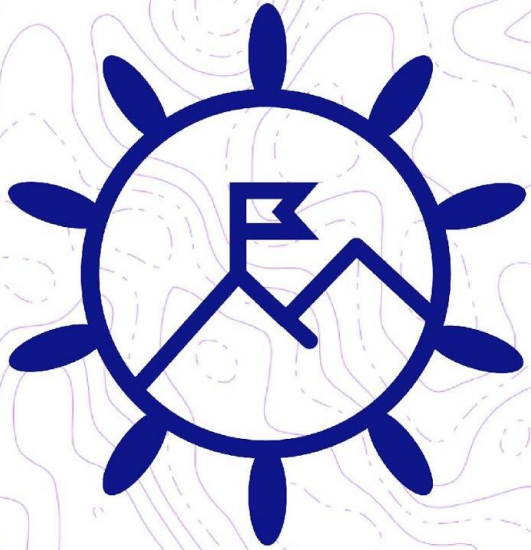


HELM SUMMIT

Day 2 Operations of Helm Charts

Dexter Horthy

@dexhorthy

@dexhorthy

Quietly powering the worlds best enterprise software

Dex
@dexhorthy

Deployed my blog on Kubernetes

Day 2 Operations of Helm Charts

Dexter Horthy

Kubernetes != Configuration as Code

Why This Matters

How you can leverage it

Great for Helm Users

Great for Chart Maintainers

Great for the larger Kubernetes community.

Configuration as Code?

Ways to deploy code

Whose code?

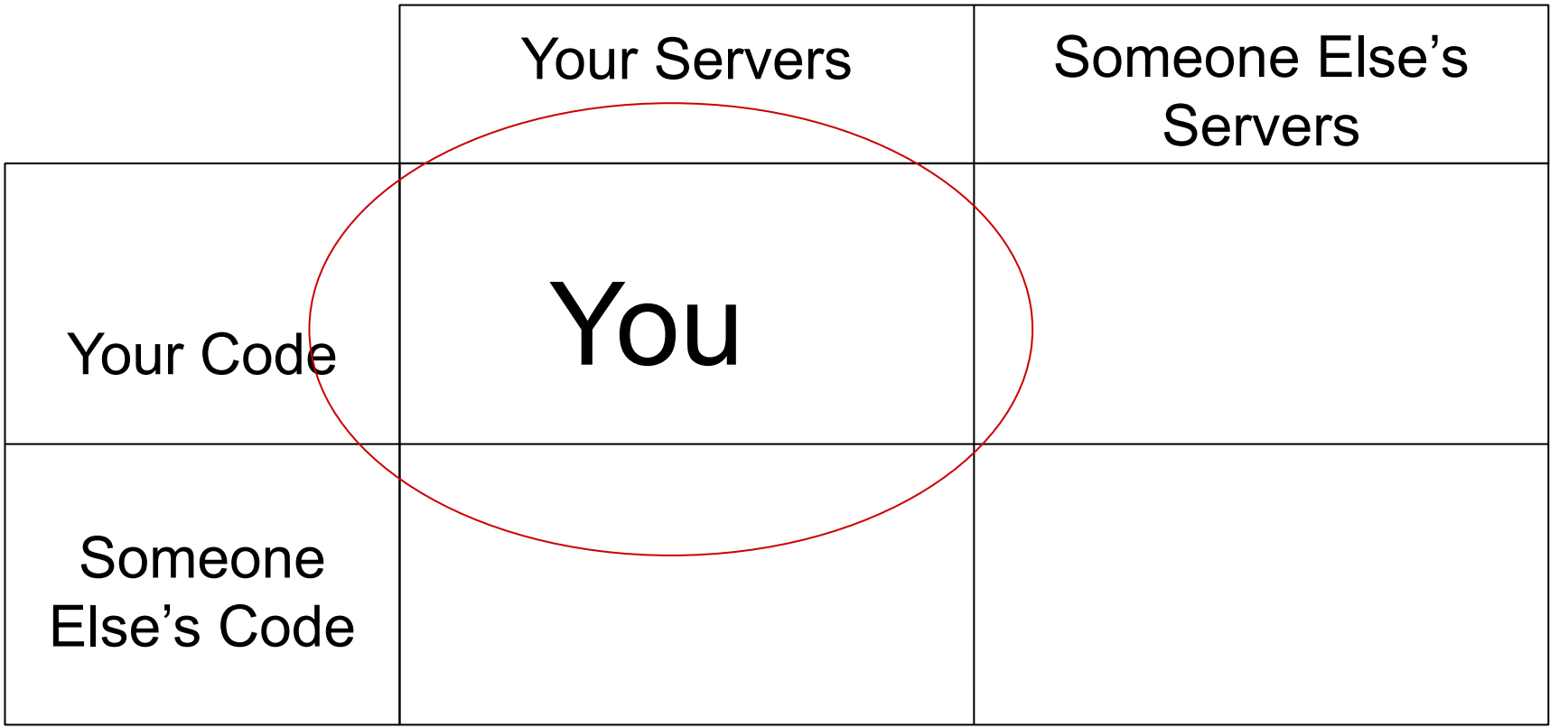
Where?

	Your Servers	Someone Else's Servers
Your Code		
Someone Else's Code		

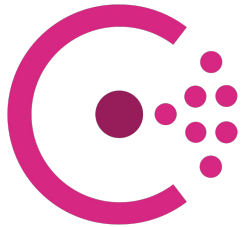
90%

Your code → your servers

	Your Servers	Someone Else's Servers
Your Code		
Someone Else's Code		


Your Servers
Someone Else's Servers
Your Code
You
Someone Else's Code

OTS In Kubernetes



HashiCorp
Consul

OPENFAAS

elastic

anchore

	Your Servers	Someone Else's Servers
Your Code	You	
Someone Else's Code	Also You	

	Your Servers	Someone Else's Servers
Your Code		You?
Someone Else's Code		

	Your Servers	Someone Else's Servers
Your Code		
Someone Else's Code		 Google Cloud   Microsoft Azure

	Your Servers	Someone Else's Servers
Your Code	1st Party	OTS Maintainer
Someone Else's Code	OTS Consumer	

OTS vs First Party

Configuration As Code is great for First Party Deployment

1st Party

- Limited deployment environments
- Direct access to source configuration
- Direct access to application experts

1st Party

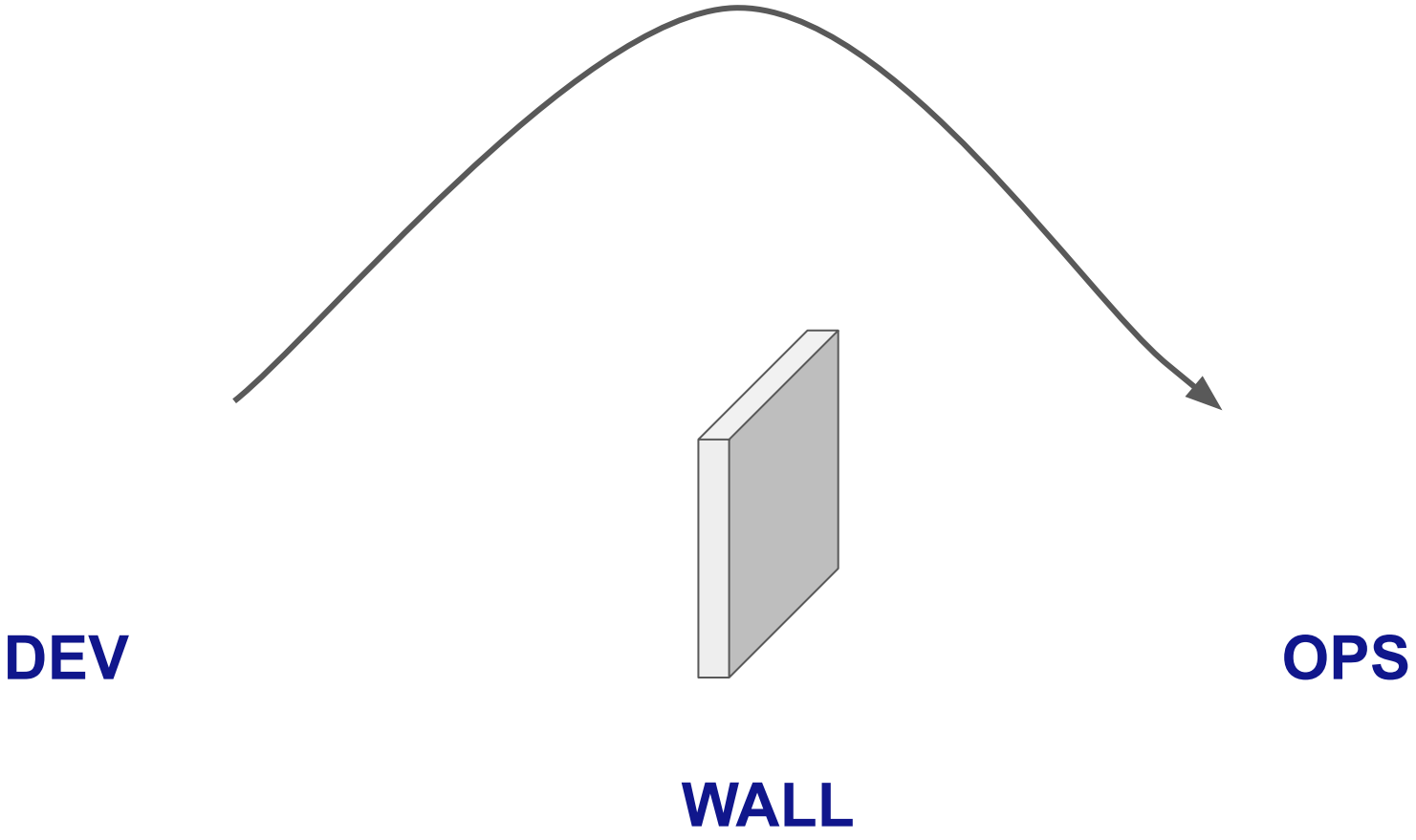
- Easy to change
- Few changes needed

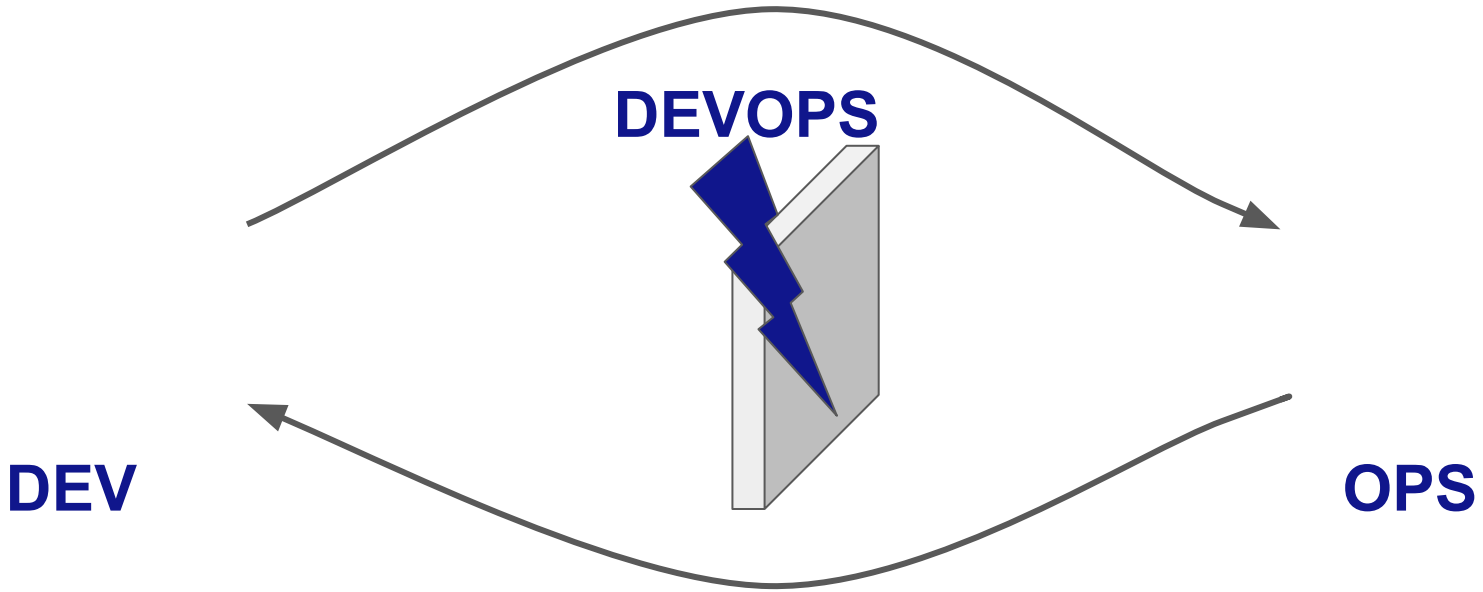
3rd Party

- Infinite Deployment Environments
- Limited control over source configuration

3rd Party

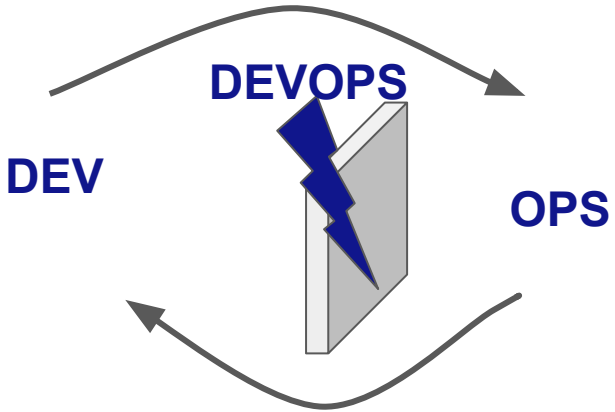
- Many changes needed
- Harder to change

