Kubernetes manifests management & operation in Mercari

@b4b4r07 (Apr 22, 2019) / Kubernete 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.

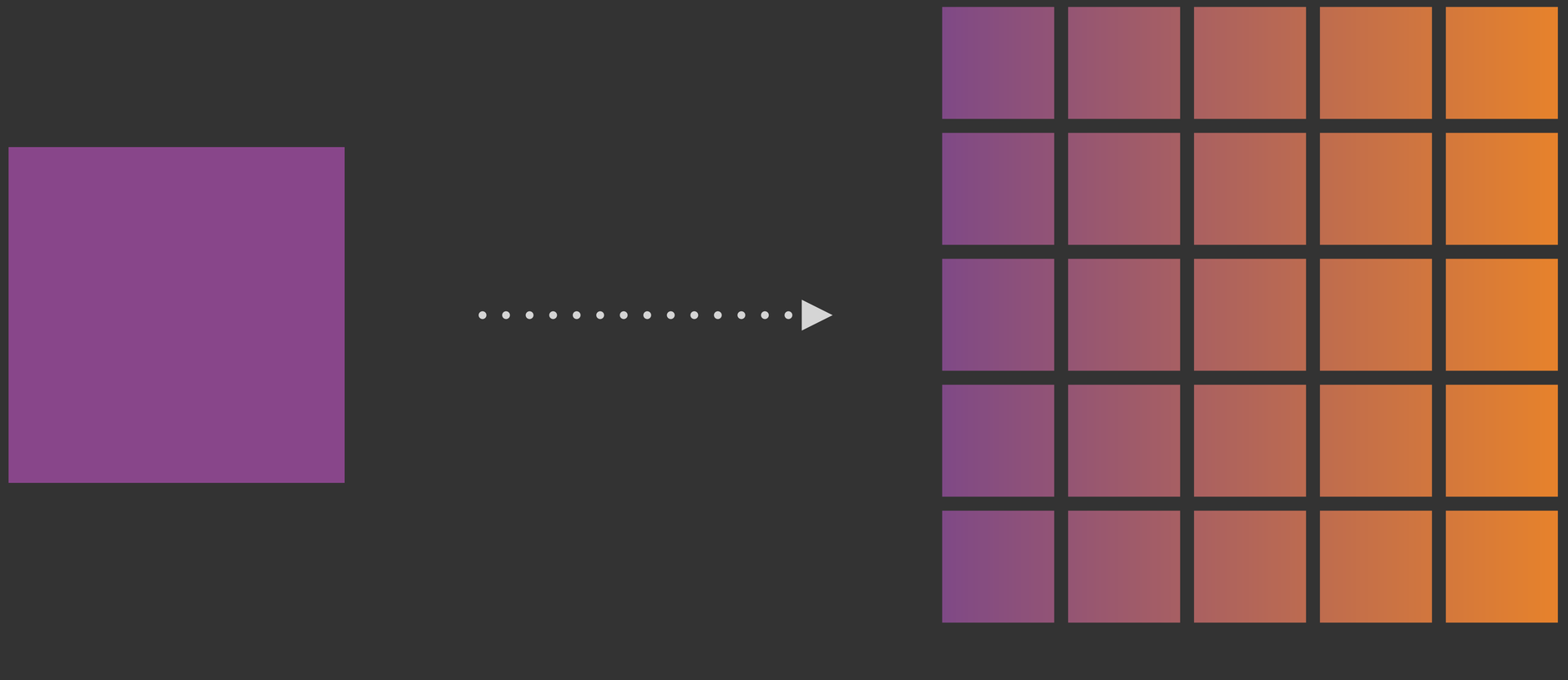
Ship it.

No meetups. Printable shipping label emailed to seller.

Get paid.

