Eating Your Vegetables: How to Manage 2.5 Million Lines of YAML

Jesse Suen (Intuit)
Danny Thomson (Stytch)

Jesse Suen
Principal Software Engineer, Intuit

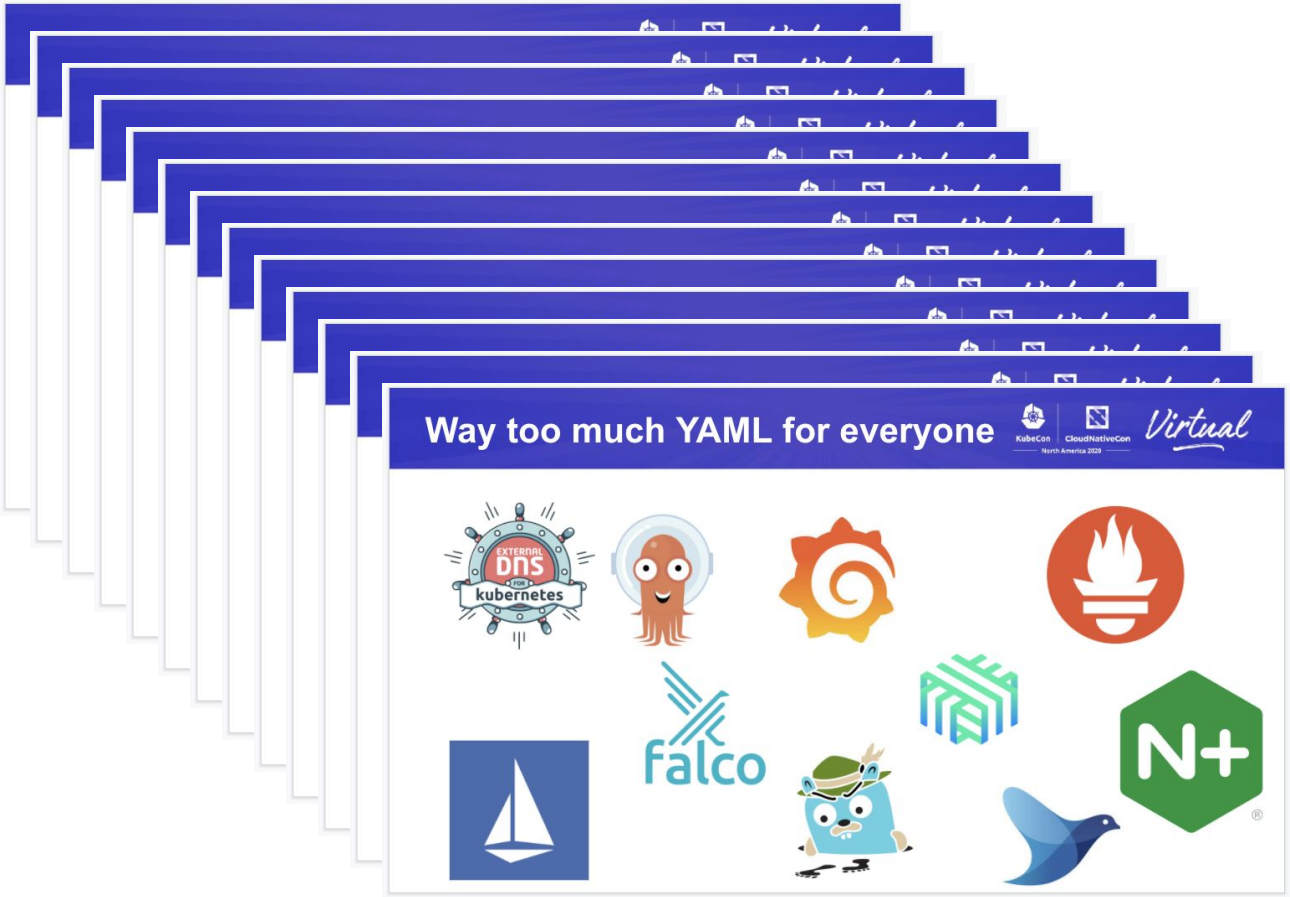
Daniel Thomson
Software Engineer, Stytch

K8s YAML is very powerful, but hard


```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: guestbook
spec:
 replicas: 3
 selector:
  ...
 affinity:
   podAntiAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
     - labelSelector:
         matchExpressions:
         - key: app
           operator: In
           values:
           - web
     topologyKey: "kubernetes.io/hostname"
 template:
   metadata:
     labels:
       app: web
   spec:
    containers:
    - name: app
      image: guestbook:1.0
```

