KubeCon | CloudNativeCon

North America 2024

KubeCon | CloudNativeCon

North America 2024

Upgrade Safely

Avoid the Pitfalls of Kubernetes Versioning

Rob Scott, Google

@robertjscott.ca

How many of you have been burned by an upgrade gone wrong?

Painful Upgrade #1:

Ingress

Kubernetes v1.22 Upgrade Story

- Upgrade cluster to Kubernetes v1.22
- Prod goes down
- Notice that Ingress controller is in CrashLoopBackoff
- Panic

So what went wrong?

API Group And Version

$$\texttt{apiVersion: }\overbrace{\texttt{networking.k8s.io}}^{\textbf{API Group}}\texttt{/}\overbrace{\texttt{v1beta1}}^{\textbf{API Version}}$$

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: minimal-ingress
```

Kubernetes Version

- Semantic Versioning
  - Major versions can include breaking changes - unlikely to happen
  - Minor versions released 3x/year, include new features and APIs
  - Patch versions for patches and bug fixes

$$\overbrace{1}^{\text{Major}}.\overbrace{31}^{\text{Minor}}.\overbrace{1}^{\text{Patch}}$$

What happened

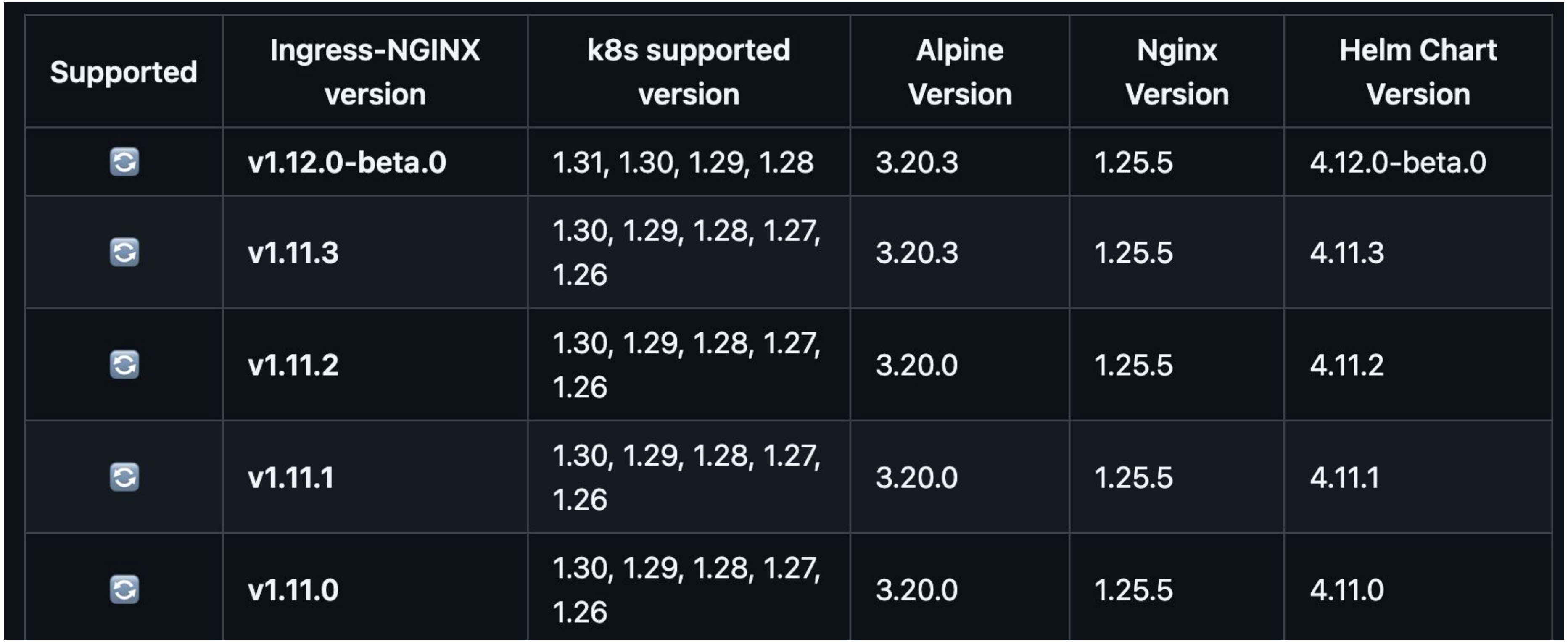
	v1.16	v1.17	v1.18	v1.19	v1.20	v1.21	v1.22	v1.23	v1.24
v1beta1	🟩	🟩	🟩	🟩	🟩	🟩			
v1				🟩	🟩	🟩	🟩	🟩	🟩

- Ingress was stuck in beta for ~5 years
- Kubernetes v1.19
  - Ingress v1 is announced
  - Ingress v1beta1 is deprecated
- Kubernetes v1.22 (one year later)
  - Ingress v1beta1 is removed

What happened

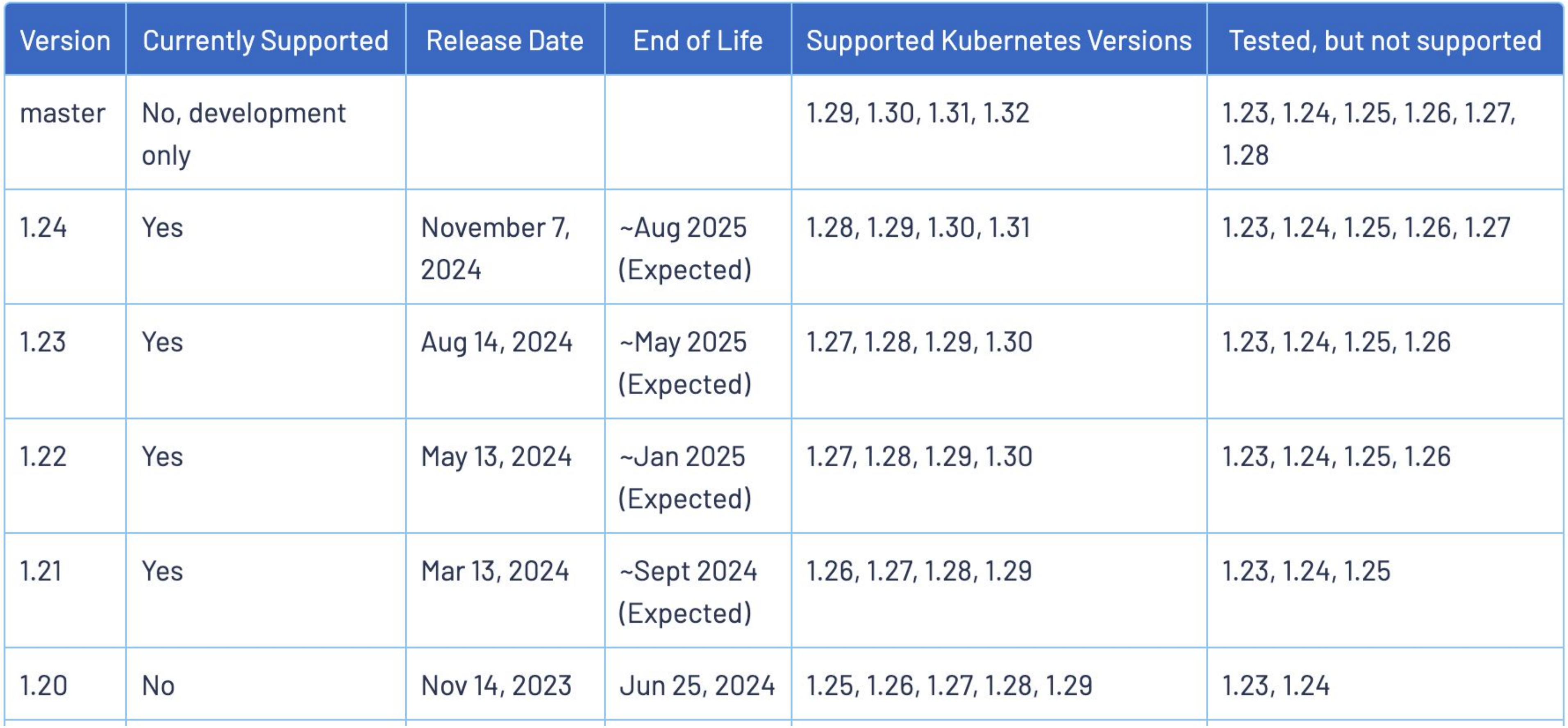
- The vast majority of Kubernetes controllers use a single API version to access an API
- Most Ingress controllers try to support multiple Kubernetes versions with a single release to help provide seamless upgrades
- This Ingress upgrade gave controller authors two options:
  - Rearchitect their controllers to support multiple versions of the Ingress API
  - Don’t try to support both Kubernetes v1.19 and v1.22 in the same version of their controller

Ingress-NGINX Supported Versions



Supported	Ingress-NGINX version	k8s supported version	Alpine Version	Nginx Version	Helm Chart Version
🔄	**v1.12.0-beta.0**	1.31, 1.30, 1.29, 1.28	3.20.3	1.25.5	4.12.0-beta.0
