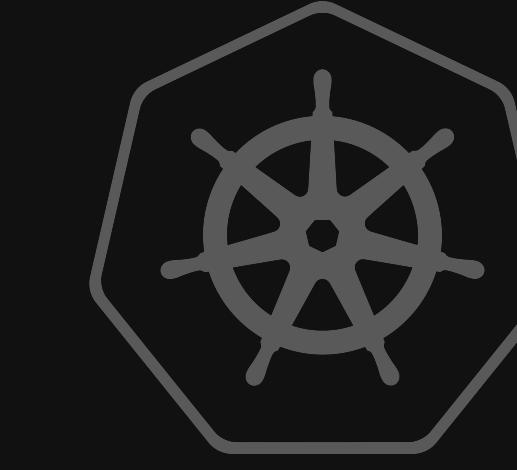


Apply

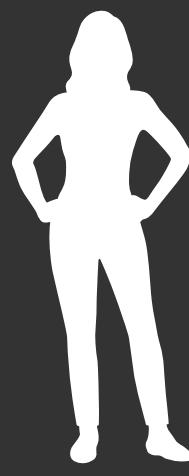
Some points...



Write



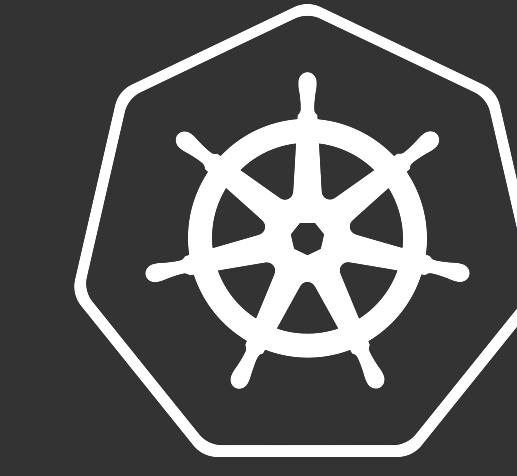
Apply



Write



Apply

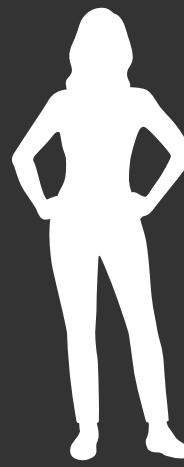


How should we review it?



How should we apply it?

How should we control it?



Write



How should we manage it?

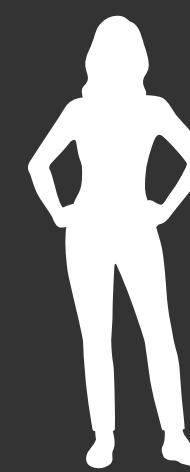


Apply

How should we review it?

How should we apply it?

YAML Management

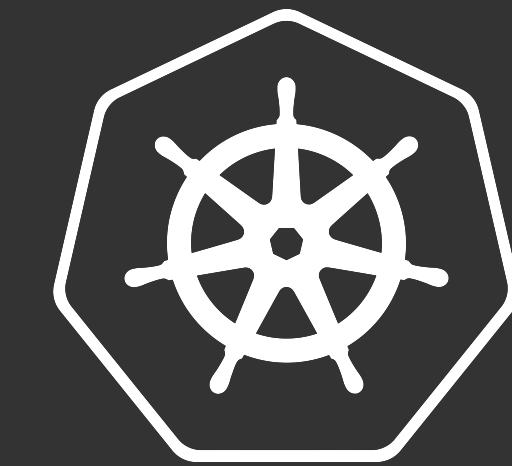


How should we control it?

Write



How should we manage it?



Apply

How should we review it?

YAML Operations

How should we apply it?

YAML Management

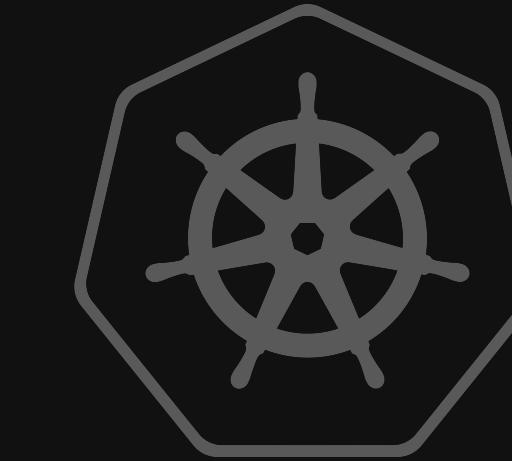
How should we control it?



How should we manage it?



Write



Apply

Our practices are...

How should we review it?

YAML Operations

How should we apply it?

4. Directories

~~How should we control it?~~



Write



~~How should we review it?~~

1. Pull Requests

3. Monorepo

~~How should we manage it?~~



Apply

~~How should we apply it?~~

2. GitOps w/ kubectl

.....►

.....►

Let's see the each details

1. GitHub Pull Requests



Kubernetes Object Management

Imperative
commands

Imperative
object
configuration

Declarative
object
configuration

Kubernetes Object Management

```
kubectl run nginx  
--image nginx
```

Imperative
object
configuration

Declarative
object
configuration

Kubernetes Object Management

```
kubectl run nginx  
--image nginx
```

```
kubectl create  
-f nginx.yaml
```

Declarative
object
configuration

Kubernetes Object Management

```
kubectl run nginx  
--image nginx
```

```
kubectl create  
-f nginx.yaml
```

```
kubectl apply  
-f manifests/
```

Kubernetes Object Management

- It can be stored in VCS such as Git.
- It can integrate with processes such as reviewing changes.
- It has better support for operating on directories and automatically detecting operation types per-object.

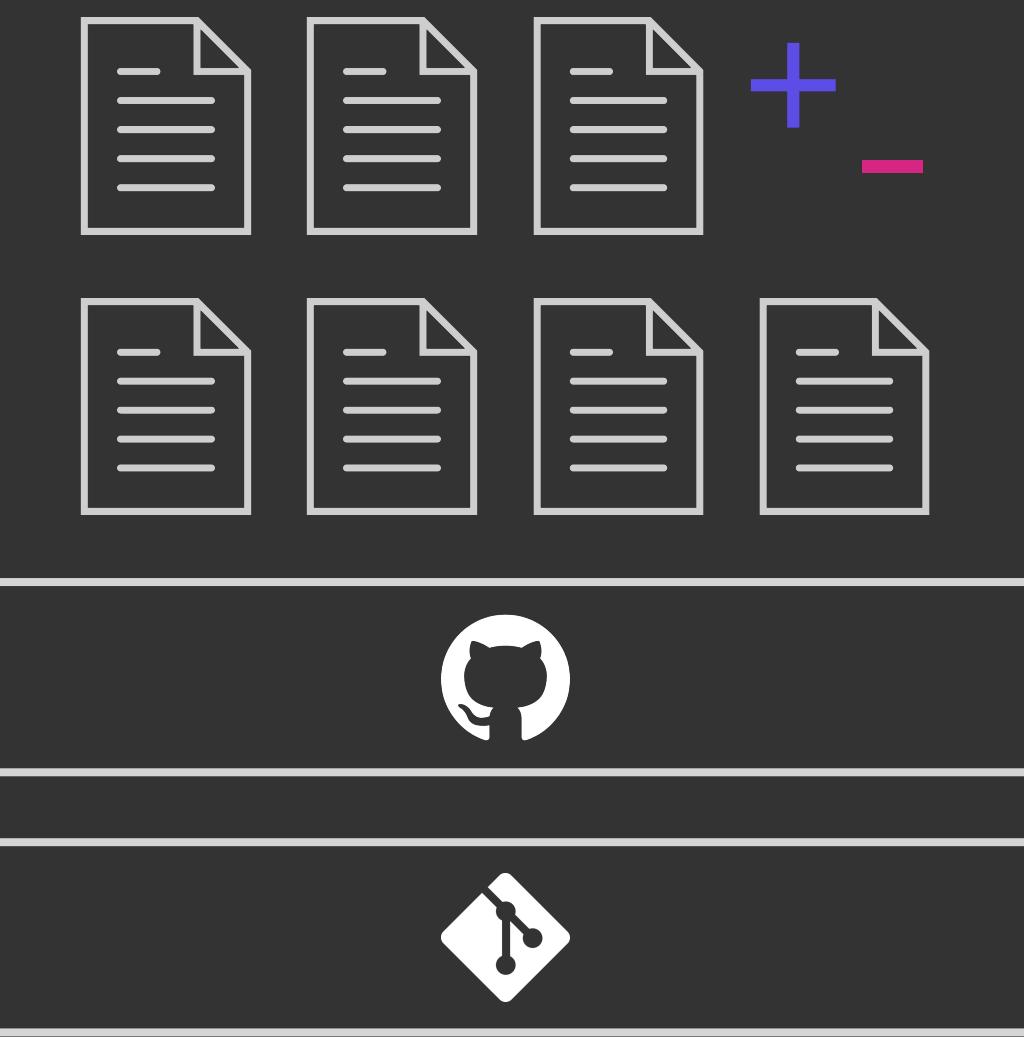


Declarative
object
configuration

Pull Requests

- Easy to review
- Easy to track operations and changes
- Easy to recover from unexpected things
- Just reverting

2. GitOps



GitOps - Operations by Pull Requests

A screenshot of a web browser window displaying a blog post from [weave.works](https://www.weave.works). The browser has a light gray header with standard OS X-style controls (red, yellow, green buttons, back/forward, search, refresh, and tabs). The main content area shows the Weaveworks logo at the top left, followed by the date "AUGUST 07, 2017". Below the date are three blue category links: "Gitops", "Kubernetes", and "Product features". The main title of the post is "GitOps - Operations by Pull Request". A horizontal line separates the title from the content. The first paragraph of the post reads: "At Weaveworks, developers are responsible for operating our Weave Cloud SaaS. ‘GitOps’ is our name for how we use developer tooling to drive operations. This post talks about GitOps, which is 90% best practices and 10% cool new stuff that we needed to build. Fair warning: this is what works for us, and dear reader, you may disagree." The second paragraph continues: "Git is a part of every developer’s toolkit. Using the practices outlined in this post, our developers operate Kubernetes via Git. In fact, we manage and monitor all of our applications and the whole ‘cloud native stack’ using GitOps. It feels natural and less intimidating to learn, and the tools themselves are very simple." To the right of the text, there is a large orange circular icon containing a white speech bubble with a smiley face.

weaveworks

AUGUST 07, 2017

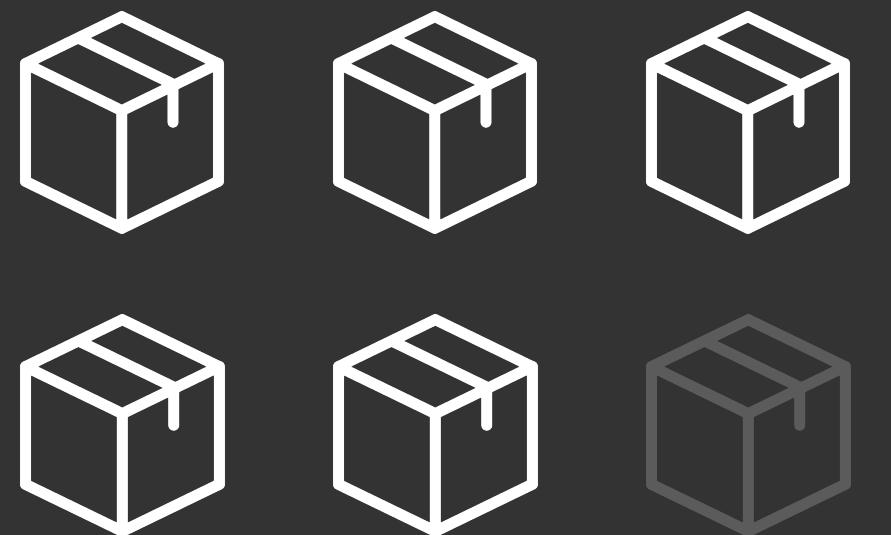
Gitops | Kubernetes | Product features

GitOps - Operations by Pull Request

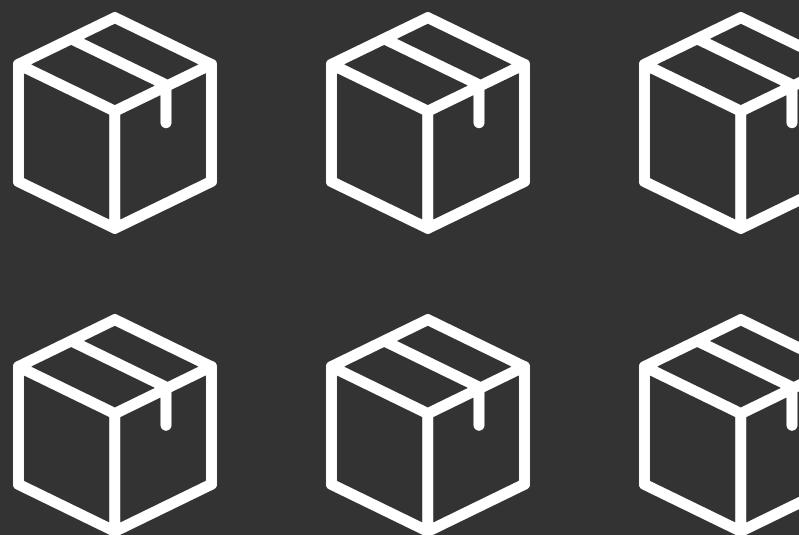
At Weaveworks, developers are responsible for operating our Weave Cloud SaaS. “GitOps” is our name for how we use developer tooling to drive operations. This post talks about GitOps, which is 90% best practices and 10% cool new stuff that we needed to build. Fair warning: this is what works for us, and dear reader, you may disagree.

Git is a part of every developer’s toolkit. Using the practices outlined in this post, our developers operate Kubernetes via Git. In fact, we manage and monitor all of our applications and the whole ‘cloud native stack’ using GitOps. It feels natural and less intimidating to learn, and the tools themselves are very simple.

