KUBE CAMPUS BY KASTEN

Exploring Kubernetes Custom Resource Definitions

Take your Kubernetes Knowledge to the Next Level!

Thomas Keenan
Sr. Product Marketing Mgr.
Kasten by Veeam

Ali Dowair
Software Engineer
Kasten by Veeam

Dave Smith-Uchida
Technical Leader
Kasten by Veeam

Housekeeping

1. Ask Questions in Q and A window, chat in Chat Window
2. Online Poll – we'd love your inputs
3. Stay until the end - $200 Gift Card
4. We are recording – Slides and Replay will be shared via email
5. Register for Learning labs at KubeCampus.io

KUBE CAMPUS BY KASTEN

Agenda

1. What is a CRD Based API?
2. Examples of CR Based APIs
3. Security, Parallelism and Performance Considerations
4. Implementation Approaches
5. CRD Hands on Demonstration
6. Q and A
7. Gift Card Drawing

KUBE CAMPUS BY KASTEN

Kubernetes evolution

Kubernetes started as a way to deploy and run containers at scale

Configurations were relatively static

- YAMLs once deployed were not changed much
- Applications were not Kubernetes-aware

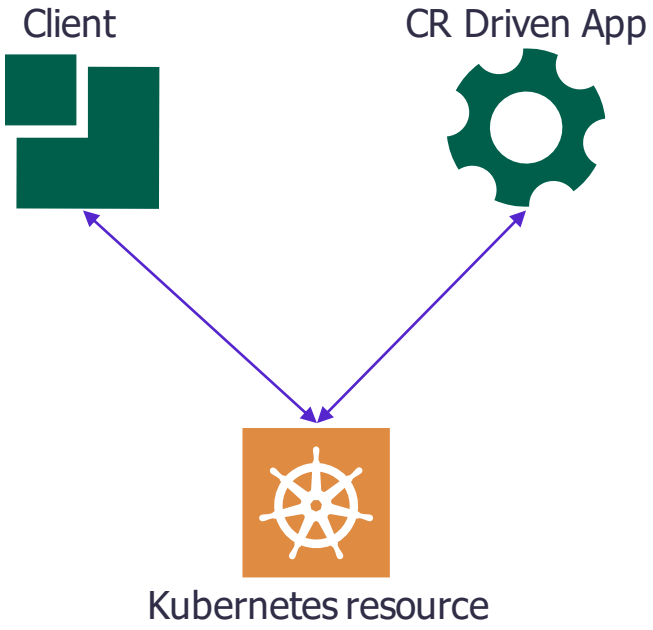
Kubernetes is evolving into more of an operating system for running applications

Applications are being built as Kubernetes applications

- Kubernetes mechanisms are being used to control applications
- Applications can be composed of other applications
- Custom resources are an enabler of Kubernetes Applications

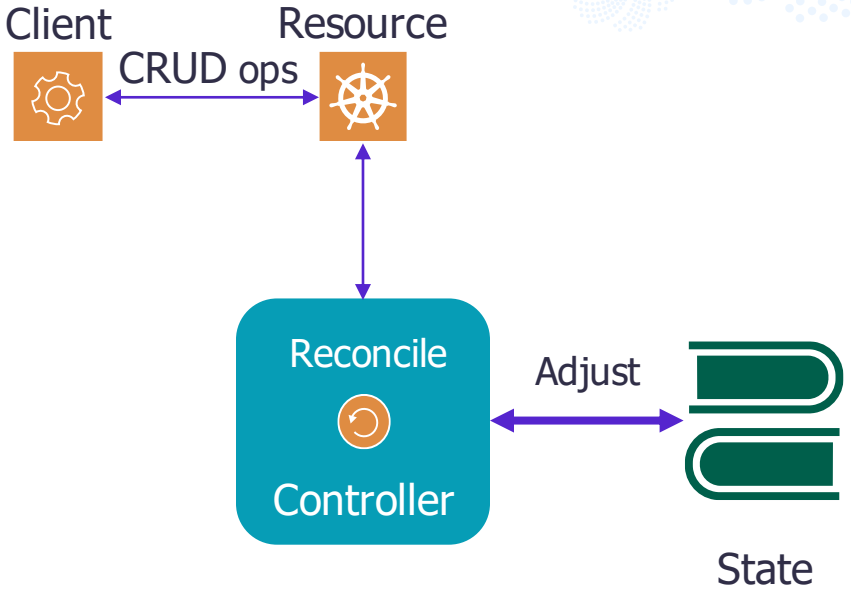
What is a Custom Resource Based API ?