🔄	**v1.11.3**	1.30, 1.29, 1.28, 1.27, 1.26	3.20.3	1.25.5	4.11.3
🔄	**v1.11.2**	1.30, 1.29, 1.28, 1.27, 1.26	3.20.0	1.25.5	4.11.2
🔄	**v1.11.1**	1.30, 1.29, 1.28, 1.27, 1.26	3.20.0	1.25.5	4.11.1
🔄	**v1.11.0**	1.30, 1.29, 1.28, 1.27, 1.26	3.20.0	1.25.5	4.11.0

Istio Supported Versions



Version	Currently Supported	Release Date	End of Life	Supported Kubernetes Versions	Tested, but not supported
master	No, development only			1.29, 1.30, 1.31, 1.32	1.23, 1.24, 1.25, 1.26, 1.27, 1.28
1.24	Yes	November 7, 2024	~Aug 2025 (Expected)	1.28, 1.29, 1.30, 1.31	1.23, 1.24, 1.25, 1.26, 1.27
1.23	Yes	Aug 14, 2024	~May 2025 (Expected)	1.27, 1.28, 1.29, 1.30	1.23, 1.24, 1.25, 1.26
1.22	Yes	May 13, 2024	~Jan 2025 (Expected)	1.27, 1.28, 1.29, 1.30	1.23, 1.24, 1.25, 1.26
1.21	Yes	Mar 13, 2024	~Sept 2024 (Expected)	1.26, 1.27, 1.28, 1.29	1.23, 1.24, 1.25
1.20	No	Nov 14, 2023	Jun 25, 2024	1.25, 1.26, 1.27, 1.28, 1.29	1.23, 1.24

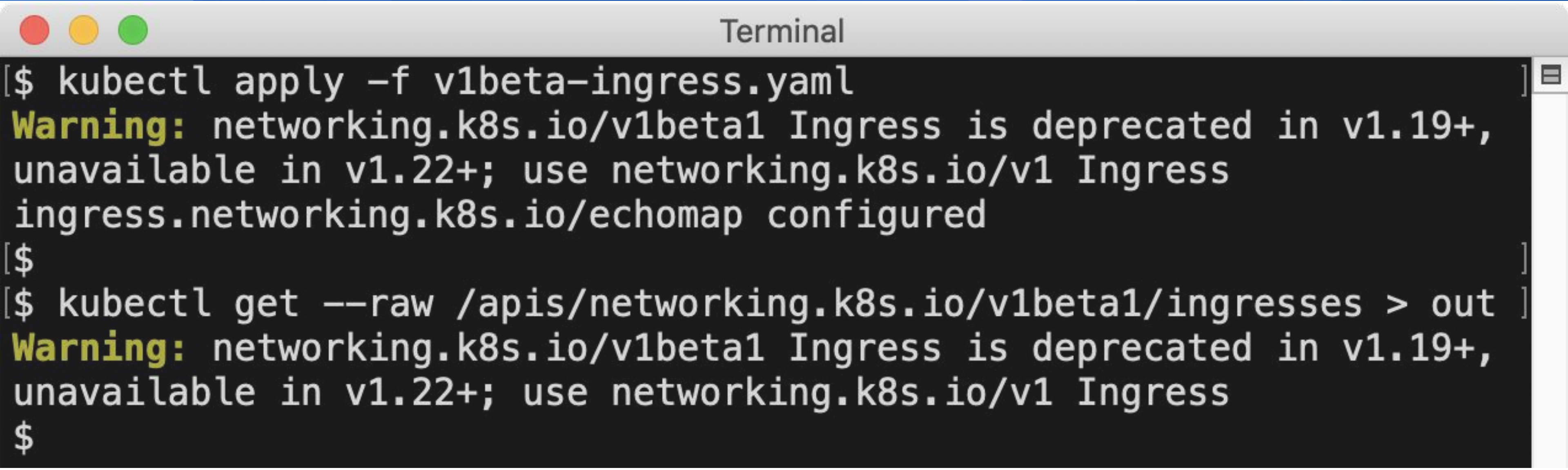
How could I avoid this?

But I Switched Everything to v1