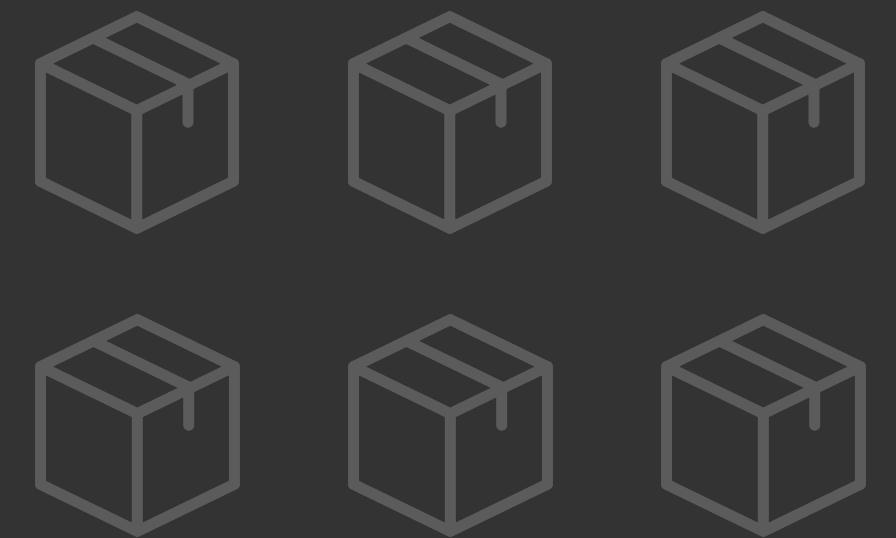
Start 6 servers



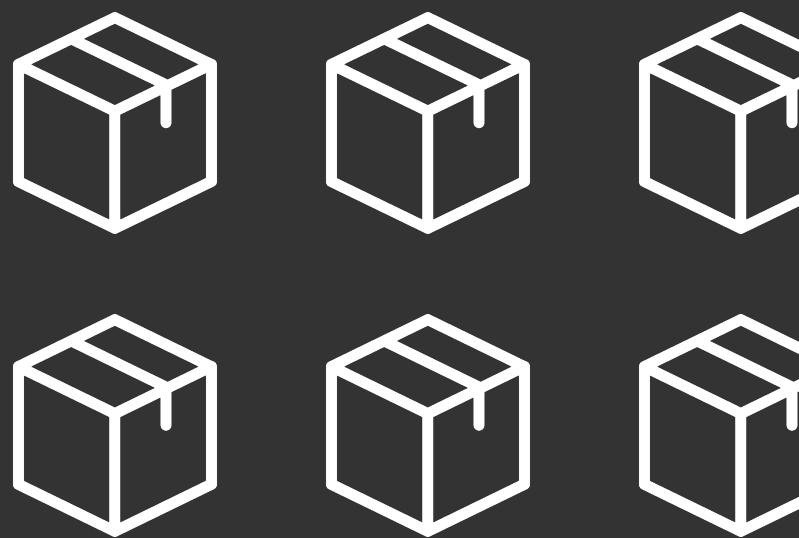
There are 6 servers



Start 6 servers



There are 6 servers



Let's see Weaveworks case



Kubernetes Cluster



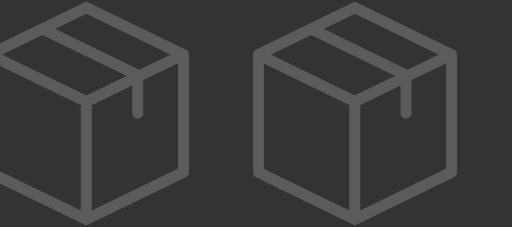
GitHub Repository



Live Objects



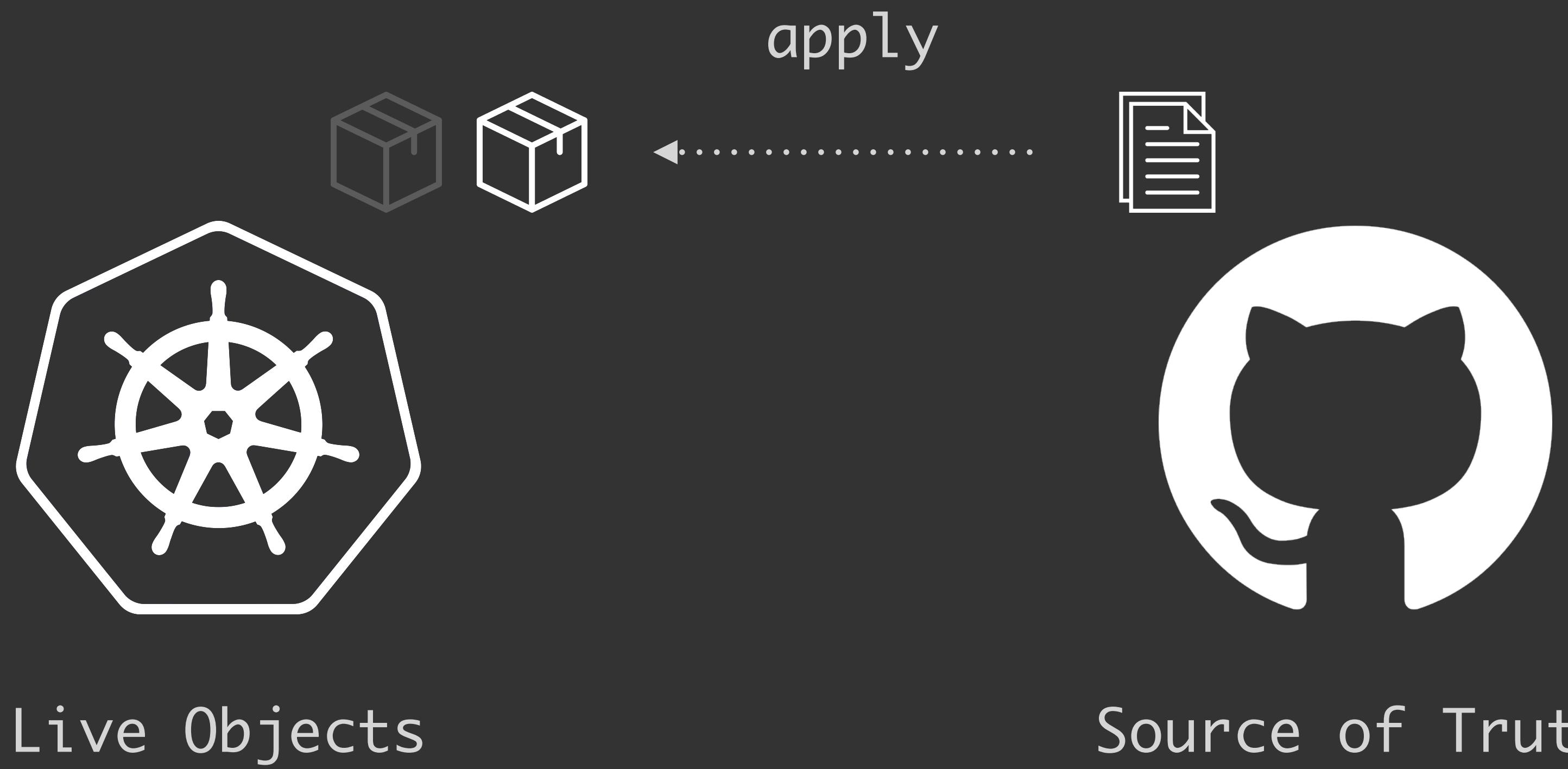
Source of Truth

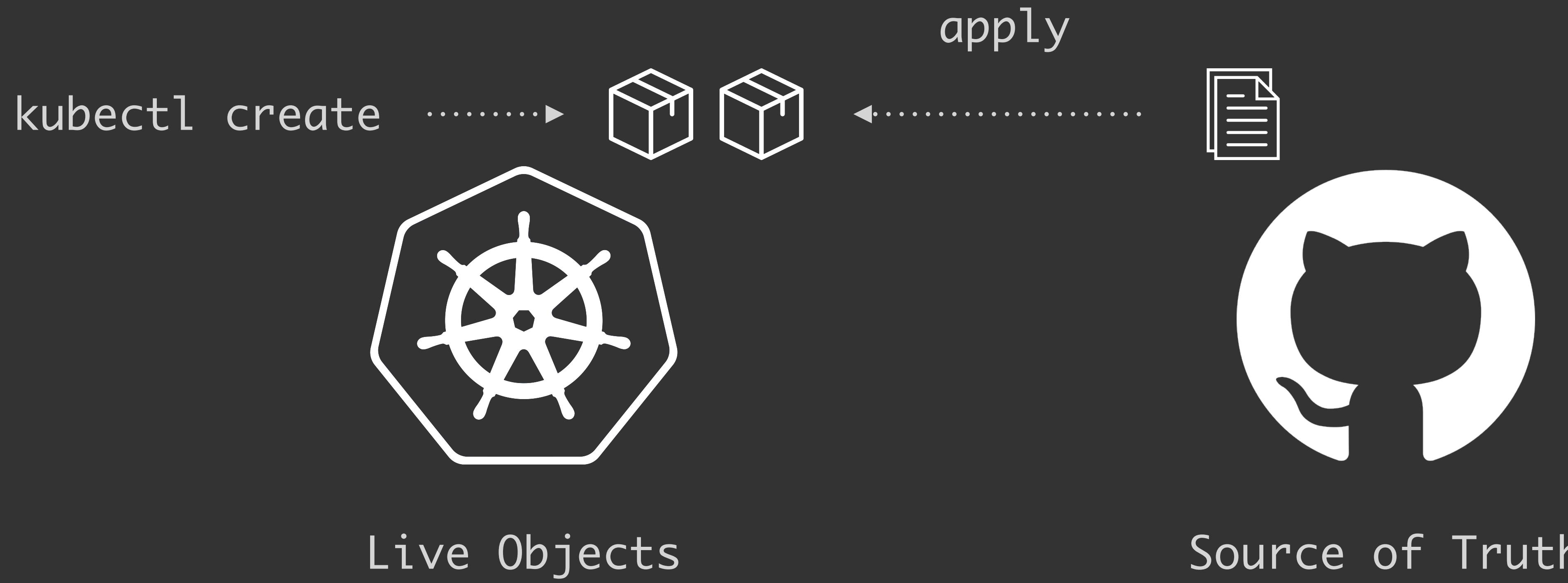


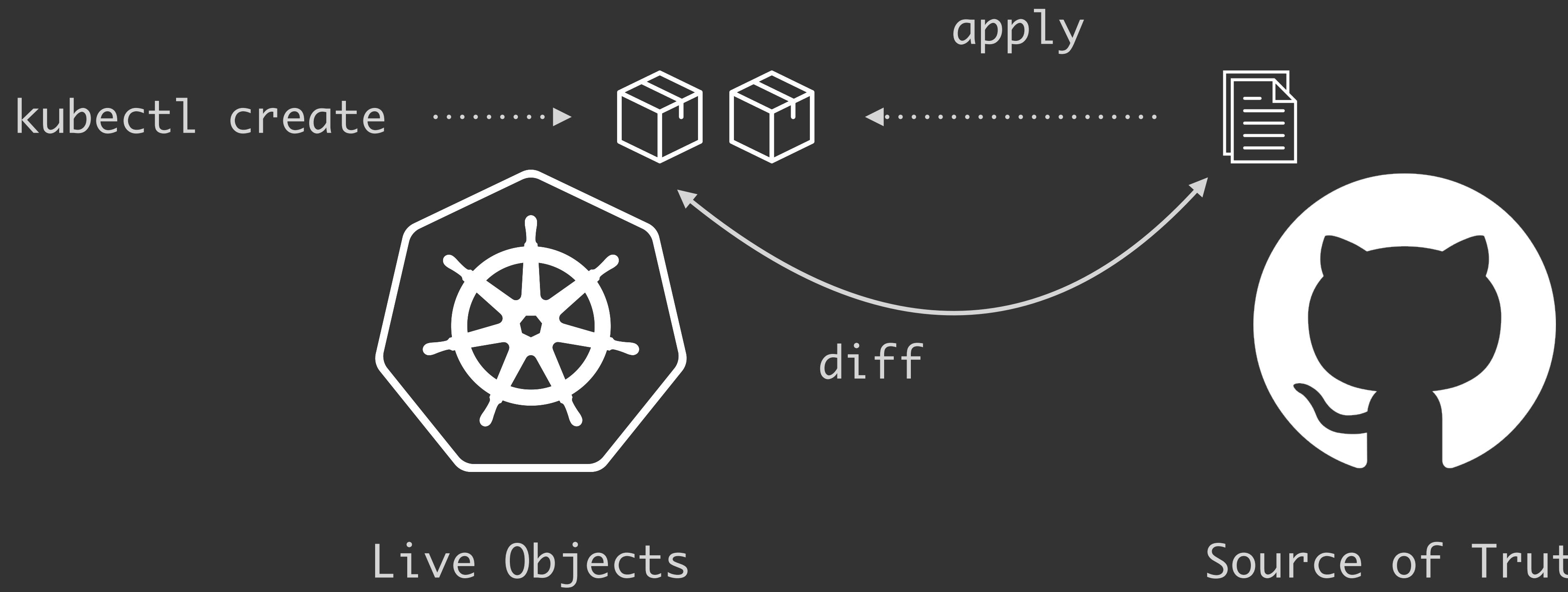
Live Objects

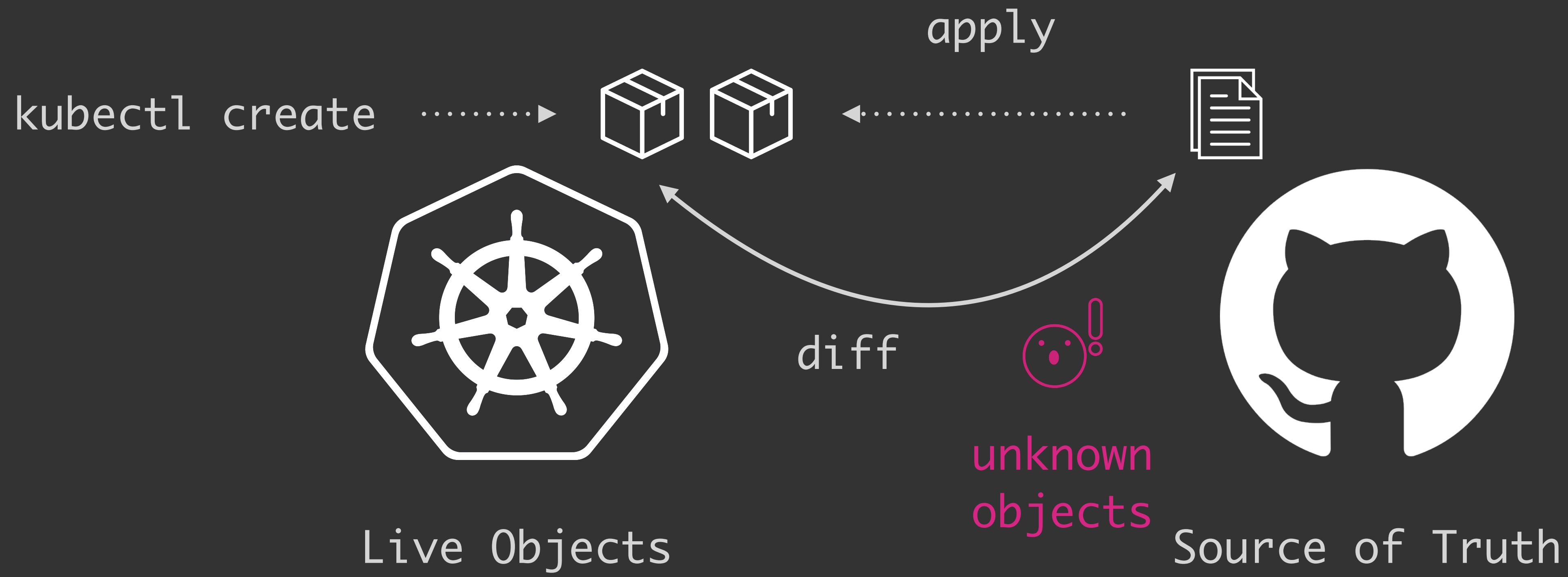


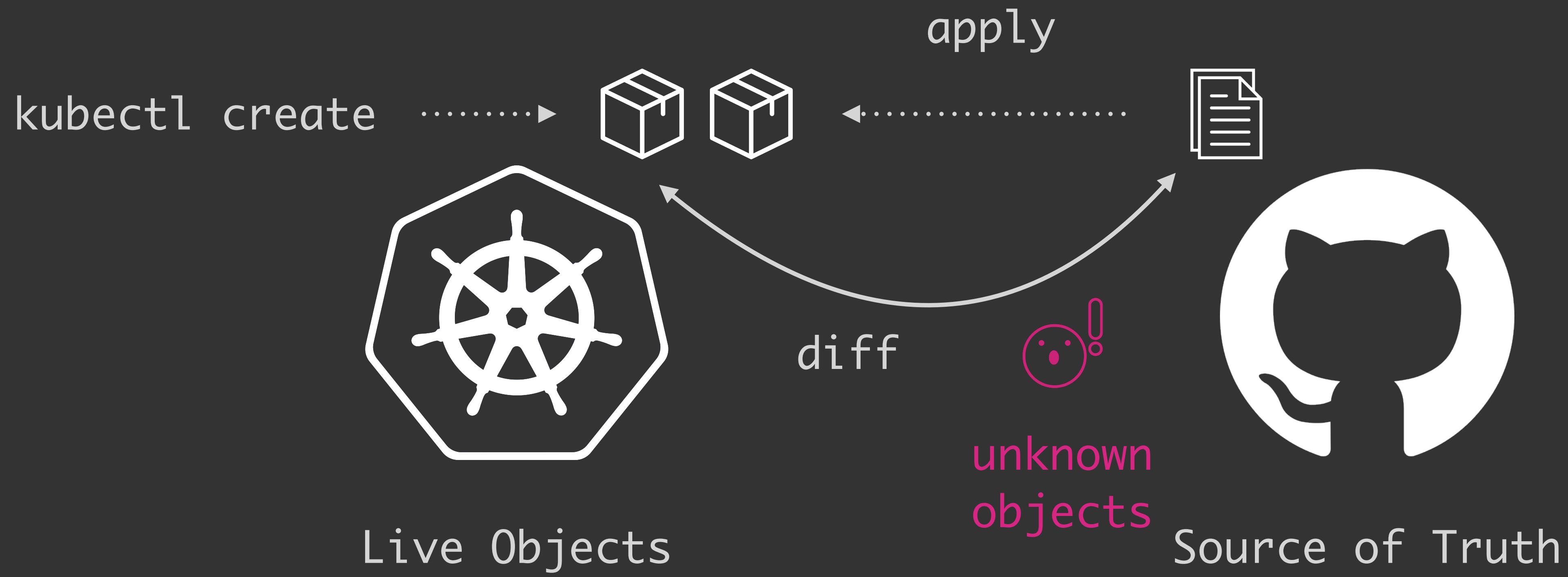
Source of Truth





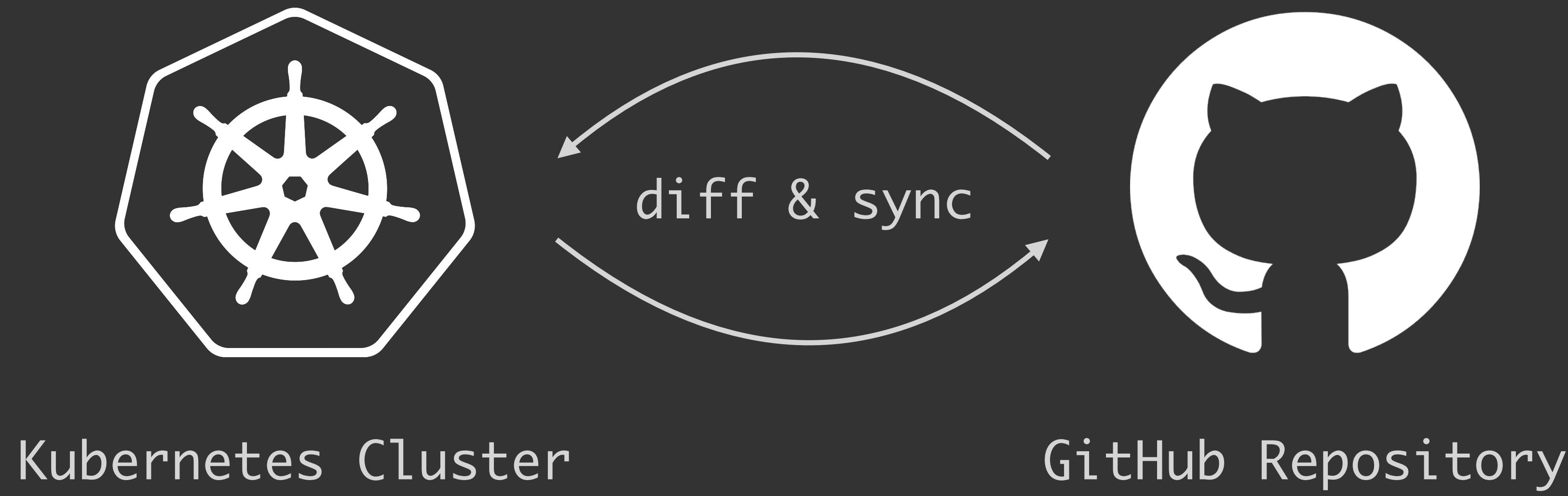




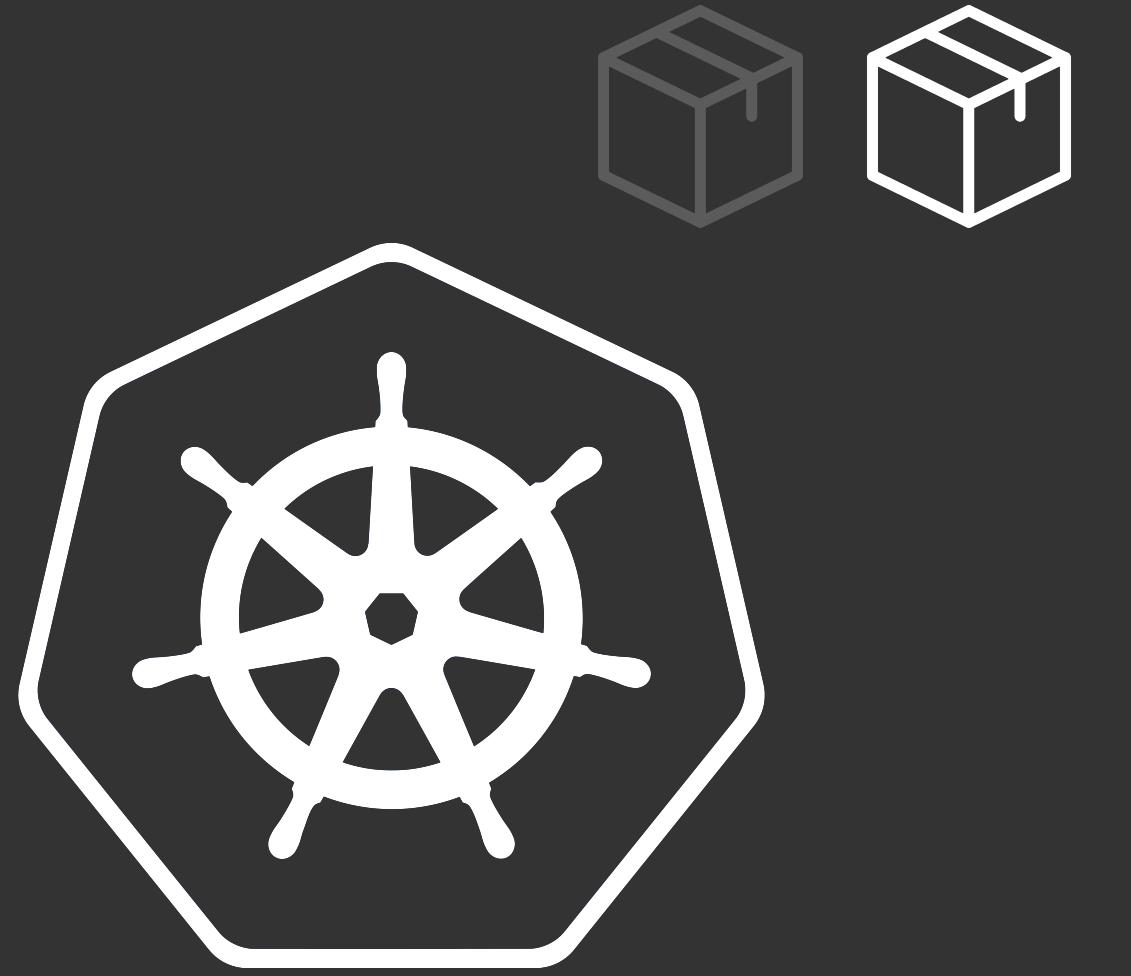


Git is the source of truth

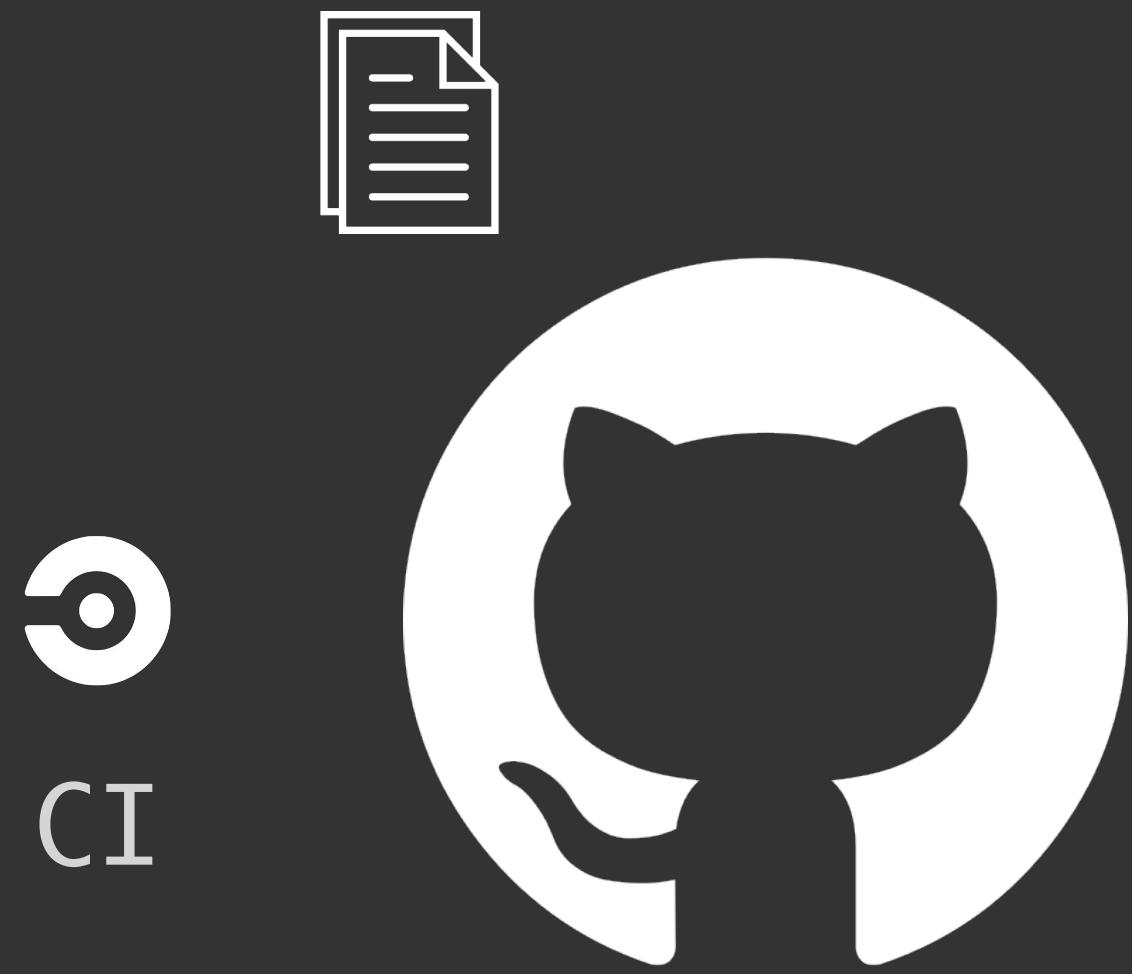
Pull strategy



Let's see our case

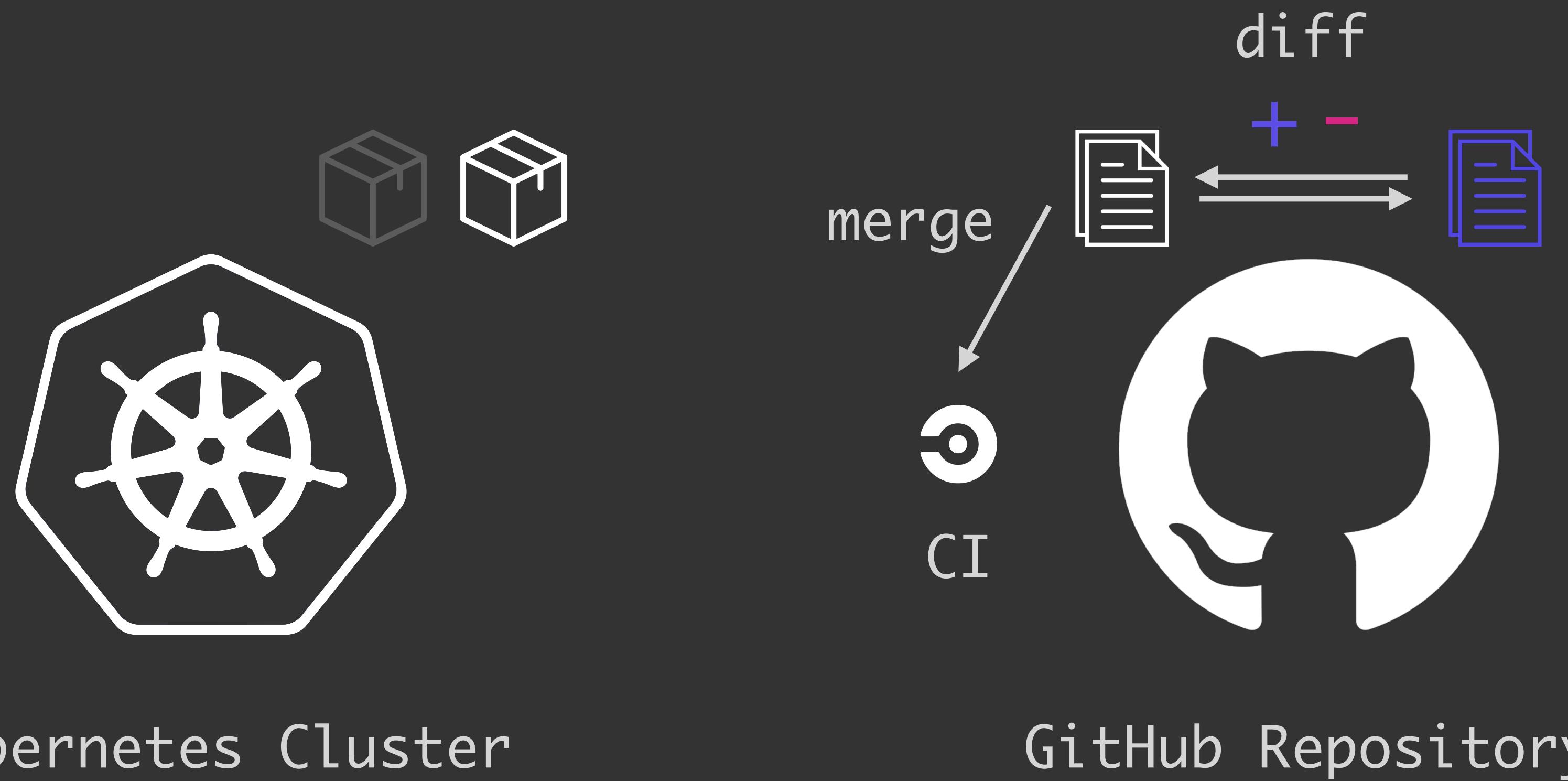


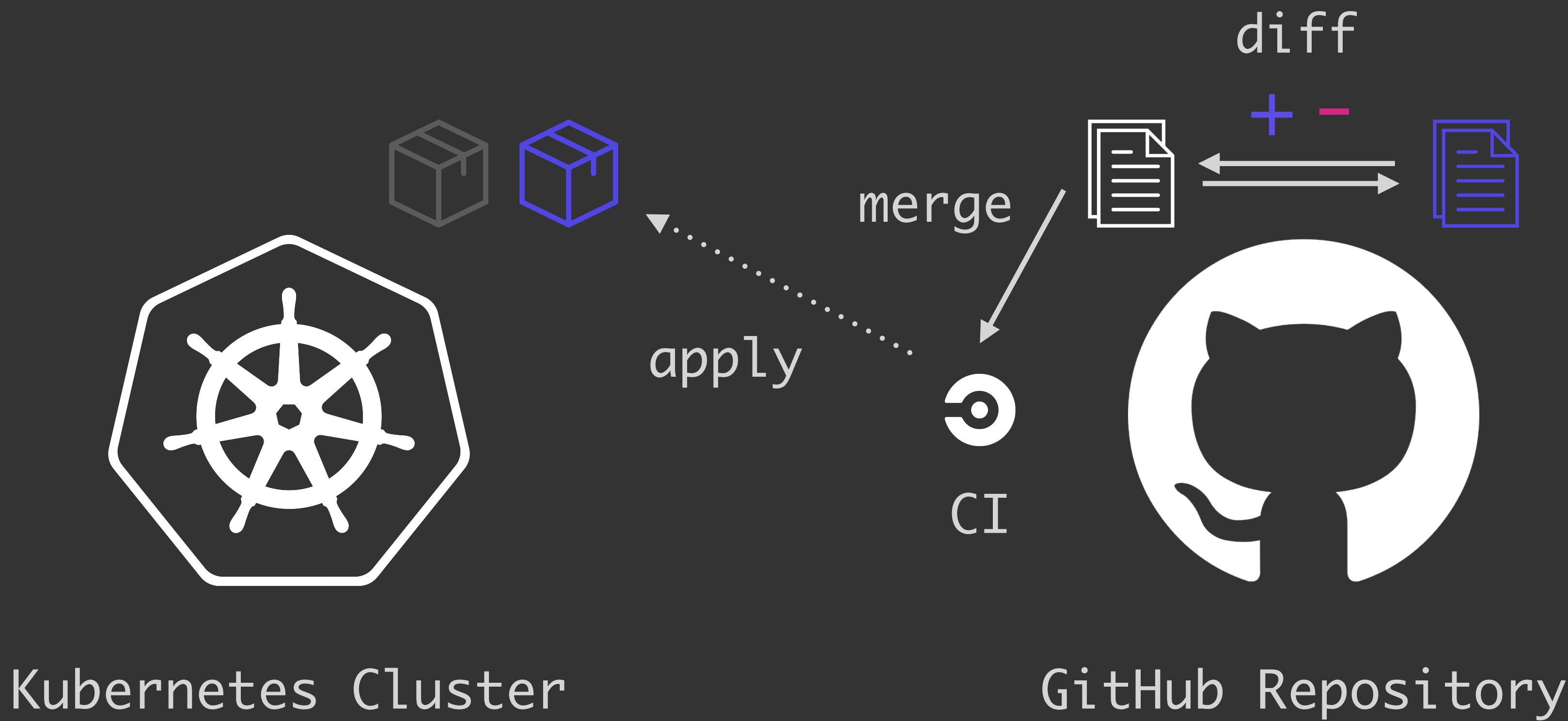
Kubernetes Cluster

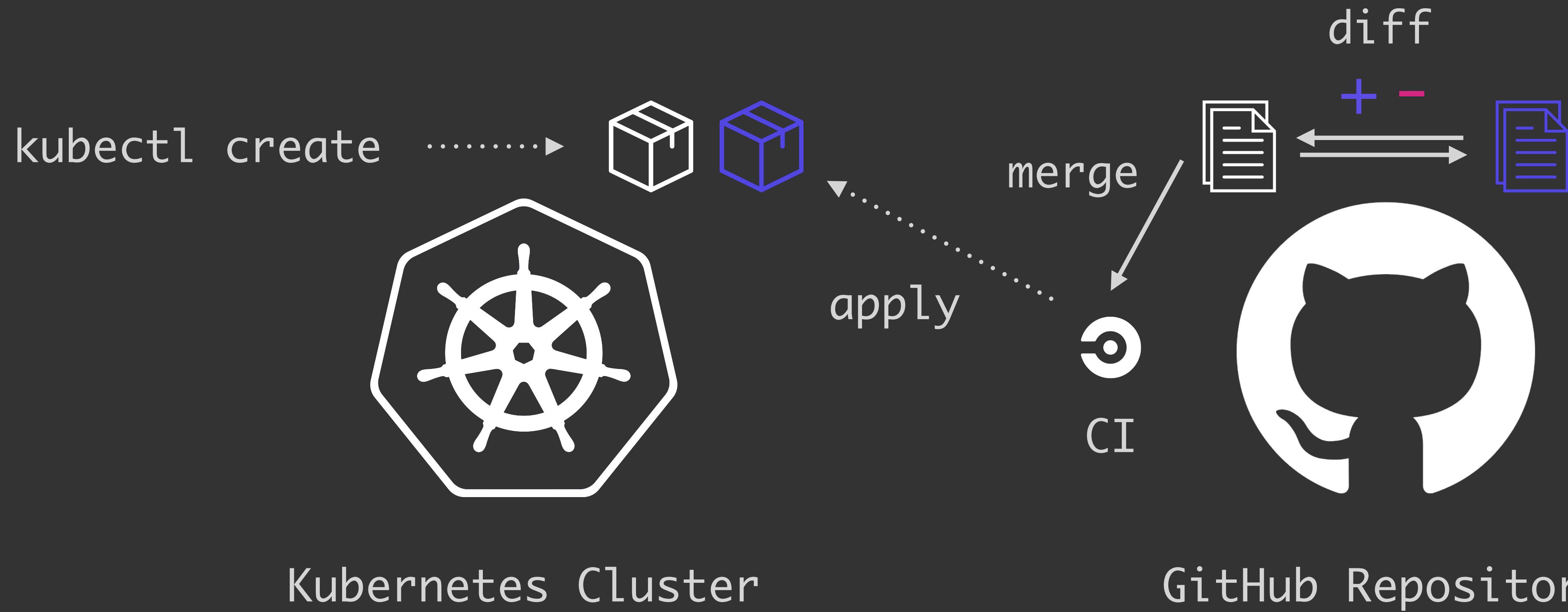


GitHub Repository

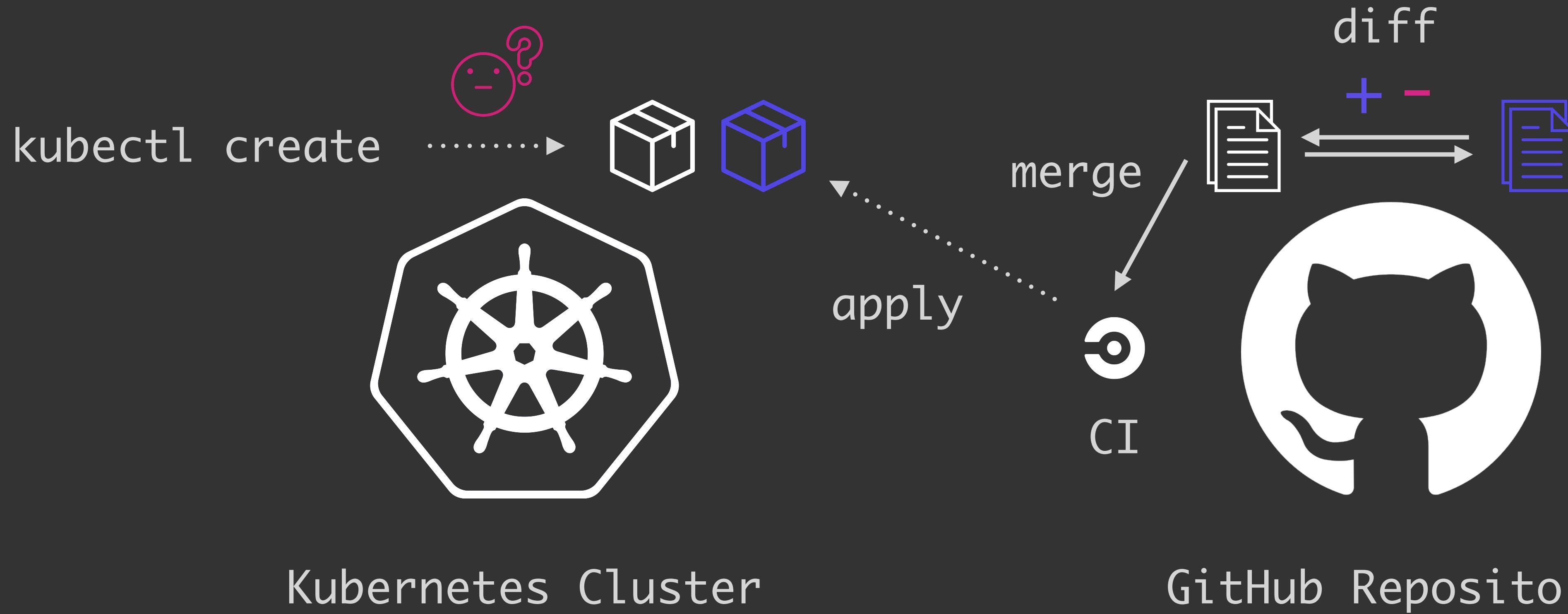




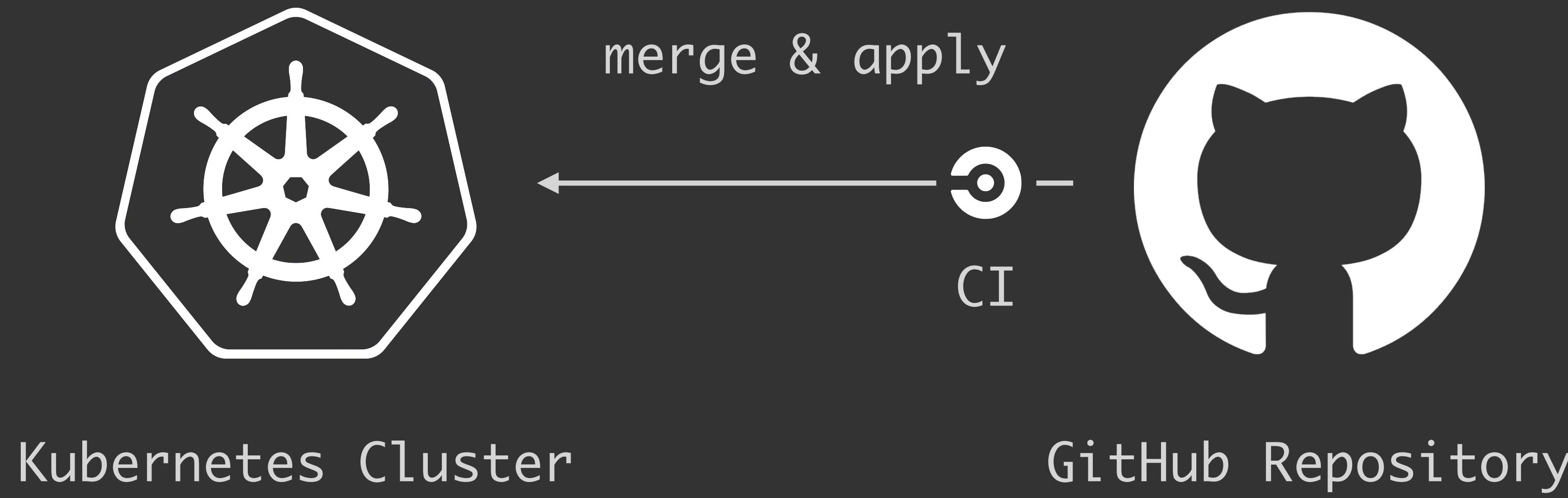




not implemented yet
(described later)

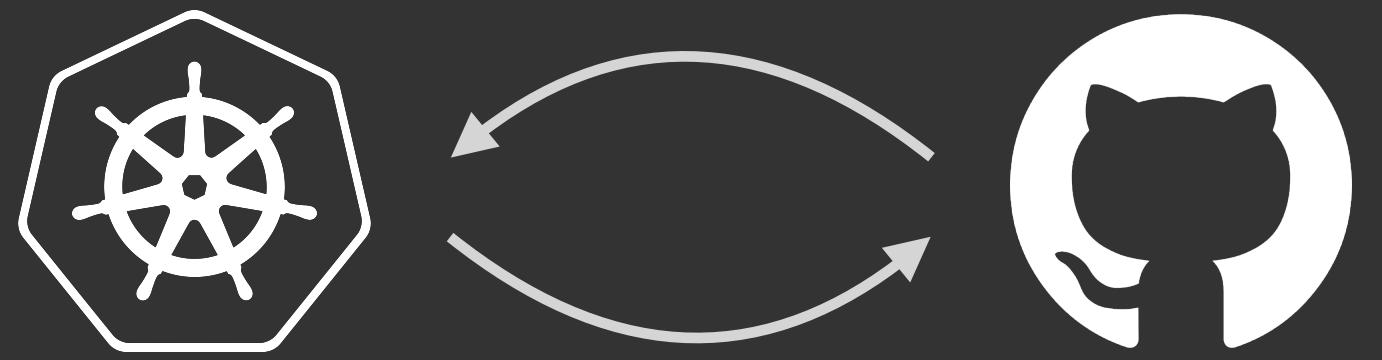


Push strategy