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

Our current status

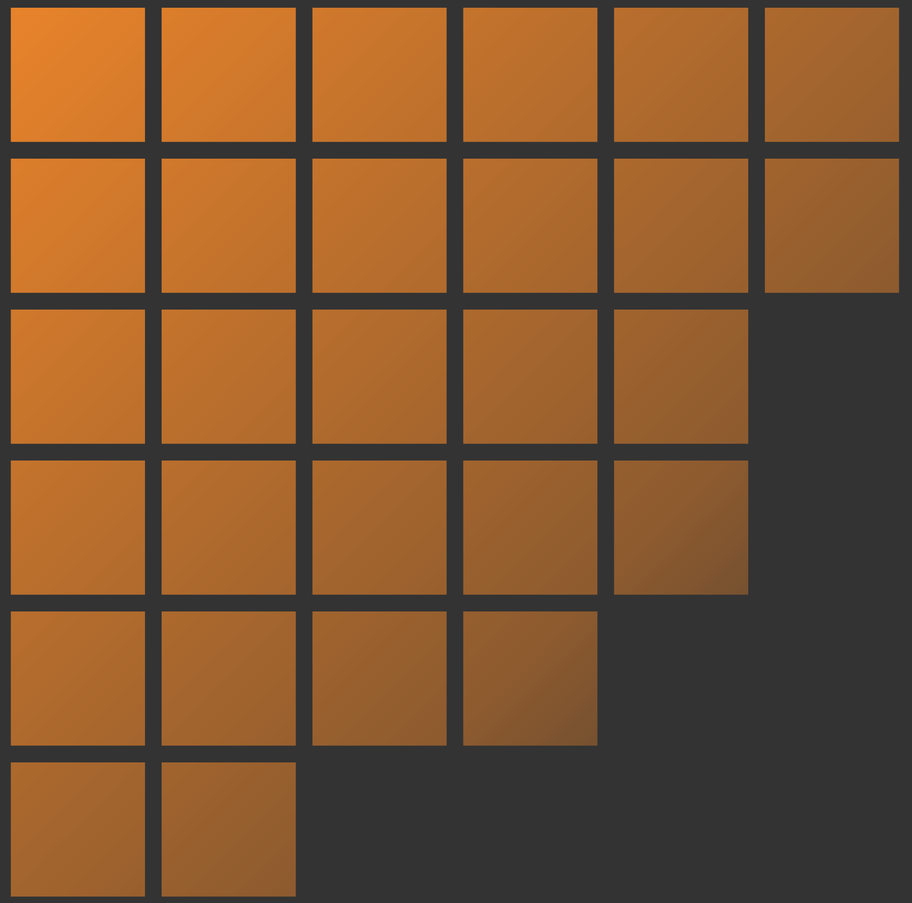


Monolith

Microservices

Our current status

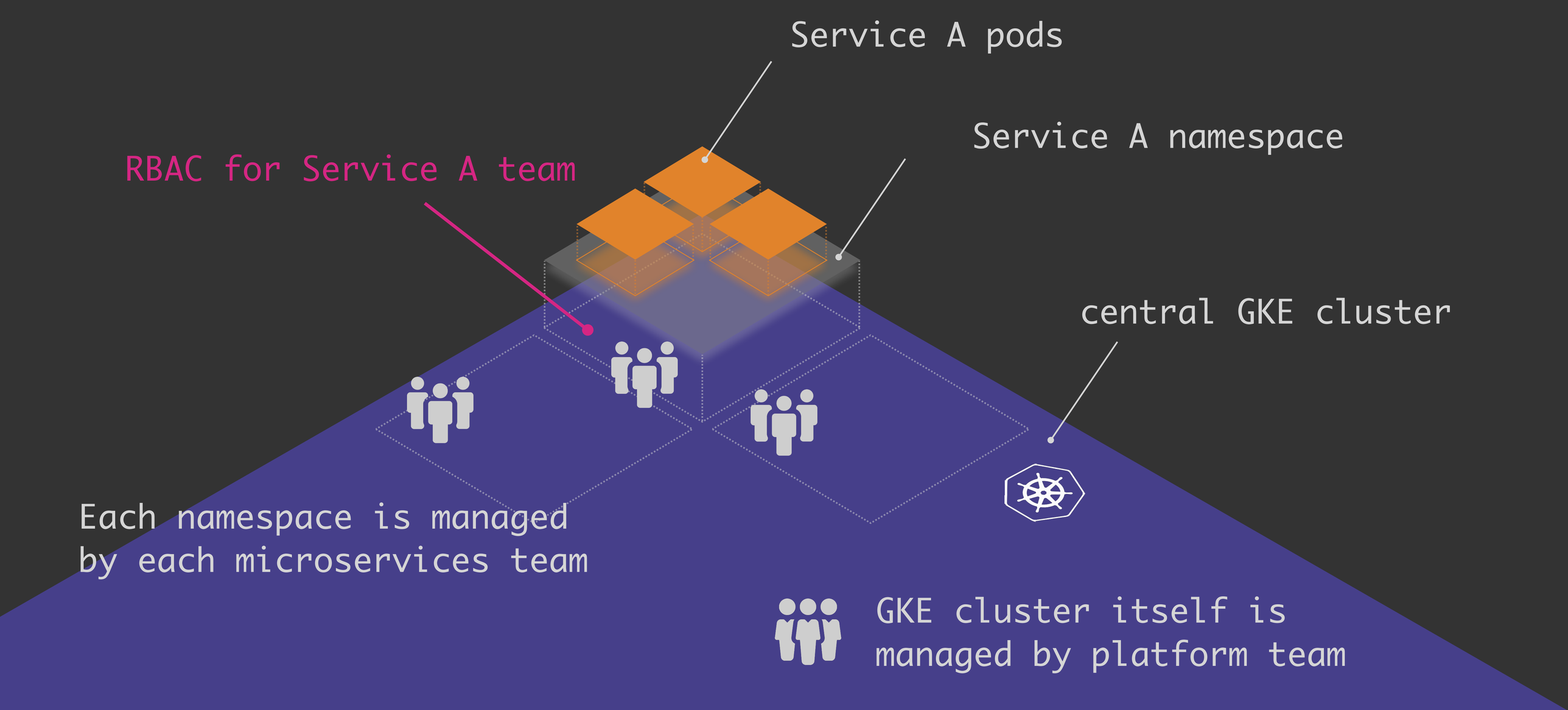
100+ microservices



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

```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

How should we review it?

How should we apply it?

How should we control it?

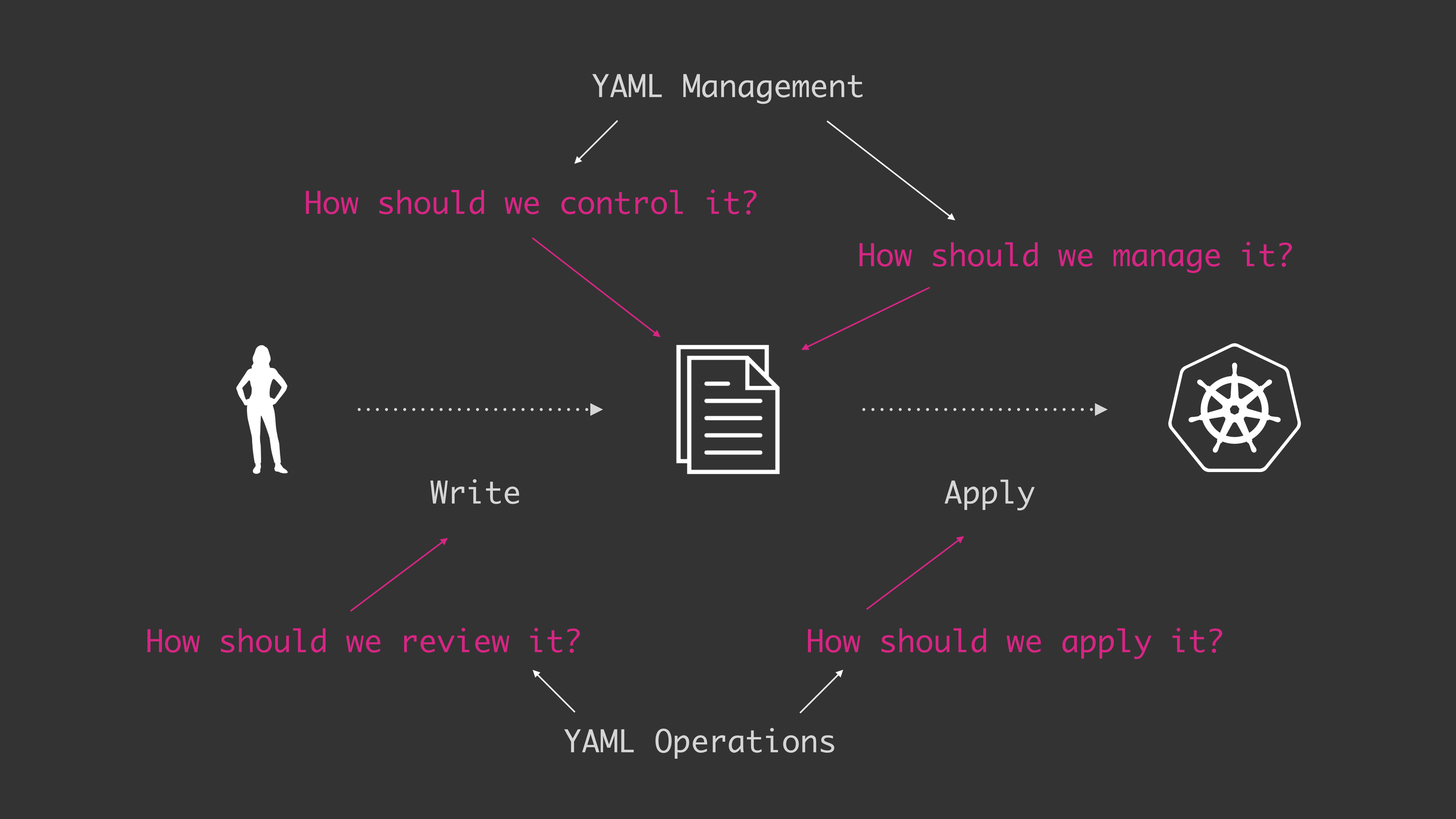
How should we manage it?

Write

Apply

How should we review it?

How should we apply it?


YAML Management
How should we control it?
How should we manage it?
Write
Apply
How should we review it?
How should we apply it?
YAML Operations

YAML Management

How should we control it?

How should we manage it?

Write

Apply

Our practices are…

How should we review it?

How should we apply it?

YAML Operations

4. Directories

~~How should we control it?~~

3. Monorepo

~~How should we manage it?~~

~~How should we review it?~~

1. Pull Requests

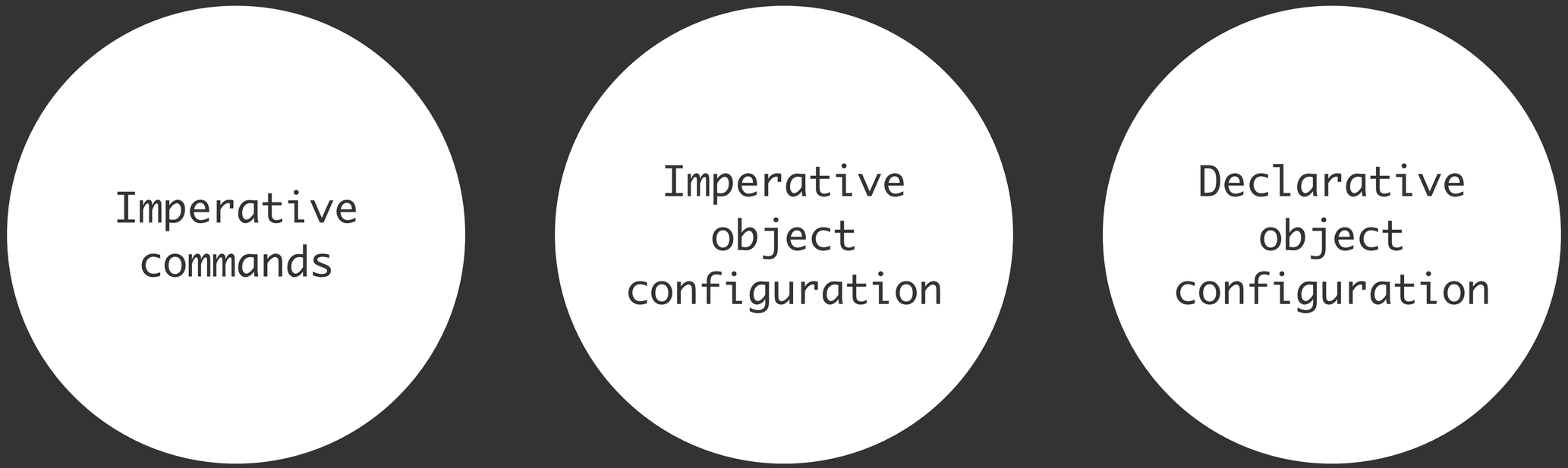
~~How should we apply it?~~

2. GitOps w/ kubectl

Let’s see the each details

1. GitHub Pull Requests

Kubernetes Object Management



https://kubernetes.io/docs/concepts/overview/object-management-kubectl/overview/

Kubernetes Object Management

kubectl run nginx --image nginx

Imperative object configuration

Declarative object configuration

https://kubernetes.io/docs/concepts/overview/object-management-kubectl/overview/

Kubernetes Object Management

kubectl run nginx --image nginx

kubectl create -f nginx.yaml

Declarative object configuration

https://kubernetes.io/docs/concepts/overview/object-management-kubectl/overview/

Kubernetes Object Management

https://kubernetes.io/docs/concepts/overview/object-management-kubectl/overview/

Kubernetes Object Management

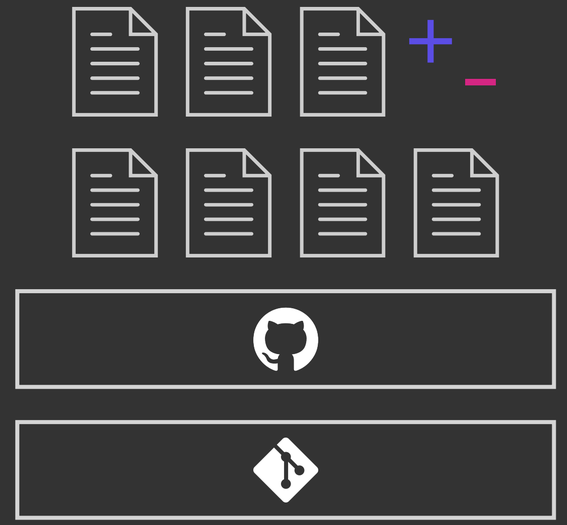
- It can be stored in VCS such as Git.
- It can integrate with processes such as reviewing changes.
- It has better support for operating on directories and automatically detecting operation types per-object.

https://kubernetes.io/docs/concepts/overview/object-management-kubectl/overview/

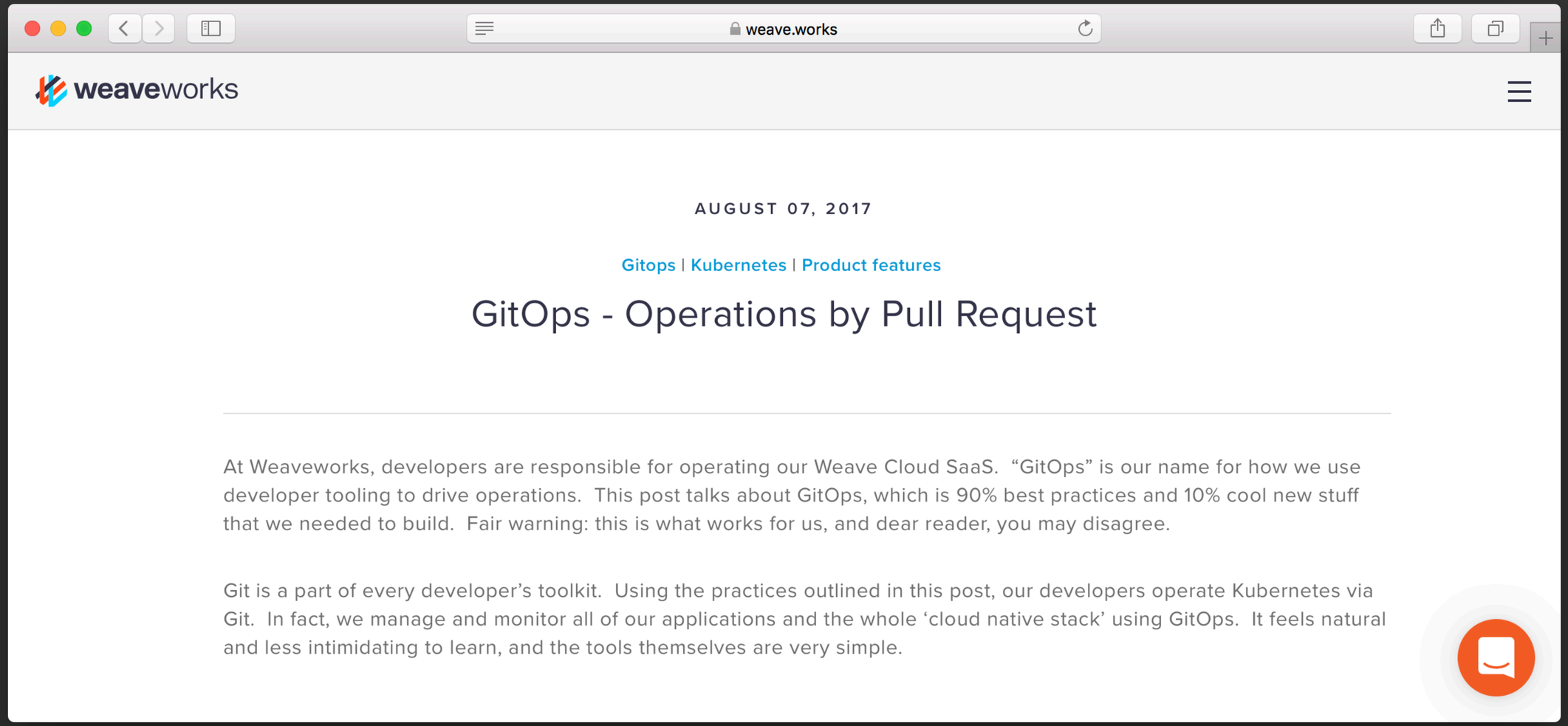
Pull Requests

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

2. GitOps

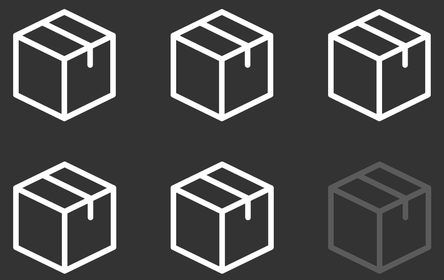


GitOps - Operations by Pull Requests

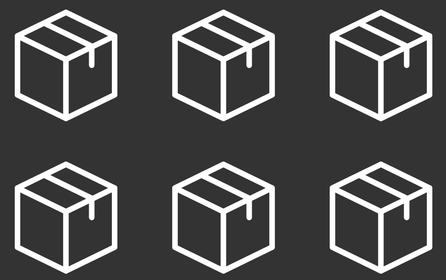


https://www.weave.works/blog/gitops-operations-by-pull-request

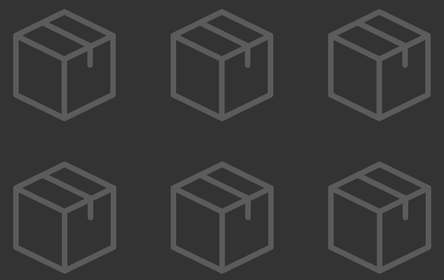
Start 6 servers



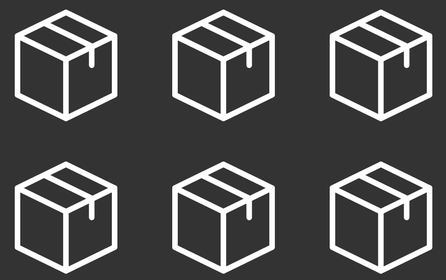