```
apiVersion: networking.k8s.io/v1~~beta1~~
kind: Ingress
metadata:
  name: minimal-ingress
```

- Users may have started using “v1” in their YAML manifests and assumed that was sufficient
- The API version in your YAML manifests is completely unrelated to the API version any controllers are using to implement the API

But What About Deprecation Warnings?



```
$ kubectl apply -f v1beta-ingress.yaml
Warning: networking.k8s.io/v1beta1 Ingress is deprecated in v1.19+,
unavailable in v1.22+; use networking.k8s.io/v1 Ingress
ingress.networking.k8s.io/echomap configured
$
$ kubectl get --raw /apis/networking.k8s.io/v1beta1/ingresses > out
Warning: networking.k8s.io/v1beta1 Ingress is deprecated in v1.19+,
unavailable in v1.22+; use networking.k8s.io/v1 Ingress
$
```

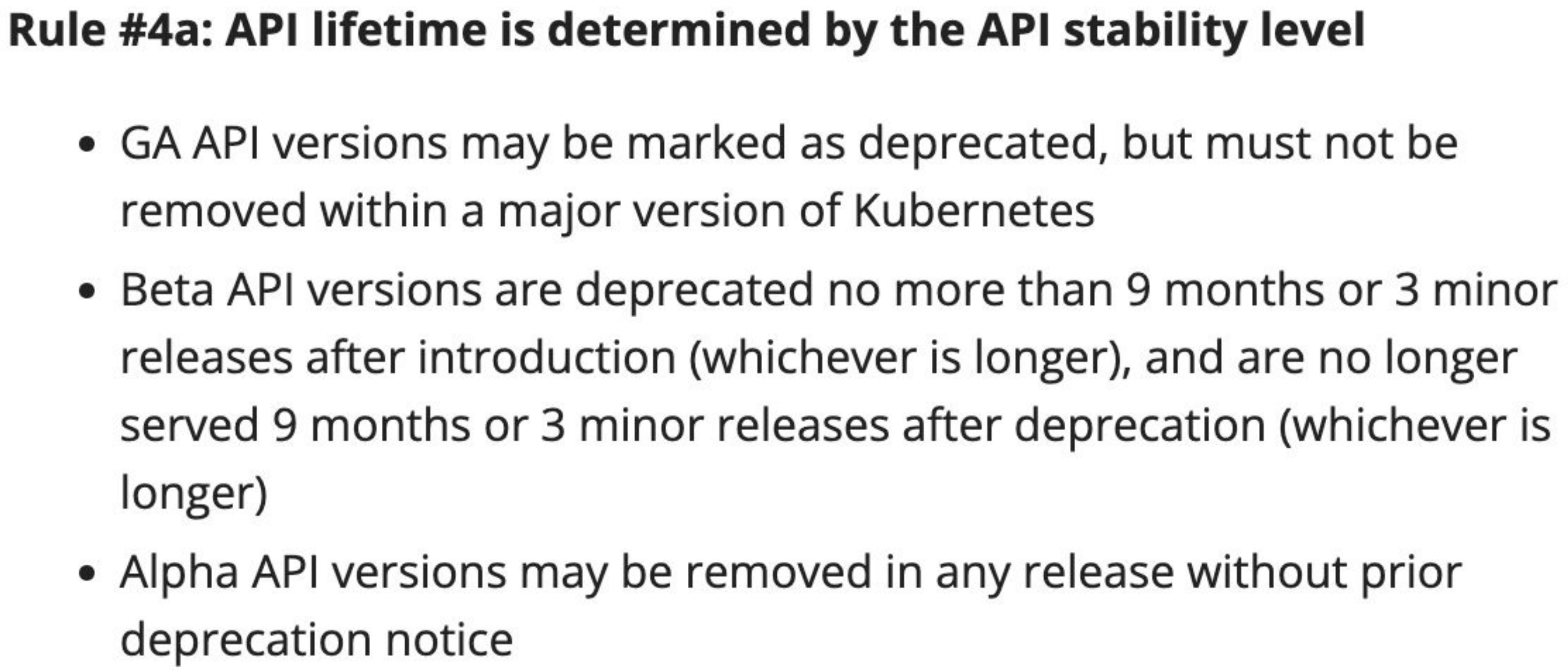
- Deprecation warnings are only visible to the client - you're not going to see deprecations warnings received by a controller

Why Remove v1beta1 So Soon?

	v1.16	v1.17	v1.18	v1.19	v1.20	v1.21	v1.22	v1.23	v1.24
v1beta1	🟩	🟩	🟩	🟩	🟩	🟩			
v1				🟩	🟩	🟩	🟩	🟩	🟩

- There would have never been a completely safe point to remove v1beta1
    - Long tail of users that are using old Ingress controllers
    - Latest release of Istio is tested against the latest 9 Kubernetes versions (3 years)

Kubernetes Deprecation Policy



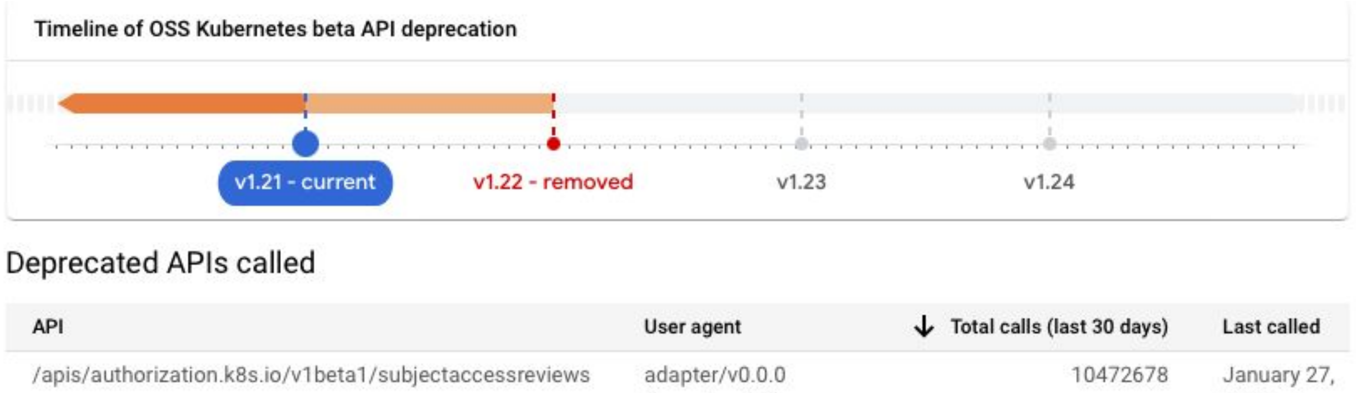
Rule #4a: API lifetime is determined by the API stability level