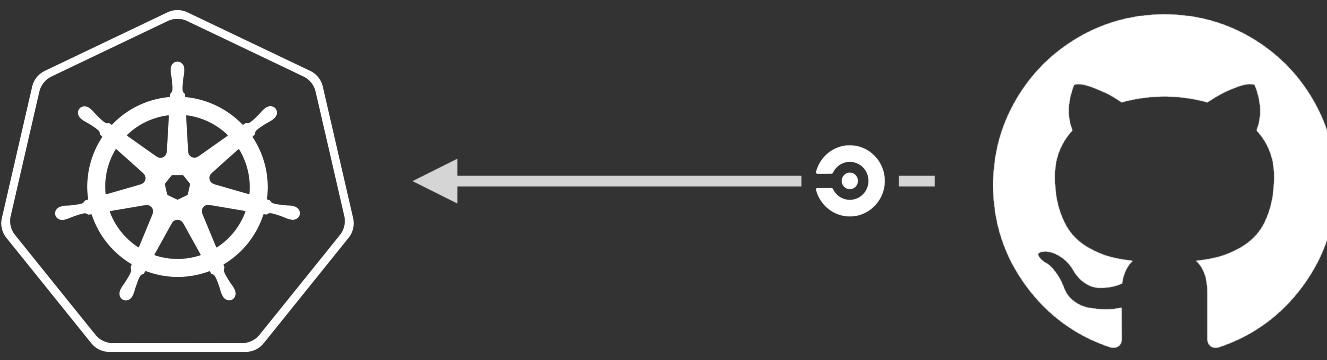
Let's see the each differences

Pull (Weaveworks case)



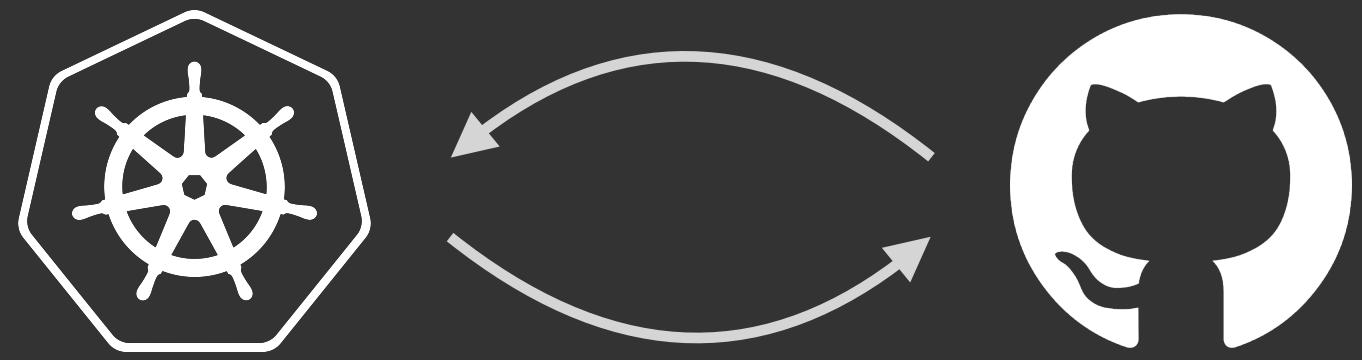
- Git can be the single source of truth
- Bit difficult to implement the sync pipeline

Push (Mercari case)



- Difficult to find divergence from the source
- A common way of applying the changes

Pull (Weaveworks case)



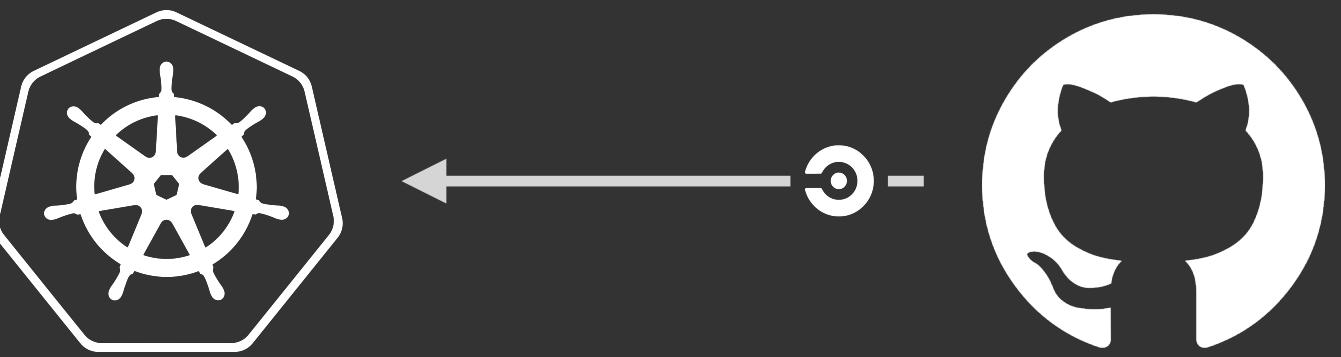
- Git can be the single source of truth
- Bit difficult to implement the sync pipeline

A screenshot of a web browser displaying a blog post from weave.works. The title of the post is "Kubernetes anti-patterns: Let's do GitOps, not CI/Ops!". The post is dated July 17, 2018, and is categorized under "Gitops". The text discusses a common anti-pattern in Kubernetes deployment automation where CI/CD tutorials recommend using Git as the single source of truth, despite it being difficult to implement a sync pipeline. A small orange circular icon with a white document symbol is visible in the bottom right corner of the post area.

We know they recommend "Pull" strategy

Push (Mercari case)

Why do we choose "Push" strategy?



