Introduction to kustomize

Kubernetes Meetup Tokyo #12, Jul 11, 2018

@spesnova

SRE at Mercari, Inc. / Kubernetes Tokyo Community Organizer

Agenda

1. Basics
2. Features
3. Keys

Tested with kustomize **v1.0.3**

Basics

What is **kustomize**?

kustomize is a command line tool

kustomize is a CLI for managing k8s style object with **declarative** way

Let’s learn a basic usage!

Basics / Hello World

Example Requirements

- 3 environments (dev, stg, prod)
- 1 deployment resource
- different replicas by environments

File Structure

```
hello-world/
├── base
│   ├── deployment.yaml
│   └── kustomization.yaml
└── overlays
    ├── production
    │   ├── replica_count.yaml
    │   └── kustomization.yaml
    └── staging
        ├── replica_count.yaml
        └── kustomization.yaml
```

File Structure

```
hello-world/
├──── base
│     ├──── deployment.yaml
│     └──── kustomization.yaml
└──── overlays
      ├──── production
      │     ├──── replica_count.yaml
      │     └──── kustomization.yaml
      └──── staging
            ├──── replica_count.yaml
            └──── kustomization.yaml
```

File Structure

```
hello-world/
├──── base
│     ├──── deployment.yaml
│     └──── kustomization.yaml
└──── overlays
      ├──── production
      │     ├──── replica_count.yaml
      │     └──── kustomization.yaml
      └──── staging
            ├──── replica_count.yaml
            └──── kustomization.yaml
```

File Structure

```
hello-world/
├── base
│   ├── deployment.yaml
│   └── kustomization.yaml
└── overlays
    ├── production
    │   ├── replica_count.yaml
    │   └── kustomization.yaml
    └── staging
        ├── replica_count.yaml
        └── kustomization.yaml
```

Base

```
hello-world/
├──── base
│      ├──── deployment.yaml
│      └──── kustomization.yaml
└──── overlays
       ├──── production
       │      ├──── replica_count.yaml
       │      └──── kustomization.yaml
       └──── staging
              ├──── replica_count.yaml
              └──── kustomization.yaml
```

Base

```yaml
# hello-world/base/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world
  template: ..
```

Base

```yaml
# hello-world/base/kustomization.yaml
resources:
- deployment.yaml
```

Staging

```
hello-world/
├──── base
│     ├──── deployment.yaml
│     └──── kustomization.yaml
└──── overlays
      ├──── production
      │     ├──── replica_count.yaml
      │     └──── kustomization.yaml
      └──── staging
            ├──── replica_count.yaml
            └──── kustomization.yaml
```

Staging

```yaml
# hello-world/staging/replica_count.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 3
```

Staging

```
# hello-world/staging/kustomization.yaml
bases:
- ../../base
patches:
-  replica_count.yaml
```

$ kustomize build

```
$ kustomize build -h
Print current configuration per contents of kustomization.yaml

Usage:
  kustomize build [path] [flags]
```

Print **staging** configuration

```
$ kustomize build hello-world/overlays/staging/
```

