



KubeCon



CloudNativeCon

Europe 2026

OpenAPI Meets Kubernetes

Auto-Generating CRDs and Operators the Smart Way



Sergiusz Urbaniak - Engineer @ MongoDB



Jose Vázquez González - Engineer @ MongoDB



What are we trying to solve

- Kubernetes is not only a workload scheduler
- Used as a foundation to build platforms
- Used to connect *external systems*
- API-centric use cases becoming more relevant
- The classic "operator" pattern here doesn't work well

Operator types



Workload
Operators

Key idea: “Run something inside Kubernetes”

- Stateful or Stateless workloads
- Databases (MongoDB, Postgres, etc.)
- ML platforms
- Reserve CPU/GPU/Memory/Network



Configuration
Operators

Key idea: “Extend what the cluster can do”

- Certificates
- Policies, RBAC
- Ingress rules



External Resource
Operators

Key idea: “Connect Kubernetes to other systems”

- MongoDB Atlas Clusters
- AWS S3 Buckets
- External DNS Records



External Resource
Operators

External Resource Operators

- **Atlas Kubernetes Operator**
Provision MongoDB Resources on Atlas (SaaS)
- **ACK** (AWS Controllers for Kubernetes)
Provision AWS resources like S3 Buckets, AKS clusters, etc
- **Azure Service Operator**
Provision resources on Azure
- **Crossplane Providers**
Agnostic provisioning of resources on any external service

Prior art - Terraform Wrapping

github.com/crossplane/upjet

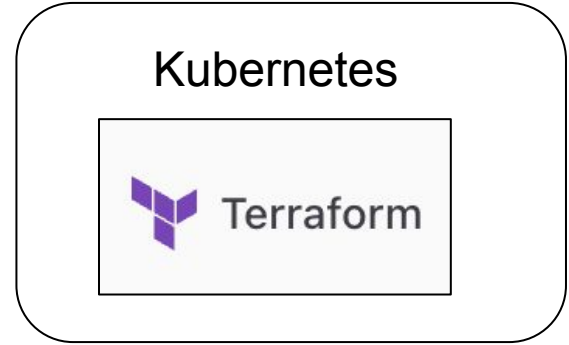
github.com/padok-team/burrito

Pros:

- Leverage existing terraform providers

Cons:

- Terraform providers are not tailored for Kube
- You must have a terraform provider
- Feels "heavyweight" (state file management)



Prior art - API to API translation

github.com/fybrik/openapi2crd

github.com/ant31/crd-validation

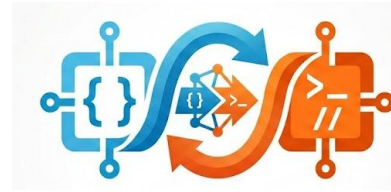
Pros:

- Lightweight
- Leverage existing APIs

Cons:

- Cannot leverage existing "providers"

CRD



OpenAPI
Protobuf
SOAP



External Resource
Operators

So how do we implement one?

- Download controller-runtime
- Spin up kind
- Warm up VSCode

```
func (r *Reconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {  
    // What now???  
}
```

Why Manual doesn't scale

Manual Toil

Every new Atlas feature implies hand-crafted CRDs, controllers, etc

Feature Lag

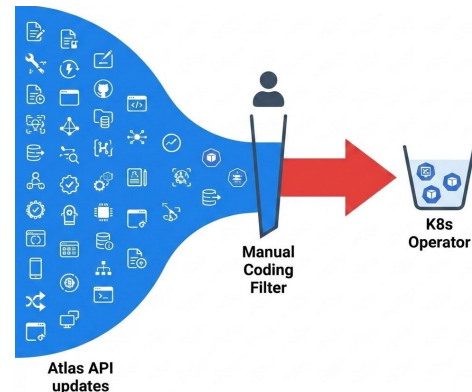
Kubernetes support trails behind the Atlas API releases

Inconsistent UX

Human-in-the-loop development leads to inconsistencies across resources

IDP Integration Demand

Platforms like [Crossplane](#)/[Backstage](#) require 1:1 API parity too expensive to maintain manually



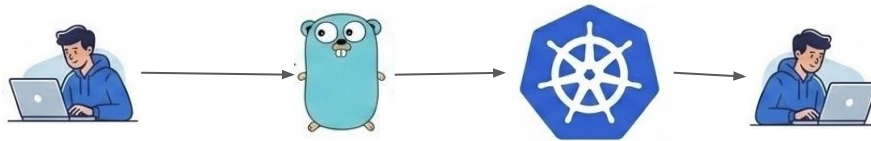
From Manual to Automated

Core Concept

An automated pipeline transforms **OpenAPI Specs** into **Fully Functional K8s Resources**