There are 6 servers



Start 6 servers



There are 6 servers



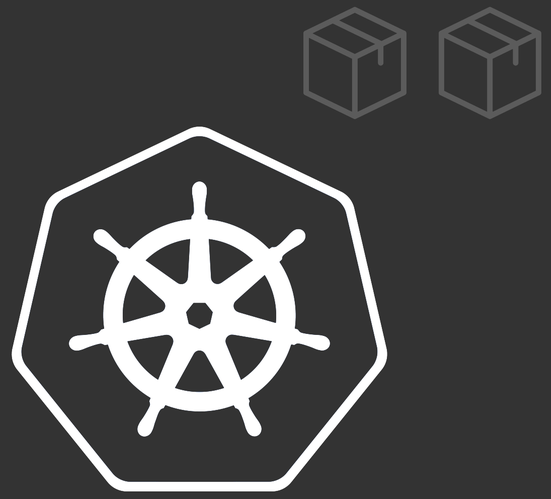
Let’s see Weaveworks case

Kubernetes Cluster

GitHub Repository

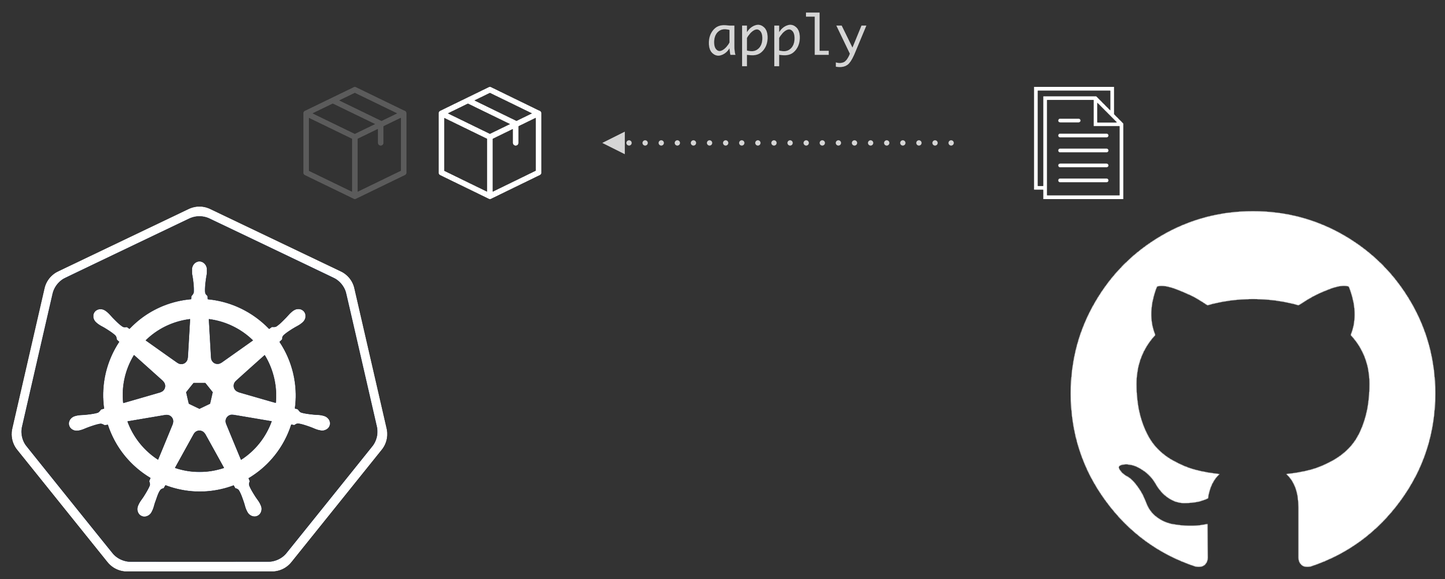
Live Objects

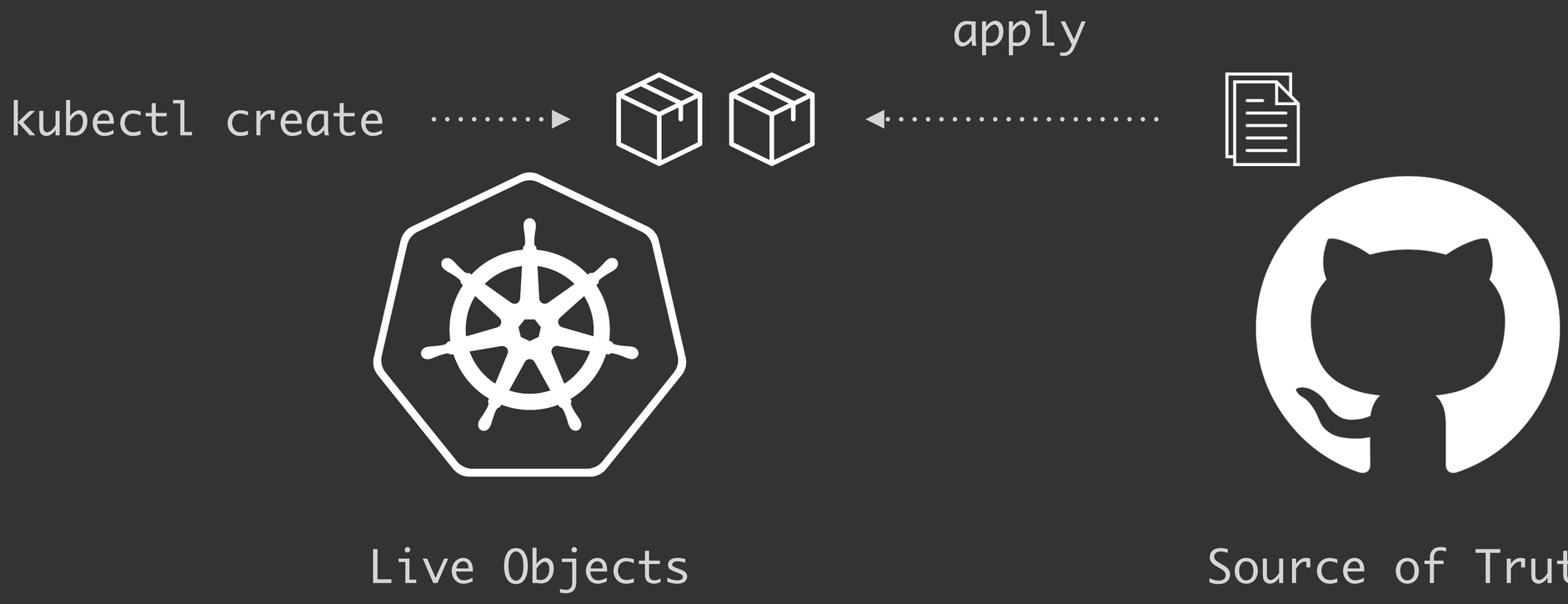
Source of Truth

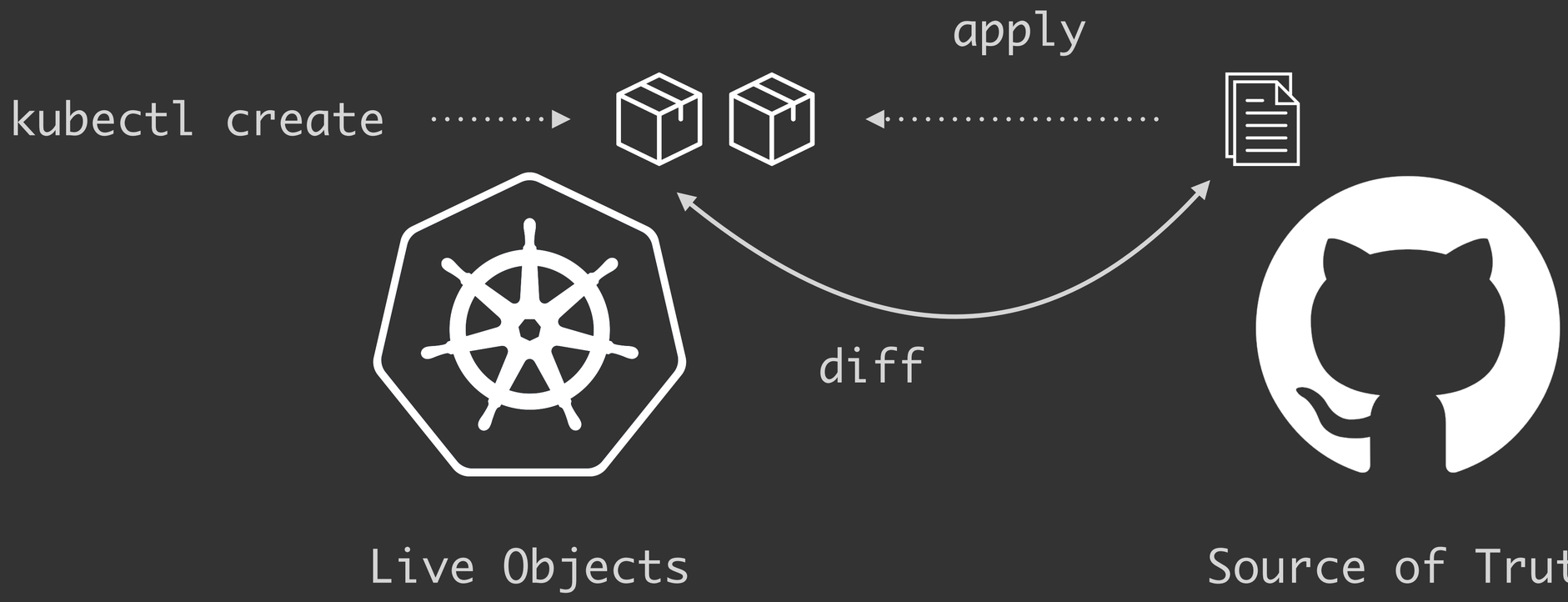


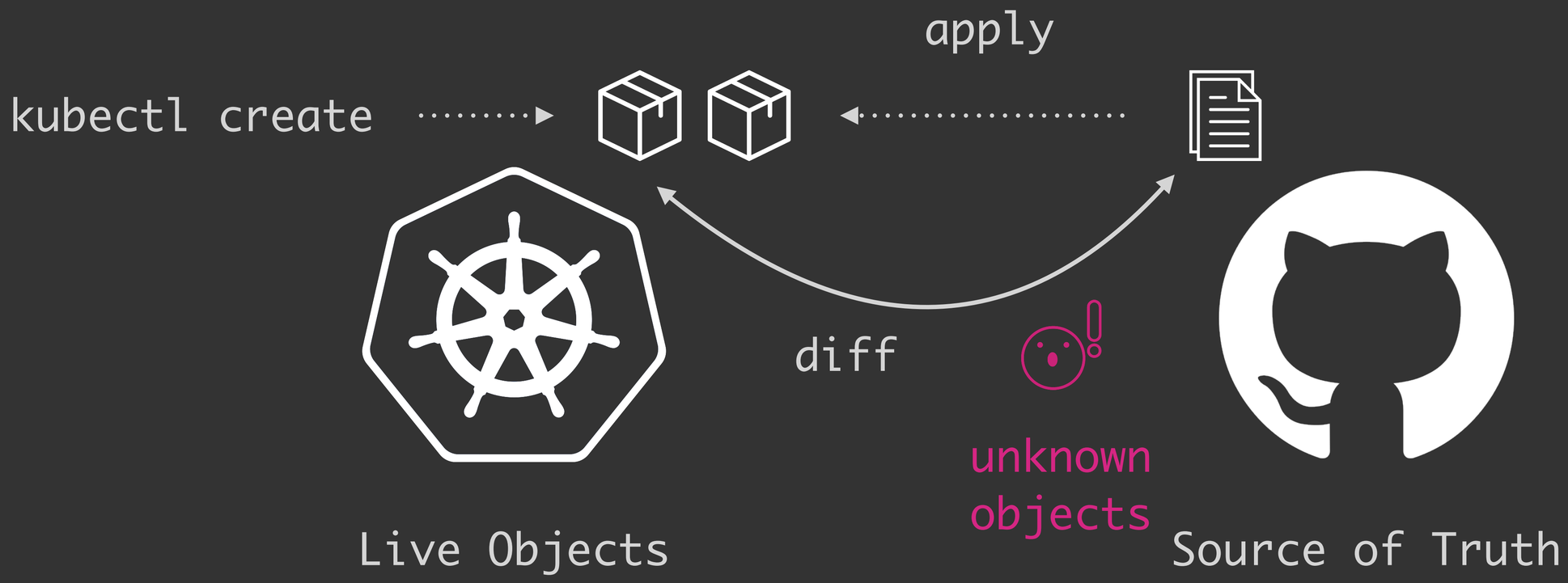
Live Objects

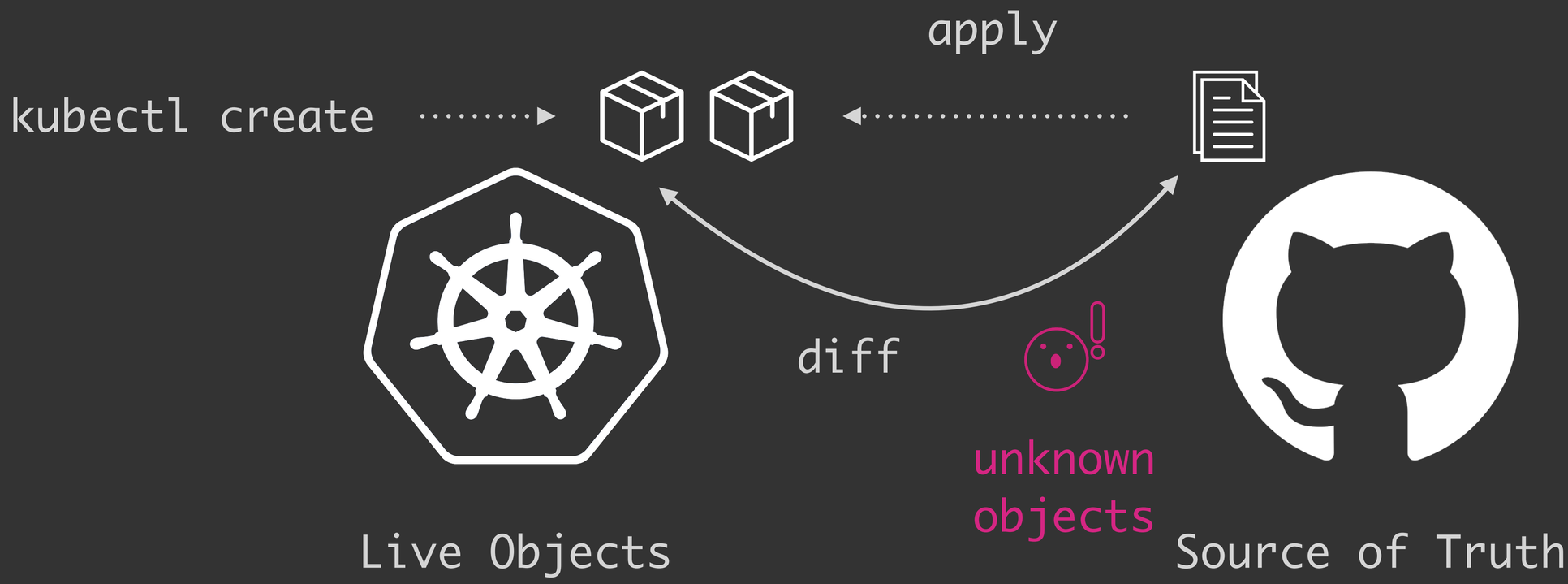
Source of Truth


apply
Live Objects
Source of Truth


apply
kubectl create
Live Objects
Source of Truth

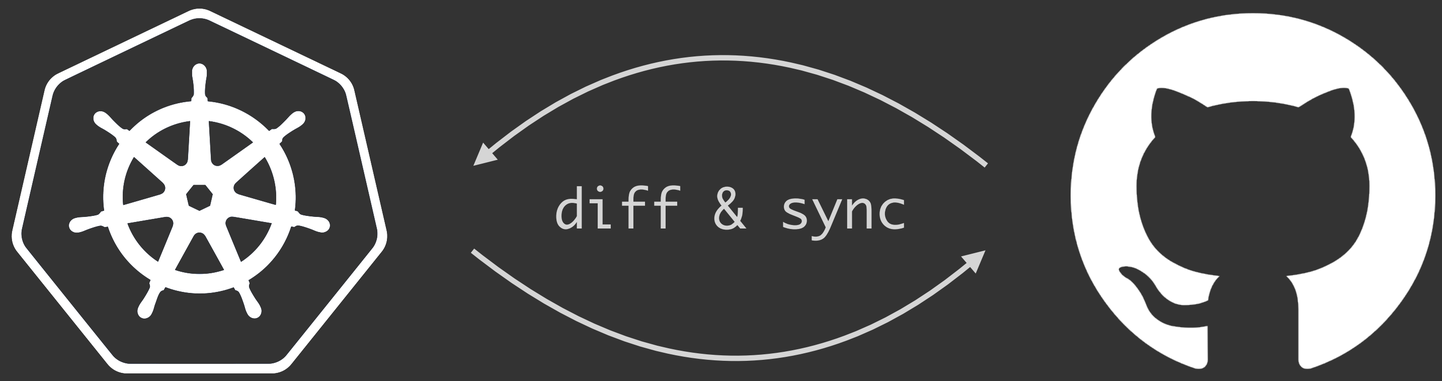

apply
kubectl create
diff
Live Objects
Source of Truth


apply
kubectl create
diff
unknown
objects
Live Objects
Source of Truth



Git is the source of truth

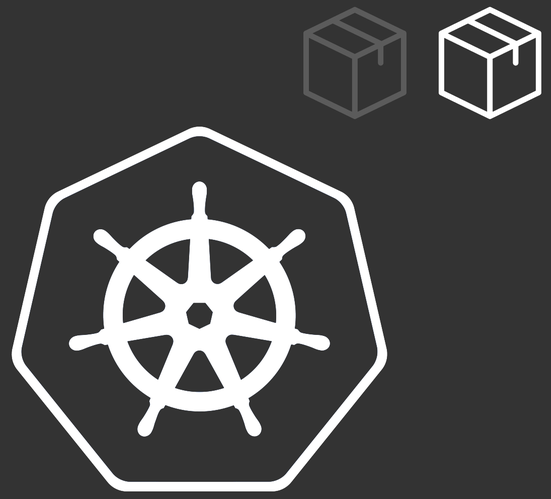
Pull strategy



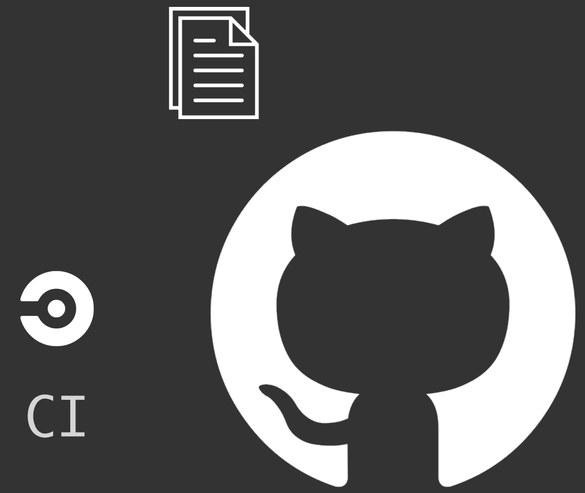
Kubernetes Cluster

GitHub Repository

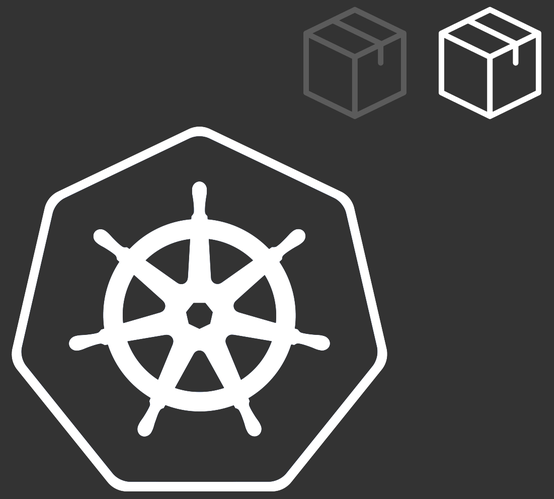
Let’s see our case



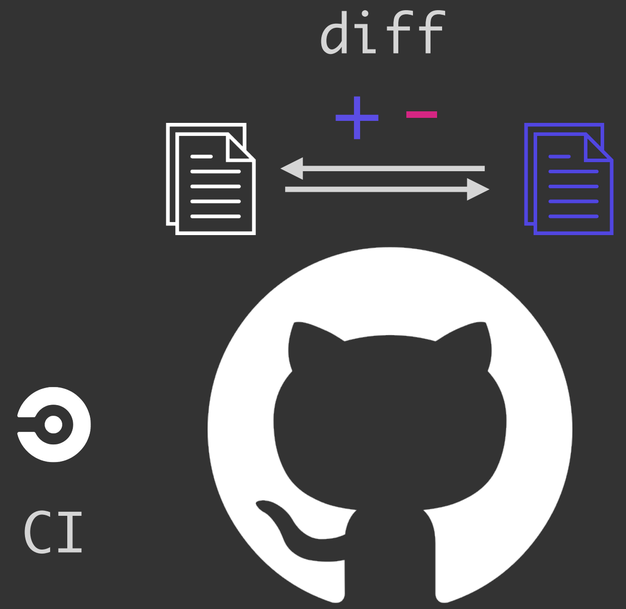
Kubernetes Cluster



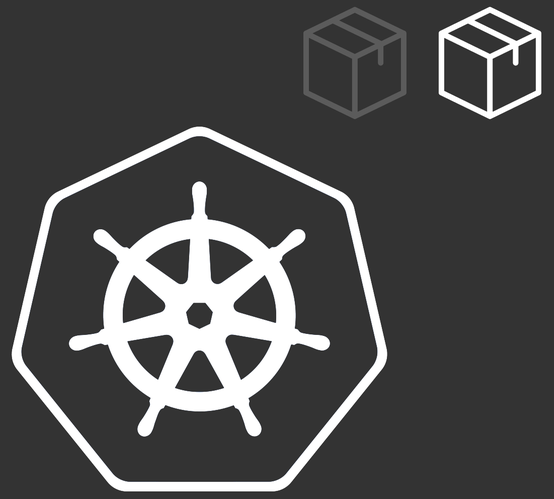
GitHub Repository



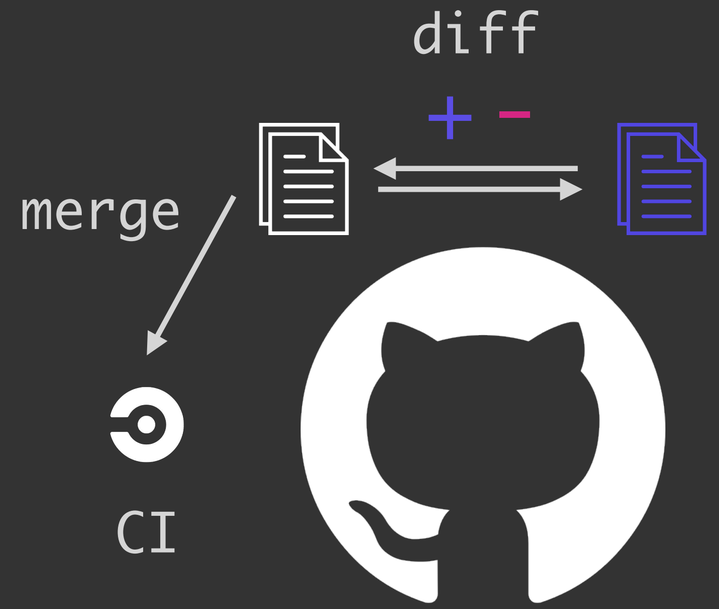
Kubernetes Cluster



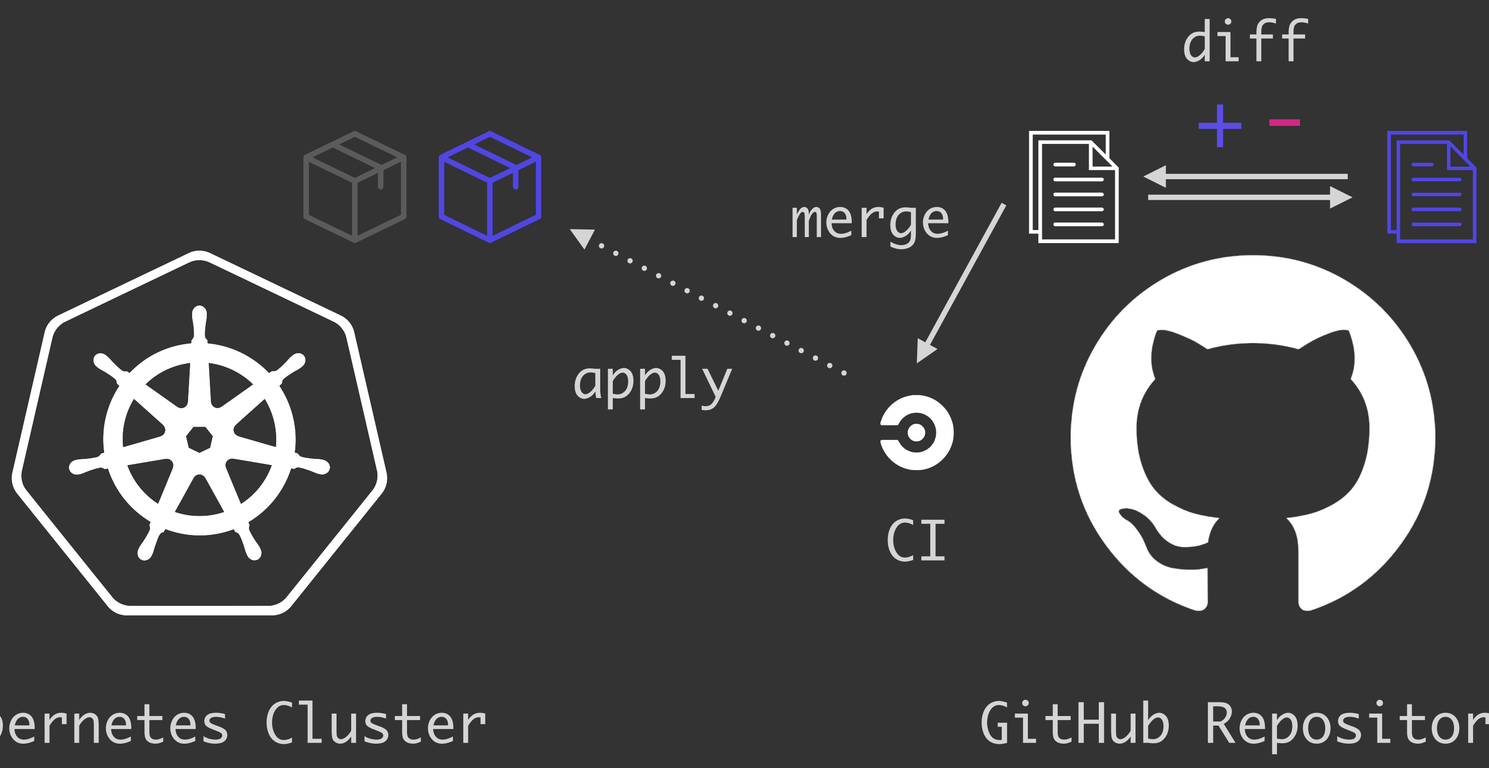
GitHub Repository

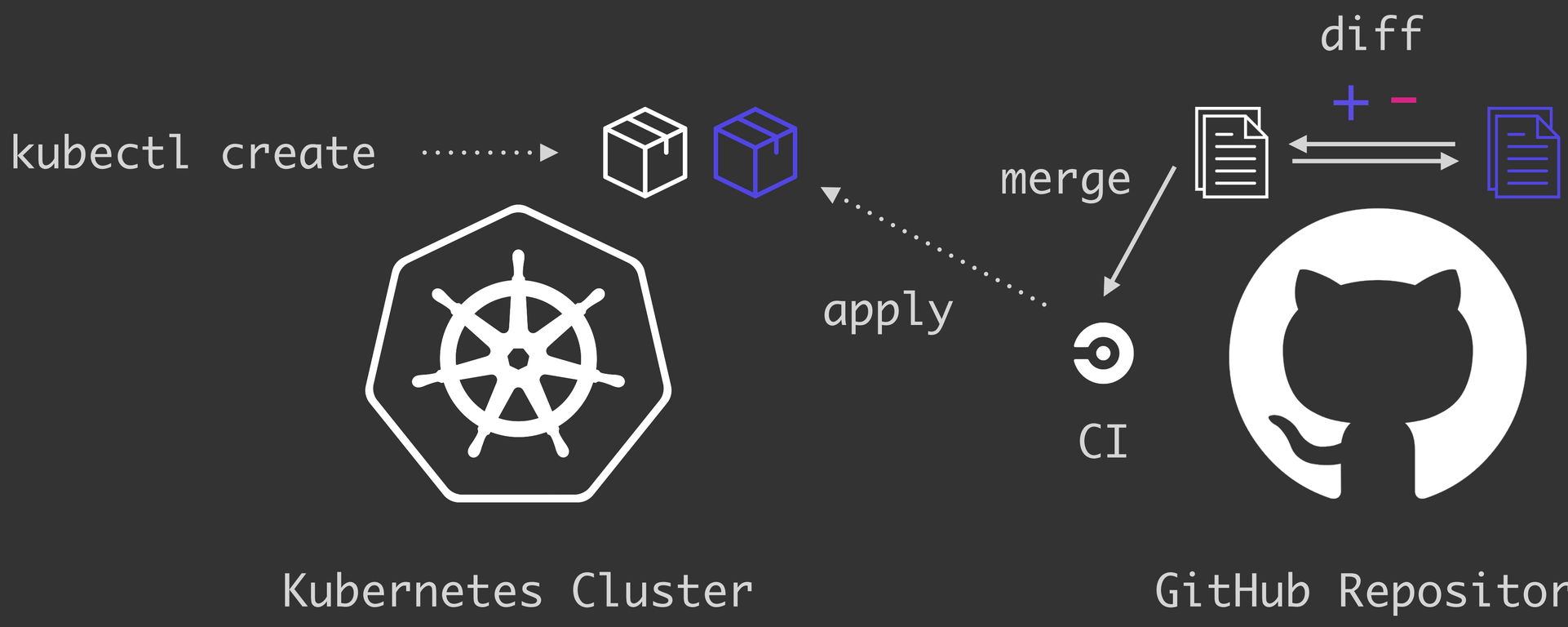


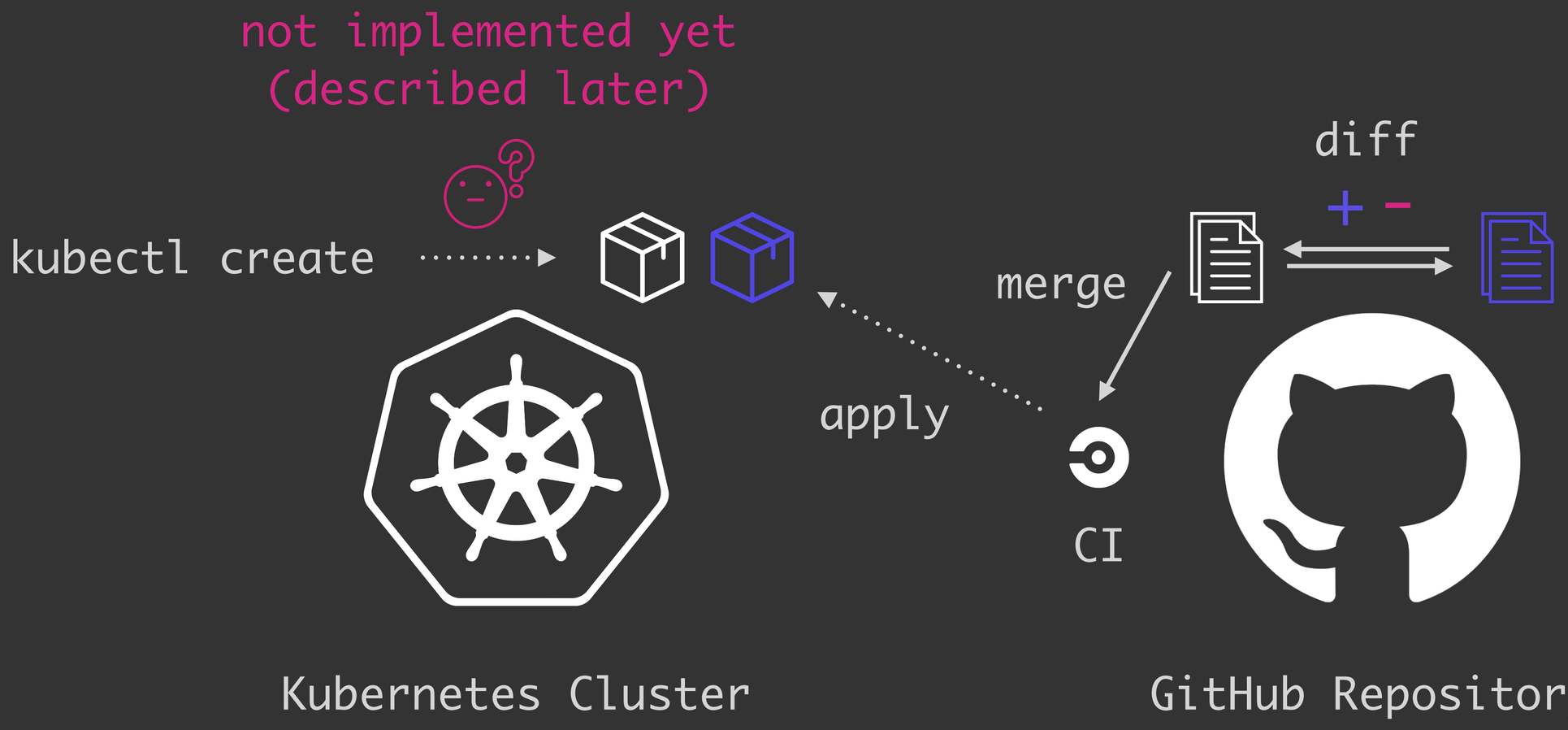
Kubernetes Cluster



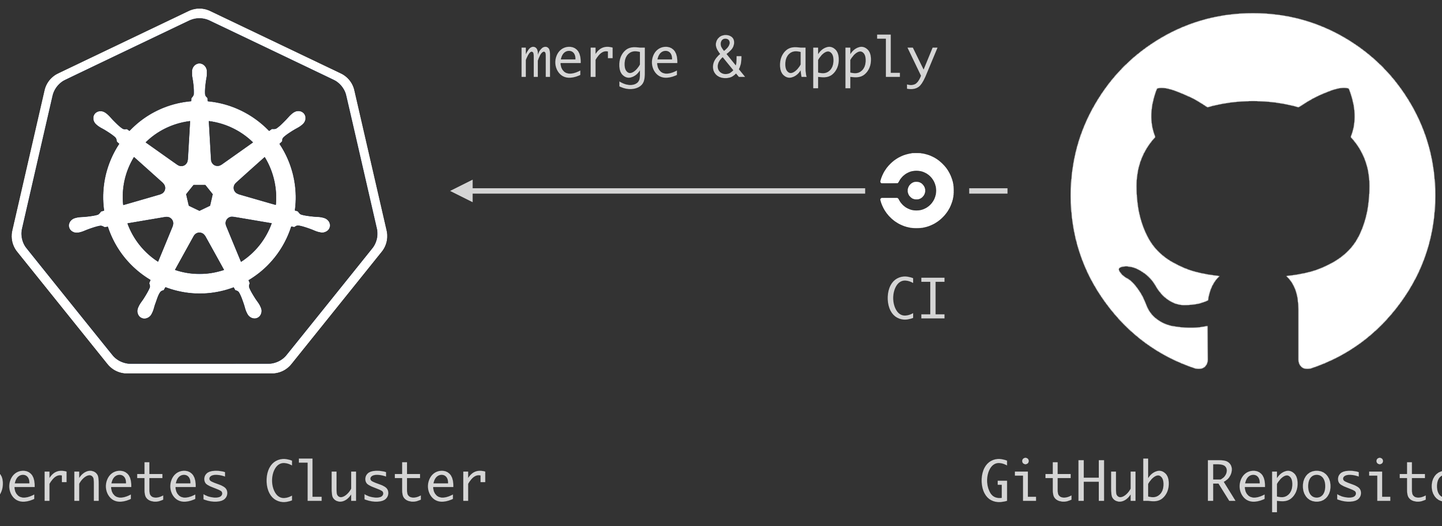
GitHub Repository


diff
+ −
merge
apply
CI
Kubernetes Cluster
GitHub Repository


kubectl create
diff
+ −
merge
apply
CI
Kubernetes Cluster
GitHub Repository

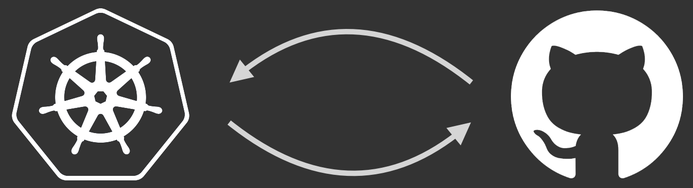

not implemented yet
(described later)
kubectl create
diff
+ -
merge
apply
CI
Kubernetes Cluster
GitHub Repository

Push strategy



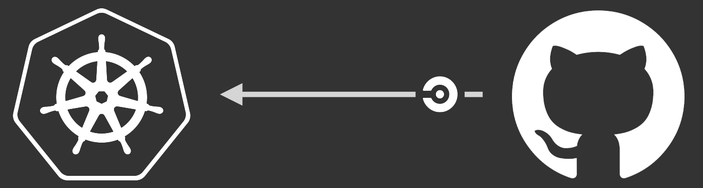
Let’s see the each differences

Pull (Weaveworks case)



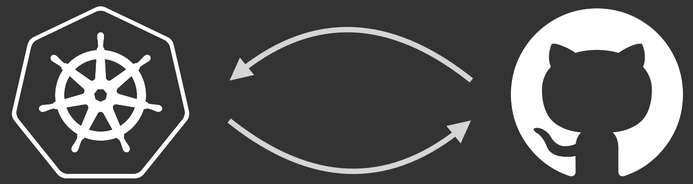
- Git can be the single source of truth
- Bit difficult to implement the sync pipeline

Push (Mercari case)

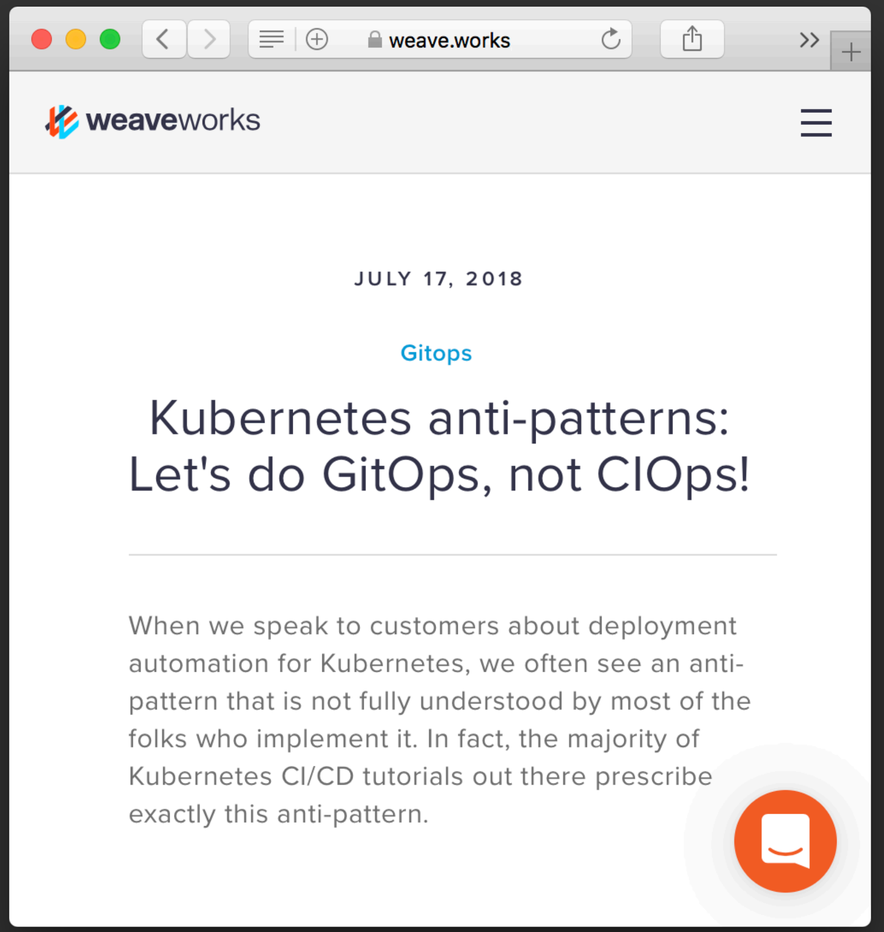


- Difficult to find divergence from the source
- A common way of applying the changes

Pull (Weaveworks case)



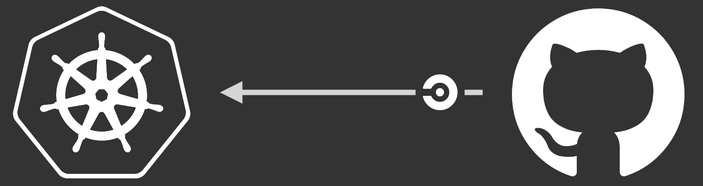
- Git can be the single source of truth
- Bit difficult to implement the sync pipeline



We know they recommend "Pull" strategy

https://www.weave.works/blog/kubernetes-anti-patterns-let-s-do-gitops-not-ciops

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?

Simple

Easy to implement

Enough to start firstly

Spinnaker

Another most bigger reason is...

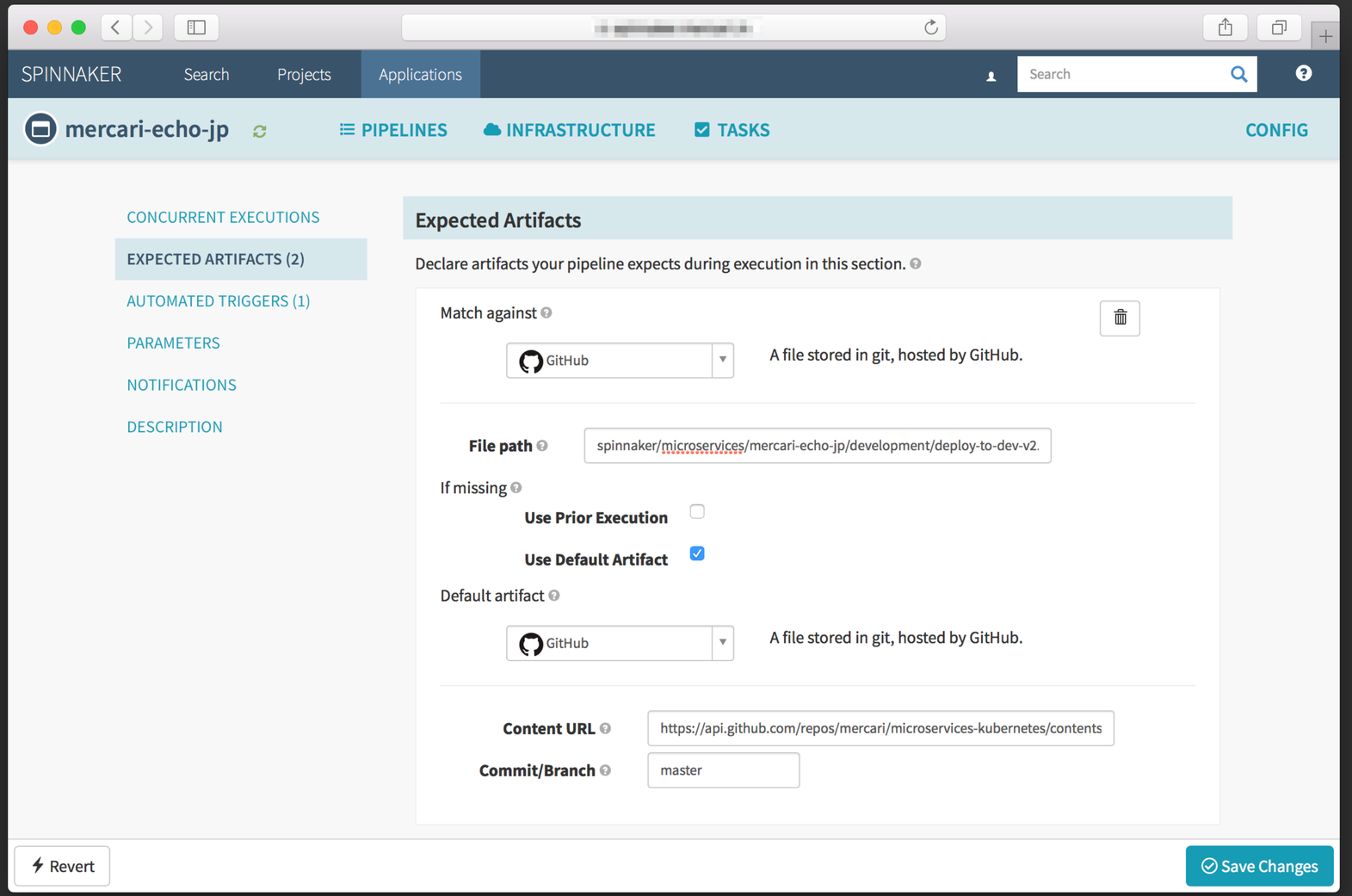
Why using Spinnaker also?

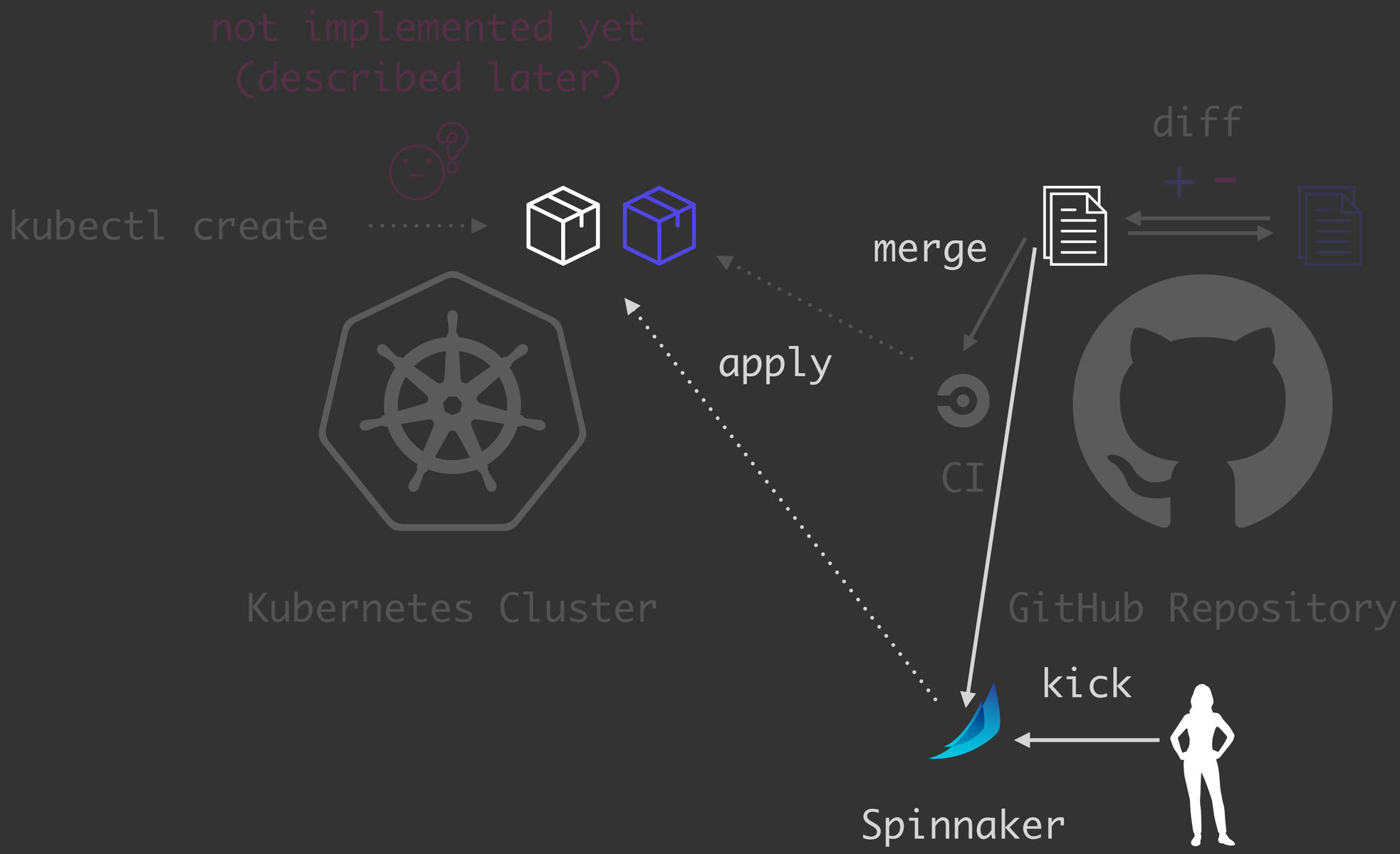
