kubernetes configuration customization

a composable approach

Nov 2018 @heavybit
jeff regan
monopole@github
jregan@google

~60

k8s configuration tools

2	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

recent -
automation broker

spreadsheet

maintained by Brian Grant

app descriptor

app discovery

cluster dashboard

package manager

lifecycle management

customization

	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	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	s://forge. /kubernetes/readme
19	KPM	s://github.
20	Nulecul	nulecule
21	Kedge	
22	OpenC	ess
23	Chartify	uby
24	Podex	ontrib/tree/master/podex
25	k8sec	
26	kb80r	ce/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		https://github.com/cststack/k8comp
34		https://github.com/tazjin/kontemplate
35		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/srvexpand
		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

app descriptor

Description, maintainer, version, ...

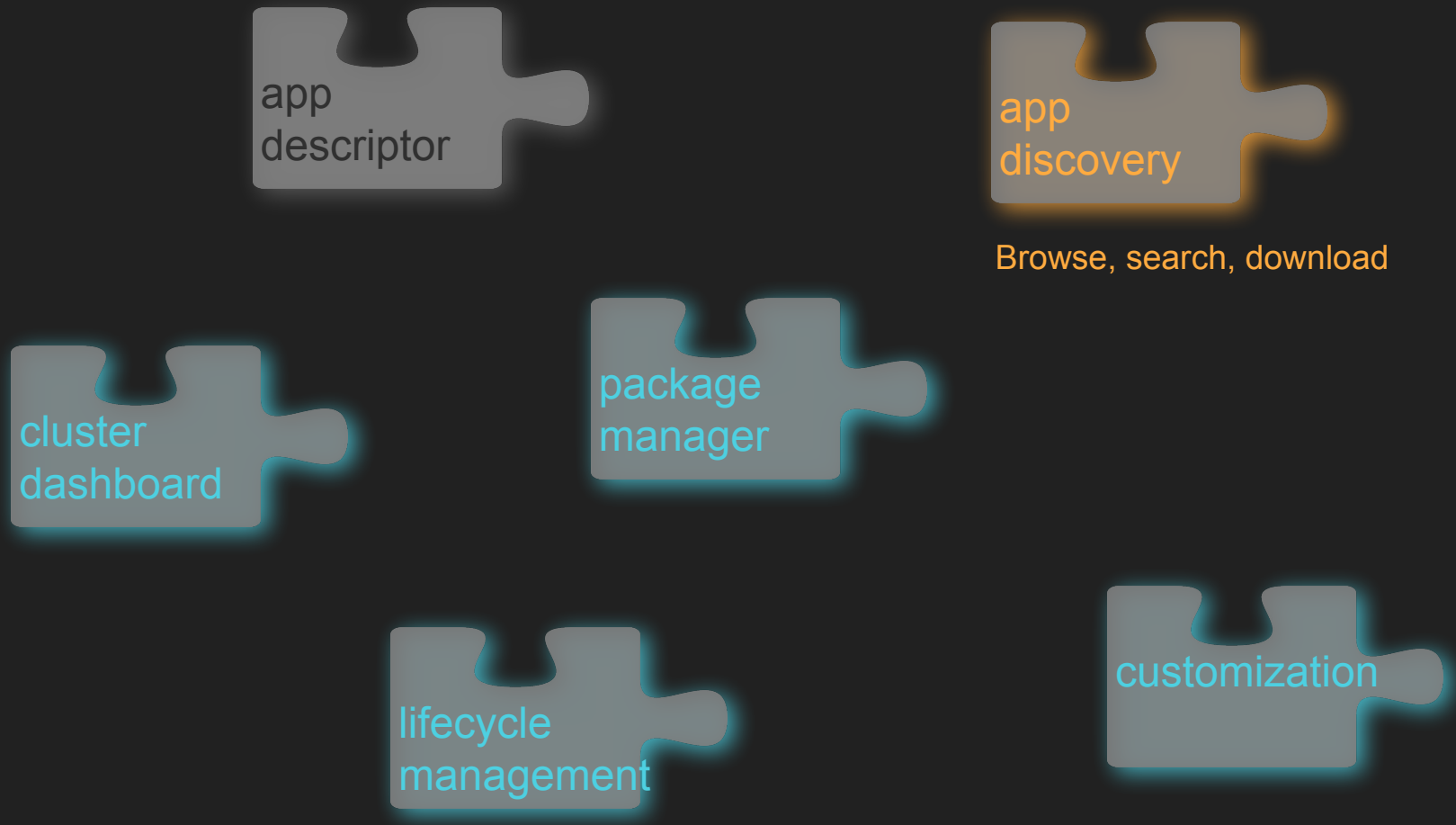
app discovery

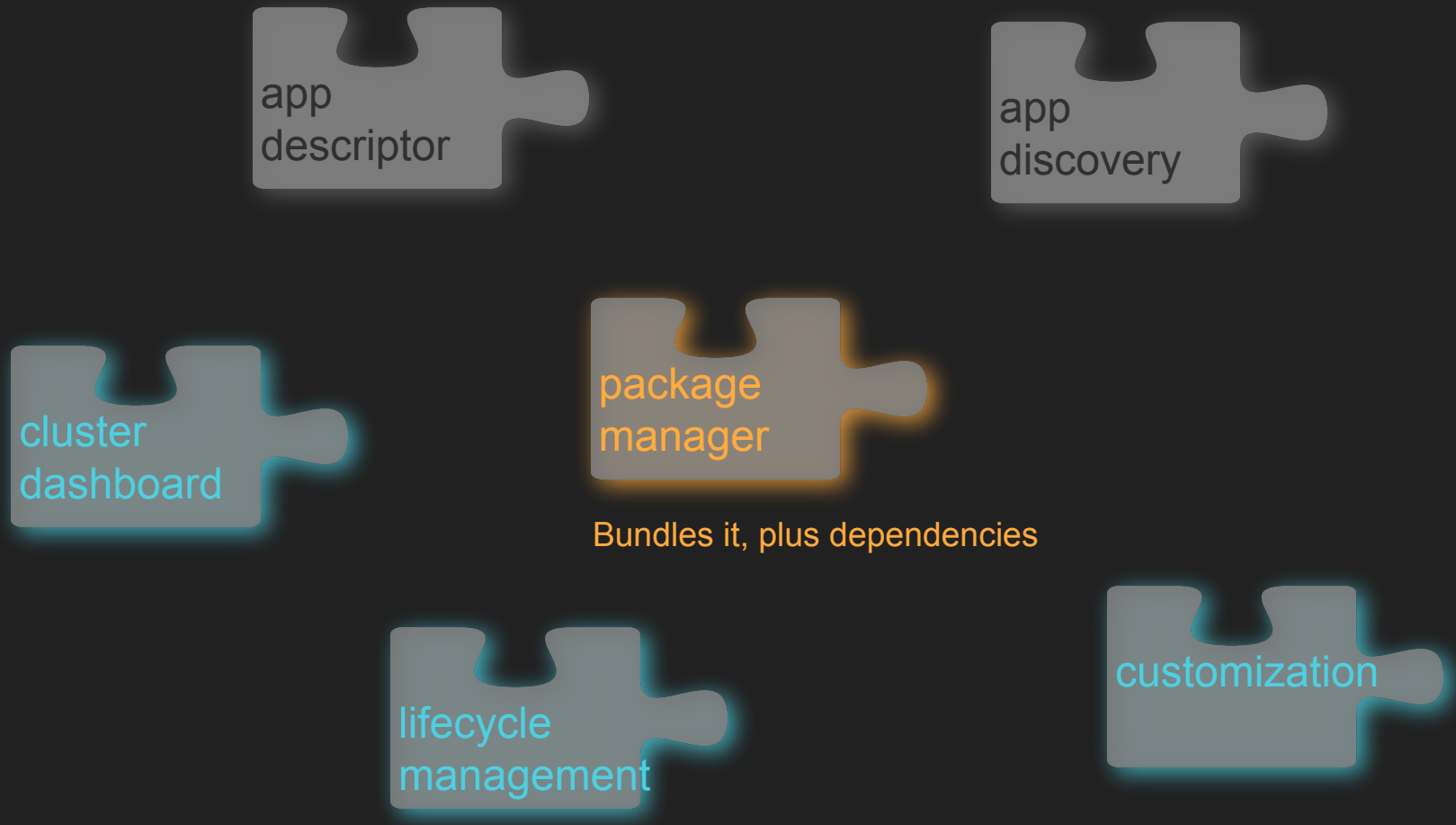
cluster dashboard

package manager

lifecycle management

customization


app descriptor
app discovery
Browse, search, download
cluster dashboard
package manager
lifecycle management
customization


app descriptor
app discovery
cluster dashboard
package manager
Bundles it, plus dependencies
lifecycle management
customization