- GA API versions may be marked as deprecated, but must not be removed within a major version of Kubernetes
- Beta API versions are deprecated no more than 9 months or 3 minor releases after introduction (whichever is longer), and are no longer served 9 months or 3 minor releases after deprecation (whichever is longer)
- Alpha API versions may be removed in any release without prior deprecation notice

https://kubernetes.io/docs/reference/using-api/deprecation-policy/

What Would Have Helped

- Increasing the time between deprecation and removal could have helped to a point, but still won't cover everyone
- Controllers could work to find a way to surface deprecation warnings they receive to users
- Some managed providers like GKE prevent upgrades if they detect usage of APIs that will be removed in the next version


Timeline of OSS Kubernetes beta API deprecation

Deprecated APIs called

API	User agent	Total calls (last 30 days)	Last called
/apis/authorization.k8s.io/v1beta1/subjectaccessreviews	adapter/v0.0.0	10472678	January 27,

OSS Projects That Can Help

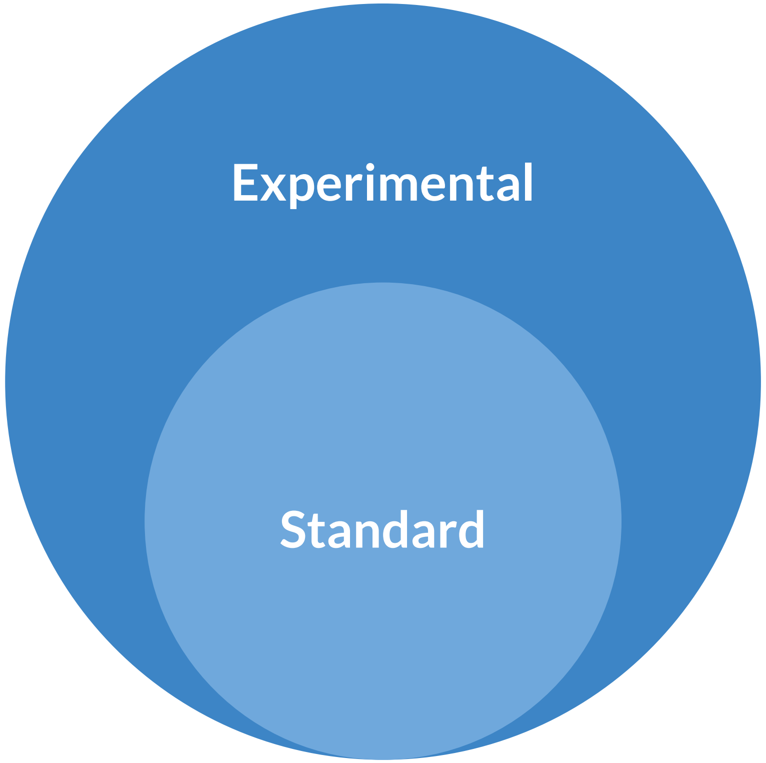
- kubepug/kubepug
- fairwindsops/pluto
- doitintl/kube-no-trouble

Painful Upgrade #2:

GRPCRoute

Gateway API Release Channels

- Experimental Channel
  - Experimental Resources
  - Experimental Fields in Stable Resources
- Standard Channel
  - GA resources and fields
- No breaking changes or deprecations in standard channel ever



Installing Gateway API Standard Channel