Way too much YAML for everyone

What about multiple clusters



Considerations

Considerations - Personas

Operators

- Deploying off-the-shelf (OTS) software to run the platform
- Want controlled and stable upgrades, semantic versions

Developers

- Building bespoke applications as a service
- Focused on business logic, and less about platform
- Want to deploy early and often, don’t care about semantic versioning

Considerations - Developer Experience

- Experience and comfort level with Kubernetes
  - Highly specialized power users vs. zero interest
  - e.g. How do you achieve basic K8s use-cases:
    - anti affinity

- Affects how you expose the platform to the user
  - e.g. Abstraction v. RAW YAML

Considerations - Control

Centralized control

- Provide standard patterns and best practices
- Easier maintenance (e.g. upgrades, deprecated APIs)
- Security restrictions (e.g. creating ClusterRoles)

Developer control

- Self-service where it makes sense
  - onboarding to new environments (e.g. additional namespaces)
  - specifying HPA metrics
  - choosing a deployment strategy (e.g. rolling update, blue-green, canary)
- Leverage documentation and automation

Approaches

Approaches - Raw YAML

Just manage Kubernetes YAML

Advantages:

- Simple and straightforward
- Full flexibility
- Nothing to learn

Disadvantages:

- Zero configuration re-use (unmaintainable)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: guestbook
spec:
 replicas: 3
 selector:
   matchLabels:
     app: guestbook
 template:
   metadata:
     labels:
       app: guestbook
   spec:
     containers:
     - image: guestbook:v1.0
       name: guestbook
       ports:
       - containerPort: 80
```

Approaches - Templating

Expose only the top level parameters to a template, which control the final output.

Advantages:

- Simpler configuration
- Flexible

Disadvantages:

- Need to parameterize everything
- Templates become complex and unreadable

Examples:

- helm, jsonnet

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: {{ include "guestbook.fullname" . }}
 labels:
   {{- include "guestbook.labels" . | nindent 4 }}
spec:
{{- if not .Values.autoscaling.enabled }}
 replicas: {{ .Values.replicaCount }}
{{- end }}
 selector:
   matchLabels:
     {{- include "guestbook.selectorLabels" . | nindent 6 }}
 template:
   metadata:
   {{- with .Values.podAnnotations }}
     annotations:
       {{- toYaml . | nindent 8 }}
   {{- end }}
     labels:
       {{- include "guestbook.selectorLabels" . | nindent 8 }}
   spec:
     {{- with .Values.imagePullSecrets }}
     imagePullSecrets:
       {{- toYaml . | nindent 8 }}
     {{- end }}
     serviceAccountName: {{ include "guestbook.serviceAccountName" . }}
...
...
```

Approaches - Overlay

Overlaying defines a common “base” to share across variants.
Each variant only contains the configuration differences for that environment.

Advantages:

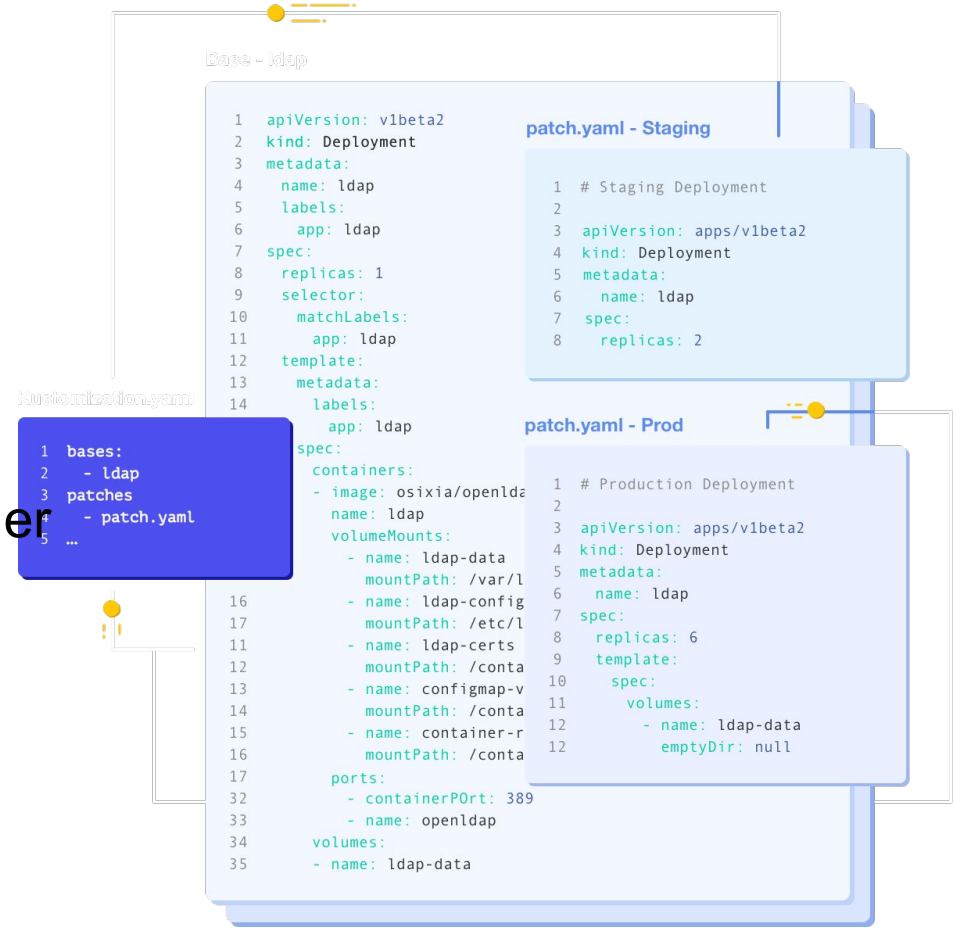
- Excellent readability
- Excellent configuration reuse
- Mostly flexible