The Paradigm Shift

- *Old Way:* Go Code → CRD (Kubebuilder)
- *New Way:* OpenAPI → CRD → Go Code + Controllers

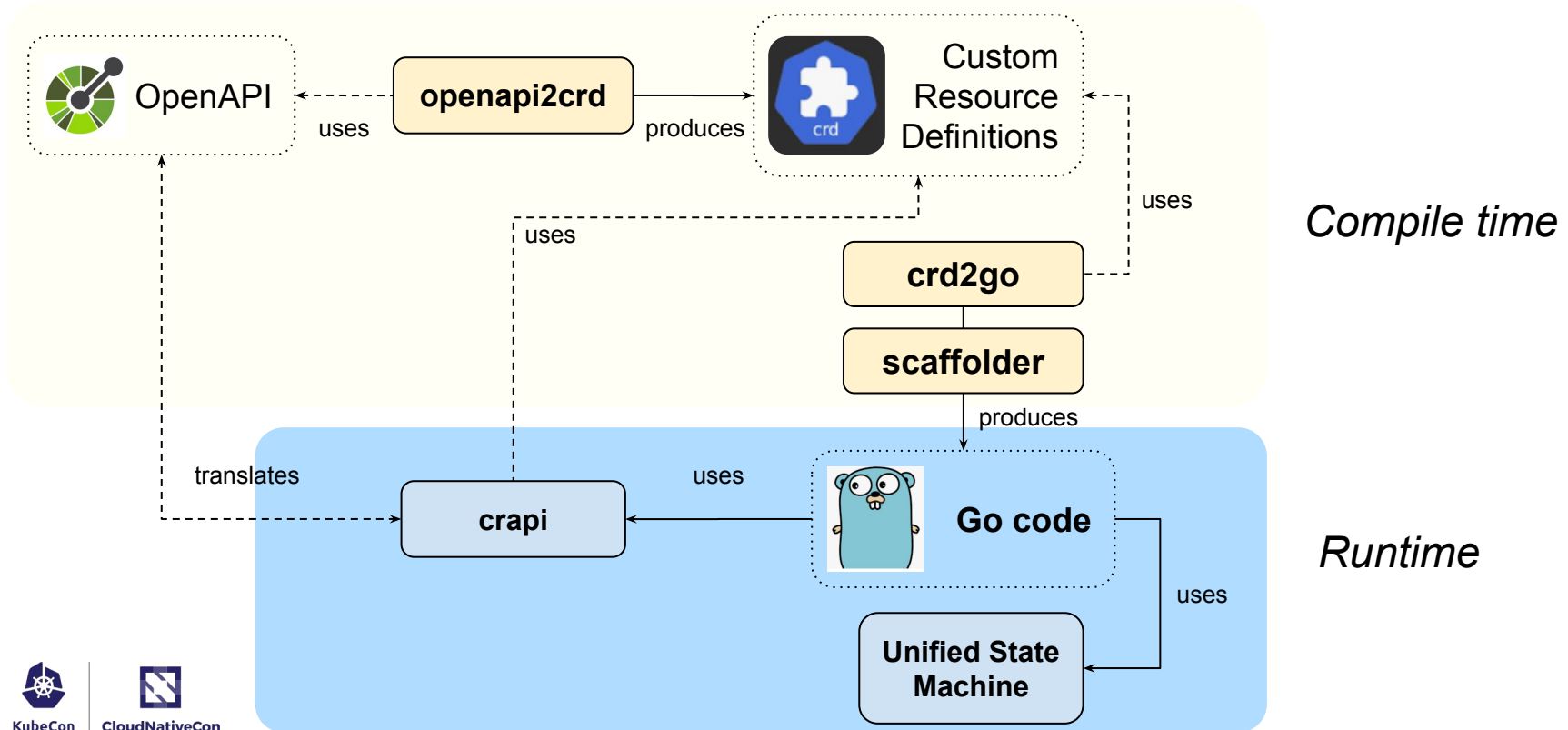


Benefits

- Eliminates the curated abstraction layer
