kubernetes configuration customization

a composable, template-free approach

Nov 2018 kubecon

上海

jeff regan
jregan@google
monopole@github

Kubernetes 配置的用户定制

声明式的途径

Nov 2018 kubecon

上海

jeff regan
jregan@google
monopole@github

~60

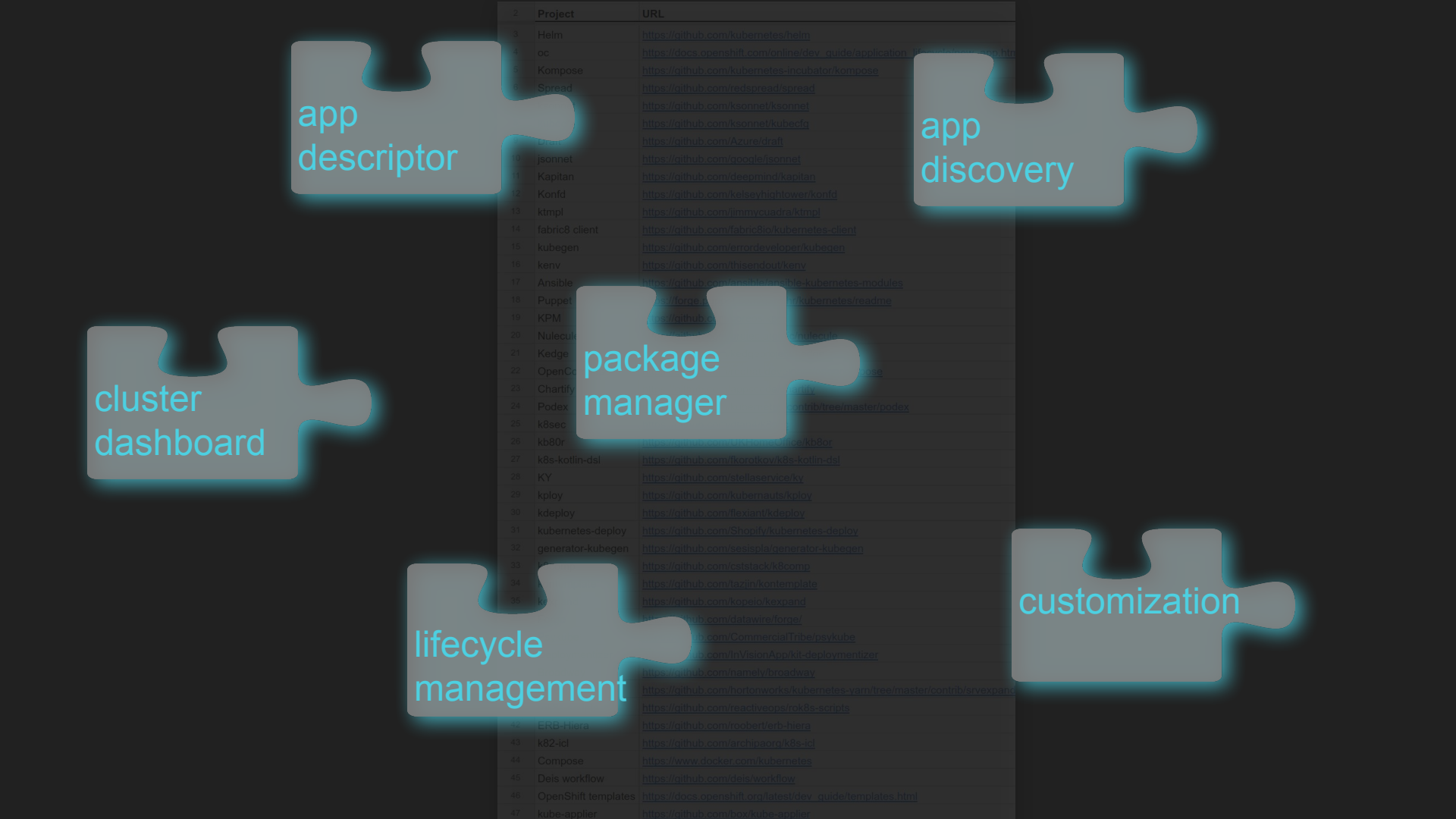
k8s configuration tools
k8s的配置工具

recent -
automation broker

	Project	URL
3	Helm	https://github.com/kubernetes/helm
4	oc	https://docs.openshift.com/online/dev_guide/application_lifecycle/new_app.htm
5	Kompose	https://github.com/kubernetes-incubator/kompose
6	Spread	https://github.com/redspread/spread
7	ksonnet	https://github.com/ksonnet/ksonnet
8	kubecfg	https://github.com/ksonnet/kubecfg
9	Draft	https://github.com/Azure/draft
10	jsonnet	https://github.com/google/jsonnet
11	Kapitan	https://github.com/deepmind/kapitan
12	Konfd	https://github.com/kelseyhightower/konfd
13	ktmpl	https://github.com/jimmycuadra/ktmpl
14	fabric8 client	https://github.com/fabric8io/kubernetes-client
15	kubegen	https://github.com/errordeveloper/kubegen
16	kenv	https://github.com/thisendout/kenv
17	Ansible	https://github.com/ansible/ansible-kubernetes-modules
18	Puppet	https://forge.puppet.com/garethr/kubernetes/readme
19	KPM	https://github.com/coreos/kpm
20	Nulecule	https://github.com/projectatomic/nulecule
21	Kedge	https://github.com/kedgeproject/kedge
22	OpenCompose	https://github.com/redhat-developer/opencompose
23	Chartify	https://github.com/appscode/chartify
24	Podex	https://github.com/kubernetes/contrib/tree/master/podex
25	k8sec	https://github.com/dtan4/k8sec
26	kb80r	https://github.com/UKHomeOffice/kb8or
27	k8s-kotlin-dsl	https://github.com/fkorotkov/k8s-kotlin-dsl
28	KY	https://github.com/stellaservice/ky
29	kploy	https://github.com/kubernauts/kploy
30	kdeploy	https://github.com/flexiant/kdeploy
31	kubernetes-deploy	https://github.com/Shopify/kubernetes-deploy
32	generator-kubegen	https://github.com/sesispla/generator-kubegen
33	k8comp	https://github.com/cststack/k8comp
34	kontemplate	https://github.com/tazjin/kontemplate
35	kexpand	https://github.com/kopeio/kexpand
36	Forge	https://github.com/datawire/forge/
37	Psykube	https://github.com/CommercialTribe/psykube
38	Deploymentizer	https://github.com/InVisionApp/kit-deploymentizer
39	Broadway	https://github.com/namely/broadway
40	Srvexpand	https://github.com/hortonworks/kubernetes-yarn/tree/master/contrib/srvexpand
41	rok8s-scripts	https://github.com/reactiveops/rok8s-scripts
42	ERB-Hiera	https://github.com/roobert/erb-hiera
43	k82-icl	https://github.com/archipaorg/k8s-icl
44	Compose	https://www.docker.com/kubernetes
45	Deis workflow	https://github.com/deis/workflow
46	OpenShift templates	https://docs.openshift.org/latest/dev_guide/templates.html
47	kube-applier	https://github.com/box/kube-applier