Disadvantages:

- Not immediately intuitive
- Lack of parameterization makes many use cases harder than necessary

Examples:

- kustomize, jsonnet



Approaches - Abstraction

Hide the details from the user with an abstraction and simpler interface.

Advantages:

- Simpler configuration
- Can implement and control organizational standards

Disadvantages:

- Less flexibility
- Eventually end up with a leaky abstraction
- No one has figured out the right abstraction yet for Kubernetes

Examples:

- pulumi, cdk8s, helm

```
name: guestbook
type: WebService
dnsname: guestbook.intuit.com
image: guestbook:v1.0
updateStrategy: BlueGreen
replicas:
  min: 3
  max: 10
mesh:
  enabled: true
```

Approaches - Codify

Just use a programing language!

Advantages:

- Leverage programming features (e.g. loops, conditionals, functions)
- Tends to go hand-in-hand with abstraction, including its benefits
- Can be tested

Disadvantages:

- Just another codebase with bugs
- Difficult to understand how code affects final result

Examples:

- cdk8s, pulumi, jsonnet

```
import { Chart } from 'cdk8s';
import { Construct } from 'constructs';
import { WebService } from
'./lib/web-service';

export class MyChart extends Chart {
 constructor(scope: Construct, ns: string) {
   super(scope, ns);

   new WebService(this, guestbook, {
     image: guestbook:v1.0',
     replicas: 3
   });
   new WebService(this, 'redis', {
     image: 'redis',
     containerPort: 6379
   });
 }
}
```

Intuit Case Study

Use Case & Requirements

Use Case

- 4,000 developers deploying SaaS applications
- Manage multiple environments
  - Namespace (qa, e2e, prd-use2, prd-usw2)
  - Mostly identical, with slight variations in config (e.g. DNS names, AWS ARNs, IAM roles)
- DevOps culture: you build it, you run it

Requirements

- Provide standard set of patterns and best practices (paved road)
- Provide flexibility (even at the cost of simplicity)
  - Exposed developers to Kubernetes YAML
- GitOps friendly

Solution - Kustomize

- Preserves the full power of Kubernetes
- Kubernetes native, well supported and documented
- Readable for both developers & platform team
- Overlay pattern promotes config re-use maintainability
- Centrally control & distribute standard patterns across organization

Kustomize's Killer Feature

Centrally Managed Remote Base

- A "catalog" of generic starter YAML
- Simple base consists of:
  - Deployment + Service + Ingress
- Advanced examples:
  - HPA, Canary Analysis
- Semantically versioned
- Provides standard patterns & best practices
  - (e.g. pod readiness gates, resource requests, ingress annotations)