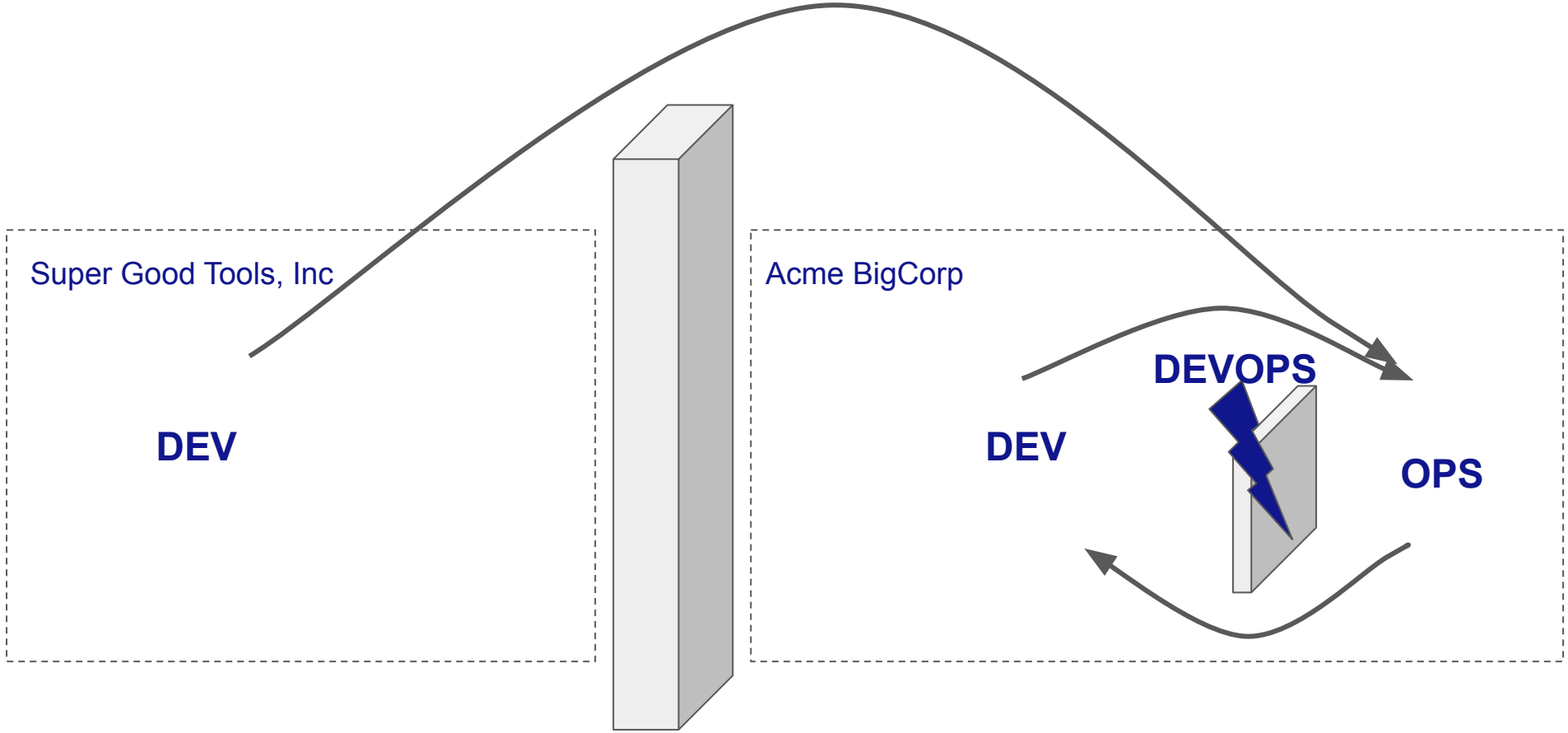

DEV
OPS
WALL


DEVOPS
DEV
OPS

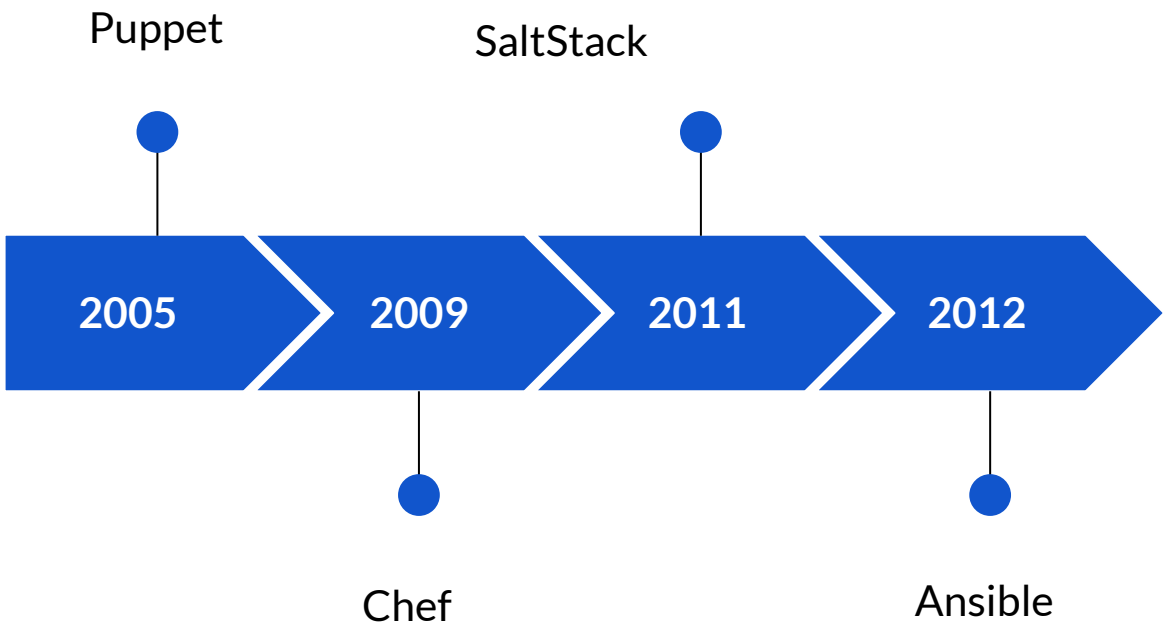
This is harder for OTS

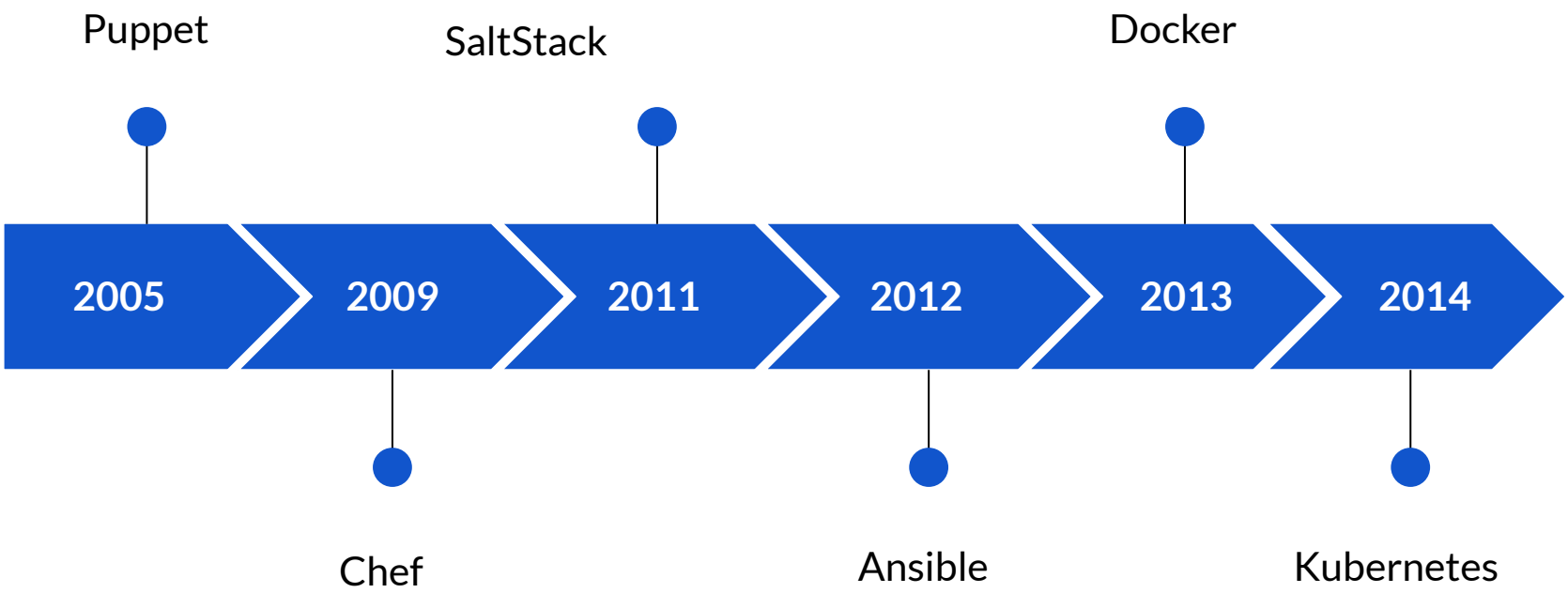
Acme BigCorp




Super Good Tools, Inc
DEV
Acme BigCorp
DEVOPS
DEV
OPS

This pre-dates Kubernetes


Puppet
SaltStack
2005
2009
2011
2012
Chef
Ansible

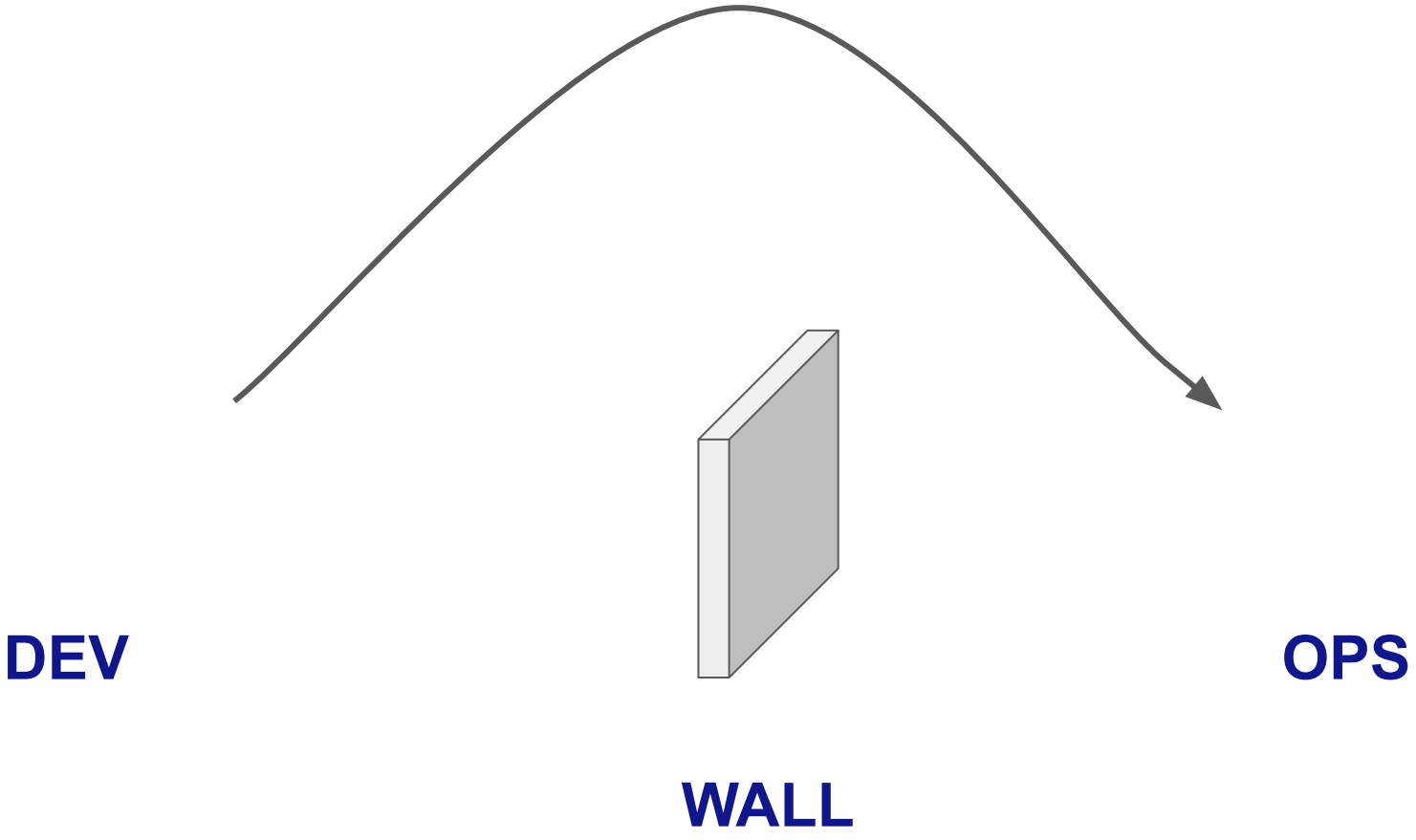

Puppet
SaltStack
Docker
2005
2009
2011
2012
2013
2014
Chef
Ansible
Kubernetes

This pre-dates Kubernetes

But by nature, Kubernetes is equipped to solve this better than its predecessors

What does this problem look like in Kubernetes?

Kubernetes has its own wall


DEV
OPS
WALL

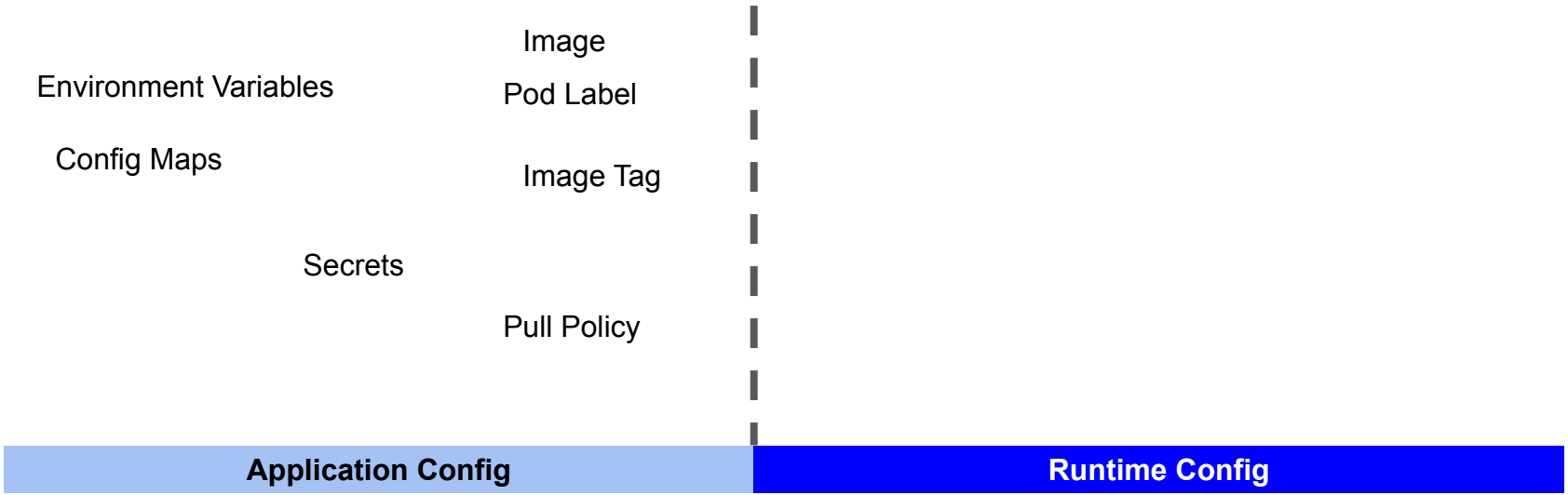
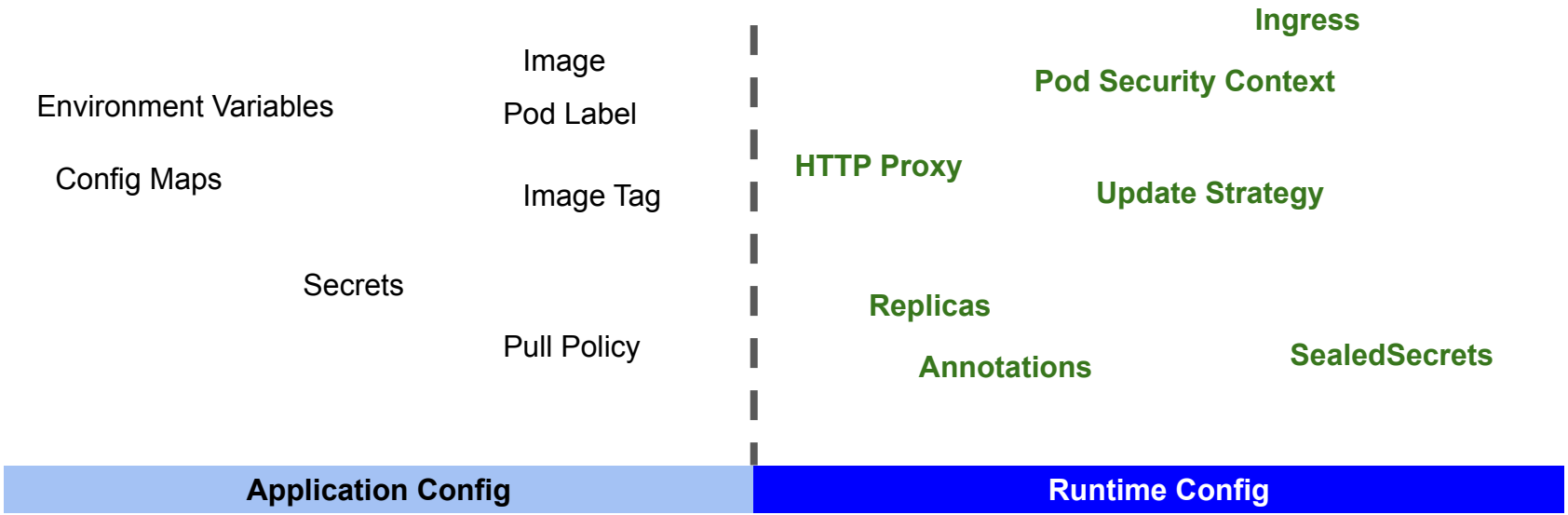
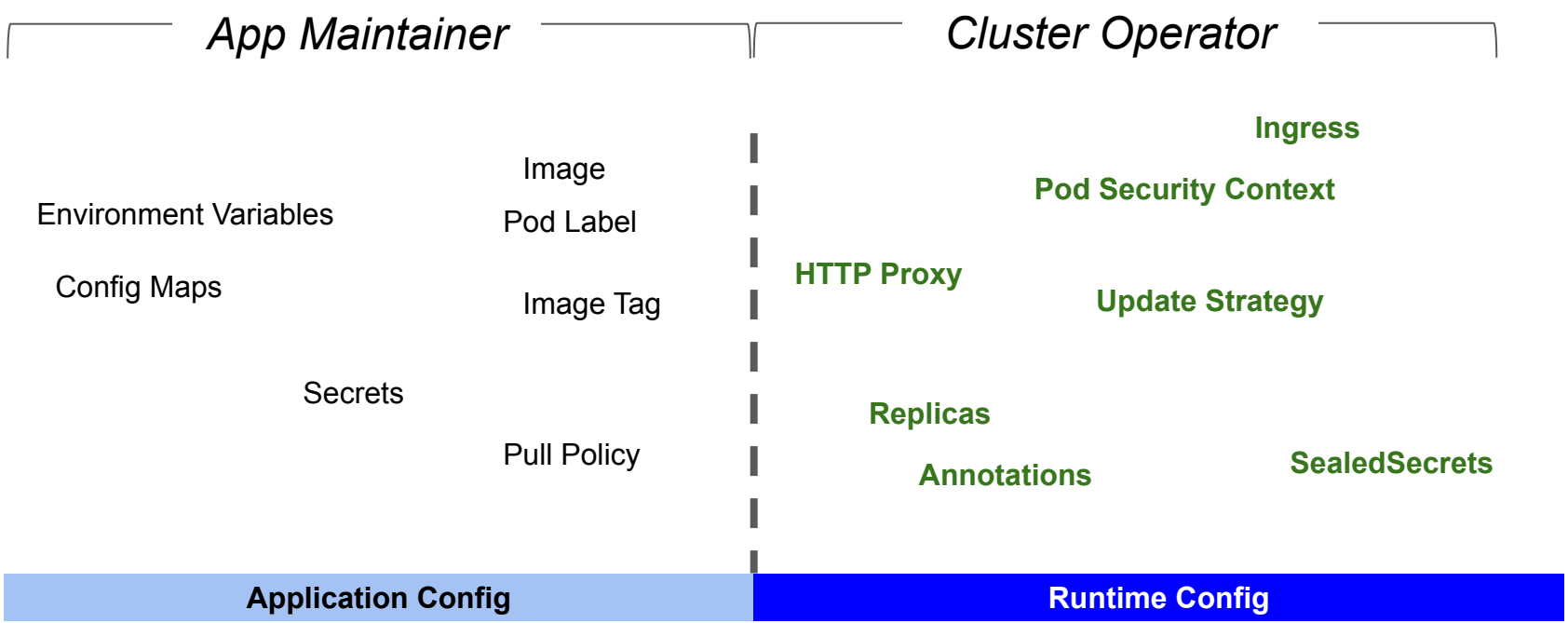