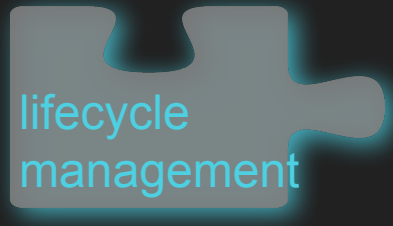
spreadsheet

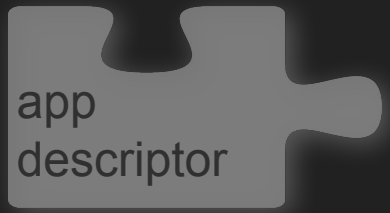
maintained by Brian Grant



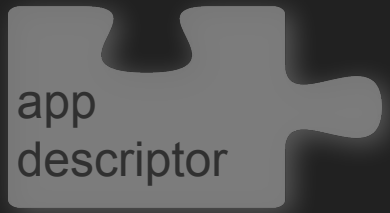
	Project	URL
3	Helm	https://github.com/kubernetes/helm
4	oc	https://docs.openshift.com/online/dev_guide/application_
5	Kompose	https://github.com/kubernetes-incubator/kompose
6	Spread	https://github.com/redspread/spread
7		https://github.com/ksonnet/ksonnet
8		https://github.com/ksonnet/kubecfg
9		https://github.com/Azure/draft
10	jsonnet	https://github.com/google/jsonnet
11	Kapitan	https://github.com/deepmind/kapitan
12	Konfd	https://github.com/kelseyhightower/konfd
13	ktmpl	https://github.com/jimmycuadra/ktmpl
14	fabric8 client	https://github.com/fabric8io/kubernetes-client
15	kubegen	https://github.com/errordeveloper/kubegen
16	kenv	https://github.com/thisendout/kenv
17	Ansible	https://github.com/ansible/ansible-kubernetes-modules
18	Puppet	https://forge.p /kubernetes/readme
19	KPM	https://github.c
20	Nulecul	/nulecul
21	Kedge	
22	OpenC	ase
23	Chartify	ify
24	Podex	ntrib/tree/master/podex
25	k8sec	
26	kb80r	https://github.com/UKHomeOffice/kb8or
27	k8s-kotlin-dsl	https://github.com/fkorotkov/k8s-kotlin-dsl
28	KY	https://github.com/stellaservice/ky
29	kploy	https://github.com/kubernauts/kploy
30	kdeploy	https://github.com/flexiant/kdeploy
31	kubernetes-deploy	https://github.com/Shopify/kubernetes-deploy
32	generator-kubegen	https://github.com/sesispla/generator-kubegen
33	k8	https://github.com/cststack/k8comp
34	K	https://github.com/tazjin/kontemplate
35	k	https://github.com/kopeio/kexpand
		ub.com/datawire/forge/
		b.com/CommercialTribe/psykube
		ub.com/InVisionApp/kit-deploymentizer
		https://github.com/namely/broadway
		https://github.com/hortonworks/kubernetes-yarn/tree/master/contrib/srvexpan
		https://github.com/reactiveops/rok8s-scripts
42	ERB-Hiera	https://github.com/roobert/erb-hiera
43	k82-icl	https://github.com/archipaorg/k8s-icl
44	Compose	https://www.docker.com/kubernetes
45	Deis workflow	https://github.com/deis/workflow
46	OpenShift templates	https://docs.openshift.org/latest/dev_guide/templates.html
47	kube-applier	https://github.com/box/kube-applier

Description, maintainer, version, ...