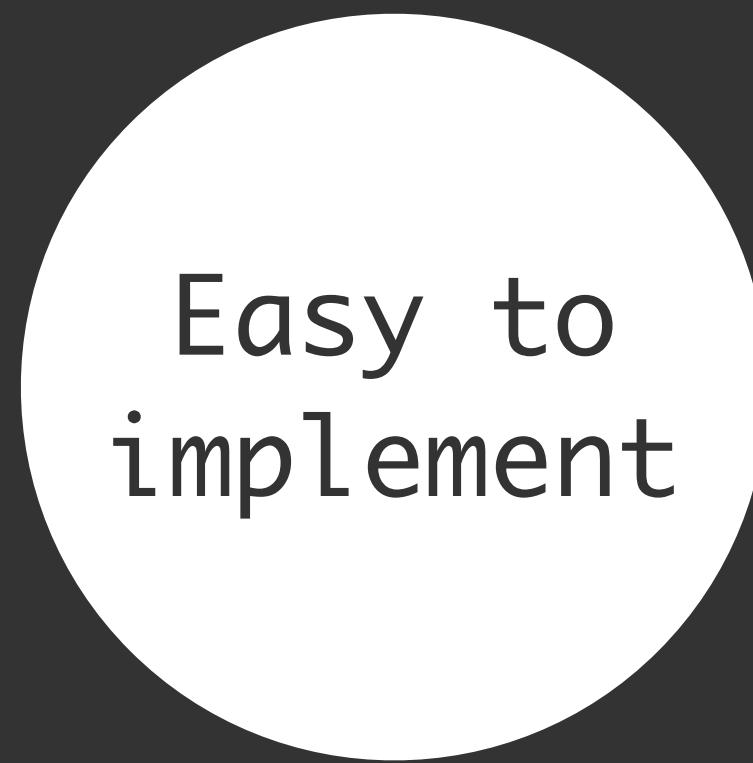
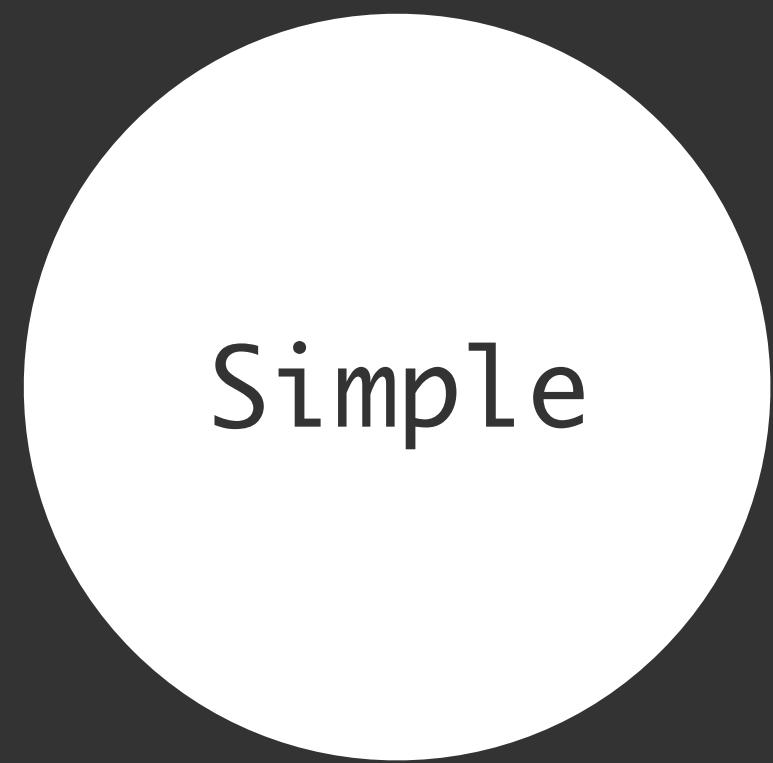
Simple

Easy to
implement

Enough
to start
firstly

- Difficult to find divergence from the source
- A common way of applying the changes

Why do we choose "Push" strategy?



Spinnaker

Another most bigger reason is...

Why using Spinnaker also?

- Our CI pipeline which runs "kubectl apply" based on the changes is triggered by merging pull requests
- However, in some resources (Job etc), we want to apply in our timing
- Spinnaker "Provider V2" can handle Kubernetes manifests declaratively



Spinnaker

SPINNAKER Search Projects Applications

mercari-echo-jp PIPELINES INFRASTRUCTURE TASKS CONFIG

CONCURRENT EXECUTIONS

EXPECTED ARTIFACTS (2)

AUTOMATED TRIGGERS (1)

PARAMETERS

NOTIFICATIONS

DESCRIPTION

Expected Artifacts

Declare artifacts your pipeline expects during execution in this section.

Match against

A file stored in git, hosted by GitHub.

File path

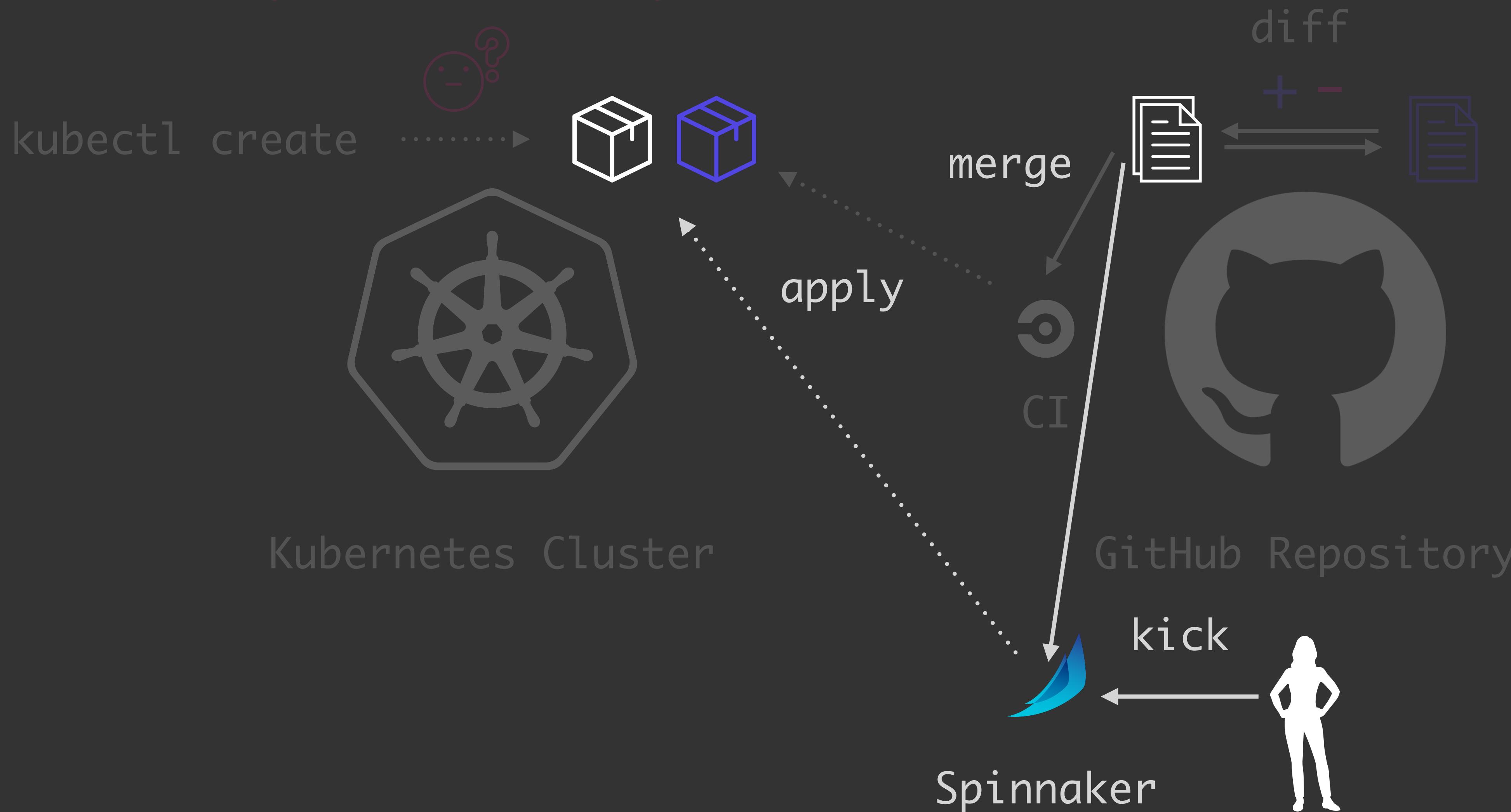
If missing Use Prior Execution Use Default Artifact

Default artifact A file stored in git, hosted by GitHub.