Image
Environment Variables
Pod Label
Config Maps
Image Tag
Secrets
Pull Policy
Application Config
Runtime Config


Ingress
Image
Pod Security Context
Environment Variables
Pod Label
HTTP Proxy
Config Maps
Image Tag
Update Strategy
Secrets
Replicas
Pull Policy
Annotations
SealedSecrets
Application Config
Runtime Config


App Maintainer
Cluster Operator
Ingress
Image
Pod Security Context
Environment Variables
Pod Label
HTTP Proxy
Config Maps
Image Tag
Update Strategy
Secrets
Replicas
Pull Policy
Annotations
SealedSecrets
Application Config
Runtime Config

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: web-app
          image: quay.io/supergoodtool/web-app:1.0.1
          imagePullPolicy: Always
          envFrom:
            secret:
              name: web-app
          env:
            - name: HTTP_PROXY
              value: someproxy.acmecorp.internal:6123
          ports:
            - name: http
              containerPort: 8090
```

Mixed together

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: web-app
          image: quay.io/supergoodtool/web-app:1.0.1
          imagePullPolicy: Always
          envFrom:
            secret:
              name: web-app
          env:
            - name: HTTP_PROXY
              value: someproxy.acmecorp.internal:6123
          ports:
            - name: http
              containerPort: 8090
```

App

Runtime

App Config

- Finite Configurations
- OTS Application knows all possible options
- Easily templatable

Runtime Config

- Infinite configurations
  - + CRDS
  - + Infrastructure
- Independent of OTS application
- Cluster Operator concerns

Application vs Runtime Config in Kubernetes

App Config

- Easily templated
- OTS application knows all possible configurations

Runtime Config

- Unlimited configurations
  - + CRDs
  - + Infrastructure
- Options are independent of OTS application
- Cluster operator should decide how to deliver app to runtime

1st Party

- Limited deployment environments
- Direct access to source configuration
- Direct access to application experts

3rd Party

- Unlimited deployment environments
- Limited Access to source configuration
- Hard to change

Maintaining an OSS chart means accepting lots of changes to add configurability

stable/prometheus

Started on Nov 14, 2016 (3 years ago) - Averaging 1+ commit per week

200 total commits to chart

70% “make X configurable”, 20% “updates”, 10% “helm chores”

Started (from *incubating*) with 415 lines in Values.yaml

Now has 1324 lines in Values.yaml

The chart has 10k+ lines of TEMPLATED yaml

As an operator, 3rd party customization

- is slower
- adds complexity

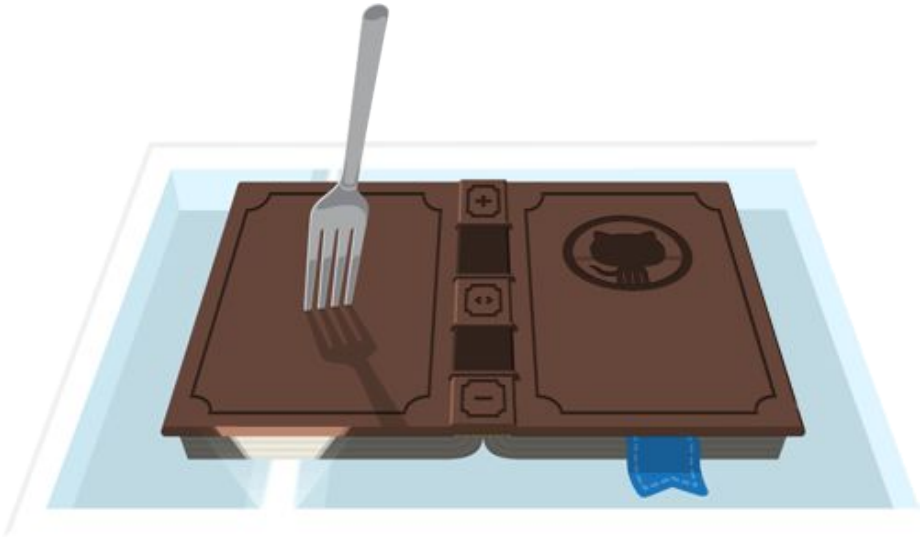
How can I get the speed and simplicity of 1st Party...

...for Charts I don’t own?

Option 1: Fork

```
# values.yaml
color: blue
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: {{ .Values.color }}
```



```yaml
# values.yaml
color: blue
replicas: 1
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: {{ .Values.replicas }}
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: {{ .Values.color }}
```

```
$ helm install --set color=yellow --set replicas=10 ...
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 10
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

Option 2: Template & Modify

```
# values.yaml
color: blue
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: {{ .Values.color }}
```

```
$ helm template --set color=yellow ...
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

```
$ helm template --set color=yellow ...
```

```
 apiVersion: apps/v1
 kind: Deployment
 metadata:
   name: web-app
 spec:
+  replicas: 10
   template:
     spec:
       containers:
         - name: web-app
           image: nginx:1.17
           env:
             - name: COLOR
               value: yellow
```

Neither of these options are great

You get customizability...

...but you become the owner

This makes Day 2 really hard!

Day 2?

Regular Updates / CD

Monitoring

Alerting

Log Aggregation

Canary Deploys

Integration Testing

Security Scanning

...

Deployment

Regular Updates / CD

Integration Testing

Security Scanning

Canary Deploys

Observability

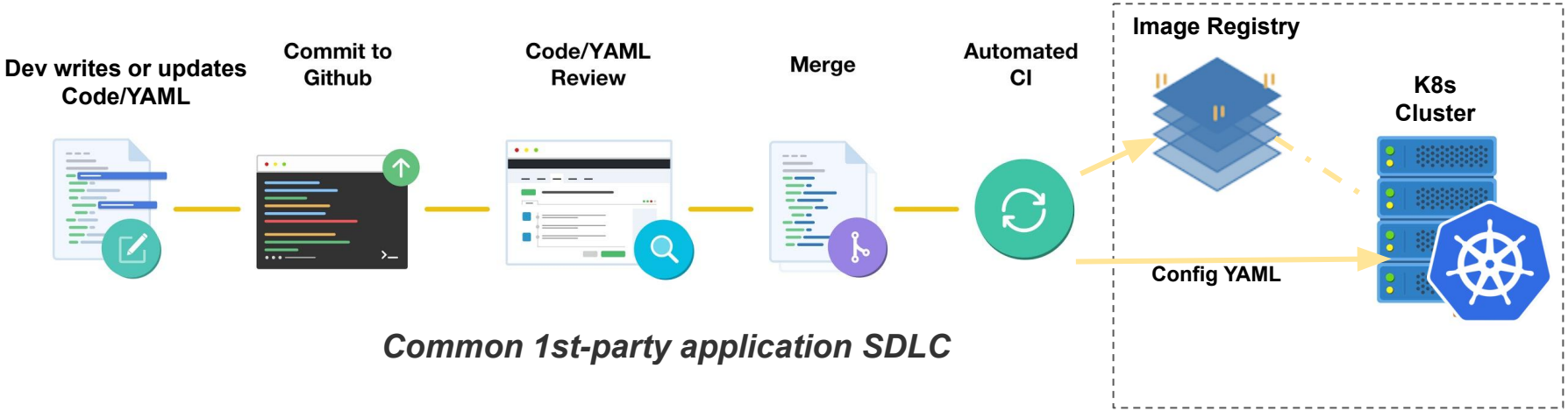
Monitoring

Alerting

Log Aggregation

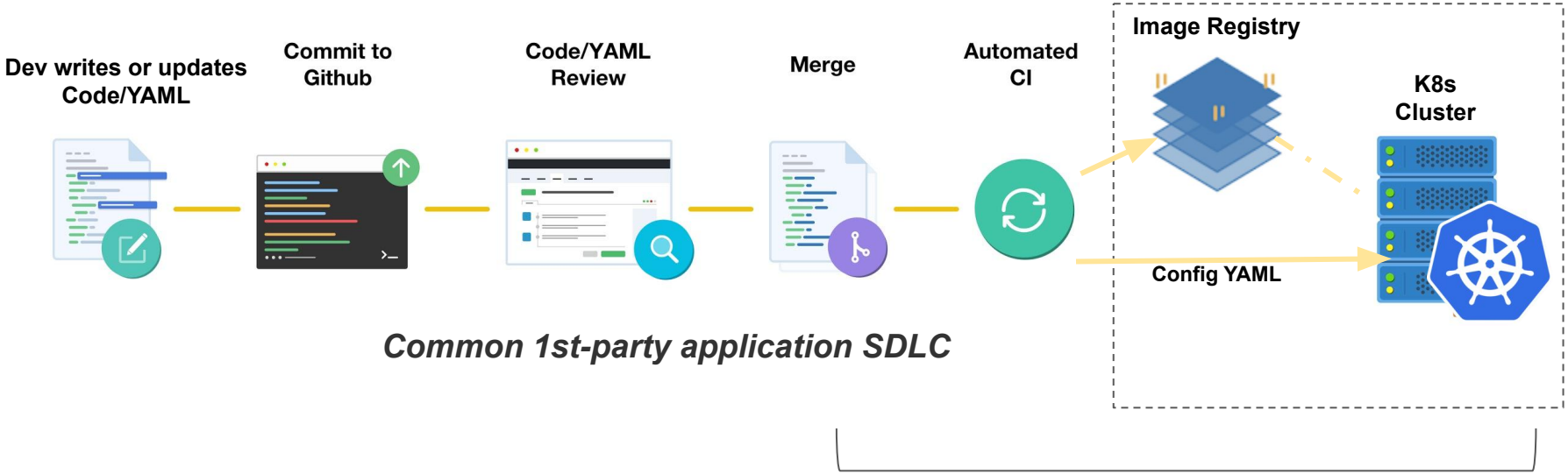
Tracing

Example Deployment Workflows



Common 1st-party application SDLC

Example Deployment Workflows



Common 1st-party application SDLC

Deployment

Regular Updates / CD
Integration Testing
Security Scanning
Canary Deploys

Observability

Monitoring
Alerting
Log Aggregation
Tracing

Day 2

Leverage the same process I use for my own code

To do...

Regular Updates / CD
Monitoring
Alerting
Log Aggregation
Canary Deploys
Integration Testing
Security Scanning

You Need...

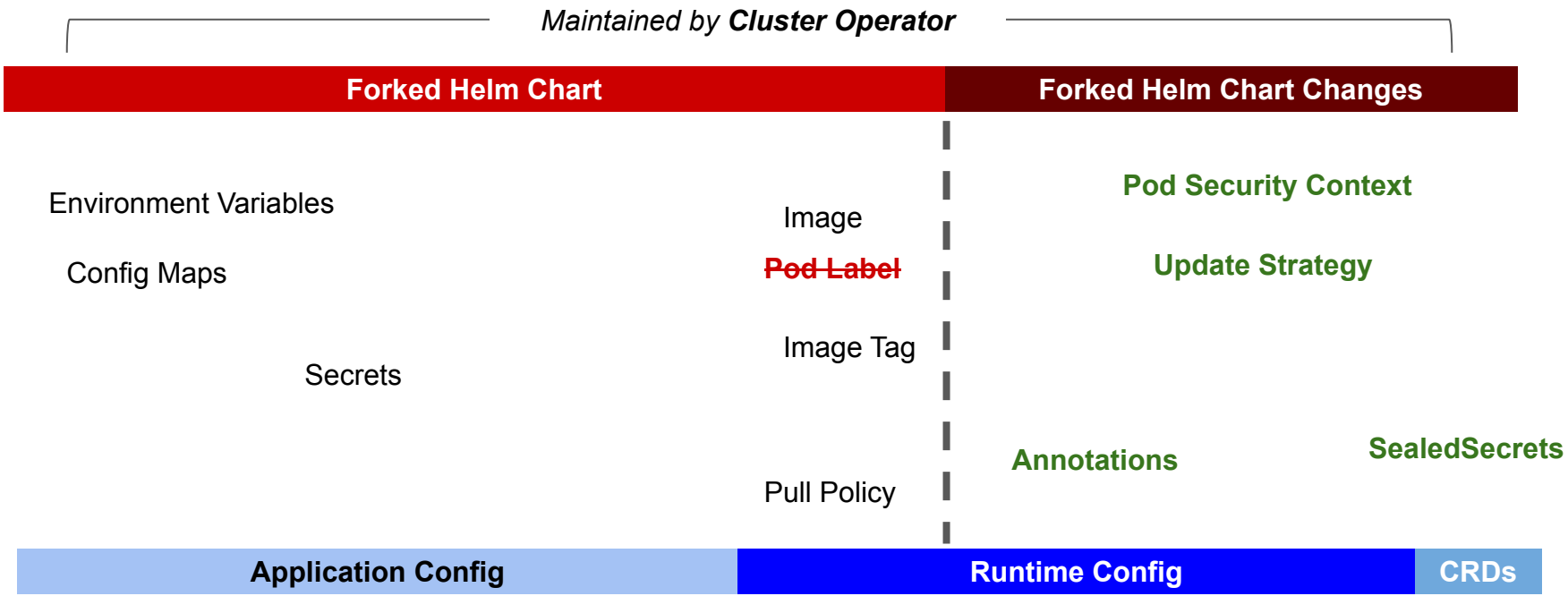
Deployment Flexibility

Deep Customization

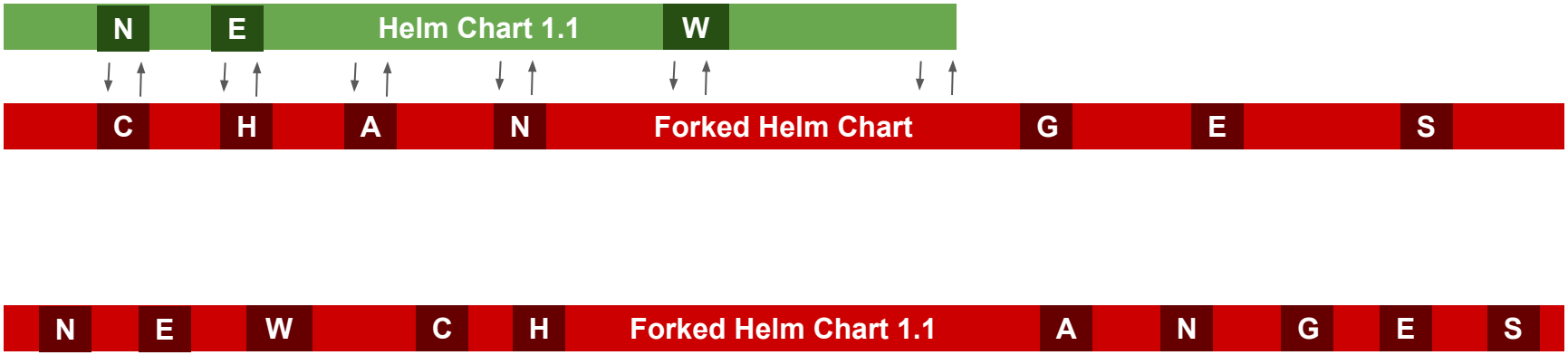
Transparency

Day 2 with forks

Real World Configuration



Forks = Manually reconciling updates



Repeated for 200+ apps at 1,000s of orgs.