- Achieves 1:1 alignment with OpenAPI



Automation Pipeline

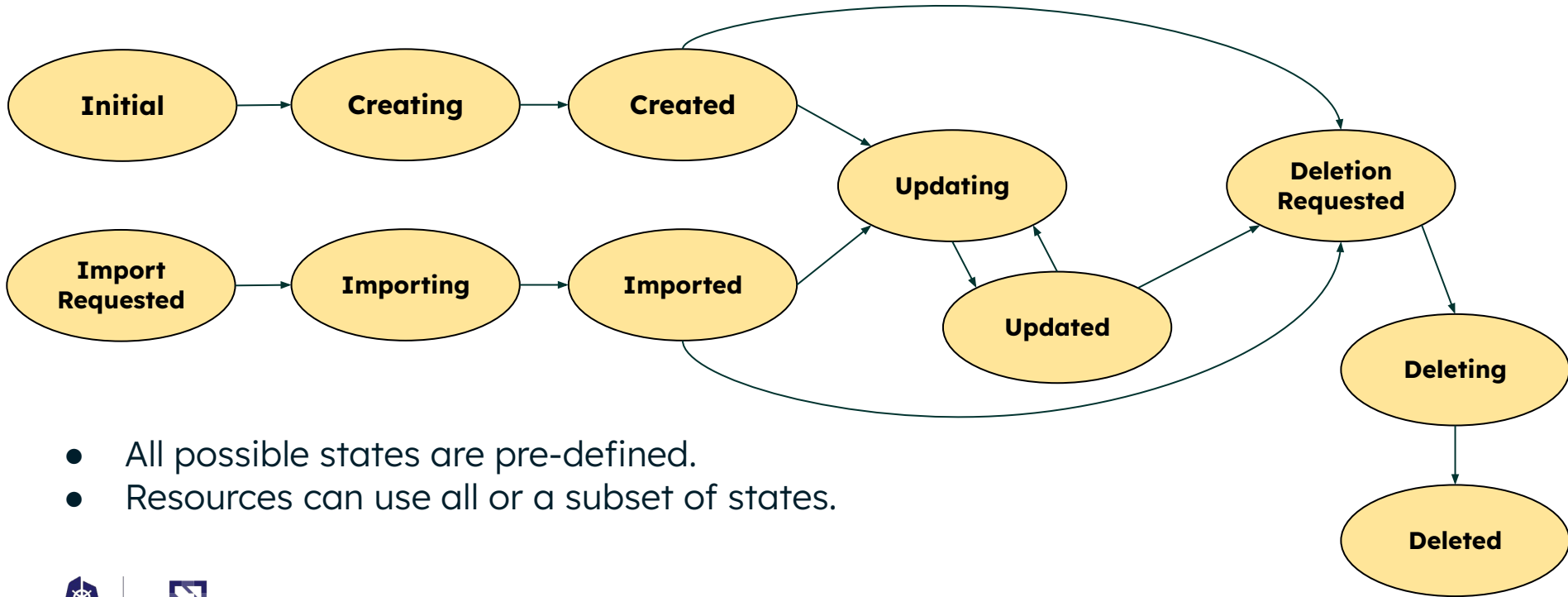


Universal State Machine

- controller-runtime is great but ...
- (consciously) leaves out implementation details