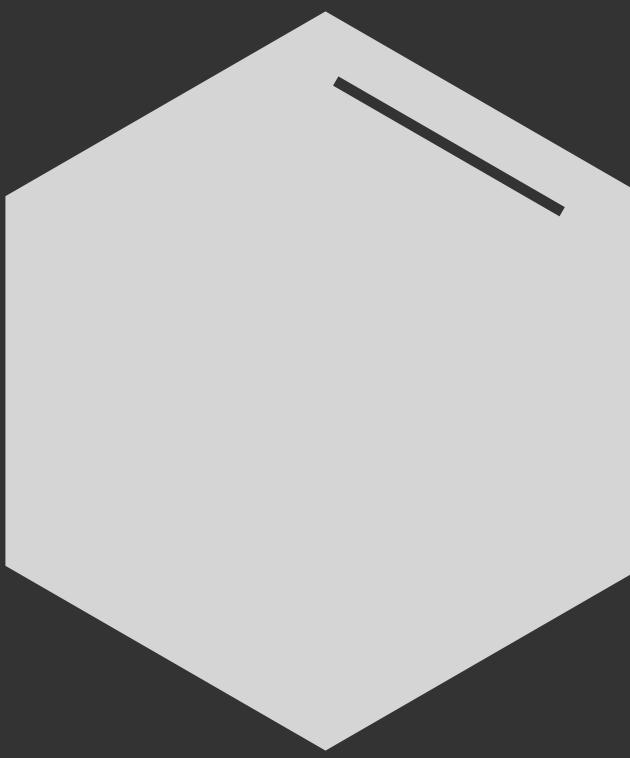
Content URL Commit/Branch

⚡ Revert

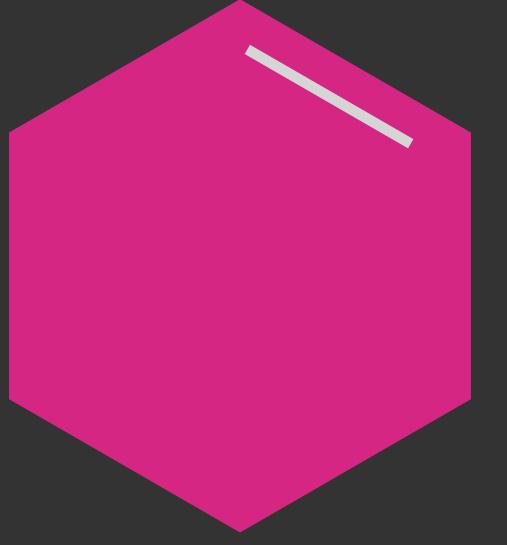
not implemented yet
(described later)



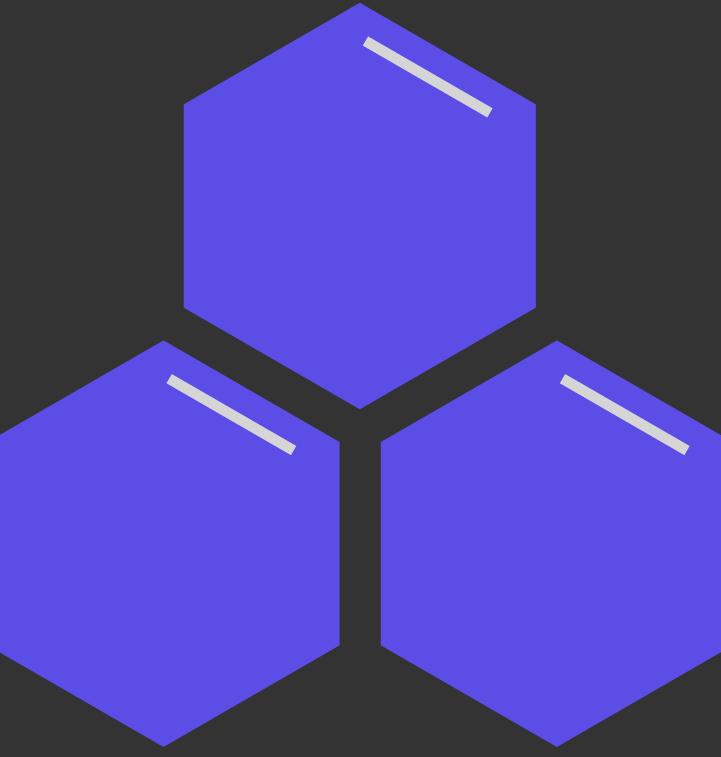
3. Monorepo



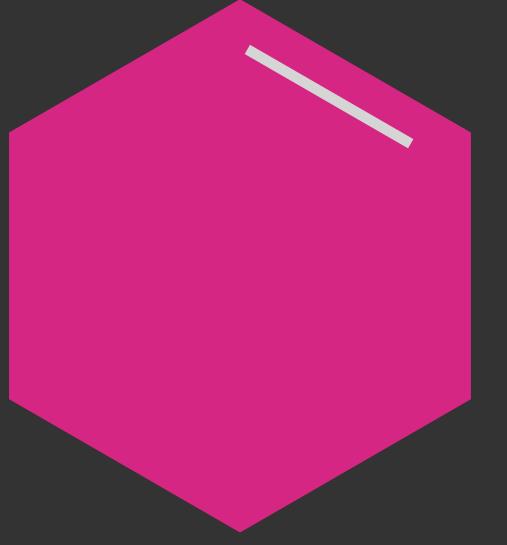
Two type of repository styles



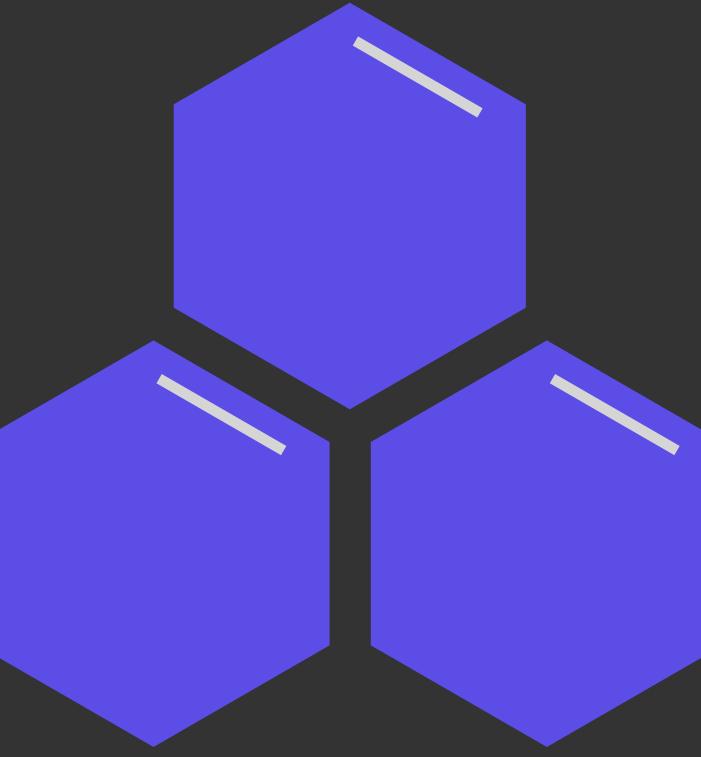
One repository



Multiple repositories

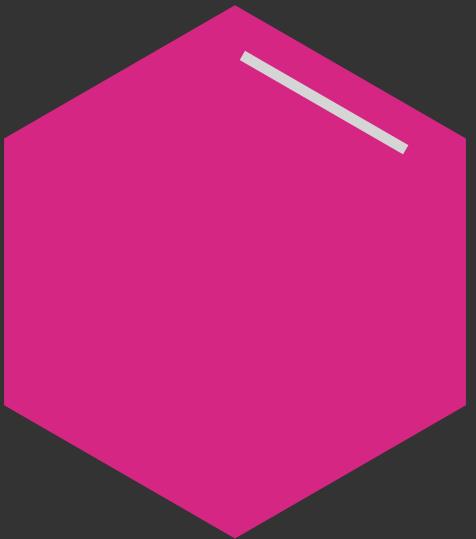


Monorepo



Polyrepo

Service A Service B Service C

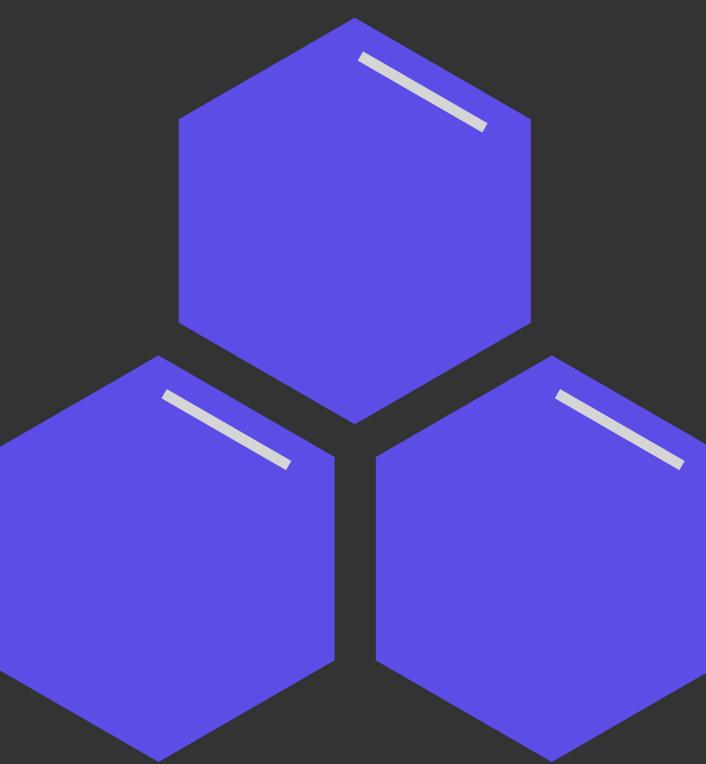


Monorepo

Service A

Service B

Service C



Polyrepo

A screenshot of a Medium post page. The header shows the Medium logo and the word "Software Engineering". A "Get started" button is in the top right. The main title is "Monorepos: Please don't!". Below it is a profile picture of Matt Klein, his name, and the date "Jan 3 · 9 min read". The text discusses monorepos vs polyrepos.

Monorepos: Please don't!

Matt Klein
Jan 3 · 9 min read

Here we are at the beginning of 2019 and I'm engaged in yet another discussion on the merits (or lack thereof) of keeping all of an organization's code in a "[monorepo](#)." For those of you not familiar with this concept, the idea behind a monorepo is to store all code in a single version control system (VCS) repository. The alternative, of course, is to store code split into many different VCS repositories, usually on a service/application/library basis. For the purpose of this post I will call the multiple repository solution a "polyrepo."

A screenshot of a Medium post page. The header shows the Medium logo and the word "Software Engineering". A "Get started" button is in the top right. The main title is "Monorepo: please do!". Below it is a profile picture of Adam Jacob, his name, and the date "Jan 4 · 7 min read". The text argues for choosing a monorepo over a polyrepo.

Monorepo: please do!

Adam Jacob
Jan 4 · 7 min read

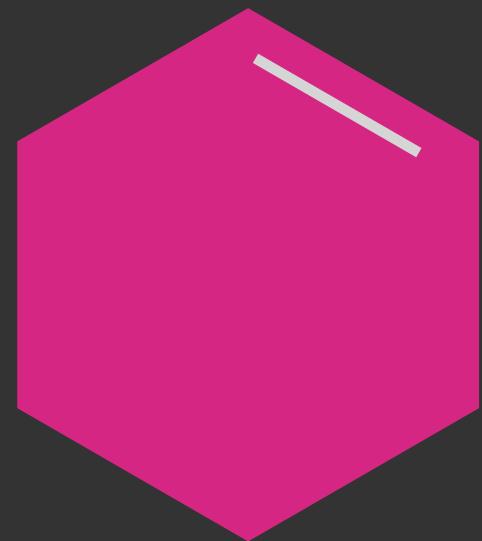
You should choose a monorepo because the default behavior it encourages in your teams is visibility and shared responsibility, especially as teams scale. You will have to invest in tooling no matter what, but it's always better when the default behavior is the behavior you want to see in your teams.

Why are we talking about this?

Matt Klein wrote "[Monorepos: Please don't!](#)"—I like Matt, I think he's very smart, and you should go read his point of view. He originally posted a poll on twitter:

<https://medium.com/@mattklein123/monorepos-please-dont-e9a279be011b>

<https://medium.com/@adamhjk/monorepo-please-do-3657e08a4b70>



Monorepo

● Advantages

- Easy to share YAML code
- Easy to be reviewed by central team
- Easy to be managed by central team
- Easy to set up CI pipeline

● Disadvantages

- Take account into repo scale up
- Take account into delegation of authority
- CI/CD: No independence in each team

● Advantages

- Build CI/CD pipeline by yourself
- No dependency of outside system
- Easy to be scaled up themselves
- Easy to change the pipeline cycle

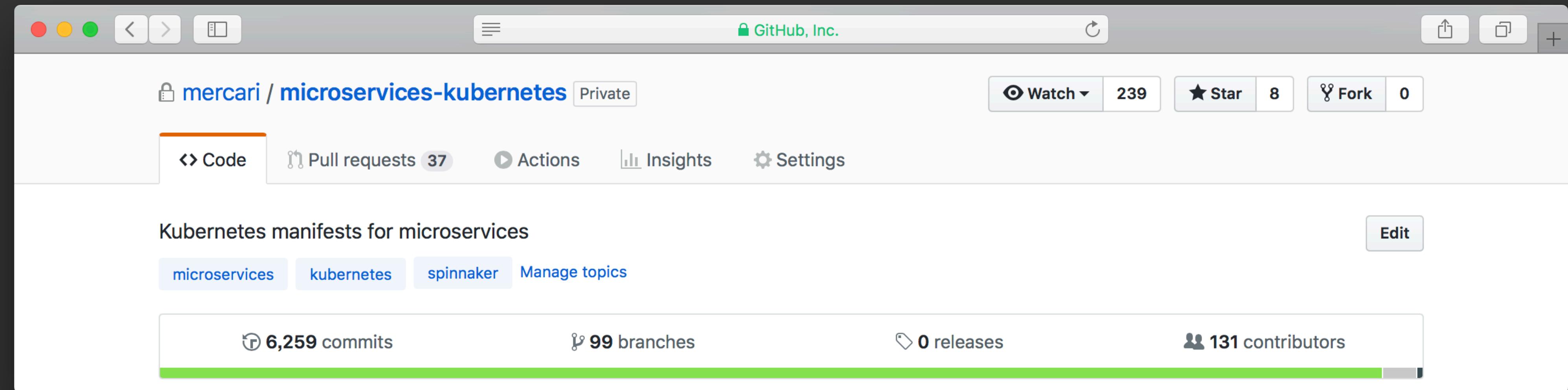


Polyrepo

● Disadvantages

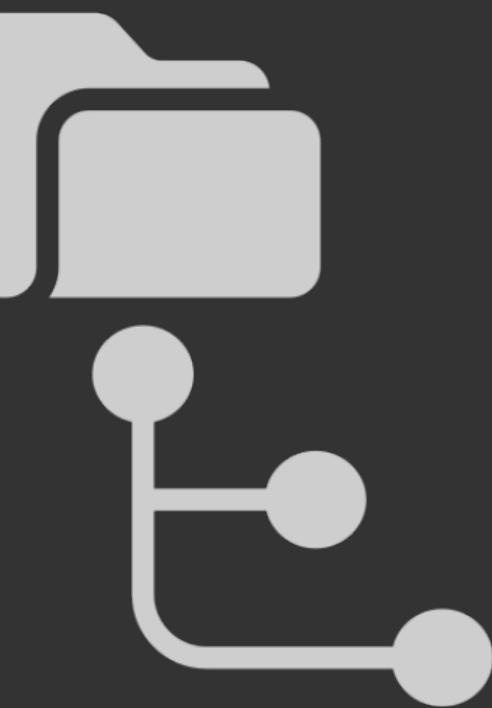
- Not easy to share YAML code
- Difficult to review by central team
- Troublesome to build CI/CD pipeline by yourself





- We chose "Monorepo" style.
- It was a good option to start small.
- The concrete reason will be shown in next section.

4. Directories



A screenshot of a GitHub repository page for "mercari / microservices-kubernetes". The repository is private, has 239 stars, and 0 forks. It contains 6,276 commits, 100 branches, 0 releases, and 131 contributors. The master branch is selected. A recent merge pull request from "mercari-echo-jp-development-branch" is shown at the top. The commit history lists various changes related to Kubernetes manifests and Spinnaker integration.