Developer Owned Deployment Repository

- Derives from central remote base
- Customized for the needs for their service

Standard Ingress Example


```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 annotations:
   alb.ingress.kubernetes.io/backend-protocol: HTTPS
   alb.ingress.kubernetes.io/load-balancer-attributes: access_logs.s3.enabled=false
   alb.ingress.kubernetes.io/certificate-arn: TODO:certificate-ARN
   alb.ingress.kubernetes.io/healthcheck-path: /health/full
   alb.ingress.kubernetes.io/healthcheck-protocol: HTTPS
   alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS": 443}]'
   alb.ingress.kubernetes.io/scheme: internet-facing
   alb.ingress.kubernetes.io/security-groups: intuit-vpn-tcp-443
   alb.ingress.kubernetes.io/ssl-policy: ELBSecurityPolicy-TLS-1-2-2017-01
   alb.ingress.kubernetes.io/subnets: PublicSubnetAz1, PublicSubnetAz2, PublicSubnetAz3
   alb.ingress.kubernetes.io/healthcheck-interval-seconds: "60"
   kubernetes.io/ingress.class: aws-alb
   external-dns.alpha.kubernetes.io/hostname: TODO:albDnsHostname
 name: ingress
spec:
 rules:
 - http:
     paths:
     - backend:
         serviceName: service
         servicePort: 443
       path: /*
```

Deployment Repository


```
guestbook
├── app-base
│   ├── kustomization.yaml
│   ├── deployment-patch.yaml
│   └── ingress-patch.yaml
└── environments
    ├── e2e-usw2
    │   ├── kustomization.yaml
    │   ├── deployment-patch.yaml
    │   └── ingress-patch.yaml
    └── prd-usw2
        ├── kustomization.yaml
        ├── deployment-patch.yaml
        └── ingress-patch.yaml
```

- app-base directory inherits from central remote base. contains the common definitions for all environments of the service.
- environments directories inherit from the base, and only include changes specific to the environment.

Deployment Repository


```
guestbook
├── app-base
│   ├── kustomization.yaml
│   ├── deployment-patch.yaml
│   └── ingress-patch.yaml
└── environments
    ├── e2e-usw2
    │   ├── kustomization.yaml
    │   ├── deployment-patch.yaml
    │   └── ingress-patch.yaml
    └── prd-usw2
        ├── kustomization.yaml
        ├── deployment-patch.yaml
        └── ingress-patch.yaml
```

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

namePrefix: guestbook-

resources:
- >
  https://github.intuit.com/dev-patterns/
  intuit-kustomize//intuit-service-base?ref=v4.0.0

patchesStrategicMerge:
- deployment-patch.yaml
- ingress-patch.yaml
```

Deployment Repository


```
guestbook
├── app-base
│   ├── kustomization.yaml
│   ├── deployment-patch.yaml
│   └── ingress-patch.yaml
└── environments
    ├── e2e-usw2
    │   ├── kustomization.yaml
    │   ├── deployment-patch.yaml
    │   └── ingress-patch.yaml
    └── prd-usw2
        ├── kustomization.yaml
        ├── deployment-patch.yaml
        └── ingress-patch.yaml
```

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../../app-base

patchesStrategicMerge:
- deployment-patch.yaml
- ingress-patch.yaml

images:
- name: guestbook
  newTag: master-9f29eb5
```

Deployment Repository


```
guestbook
├── app-base
│   ├── kustomization.yaml
│   ├── deployment-patch.yaml
│   └── ingress-patch.yaml
└── environments
    ├── e2e-usw2
    │   ├── kustomization.yaml
    │   ├── deployment-patch.yaml
    │   └── ingress-patch.yaml
    └── prd-usw2
        ├── kustomization.yaml
        ├── deployment-patch.yaml
        └── ingress-patch.yaml
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 annotations:
   alb.ingress.kubernetes.io/certificate-arn: >
     arn:aws:acm:us-west-2:1234567890:certificate/
     da97886a-11e0-4665-b767-5d8f50713da6
   external-dns.alpha.kubernetes.io/hostname: >
     guestbook-prod.intuit.com
 name: ingress
```

Where are we now?

Single Environment × Environments × Services = Deployed YAML

$$250 \times 4 \times 2500 = 2.5M$$

Base + (4 x environment overlays) × Services = Managed YAML

$$90 + (4 \times 45) \times 2500 = 675K$$

$$270 \times 2500 = 675K$$

Challenges

#1 User support

- Given a lot of foot guns
- Users will fall off the paved road

#2 Automation & migrations

- YAML is hard to “upgrade”
- Thousands of pull requests

#3 Kustomize

- Breaking behavior
- Lack of CRD support

Final Thoughts

- No perfect solution
- No one-size-fit-all, highly dependent on your organization
- At a certain scale, managing YAML is a lot of work

What's next?

- Better abstractions
- UI assisted configuration management

Resources

- Declarative Application Management Whitepaper - Bryan Grant
- https://jobs.lexver.co/stytch

http://bit.ly/gitops-and-k8s


KEEP CLOUD NATIVE
EVERYWHERE
KubeCon
CloudNativeCon
North America 2020
Virtual