Print **staging** configuration

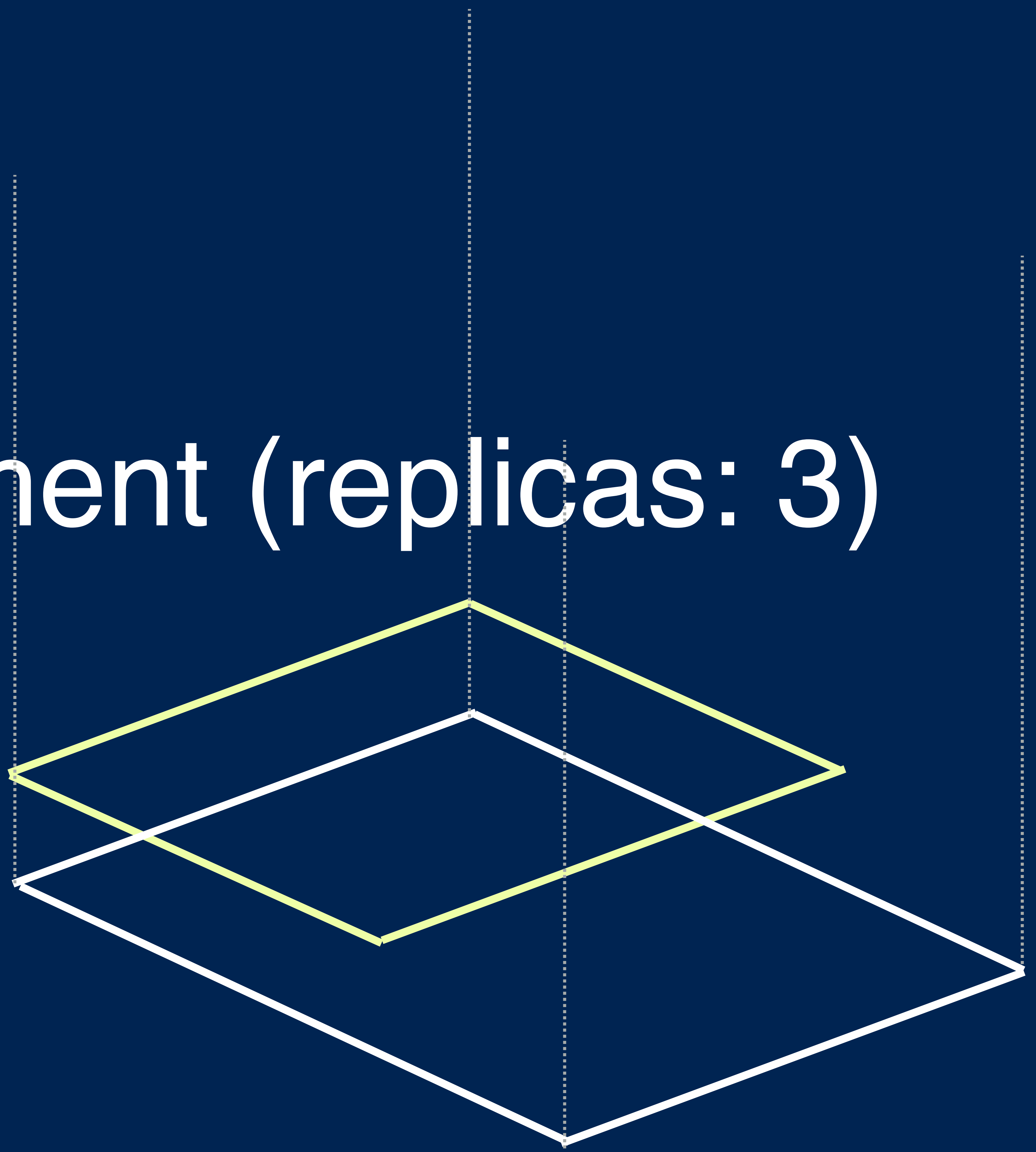
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-world
  template: ..
```

template-free customization

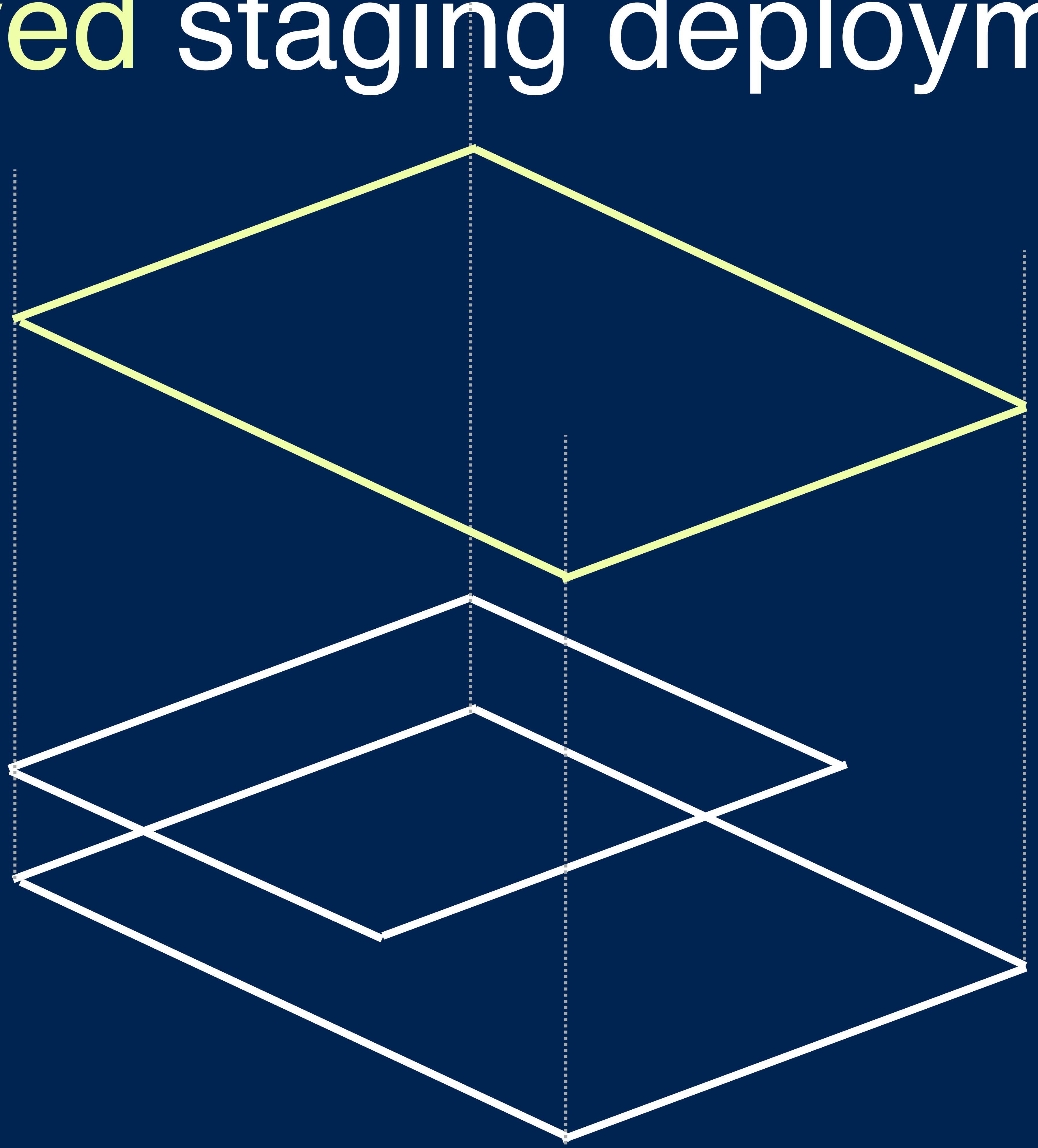
overlay customization

base deployment (replicas 1)

staging deployment (replicas: 3)



overlayed staging deployment (replicas 3)



Production