- Application does not provide its own API endpoint (REST, GRPC, etc.) but instead reacts to additions/changes/deletions of resources managed by Kubernetes
- Possible types of resources
  - Custom Resource Definitions (CRDs)
  - Aggregated APIs
  - ConfigMaps



How does a custom resource-based API work?

- General Kubernetes model
- Clients change resource to reflect desired state
- Controller (application) detects difference between desired state and actual state and tries to bring actual state to desired state
- Status of resource is updated in Kubernetes resource (is desired state and actual state the same, was there an error)
- "Relentless forward progress" – when there's an error, keep trying. If the controller restarts, keep trying



Audience Poll
CRD usage

Declarative APIs

This is the Kubernetes Way

Desired state is "declared" by writing it – system is then responsible for reconciling actual state with desired state

Traditional APIs are usually imperative – "do this now"

Example:

- Imperative API – "move box A to shelf 5"
  - When API call is made, system moves box A. If there's a problem, an error is returned. When the API call completes, box A is on shelf 5
- Declarative API – "box A is on shelf 5"
  - When API call is made, declared shelf of box A changes to 5. API call is now complete. System tries to reconcile actual shelf of box A to 5. If unable to, continues to retry.

Actions masquerading as resources

```
apiVersion:
”boxes.example.com/v1"
kind: MoveBoxAction
metadata:
  name: move-1234
spec:
  box: A
  shelf: 5
status: Succeeded/Failed
```

- What’s wrong with this?
- What order do the actions get executed today?
- Where’s the box? Needs additional resource to discover
- Why are there so many “Move Box” resources lying around? Who is cleaning them up?
- No judgement – I’ve done this
- Why? Didn’t want to add an API endpoint, needed a publish/subscribe mechanism, users wanted to interact using kubectl, etc.

A proper declarative API design

```
apiVersion:
”boxes.example.com/v1"
kind: Box
metadata:
  name: a
spec:
 shelf: 5
status:
  state:Moving/Retrying/Settled
  shelf: 5
```

Why is this better?

- Name of resource identifies Box
- Last writer wins in order of execution – no need to figure out order of operations
- No need for garbage collection
- One resource represents location of box and handles changes in location of box
- Reconciliation is just checking differences between actual state and declared state

Security

Standard Kubernetes RBAC permissions can be applied to Custom Resources.

Pros:

- Existing security model
- Authentication handled by Kubernetes API server

Cons:

- Users need to have Kubernetes API server access
- Kubernetes security not strong enough to allow external users access
  - Must give Kubernetes credentials to user
  - Could use gateway, but then security is responsibility of application again

Security usage

Some common patterns:

Control access to namespace where CRs live

- Users with read/write access can use the app
- Easy to implement, fairly coarse control

Create RBAC rules on specific resources

- Users can only read/write their resources
- Users need to know the resources, can’t list

Namespace per user

- User can read/write resources in their namespace
- Application needs to monitor multiple namespaces
- Be careful of privilege escalation paths if some shared resources lie in the controller’s namespace

Scale/Performance considerations

Scale

- Adding large numbers (>1000) of resources to Kubernetes api server not recommended
  - Aggregated API server can get around this
- Large numbers of clients not recommended

Performance

- Request/response turns into
  - write CR
  - etcd write
  - controller wakeup
  - (work)
  - CR status update
  - etcd write
  - client wakeup

Advantages & Disadvantages of CR based APIs

Pros

- Leverage K8s concepts & services:
  - RBAC
  - CRUD Operations built-in
  - Control from kubectl, K8s APIs
- Scalability
  - Easy to have multiple instances of app running
- Reliability
  - App does not need to be running for resources to be accessible

