
KubeCon
CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022
TiKV
TiKV
HELM

KubeCon CloudNativeCon North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

API Evolution with CRDs: best practices for authoring & fuzz testing APIs

James Munnelly & Andrea Tosatto, Apple

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

James Munnelly
Field Engineer
Apple

Andrea Tosatto
Site Reliability Engineer
Apple

October 24-28, 2021

Designing CRDs

- Designing a “Kubernetes-like” API is easier said than done
- There’s a number of resources available to help
  - A lot still requires intuition and carefully studying conventions

Designing CRDs

- One CRD or many?
  - `objectRef` fields help break up CRDs
  - Can increase complexity of controllers and cognitive overhead for users
  - One concept to one CRD
- Generic example:
  - One CRD modelling the ‘source’ of something (e.g. a CA, a secret store)
  - One CRD modelling a single item within that store (e.g. a Certificate, an ExternalSecret)

Writing a great schema

Writing a great schema

- Once you work out what CRDs you need, writing the schema is next…
- API conventions doc is your reference manual:

  https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-conventions.md
- Join #sig-api-machinery and ask your questions!

Writing a great schema

- Write a complete schema (controller-gen can help!)
  - Required for all v1 CRDs
  - Enables `kubectl explain` output
  - Ensures you have an exhaustive schema
  - Check the `NonStructural` condition on your CRD
  - Better validation: reject fields that are unknown (aka pruning)
    - This avoids typos, and makes it clear to users up front about their errors
    - Matches behaviour of other core built-in resources

Writing a great schema

- Utilise OpenAPI validation within your schemas
  - Allows apiserver to evaluate resource without network roundtrips
  - ***MaxLength***, ***MaxItems***, ***Minimum***, ***Maximum***
    - Helps ensure a bounded size/agreement between from platform on what's persisted
  - Regex for strings
    - Surfaces errors to users earlier on, reduces error handling code in clients
  - Enum values for fields
    - Clear to users what options are valid
  - CEL (**C**ommon-**E**xpression-**L**anguage) for more advanced/expressive validations
    - Can be used for conditional validation logic
    - https://kubernetes.io/blog/2022/09/23/crd-validation-rules-beta/

Writing a great schema

Validation Rule	Purpose
`self.minReplicas <= self.replicas`	Validate an integer field is less than or equal to another integer field
`'Available' in self.stateCounts`	Validate an entry with the 'Available' key exists in a map
`self.set1.all(e, !(e in self.set2))`	Validate that the elements of two sets are disjoint
`self == oldSelf`	Validate that a required field is immutable once it is set
`self.created + self.ttl < self.expired`	Validate that 'expired' date is after a 'create' date plus a 'ttl' duration

Source: https://kubernetes.io/blog/2022/09/23/crd-validation-rules-beta/

Writing a great schema

- Consider default values
  - Embedded in schema and simple is best
    - Allows for the apiserver to apply defaulting on read operations (backward and forward compatibility way easier)
  - Avoid defaults that are co-dependent on other fields
    - These can be confusing and lead to subtle unexpected behaviours