- Our CI pipeline which runs "kubectl apply" based on the changes is triggered by merging pull requests
- However, in some resources (Job etc), we want to apply in our timing

- Spinnaker "Provider V2" can handle Kubernetes manifests declaratively

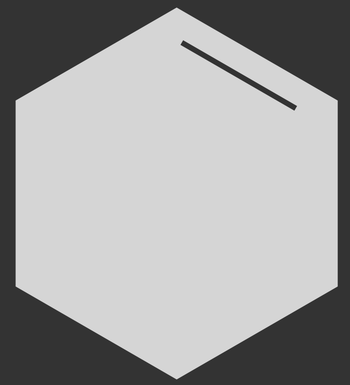
Spinnaker

https://www.spinnaker.io/reference/providers/kubernetes-v2/

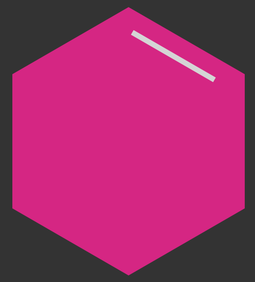

SPINNAKER
Search
Projects
Applications
Search
mercari-echo-jp
PIPELINES
INFRASTRUCTURE
TASKS
CONFIG
CONCURRENT EXECUTIONS
EXPECTED ARTIFACTS (2)
AUTOMATED TRIGGERS (1)
PARAMETERS
NOTIFICATIONS
DESCRIPTION
Expected Artifacts
Declare artifacts your pipeline expects during execution in this section.
Match against
GitHub
A file stored in git, hosted by GitHub.
File path
spinnaker/microservices/mercari-echo-jp/development/deploy-to-dev-v2.
If missing
Use Prior Execution
Use Default Artifact
Default artifact
GitHub
A file stored in git, hosted by GitHub.
Content URL
https://api.github.com/repos/mercari/microservices-kubernetes/contents
Commit/Branch
master
Revert
Save Changes


not implemented yet
(described later)
diff
+ -
kubectl create
merge
apply
CI
Kubernetes Cluster
GitHub Repository
kick
Spinnaker

3. Monorepo

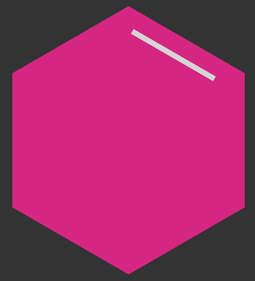


Two type of repository styles



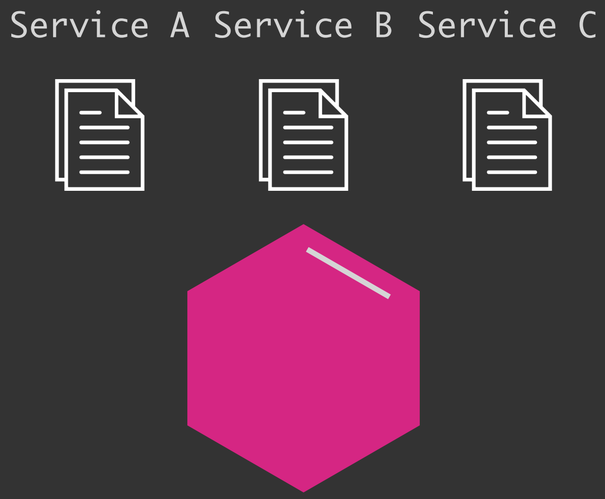
One repository

Multiple repositories

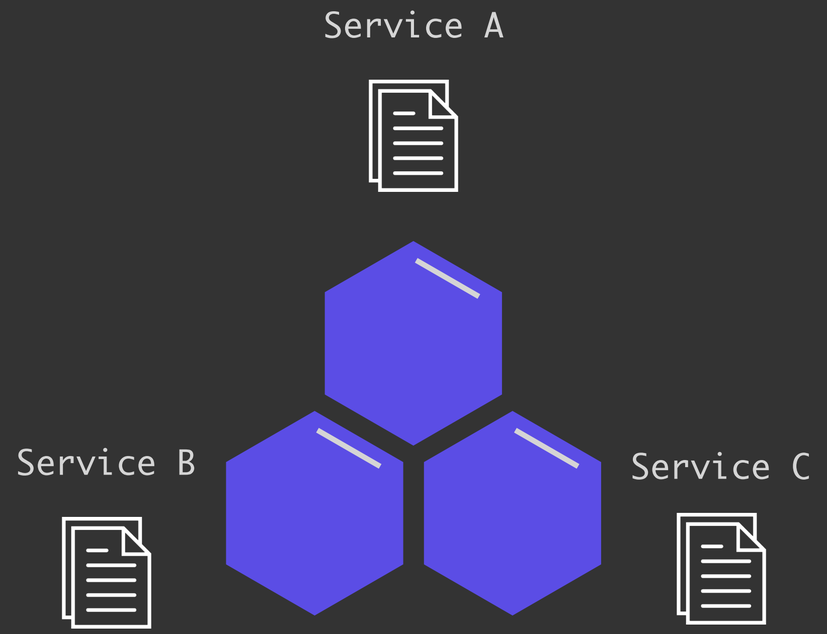


Monorepo

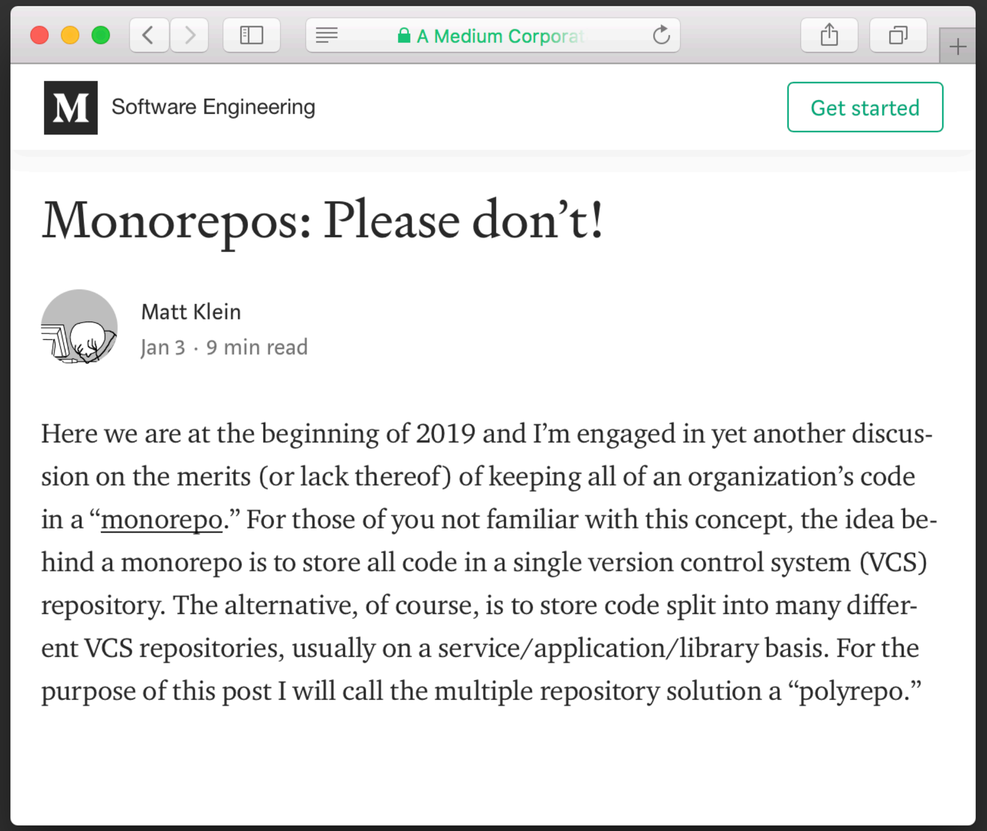
Polyrepo

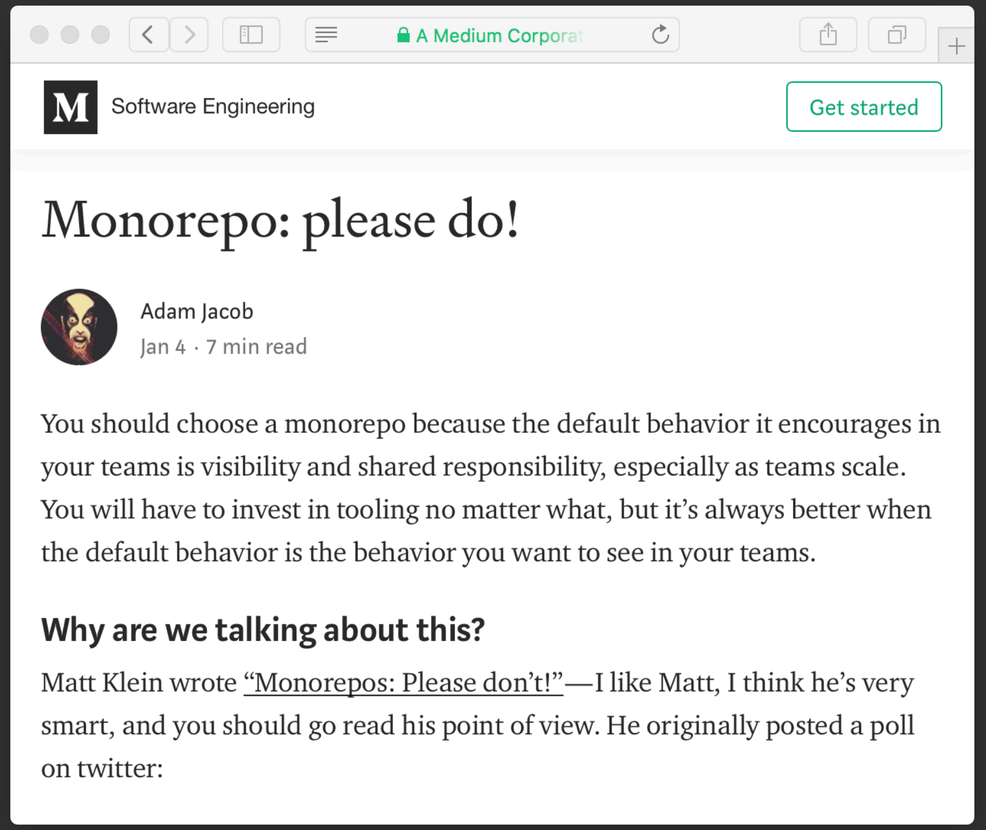


Monorepo



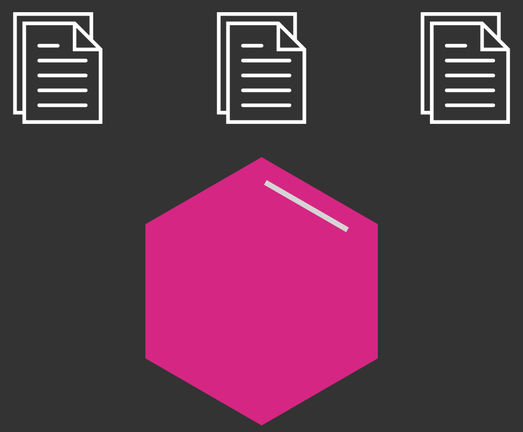
Polyrepo





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

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



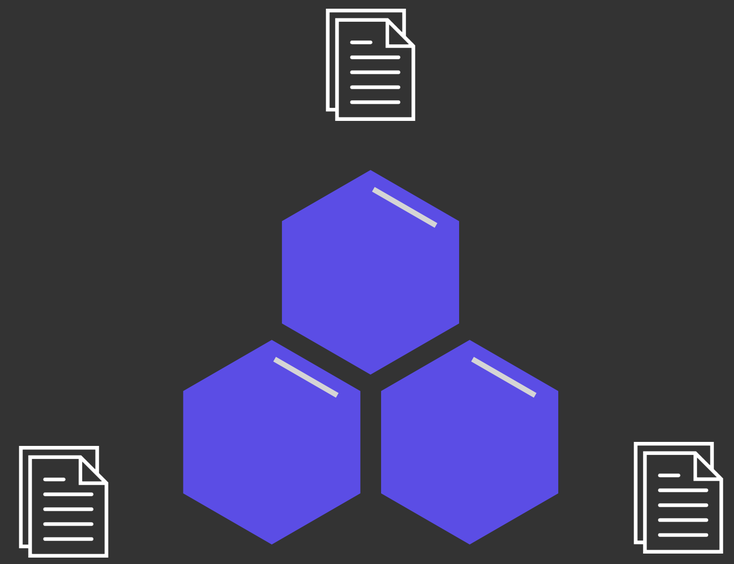
Monorepo

- 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

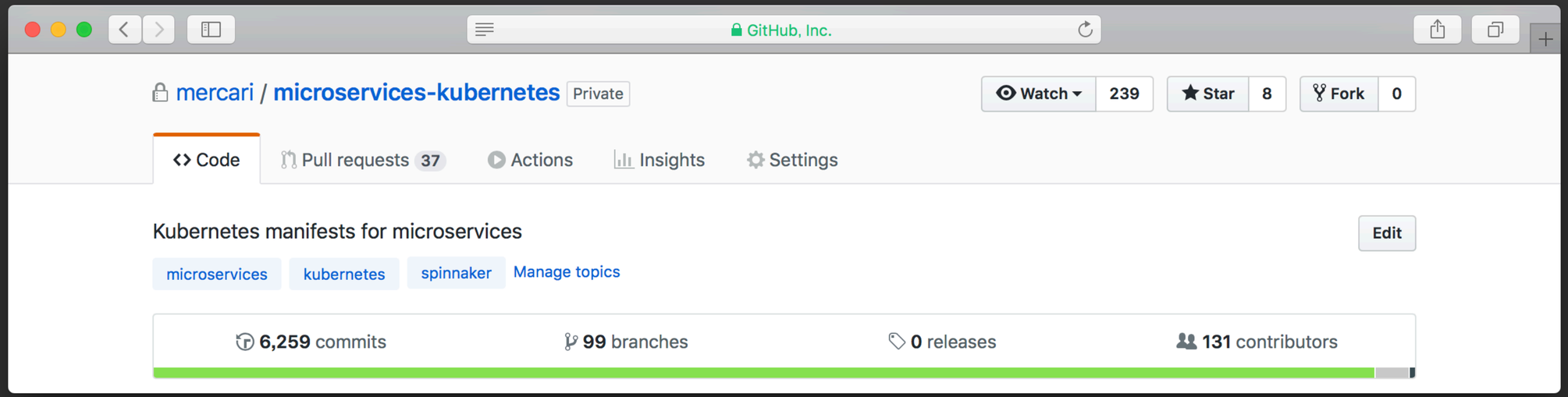
- Disadvantages
  - Take account into repo scale up
  - Take account into delegation of authority
  - CI/CD: No independence in each team

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

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

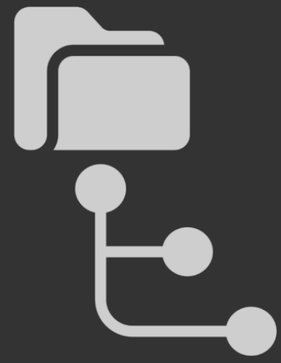


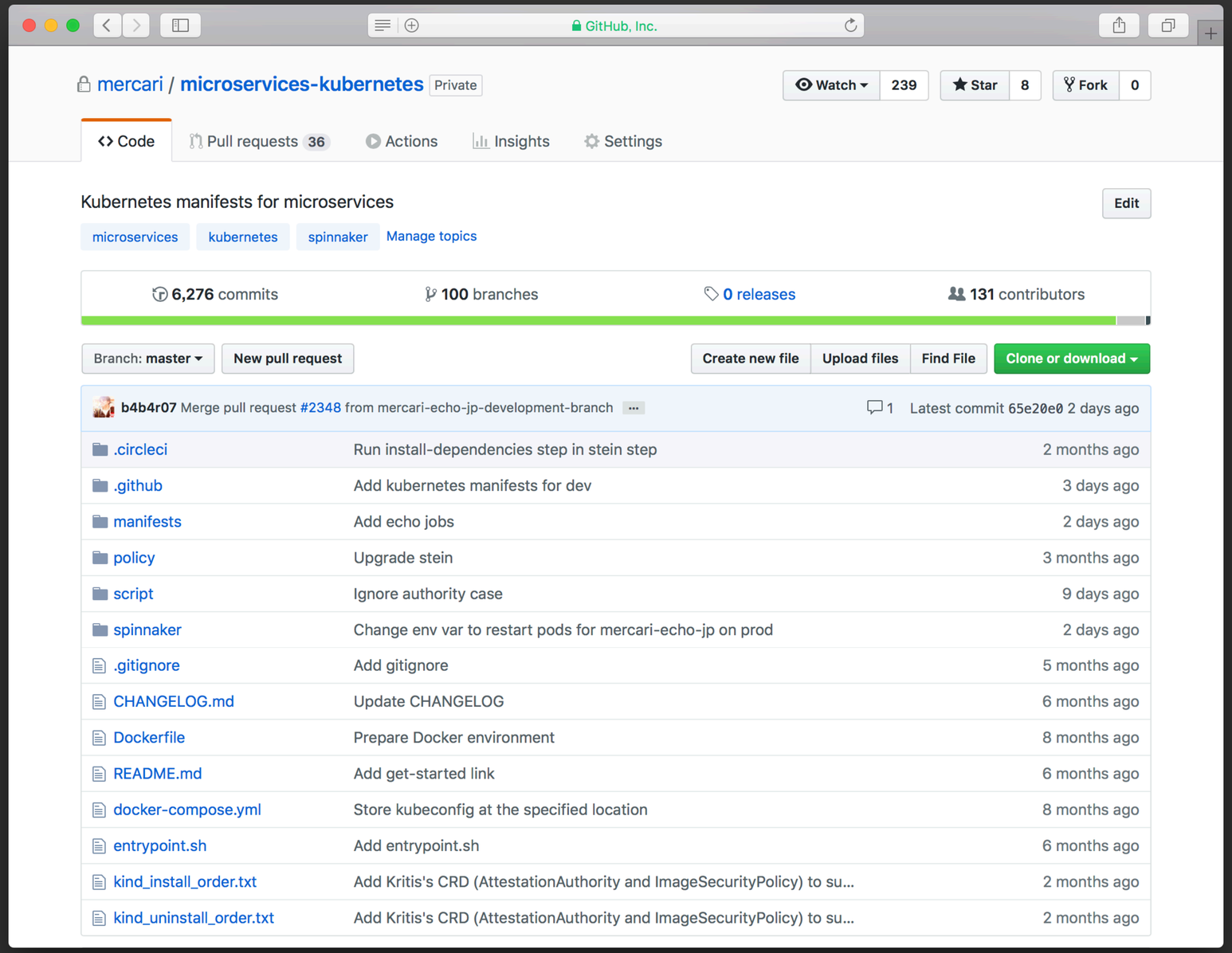
Polyrepo

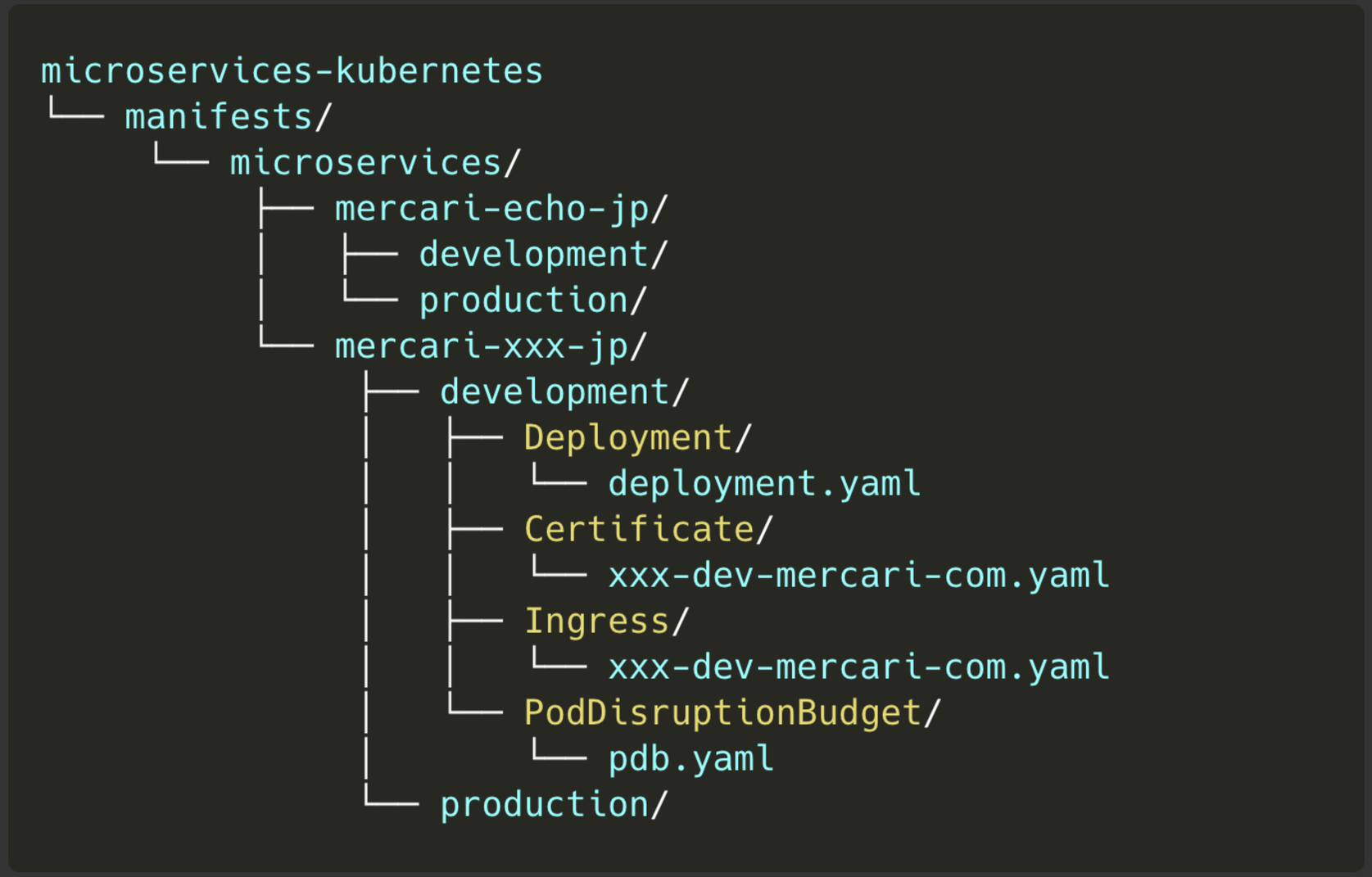


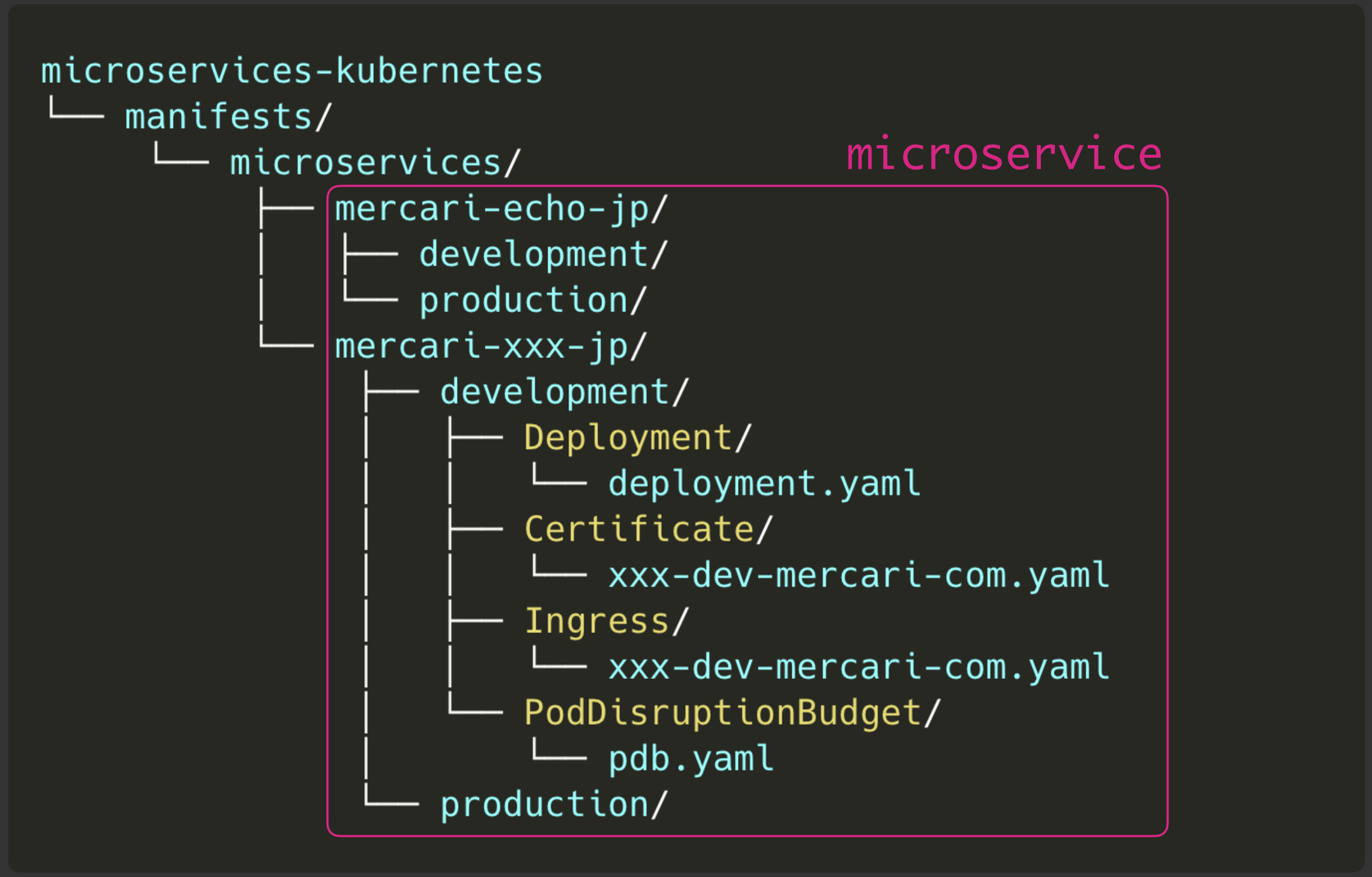
- We chose "Monorepo" style.
- It was a good option to start small.
- The concrete reason will be shown in next section.

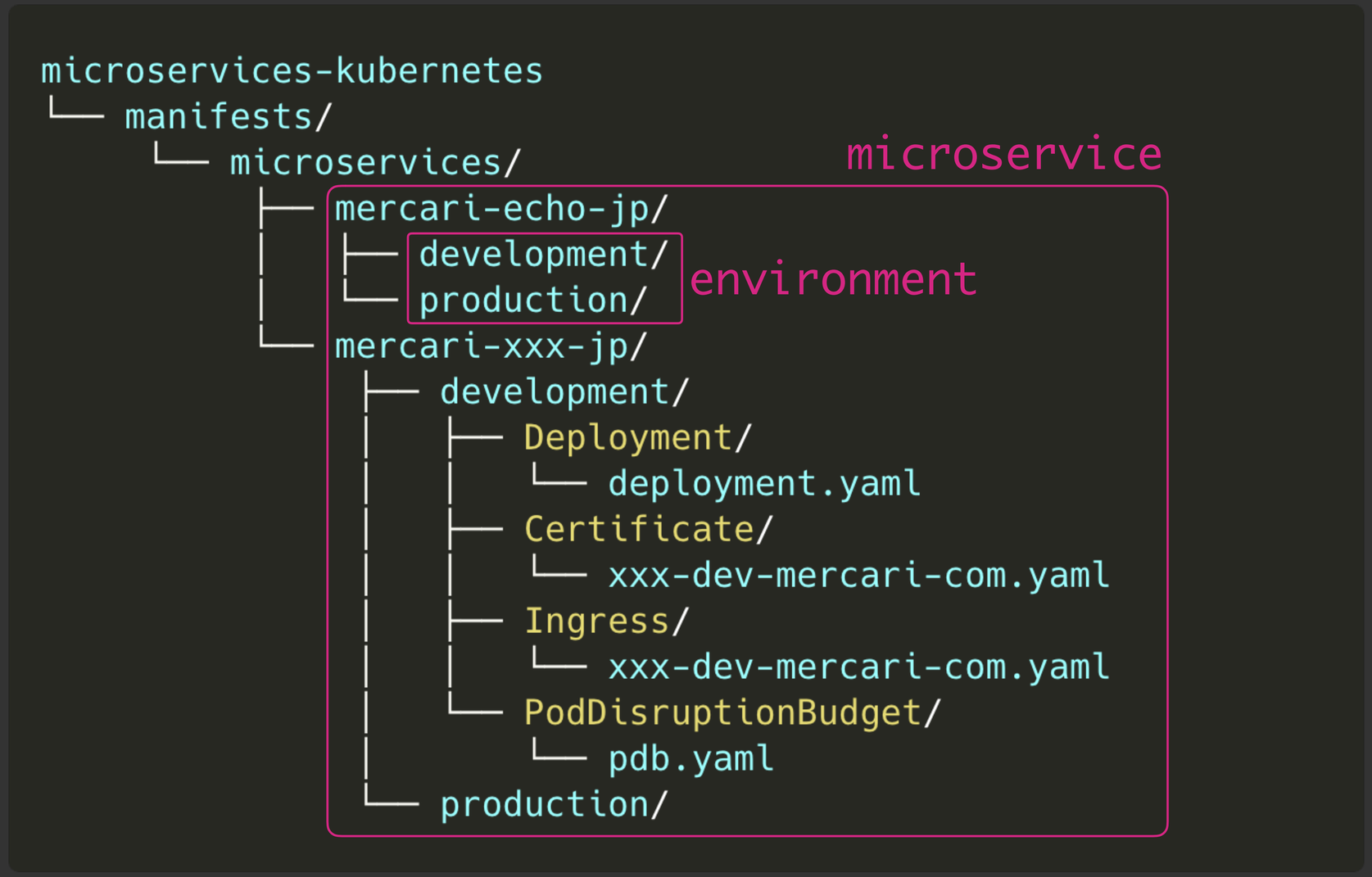
4. Directories

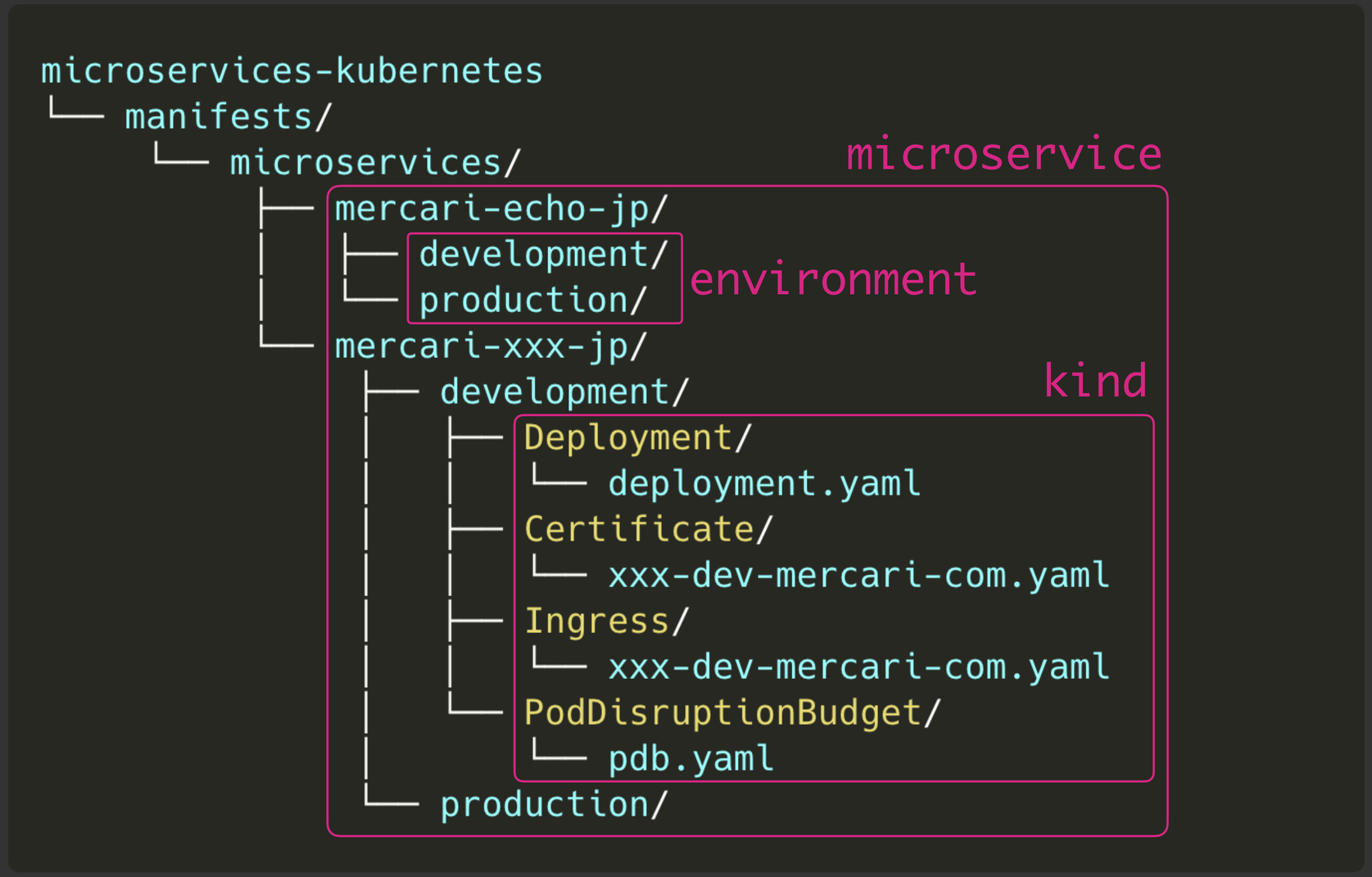


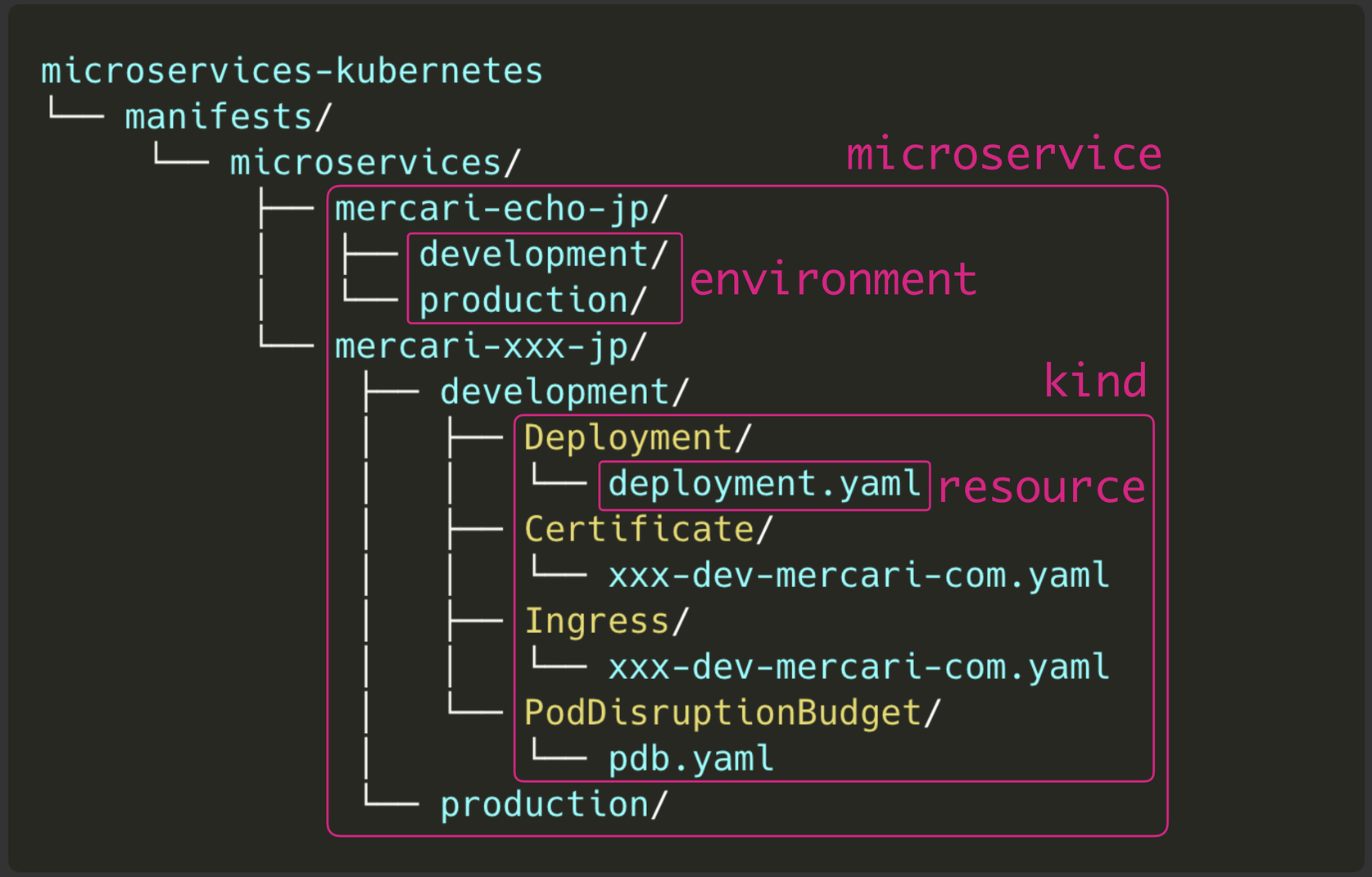

GitHub, Inc.