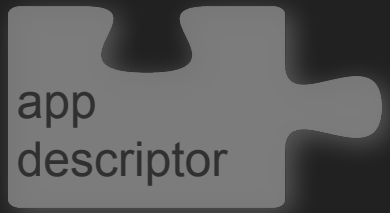




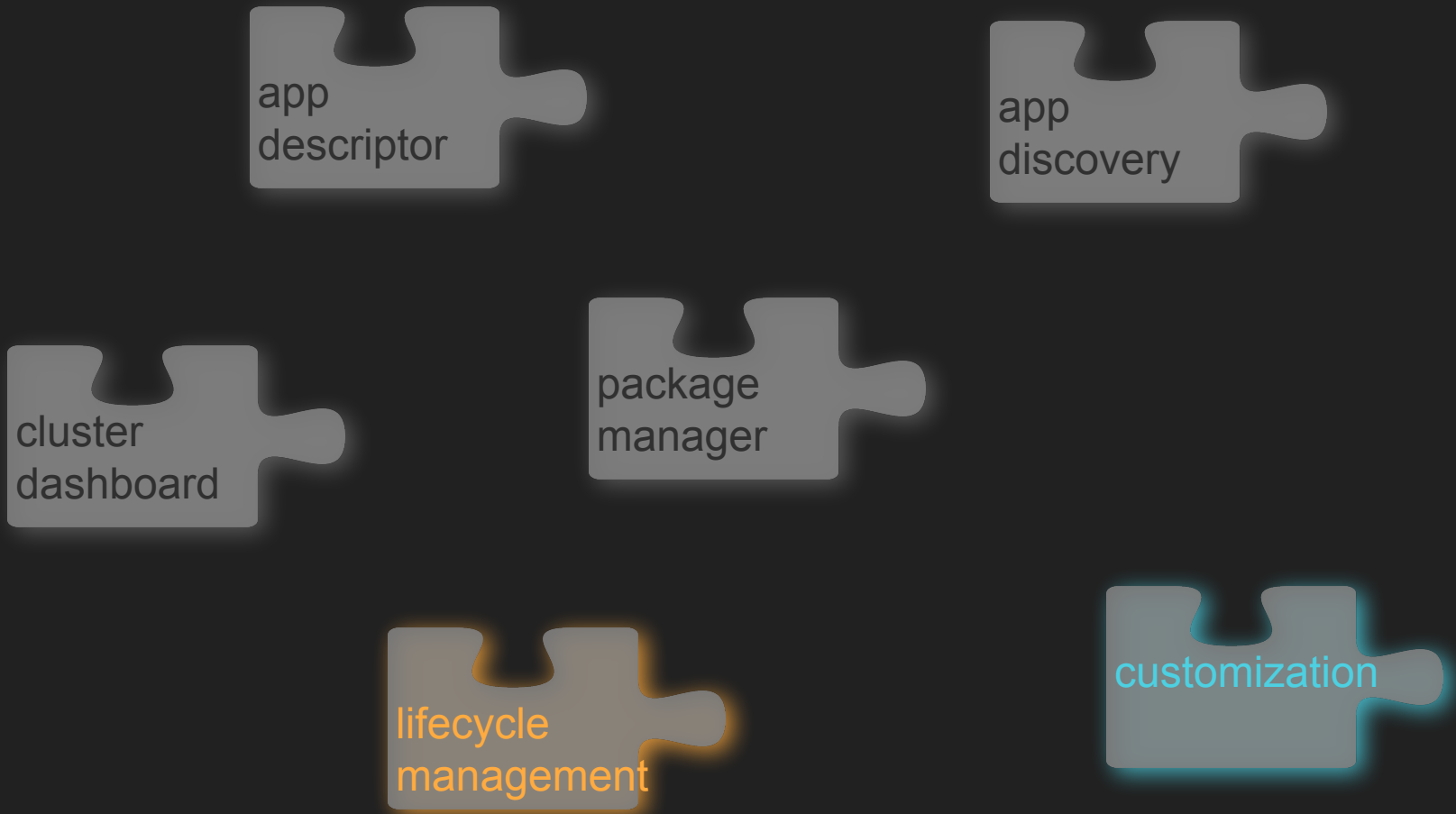
Browse, search, download

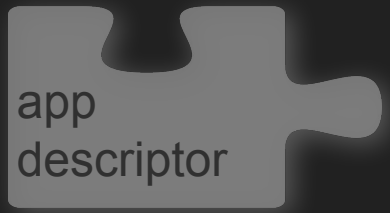


Bundling, plus dependencies



What apps are running?
Are they healthy?


app
descriptor
app
discovery
cluster
dashboard
package
manager
lifecycle
management
customization
Rollouts, rollbacks, upgrades.



Given config - adapt it to *my* needs.

kustomize

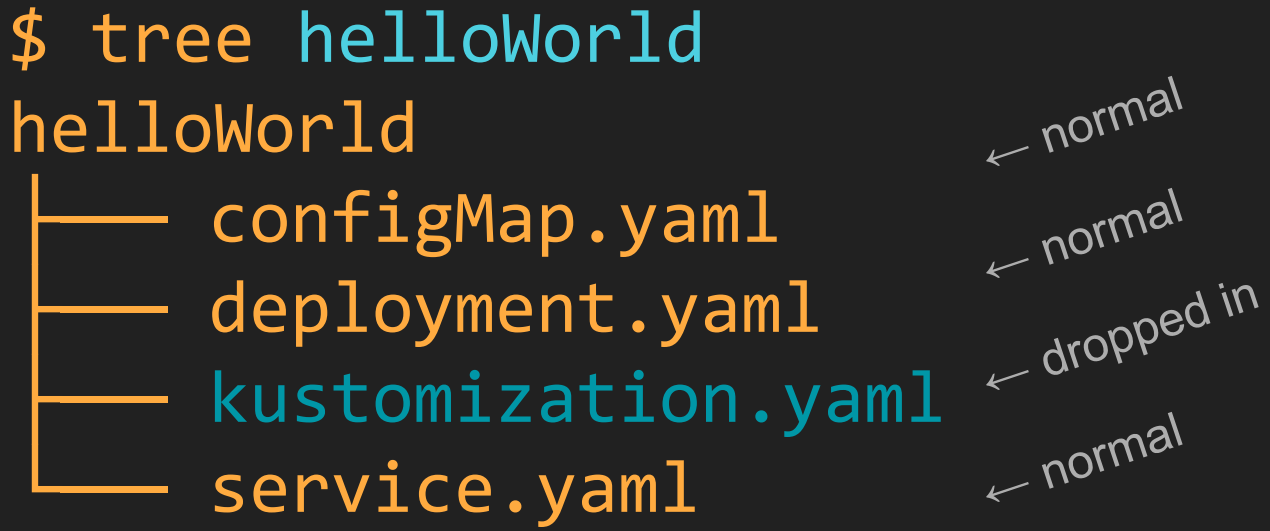
Command line tool for k8s customization.
k8s 用户化的命令行工具

Closes several old kubectl issues.
解决了一些kubectl的老问题

Composes with other tools.
可以和别的工具一起使用

sponsored by sig-CLI per this proposal

```
$ kustomize build helloWorld | \
    kubectl apply -f -
```


$ tree helloWorld
helloWorld
├── configMap.yaml
├── deployment.yaml
├── kustomization.yaml
└── service.yaml
← normal
← normal
← dropped in
← normal

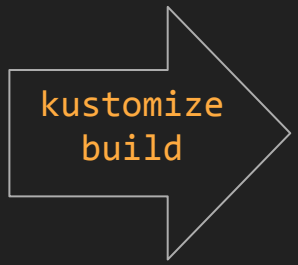
service.yaml

```
kind: Service
metadata:
 name: wordpress
spec:
 ports:
    - port: 389
    selector:
      app: wordpress
```