Choose your merge conflict!

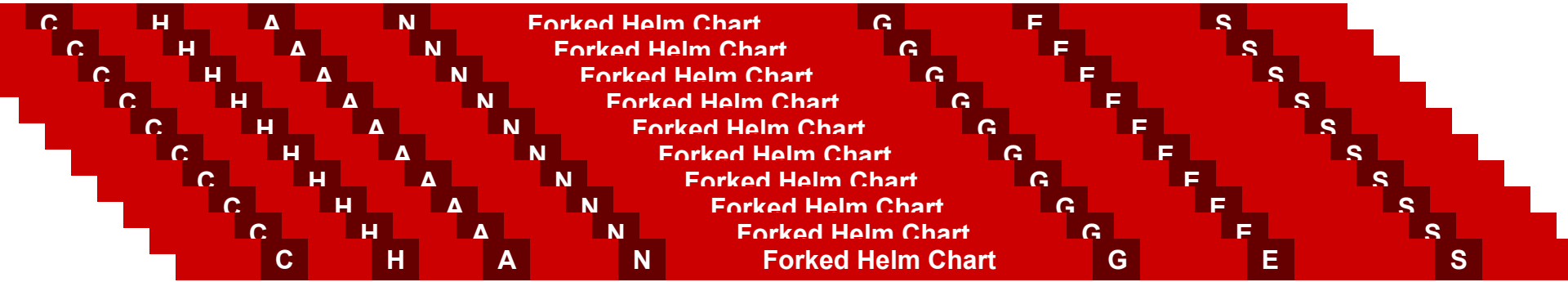
Option 1: Fork Chart

```
   {{- end }}
     labels:
<<<<<<<<<<HEAD
       {{- include "prometheus.server.labels" . | nindent 8 }}
===========
       {{- end }}
       {{- if .Values.server.statefulSet.labels}}
       {{ toYaml .Values.server.statefulSet.labels | nindent 8 }}
>>>>>>>>>>abcea52
       {{- end}}
<<<<<<<<<<HEAD
{{- if .Values.server.affinity }}
   affinity:
===========
```

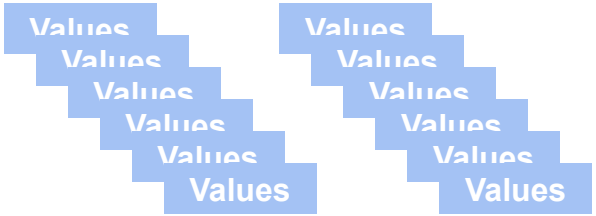
Option 2: Template + Modify

```
     labels:
<<<<<<<<<<HEAD
       app: prometheus
       heritage: Tiller
===========
       app: prometheus
       myapp.io/source: external
       heritage: Tiller
>>>>>>>>>>abcea52
     affinity:
```

What cluster operators manage in reality



Vs. ideally



Manual processes cause problems

“I’ll check on it once in a while”

“If it’s working, don’t fiddle with it”

“We check for updates quarterly”

Updating is important

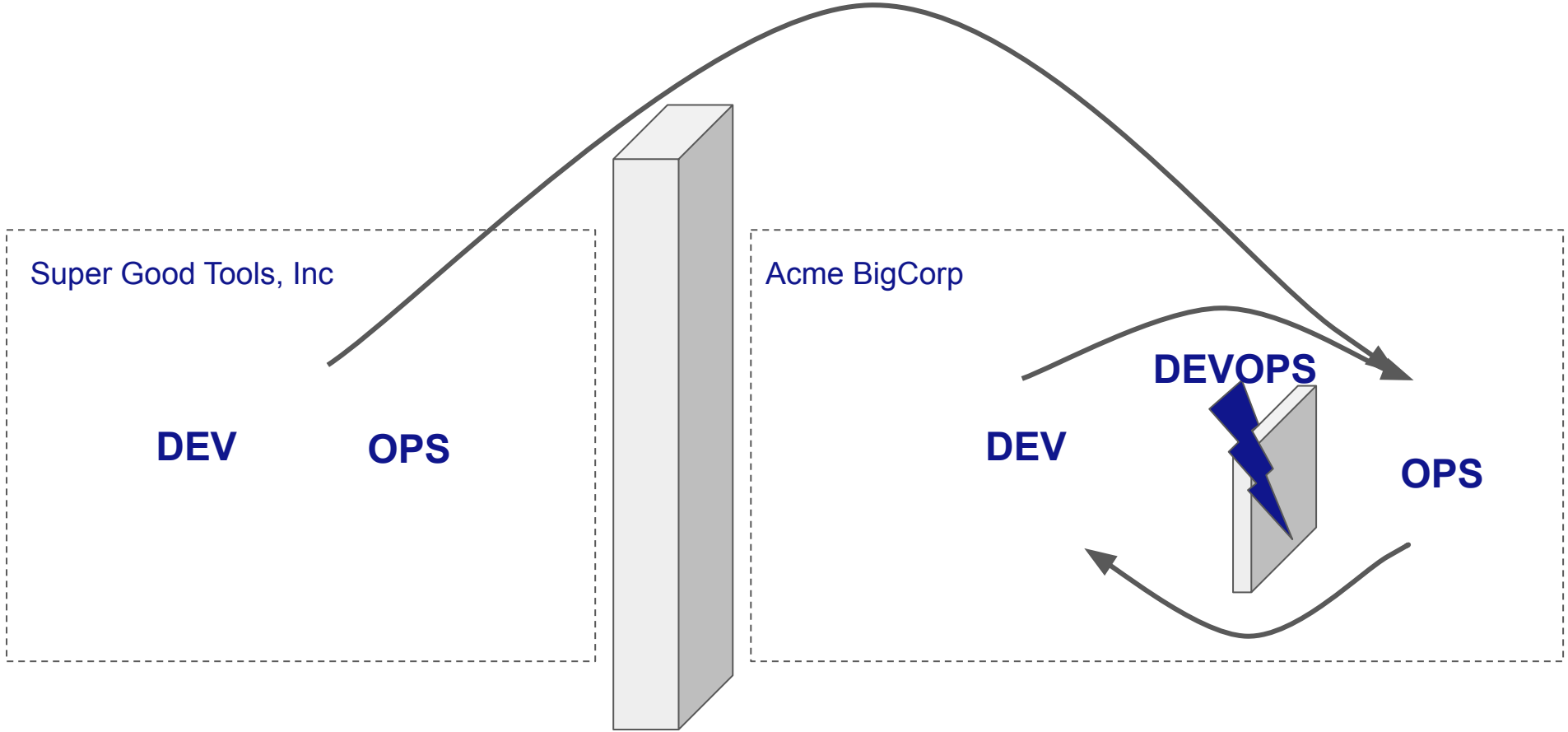
Security

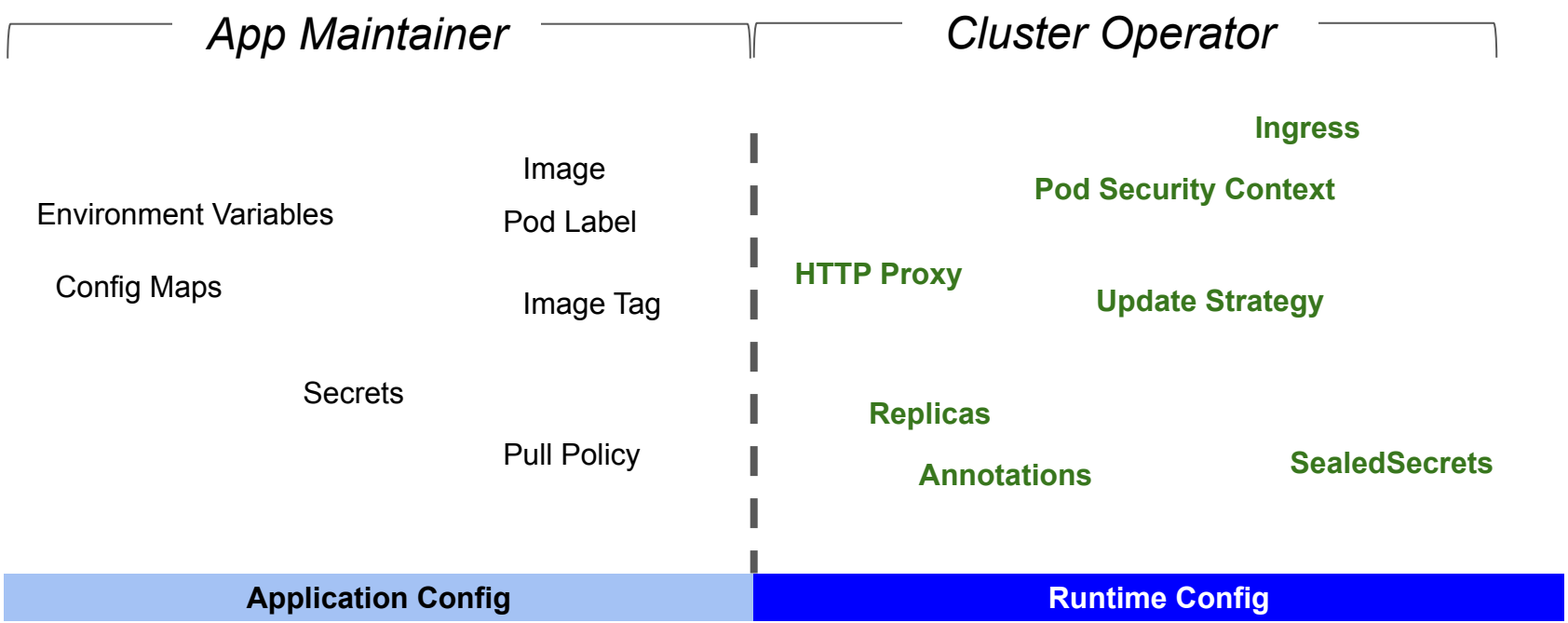
Features

Fixes

How do I get better at this?

Embrace the wall


Super Good Tools, Inc
DEV
OPS
Acme BigCorp
DEVOPS
DEV
OPS


App Maintainer
Cluster Operator
Ingress
Image
Pod Security Context
Environment Variables
Pod Label
HTTP Proxy
Config Maps
Image Tag
Update Strategy
Secrets
Replicas
Pull Policy
Annotations
SealedSecrets
Application Config
Runtime Config

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: web-app
          image: quay.io/supergoodtool/web-app:1.0.1
          imagePullPolicy: Always
          envFrom:
            secret:
              name: web-app
          env:
            - name: HTTP_PROXY
              value: someproxy.acmecorp.internal:6123
          ports:
            - name: http
              containerPort: 8090
```

App

Runtime


HELM
Kustomize.io

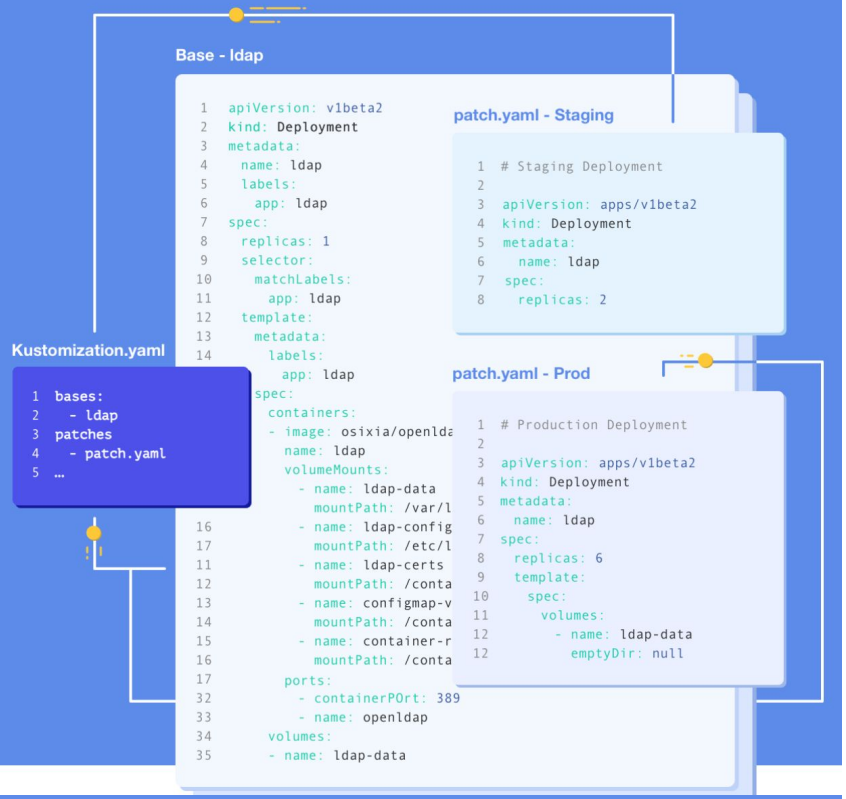
Kustomize.io

Docs Overview Resources Community

Declarative Configuration for Kubernetes

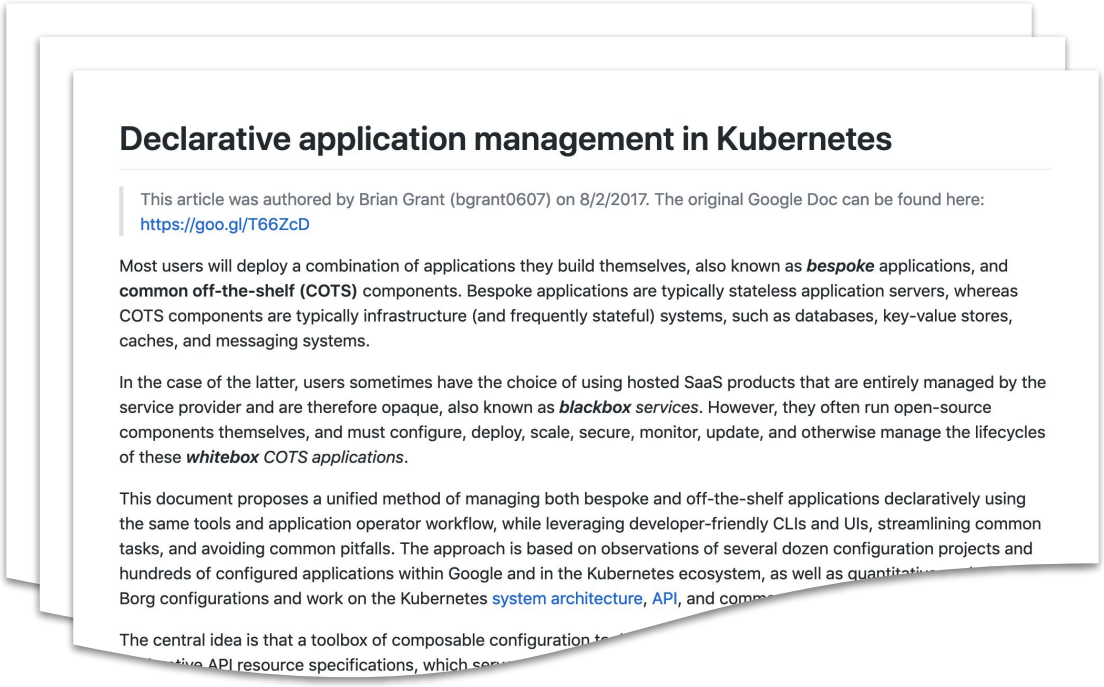
Kustomize introduces a template-free way to customize application configuration that simplifies the use of off-the-shelf applications.

Install View Docs



❤ Website contributed by Replicated

Inspired by Brian Grant's whitepaper



Declarative application management in Kubernetes

> This article was authored by Brian Grant (bgrant0607) on 8/2/2017. The original Google Doc can be found here: https://goo.gl/T66ZcD

Most users will deploy a combination of applications they build themselves, also known as ***bespoke*** applications, and **common off-the-shelf (COTS)** components. Bespoke applications are typically stateless application servers, whereas COTS components are typically infrastructure (and frequently stateful) systems, such as databases, key-value stores, caches, and messaging systems.

In the case of the latter, users sometimes have the choice of using hosted SaaS products that are entirely managed by the service provider and are therefore opaque, also known as ***blackbox** services*. However, they often run open-source components themselves, and must configure, deploy, scale, secure, monitor, update, and otherwise manage the lifecycles of these ***whitebox** COTS applications*.

This document proposes a unified method of managing both bespoke and off-the-shelf applications declaratively using the same tools and application operator workflow, while leveraging developer-friendly CLIs and UIs, streamlining common tasks, and avoiding common pitfalls. The approach is based on observations of several dozen configuration projects and hundreds of configured applications within Google and in the Kubernetes ecosystem, as well as quantitati
Borg configurations and work on the Kubernetes system architecture, API, and comm

The central idea is that a toolbox of composable configuration t
ative API resource specifications, which ser

Template Free Configuration

=

No code!

Chef