Kubernetes manifests for microservices

[Edit](#)

microservices kubernetes spinnaker Manage topics

6,276 commits 100 branches 0 releases 131 contributors

Branch: master ▾ New pull request Create new file Upload files Find File Clone or download ▾

b4b4r07 Merge pull request #2348 from mercari-echo-jp-development-branch ... 1 Latest commit 65e20e0 2 days ago

.circleci	Run install-dependencies step in stein step	2 months ago
.github	Add kubernetes manifests for dev	3 days ago
manifests	Add echo jobs	2 days ago
policy	Upgrade stein	3 months ago
script	Ignore authority case	9 days ago
spinnaker	Change env var to restart pods for mercari-echo-jp on prod	2 days ago
.gitignore	Add gitignore	5 months ago
CHANGELOG.md	Update CHANGELOG	6 months ago
Dockerfile	Prepare Docker environment	8 months ago
README.md	Add get-started link	6 months ago
docker-compose.yml	Store kubeconfig at the specified location	8 months ago
entrypoint.sh	Add entrypoint.sh	6 months ago
kind_install_order.txt	Add Kritis's CRD (AttestationAuthority and ImageSecurityPolicy) to su...	2 months ago
kind_uninstall_order.txt	Add Kritis's CRD (AttestationAuthority and ImageSecurityPolicy) to su...	2 months ago

```
microservices-kubernetes
└ manifests/
    └ microservices/
        ┌ mercari-echo-jp/
        │   └ development/
        │       └ production/
        └ mercari-xxx-jp/
            ┌ development/
            │   ┌ Deployment/
            │   └ deployment.yaml
            └ Certificate/
                └ xxx-dev-mercari-com.yaml
            ┌ Ingress/
            └ PodDisruptionBudget/
                └ pdb.yaml
            └ production/
```

```
microservices-kubernetes
└ manifests/
    └ microservices/
        ┌── mercari-echo-jp/
        │   ├── development/
        │   └── production/
        └── mercari-xxx-jp/
            ├── development/
            │   ├── Deployment/
            │   │   └── deployment.yaml
            │   ├── Certificate/
            │   │   └── xxx-dev-mercari-com.yaml
            │   ├── Ingress/
            │   │   └── xxx-dev-mercari-com.yaml
            │   └── PodDisruptionBudget/
            │       └── pdb.yaml
            └── production/
```

microservice

```
microservices-kubernetes
└ manifests/
    └ microservices/
        ┌ mercari-echo-jp/
        ┌└ development/
        ┌└ production/
        └ mercari-xxx-jp/
            └ development/
                ┌ Deployment/
                ┌└ deployment.yaml
                ┌ Certificate/
                ┌└ xxx-dev-mercari-com.yaml
                ┌ Ingress/
                ┌└ xxx-dev-mercari-com.yaml
                └ PodDisruptionBudget/
                    └ pdb.yaml
            └ production/
```

microservice
environment

```
microservices-kubernetes
└ manifests/
    └ microservices/
        ┌ mercari-echo-jp/
        ┌└ development/
        ┌└ production/
        └ mercari-xxx-jp/
            └ development/
                ┌ Deployment/
                ┌└ deployment.yaml
                ┌ Certificate/
                ┌└ xxx-dev-mercari-com.yaml
                ┌ Ingress/
                ┌└ xxx-dev-mercari-com.yaml
                ┌ PodDisruptionBudget/
                ┌└ pdb.yaml
                └ production/
```

microservice
environment
kind

```
microservices-kubernetes
└ manifests/
    └ microservices/
        ┌ mercari-echo-jp/
        ┌└ development/
        ┌└ production/
        └ mercari-xxx-jp/
            └ development/
                ┌ Deployment/
                ┌└ deployment.yaml
                └ Certificate/
                    └ xxx-dev-mercari-com.yaml
                ┌ Ingress/
                ┌└ xxx-dev-mercari-com.yaml
                └ PodDisruptionBudget/
                    └ pdb.yaml
                └ production/
```

microservice
environment
kind
resource

Spinnaker case

```
microservices-kubernetes
└── manifests/
    └── spinnaker/
        └── microservices/
            ├── mercari-xxx-jp/
            │   ├── development/
            │   └── production/
            └── mercari-echo-jp/
                ├── development/
                │   ├── deploy-to-dev-v2.yaml
                │   └── deploy-cronjobs-dev-v2.yaml
                └── production/
                    └── deploy-to-prod-v2.yaml
```

```
microservices-kubernetes
└── manifests/
    └── spinnaker/
        └── microservices/
            ├── mercari-xxx-jp/
            │   ├── development/
            │   └── production/
            └── mercari-echo-jp/
                ├── development/
                │   ├── deploy-to-dev-v2.yaml
                │   └── deploy-cronjobs-dev-v2.yaml
                └── production/
                    └── deploy-to-prod-v2.yaml
```

microservice

```
microservices-kubernetes
└── manifests/
└── spinnaker/
    └── microservices/
        ├── mercari-xxx-jp/
        │   └── development/
        │       └── production/
        └── mercari-echo-jp/
            ├── development/
            │   └── deploy-to-dev-v2.yaml
            └── production/
                └── deploy-to-prod-v2.yaml
```

microservice
environment

```
microservices-kubernetes
└── manifests/
└── spinnaker/
    └── microservices/
        ├── mercari-xxx-jp/
        │   ├── development/
        │   └── production/ environment
        └── mercari-echo-jp/
            ├── development/ pipeline
            │   ├── deploy-to-dev-v2.yaml yaml
            │   └── deploy-cronjobs-dev-v2.yaml
            └── production/
                └── deploy-to-prod-v2.yaml
```


The screenshot shows the Spinnaker web interface for managing pipelines. The top navigation bar includes 'SPINNAKER', 'Search', 'Projects', 'Applications', and a user profile icon. The main header displays the project name 'mercari-echo-jp' and tabs for 'PIPELINES', 'INFRASTRUCTURE', 'TASKS', and 'CONFIG'. The 'PIPELINES' tab is selected.

The left sidebar contains sections for 'CONCURRENT EXECUTIONS', 'EXPECTED ARTIFACTS (2)', 'AUTOMATED TRIGGERS (1)', 'PARAMETERS', 'NOTIFICATIONS', and 'DESCRIPTION'.

The 'EXPECTED ARTIFACTS' section is currently active, titled 'Expected Artifacts'. It instructs the user to declare artifacts expected during execution. A 'Match against' dropdown is set to 'GitHub', with a note: 'A file stored in git, hosted by GitHub.' A trash icon is available to remove this entry.

Below this, under 'File path', the value 'spinnaker/microservices/mercari-echo-jp/development/deploy-to-dev-v2.' is listed. Under 'If missing', the 'Use Default Artifact' checkbox is checked. A 'Default artifact' section follows, with a 'GitHub' dropdown and a note: 'A file stored in git, hosted by GitHub.'

At the bottom of the configuration area, there are 'Revert' and 'Save Changes' buttons. The 'Save Changes' button is highlighted with a pink border.

Pipeline

The screenshot shows the Spinnaker interface for the **mercari-echo-jp** application. The top navigation bar includes **SPINNAKER**, **Search**, **Projects**, **Applications** (selected), **b4b4r07@mercari.com**, **Search**, and **Help**. The main header has tabs for **PIPLINES**, **INFRASTRUCTURE**, **TASKS**, and **CONFIG**.

The left sidebar contains sections for **SEARCH**, **PIPELINES** (with a checkbox for "deploy to dev v2" and a "Reorder Pipelines" button), and **STATUS**.

The central area displays the **GKE-CLUSTER-DEV-ASIA-NORTHEAST1-B-V2** pipeline. It shows a single execution entry:

- DOCKER REGISTRY**: `mercari-echo-jp/mercari-echo-jp` (4 days ago)
- Status**: SUCCEEDED
- Duration**: 00:31
- Details**: `spinnaker/mercari-echo-jp/dev...` Version master, `gcr.io/mercari-echo-jp/mercari-...` Version master-20190417081926

Buttons for **Configure** and **Start Manual Execution** are also present.

Pipeline

How do we apply the manifest changes?

Just run "kubectl"



<https://groups.google.com/forum/#!msg/kubernetes-sig-cli/M6t40JP6n0g/U6Snz-bsFQAJ>

kubectl

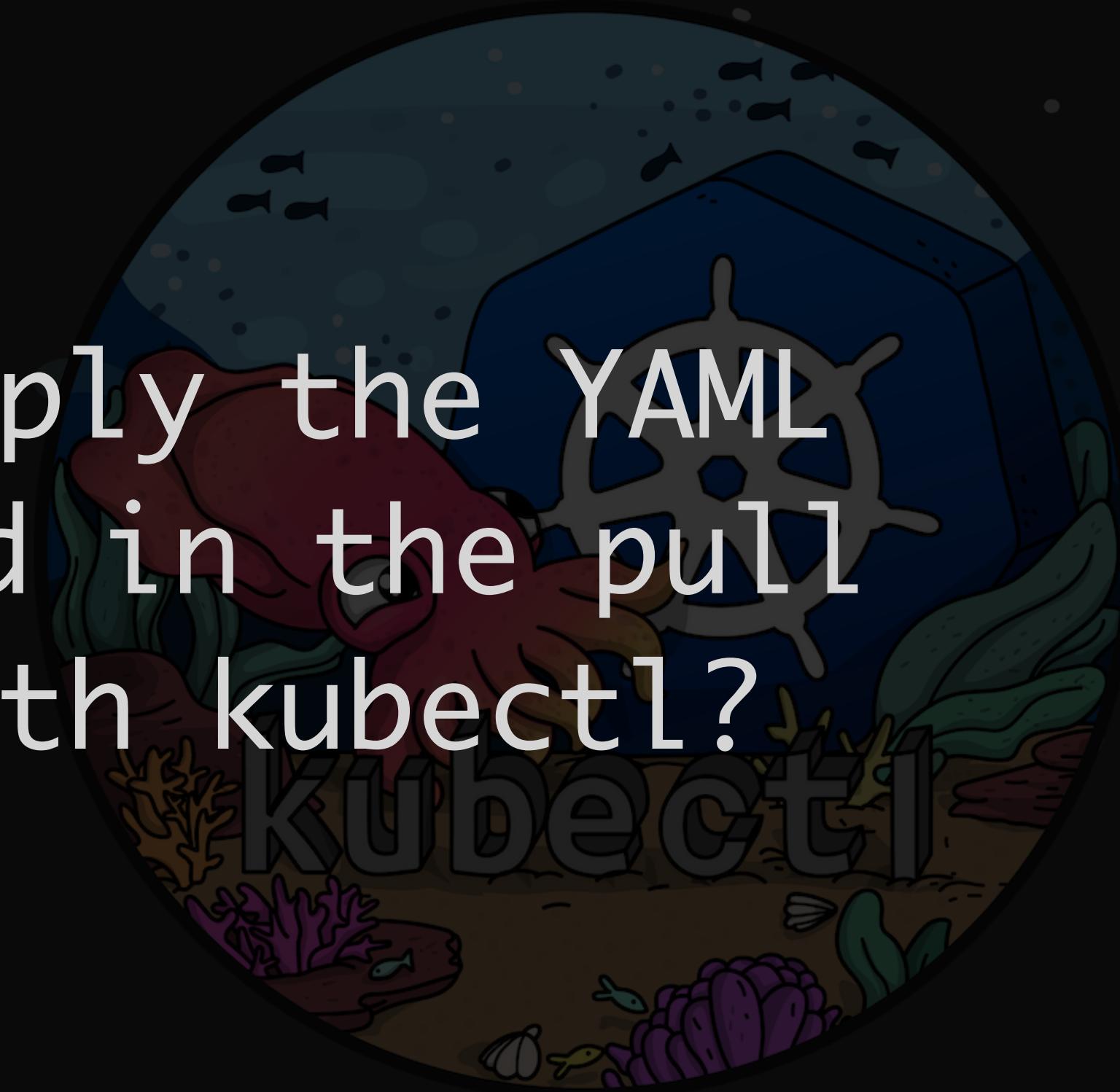
- Microservices developers not only develop but also operate the service by themselves
- So they are familiar with kubectl basically
- It means less learning cost than introducing other tools



kubectl

- Microservices developers not only develop but also operate the service by themselves
- So they are familiar with kubectl basically
- It means less learning cost than introducing other tools

How do we apply the YAML files changed in the pull requests with kubectl?

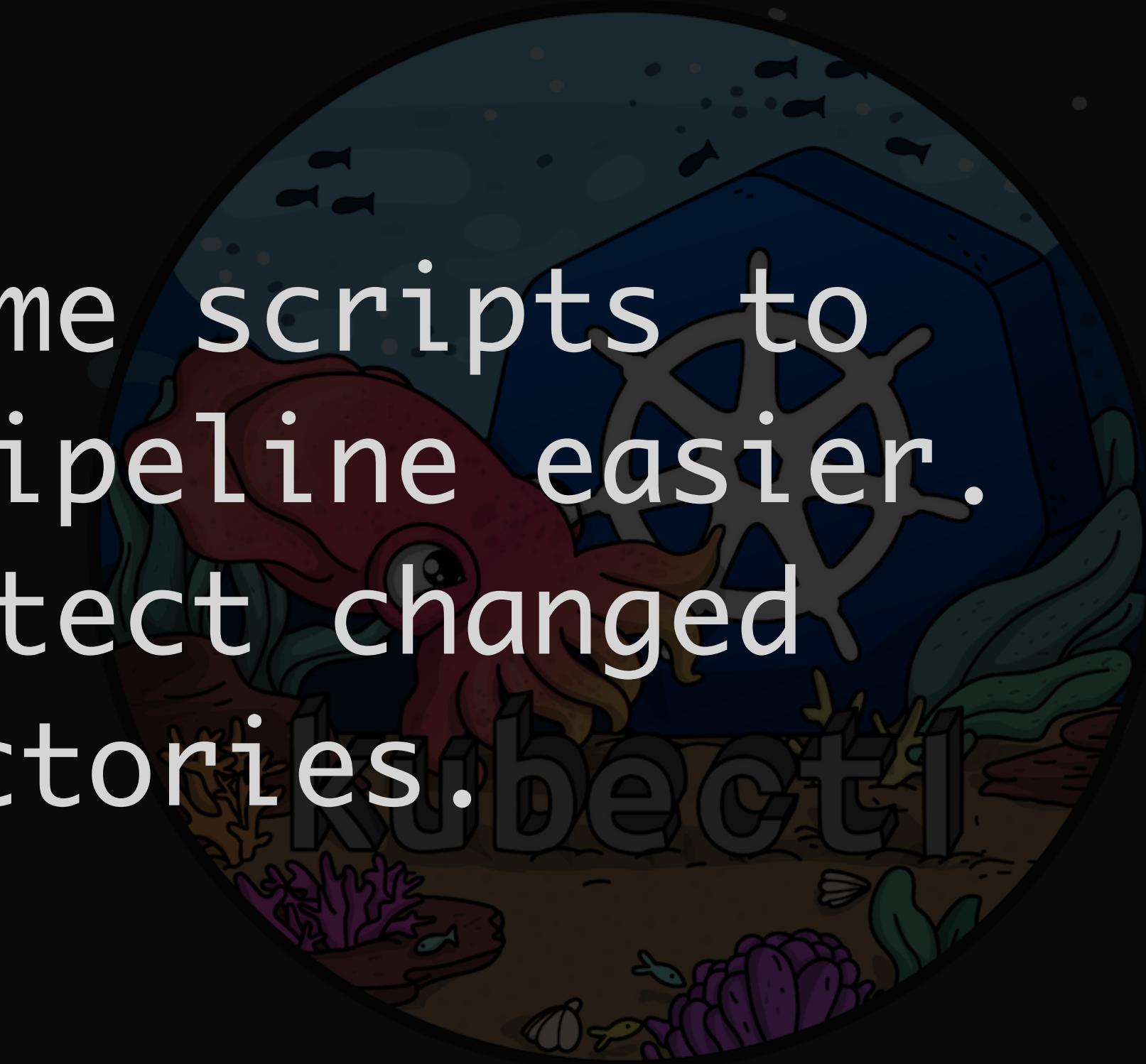


kubectl

- Microservices developers not only develop but also operate the service by themselves
- So they are familiar with kubectl basically
- It means less learning cost than introducing other tools



We have some scripts to make apply pipeline easier.
It can detect changed directories.