```
➜  ~ kubectl apply -f https://github.com/kubernetes-sigs/gateway-api/releases/download/v1.2.0/standard-install.yaml
...

The CustomResourceDefinition "grpcroutes.gateway.networking.k8s.io" is invalid: status.storedVersions[0]: Invalid v
alue: "v1alpha2": must appear in spec.versions
```

What?

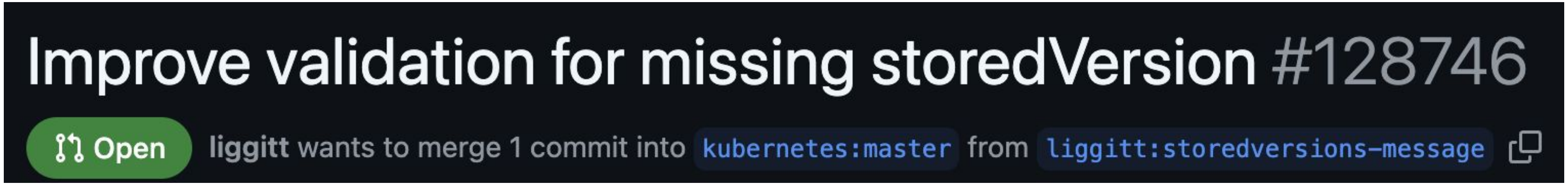
```
The CustomResourceDefinition
"grpcroutes.gateway.networking.k8s.io"
is invalid:
```

```
status.storedVersions[0]:
Invalid value: "v1alpha2": must
appear in spec.versions
```

A Better Message

```
The CustomResourceDefinition "grpcroutes.gateway.networking.k8s.io"
is invalid:
```

`status.storedVersions[0]: Invalid value: "v1alpha2": missing from spec.versions; "v1alpha2" was previously a storage version, and must remain in spec.versions until a storage migration ensures no data remains persisted in "v1alpha2" and removes "v1alpha2" from status.storedVersions`



Improve validation for missing storedVersion #128746

liggitt wants to merge 1 commit into `kubernetes:master` from `liggitt:storedversions-message`

Storage Version

- Every Kubernetes API has a specified "storage version"
- This is the API version used to persist the data in etcd
- CRDs have a `storedVersions` status field
- Kubernetes won't let you upgrade to a CRD that doesn't include a schema for an API version that has been stored

```yaml
status:
  storedVersions:
  - v1alpha2
  - v1
```

But all my manifests say “v1”

KubeCon CloudNativeCon North America 2024

- The API version in manifests is used in the call to API Server
- API Server will still translate that to the storage version configured for the API