```
user 'a user' do
  comment 'A random user'
  uid '1234'
  gid '1234'
  home '/home/random'
  shell '/bin/bash'
  password '$1$JJsvHslasdfjVEroftprNn4JHtDi'
end
```

https://docs.chef.io/resource_user.html

Chef

```
user_home = "/home/#{node['cookbook_name']['user']}"

user node['cookbook_name']['user'] do
  gid node['cookbook_name']['group']
  shell '/bin/bash'
  home user_home
  system true
  action :create
end
```

https://docs.chef.io/resource_user.html

Ansible

```
---
- name: Install nginx
  yum: name=nginx state=present

- name: Copy nginx configuration for wordpress
  template: src=default.conf dest=/etc/nginx/conf.d/default.conf
  notify: restart nginx
```

Ansible

```
- name: here, 'users' contains the above list of employees
  mysql_user:
    name: "{{ item[0] }}"
    priv: "{{ item[1] }}.*:ALL"
    append_privs: yes
    password: "foo"
  with_nested:
    - "{{ users }}"
    - [ 'clientdb', 'employeedb', 'providerdb' ]
```

ダディさま
@leifwalsh

OH: “I Can’t Believe It’s Not YAML™” — @thejunglejane

4:56 PM · May 13, 2019 · Twitter for iPhone

Helm doesn’t quite solve this

Helm prevents accessing the Kubernetes API.

Helm only allows for configuration of items in the chart, not additions or deletions or non-standard changes.

```
20      spec:
21        affinity:
22          podAntiAffinity:
23          {{- if eq .Values.AntiAffinity "hard" }}
24            requiredDuringSchedulingIgnoredDuringExecution:
25            - topologyKey: "kubernetes.io/hostname"
26              labelSelector:
27                matchLabels:
28                  app:  {{ template "memcached.fullname" . }}
29                  release: {{ .Release.Name | quote }}
30          {{- else if eq .Values.AntiAffinity "soft" }}
31            preferredDuringSchedulingIgnoredDuringExecution:
32            - weight: 5
33              podAffinityTerm:
34                topologyKey: "kubernetes.io/hostname"
35                labelSelector:
36                  matchLabels:
37                    app:  {{ template "memcached.fullname" . }}
38                    release: {{ .Release.Name | quote }}
39          {{- end }}
```

How to access the Kubernetes API directly with Helm

```
155                port: metrics
156              initialDelaySeconds: {{ .Values.metrics.readinessProbe.initialDelaySeconds }}
157              timeoutSeconds: {{ .Values.metrics.readinessProbe.timeoutSeconds }}
158            resources:
159  {{ toYaml .Values.metrics.resources | indent 10 }}
160          {{- end }}
161          volumes:
162          {{- if .Values.configurationFiles }}
163          - name: configurations
```

How can Kustomize help?

Patch. Don’t Fork.

Helm Chart

The base is the rendered Helm chart. Ship will create this for you and it’s generated from the original.

Custom Values

The changes you would have made in a fork, i.e. any advanced customization (additions, deletions or changes).

→

Base

The base is the rendered Helm chart. Ship will create this for you and it’s generated from the helm chart.

Patches

The changes you would have made in a fork, i.e. any advanced customization (additions, deletions.

=

Deployable App

Patch. Don’t Fork.

Helm Chart

The base is the rendered Helm chart. Ship will create this for you and it’s generated from the original.

Custom Values

The changes you would have made in a fork, i.e. any advanced customization (additions, deletions or changes).

→

Base

The base is the rendered Helm chart. Ship will create this for you and it’s generated from the helm chart.

Patches

The changes you would have made in a fork, i.e. any advanced customization (additions, deletions.

=

Deployable App

Application Config | **Runtime Config**

```
# values.yaml
color: blue
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: {{ .Values.color }}
```

Application Config

```
$ helm template --set color=yellow ...
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

Application Config

Templated

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

Application Config

Patch

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 10
```

Runtime Config

```
$ kustomize build ./
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 10
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

Runtime Config

Application Config

```
$ kubectl kustomize ./
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 10
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

Runtime Config

Application Config

Templated

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

Application Config

Patch

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 10
  template:
    spec:
      containers:
        - name: web-app
          imagePullPolicy: Always
```

Runtime Config

```
$ kubectl kustomize ./
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 10
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          imagePullPolicy: Always
          env:
            - name: COLOR
              value: yellow
```

Runtime Config

Runtime Config

Application Config

App & Runtime config mixed in a single object...

...but stored separately

Base

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

Application Config

Overlay

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 10
  template:
    spec:
      containers:
        - name: web-app
          imagePullPolicy: Always
```

Runtime Config

Base

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  template:
    spec:
      containers:
        - name: web-app
          image: nginx:1.17
          env:
            - name: COLOR
              value: yellow
```

overlays/staging

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  template:
```

overlays/production

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 10
  template:
```

What does this have to do with Day 2 operations?

Declarative application management in Kubernetes

> This article was authored by Brian Grant (bgrant0607) on 8/2/2017. The original Google Doc can be found here: https://goo.gl/T66ZcD

Most users will deploy a combination of applications they build themselves, also known as ***bespoke*** applications, and **common off-the-shelf (COTS)** components. Bespoke applications are typically stateless application servers, whereas COTS components are typically infrastructure (and frequently stateful) systems, such as databases, key-value stores, caches, and messaging systems.

In the case of the latter, users sometimes have the choice of using hosted SaaS products that are entirely managed by the service provider and are therefore opaque, also known as ***blackbox*** *services*. However, they often run open-source components themselves, and must configure, deploy, scale, secure, monitor, update, and otherwise manage the lifecycles of these ***whitebox*** *COTS applications*.

This document proposes a unified method of managing both bespoke and off-the-shelf applications declaratively using the same tools and application operator workflow, while leveraging developer-friendly CLIs and UIs, streamlining common tasks, and avoiding common pitfalls. The approach is based on observations of several dozen configuration projects and hundreds of configured applications within Google and in the Kubernetes ecosystem, as well as quantitativ
Borg configurations and work on the Kubernetes system architecture, API, and comm

The central idea is that a toolbox of composable configuration t
ative API resource specifications, which ser

“The user can periodically rebase their base”

This is a git merge conflict-free process.

No Templates, No Code, No Control Flow

Structured Object Merge

Still a merge, still room for conflicts...

...but in 90% of cases you can pull upstream with no effort.

```
   {{- end }}
     labels:
<<<<<<<<<<HEAD
       {{- include "prometheus.server.labels" . | nindent 8 }}
===========
       {{- end }}
       {{- if .Values.server.statefulSet.labels}}
       {{ toYaml .Values.server.statefulSet.labels | nindent 8 }}
>>>>>>>>>>abcea52
       {{- end}}
<<<<<<<<<<HEAD
{{- if .Values.server.affinity }}
   affinity:
===========
   spec:
>>>>>>>>>>abcea52
```

Kustomize emits plain YAML

Grokkable Diffs!

```
320 -         image: k8s.gcr.io/kube-state-metrics:v1.5.0
321           imagePullPolicy: IfNotPresent
322           name: kube-state-metrics
323           ports:
    @@ -329,6 +361,7 @@ spec:
329             initialDelaySeconds: 5
330             timeoutSeconds: 5
331           resources: null
332         securityContext:
333           fsGroup: 65534
334           runAsUser: 65534
    @@ -362,14 +395,14 @@ spec:
362           - name: DD_LOG_LEVEL
363             value: INFO
364           - name: KUBERNETES
365 -           value: "yes"
366           - name: DD_KUBERNETES_KUBELET_HOST
367             valueFrom:
368               fieldRef:
369                 fieldPath: status.hostIP
370           - name: DD_HEALTH_PORT
371             value: "5555"
372 -         image: datadog/agent:6.10.1
373           imagePullPolicy: IfNotPresent
```

```
352 +         image: quay.io/coreos/kube-state-metrics:v1.6.0
353           imagePullPolicy: IfNotPresent
354           name: kube-state-metrics
355           ports:
361             initialDelaySeconds: 5
362             timeoutSeconds: 5
363           resources: null
364 +       hostNetwork: false
365         securityContext:
366           fsGroup: 65534
367           runAsUser: 65534
395           - name: DD_LOG_LEVEL
396             value: INFO
397           - name: KUBERNETES
398 +           value: true
399           - name: DD_KUBERNETES_KUBELET_HOST
400             valueFrom:
401               fieldRef:
402                 fieldPath: status.hostIP
403           - name: DD_HEALTH_PORT
404             value: "5555"
405 +         image: datadog/agent:6.13.0
406           imagePullPolicy: IfNotPresent
```

Patch. Don’t Fork.

Helm Chart

The base is the rendered Helm chart. Ship will create this for you and it’s generated from the original.

Custom Values

The changes you would have made in a fork, i.e. any advanced customization (additions, deletions or changes).

→

Base

The base is the rendered Helm chart. Ship will create this for you and it’s generated from the helm chart.

Patches

The changes you would have made in a fork, i.e. any advanced customization (additions, deletions.

=

Deployable App

Easy to customize, easy to update

How do I operationalize this?

Introducing `kots`

kots

- “Kubernetes Off the Shelf (Software)”
- Successor to Replicated Ship
- Optimized for Helm Charts
- No more forking charts
- Streamlines advanced configuration with Kustomize
- Pluggable into existing workflows
- GitOps release management
- Automates merging upstream changes

github.com/replicatedhq/kots

Patch. Don’t Fork.

Helm Chart

The base is the rendered Helm chart. Ship will create this for you and it’s generated from the original.

Custom Values

The changes you would have made in a fork, i.e. any advanced customization (additions, deletions or changes).

→

Base

The base is the rendered Helm chart. Ship will create this for you and it’s generated from the helm chart.

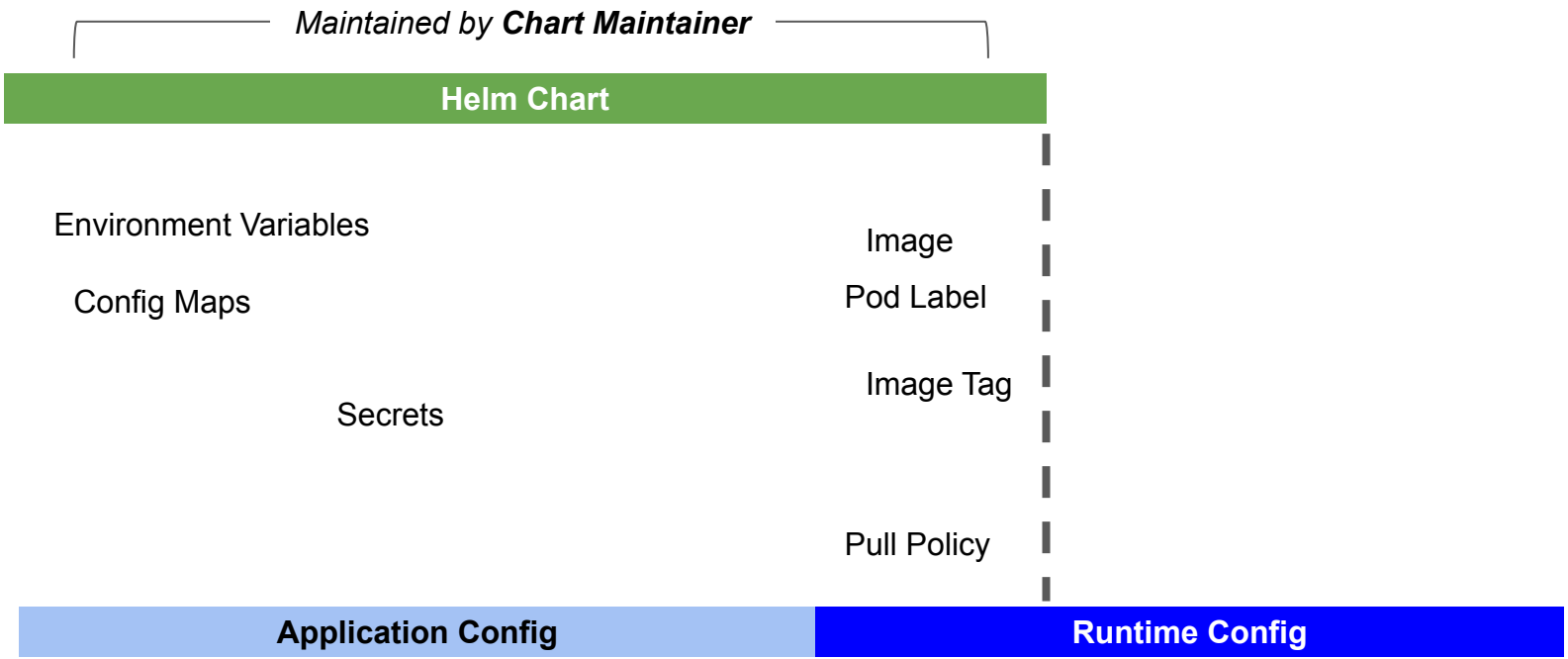
Patches

The changes you would have made in a fork, i.e. any advanced customization (additions, deletions.