```
hello-world/
├──── base
│     ├──── deployment.yaml
│     └──── kustomization.yaml
└──── overlays
      ├──── production
      │     ├──── replica_count.yaml
      │     └──── kustomization.yaml
      └──── staging
            ├──── replica_count.yaml
            └──── kustomization.yaml
```

Production

```yaml
# hello-world/production/replica_count.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 7
```

Production

```
# hello-world/production/kustomization.yaml
bases:
- ../../base
patches:
-  replica_count.yaml
```

Print **production** configuration

```
$ kustomize build hello-world/overlays/production/
```

Print **production** configuration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 7
  selector:
    matchLabels:
      app: hello-world
  template: ..
```

Apply printed configuration

```
$ kustomize build [PATH] | kubectl apply -f -
```

Basics / Motivation

Declarative specification

is the recommended way

However…

It’s difficult to use only current kubectl
to follow declarative way…

Then…

Another Tools are required

- Helm
- Ksonnet
- Kapitan
- Forge
- Ktmpl
- etc…

Drawbacks of those tools

1. I have to learn new tools…
2. I have to learn new DSL… (complicated!)
3. I have to teach new concepts to teams…

Features

Features / Name Prefix

Name Prefix

```
# overlays/production/kustomization.yaml
namePrefix: prod-
bases:
- ../../base
patches:
-  replica_count.yaml
```

Name Prefix

```
$ kustomize build hello-world/overlays/production/
```

Name Prefix

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prod-hello-world
spec:
  replicas: 7
  selector:
    matchLabels:
      app: hello-world
  template: ..
```

Features / Common Labels

Common Labels

```yaml
# base/kustomization.yaml
commonLabels:
  owner: spesnova
resources:
- deployment.yaml
```

Common Labels

```
$ kustomize build hello-world/overlays/production/
```

Common Labels

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
  labels:
    owner: spesnova
spec:
  replicas: 7
  selector:
    matchLabels:
      app: hello-world
  template: ..
```

Features / Common Annotattion

Common Annotations

```
# base/kustomization.yaml
commonAnnotations:
  description: This is Hello World App
resources:
- deployment.yaml
```