mercari / microservices-kubernetes Private
Watch 239
Star 8
Fork 0
Code
Pull requests 36
Actions
Insights
Settings
Kubernetes manifests for microservices
Edit
microservices
kubernetes
spinnaker
Manage topics
6,276 commits
100 branches
0 releases
131 contributors
Branch: master
New pull request
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/
xxx-dev-mercari-com.yaml
PodDisruptionBudget/
pdb.yaml
production/

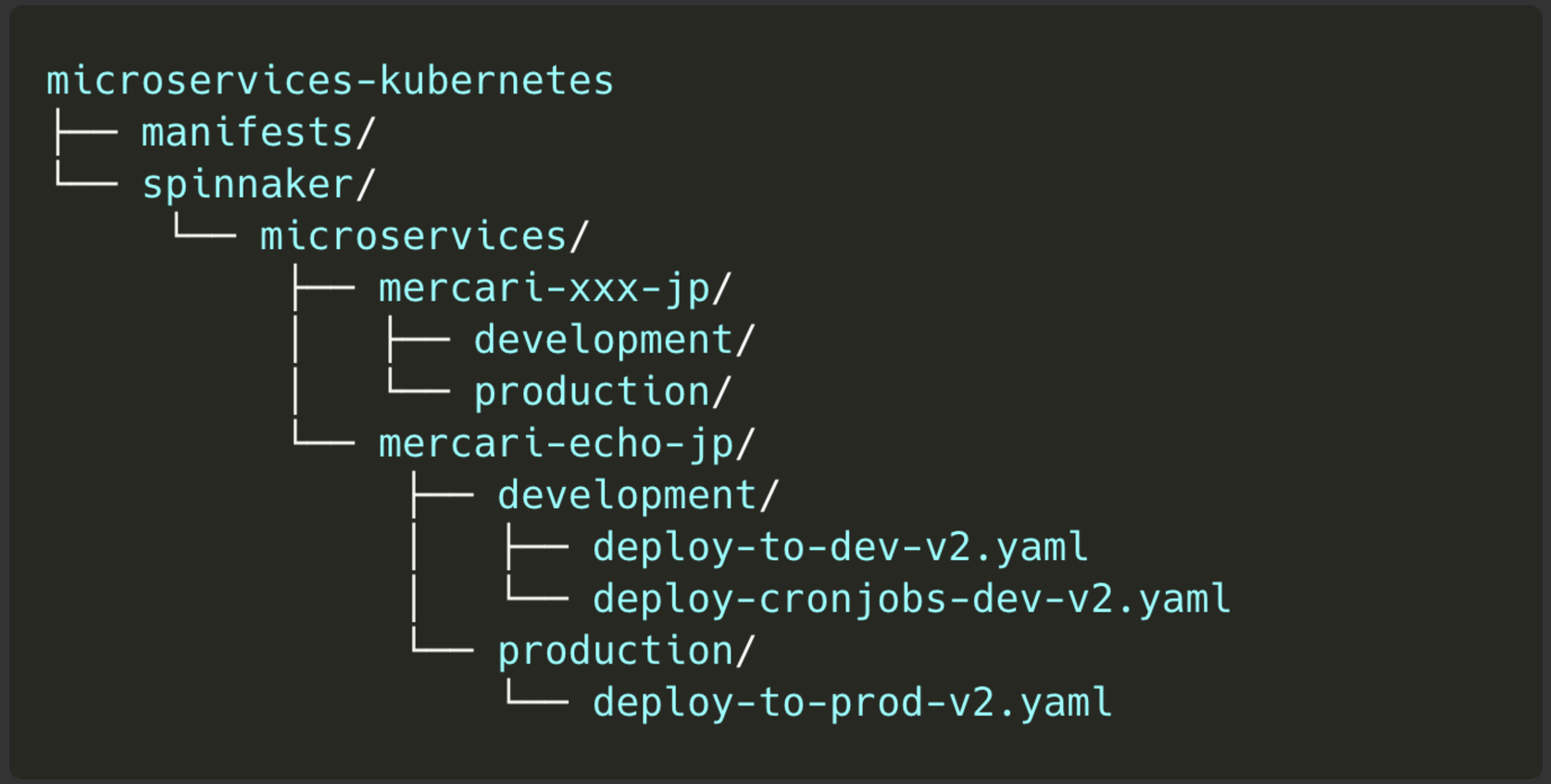

microservices-kubernetes
manifests/
microservices/
microservice
mercari-echo-jp/
development/
production/
mercari-xxx-jp/
development/
Deployment/
deployment.yaml
Certificate/
xxx-dev-mercari-com.yaml
Ingress/
xxx-dev-mercari-com.yaml
PodDisruptionBudget/
pdb.yaml
production/

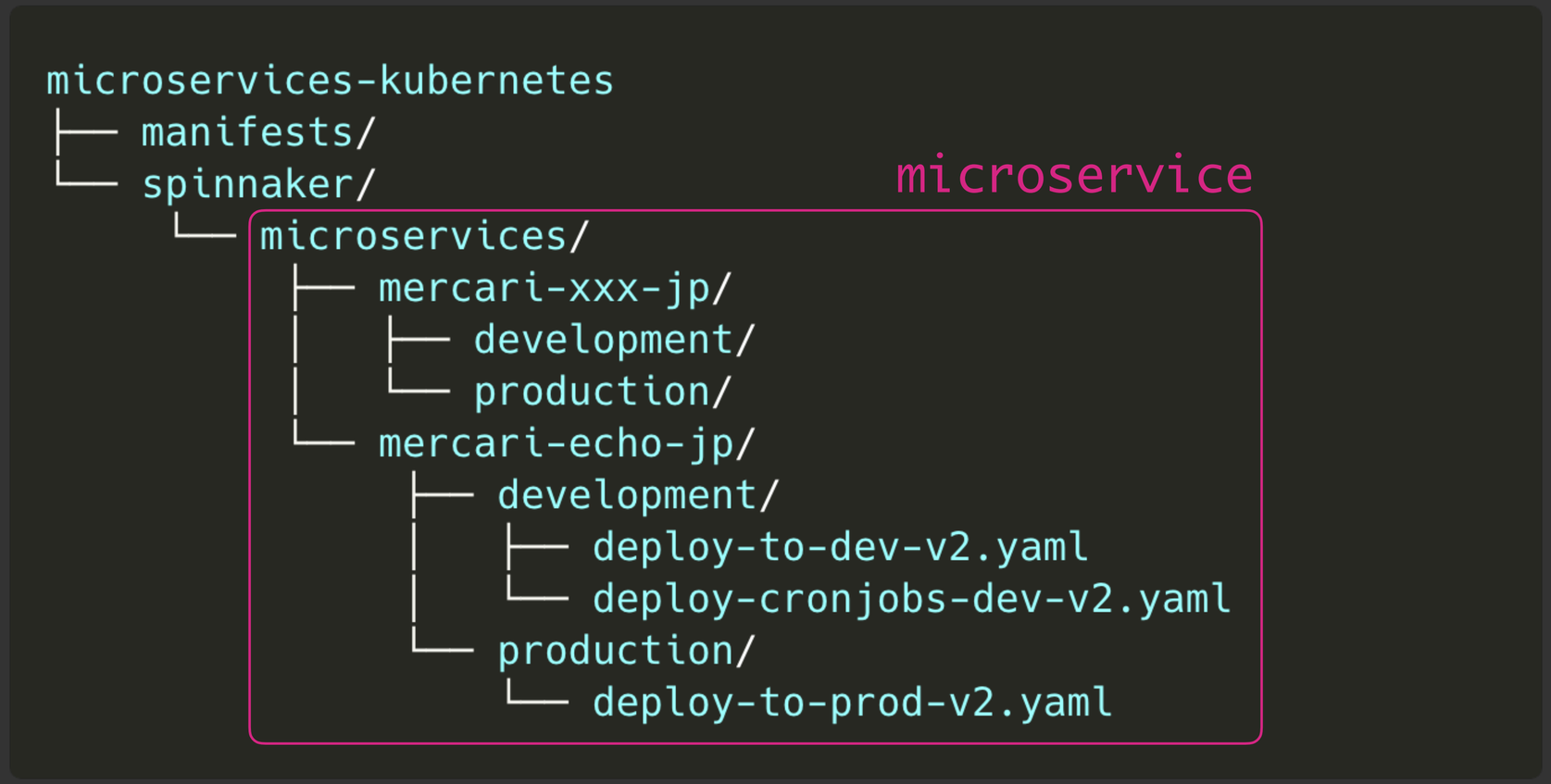

microservices-kubernetes
manifests/
microservices/
microservice
mercari-echo-jp/
development/
production/
environment
mercari-xxx-jp/
development/
Deployment/
deployment.yaml
Certificate/
xxx-dev-mercari-com.yaml
Ingress/
xxx-dev-mercari-com.yaml
PodDisruptionBudget/
pdb.yaml
production/

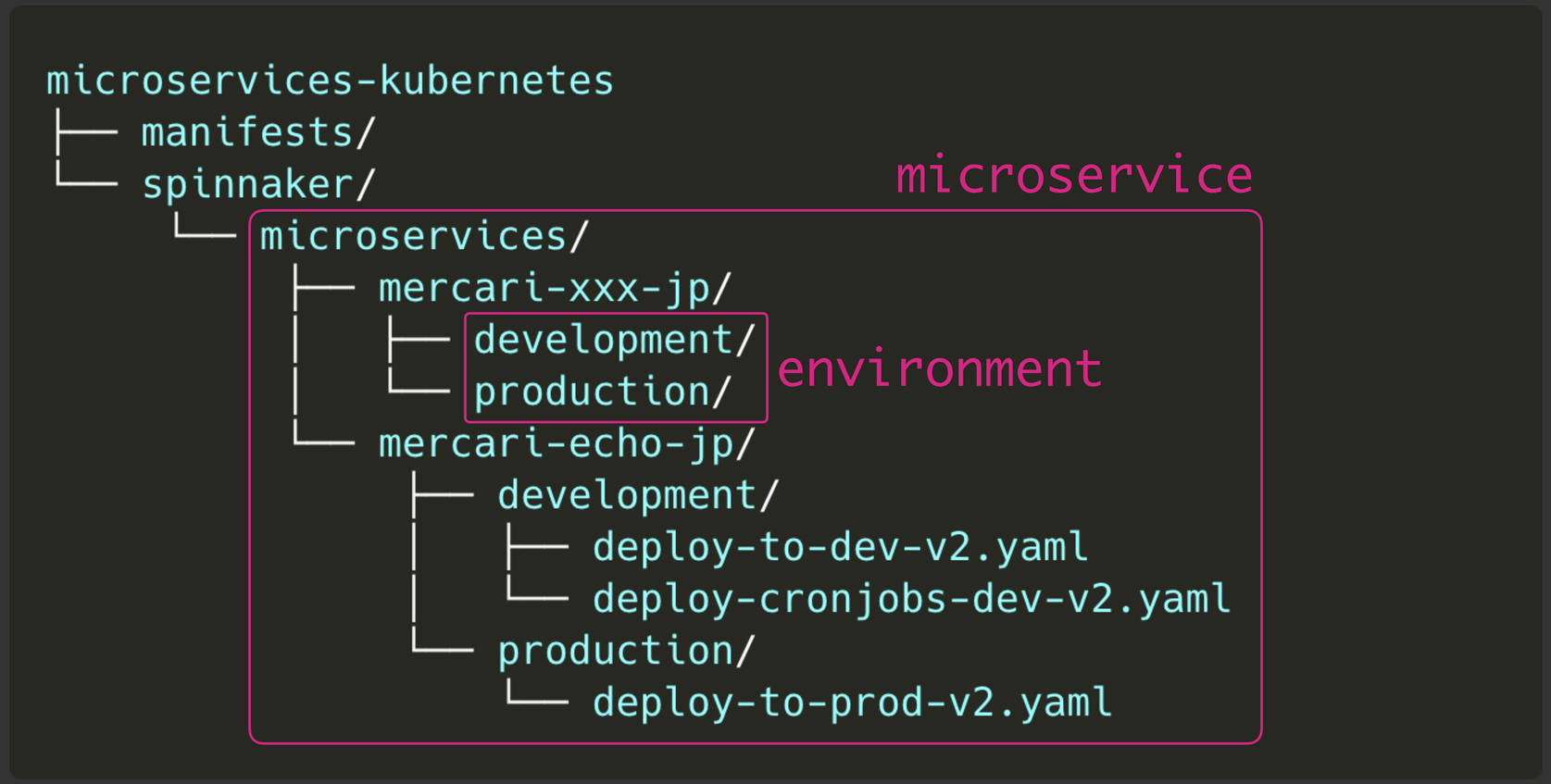

microservices-kubernetes
manifests/
microservices/
microservice
mercari-echo-jp/
development/
production/
environment
mercari-xxx-jp/
development/
kind
Deployment/
deployment.yaml
Certificate/
xxx-dev-mercari-com.yaml
Ingress/
xxx-dev-mercari-com.yaml
PodDisruptionBudget/
pdb.yaml
production/

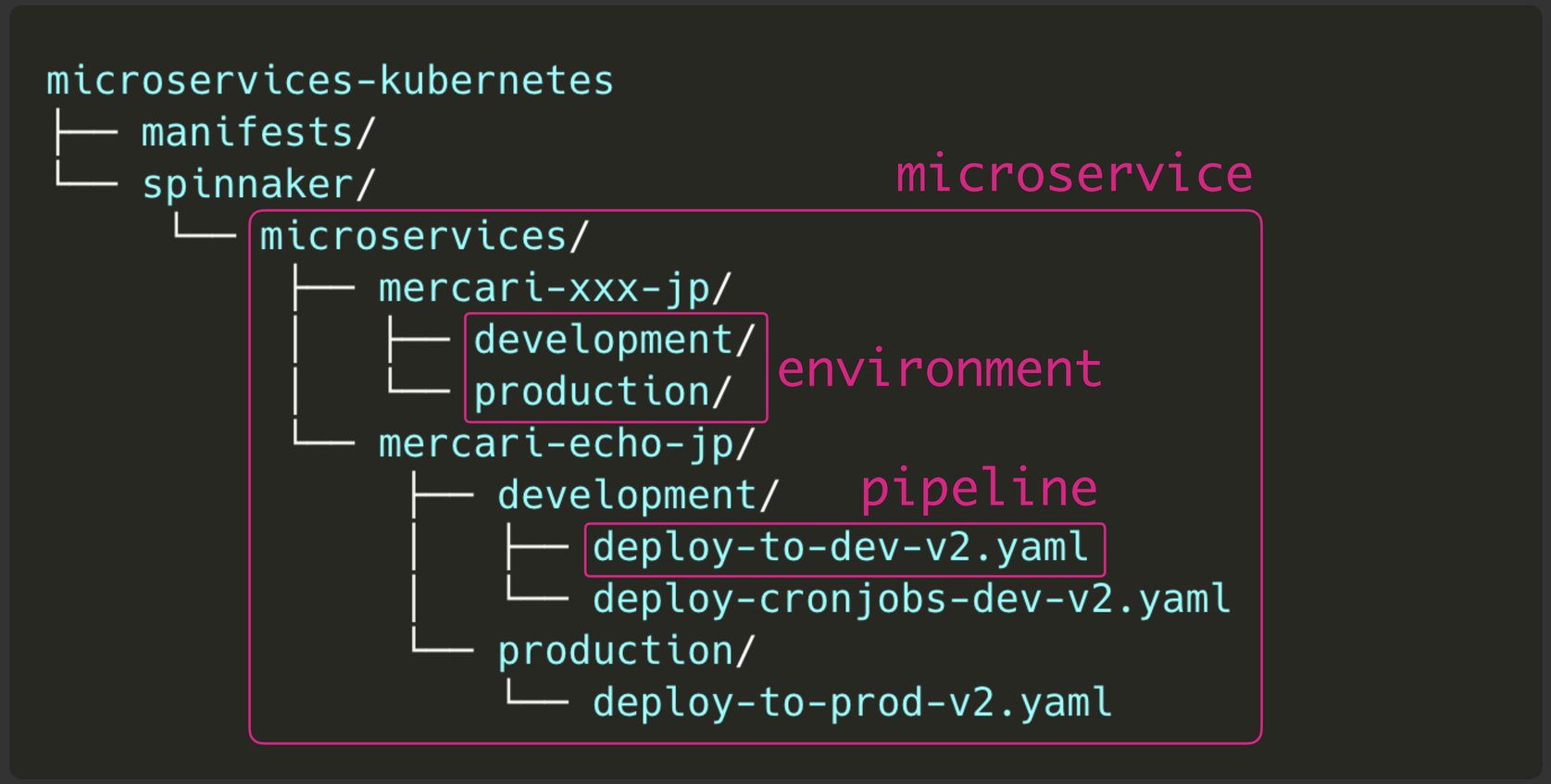

microservices-kubernetes
manifests/
microservices/
microservice
mercari-echo-jp/
development/
production/
environment
mercari-xxx-jp/
development/
kind
Deployment/
deployment.yaml
resource
Certificate/
xxx-dev-mercari-com.yaml
Ingress/
xxx-dev-mercari-com.yaml
PodDisruptionBudget/
pdb.yaml
production/

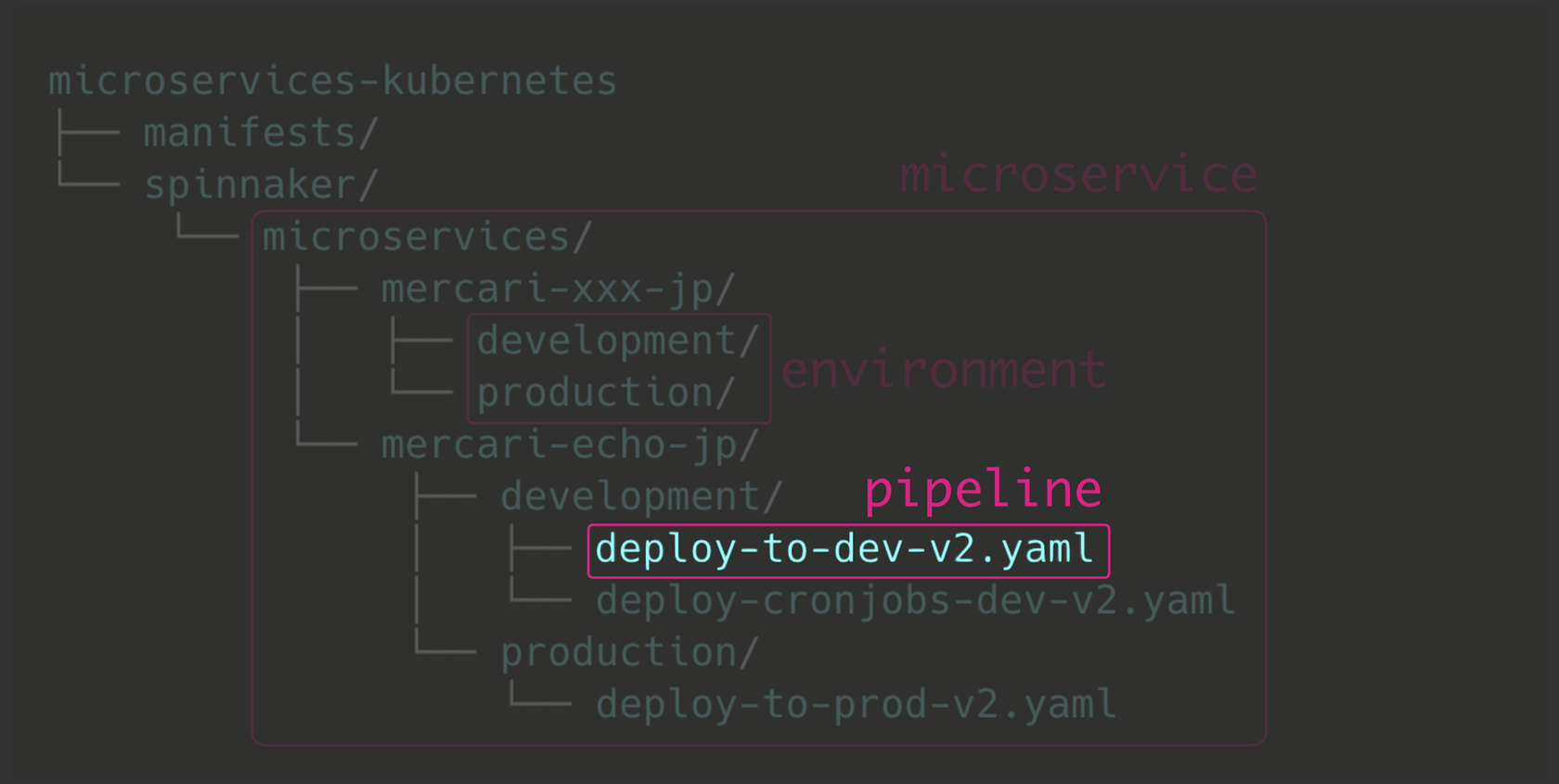
Spinnaker case

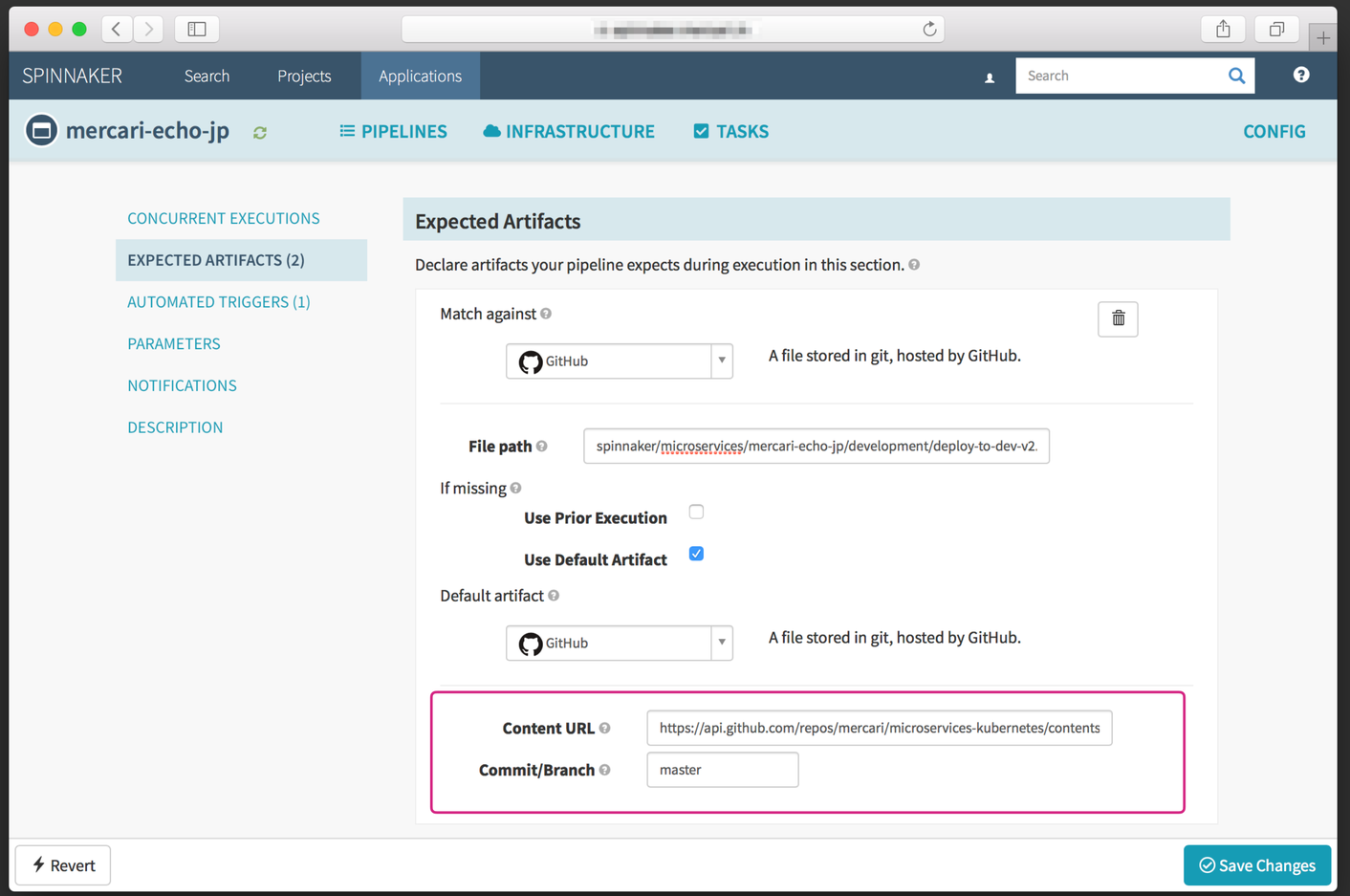

microservices-kubernetes
├── manifests/
└── spinnaker/
    └── microservices/
        ├── mercari-xxx-jp/
        │   ├── development/
        │   └── production/
        └── mercari-echo-jp/
            ├── development/
            │   ├── deploy-to-dev-v2.yaml
            │   └── deploy-cronjobs-dev-v2.yaml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


microservices-kubernetes
├── manifests/
└── spinnaker/
microservice
└── 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

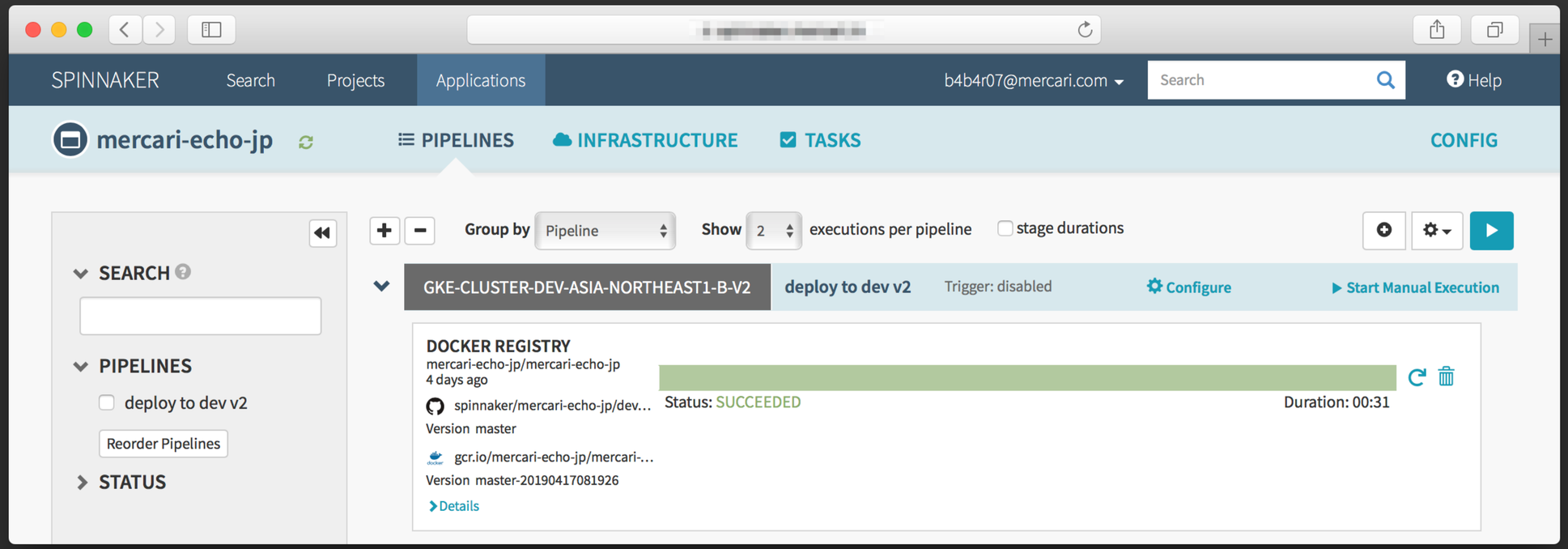

microservices-kubernetes
├── manifests/
└── spinnaker/
└── microservices/
microservice
├── mercari-xxx-jp/
│ ├── development/
│ └── production/
environment
└── mercari-echo-jp/