Let's say we'd add new manifest

```
microservices-kubernetes
└ manifests/
    └ microservices/
        └ mercari-echo-jp/
            ├── development/
            │   ├── Deployment/
            │   │   └── deployment.yaml
            │   ├── Certificate/
            │   │   └── xxx-dev-mercari-com.yaml
            │   ├── Ingress/
            │   │   └── xxx-dev-mercari-com.yaml
            │   └── PodDisruptionBudget/
            │       └── pdb.yaml
            └── production/
```

```
microservices-kubernetes
└ manifests/
    └ microservices/
        └ mercari-echo-jp/
            ├ development/
            │   └ Deployment/
            │       └ deployment.yaml
            └ Certificate/
                └ xxx-dev-mercari-com.yaml
            └ Ingress/
                └ xxx-dev-mercari-com.yaml
            └ PodDisruptionBudget/
                └ pdb.yaml
                    └ pdb-server.yaml added
            └ production/
```

```
microservices-kubernetes
└── manifests/
    └── microservices/
        └── mercari-echo-jp/
            ├── development/
            │   └── Deployment/
            │       └── deployment.yaml
            ├── Certificate/
            │   └── xxx-dev-mercari-com.yaml
            ├── Ingress/
            │   └── xxx-dev-mercari-com.yaml
            └── PodDisruptionBudget/
                └── pdb.yaml
                    └── pdb-server.yaml added
            └── production/
```

Helper bash scripts detect changed
directory from the PR

manifests/microservices/mercari-echo-jp/development/PodDisruptionBudget

```
changed_files() {
    declare basedir="${1}"
    declare current_branch="$(git rev-parse --abbrev-ref @)"
    if [[ ${current_branch} == "master" ]]; then
        # (apply)
        # In the master branch, when listing files edited
        # you need to compare with previous merge commit
        git diff --name-only "HEAD^" "HEAD" "${basedir}"
    else
        # (plan)
        # In the topic branch, when listing files edited in the branch,
        # you need to compare with the commit at the time
        # the branch was created
        # https://git-scm.com/docs/git-merge-base
        git diff --name-only $(git merge-base origin/HEAD HEAD) "${basedir}"
    fi
}
```

```
changed_dirs() {
    # Note:
    #   If these files are changed
    #     - manifests/microservices/x/development/Ingress
    #     - manifests/microservices/x/development/PersistentVolumeClaim
    #     - manifests/microservices/y/production/PersistentVolumeClaim
    #     - manifests/microservices/y/production/Pod
    #     - manifests/microservices/y/production/PodDisruptionBudget
    #   The files we have to pass to script/apply are only two dirs
    #     - manifests/microservices/x/development
    #     - manifests/microservices/y/production
    declare basedir="${1}"
    for file in $(changed_files "${basedir}")
    do
        get_target_dir "${file}"
    done | sort | uniq
}
```

```
changed_dirs() {  
    # Note:  
    # If those files are changed  
    # - manifests/microservices/x/development/Ingress  
    # - manifests/microservices/x/development/PersistentVolumeClaim  
    # - manifests/microservices/y/production/PersistentVolumeClaim  
    # - manifests/microservices/y/production/Pod  
    # - manifests/microservices/y/production/PodDisruptionBudget  
    # The files we have to pass to script/apply are only two dirs  
    # - manifests/microservices/x/development  
    # - manifests/microservices/y/production  
    declare basedir="${1}"  
    for file in $(changed_files "${basedir}")  
    do  
        get_target_dir "${file}"  
        done | sort | uniq  
    }  
  
    ● Namespace must be created before all other resources  
    ● ConfigMap must be created before Deployment
```



```
// InstallOrder is the order in which manifests should be installed (by Kind).
// Those occurring earlier in the list get installed before those occurring later in the list.
var InstallOrder SortOrder = []string{
    "Namespace",
    "ResourceQuota",
    "PodSecurityPolicy",
    "Secret",
    "ConfigMap",
    "PersistentVolume",
    "PersistentVolumeClaim",
    "CustomResourceDefinition",
    "Role",
    "RoleBinding",
    "Service",
    "DaemonSet",
    "Pod",
    "ReplicaSet",
    "Deployment",
    "StatefulSet",
    "Job",
    "CronJob",
    "Ingress",
}
```

```
// InstallOrder is the order in which manifests should be installed (by Kind).  
// Those occurring earlier in the list get installed before those occurring later in the list.  
var InstallOrder SortOrder = []string{  
    "Namespace",  
    "ResourceQuota",  
    "PodSecurityPolicy",  
    "Secret",  
    "ConfigMap",  
    "PersistentVolume",  
    "PersistentVolumeClaim",  
    "CustomResourceDefinition",  
    "Role",  
    "RoleBinding",  
    "Service",  
    "DaemonSet",  
    "Pod",  
    "ReplicaSet",  
    "Deployment",  
    "StatefulSet",  
    "Job",  
    "CronJob",  
    "Ingress",  
}  
  
→ kind_install_order.txt
```



ResourceQuota
Secret
ConfigMap
PersistentVolume
PersistentVolumeClaim
ServiceAccount
Role
RoleBinding
Service
DaemonSet
Pod
ReplicaSet
Deployment
StatefulSet
Job
CronJob
Ingress
HorizontalPodAutoscaler
NetworkPolicy
PodDisruptionBudget

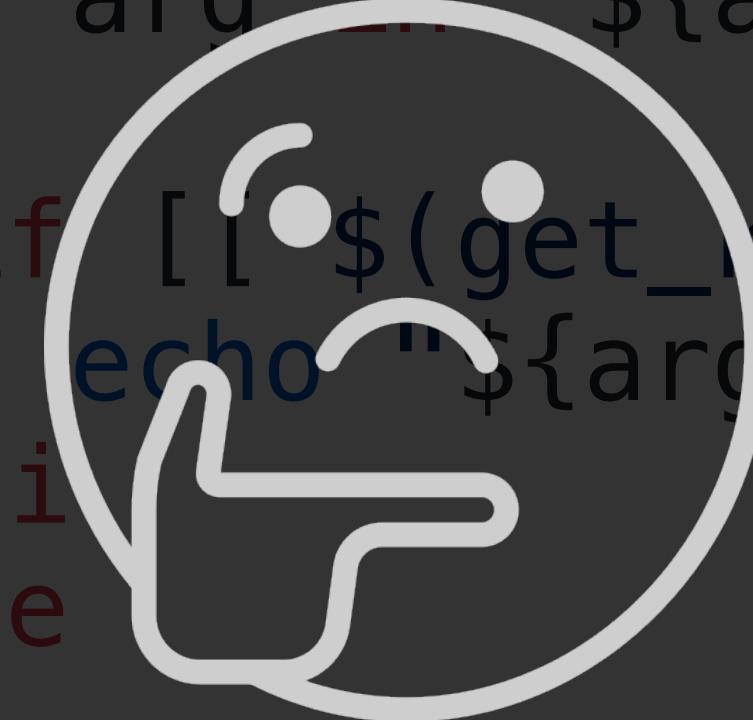
```
sort_kinds_by_install_order() {
    kinds=( $(cat "kind_install_order.txt") )
    args=( "${@}" )
    for kind in "${kinds[@]}"
    do
        for arg in "${args[@]}"
        do
            if [[ $(get_kind "${arg}") == ${kind} ]]; then
                echo "${arg}"
            fi
        done
    done
}
```

..../development/Deployment
..../development/ConfigMap
..../development/PodDisruptionBudget

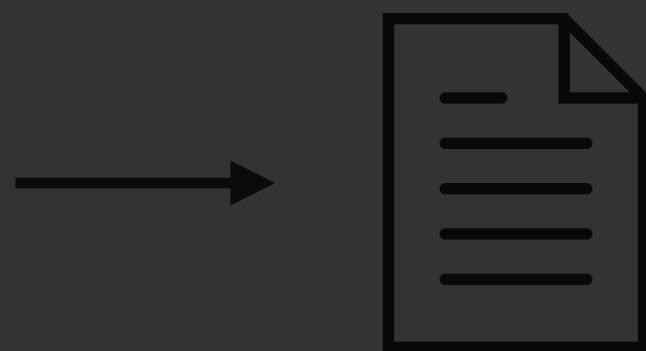


ConfigMap
Deployment
PodDisruptionBudget

```
sort_kinds_by_install_order() {  
    kinds=( $(cat "kind_install_order.txt") )  
    args=( "${@}" )  
    for kind in "${kinds[@]}"  
    do  
        for arg in "${args[@]}"  
        do  
            if [[ ${get_kind "${arg}"} == ${kind} ]]; then  
                echo "${arg}"  
            fi  
        done  
    done  
}
```



.../development/Deployment
.../development/ConfigMap
.../development/PodDisruptionBudget



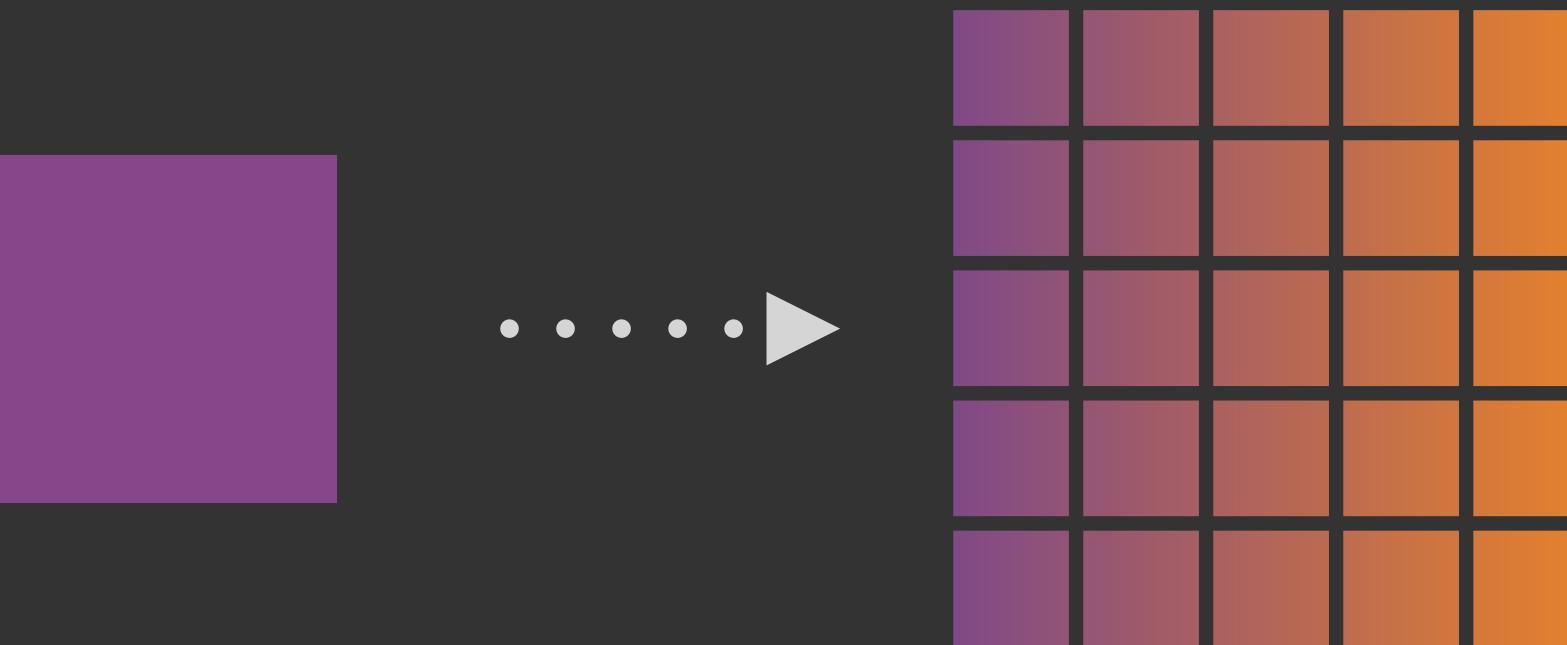
kind_install_order.txt

→ ConfigMap
→ Deployment
→ PodDisruptionBudget

How about using kustomize?

```
microservices-kubernetes
└ manifests/
    └ microservices/
        └ mercari-echo-jp/
            ├── base/
            │   ├── kustomization.yaml
            │   └── Deployment/
            │       └── deployment.yaml
            ├── development/
            │   ├── kustomization.yaml
            │   ├── Deployment/
            │   │   └── deployment.yaml
            │   ├── Certificate/
            │   │   └── xxx-dev-mercari-com.yaml
            │   ├── Ingress/
            │   │   └── xxx-dev-mercari-com.yaml
            │   └── PodDisruptionBudget/
            │       └── pdb.yaml
            └── production/
```

It's easy to introduce our design but it has new concepts of "overlay" etc.



- Our microservices is on the way
- So developers are in the middle of being microservices developers
- They have to learn a lot of things:
 - Kubernetes, Kubernetes YAML itself, Spinnaker, etc...
 - So introducing kustomize feature to our pipeline was not now.
In the future.

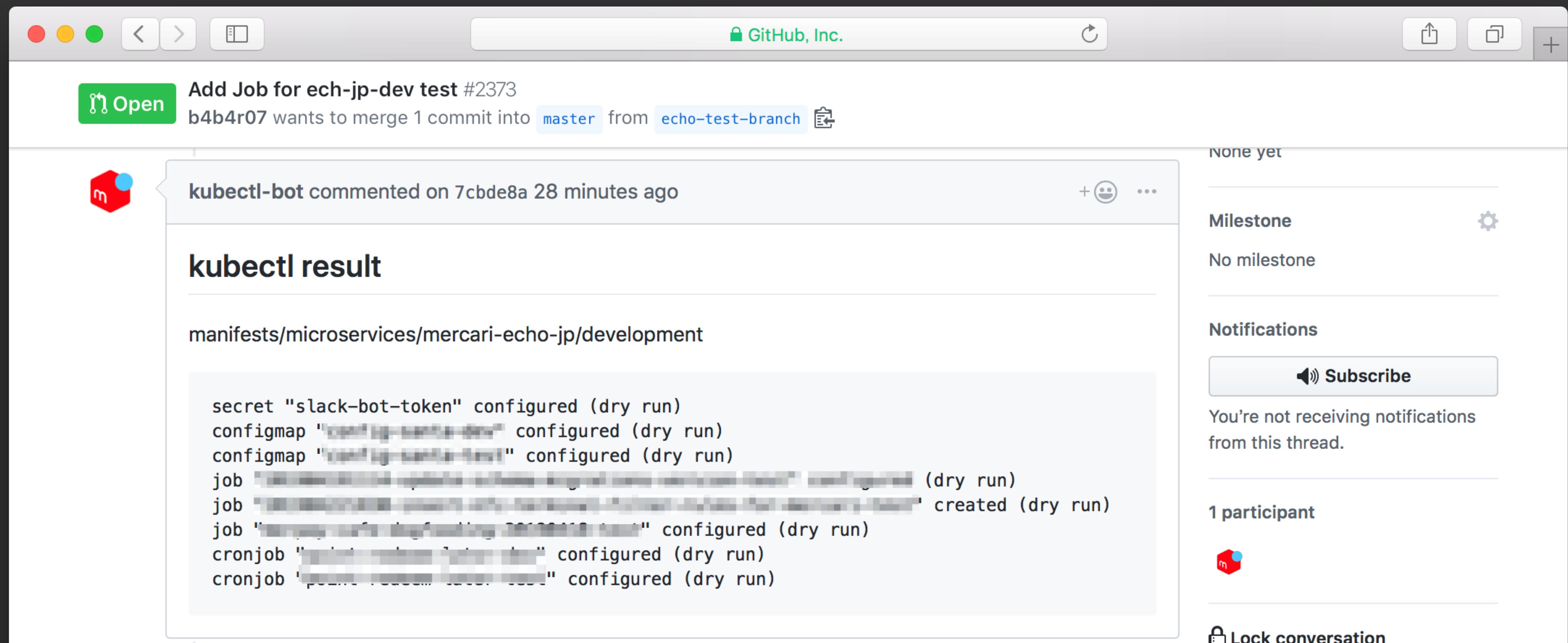


How do we apply PodDisruptionBudget?

- Some resource kinds (e.g., PodDisruptionBudget) cannot update in-place
- It means we cannot update existing PodDisruptionBudget or StatefulSet by kubectl apply.

```
kubectl_apply() {
# ...
for manifest in $(sort_kinds_by_install_order "${manifests[@]}"); do
  case ${kind} in
    PodDisruptionBudget) # Need to be recreated if it already exists
      if kubectl get -n "${namespace}" "${kind}" "${resource}"; then
        kubectl delete -n "${namespace}" "${kind}" "${resource}"
      fi
      kubectl apply -n "${namespace}" -f "${manifes}" ;;
    Secret)
      ansible-vault view "${manifest}" \
        | kubectl apply -n "${namespace}" -f - ;;
    *)
      kubectl apply -n "${namespace}" -f "${manifest}" ;;
  esac
done}
```

In order to deal with those special kinds, we prepare for easy wrapper script for kubectl.



The delegation of directory authority

```
microservices-kubernetes
└── manifests/
    └── microservices/
        ├── mercari-echo-jp/
        │   ├── development/
        │   └── production/
        └── mercari-xxx-jp/
            ├── development/
            │   ├── Deployment/
            │   │   └── deployment.yaml
            │   ├── Certificate/
            │   │   └── xxx-dev-mercari-com.yaml
            │   ├── Ingress/
            │   │   └── xxx-dev-mercari-com.yaml
            │   └── PodDisruptionBudget/
            │       └── pdb.yaml
            └── production/
```

GitHub CODEOWNERS feature

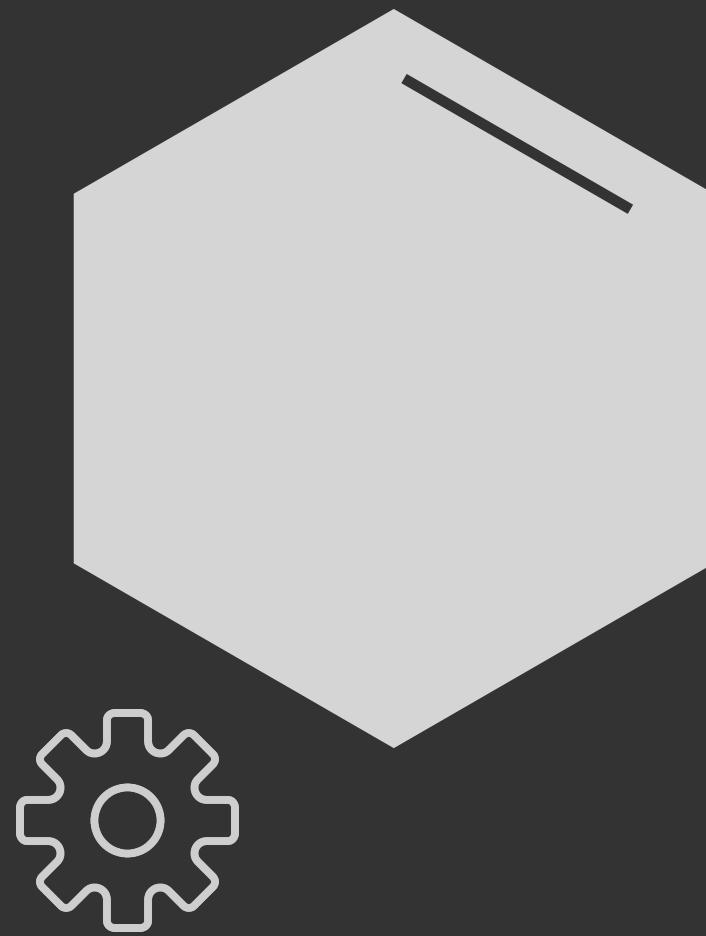
The screenshot shows a GitHub pull request interface with the following details:

- Code owner review required**: Waiting on code owner review from octocat and/or github/js. [Learn more.](#)
- octocat was requested for review as a code owner
- github/js was requested for review as a code owner
- Merging is blocked**: Merging can be performed automatically with one approved review.
- As an administrator, you may still merge this pull request.
- Merge pull request** button
- You can also [open this in GitHub Desktop](#) or view [commands](#)