app descriptor

app discovery

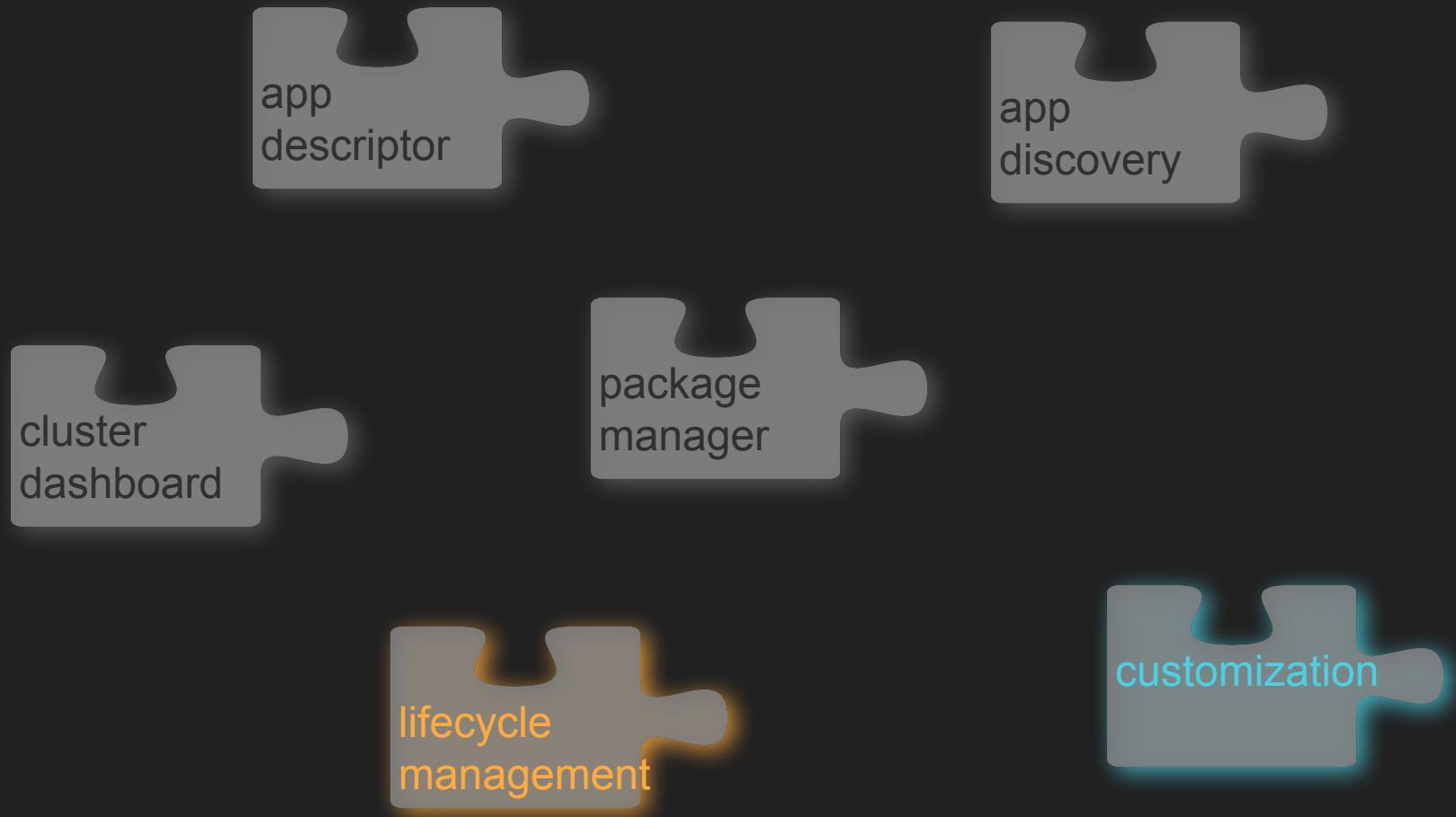
package manager

cluster dashboard

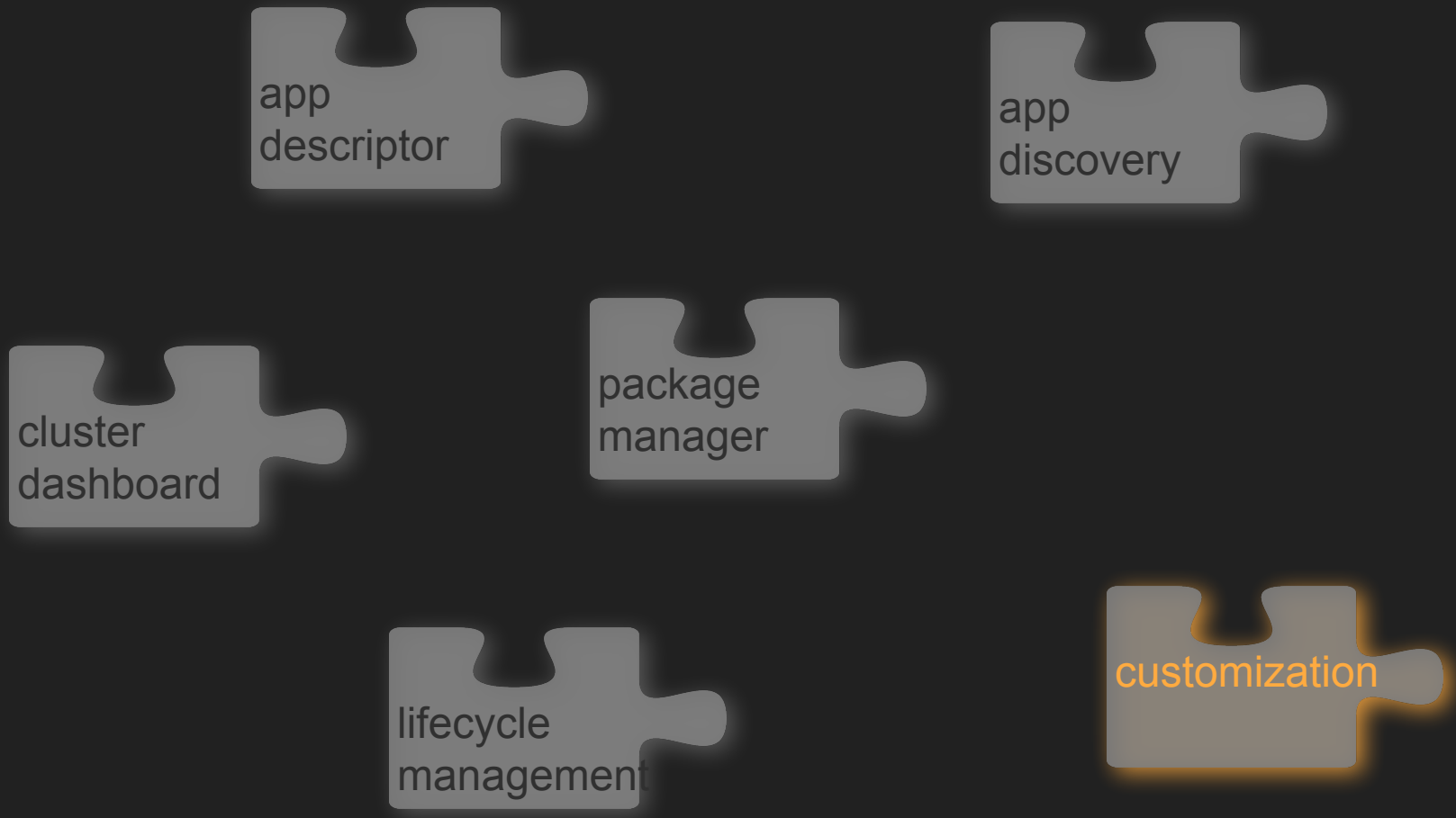
What apps are running?
Are they healthy?

lifecycle management

customization



Rollouts, rollbacks, upgrades.



Adjust it to do what I want.

kustomize

Command line tool for k8s customization.

Closes several old kubectl issues.

Composes with other tools.