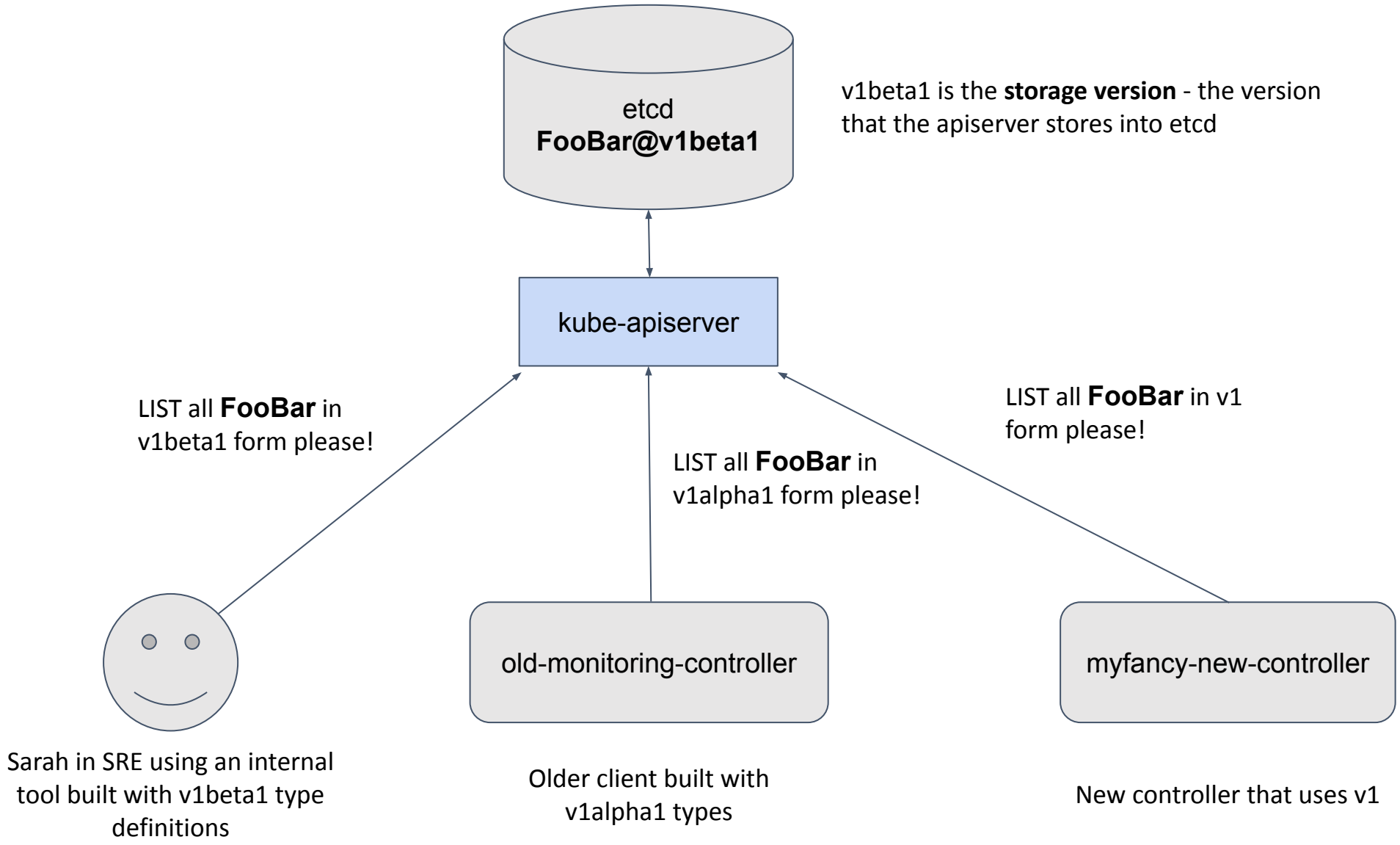
Versioning & conversion

Versioning & conversion

Versioning of Kubernetes APIs is a fundamental principle of how Kubernetes evolves

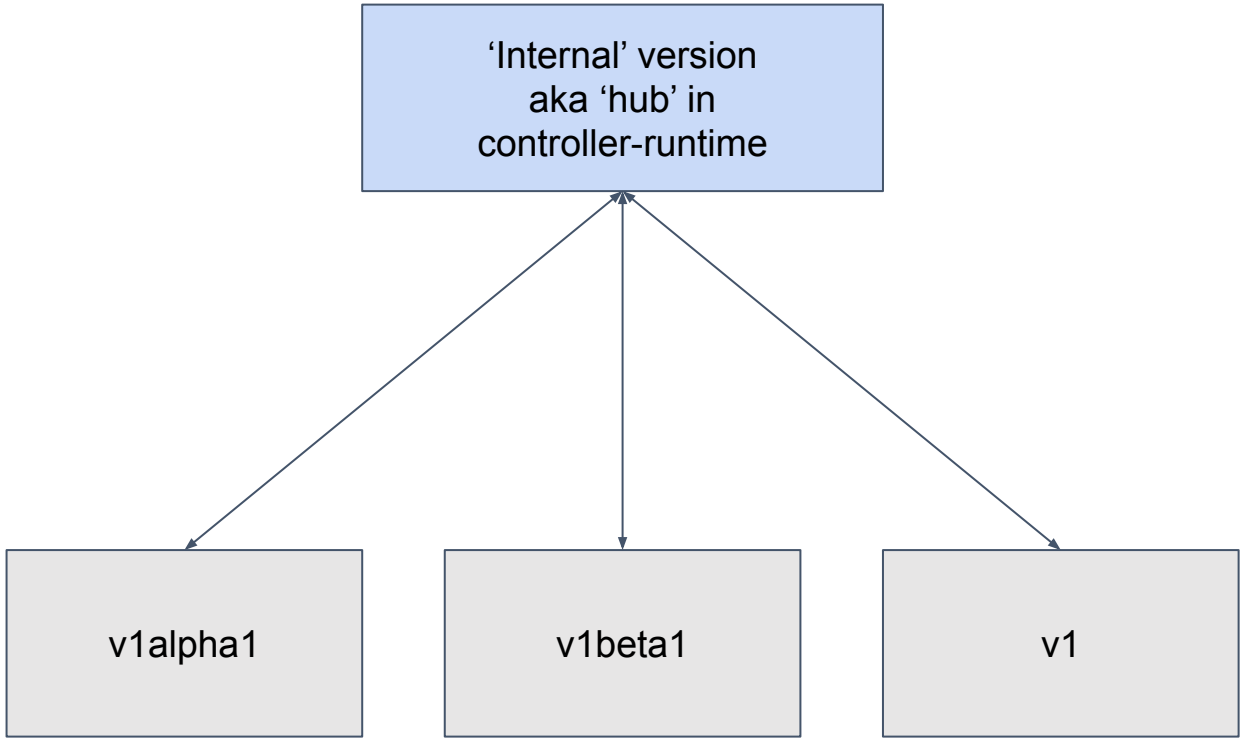
- A version is:
  - An API endpoint that offers backward compatibility
  - A specific view of a resource type stored in etcd
- Conversion enables the apiserver to return any object, regardless of its stored version, in a particular API version.
  - This allows clients built against **older** versions of the API to continue to function when new versions are released
  - Eventually, **non-GA** versions will be removed
  - Full details: https://kubernetes.io/docs/reference/using-api/deprecation-policy/