kustomization.yaml

```
resources:
- service.yaml


namePrefix: demo-
```

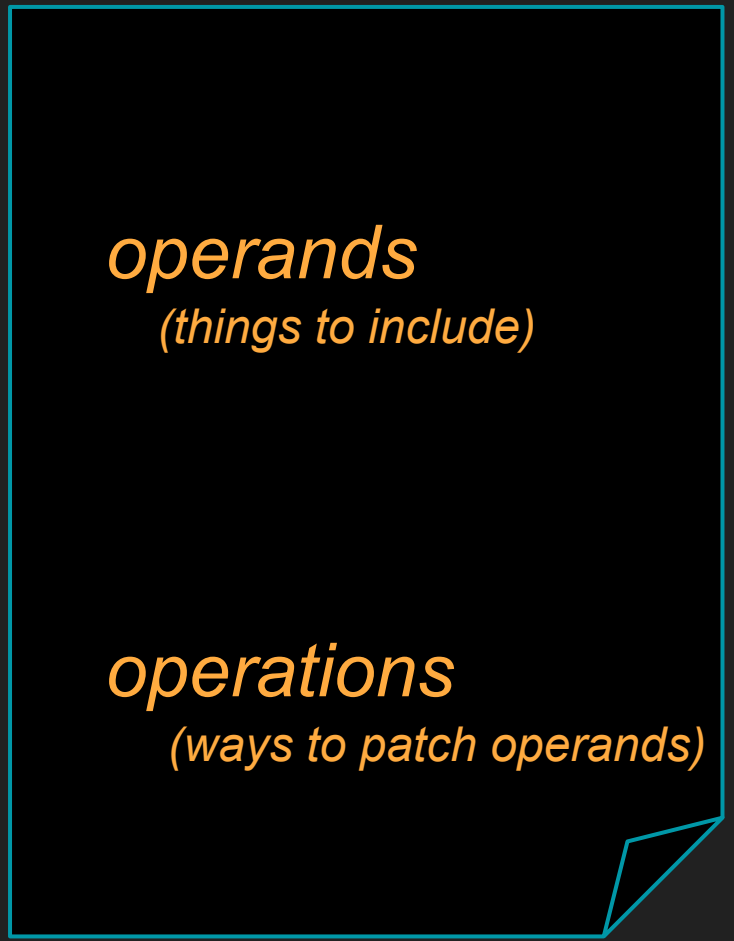


/dev/stdout

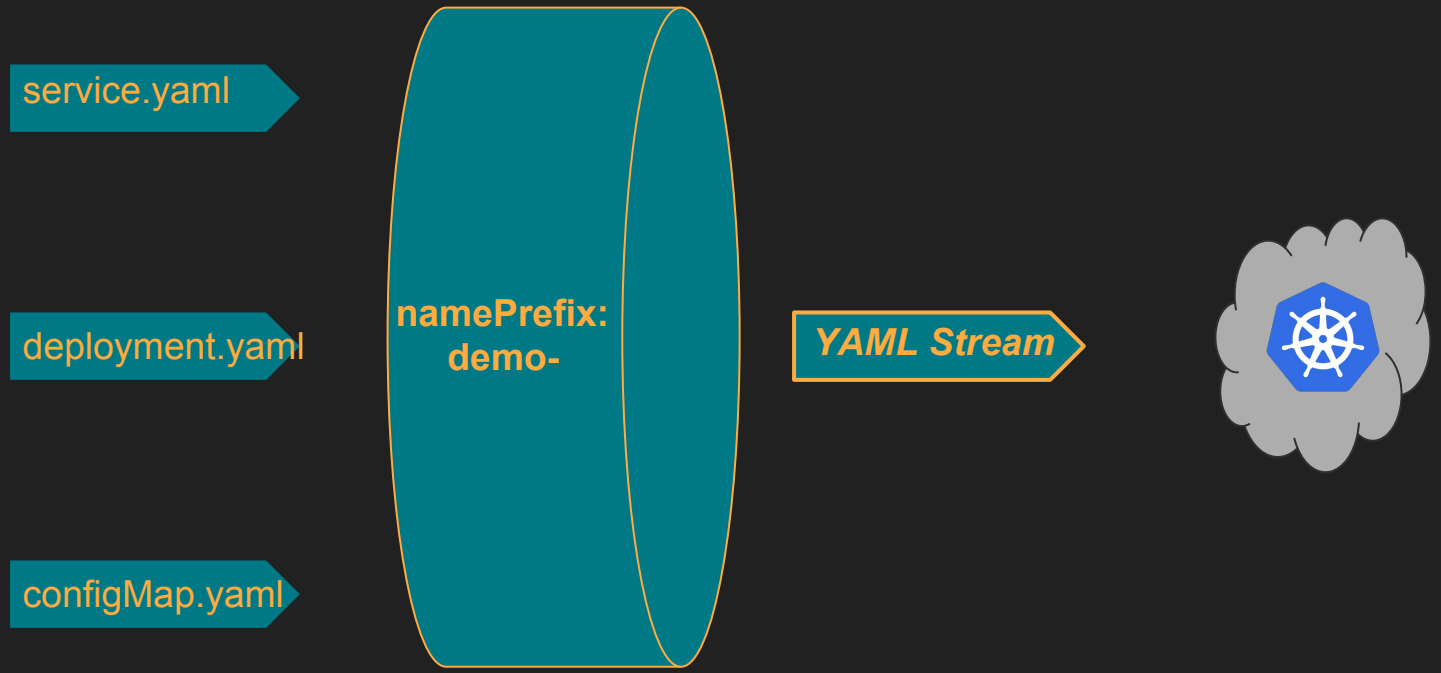
```
kind: Service
metadata:
 name: demo-wordpress
spec:
 ports:
    - port: 389
    selector:
      app: wordpress
```

This is k8s-aware patching.