sponsored by sig-CLI per this proposal

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

```
$ tree helloWorld
helloWorld
├── configMap.yaml
├── deployment.yaml
├── kustomization.yaml
└── service.yaml
```

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
 name: wordpress
spec:
 ports:
    - port: 389
    selector:
      app: wordpress
```

kustomization.yaml

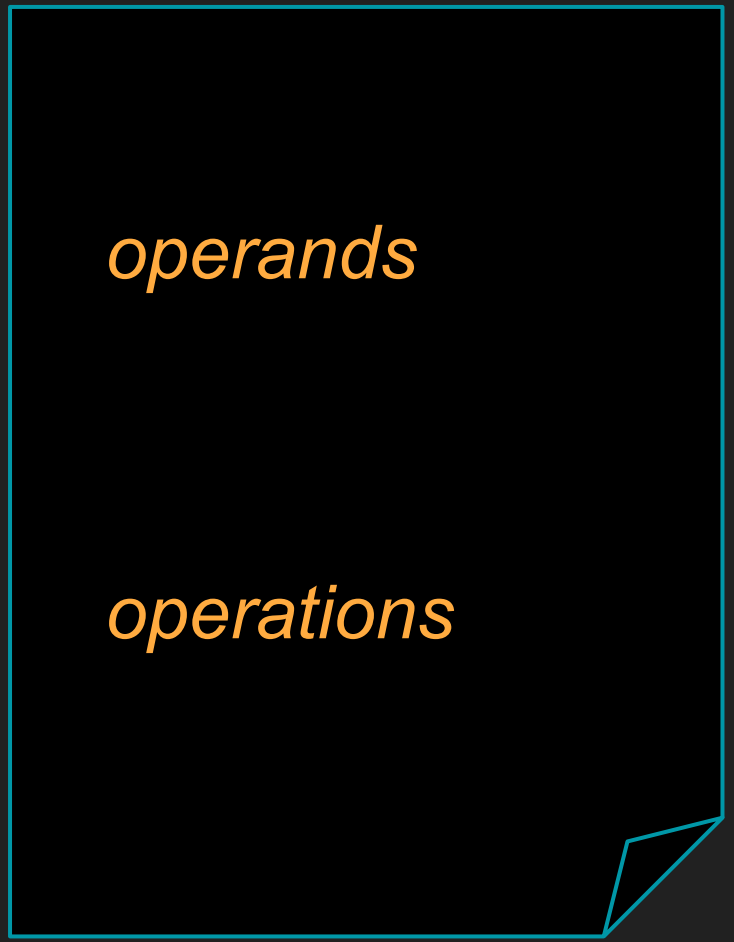
```yaml
resources:
- service.yaml