├── development/
│ ├── deploy-to-dev-v2.yaml
│ └── deploy-cronjobs-dev-v2.yaml
└── production/
└── deploy-to-prod-v2.yaml


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
environment
pipeline


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


SPINNAKER
Search
Projects
Applications
Search
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
GitHub
A file stored in git, hosted by GitHub.
File path
spinnaker/microservices/mercari-echo-jp/development/deploy-to-dev-v2.
If missing
Use Prior Execution
Use Default Artifact
Default artifact
GitHub
A file stored in git, hosted by GitHub.
Content URL
https://api.github.com/repos/mercari/microservices-kubernetes/contents
Commit/Branch
master
Revert
Save Changes

Pipeline


SPINNAKER
Search
Projects
Applications
b4b4r07@mercari.com
Search
Help
mercari-echo-jp
PIPELINES
INFRASTRUCTURE
TASKS
CONFIG
Group by
Pipeline
Show
2
executions per pipeline
stage durations
SEARCH
PIPELINES
deploy to dev v2
Reorder Pipelines
STATUS
GKE-CLUSTER-DEV-ASIA-NORTHEAST1-B-V2
deploy to dev v2
Trigger: disabled
Configure
Start Manual Execution
DOCKER REGISTRY
mercari-echo-jp/mercari-echo-jp
4 days ago
spinnaker/mercari-echo-jp/dev...
Version master
gcr.io/mercari-echo-jp/mercari-...
Version master-20190417081926
Details
Status: SUCCEEDED
Duration: 00:31

Pipeline

How do we apply the manifest changes?

Just run "kubectl"

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

kubectl

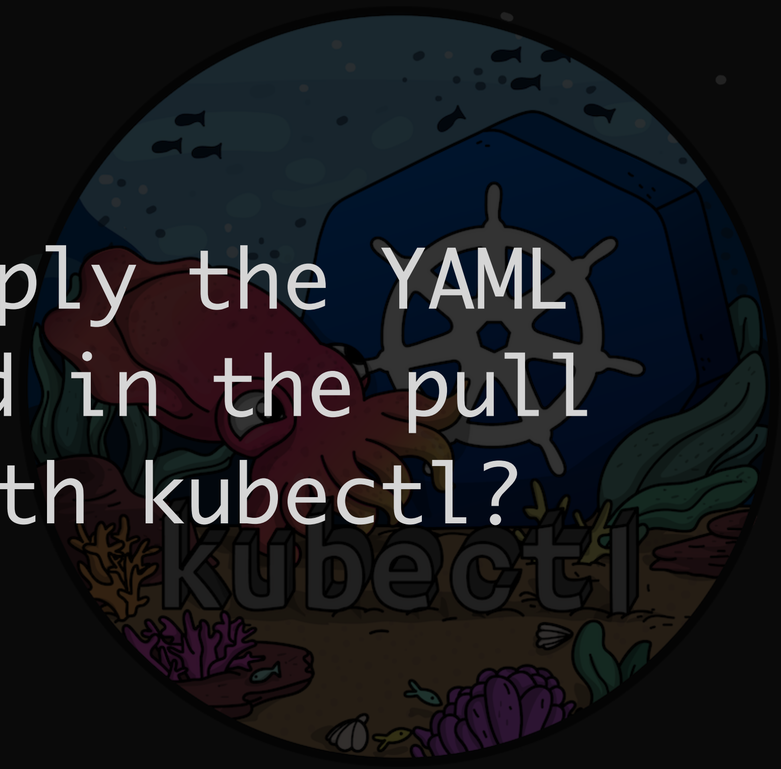
- Microservices developers not only develop but also operate the service by themselves
- So they are familiar with kubectl basically
- It means less learning cost than introducing other tools

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

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

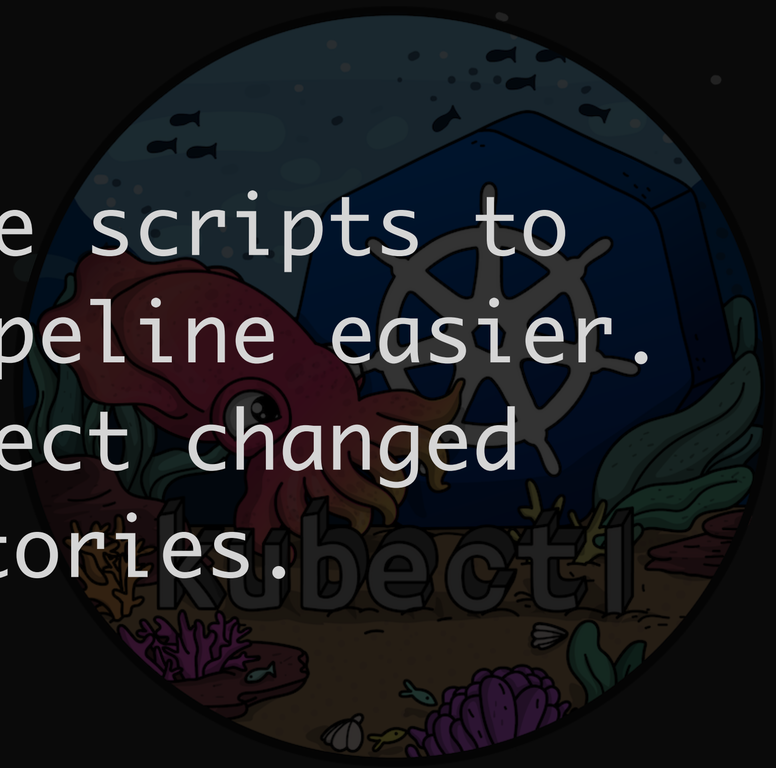


https://groups.google.com/forum/#!msg/kubernetes-sig-cli/M6t40JP6n0g/U6Snz-bsFQAJ

kubectl

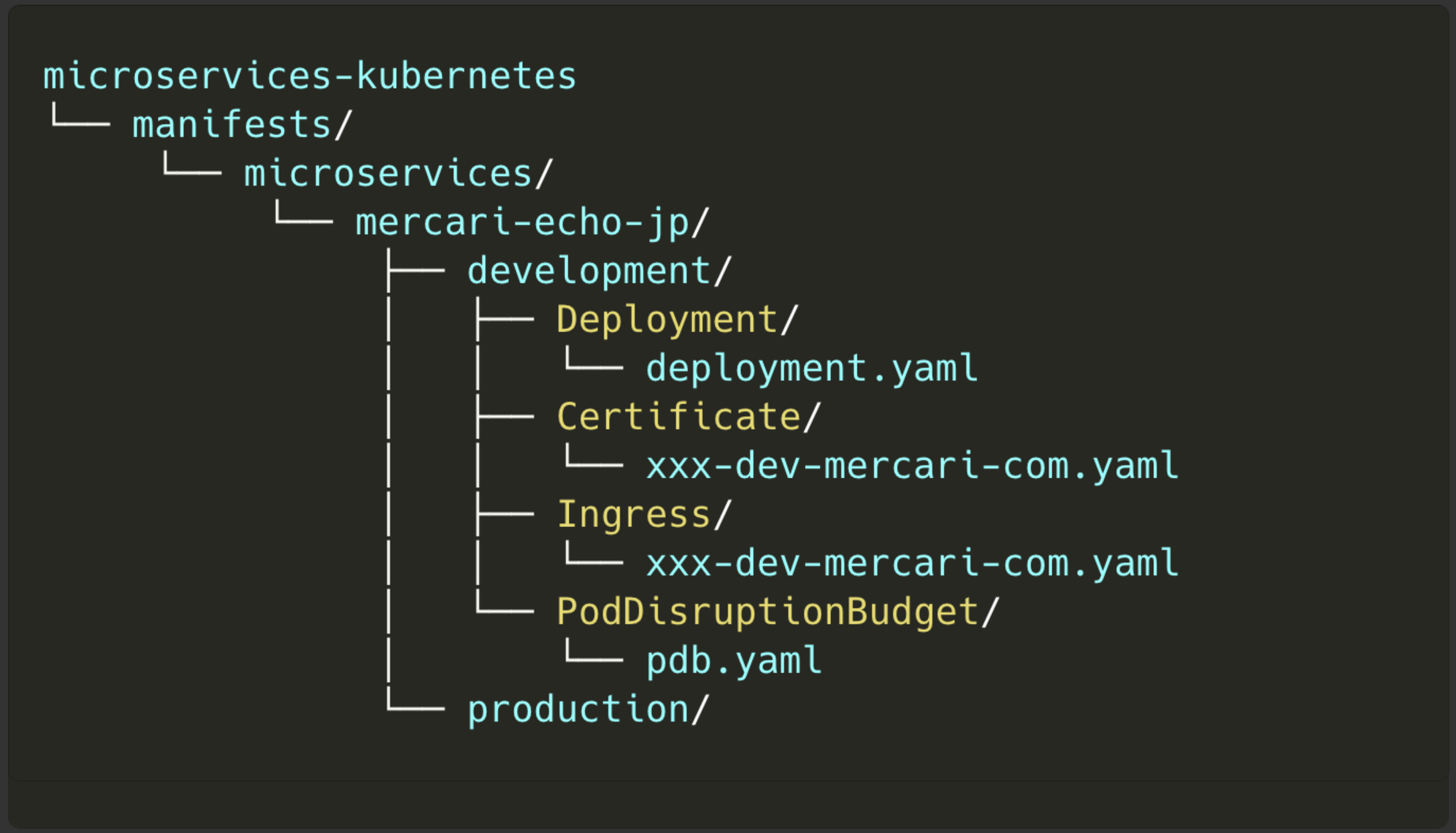
- Microservices developers not only develop but also operate the service by themselves
- So they are familiar with kubectl basically
- It means less learning cost than introducing other tools

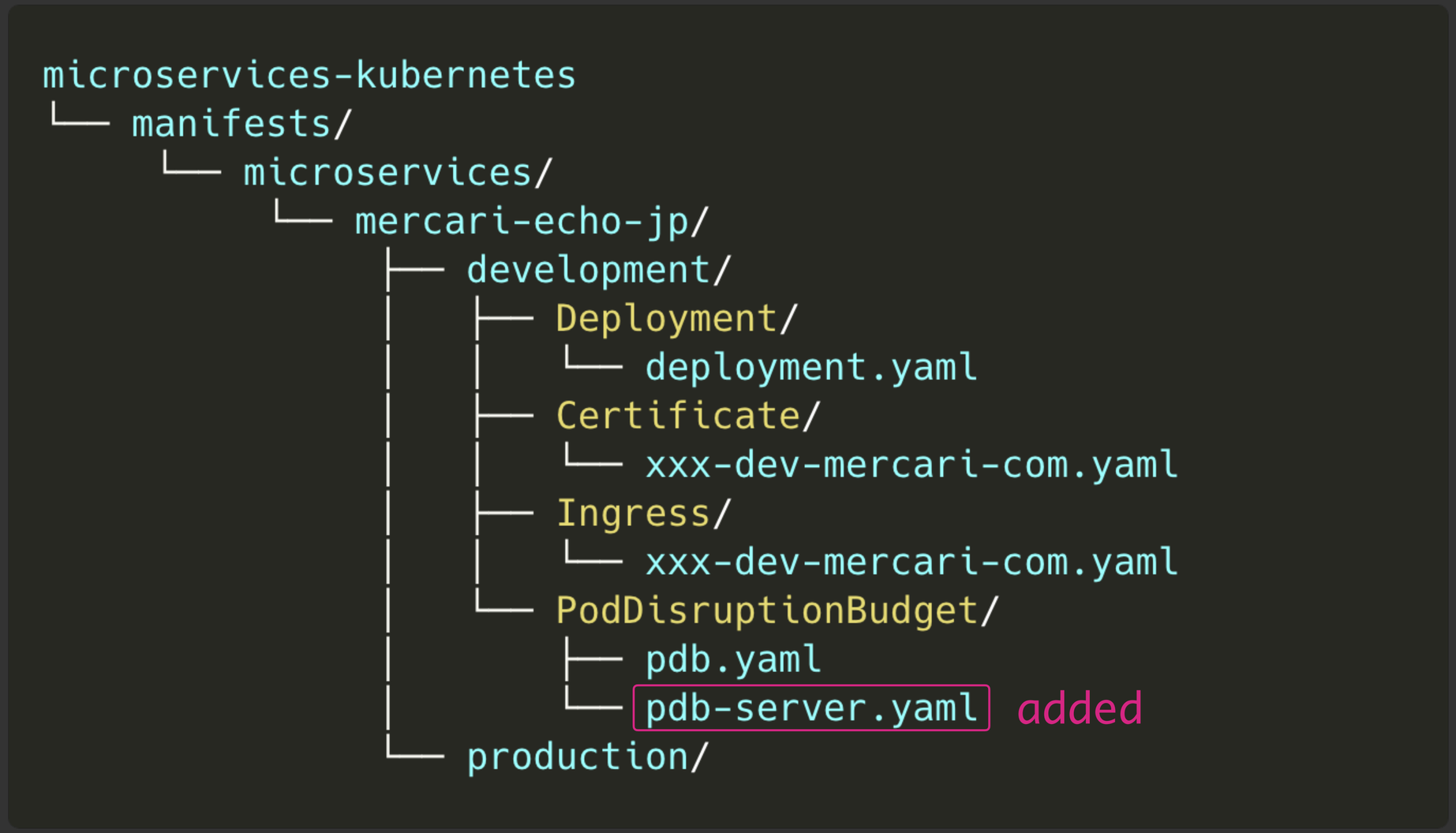
We have some scripts to make apply pipeline easier.
It can detect changed directories.



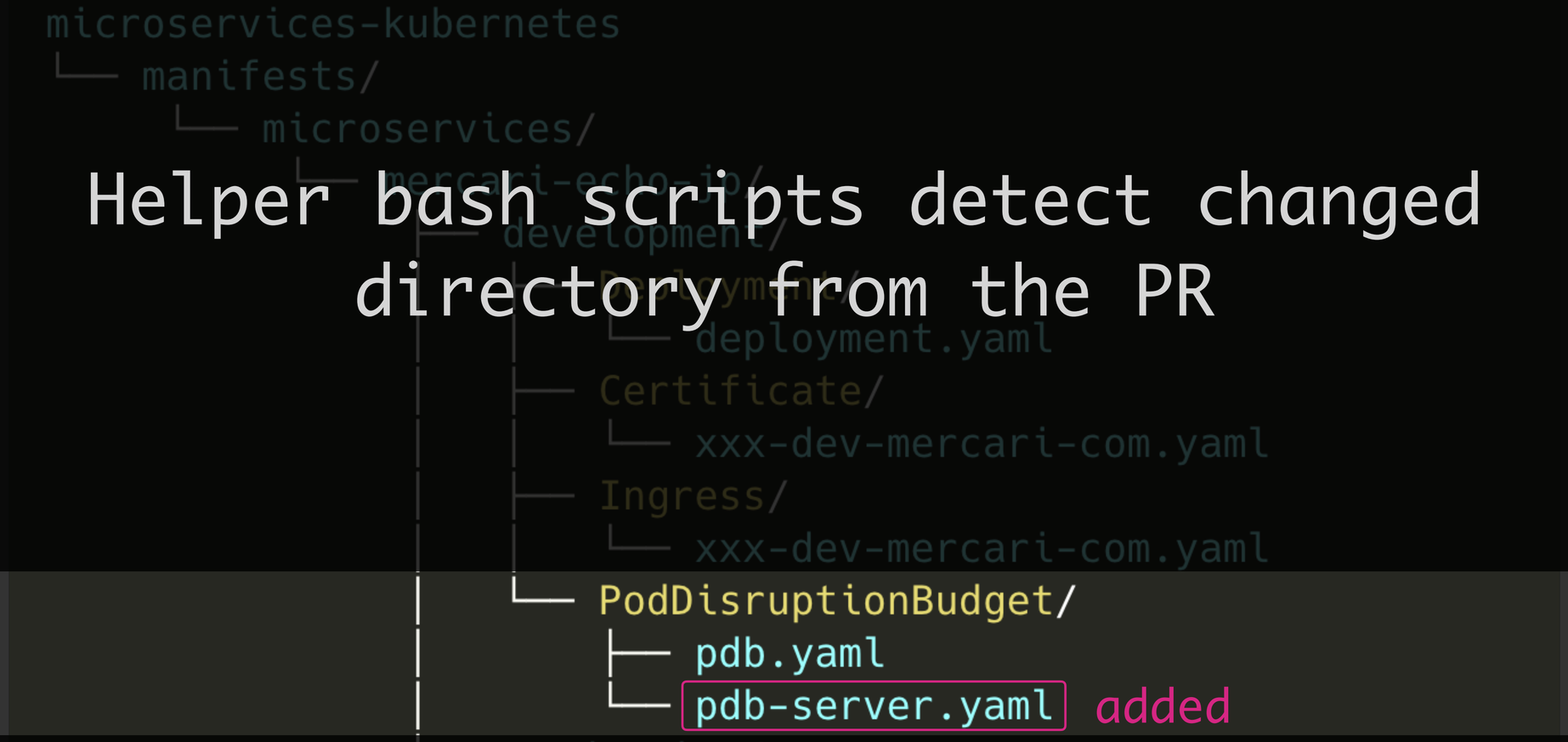
https://groups.google.com/forum/#!msg/kubernetes-sig-cli/M6t40JP6n0g/U6Snz-bsFQAJ

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/

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 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
}
```