Cons

- Application is tied to Kubernetes
- Declarative API not always suitable
- Large number of resource (>1000) not recommended for K8s API Server
- Performance

Two ways to implement a K8s Resource based application

CRDs and Controllers

- Schema for resources is registered in Kubernetes with Custom Resource Definitions
- Application (Controller) reacts to changes in resources and keeps state in resources

Aggregated API server

- Endpoint for resource type is registered with Kubernetes API server
- Operations on that resource type are delegated to the application (aggregated API server)

Key difference:

- CRDs are easier to implement, aggregated API server better for large numbers of resources, performance

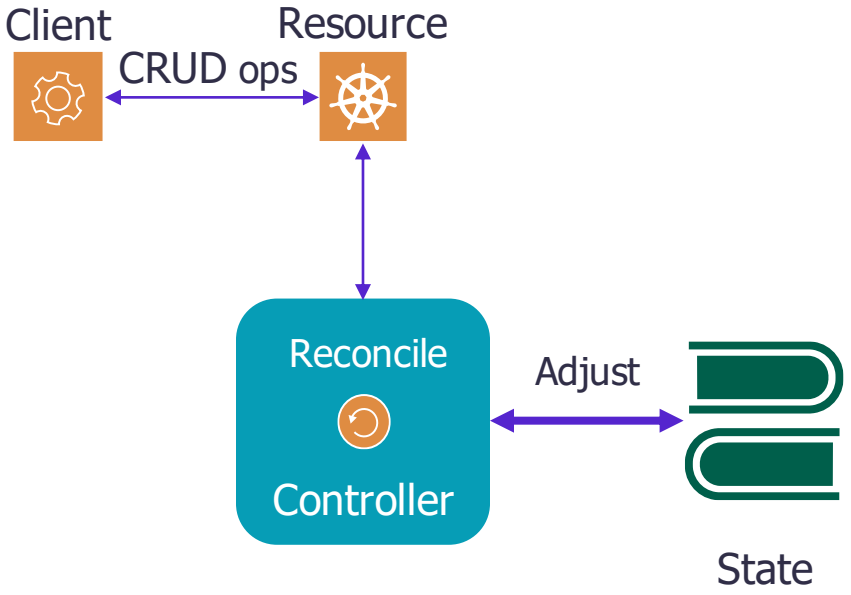
Controller/CRD application

Most common model for CR based applications

Custom Resources are stored by Kubernetes API server

Controller(s) reconcile desired state with actual state

- Persistence, storage, API endpoints handled by Kubernetes
- Client and application fully disconnected – if application crashes, no actions needed on client side