namePrefix: demo-
```

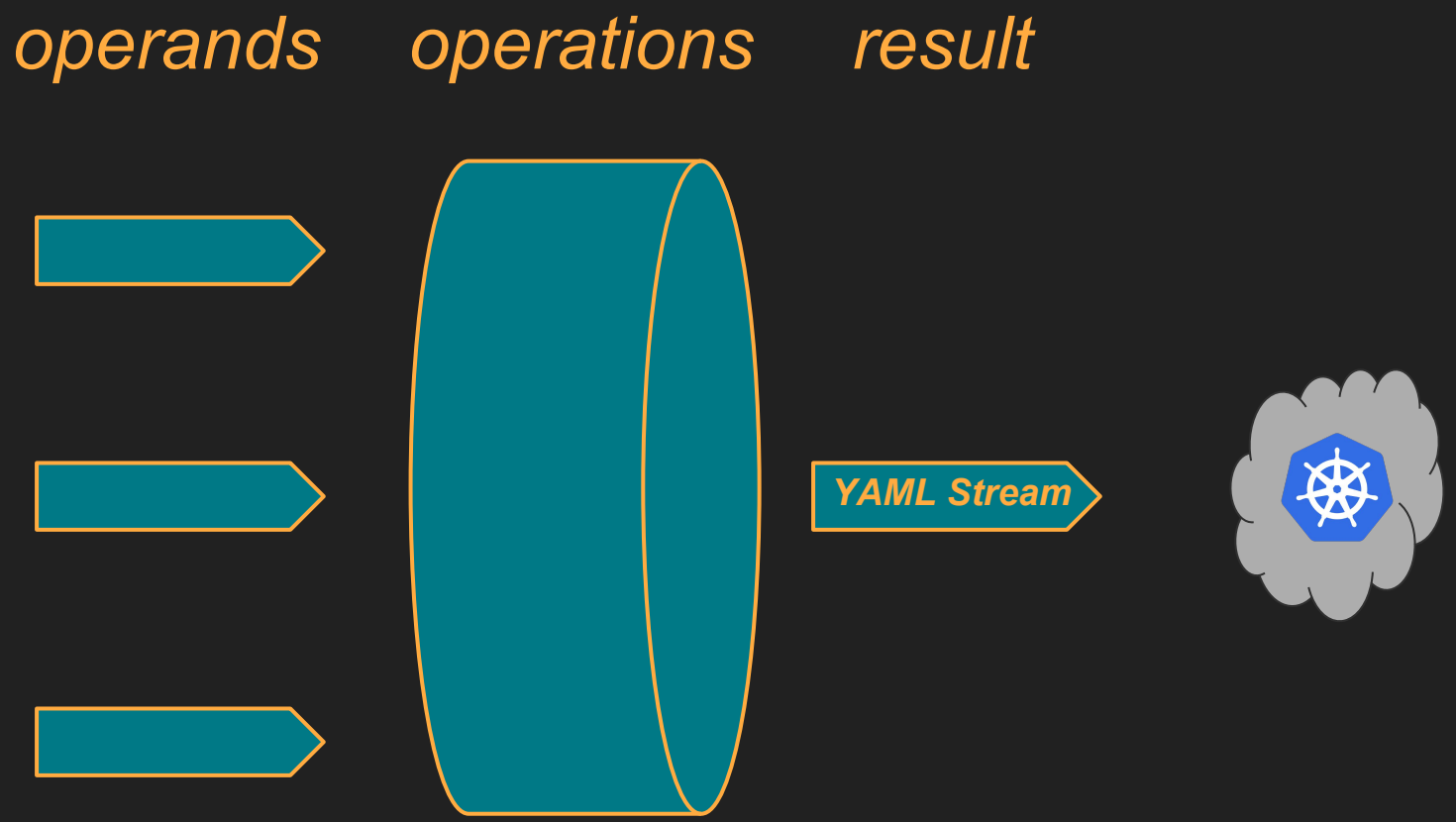
kustomize build

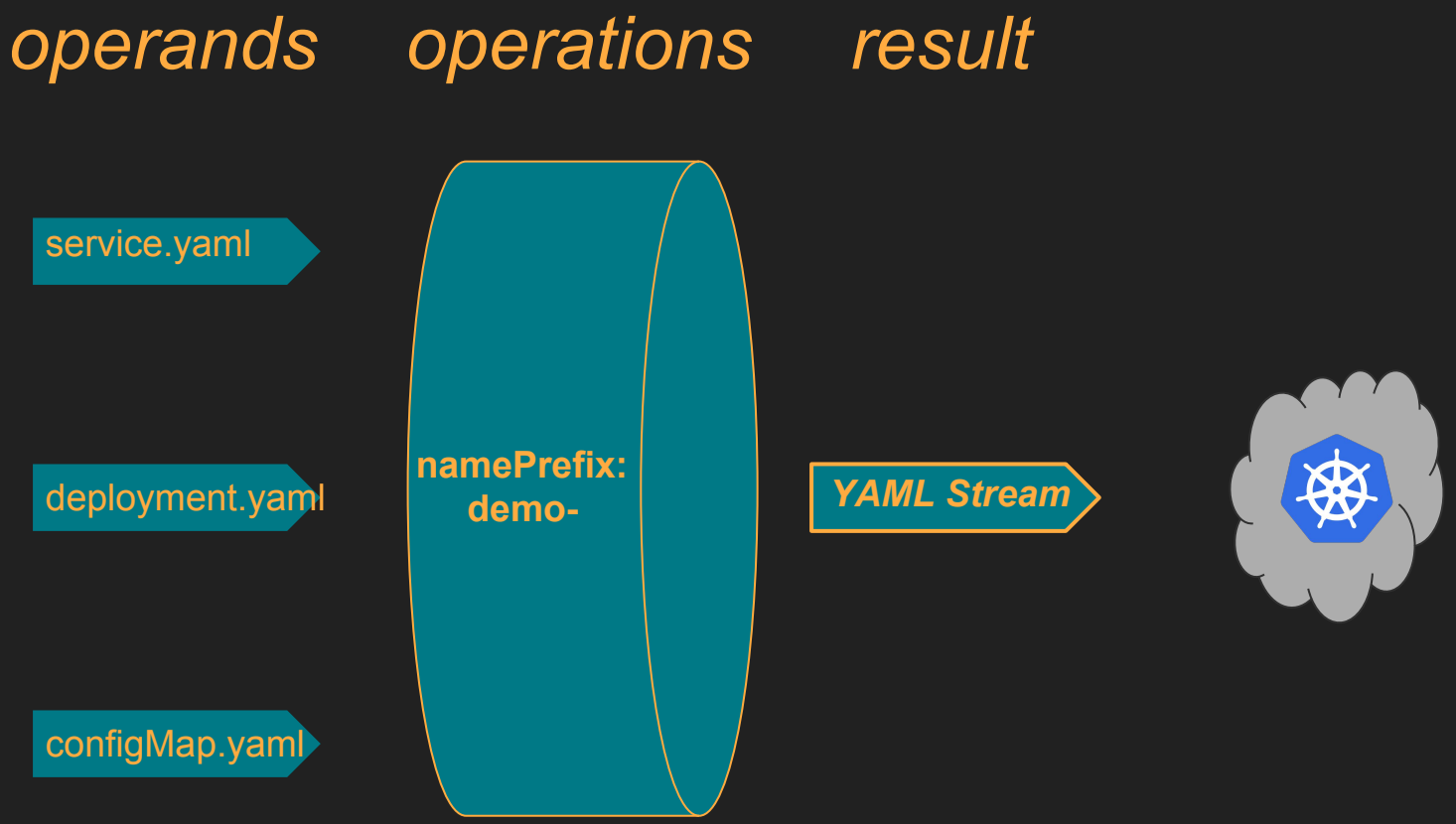
/dev/stdout

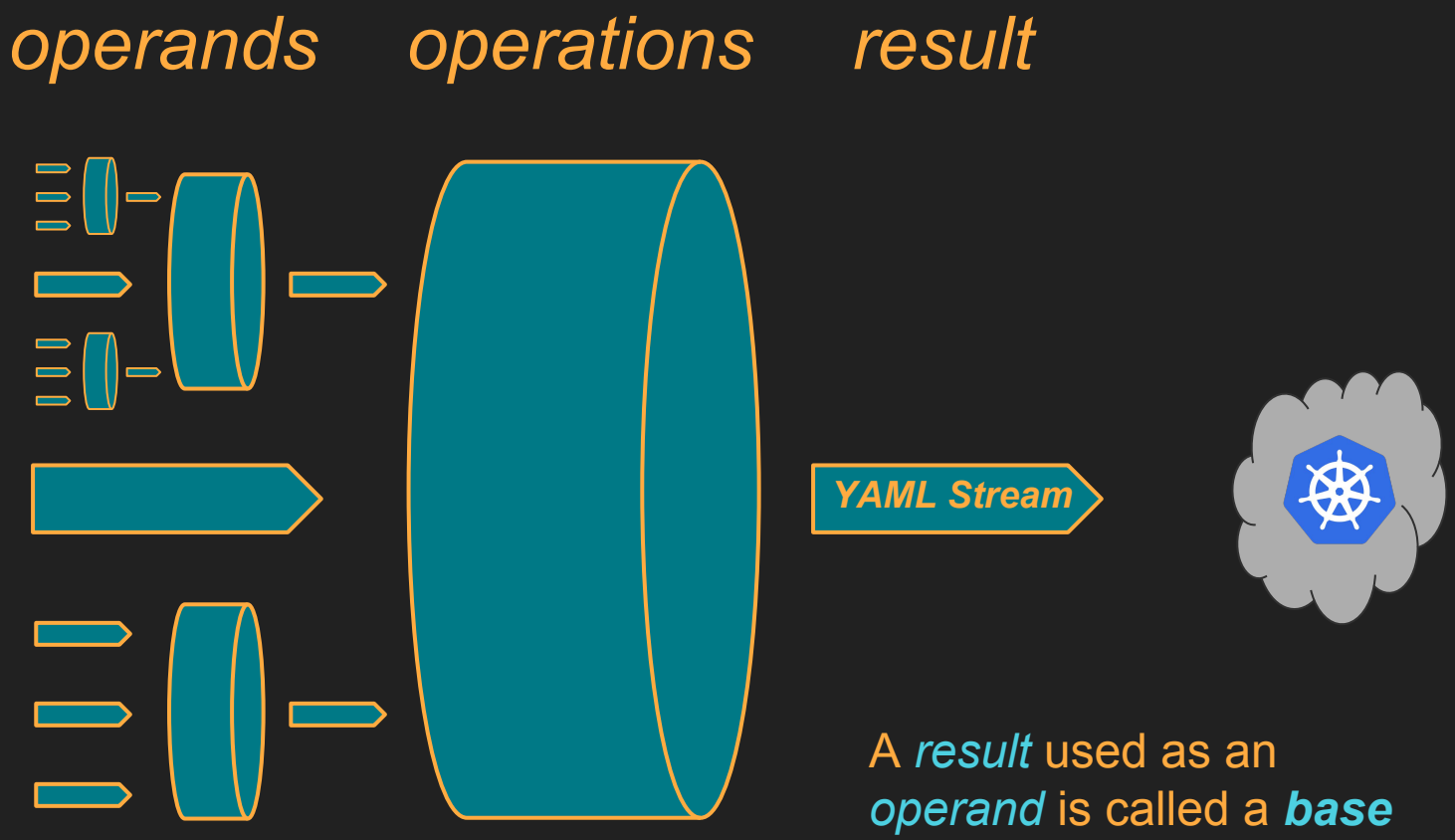
```yaml
apiVersion: v1
kind: Service
metadata:
 name: demo-wordpress
spec:
 ports:
    - port: 389
    selector:
      app: wordpress
```

`kustomization.yaml` =




operands
operations
result
YAML Stream


operands
operations
result
service.yaml
deployment.yaml
configMap.yaml
namePrefix:
demo-
YAML Stream


operands
operations
result
YAML Stream
A result used as an operand is called a base

`kustomize` reads plain kubernetes yaml.

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

Drop in a kustomization.yaml file to start.

Things one might want to customize

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

$ 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 #1 Variants