How do we get the order of files to be applied?

- We have to pay attention the order to apply multiple manifests
- 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",
}
```

https://github.com/helm/helm/blob/v2.10.0/pkg/tiller/kind_sorter.go

```go
// 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",
}
```



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

kind_install_order.txt

https://github.com/helm/helm/blob/v2.10.0/pkg/tiller/kind_sorter.go

```
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
```

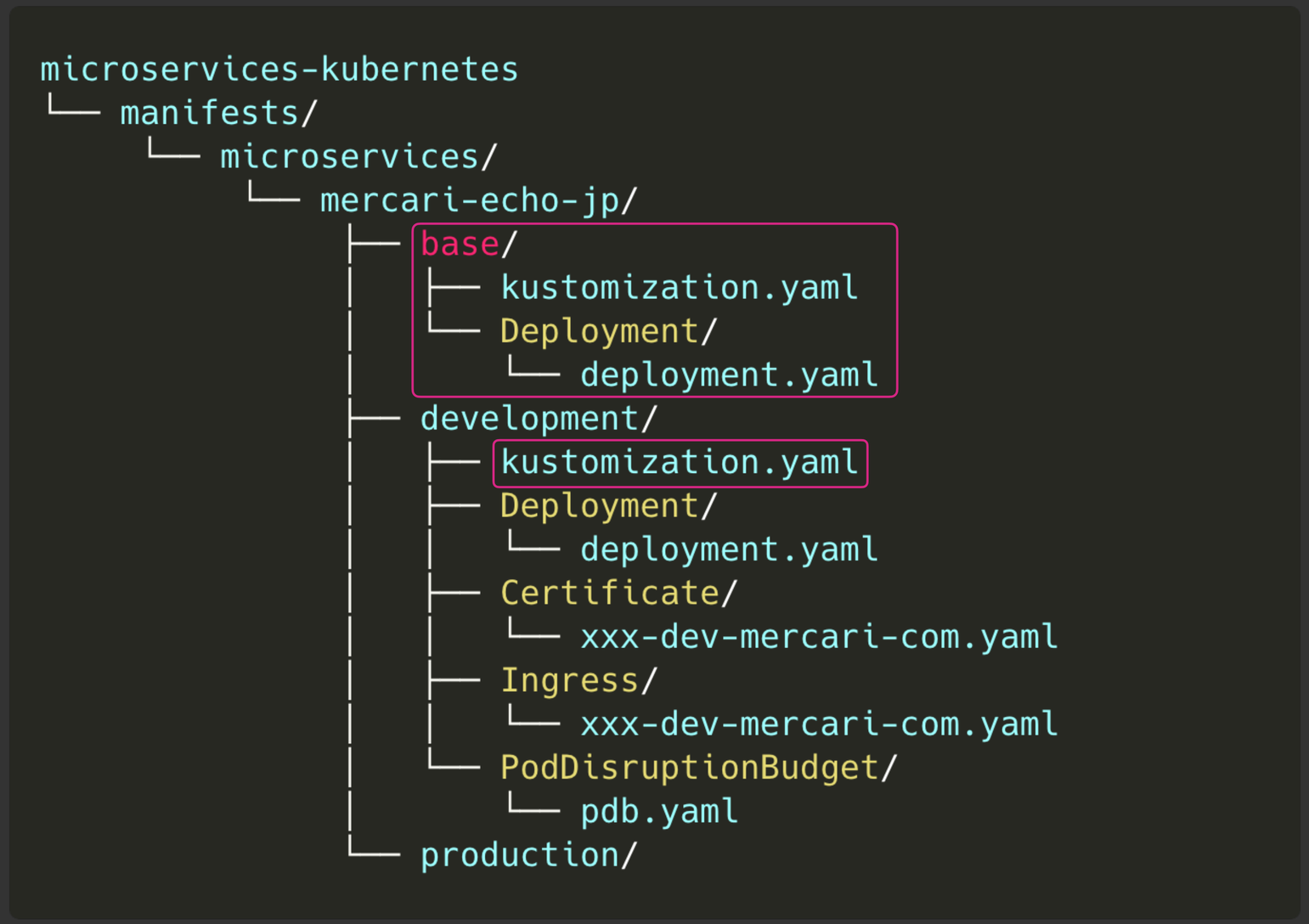
```
sort_kinds_by_install_order() {
  kinds=( $(cat "kind_install_order.txt") )
  args=( "${@}" )
  for kind in "${kinds[@]}"
  do
    for arg in "${args[@]}"
    do
      if [[ $(get_kind "${arg}") == ${kind} ]]; then
        echo "${arg}"
      fi
    done
  done
}
```

How about using kustomize?

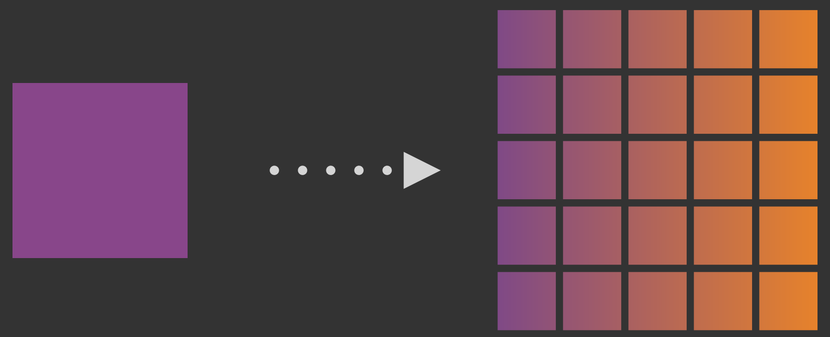
```
.../development/Deployment
.../development/ConfigMap
.../development/PodDisruptionBudget
```

```
ConfigMap
Deployment
PodDisruptionBudget
```

kind_install_order.txt



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



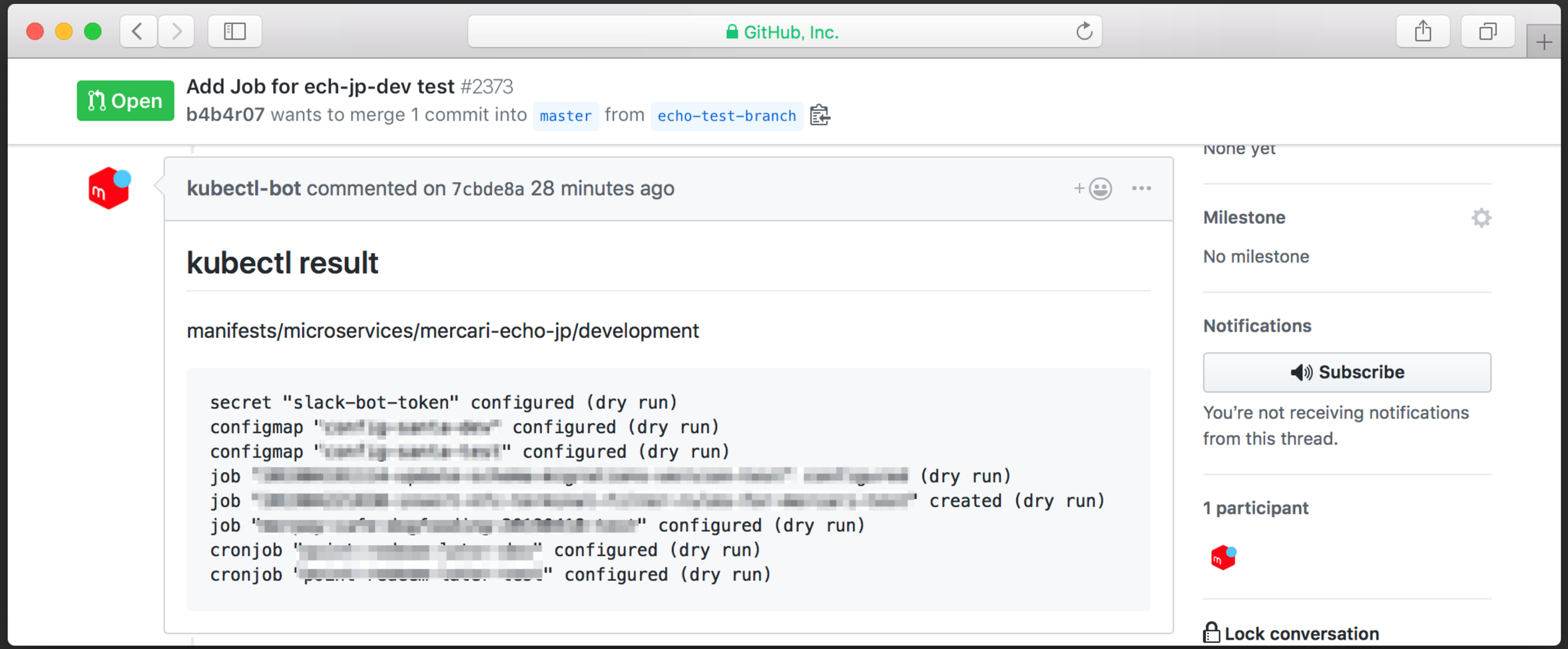
- Our microservices is on the way
- So developers are in the middle of being microservices developers
- They have to learn a lot of things:
  - Kubernetes, Kubernetes YAML itself, Spinnaker, etc...
  - So introducing kustomize feature to our pipeline was not now. In the future.

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.

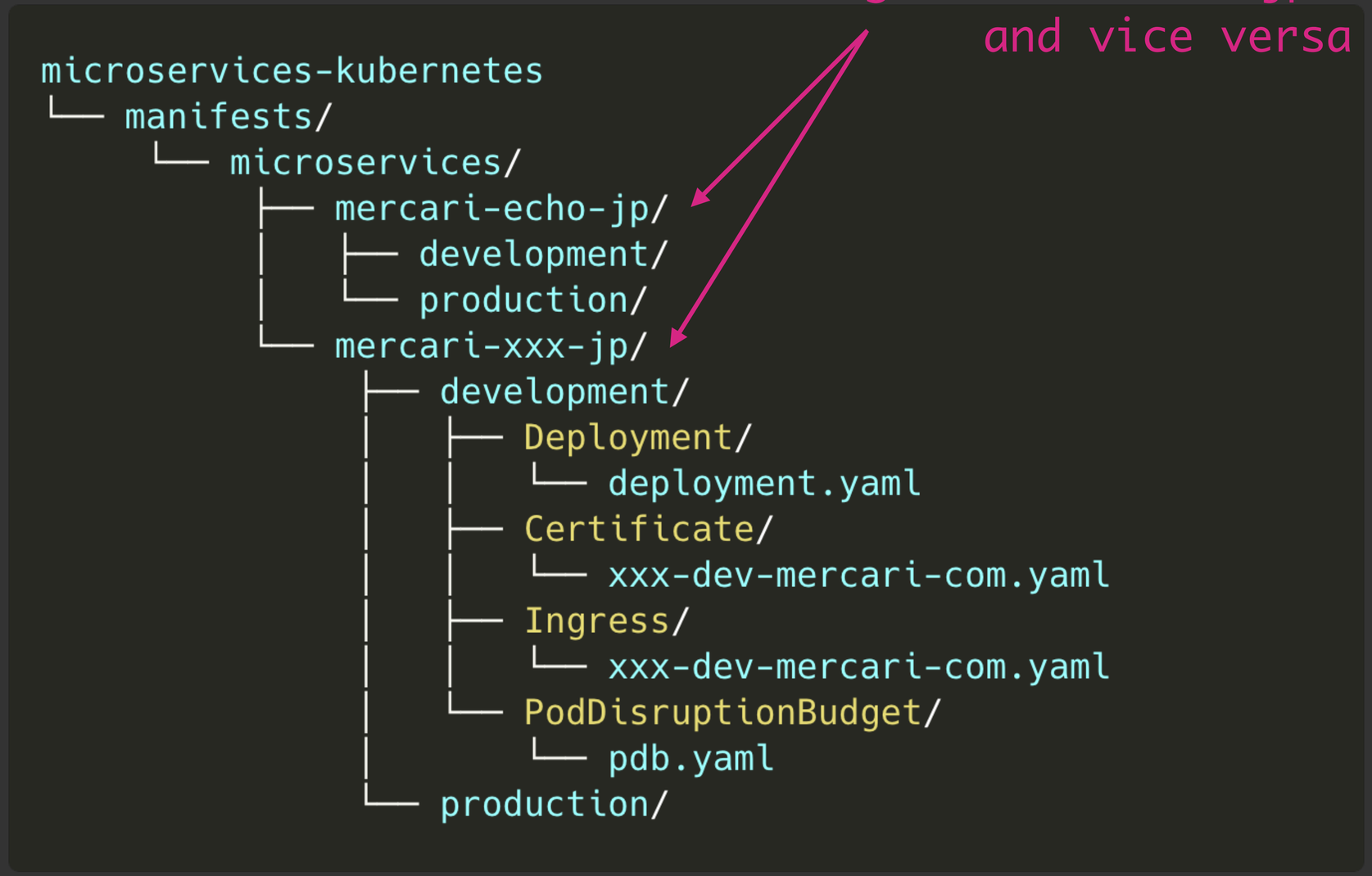
```
kubectl_apply() {
  # ...
  for manifest in $(sort_kinds_by_install_order "${manifests[@]}"); do
    case ${kind} in
      PodDisruptionBudget) # Need to be recreated if it already exists
        if kubectl get -n "${namespace}" "${kind}" "${resource}"; then
          kubectl delete -n "${namespace}" "${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.