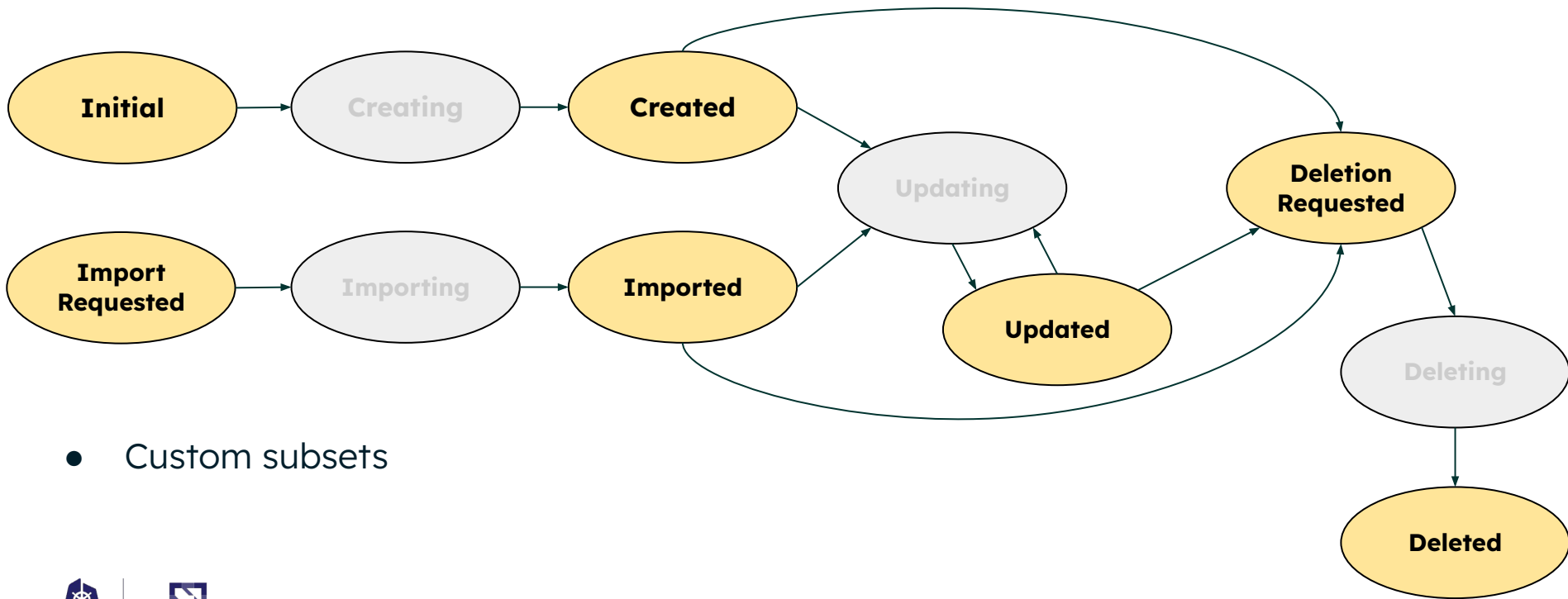
```
func (r *Reconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {  
    // What now???  
}
```

Universal State Machine



- All possible states are pre-defined.
- Resources can use all or a subset of states.

Universal State Machine



- Custom subsets

Universal State Machine

```
func (h *Handler) HandleInitial(ctx context.Context, r *generated.Resource) (ctrlstate.Result, error) {  
    // Create resource ...  
  
    return result.NextState(state.StateCreated, "Resource created")  
}  
  
func (h *Handler) HandleCreated(ctx context.Context, r *generated.Resource) (ctrlstate.Result, error) {  
    // Update resource ...  
  
    return result.NextState(state.StateUpdated, "Resource updated")  
}  
  
func (h *Handler) HandleDeletionRequested(ctx context.Context, r *generated.Resource) (ctrlstate.Result, error) {  
    // Delete resource ...  
  
    return result.NextState(state.StateDeleted, "Resource deleted")  
}
```

openapi2crd

openapispec.yaml

```
components:
  schemas:
    Resource:
      type: object
      properties:
        field:
          type: string
        id:
          type: string
```



config.yaml

```
kind: Config

openapi:
  - name: v1
    path: openapispec.yaml
crd:
  - gvk:
      version: v1
      kind: Resource
      group: atlas.generated.mongodb.com
  ...
```

**./openapi2crd --config
config.yaml**

generated-crd.yaml

```
kind: Resource
apiVersion: my-operator.generated.com/v1

metadata:
  name: foo
  namespace: foo

spec:
  v1
    entry:
      field: "NONE"

status:
  v1:
    id: "67d02fa9a2ae5373a4bb106e"
```



crd2go

CLI tool

Input

- Standard CRD YAMLs from **openapi2crd**

Output

- Go package with CRDs as Go types

Generic & Open Source:

<https://github.com/crd2go/crd2go>

Features

- “Smart” type naming (WIP)
- Custom renames & reservations
- Ad-hoc imports
- [Kubebuilder](#) covers deep copy generation
- Plugins for extra code generation
- Optional apply configuration annotations

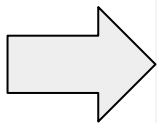
crd2go sample

```
kind: CustomResourceDefinition
metadata:
  name: groups.atlas.generated.mongodb.com

spec:
  group: atlas.generated.mongodb.com

  names:
    categories:
    - atlas
    kind: Group
    listKind: GroupList
    plural: groups
    shortNames:
    - ag
    singular: group
    scope: Namespaced
    versions:
  ...
```

crds.yaml



```
// Code generated by crd2go. DO NOT EDIT.
package v1
```

group.go

```
import (
  ...
)

func init() {
  SchemeBuilder.Register(&Group{})
  SchemeBuilder.Register(&GroupList{})
}

//
+k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object
// +kubebuilder:object:root=true
type Group struct {
  metav1.TypeMeta `json:",inline"`
  metav1.ObjectMeta `json:"metadata,omitempty"`
  Spec GroupSpec `json:"spec,omitempty"`
  Status GroupStatus `json:"status,omitempty"`
}
...
```