(dev, staging and production)

common resources

r1.yaml, r2.yaml, ...

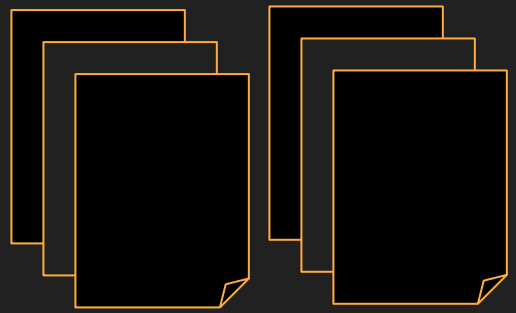


image: hourly build

dev

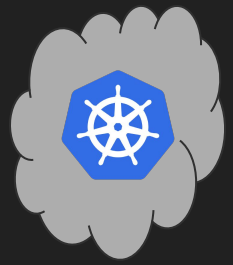


image: passed QA
1% of prod traffic

staging

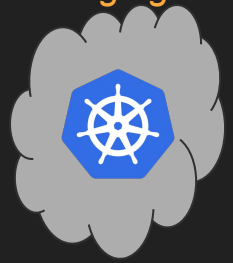
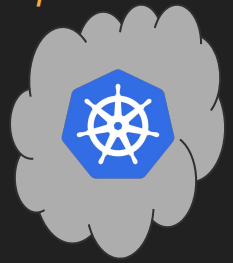


image: v2.3.8
replicas: 3000
cpu: 100

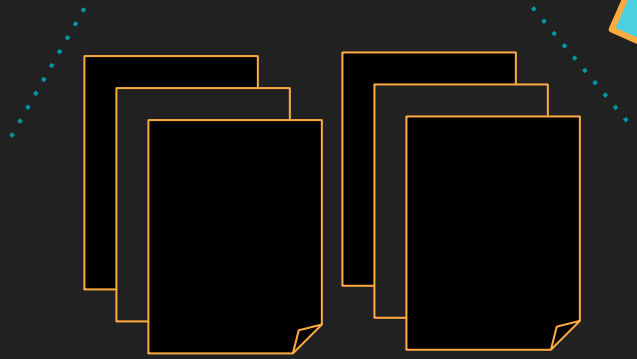
production



Use Case #1 Variants
(dev, staging and production)

kustomization.yaml

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



kustomization.yaml

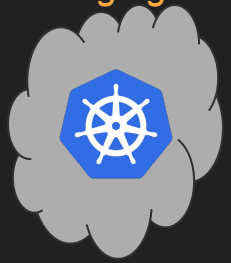
```
namePrefix: dev-
bases:
- ../../base
```

dev

kustomization.yaml

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

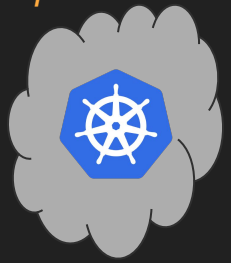
staging



kustomization.yaml

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

production