CRD Demo

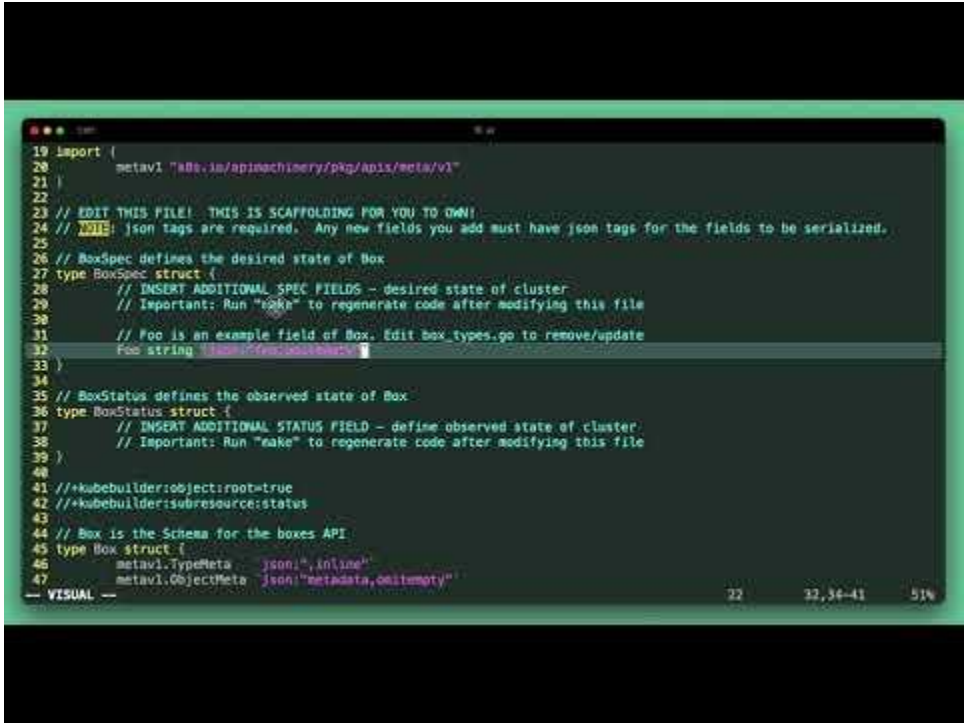
Options to build CRD APIs

1. Kubebuilder
   - Open-source tool
   - Quickly develop and deploy CRD APIs and webhooks
2. Operator SDK
   - Also open-source, built on top of Kubebuilder
   - Integrates with Operator Lifecycle Manager
   - End-to-End testing framework
   - Best practices scorecard

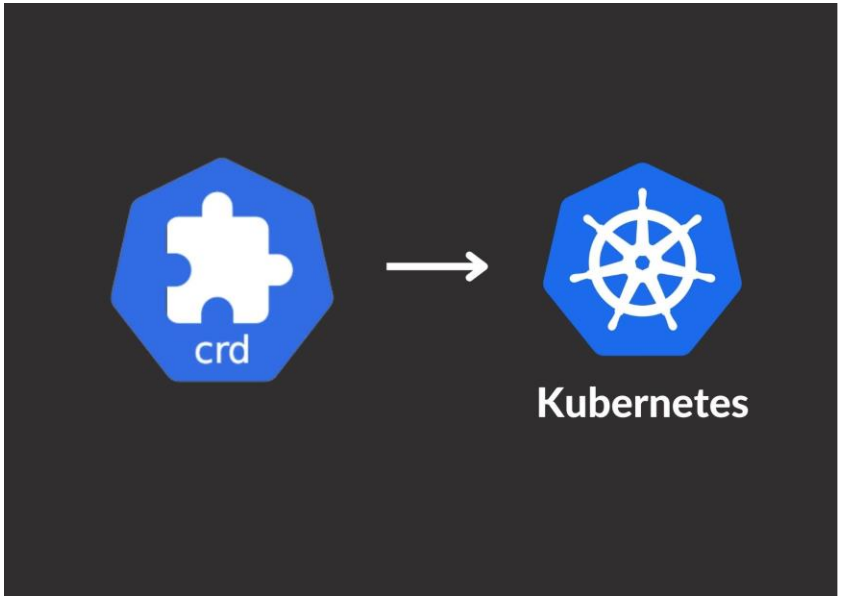
Kubebuilder: steps

1. Init go module
2. Init Kubebuilder project
3. Create CRD, controller scaffolding
4. Define CRD type (spec and status)
5. Implement controller (reconciler) logic
6. Install CRD and run/deploy controller

Demo Recording



General conclusions



1. CR driven apps are mostly used for Kubernetes controlled resources and system utilities
2. Suitable for regular applications with these conditions:
   - Dependency on Kubernetes is acceptable/desirable
   - Client access to Kubernetes is acceptable/desirable
   - Operations can be fit into a declarative model
   - Relatively low volume of operations (10s to 100s ops/sec; not 1000+)
   - Total number of resources to be represented should be in the 1000's range