Versioning & conversion



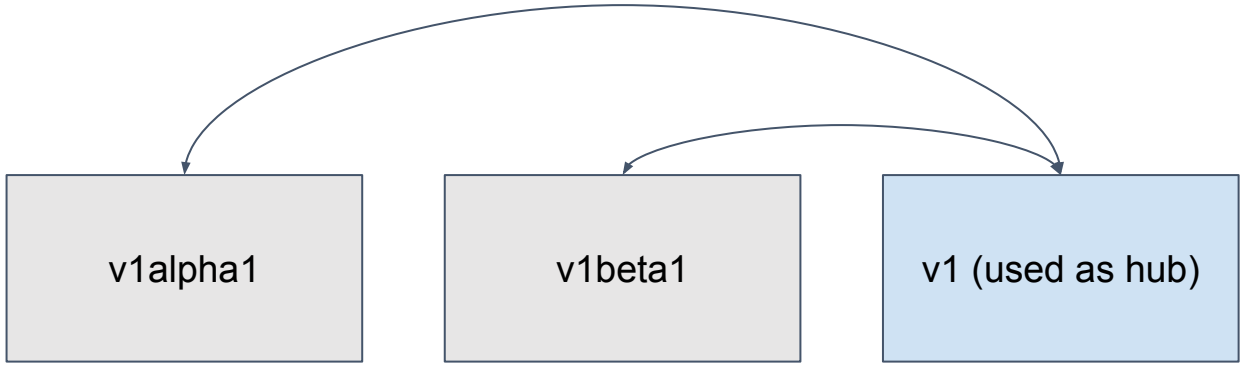
Versioning & conversion

client-go/core k8s



Versioning & conversion

controller-runtime/kubebuilder



Versioning & conversion

So what can we do with CRDs?

Versioning & conversion

- ***Nothing*** - v1alpha1 forever!
- ***No-op conversion*** - try and get it right first time…
- ***Conversion webhooks*** - powerful, full flexibility..
  - Operational risks
  - Maintenance burden
  - TLS/security is very important (what if your conversion webhook *lied*!)
  - This is an extension of your control plane

Versioning & conversion

- ***Webhooks: what did we learn?***
  - A great escape hatch, but easy to get wrong
  - Critical apiserver dependency (garbage collection stops if unavailable)
  - Most organisations don’t have much operational experience
  - Can be subtle versioning issues

Versioning & conversion

- **Key takeaways if you cannot avoid it:**
  - All conversions must be bi-directional and lossless (roundtrip testing ensures this!)
  - Minimise deprecation cycle time - removes the need for webhooks altogether
  - Consider publishing v1-only variant of your app for those that don’t want to run webhooks
  - Should have **NO** external dependencies/inputs