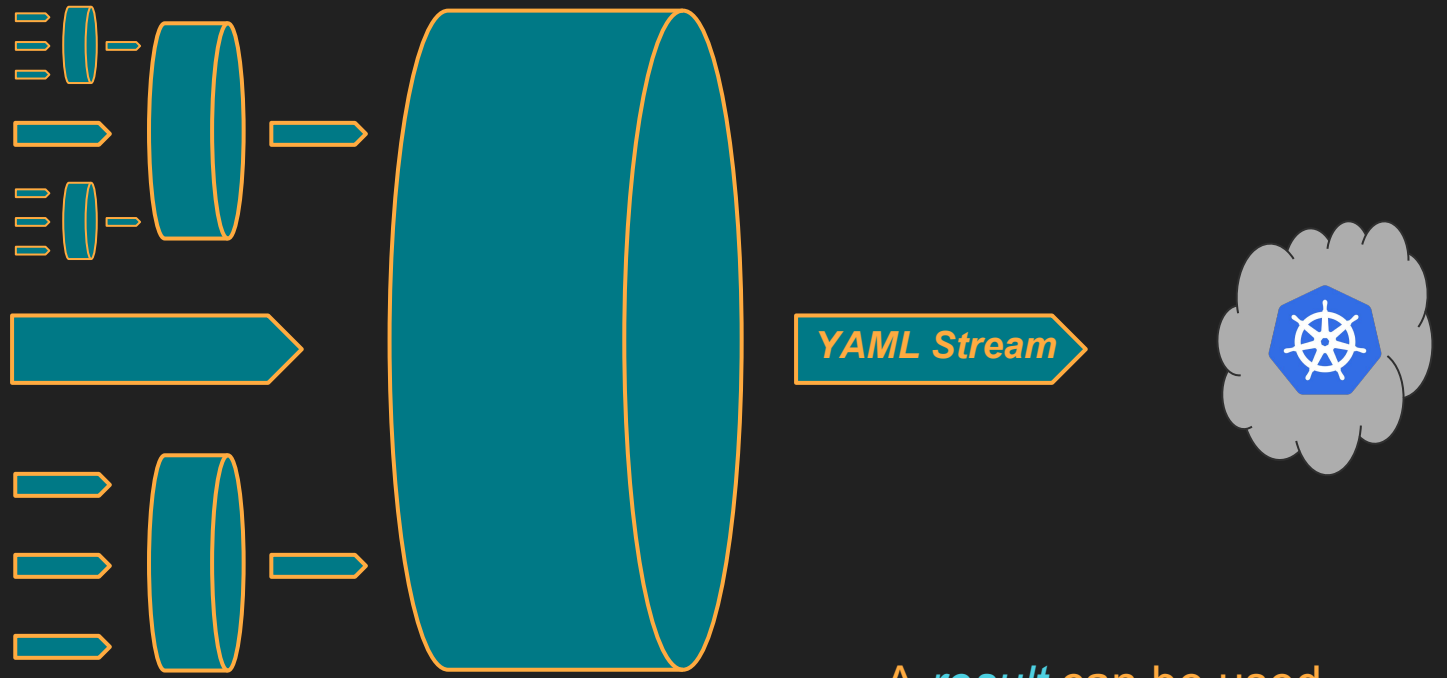
`kustomization.yaml` =



operands *operations* *result*



operands *operations* *result*



A *result* can be used as an *operand*.

`kustomize` input is plain kubernetes yaml.

You can *`kubectl apply`* that yaml without `kustomize`.

To start customizing, just add a kustomization.yaml file.

Use Case #1　Variants 不同环境
(development, staging and production)

common resources

r1.yaml, r2.yaml, ...

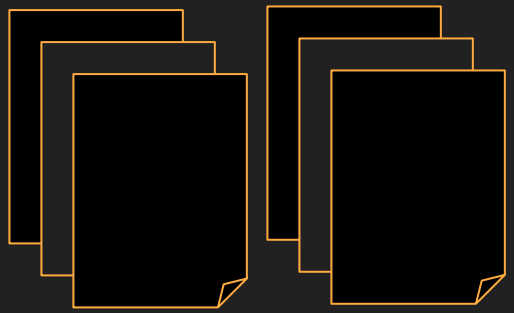


image: hourly build

dev

image: passed QA
1% of prod traffic

staging

image: v2.3.8
replicas: 3000
cpu: 100

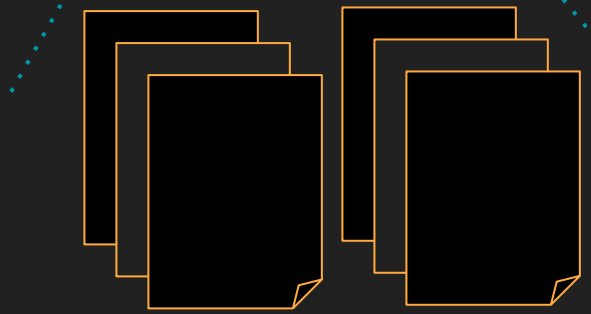
production

Use Case #1　Variants 不同环境

(development, staging and production)

kustomization.yaml

```
resources:
- r1.yaml
  r2.yaml
  ...
```



kustomization.yaml

```
namePrefix: dev-
bases:
- ../../base
```

dev

kustomization.yaml

```
namePrefix: staging-
newTag: qa
bases:
- ../../base
```

staging

kustomization.yaml

```
namePrefix: prod-
newTag: v2.3.8
bases:
- ../../base
```

production

Use Case #1　Variants 不同环境
(development, staging and production)

File layout:

```
$ tree wordpress
wordpress
├── base
│   ├── kustomization.yaml
│   ├── deployment.yaml
│   ├── env.txt
│   └── service.yaml
└── overlays
    ├── production
    │   └── kustomization.yaml
    │   ├── replica_count.yaml
    │   └── cpu_count.yaml
    └── staging
    │   ├── kustomization.yaml
    │   └── ...
    └── dev
        ├── kustomization.yaml
        └── ...
```

kustomization.yaml

```
commonLabels:
  app: wordpress
resources:
- deployment.yaml
- service.yaml
configMapGenerator:
- name: wordpress-map
  files:
  - env.txt
```

service.yaml

```
kind: Service
metadata:
 name: wordpress
spec:
 ports:
    - port: 389
```

deployment.yaml

```
kind: Deployment
metadata:
  name: wordpress
spec:
  replicas: 1
  template: ...
```

Use Case #1　Variants 不同环境
(development, staging and production)

kustomization.yaml

```
namePrefix: prod-
commonLabels:
  variant: prod
commonAnnotations:
  note: I'm Prod!
bases:
- ../../base
patchesStrategicMerge:
- replica_count.yaml
- cpu_count.yaml
```

replica_count.yaml

```
kind: Deployment
metadata:
  name: wordpress
spec:
  replicas: 80
```

cpu_count.yaml

```
kind: Deployment
metadata:
  name: wordpress
spec:
  template:
    spec:
      containers:
      - name: my-container
        resources:
          limits:
            cpu:  7000m
```

```
$ tree wordpress
wordpress
├── base
│   ├── kustomization.yaml
│   ├── deployment.yaml
│   ├── env.txt
│   └── service.yaml
└── overlays
    ├── production
    │   ├── kustomization.yaml
    │   ├── replica_count.yaml
    │   └── cpu_count.yaml
    └── staging
    │   ├── kustomization.yaml
    │   └── ...
    └── dev
        ├── kustomization.yaml
        └── ...
```

Deploy production:

```
$ kustomize build \
    wordpress/overlays/production |\
    kubectl apply -f -
```

```
$ tree wordpress
wordpress
├── base
│   ├── kustomization.yaml
│   ├── deployment.yaml
│   ├── env.txt
│   └── service.yaml
└── overlays
    ├── production
    │   ├── kustomization.yaml
    │   ├── replica_count.yaml
    │   └── cpu_count.yaml
    └── staging
    │   ├── kustomization.yaml
    │   └── ...
    └── dev
        ├── kustomization.yaml
        └── ...
```

Deploy staging:

```
$ kustomize build \
    wordpress/overlays/staging |\
    kubectl apply -f -
```

```
$ tree wordpress
wordpress
├── base
│   ├── kustomization.yaml
│   ├── deployment.yaml
│   ├── env.txt
│   └── service.yaml
└── overlays
    ├── production
    │   ├── kustomization.yaml
    │   ├── replica_count.yaml
    │   └── cpu_count.yaml
    └── staging
    │   ├── kustomization.yaml
    │   └── ...
    └── dev
        ├── kustomization.yaml
        └── ...
```

```
$ kustomize build target
```