Common Annotations

```
$ kustomize build hello-world/overlays/production/
```

Common Annotations

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
  annotations:
    description: This is Hello World App
spec:
  replicas: 7
  selector:
    matchLabels:
      app: hello-world
  template: …
```

Features / ConfigMap Generator

ConfigMap Generator

```
# base/kustomization.yaml
resources:
- deployment.yaml
configMapGenerator:
- name: hello-config
  files:
  - hello.config
```

ConfigMap Generator

```
# hello.config
name=hello-world
region=tokyo
```

ConfigMap Generator

```
$ kustomize build hello-world/overlays/production/
```

ConfigMap Generator

```
apiVersion: v1
data:
  hello.config: |
    name=hello-world
    region=tokyo
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: hello-config-4g5t58m8t5
---
apiVersion: apps/v1
kind: Deployment
...
```

Hash suffix

```
apiVersion: v1
data:
  hello.config: |
    name=hello-world
    region=tokyo
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: hello-config-4g5t58m8t5
---
apiVersion: apps/v1
kind: Deployment
...
```

Hash suffix

```
# hello.config
name=hello-world
region=london
```

Hash suffix

```
apiVersion: v1
data:
  hello.config: |
    name=hello-world
    region=tokyo
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: hello-config-bdmmkghm2m
---
apiVersion: apps/v1
kind: Deployment
...
```

Features / Secrets Generator (skip)

Features / Diff

$ kustomize diff

```
$ kustomize diff hello-world/overlays/production/
```

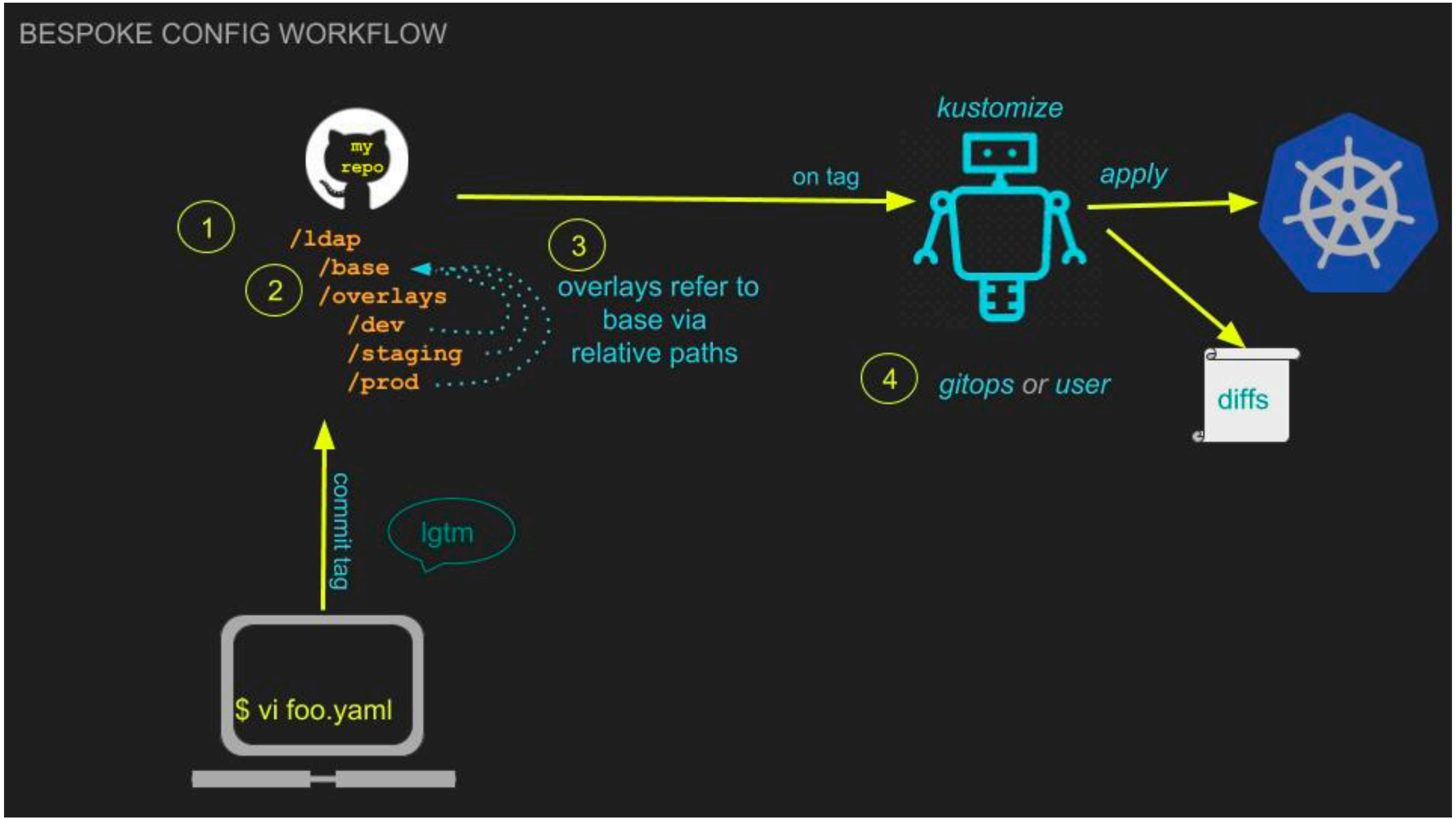
$ kustomize diff