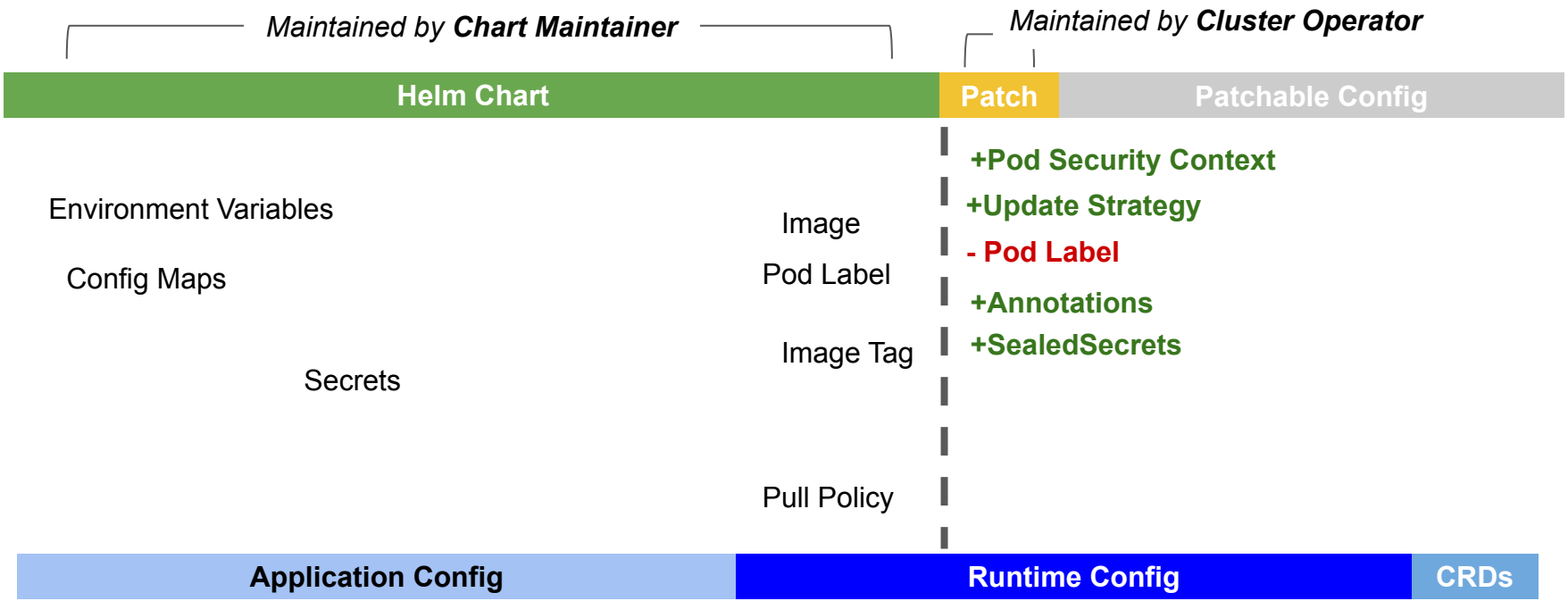
=

Deployable App

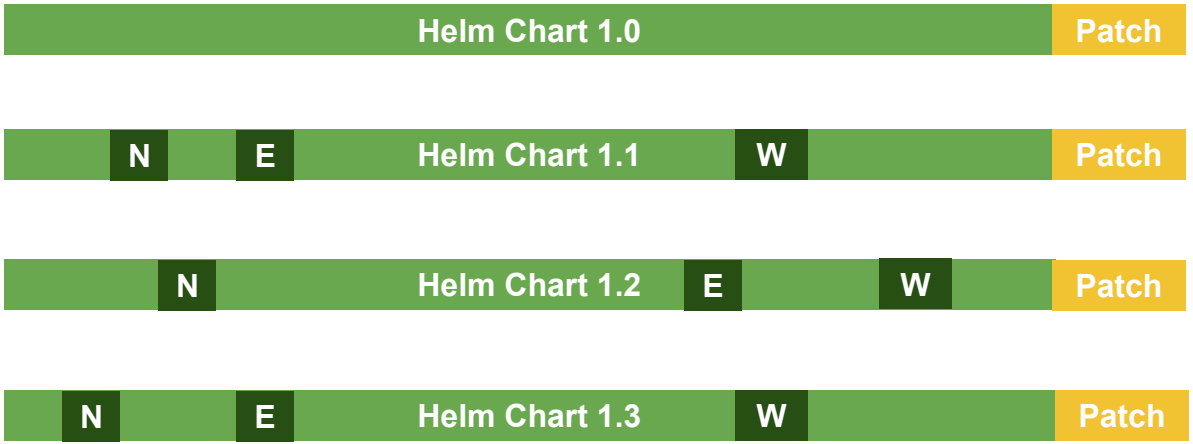
Advanced Configuration with kots



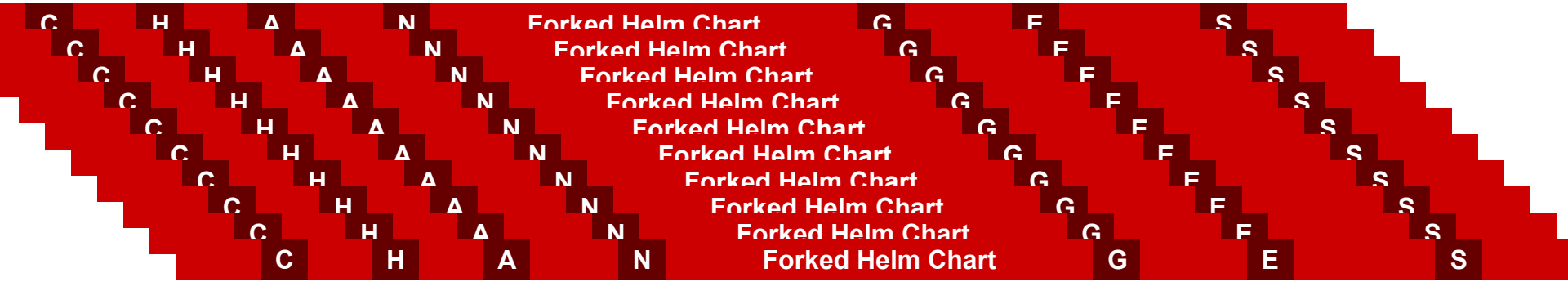
Advanced Configuration with kots



kots = Automatically reconciling updates



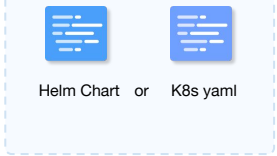
Reality with Helm forks



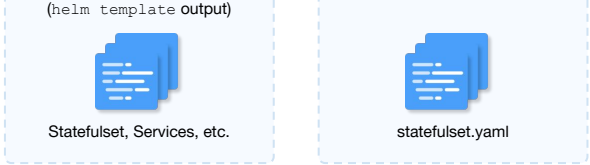
Vs. reality with patches

Midstreams vs. Downstream

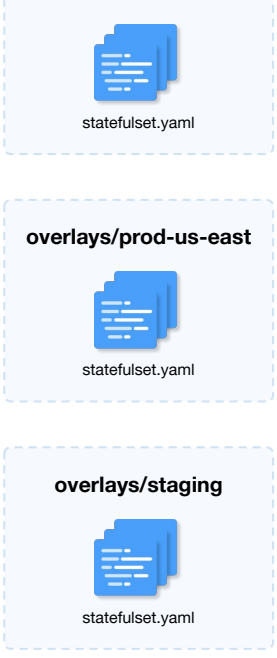
Upstream
An upstream is an OTS application managed by another party



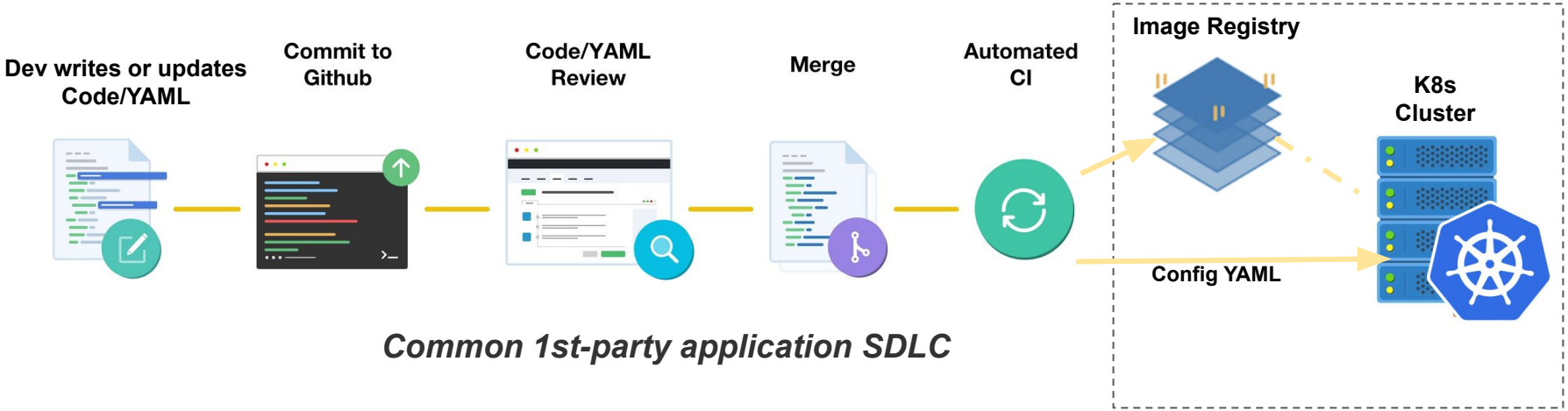
Midstream
A midstream is the organization-approved version of the application, built from the operator-supplied helm values.yaml and top-level kustomize patches



Downstreams
Downstreams capture last-mile changes to be made in specific environments

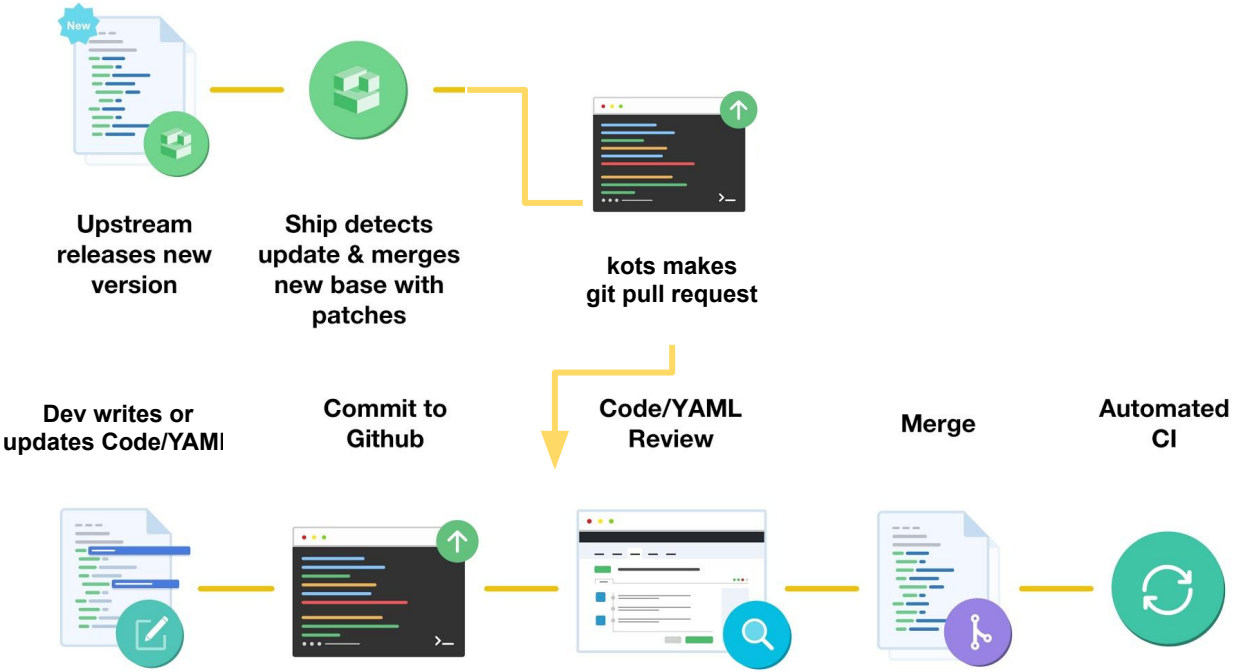


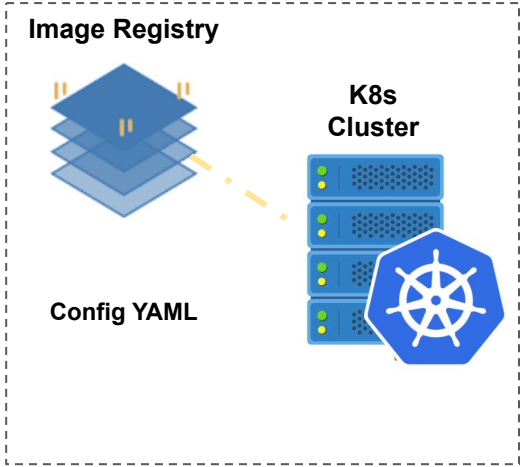
Example Deployment Workflows



Common 1st-party application SDLC

Example Deployment Workflows *(OSS / COTS with kots)*



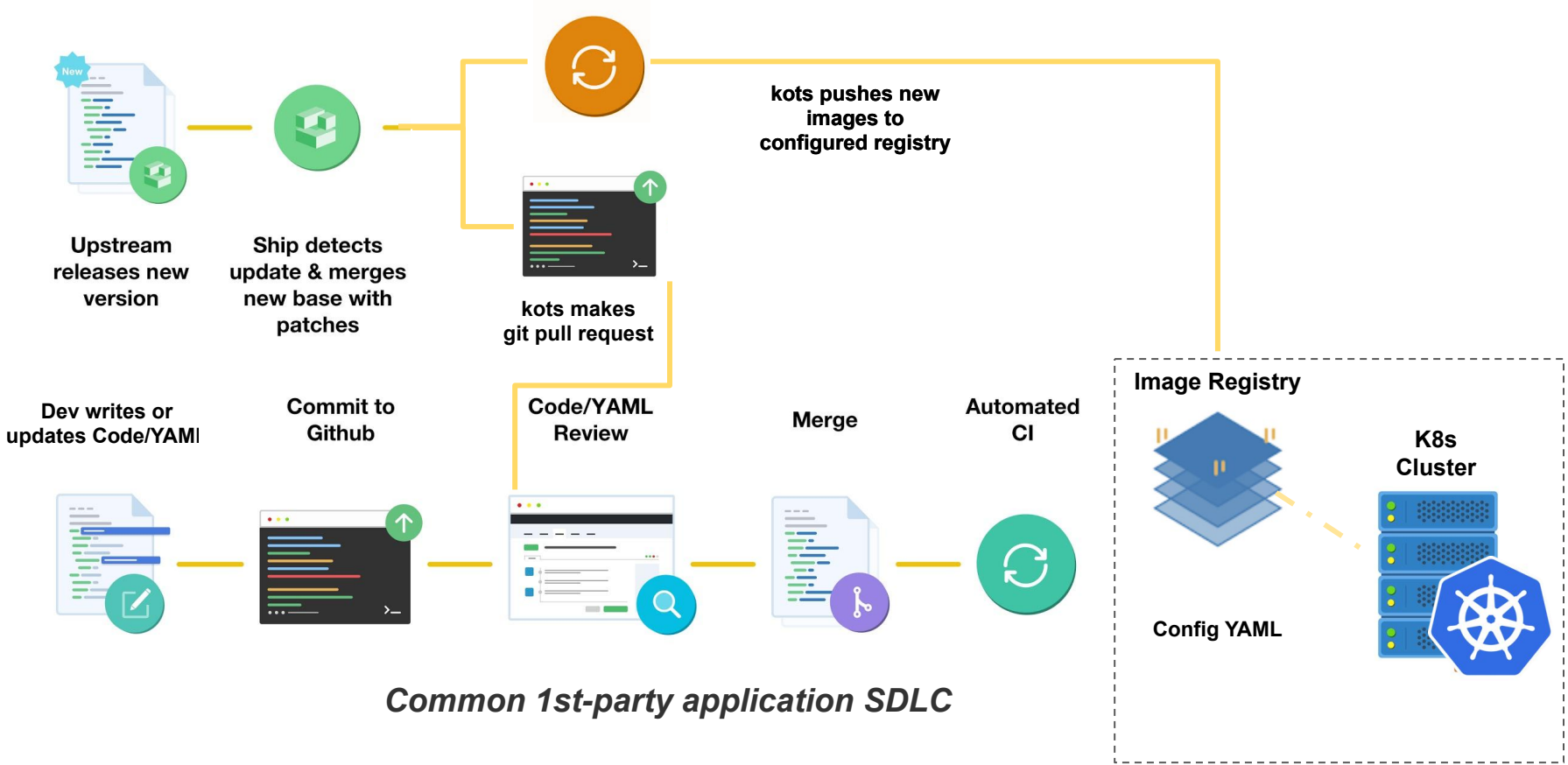


Common 1st-party application SDLC

```
320 -         image: k8s.gcr.io/kube-state-metrics:v1.5.0
321           imagePullPolicy: IfNotPresent
322           name: kube-state-metrics
323           ports:
    @@ -329,6 +361,7 @@ spec:
329             initialDelaySeconds: 5
330             timeoutSeconds: 5
331           resources: null
332         securityContext:
333           fsGroup: 65534
334           runAsUser: 65534
    @@ -362,14 +395,14 @@ spec:
362           - name: DD_LOG_LEVEL
363             value: INFO
364           - name: KUBERNETES
365 -           value: "yes"
366           - name: DD_KUBERNETES_KUBELET_HOST
367             valueFrom:
368               fieldRef:
369                 fieldPath: status.hostIP
370           - name: DD_HEALTH_PORT
371             value: "5555"
372 -         image: datadog/agent:6.10.1
373           imagePullPolicy: IfNotPresent
```