1. load universal k8s object descriptions
2. read `kustomization.yaml` from `target`
3. kustomize bases *(recurse 2-5)*
4. load and/or generate resources
5. apply `target's` kustomization operations
6. fix name references
7. emit yaml

Use Case #2　Feeding customized names to containers
把用户化的名字放入容器

kustomization.yaml

```
vars:
  - name: MYSQL_SERVICE
    objref:
        kind: Service
        name: mysql
        apiVersion: v1
      fieldref:
        fieldpath: metadata.name
patchesStrategicMerge:
  - patch.yaml
```

patch.yaml

```
kind: Deployment
metadata:
  name: wordpress
spec:
  template:
    spec:
      initContainers:
      - name: init-command
        image: debian
        command:
        - "curl $(MYSQL_SERVICE)"
      containers:
      - name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: $(MYSQL_SERVICE)
```

/dev/stdout

```
apiVersion: v1
kind: Deployment
...
spec:
  initContainers:
  - command:
    - curl demo-mysql
  containers:
  - env:
    - name: WORDPRESS_DB_HOST
      value: demo-mysql
```

Use Case #3 ConfigMaps generated from multiple sources
归并属性

base

production overlay

/dev/stdout

kustomization.yaml

```
configMapGenerator:
- name: myCMap
  files:
  - common.properties
```

common.properties

```
color=blue
height=10m
```

kustomization.yaml

```
bases:
- ../../base
namePrefix: prod-
configMapGenerator:
- name: myCMap
  behavior: merge
  files:
  - secret.properties
```

secret.properties

```
dbpassword=foo
```

```
kind: ConfigMap
metadata:
  name: prod-myCMap-b5m75cxc
data:
  color=blue
  height=10m
  dbpassword=foo
```

… so on for staging and development variants.

Properties can be owned by *different* teams.

It’s all *patching*.

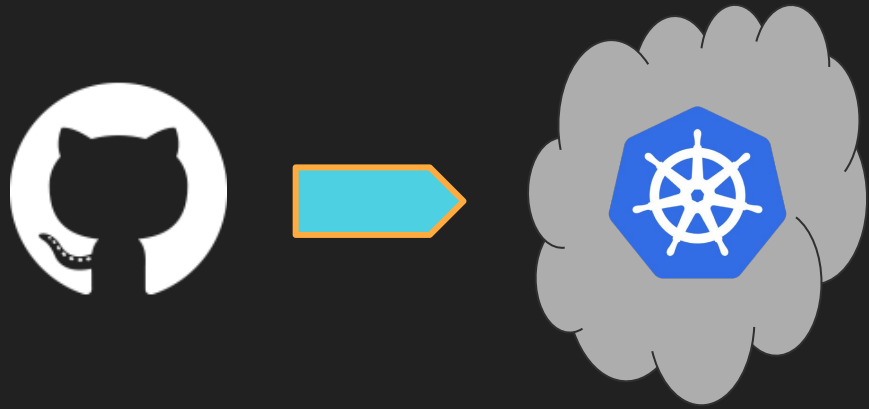
kustomize is just a means to manage k8s-targeted patching.

Things one might want to customize:
你可能想配置以下内容

context	namespaces, names, labels
container	image tag, args, env, config files, secrets, static data
budgets	replicas, cpu, memory, volume source
policies	RBAC, pod security, network

Use Case #4　Deploy from version control
从版本控制进行部署

```
$ kustomize build \
      github.com/kubernetes-sigs/kustomize/examples/helloWorld |\
      kubectl apply -f -
```



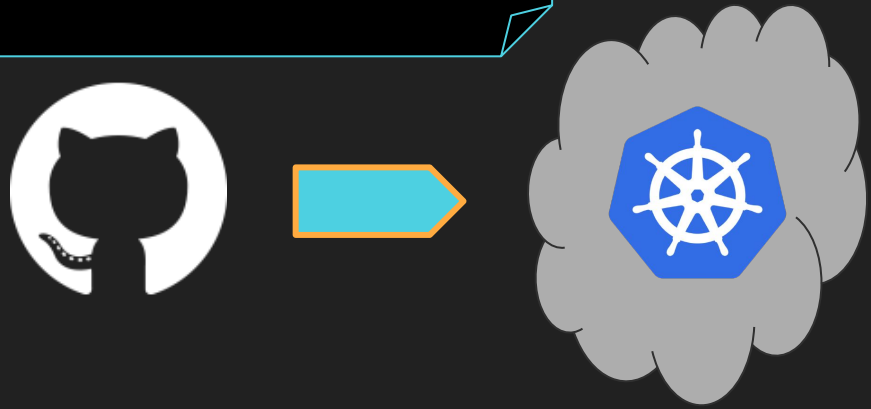
Use Case #4　Deploy from version control
从版本控制进行部署

This deploys from a git tag, merely adding a name prefix.

kustomization.yaml

```
...

namePrefix: hello-
bases:
- github.com/kubernetes-sigs/kustomize/examples/multibases?ref=v1.0.6
```



Use Case #5 Rollback 回滚

```
$ cd target
$ git checkout HEAD~1
$ kustomize build . | kubectl apply -f -
```

Verify: `kustomize build . | kubectl diff -f -`

And/or use `kustomization.yaml` to add git tag annotations to all resources, then verify rollback with kubectl describe.

Use Case #6 Keep up with upgrades

如何和别人的配置升级保持一致？

Fork their config.

Add a `kustomization.yaml` to adapt it to your needs.

Periodically *`git rebase`* to keep up.