Versioning & conversion

- **What else? What's the future look like?**
  - *kcp* has been exploring conversions through CEL (**C**ommon-**E**xpression-**L**anguage)
  - Allows for in-process conversions with a domain-specific language that hides sharp edges
  - Performant, secure and avoids network round-trips & operational burden
  - Very exciting and interesting work
  - https://github.com/kcp-dev/kcp/pull/2105

Validating & mutating

Validating & mutating

- **Validating webhooks** can be used to apply additional constraints on resources after schema validation is applied
  - Best considered as policy control rather that defining specific properties of types
- **Mutating webhooks** are used to apply additional changes to resources as they are being persisted (either CREATE or UPDATE)
  - Very flexible
  - NOT applied on read operations, unlike defaulting

Validating & mutating

Webhooks:

- Used when more expressive logic is required
- Can be used to add additional validation to any resource type beyond schema based
  - Example: validating Pod resources always set X `priorityClassName`
  - OPA, Kyverno are examples of dynamic policy webhooks
- Have access to full request context to validate & mutate
- Can be used to extract the requesting user, request URI, identify versions etc
- Can be used to prompt users with warnings: https://kubernetes.io/blog/2020/09/03/warnings/

Validating & mutating

Operationalising webhooks:

- TLS
- Scalability
- Availability (make sure platform teams are supporting webhooks)
- Ensuring upgrades are tied to the lifecycle of the CRD
    - Upgrade webhooks **before** CRDs are upgraded

Validating & mutating

What's the future?

- KEP-3488: CEL based admission control KEP

```yaml
# Policy definition
apiVersion: admissionregistration.k8s.io/v1alpha1
kind: ValidatingAdmissionPolicy
metadata:
  name: "replicalimit-policy.example.com"
spec:
  paramSource:
    group: rules.example.com
    kind: ReplicaLimit
    version: v1
  matchConstraints:
    resourceRules:
    - apiGroups:   ["apps"]
      apiVersions: ["v1"]
      operations:  ["CREATE", "UPDATE"]
      resources:   ["deployments"]
  validations:
    - name: max-replicas
      expression: "object.spec.replicas <= params.maxReplicas"
      messageExpression: "'object.spec.replicas must be no greater than ' + string(params.maxReplicas)"
      reason: Invalid
      # ...other rule related fields here...
```

Testing methodology

Testing methodology

- Separate and consider **apiserver** testing vs **controller** testing
- We need unit, integration and end-to-end tests for **both**

Testing methodology

- apiserver's/CRDs:
  - **Unit testing (in-memory, aka `go test`):**
    - Conversion roundtrip testing (`roundtrip.RoundTripTestForAPIGroup`)
      - https://github.com/kubernetes/apimachinery/blob/release-1.25/pkg/api/apitesting/roundtrip/roundtrip.go
    - Schema fuzz tests ensure a complete schema (munnerz/crd-schema-fuzz)
    - Webhooks: unit test your validation & mutation functions
    - **For schema validation**, write a corpus of valid and invalid resources and use a static analysis tool to apply your schema to them: https://github.com/yannh/kubeconform

Testing methodology

- apiserver's/CRDs:
  - **Integration testing (envtest/in-memory etcd & apiserver):**
    - Conversion functions correctly: create A@**v1alpha1**, read at **v1beta1**
    - Defaulting (create a resource and read it back - are defaults set?)
    - Validation (create a resource with an invalid value - does it fail?)

Testing methodology

- apiserver’s/CRDs:
  - **e2e testing (kind/minikube)**
    - Do your deployment manifests work?
    - Correct RBAC configuration, webhook TLS etc

Testing methodology

- controllers:
  - **Unit testing (in-memory, aka `go test`):**
    - Calling `Reconcile` once with a given input and asserting the actions taken
    - Useful for simulating errors and testing error handling behaviour

Testing methodology

- controllers:
  - **Integration testing (envtest/in-memory etcd & apiserver):**
    - Brings up a full reconciler that'll sync multiple times
    - Light-weight with no controller-manager/scheduler
    - Most of your tests will be written like this

Testing methodology

- controllers:
  - **e2e testing (kind/minikube)**
    - Testing end-to-end user facing behaviour
    - Ensures that if a collection of objects is created, the right things happen
    - Write tests that follow your own docs

Closing thoughts

Closing thoughts

- Rely on the schema as much as possible
- Be aware of the cost of webhook sprawl
- There’s lots out there to help with testing nowadays. envtest is fast and powerful.
- Test your error cases, how does your controller behave with a slow cache?

Questions?

KubeCon | CloudNativeCon
North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

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