KubeCon | CloudNativeCon

Europe 2023

KubeCon | CloudNativeCon

Europe 2023

Mission Accomplished: Kubernetes Is Not a Monorepo. Now Our Work Begins!

Justin Santa Barbara @Google
Ciprian Hacman @Microsoft

About Us

Justin has been contributing to Kubernetes since 2014, and loves helping users adopt and grow their use of Kubernetes - initially as the primary maintainer of the kubernetes AWS support, he also started the kOps project. He joined Google in 2018 to work full time on Kubernetes, focusing on kubernetes configuration management.

Ciprian is a DevOps/Software Engineer, helping companies modernize their infrastructure and migrate to Kubernetes. He is also an open source project maintainer for the kOps project, etcd-manager, cloud-provider-aws and frequent contributor to other projects in the Kubernetes ecosystem.

Summary

Why did we split up the monorepo?

What did we lose?

How can we get it back?

The beginning

In the beginning:

- One kubernetes repo: github.com/kubernetes/kubernetes
- All components versioned together
- End-to-end tested on multiple clouds on every PR.

The beginning

Very easy to make big changes across layers

- One PR contains everything

We had some tests on AWS and GCP

- Including testing cloud provider functionality
- There were also cloud providers running their own tests and uploading the results

Why did we split up the monorepo?

So this all seems pretty good, why did we stop doing this?

We had a lot of **technical** issues:

- Github notifications
- Github volume - PRs would fail to load
- We couldn’t merge fast enough - introduced things like bulk merge in tide

Why did we split up the monorepo?

So this all seems pretty good, why did we stop doing this?

We had a lot of **people** issues:

- Hard to route PRs to people.
- Reviews ended up being bottlenecked on a few key people
  - OWNERs meant those people were needed on **more** PRs
- Hard to coordinate everyone onto the same release schedule
- Hard to coordinate with all cloud providers

Why did we split up the monorepo?

So this all seems pretty good, why did we stop doing this?

It enabled **architectural** improvements:

- Creating stricter architectural boundaries
- Projects depending on pieces of K8s, like client-go, but had to pull all K8s as a dependency

Why did we split up the monorepo?

OK but really, **why**?

We didn’t want to be a centralized project where we act as gatekeepers.

We hoped to spawn an industry of tools that work together - the Cloud Native ecosystem.

Strategies:

- CRDs
- Webhooks
- “You can do this outside of k/k”
- CRI/CNI/CSI

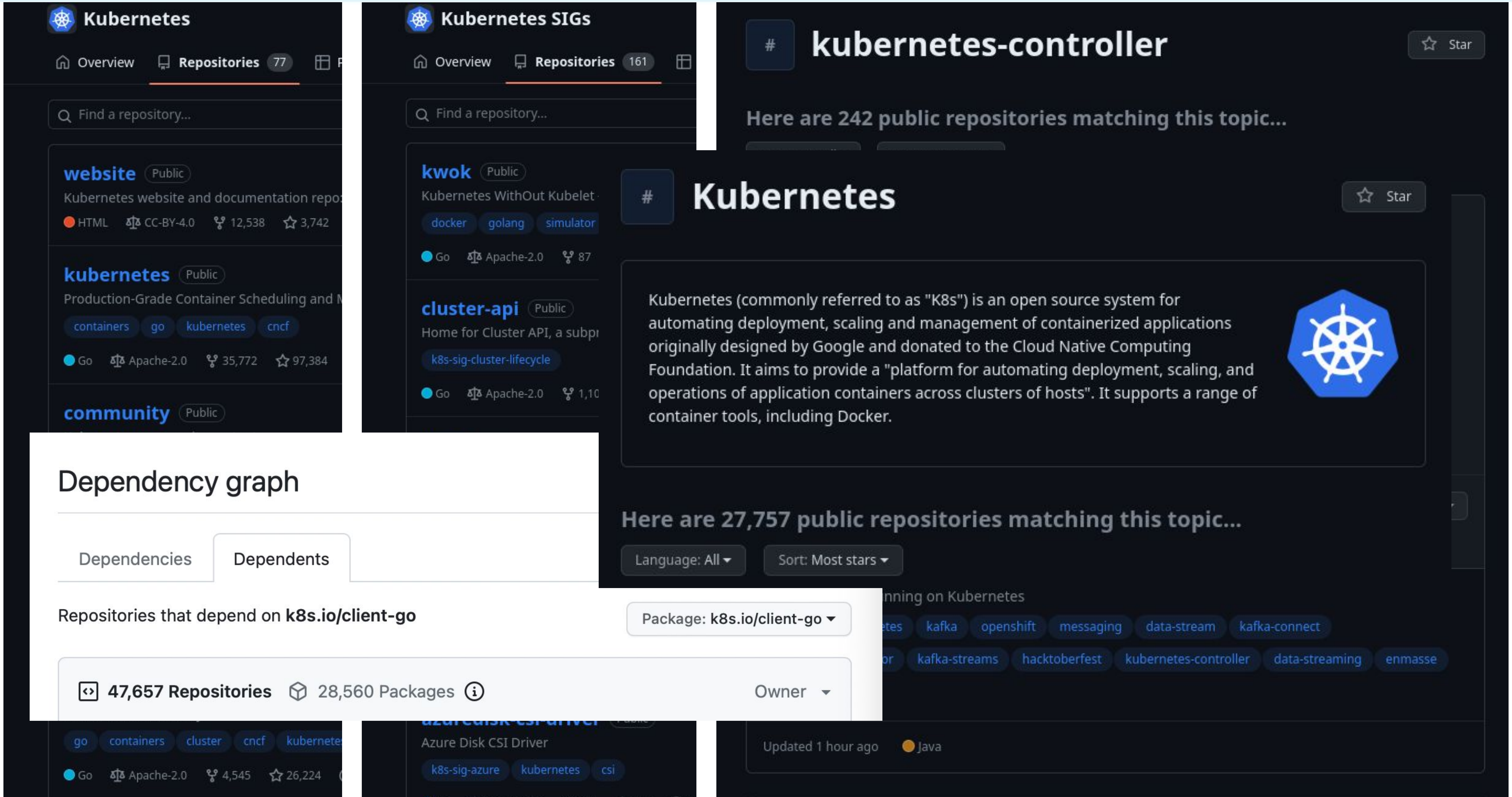
Mission Accomplished!

With the 1.27 release, the cloud providers are no longer in the k/k repository.

These were the last to go, so now we have broken up the monorepo:

- CSI
- CRI
- CNI
- Cloud providers

All developed in different github repositories.



Kubernetes

Overview · Repositories 77

Find a repository...

website Public
Kubernetes website and documentation repo
HTML · CC-BY-4.0 · 12,538 · 3,742

kubernetes Public
Production-Grade Container Scheduling and M
containers · go · kubernetes · cncf
Go · Apache-2.0 · 35,772 · 97,384

community Public

go · containers · cluster · cncf · kubernete
Go · Apache-2.0 · 4,545 · 26,224

Kubernetes SIGs

Overview · Repositories 161

Find a repository...

kwok Public
Kubernetes WithOut Kubelet
docker · golang · simulator
Go · Apache-2.0 · 87

cluster-api Public
Home for Cluster API, a subp
k8s-sig-cluster-lifecycle
Go · Apache-2.0 · 1,10

azuredisk-csi-driver Public
Azure Disk CSI Driver
k8s-sig-azure · kubernetes · csi

kubernetes-controller

Star

Here are 242 public repositories matching this topic...

nning on Kubernetes
etes · kafka · openshift · messaging · data-stream · kafka-connect
or · kafka-streams · hacktoberfest · kubernetes-controller · data-streaming · enmasse

Updated 1 hour ago · Java

Kubernetes

Star

Kubernetes (commonly referred to as "K8s") is an open source system for automating deployment, scaling and management of containerized applications originally designed by Google and donated to the Cloud Native Computing Foundation. It aims to provide a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts". It supports a range of container tools, including Docker.

Here are 27,757 public repositories matching this topic...

Language: All · Sort: Most stars

Dependency graph

Dependencies · Dependents

Repositories that depend on **k8s.io/client-go**

Package: k8s.io/client-go

47,657 Repositories · 28,560 Packages

Owner

What have we gained?