No merge conflicts ever → *upgrade can be automated.*

A git repo of resources,

with kustomizations describing variants,

is a simple, powerful way to manage shared configuration patching.


```
yourApp
├── README.md
├── base
│   ├── kustomization.yaml
│   ├── deployment.yaml
│   ├── env.txt
│   └── service.yaml
└── overlays
    ├── production
    │   ├── kustomization.yaml
    │   ├── replica_count.yaml
    │   └── cpu_count.yaml
    └── staging
    │   ├── kustomization.yaml
    │   └── ...
    │...
```

DEMO

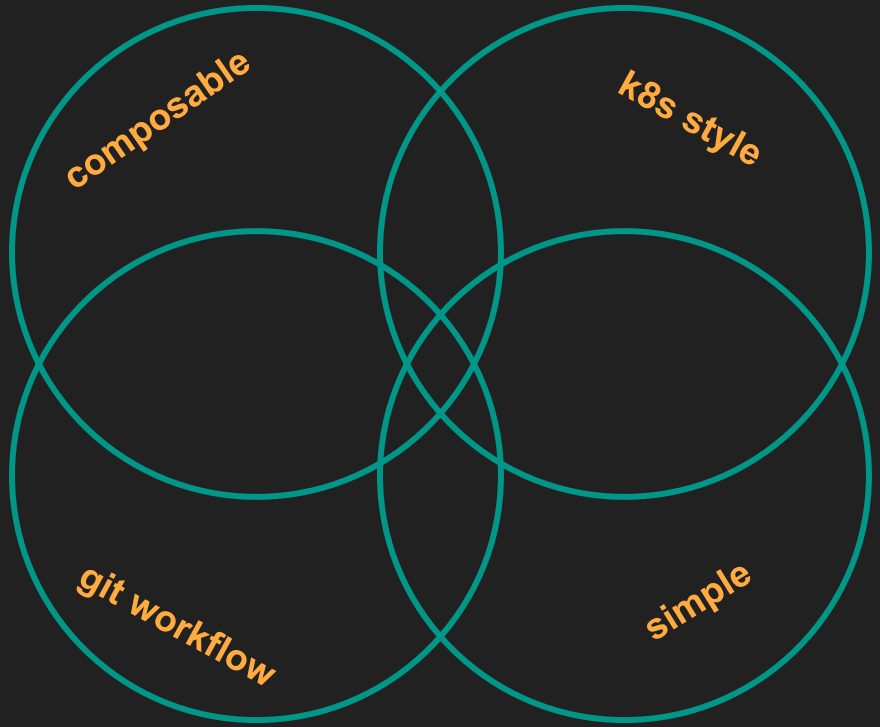
No - too boring

不要啊！好无聊！

It’s just YAML to stdout! Try these examples

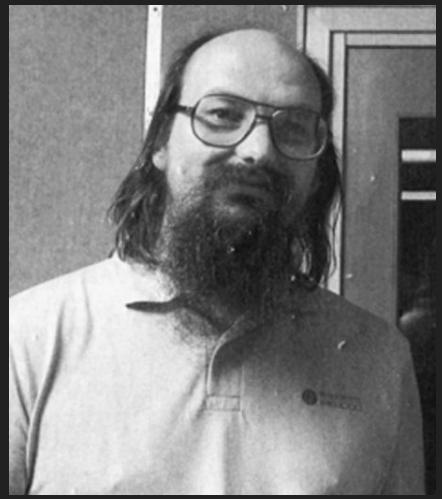
kustomize design goals
kustomize的设计目标

Talk not over yet...
almost!



composable
可组合性

- plain text
- do one thing
- pipe friendly
- say nothing



In 1973 Douglas McIlroy encourages Ken Thompson to add pipes to unix. Doug wrote diff, tee, tr, echo, sort, etc.

```
$ kustomize build $target | kubectl apply -f -
```

k8s style
k8s风格

patch raw yaml resources

extensible to CRDs

targets *kubectl apply*

use patch concepts already in *apply*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.8
        ports:
        - containerPort: 80
...
```

git workflow
工作流

declarative YAML only

diff cluster against git

share bases via git

capture upgrades with rebase