KubeCon



CloudNativeCon

Europe 2026

crapi

“CR-API” translates between CRD and OpenAPI Go types

client.Objects (CRD and references, e.g secrets)  An OpenAPI Go type

Evolution:

“Apples to Pears” comparisons → Static manual **Translation Layer** → **crapi**

Properties:

- Is a **Runtime** Library, not a code generator
- ~ marshaller/unmarshaller
- Inputs: Go types plus CRD schema including extensions
 - Extensions allow expanding & collapsing references (**secrets**)
- Outputs: Go types

crapi - sample

```
request := &generated.Resource{}

apiRequest := &api.Resource{}
err := h.translator.ToAPI(apiRequest, request)

if err != nil { ... }
```

ToAPI translates Kubernetes to API structs

```
apiResponse, err := apiClient.Invoke(apiRequest)
if err != nil { ... }

response := &generated.Resource{}
_, err = h.translator.FromAPI(response, apiResponse)
if err != nil { ... }
```

FromAPI translates back to Kubernetes structs

controller scaffolder

CLI tool

Input:

- CRD YAMLs with extensions from **openapi2crd**

Output:

- Go package with controller scaffolding code

Custom code for now

Generates

- Controller scaffolder
 - Fully auto-generated files
 - And files to be completed manually
- Indexers
- Exporters (convert API resources to CRDs)

controller scaffolder - sample

```
jen.Id(refVarName).Op(":=").Op("&").Add(refPkgQual).Values(),
jen.Err().Op(":=").Id("h").Dot("kubeClient").Dot("Get").Call(
    jen.Id("ctx"),
    jen.Qual("...", "ObjectKey").Values(jen.Dict{
        jen.Id("Name"):      jen.Id(fieldAccessPath).Dot("Name"),
        jen.Id("Namespace"):  jen.Id(resourceVarName).Dot("GetNamespace").Call(),
    }),
    jen.Id(refVarName),
),
...
```



```
group := &generated.Group{}
err := h.kubeClient.Get(ctx, client.ObjectKey{
    Name:      cluster.Spec.GroupRef.Name,
    Namespace: cluster.GetNamespace(),
}, group)
...
```

DEMO

The image shows a split-screen view. On the left is the MongoDB Atlas web interface, and on the right is a terminal window.

MongoDB Atlas Interface:

- Header: All Projects | Cloud: MongoDB
- URL: cloud-qa.mongodb.com/v2#/org/655dfe3e401cb80283fe...
- Organization: e2e-s.urbania OK
- Section: All Projects (with a 'New Project' button)
- Message: "No Projects Yet" with an illustration of a laptop and a gear.
- Text: "To get started, create a project. Projects act as secure containers for clusters and resources in Atlas." (with a "Create new project" button)
- System Status: All Good | Last Login: 104.30.164.5 | Version: 85a98d39a5@v20260304
- Atlas Plan: NDS | Effective Plan: NDS | Central URL: https://cloud-qa.mongodb.com
- Organization Name: e2e-s.urbania
- Footer: ©2026 MongoDB, Inc. | Status | Terms | Privacy | Atlas Blog | Contact Sales

Terminal Window:

```
s.urbania@M-6KXKG5VH9C ~->   
  
k9s ~/s/mongodb-atlas-kubernetes  
K9s v0.50.18 kind-kind:kind-kind:v1.35.0 0%::0%  
groups(all)[0]  
NAMESPACE↑ NAME READY REASON STATE  
  
<GROUP>
```



KubeCon



CloudNativeCon

Europe 2026

It's all Open Source!

- openapi2crd: github.com/mongodb/.../openapi2crd
- crd2go: github.com/crd2go/crd2go
- crapi: github.com/mongodb/.../crapi
- scaffolder: github.com/mongodb/.../tools/scaffolder
- Unified State Machine: github.com/mongodb/.../controller/state

Plans

- Currently some parts specific to MongoDB and Atlas SaaS
- Some parts still part of the operator repo (github.com/.../mongodb-atlas-kubernetes)
- Promote all tools to github.com/crd2go
- Depending on community feedback we want to push this further!

Conclusions

- We're looking for feedback is this useful for you?
- API-first approach What is your opinion?
- Inclusion into kubebuilder/controller-runtime Bad idea or good idea?



KubeCon



CloudNativeCon

Europe 2026