```
apiVersion: gateway.networking.k8s.io/v1
kind: GRPCRoute
...
matches:
  - method:
      service: com.example.User
      method: Login
    headers:
      values:
        version: "2"
  - method:
      service: com.example.v2.User
      method: Login
```

So Where Is Storage Version Defined?

- Each CRD can define multiple API versions
- Only one can be marked as the storage version
- If a version has ever been used as a storage version, it will be added to `status.storedVersions`
- This list is not automatically pruned

```yaml
spec:
  versions:
  - name: v1alpha2
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        ...
  - name: v1
    served: true
    storage: false
    schema:
      openAPIV3Schema:
        ...
```

Revisiting the Error Message

KubeCon CloudNativeCon North America 2024

```
The CustomResourceDefinition
"grpcroutes.gateway.networking.k8s.io" is
invalid:

status.storedVersions[0]: Invalid
value: "v1alpha2": missing from
spec.versions; "v1alpha2" was
previously a storage version, and
must remain in spec.versions
until a storage migration ensures
no data remains persisted in
"v1alpha2" and removes "v1alpha2"
from status.storedVersions
```

```
spec:
~~  versions:~~
~~  - name: v1alpha2~~
~~    served: true~~
~~    storage: true~~
~~    schema:~~
~~      openAPIV3Schema:~~
~~        ...~~
  - name: v1
    served: true
    storage: false
    schema:
      openAPIV3Schema:
        ...
```

Migration Steps

1. Ensure your CRD has the desired storage version
2. Update all resources with some kind of no-op update (empty patch)
3. Remove the old version from `status.storedVersions`
4. Upgrade your CRD to remove the old version from `spec.versions`

```yaml
spec:
  versions:
  - name: v1alpha2
    served: true
    storage: false
    schema:
      openAPIV3Schema:
        ...
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        ...
```

StorageVersionMigration

- StorageVersionMigration can help automate step 2 for you
- Alpha in Kubernetes v1.30

```yaml
kind: StorageVersionMigration
apiVersion: storagemigration.k8s.io/v1alpha1
metadata:
  name: grpcroute-to-ga
spec:
  resource:
    group: gateway.networking.k8s.io
    version: v1
    resource: grpcroutes
```

Migration Steps

1. Ensure your CRD has the desired storage version
2. **Update all resources with some kind of no-op update (empty patch)**
3. Remove the old version from `status.storedVersions`
4. Upgrade your CRD to remove the old version from `spec.versions`