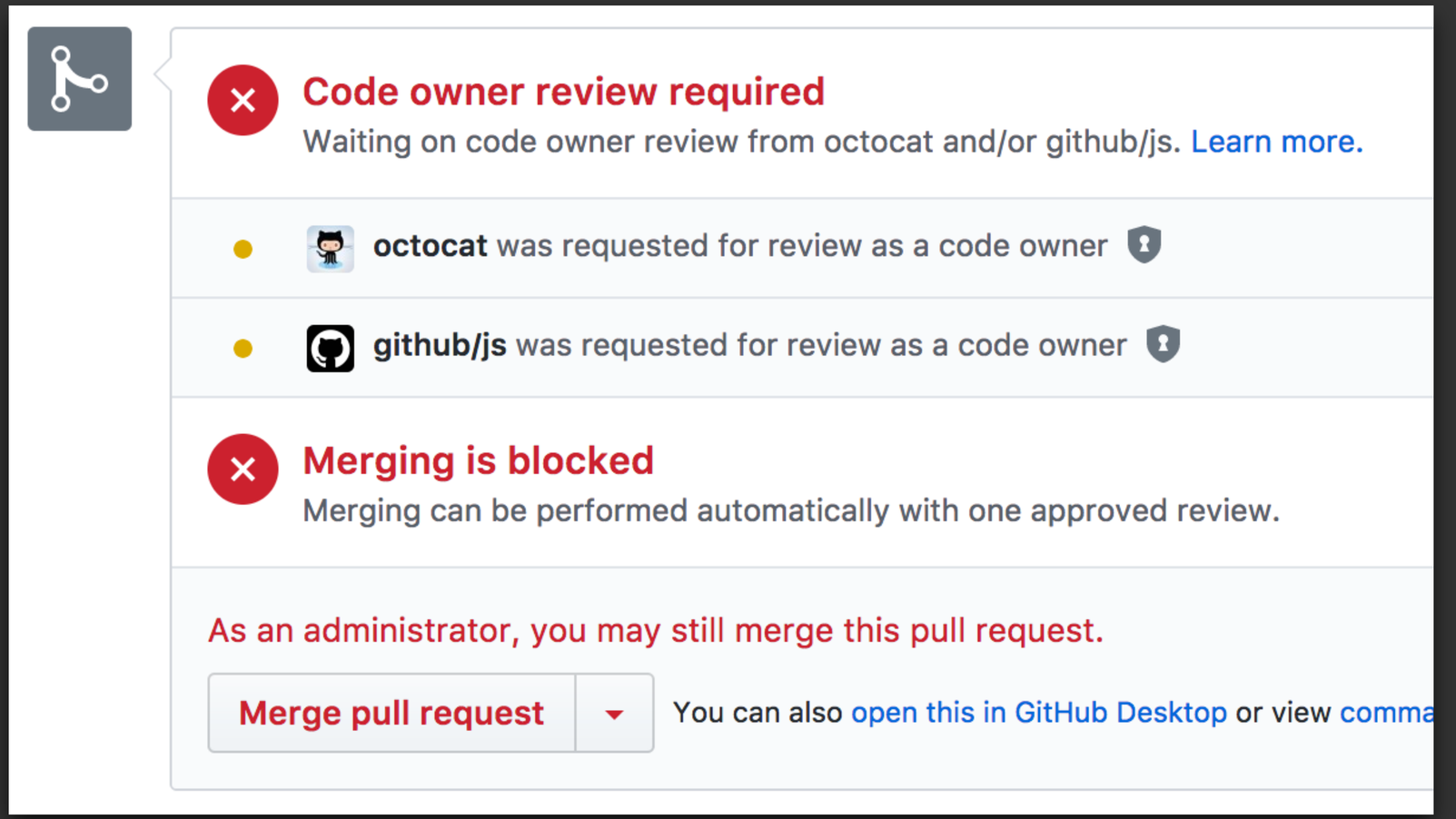

GitHub, Inc.
Open
Add Job for ech-jp-dev test #2373
b4b4r07 wants to merge 1 commit into master from echo-test-branch
kubectl-bot commented on 7cbde8a 28 minutes ago
kubectl result
manifests/microservices/mercari-echo-jp/development
secret "slack-bot-token" configured (dry run)
configmap "[redacted]" configured (dry run)
configmap "[redacted]" configured (dry run)
job "[redacted]" (dry run)
job "[redacted]" created (dry run)
job "[redacted]" configured (dry run)
cronjob "[redacted]" configured (dry run)
cronjob "[redacted]" configured (dry run)
None yet
Milestone
No milestone
Notifications
Subscribe
You're not receiving notifications from this thread.
1 participant
Lock conversation

The delegation of directory authority

mercari-echo-jp team should not be able
to change mercari-xxx-jp's team code,
and vice versa



GitHub CODEOWNERS feature



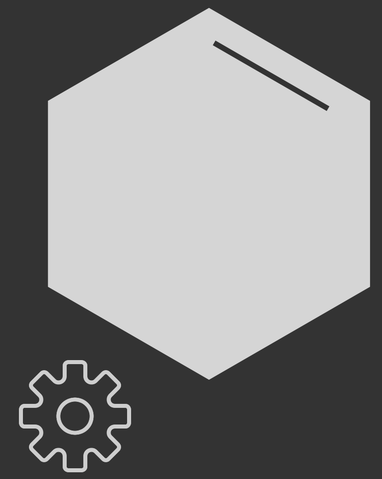
https://blog.github.com/2017-07-06-introducing-code-owners/

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
```

https://help.github.com/articles/about-codeowners/

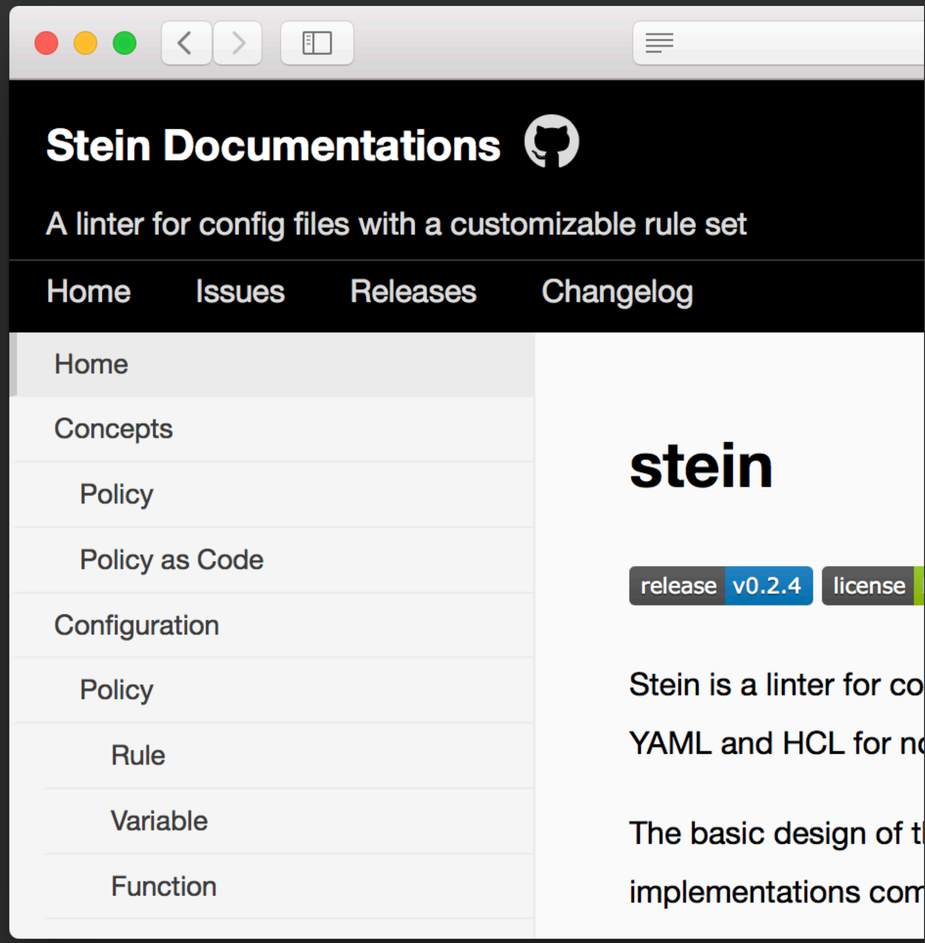
Repository Ecosystem



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

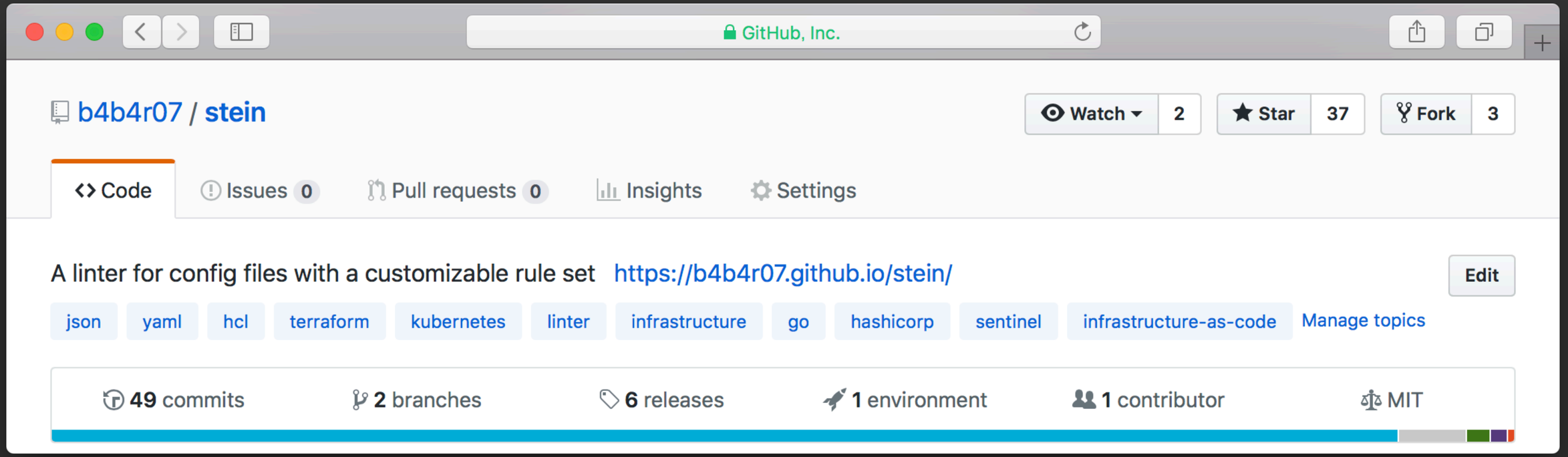


[Stein Documentations]

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...?

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



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"

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

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

rule definition

the condition this rule fails or not

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") != ""}",
  ]

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

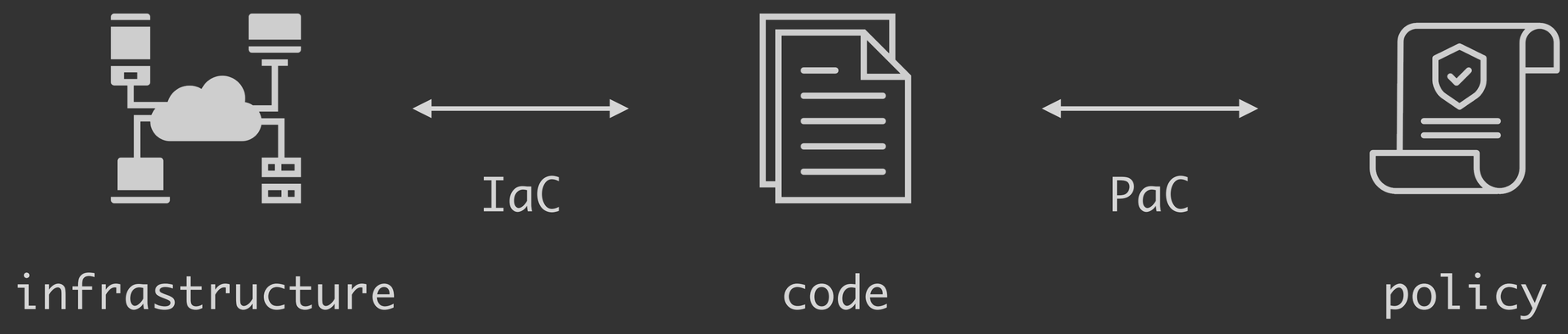
stein supports many built-in functions like Terraform

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)
```

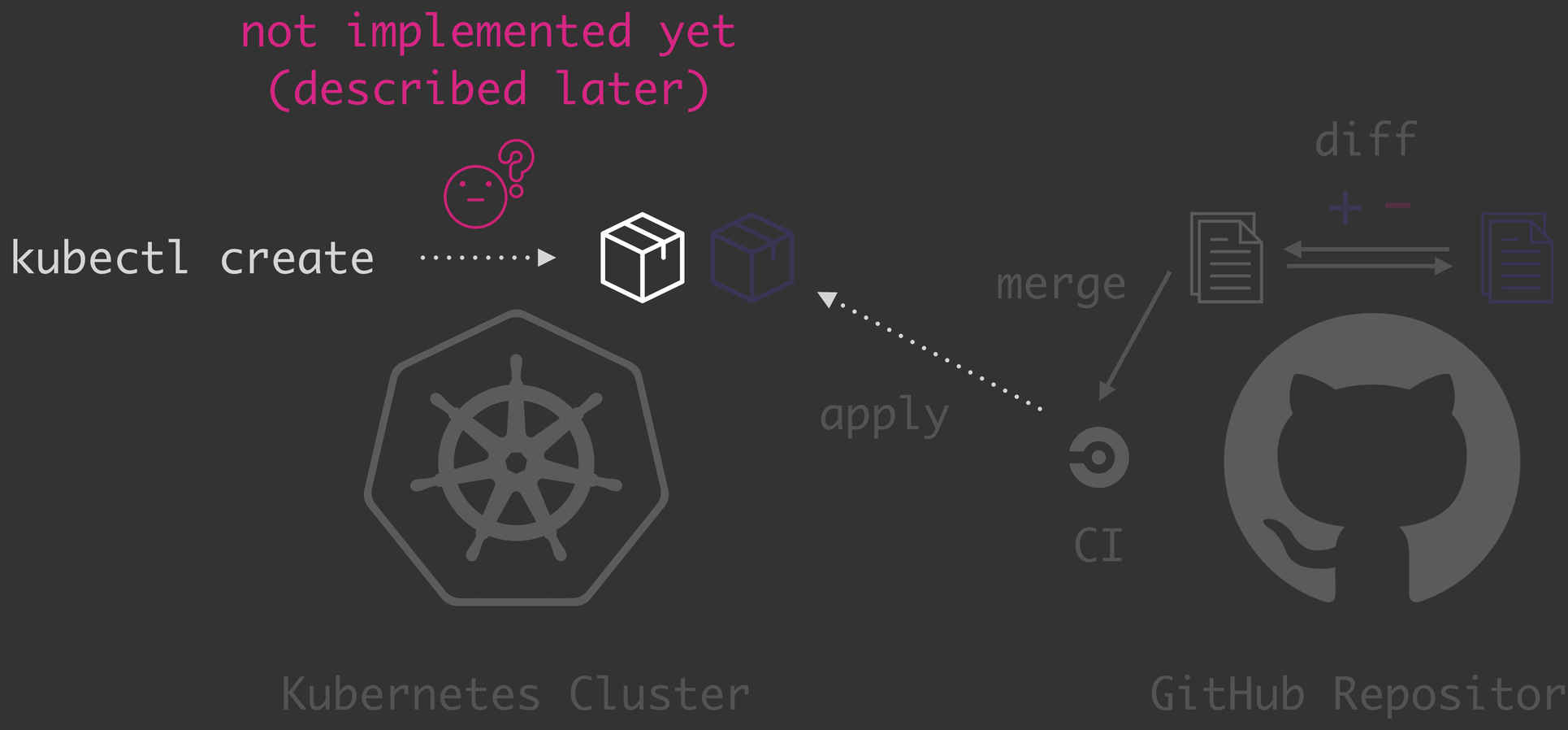
- Stein checks the policy files and applies them to your config files. If there are violation rules, Stein returns exit code 1.
- Stein can work as a linter in CI step etc well.

Policy as Code

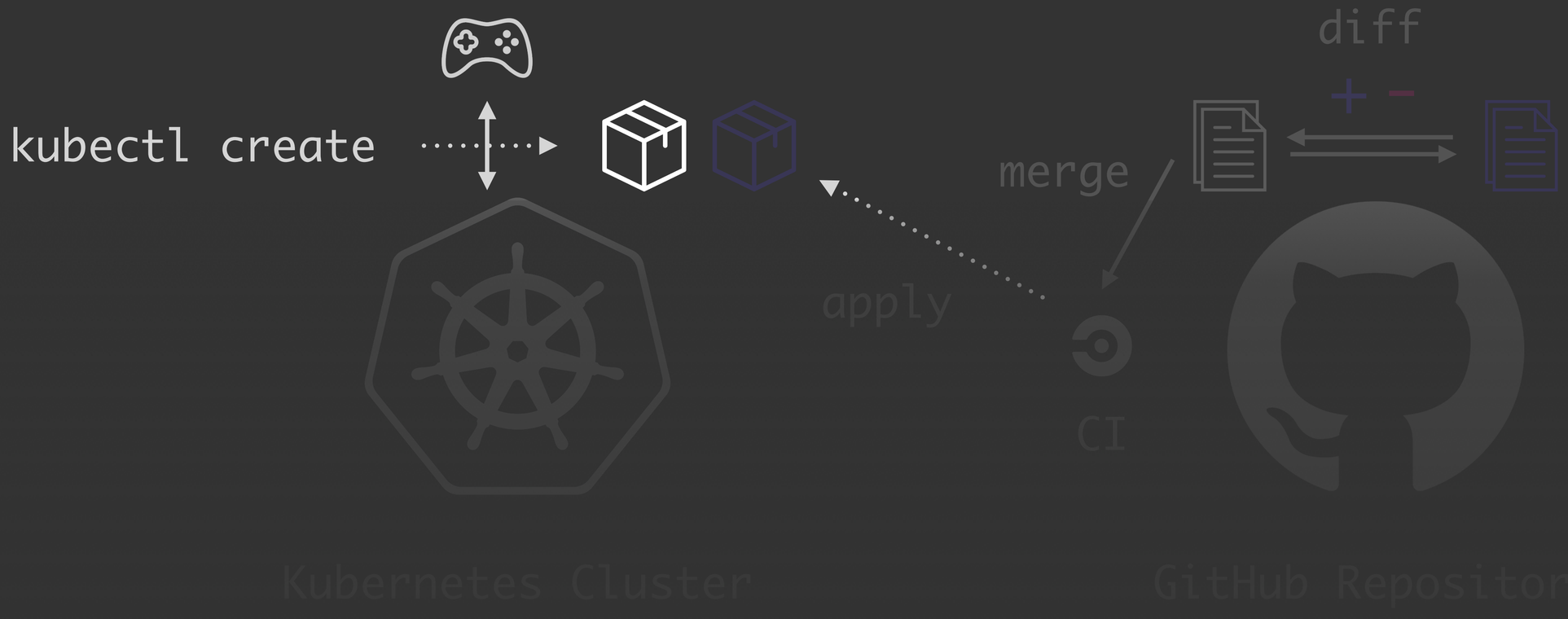


- Stein concepts and design comes from HashiCorp Sentinel one.
- "Policy as Code" (PaC) is provided by HashiCorp and Sentinel.
- PaC means the way to describe "ideal configuration files" and force it upon real configuration files.

Why Policy as Code? - HashiCorp Blog
Policy as Code - Sentinel by HashiCorp


not implemented yet
(described later)
kubectl create
diff
+ −
merge
apply
CI
Kubernetes Cluster
GitHub Repository

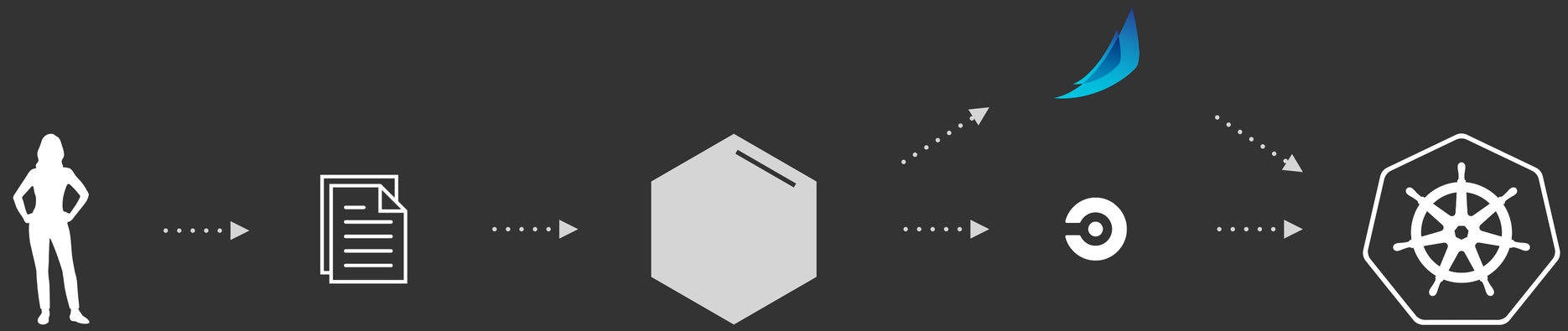
Stein: Adminmission Controller



- 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

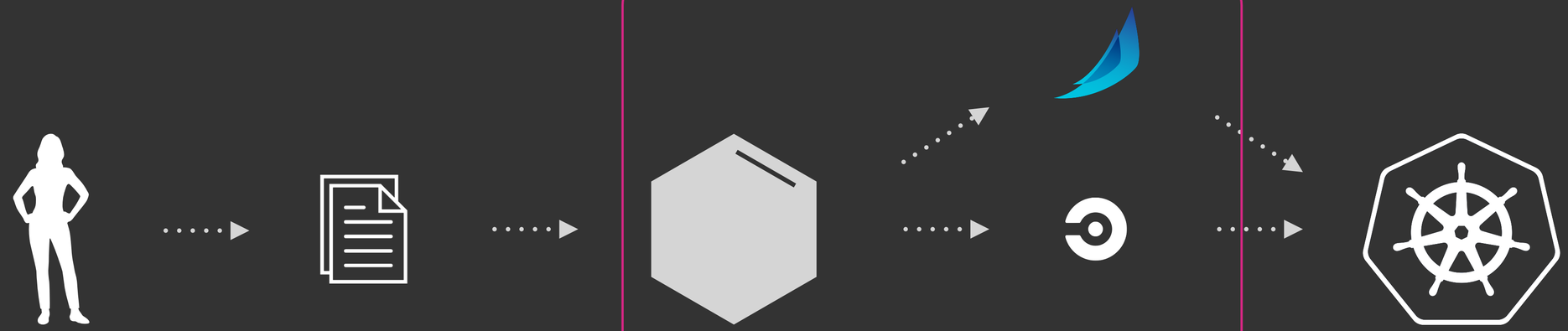
3. Run apply (dry-run)



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