```
@@ -3,7 +3,7 @@
 metadata:
   name: hello-world
 spec:
-  replicas: 1
+  replicas: 7
   selector:
     matchLabels:
       app: hello-world
```

Features / Substitute (skip)

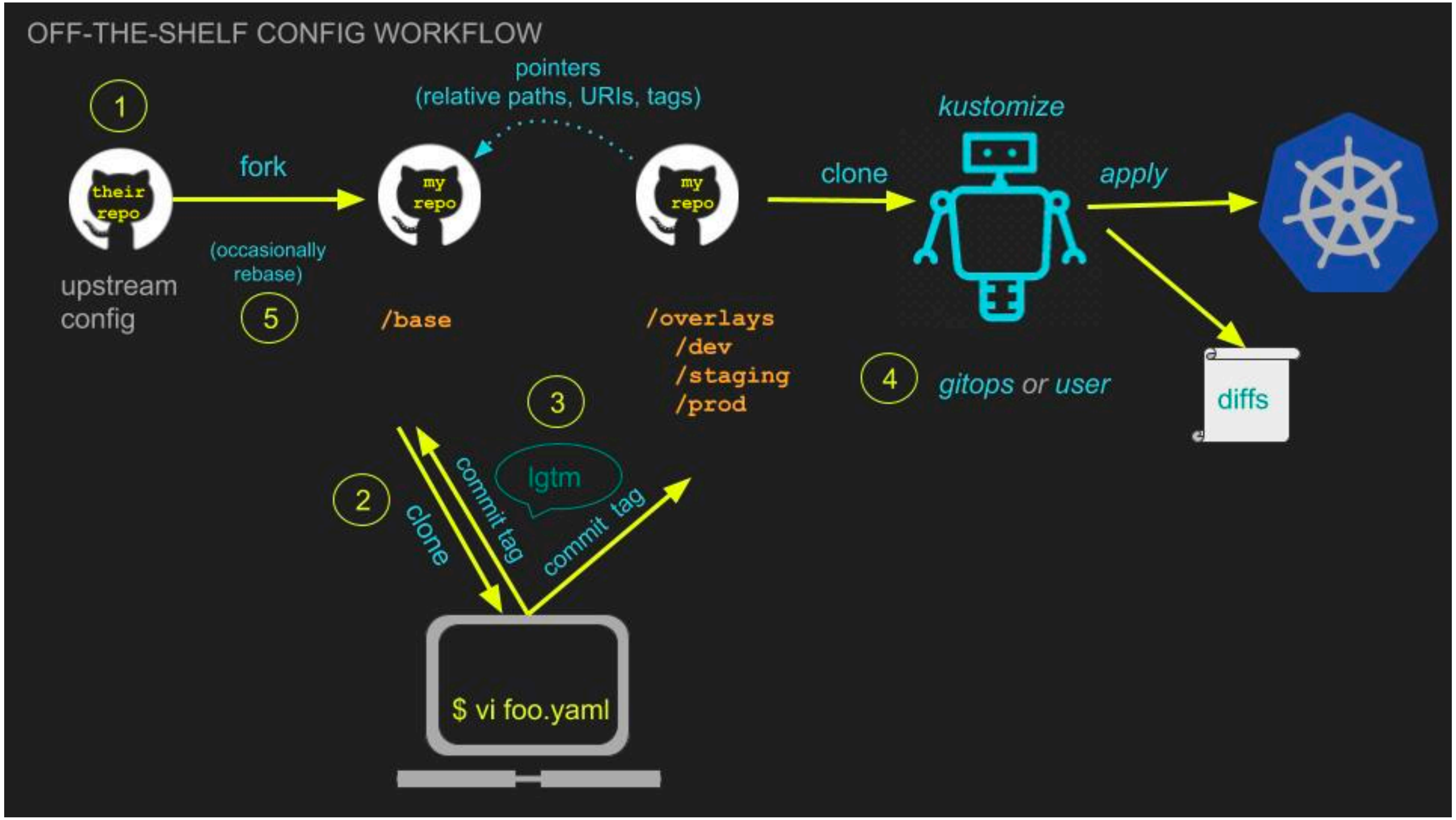
Workflows / Bespoke config

Bespoke config



Workflows / Off-the-shelf config

Off-the-shelf config



Keys

Keys / Overlay vs Template

Drawbacks of Templating

1. Can only override parameterized config
2. DSL is too complicated for human
3. Most tools can not read DSL

Example

1. I’m using official Redis Helm chart
2. I want to add annotation
3. Annotations are not defined in the chart…
4. …**Fork**?

With kustomize

You can override **any** part of config with kustomize

Keys / Single source of truth

Before kustomize

1. There is a config file “hello.config”
2. Copy contents of the file
3. Paste it into configMap
4. … I have **2 config sources…**

ConfigMap Generator