```yaml
spec:
  versions:
  - name: v1alpha2
    served: true
    storage: false
    schema:
      openAPIV3Schema:
        ...
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        ...
```

Why is this so Complicated?

- Removing an API version is inherently dangerous - risk of data loss without following these steps
- In Gateway API, we go to extreme lengths to avoid ever having to deal with this in standard channel
    - That means no alpha API versions in standard CRDs ever
    - That means that in some cases migrating from experimental -> standard, will require going through this process if you want to avoid recreating your configuration
- There's a lot of ongoing work to make this less painful

Removing the Upgrade Path

- This migration path is only really necessary if you want to keep your experimental/alpha config as you move to GA API
- Alternatively you could just delete and recreate the CRD if you're ok with recreating your GRPCRoutes
- In Gateway API, there's been a lot of discussion about using different API groups for standard and experimental channel
    - Would remove this storage version migration problem altogether
    - Would allow experimental and standard resources to coexist
    - https://github.com/kubernetes-sigs/gateway-api/discussions/3106

Painful Upgrade #3:

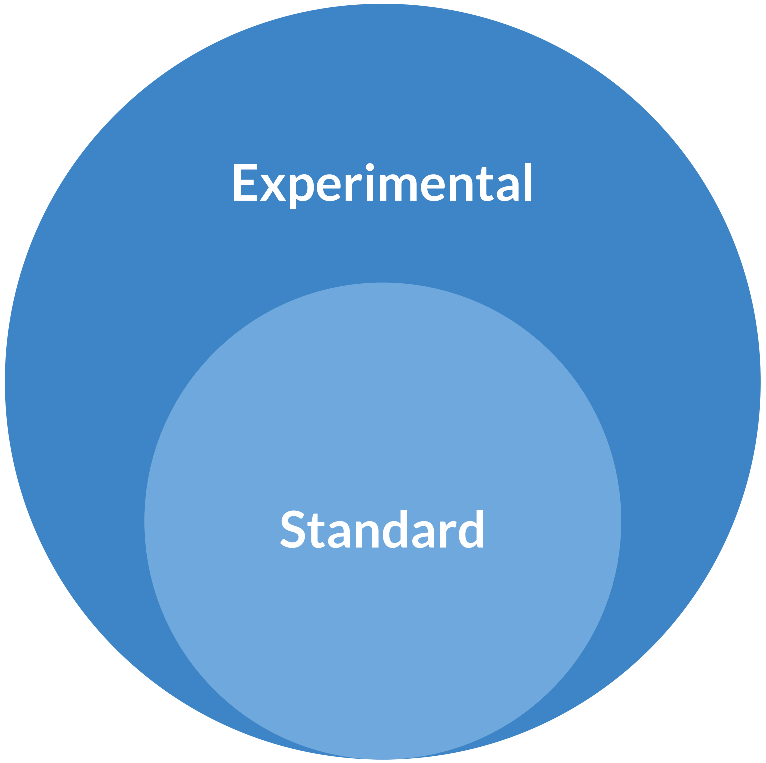
Dropped Fields

Feature Gates

- In k/k, we have feature gates that guard each new field to stable APIs
- Add new field to Service API
    - It’s not safe for new fields to go straight to GA
    - Starts hidden behind alpha feature gate
    - When it meets graduation criteria, it graduates to beta, then GA

Gateway API Release Channels

- To represent that in Gateway API we created “Experimental Channel”
    - All feature gates on
- Standard Channel
    - GA resources and fields only



Upgrades Gone Wrong

- Joe installs Gateway API v1.2 standard channel
- Sue wants to use an experimental authorization feature so installs experimental HTTPRoute CRD
- Joe upgrades to Gateway v1.3 using standard channel
- Experimental authorization feature disappears - everything gets through to Sue’s app

How Do I Avoid This?

- Not unique to Gateway API
  - The same thing could happen if you installed an older version of CRDs
- CRDs are cluster-scoped resources that should be managed centrally
  - Be careful with how many people you allow to manage CRDs
- Communicate
- Use a cluster provider that manages these CRDs for you

Recap

Recap

- Each Kubernetes API can expose multiple API versions (v1alpha1, v1)
    - When a resource is saved, it is persisted with the storage version that is configured at that point in time
    - The API version used in your manifests does not affect the version that is used to store the resource
- CRDs provide infinite flexibility and also some sharp corners
    - Nothing to prevent you from installing an older version of a CRD with less fields
    - Migrating storage versions can be painful

Recap

HTTPRoute API Versions

	v0.5	v0.6	v0.7	v0.8	v1.0	v1.1	v1.2
v1beta1	🟩	🟩	🟩	🟩	🟩	🟩	🟩
v1					🟩	🟩	🟩

- Each controller reads and writes using a specific API version, generally trying to optimize for the widest possible range of supported versions

Recommendations

Recommendations

- Please use alpha API versions, but only if:
  - You're OK with breaking changes that require recreating the configuration or running through a storage version migration
  - You're using them in a non-prod environment
- Keep an inventory of each controller that is relying on an API
  - Ensure those controllers support the new version of the API before upgrading
- CRD management should be centralized
  - A single team should own this
  - Or just let your cluster provider handle this

In Progress

Gateway API CRD Management

- Formalizing CRD management guidelines
    - https://github.com/kubernetes-sigs/gateway-api/discussions/2655
- Provide stronger isolation between release channels
    - https://github.com/kubernetes-sigs/gateway-api/discussions/3106
- Improve CRD management with Helm chart
    - https://github.com/kubernetes-sigs/gateway-api/issues/3288

Storage Version Migration

- Move Storage Version Migrator in-tree
  - https://github.com/kubernetes/enhancements/issues/4192
- Cluster API's Storage Version Migrator
  - https://github.com/kubernetes-sigs/cluster-api/pull/6749

Get Involved

- Weekly community meetings
- Contributors from all backgrounds welcome

gateway-api.sigs.k8s.io

#sig-network-gateway-api

kubernetes-sigs/gateway-api

For API Authors

Still Don't Do What Charlie Don't Does - Making CRD Changes Safer

Nick Young, Isovalent

251 AD

We’re Hiring

Google Cloud

Questions?