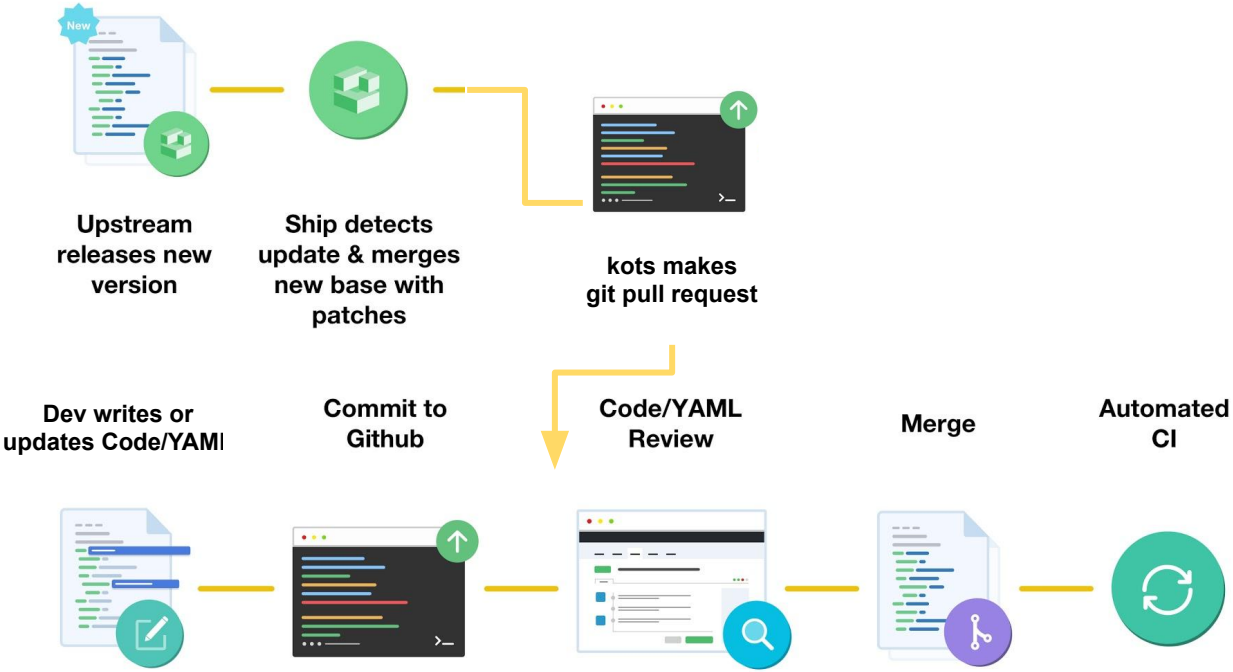
```
352 +         image: quay.io/coreos/kube-state-metrics:v1.6.0
353           imagePullPolicy: IfNotPresent
354           name: kube-state-metrics
355           ports:
361             initialDelaySeconds: 5
362             timeoutSeconds: 5
363           resources: null
364 +       hostNetwork: false
365         securityContext:
366           fsGroup: 65534
367           runAsUser: 65534
395           - name: DD_LOG_LEVEL
396             value: INFO
397           - name: KUBERNETES
398 +           value: true
399           - name: DD_KUBERNETES_KUBELET_HOST
400             valueFrom:
401               fieldRef:
402                 fieldPath: status.hostIP
403           - name: DD_HEALTH_PORT
404             value: "5555"
405 +         image: datadog/agent:6.13.0
406           imagePullPolicy: IfNotPresent
```

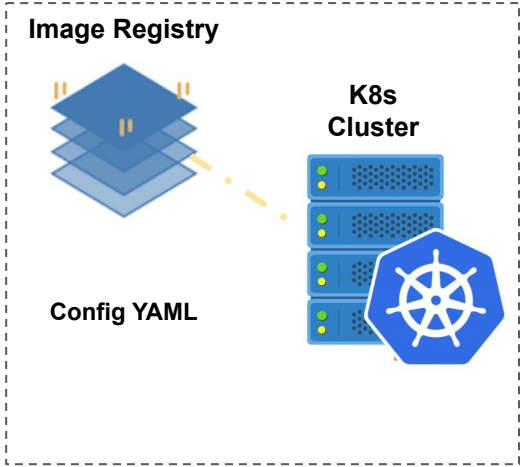
Example Deployment Workflows *(OSS / COTS with kots)*



Common 1st-party application SDLC

Example Deployment Workflows *(OSS / COTS with kots)*



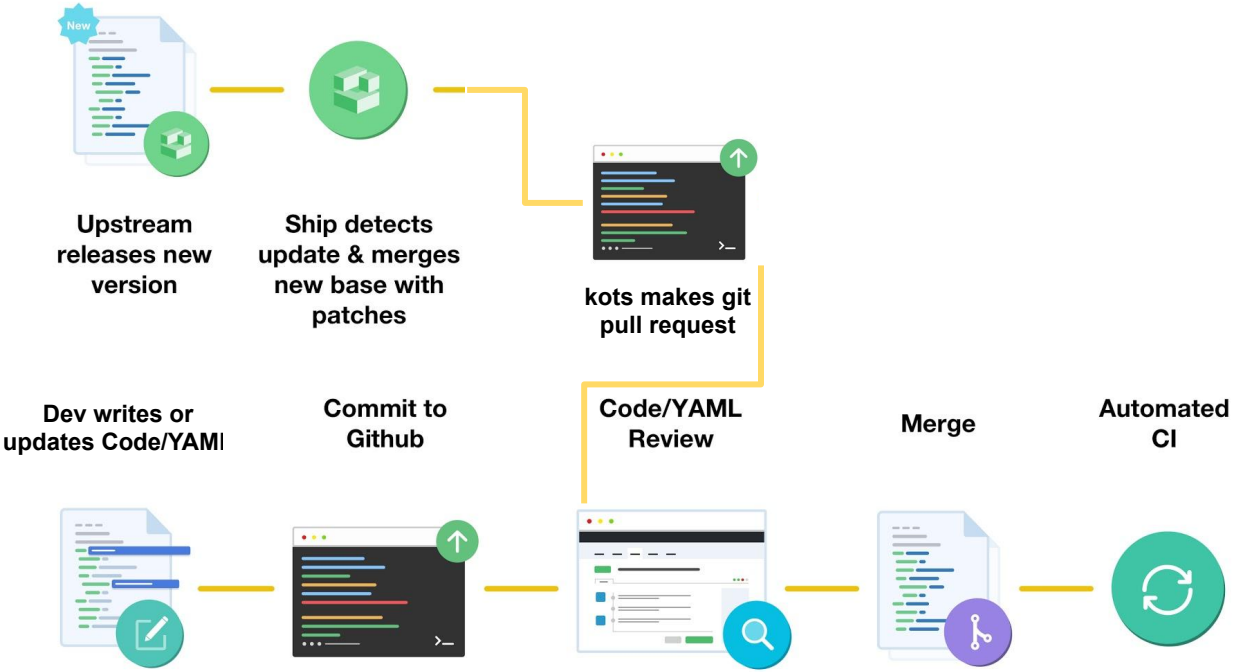


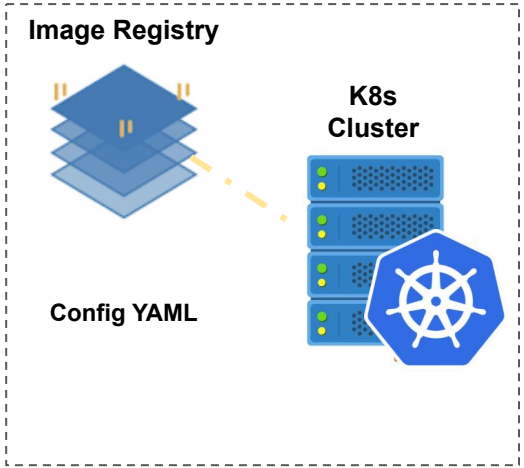
Common 1st-party application SDLC

Let’s install a community chart, with my custom requirements.

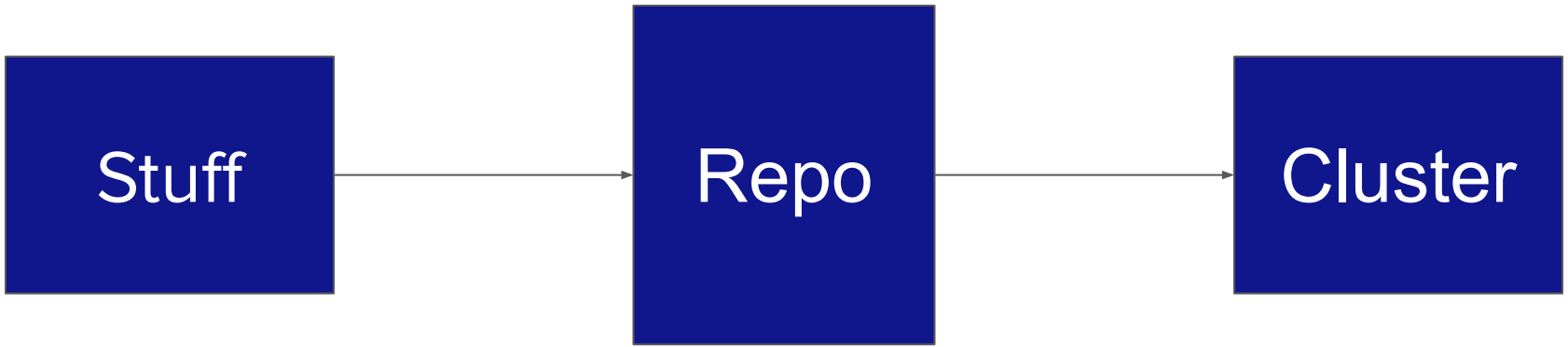
Assume I’m using gitops

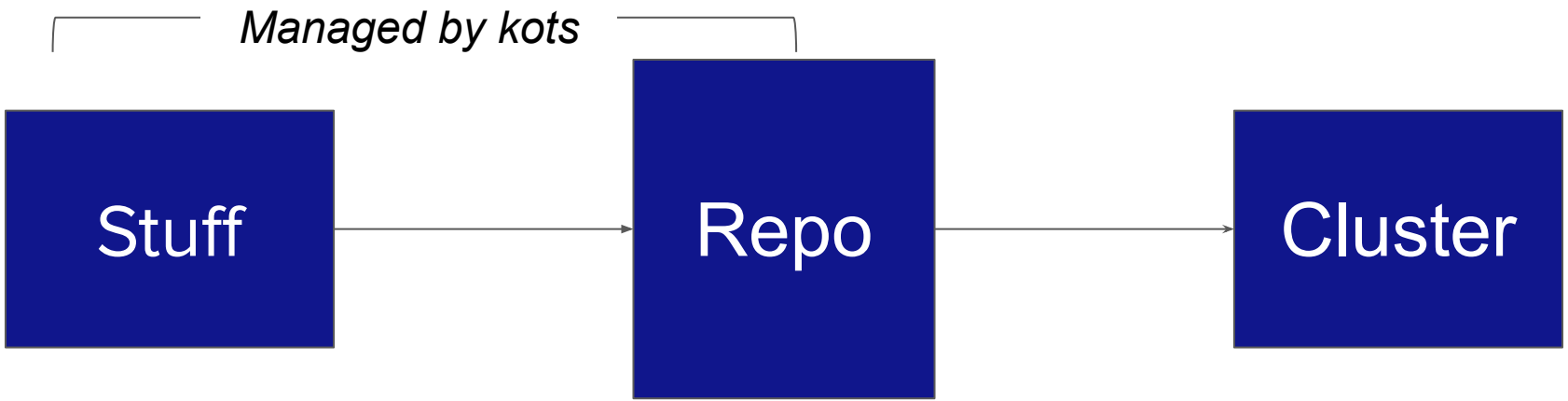
Example Deployment Workflows *(OSS / COTS with ship)*





Common 1st-party application SDLC


Stuff
Repo
Cluster


Managed by kots
Stuff
Repo
Cluster

better-db / chart

Unwatch 3 | Star 1 | Fork

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Security | Insights

No description, website, or topics provided.

70 commits | **13** branches | **2** releases | **2** contributors

Branch: **master** | New pull request | Create new file | Upload files | Find File | Clone or downl

dexhorthy Merge pull request #21 from better-db/0.2.0 ... Latest commit `15186d0` 11 hour

templates	v0.2.0 Bump image tag, add log level param	12 hours
.gitignore	first commit	last
.helmignore	first commit	last
Chart.yaml	v0.2.0 Bump image tag, add log level param	12 hours
README.md	first commit	last
values.yaml	v0.2.0 Bump image tag, add log level param	12 hours

```yaml
replicaCount: 1

image:
  repository: nginx
  tag: 1.15.2
  pullPolicy: IfNotPresent

logLevel: info

service:
  type: ClusterIP
  port: 80

securityContext:
  allowPrivilegeEscalation: true

resources:
  # limits:
  #  cpu: 100m
  #  memory: 128Mi
  # requests:
  #  cpu: 100m
  #  memory: 128Mi

nodeSelector: {}

tolerations: []

affinity: {}
```