Learn More

Join us @ KubeCampus.io

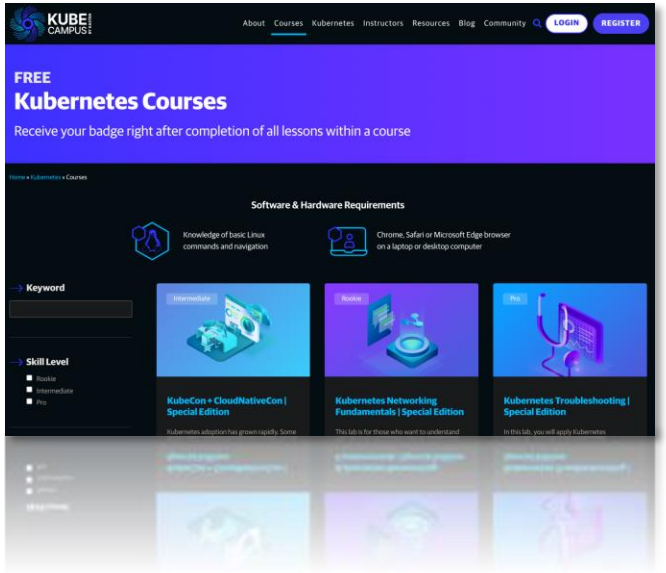
Addressing community input for more Kubernetes education

A **FREE** resource welcoming all learner levels

Self-paced, hands-on labs covering Kubernetes fundamentals, backup and DR in Kubernetes

Courses Include:

- Kubernetes Principles
- Understanding Applications in Kubernetes
- Understanding Security in Kubernetes
- Understanding DR in Kubernetes
- Kubernetes Management and Observability



Expert Kubernetes Instructors & Community Interaction

KASTEN by Veeam

Courses

Introduction

KUBE CAMPUS BY KASTEN

Hardware, software and knowledge requirements

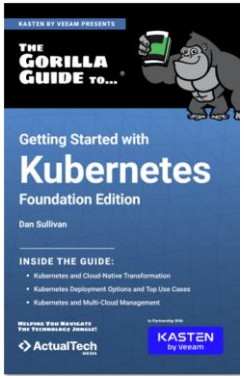
- Knowledge of Basic Linux commands and navigation
- Laptop with 4 GB of memory and 20 GB of hard drive available
- Windows 10 , Mac OS, Linux
- Chrome, Microsoft Edge, Chromium Browser, Safari

Courses Structure

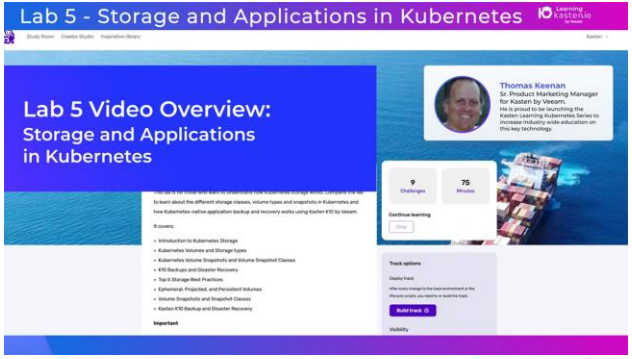
- Review of concepts from pre-work including blog, ppt, VOD and Kasten K10 docs for advanced users
- Hands on lab, following specific Kubernetes commands to achieve mastery and success
- Badging and added resources awarded for each course completed

Insightful Content

Ebooks, Blog Posts, Webinars



Worth the Fight: An Engineer's Path to Kubernetes Proficiency – Part 1 of a 2 part series



Ebooks

- Gorilla Guides – Getting Started, Storage, Security, Observability
- Dummies Guide

Blog Posts

- Diverse topics of interest to Kubernetes Community
  - Application Consistency, OIDC, Networking, Security, Minikube
- Beginner and Pro Journeys
- Submit yours today! Contact@KubeCampus.io

Video Recordings and Webinars

- VODs accompany each Lab
- Live Webinars on Security, Storage, Application Monitoring and other topics
  - Also available on demand

KUBE CAMPUS BY KASTEN

New Promotions

- **Refer a Friend by March 17**:
  - ✓ Sign in with your KubeCampus account, get your referral code and share it with your buddies. If your referred friend registers for KubeCampus, you both will have the chance to win a $100 gift card.
  - ✓ **BONUS:** The learner who gets more friends registered (only new registrations accepted) will automatically win a $100 gift card!

https://kubecampus.io/referral-program/

KUBE CAMPUS BY KASTEN

Questions ?

The $200 gift card winner is...

Thank You