```
# base/kustomization.yaml
resources:
- deployment.yaml
configMapGenerator:
- name: hello-config
  files:
  - hello.config
```

ConfigMap Generator

```
# hello.config
name=hello-world
region=tokyo
```

ConfigMap Generator

```
$ kustomize build hello-world/overlays/production/
```

ConfigMap Generator

```
apiVersion: v1
data:
  hello.config: |
    name=hello-world
    region=tokyo
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: hello-config-4g5t58m8t5
---
apiVersion: apps/v1
kind: Deployment
...
```

After kustomize

1. There is a config file “hello.config”
2. Run “kustomize build”
3. kustomize generates configMap
4. The config source is **only** “hello.config”

Keys / Rolling ConfigMap Update

Updating existing configMap

1. Update contents of existing configMap
2. Deployment itself is not changed…
3. Deployment still reads old configMap…

Hash suffix

```
apiVersion: v1
data:
  hello.config: |
    name=hello-world
    region=tokyo
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: hello-config-4g5t58m8t5
---
apiVersion: apps/v1
kind: Deployment
...
```

Rolling ConfigMap Update

1. Update contents of configMap
2. kustomize prints **new** configMap
3. Update configMap name in deployment
4. Deployment reads new configMap

Keys / Teaching native k8s APIs

*kustomize **exposes** and **teaches native k8s APIs**, rather than hiding them.*

https://github.com/kubernetes-sigs/kustomize/blob/master/docs/glossary.md

Using Native Kubernetes API

Same as **kubernetes manifest**

Using Native Kubernetes API

1. Lower learning cost
2. Deeper understanding about Kubernetes

Keys / Rollback

Rollback

```
$ git checkout XXXXXX
$ kustomize build [PATH] | kubectl apply -f -
```

Rollback

kustomize rollback is very good for GitOps.

However, I also like heroku style rollback such as “helm status”, “helm history”, “helm rollback”.

Helm provides us logical group of k8s resources as “application”. kustomize doesn’t.

Related issue

Kubernetes Application proposal KEP

https://github.com/kubernetes/community/pull/1629

Keys / might be moved to kubectl

*Kustomize was initially developed as its own cli, however once it has matured, i**t should be published as a subcommand of kubectl** or as a statically linked plugin.*

https://github.com/kubernetes/community/blob/master/keps/sig-cli/0008-kustomize.md#implementation-detailsnotesconstraints-optional

Keys / See design doc!

It’s **awesome**!

https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/declarative-application-management.md

https://github.com/kubernetes/community/blob/master/keps/sig-cli/0008-kustomize.md#implementation-detailsnotesconstraints-optional

It’s **awesome**!

If kustomize looks easy to use for you,
I think it comes from **good design**!

Questions

Can I **delete** labels with overlay?

As far as I know, you can not for now

End