Installing Better DB

Upstream

better-db/chart

Midstream

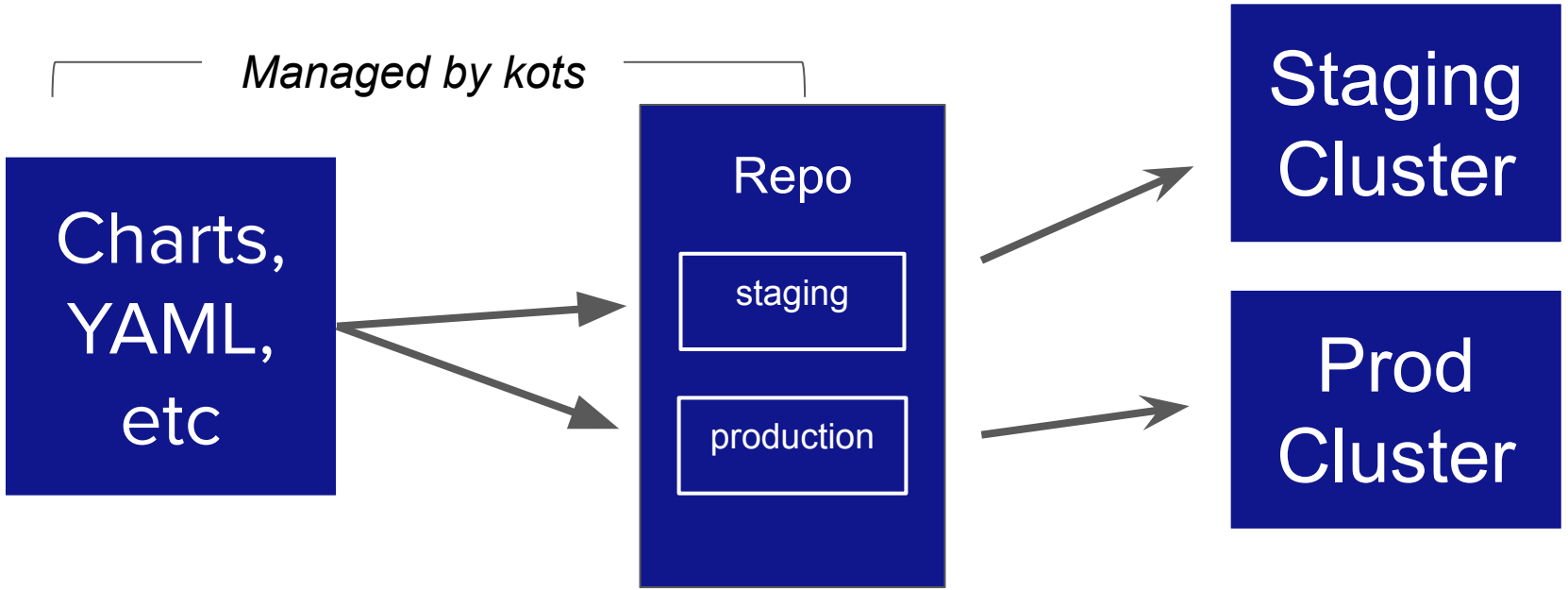
base

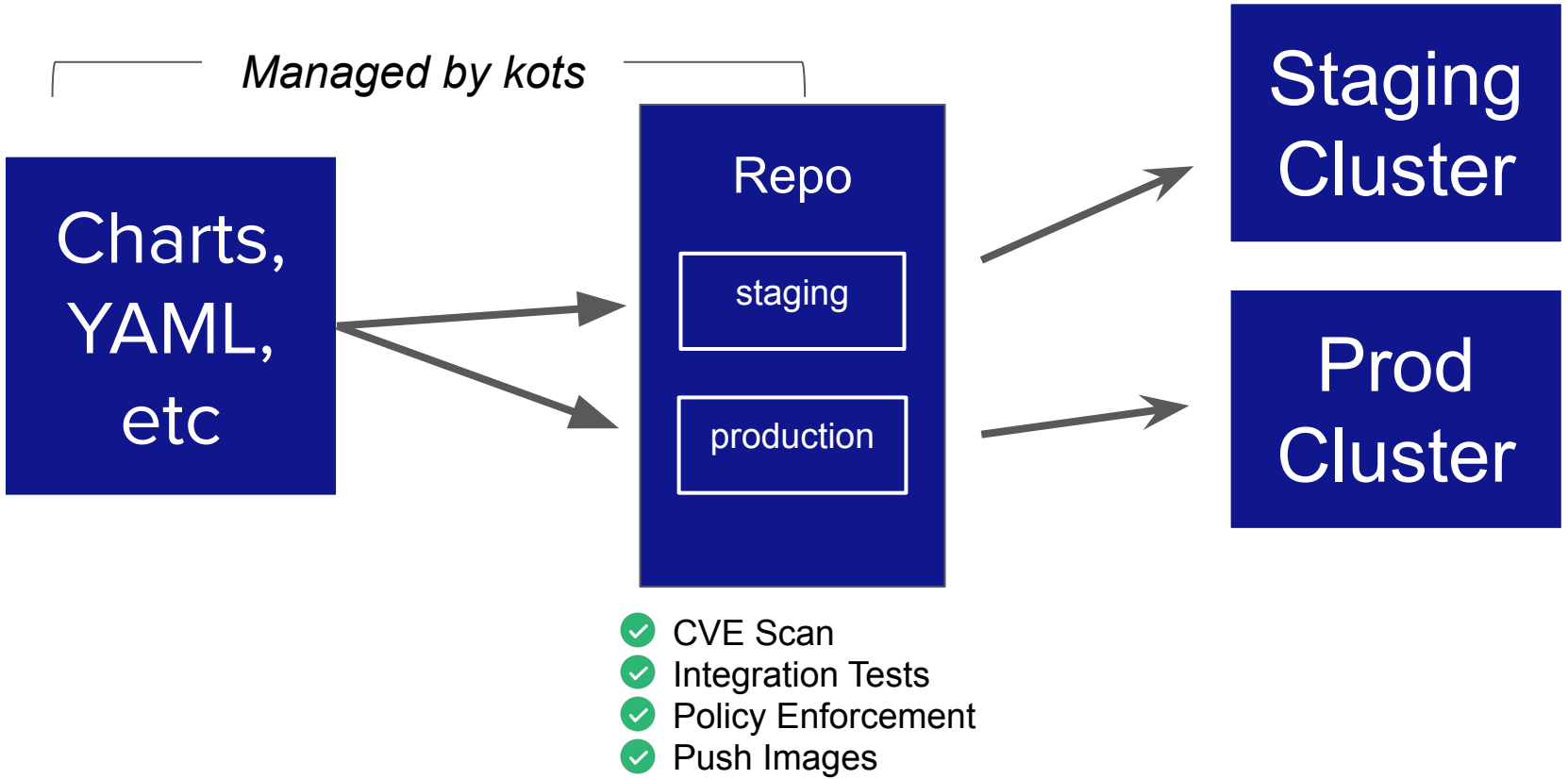
overlays/midstream

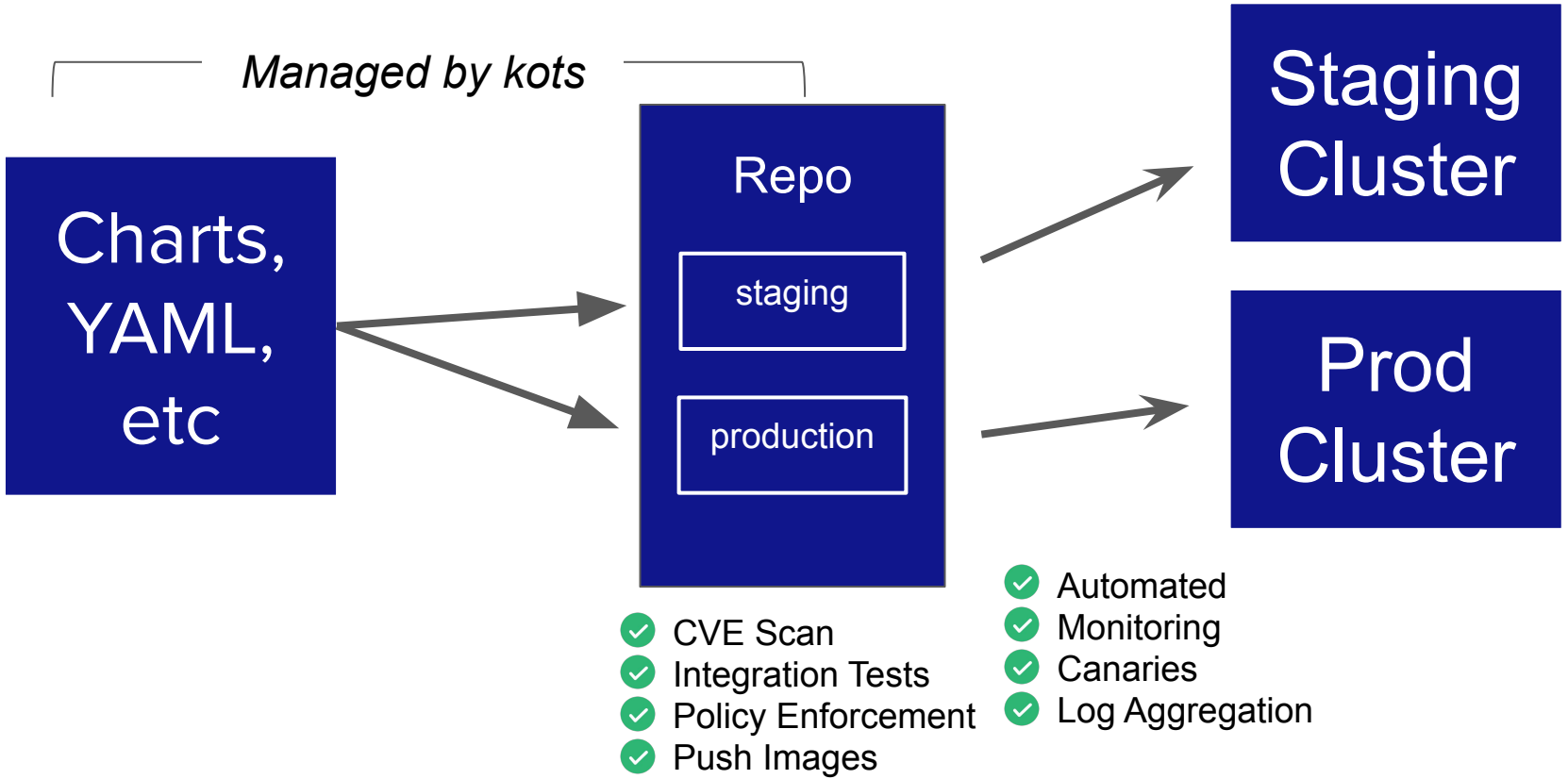
Downstreams

overlays/production

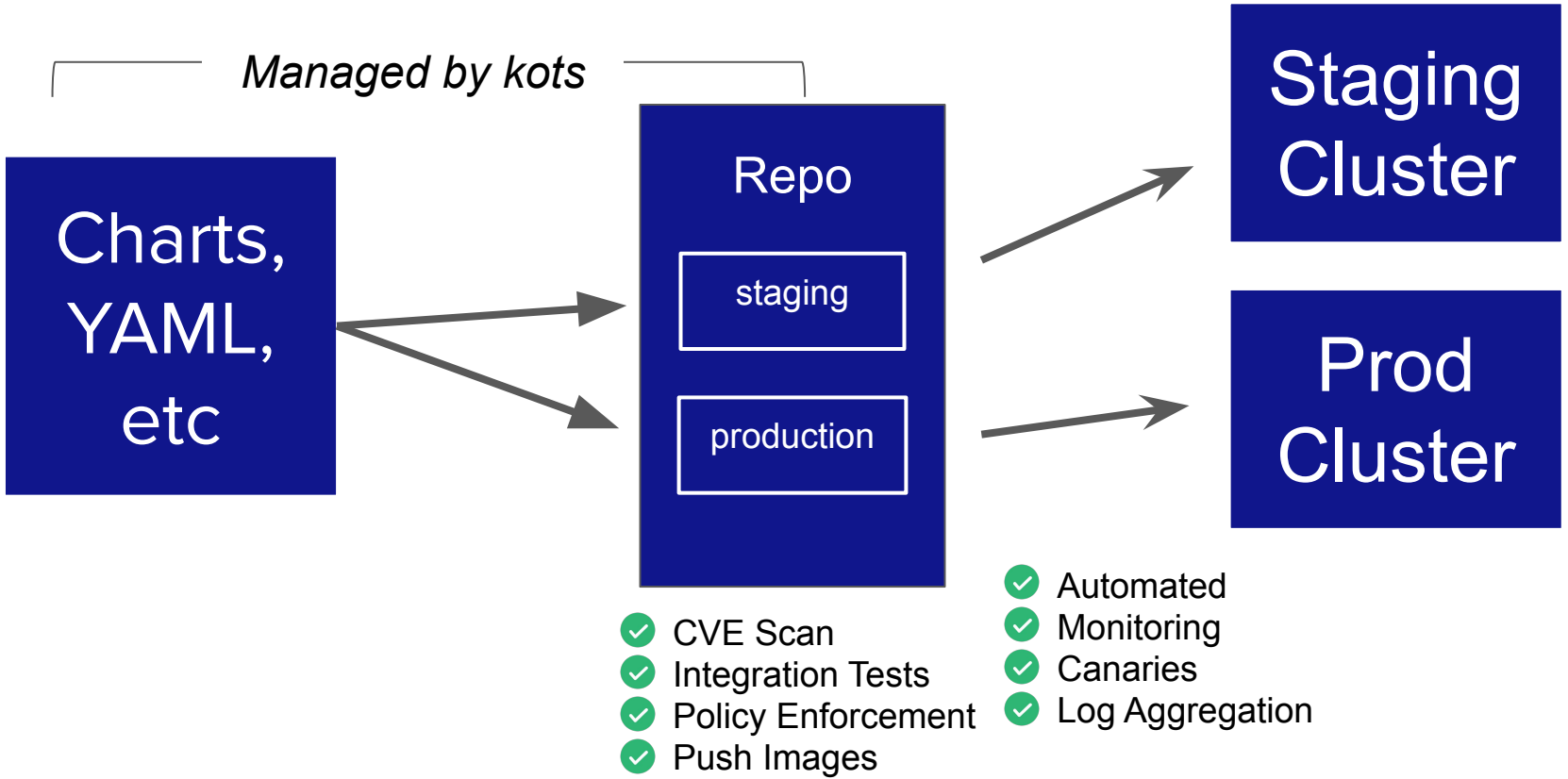
overlays/staging


Managed by kots
Charts, YAML, etc
Repo
staging
production
Staging Cluster
Prod Cluster


Managed by kots
Charts, YAML, etc
Repo
staging
production
Staging Cluster
Prod Cluster
CVE Scan
Integration Tests
Policy Enforcement
Push Images

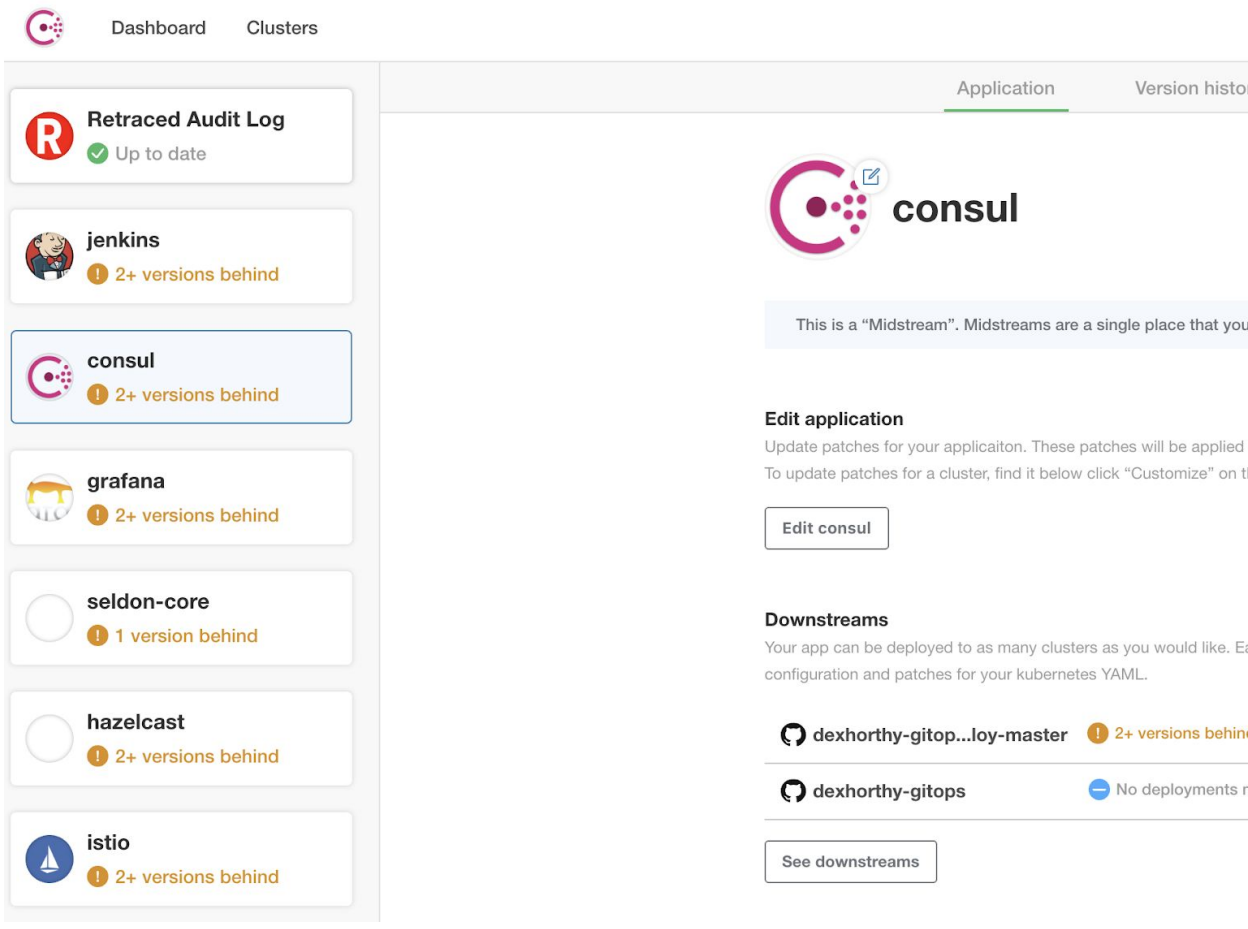

Managed by kots
Charts, YAML, etc
Repo
staging
production
Staging Cluster
Prod Cluster
CVE Scan
Integration Tests
Policy Enforcement
Push Images
Automated
Monitoring
Canaries
Log Aggregation

kots + kotsadm Demo


Managed by kots
Charts, YAML, etc
Repo
staging
production
Staging Cluster
Prod Cluster
CVE Scan
Integration Tests
Policy Enforcement
Push Images
Automated
Monitoring
Canaries
Log Aggregation

kotsadm

- OSS k8s app
- Multi-app
- Gitops-enabled (via Automatic PRs)
- Multi-Downstream
- Cluster Preflights
- Sophisticated Troubleshooting



Kots Wins

Operator

Own less yaml

Use your own process (e.g. GitOps)

Customize everything

Transparency

Maintainer

Write mostly YAML

Only template app config

Nobody Forks your chart

Community

Makes helm an even more flexible packaging format

Enables first-time and advanced users

What's Next

Download Kots: https://github.com/replicatedhq/kots

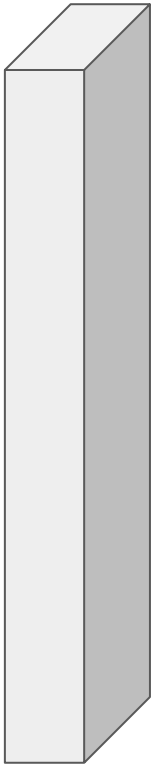
Kots pull: **`kubectl kots pull helm://stable/grafana`**

Kots install: **`kubectl kots install helm://stable/grafana`**

Outdated: **`kubectl outdated`** *Discover outdated cluster images*

Unfork: **`kubectl unfork`** *Unfork your deployed helm charts*

#ship at kubernetes.slack.com



Embrace the wall

Thanks!

Day 2 Operations of Helm Charts

Dexter Horthy