```
# config upgrade script

$ cd yourConfigDirectory
$ git fetch upstream
$ git rebase upstream/master
$ git push origin master
```

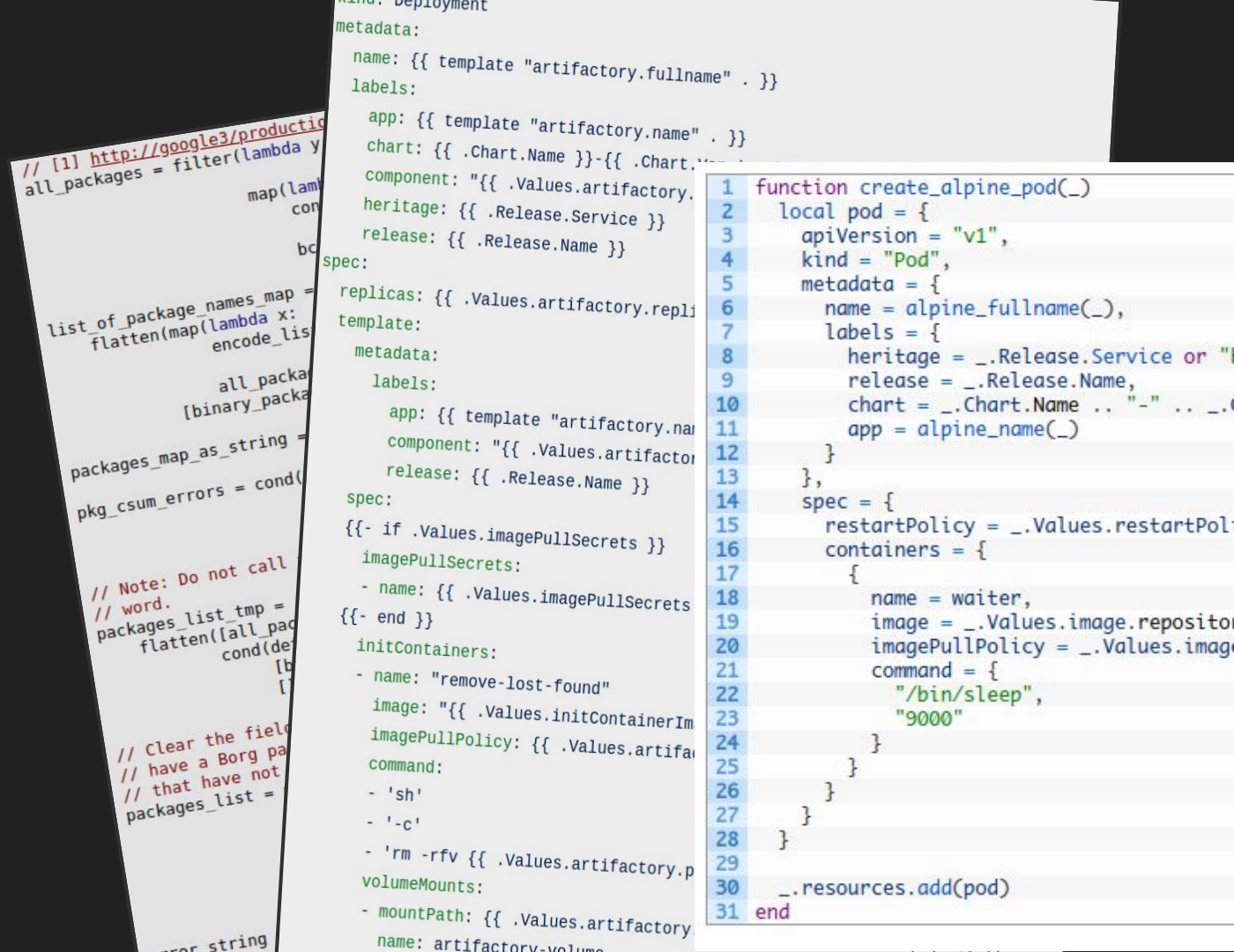
simple
简单

no templating

no new language

no unintentional API

no foreign notion of a
‘*package*’ or ‘*application*’



Already integrated into skaffold, kubebuilder, Replicated Ship, ...

Coming soon:

kubectl integration `$ kubectl apply -f target`

← will honor kustomization.yaml if present

example site like *godoc.org*, but for kustomizations

Thanks!	非常感谢！
contributors	<- please file issues / help
replicated	kustomize.io!
sig-cli	sponsoring kustomize!
brian grant	principal eng @google, k8s founder, author of declarative application management in Kubernetes, the inspiration for kustomize.


Questions
提问时间

backup slides

Template / DSL paradox:

The more people you try to please, the worse it gets.

Helm Strategy

1. Download *someChart*
2. `$ helm template someChart --output_dir target`
3. Add your own kustomization.yaml to target.
4. Commit target to git.
5. Periodically do (1) again and *manually* merge changes.
   ...Or not. Do you think it's important?

What would you say you do here?

I find an example.
Then I kustomize it.

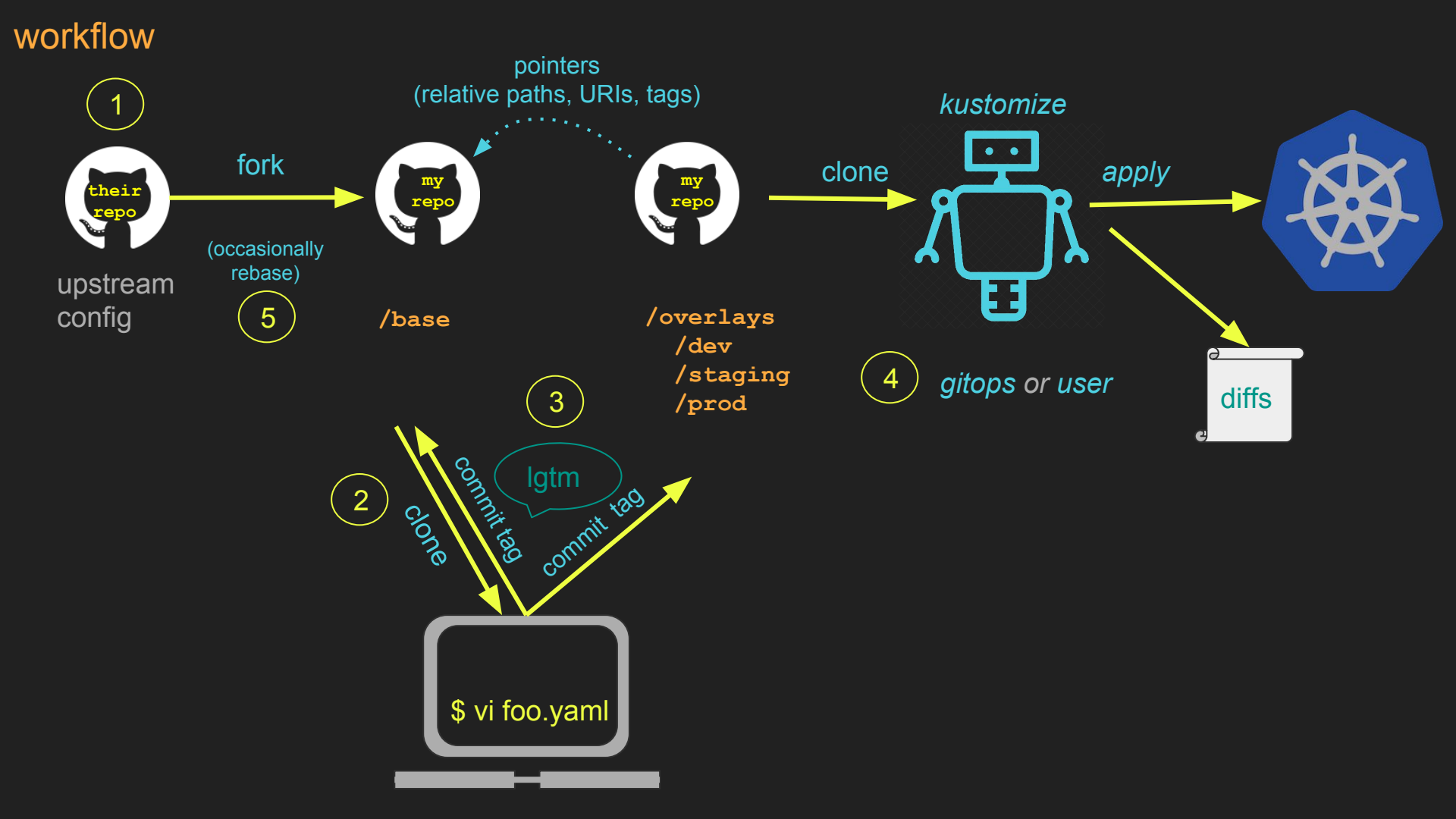
Stills from from Office Space (1999) Twentieth Century Fox

operands

- *resources* - *file names on disk*
- *generated resources* - *instructions*
- *CRDs* - *expands the list of recognized resources*
- *bases* - *nested kustomizations*

operations

- *add name prefix*
- *add labels and annotations*
- *patch… (etc.)*


workflow
1
their repo
upstream config
fork
(occasionally rebase)
5
pointers
(relative paths, URIs, tags)
my repo
/base
my repo
/overlays
/dev
/staging
/prod
clone
kustomize
apply
4
gitops or user
diffs
3
lgtm
2
clone
commit tag
commit tag
$ vi foo.yaml

Config sharing:

- Share your resource set with the team / company / world.
  They kustomize it, and rebase periodically.

- Dev-ops shares common resources to company eng teams.
  They add their own kustomizations.

- Share common resources to different environments -
  dev/staging/production, blue/green, etc.,
  kustomizing the differences.