GitHub CODEOWNERS feature

```
# This is a CODEOWNERS file to define individuals or teams that are responsible
# for code in a repository.
#
# For more detail about CODEOWNERS, refer to following articles:
#   - https://help.github.com/articles/about-codeowners/
#   - https://blog.github.com/2017-07-06-introducing-code-owners/
#
# These owners will be the default owners for everything in the repo.
★ @mercari/microservices-platform
#
# Only platform team can approve CODEOWNERS file's change
/.github/CODEOWNERS @mercari/microservices-platform
#
# mercari-echo-jp
/manifests/microservices/mercari-echo-jp/development/ @mercari/mercari-echo-jp-dev
/manifests/microservices/mercari-echo-jp/production/ @mercari/mercari-echo-jp-prod
/spinnaker/microservices/mercari-echo-jp/development/ @mercari/mercari-echo-jp-dev
/spinnaker/microservices/mercari-echo-jp/production/ @mercari/mercari-echo-jp-prod
```

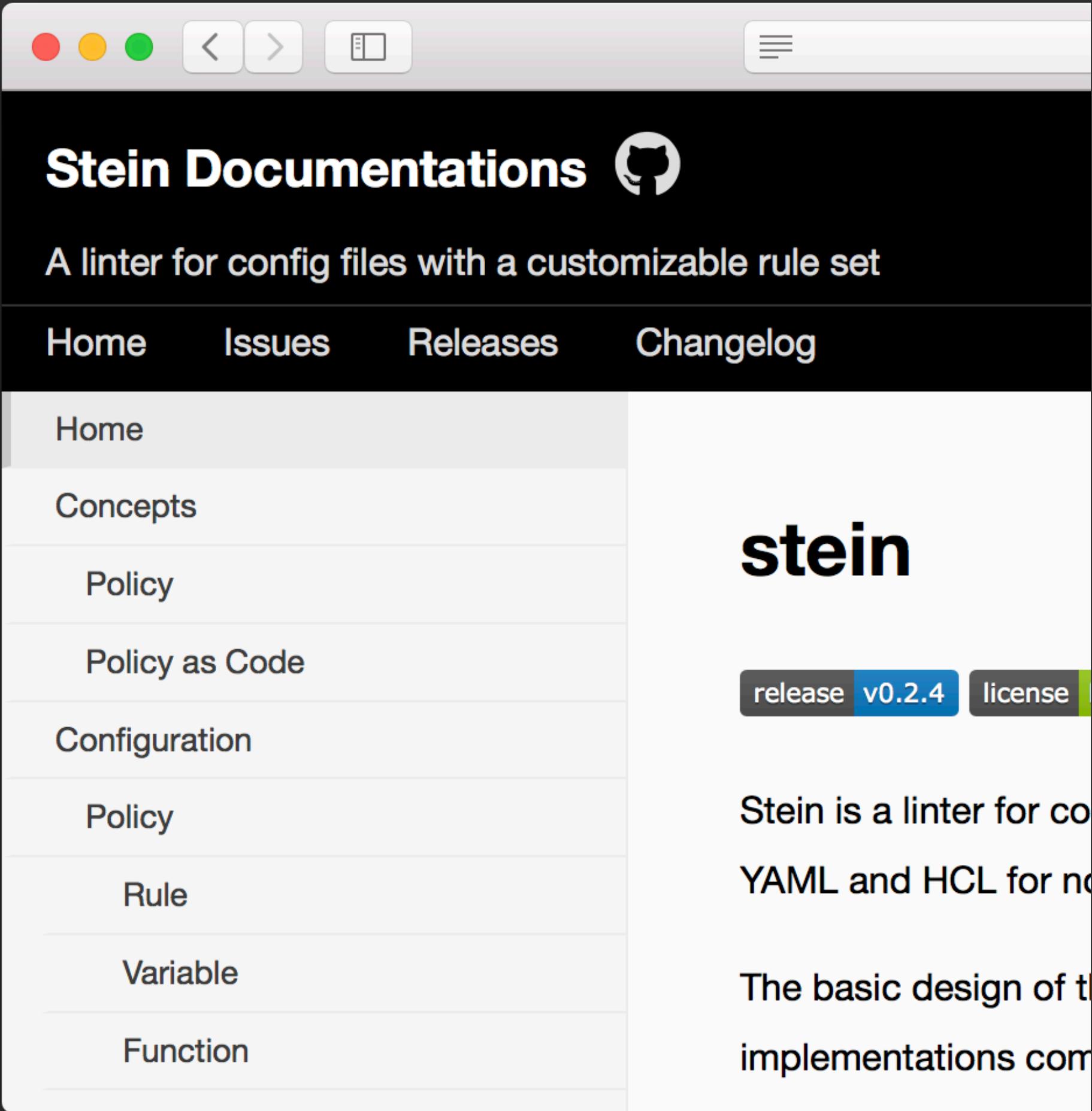
Repository Ecosystem



Repository Ecosystem

- Our microservices-kubernetes repository has some awesome tools to make it maintain easier and more handy like the ecosystem

- One of those tools is a linter for Kubernetes YAML: Stein



Repository Ecosystem

- For example, let's say you don't make the developers omit `metadata.namespace` field in their YAMLs to prevent from unexpected apply
- However, do you have a way to do it in existing tools...?

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  namespace: x-echo-jp-dev
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
```

A screenshot of a GitHub repository page for 'b4b4r07/stein'. The page shows the repository's details, including its name, description, topics, and statistics. The 'Code' tab is selected.

Repository Details:

- Name:** b4b4r07 / stein
- Description:** A linter for config files with a customizable rule set <https://b4b4r07.github.io/stein/>
- Topics:** json, yaml, hcl, terraform, kubernetes, linter, infrastructure, go, hashicorp, sentinel, infrastructure-as-code
- Contributors:** 1 contributor
- Licensing:** MIT

Statistics:

- 49 commits
- 2 branches
- 6 releases
- 1 environment

Stein can do that.

Stein configuration

```
rule "namespace_specification" {
    description = "Check namespace name is not empty"

    conditions = [
        "${jsonpath(\"metadata.namespace\") != \"\"}",
    ]

    report {
        level   = "ERROR"
        message = "Namespace is not specified"
    }
}
```

Stein allows you to enforce the rule defined by you based on your policy upon your YAML.

Stein configuration

```
rule "namespace_specification" {  
    description = "Check namespace name is not empty"
```

rule definition

```
    conditions = [  
        "${jsonpath(\"metadata.namespace\") != \"\"}",  
    ]
```

the condition
this rule
fails or not

```
    report {  
        level = "ERROR"  
        message = "Namespace is not specified"  
    }  
}
```

if it fails it
returns 1 with
message according to
this block

Stein allows you to enforce the rule defined by you based on your policy upon your YAML.

Stein configuration

```
rule "namespace_specification" {
  description = "Check namespace name is not empty"

  conditions = [
    "${jsonpath(\"metadata.namespace\") != \"\"}",
  ]
  Stein supports many built-in functions like Terraform

  report {
    level    = "ERROR"
    message = "Namespace is not specified"
  }
}
```

stein can interpret HCL like Terraform

```
$ stein apply  
x-echo-jp/development/Pod/test.yaml  
[ERROR] rule.namespace_specification Namespace is not specified  
=====  
7 error(s), 2 warn(s)
```

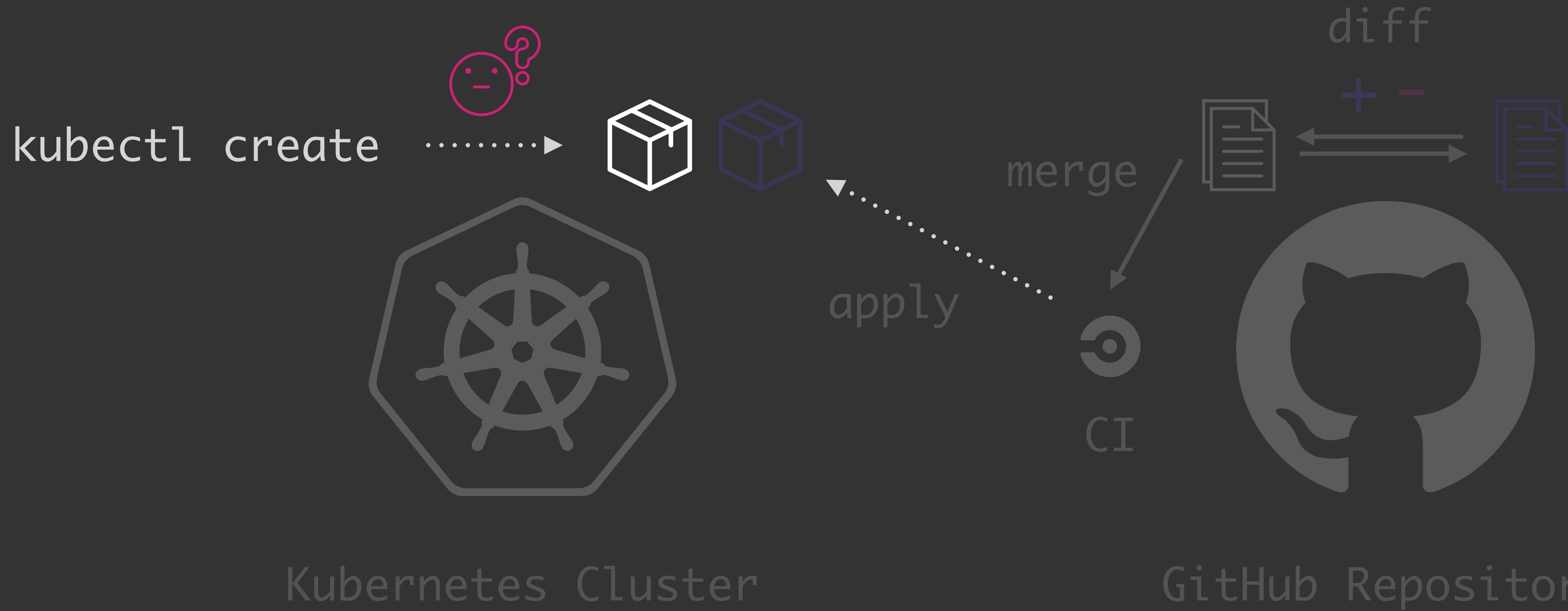
- Stein checks the policy files and applies them to your config files. If there are violation rules, Stein returns exit code 1.
- Stein can work as a linter in CI step etc well.

Policy as Code

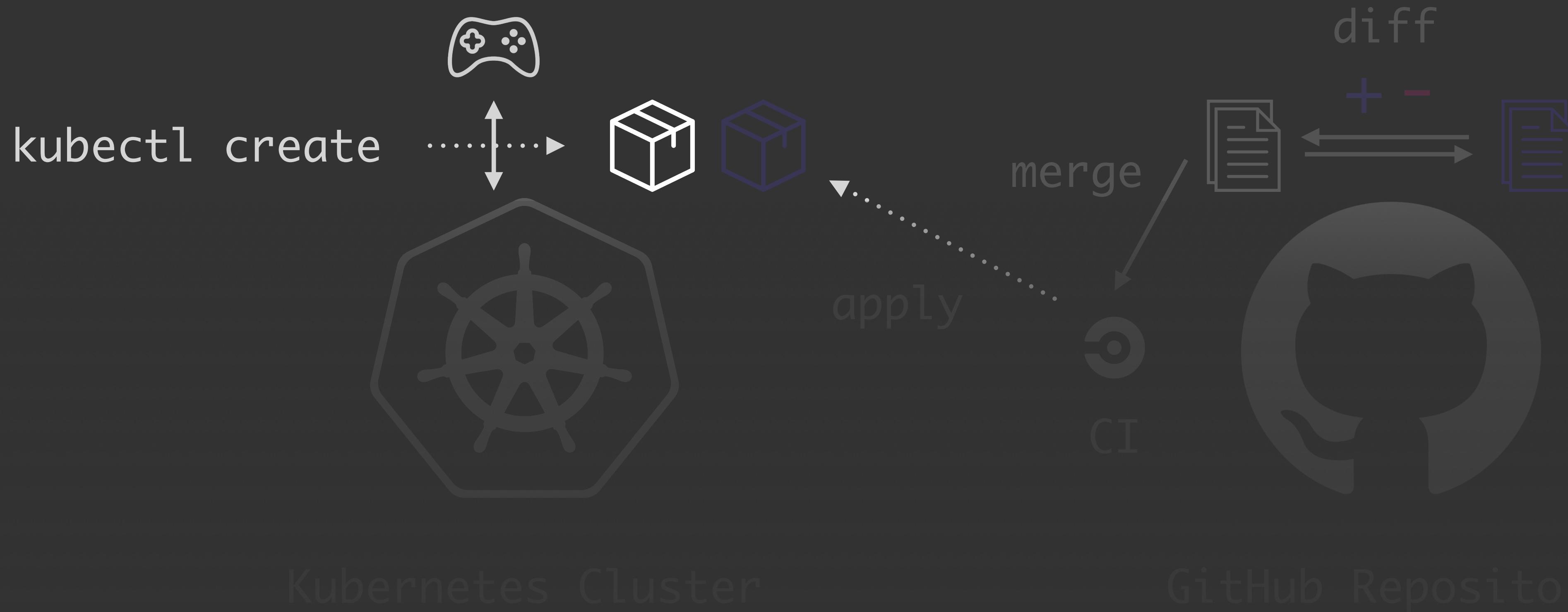


- Stein concepts and design comes from HashiCorp Sentinel one.
- "Policy as Code" (PaC) is provided by HashiCorp and Sentinel .
- PaC means the way to describe "ideal configuration files" and force it upon real configuration files.

not implemented yet
(described later)



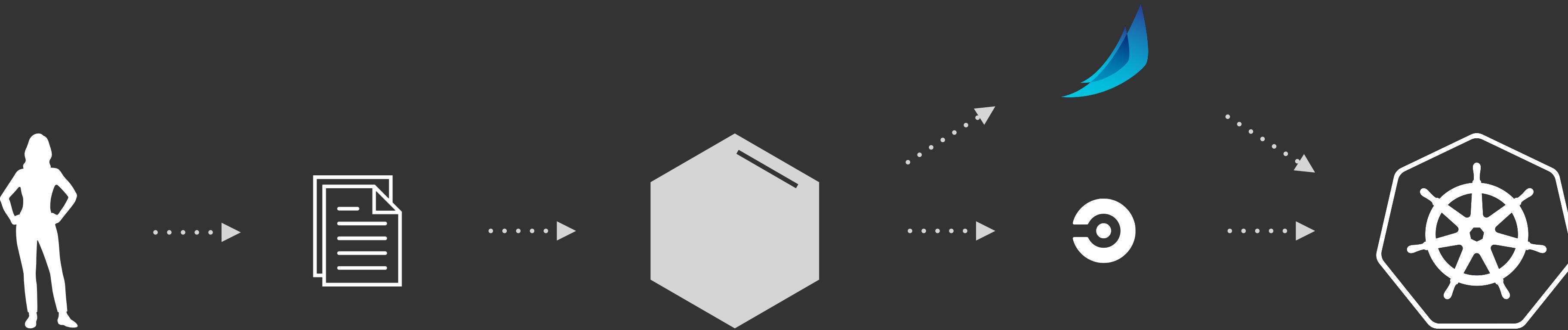
Stein: Adminmission Controller



- TODO: Stein can work the admission controller also.
 - By doing so, it is possible to check whether YAML having violated rules is going to be applied.
 - It can be compatible with "Push" strategy.

Recap

3. Run apply (dry-run)



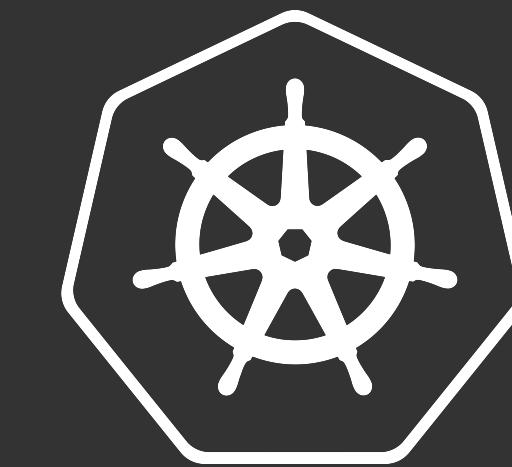
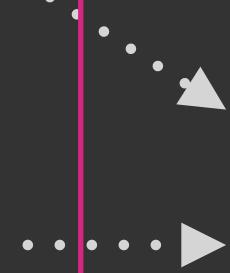
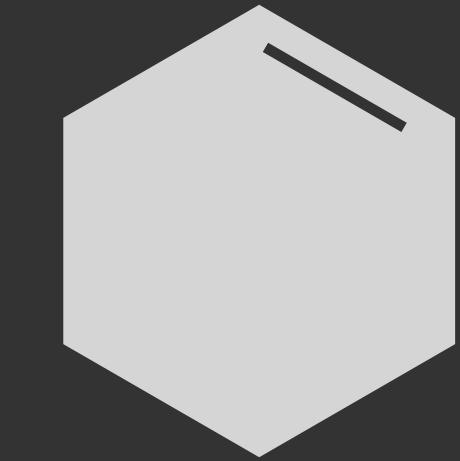
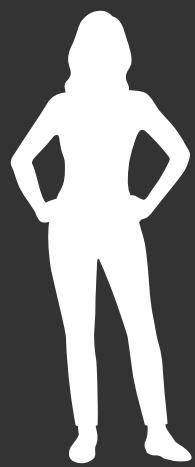
1. Write manifests

2. Send Pull Request

- kubectl pipeline
- stein lint step
- Dir base delegation
- apply when merged

3. Run apply (dry-run)

1. Write manifests
2. Send Pull Request



- kubectl pipeline
- stein lint step
- Dir base delegation
- apply when merged

We can provide the common resources
to all microservices

- By using Monorepo style,
 - we can provide the common guard rail to start to develop & operate their own microservices
 - apply pipeline
 - review by central team
 - common lint step
- Of course, it has also disadvantages
- It's trade-offs for scaling up

●Thank you