Innovation can now happen **outside** of k/k.

And it is: Karpenter, operators, etc

IPv6 is an interesting story

Even when Docker was removed, kube kept on trucking with containerd, and CRI-O.

Much easier to contribute to individual CSI / CNI / Cloud providers etc.

What have we lost?

The experience for Kubernetes administrators is worse:

- We no longer test everything together (at least not in k/k)
- More components makes Kubernetes harder to install
- More components causes more complexity on upgrades

Can we get back to where we were?

Should we put everything back together for testing?

- Is this reassembling the monorepo?

Is this even practical?

- Choosing versions is difficult

  Exponential complexity means distros must choose subsets

kOps does this

But so do others distros …

… so does every project that just wants to run e2e tests…

… and we all do it separately

Case-study: cloud-provider-gcp

We had a bug on kOps + (external) cloudprovider-gcp + “IP-Alias” networking, it wasn’t passing e2e tests.

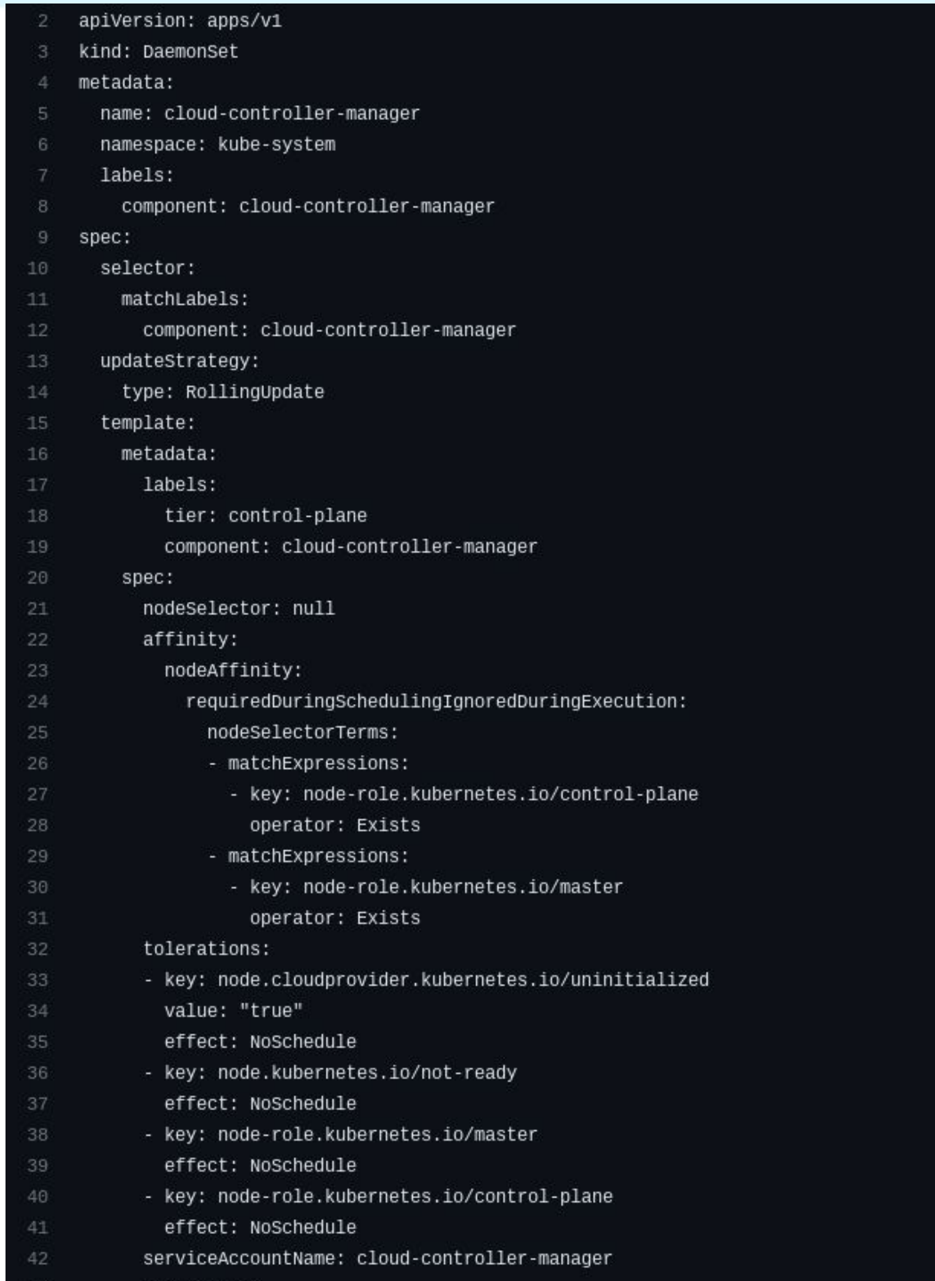
Spoiler:

- kOps manifest for cloud-provider-gcp was different from the manifest in the cloud-provider-gcp repo
- It turned out **both** sides were subtly broken here

Catching failures in e2e is “too late”

What we’re heading for is:

- cloud-provider-gcp can run tests with kOps
- cloud-provider-gcp publishes a manifest that they tested
- kOps consumes that manifest, not some other manifest of our own creation
- Other tools can also add their own tests to cloud-provider-gcp and also consume the tested manifest.



```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cloud-controller-manager
  namespace: kube-system
  labels:
    component: cloud-controller-manager
spec:
  selector:
    matchLabels:
      component: cloud-controller-manager
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        tier: control-plane
        component: cloud-controller-manager
    spec:
      nodeSelector: null
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: node-role.kubernetes.io/control-plane
                operator: Exists
            - matchExpressions:
              - key: node-role.kubernetes.io/master
                operator: Exists
      tolerations:
      - key: node.cloudprovider.kubernetes.io/uninitialized
        value: "true"
        effect: NoSchedule
      - key: node.kubernetes.io/not-ready
        effect: NoSchedule
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      - key: node-role.kubernetes.io/control-plane
        effect: NoSchedule
      serviceAccountName: cloud-controller-manager
```

Manifests, manifests everywhere

We are asking components to publish working manifests, not just container images.

If all the Kubernetes components start testing their manifest in fully-assembled distributions (kOps and others), then all the “distros” can consume these manifests.

We should catch all single-source bugs in this way (where a bug is introduced by one component)

“Combination” bugs will be caught by the distros - and are hopefully much rarer.

Problems of manifests

Components should think about upgrades:

- Immutability
- Disruption
- Skew

Non-kube objects:

- IAM policies
- Firewall rules
- TLS certificates

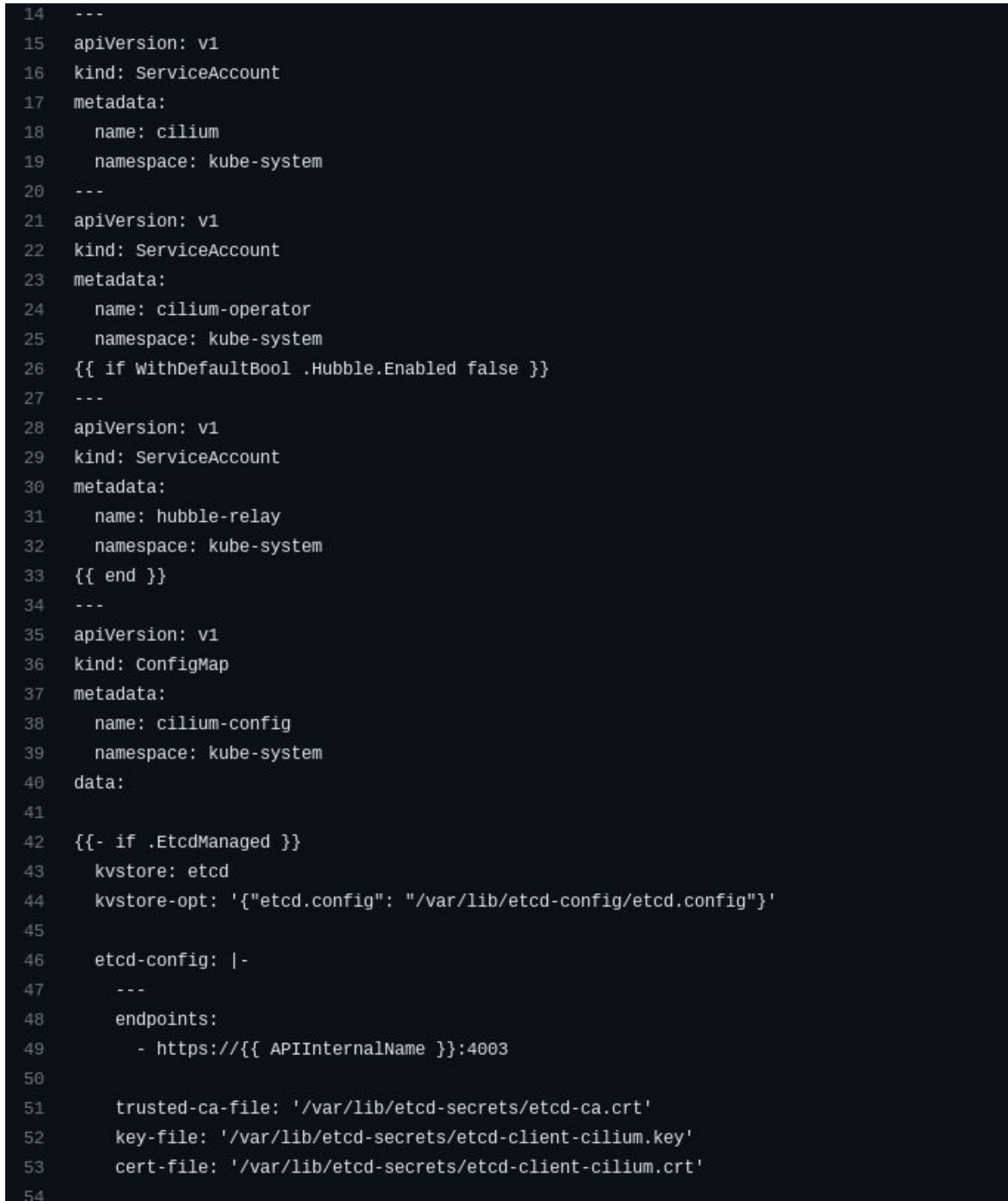
Problems of manifests

What about parameters:

- Cluster name
- Cluster CIDR
- etc

Variants:

- Encryption enabled / not enabled.
- We can publish multiple manifests but this breaks down with too many variants.



```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cilium
  namespace: kube-system
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cilium-operator
  namespace: kube-system
{{ if WithDefaultBool .Hubble.Enabled false }}
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: hubble-relay
  namespace: kube-system
{{ end }}
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: cilium-config
  namespace: kube-system
data:

{{- if .EtcdManaged }}
  kvstore: etcd
  kvstore-opt: '{"etcd.config": "/var/lib/etcd-config/etcd.config"}'

  etcd-config: |-
    ---
    endpoints:
      - https://{{ APIInternalName }}:4003

    trusted-ca-file: '/var/lib/etcd-secrets/etcd-ca.crt'
    key-file: '/var/lib/etcd-secrets/etcd-client-cilium.key'
    cert-file: '/var/lib/etcd-secrets/etcd-client-cilium.crt'
```

Is this helm charts?

Helm charts are close but not a perfect fit:

- Will the project really support every combination of parameters? Are they all tested?
- There are no standards for how to name/structure parameters; they are hard to consume
- This becomes an API; it has to remain stable across upgrades

The Proposed Contract

We ask:

Components publish manifests

(How they should be used in production)

Components test with this manifest

(Ideally with production tooling)

We promise:

We will help you test your manifest with production tooling

(kOps and hopefully others)

We will not modify your manifest

(without at least talking to you first!)

Kube-baya

This won’t happen accidentally; the organizational work is at least as hard as the technical work.

Together we can build a reliable and easy Kubernetes experience, while allowing more choice and experimentation.

Please scan the QR Code above
to leave feedback on this session