```
$ kustomize build wordpress/base
```

kustomization.yaml

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

service.yaml

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

deployment.yaml

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

```
$ tree wordpress
wordpress
├── base
│   ├── kustomization.yaml
│   ├── deployment.yaml
│   ├── env.txt
│   └── service.yaml
└── overlays
    ├── production
    │   └── kustomization.yaml
    │   ├── deployment.yaml
    ├── README.md
    └── staging
        ├── kustomization.yaml
        ├── config.env
        └── deployment.yaml
```

```
$ kustomize build wordpress/overlays/production
```

```
$ 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

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

replica_count.yaml

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

cpu_count.yaml

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

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
        └── ...
```

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.)*

Use Case #2 Feeding customized names to containers

kustomization.yaml

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

patch.yaml

```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

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

Use Case #3 Merge properties from multiple sources

base

kustomization.yaml

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

common.properties

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

production overlay

kustomization.yaml

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

secret.properties

```
dbpassword=foo
```

/dev/stdout

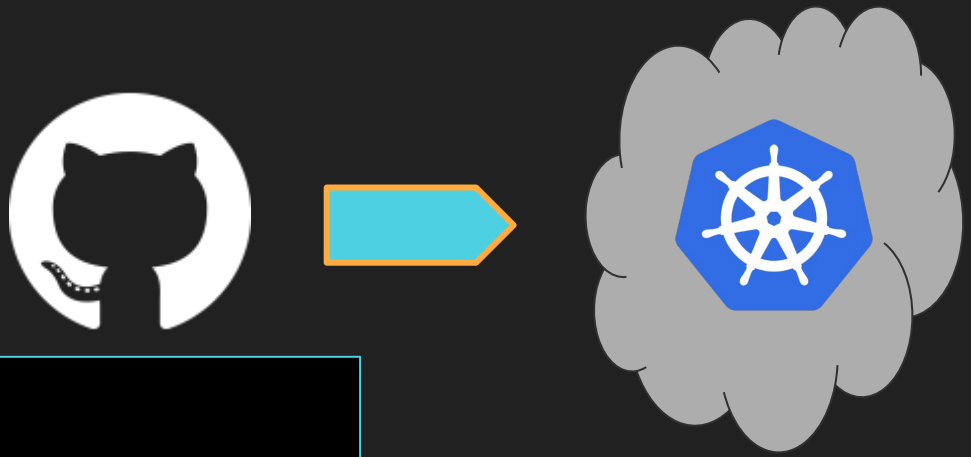
```
kind: ConfigMap
metadata:
  name: prod-myCMap-b5m75cxc
data:
  color=blue
  height=10m
  dbpassword=foo
```

… so on for staging and development variants.

Properties can be owned by different teams.

Use Case #4 Deploy from version control

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



kustomization.yaml

```
...

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

Keeping up with someone else's configuration upgrades ?

Fork it to make a base.

Customize it.

Occasionally git rebase.

A git repo,

with kustomizations describing your variants,

is a simple yet powerful way start using k8s.

```
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
    │   └── ...
    │...
```

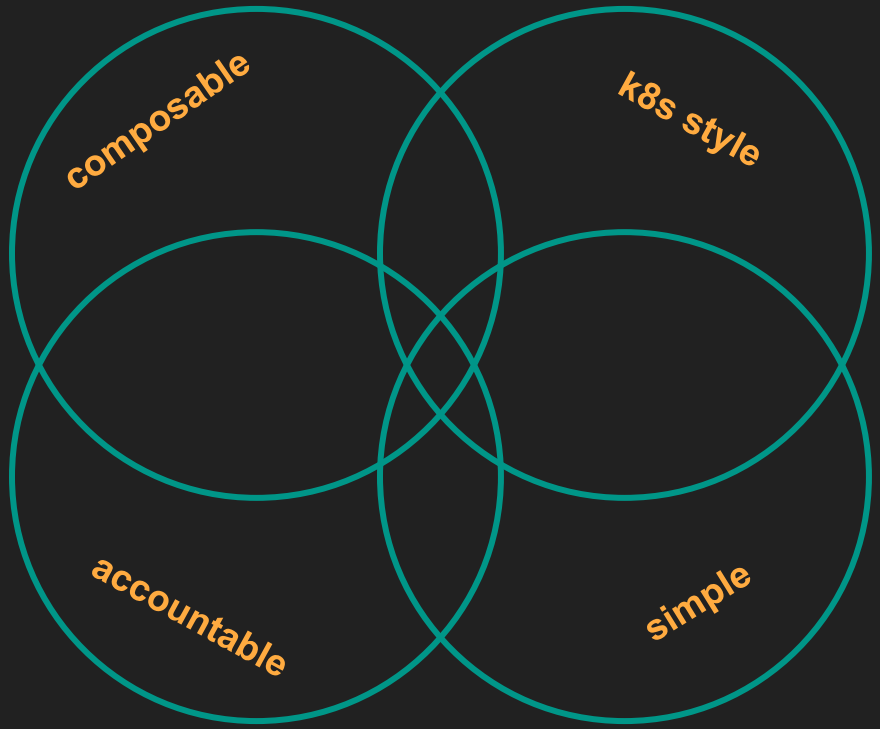
DEMO

No - too boring

try these examples

kustomize design goals

Talk not over yet...
almost!



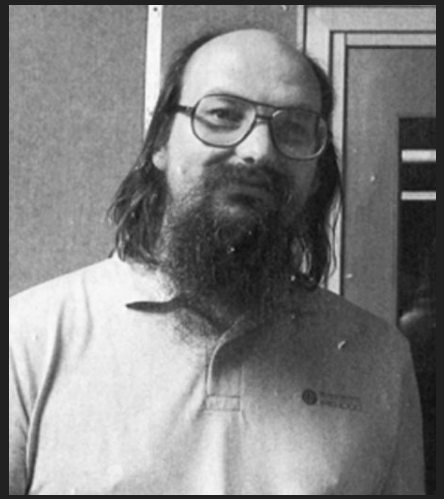
composable

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

recognizable yaml resources

names, labels, etc.

extensible (OpenAPI, CRDs)

existing patch concepts

targets *kubectl apply*

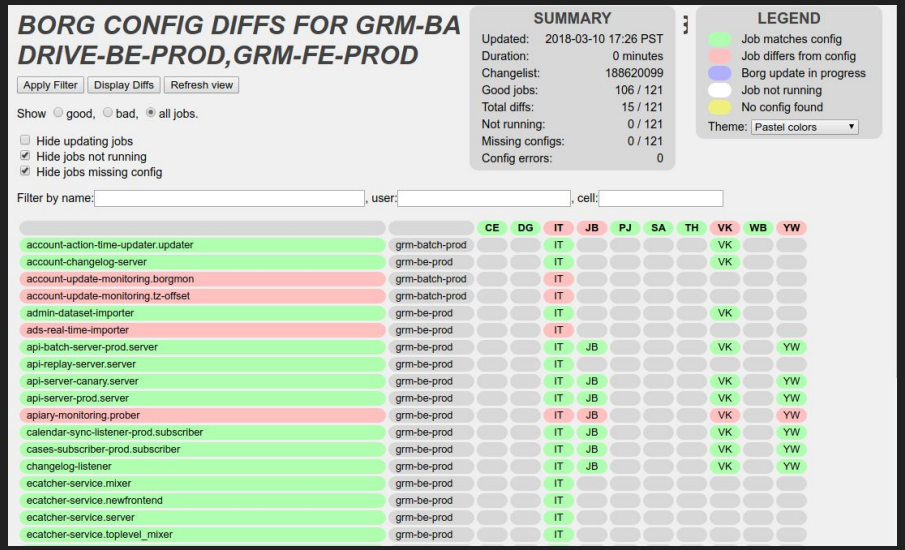
```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # Update the replicas from 2 to 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.8
        ports:
        - containerPort: 80
```

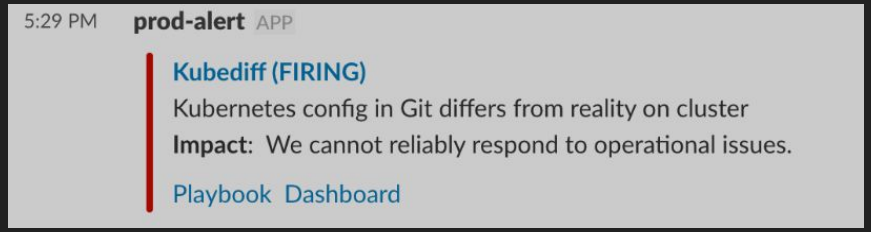
accountable

declarative only

gitops by design

- diff cluster against git
- share bases via git
- fork/rebase flow





weaveworks

simple

- no templating
- no new logic
- no unintentional API
- no forced notion of a *‘package’* or *‘application’*

```
// [1] http://google3/production/borg/templa
all_packages = filter(lambda y: !(defined_e
                                  && y.pe
                 map(lambda x:
                       cond(defined_exp
                            set_cell_f
                       bcl.get_objects


list_of_package_names_map =
    flatten(map(lambda x:
                encode_list_of_strings(

                all_packages),
            [binary_package_map])

packages_map_as_string = encode_list_

pkg_csum_errors = cond(match(package
                       "borgcfg was
                       "for one of


// Note: Do not call this variable
// word.
packages_list_tmp =
    flatten([all_packages],
            cond(defined_expr(bi
                 [binary_package
                 []))


// Clear the fields required b
// have a Borg package type se
// that have not opted-in to
packages_list = map(lambda x:
                    cond(def
                         x.b
                         cle
                         x)
                    package


error string = pkg_csum_er
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "artifactory.fullname" . }}
  labels:
    app: {{ template "artifactory.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    component: "{{ .Values.artifactory.name }}"
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
spec:
  replicas: {{ .Values.artifactory.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "artifactory.name" . }}
        component: "{{ .Values.artifactory.name }}"
        release: {{ .Release.Name }}
    spec:
      {{- if .Values.imagePullSecrets }}
      imagePullSecrets:
      - name: {{ .Values.imagePullSecrets }}
      {{- end }}
      initContainers:
      - name: "remove-lost-found"
        image: "{{ .Values.initContainerImage }}"
        imagePullPolicy: {{ .Values.artifactory.image.pullPolicy }}
        command:
        - 'sh'
        - '-c'
        - 'rm -rfv {{ .Values.artifactory.persistence.mountPath }}/los
        volumeMounts:
        - mountPath: {{ .Values.artifactory.persistence.mountPath | quote }}
          name: artifactory-volume
      - name: "wait-for-db"
```

Forcing tuple merges

It's possible to force conflicting overrides
required. That said, if you *really* need to, th

```
// force_merge.borg

A = {
  x = "{A}"
  y = "{A}"
}

B = {
  y = "{B}"
  z = "{B}"
}

// illegal
C = A + B

// the lambda is called _only_ for
// where it selects the second val
D = mktuple(A.items() + B.items(),
```

We can see the results:

```
$ borgcfg force_merge.borg print C
{
  x = '{A}'
  y = __bad__
  z = '{B}'
}
force_merge.borg:6: error: Mismatc
```

Future

kubectl integration	`$ kubectl apply -f target` also: *well defined pruning*
search engine	like *godoc.org*, but for kustomizations
helm integration	directly kustomize a helm URL

Thanks!

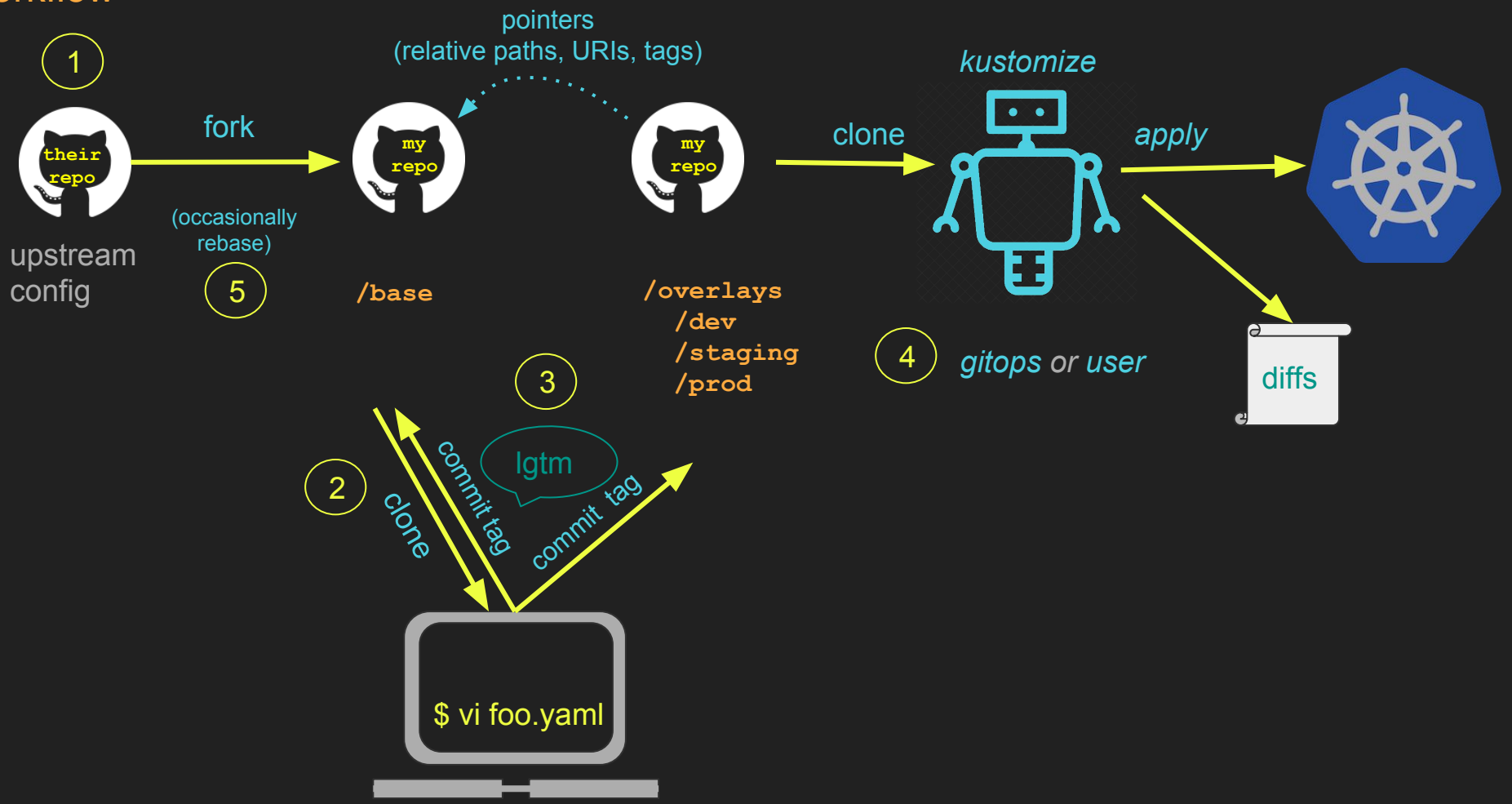
contributors	<- please file issues / help
replicated	on-prem virtuosos, kustomize.io!
heavybit	hosting!
CNCF, sig-cli	cat herding, sponsoring kustomize
brian grant	principal eng @google, k8s founder, author of declarative application management in Kubernetes (inspiration for kustomize)


questions

What would you say you do here?

I find an example.
Then I kustomize it.

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