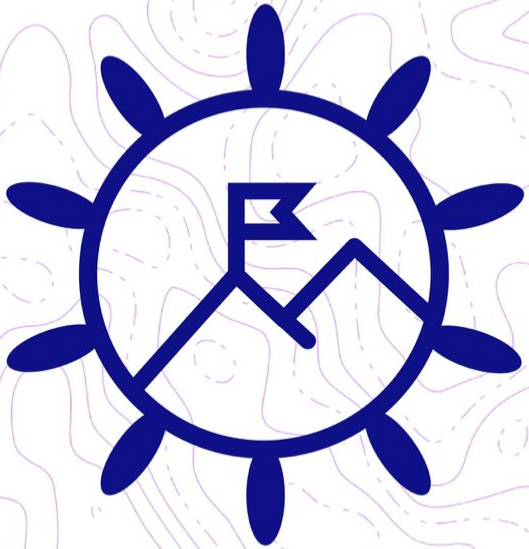
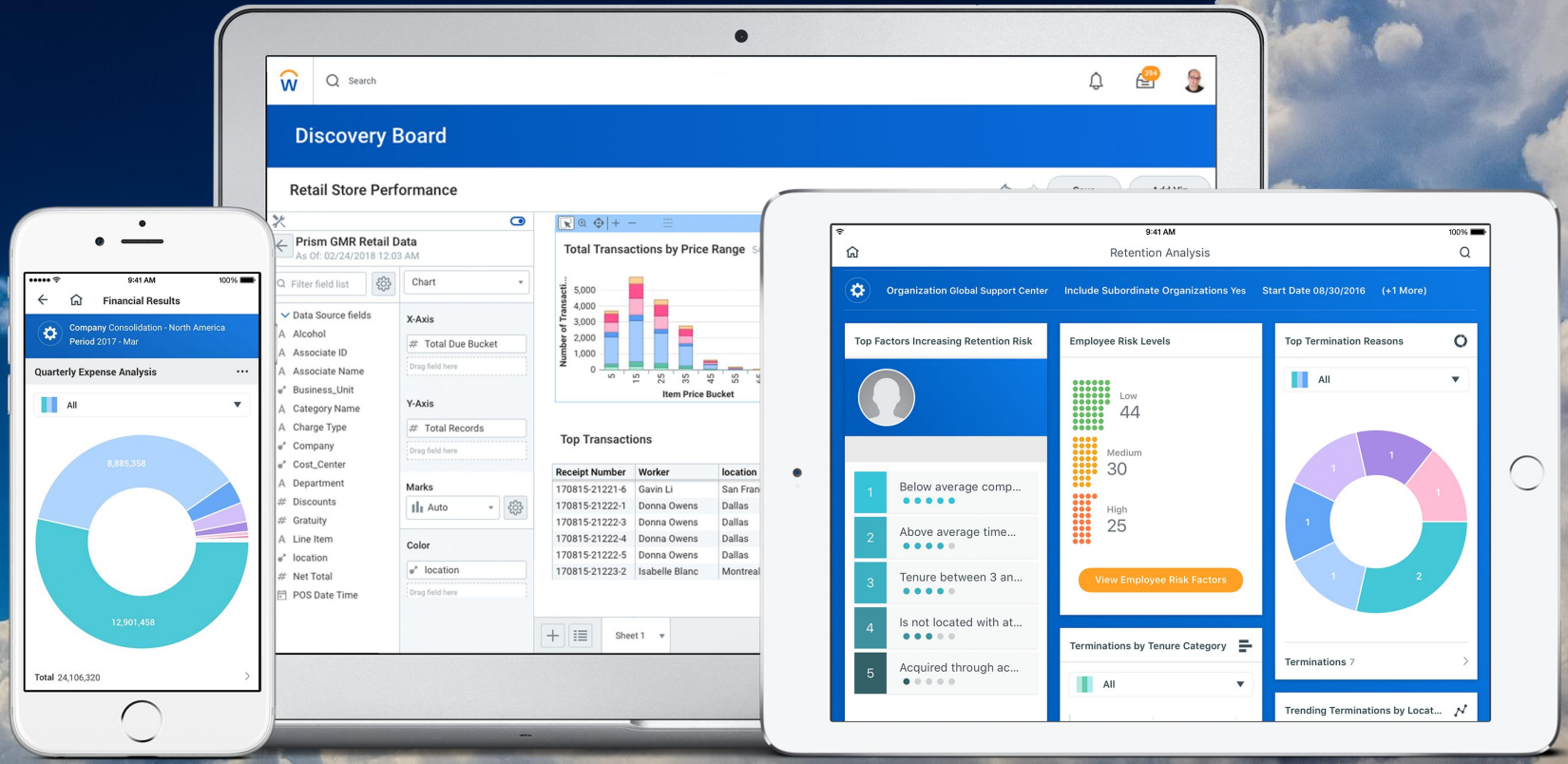


HELM SUMMIT

Automated Helm Deployments Using Custom Controllers & Operators

Pauline Lallinec, Workday Public Cloud

Helm Summit EU, September 2019


Search
Discovery Board
Retail Store Performance
Prism GMR Retail Data
As Of: 02/24/2018 12:03 AM
Filter field list
Chart
Data Source fields
Alcohol
Associate ID
Associate Name
Business_Unit
Category Name
Charge Type
Company
Cost_Center
Department
Discounts
Gratuity
Line Item
location
Net Total
POS Date Time
X-Axis
Total Due Bucket
Y-Axis
Total Records
Marks
Auto
Color
location
Total Transactions by Price Range
Number of Transacti...
Item Price Bucket
Top Transactions
Receipt Number
Worker
location
170815-21221-6
Gavin Li
San Fran
170815-21222-1
Donna Owens
Dallas
170815-21222-3
Donna Owens
Dallas
170815-21222-4
Donna Owens
Dallas
170815-21222-5
Donna Owens
Dallas
170815-21223-2
Isabelle Blanc
Montreal
Sheet 1
9:41 AM
100%
Financial Results
Company Consolidation - North America
Period 2017 - Mar
Quarterly Expense Analysis
All
8,885,358
12,901,458
Total 24,106,320
9:41 AM
100%
Retention Analysis
Organization Global Support Center
Include Subordinate Organizations Yes
Start Date 08/30/2016
(+1 More)
Top Factors Increasing Retention Risk
1
Below average comp...
2
Above average time...
3
Tenure between 3 an...
4
Is not located with at...
5
Acquired through ac...
Employee Risk Levels
Low
44
Medium
30
High
25
View Employee Risk Factors
Terminations by Tenure Category
All
Top Termination Reasons
All
Terminations 7
Trending Terminations by Locat...

What is Workday?

2800+
Customers

40 million
users

100 billion
transactions*

* FY 2019

Workday + Public Cloud = Scylla 🦑

Amazon AWS (US, EU, Canada) 🌍

3 teams, 2 continents 👩‍👧‍👧

👩‍💻 Software Engineer - DevOps

🎤 Non-stop karaoke machine

@plallin


The need for Helm release automation

The need for Helm release automation

Problem
Need to ship more Kubernetes resources

Solution
Helm for Kubernetes packaging + versioning

Next
Deploy Helm releases reliably

The need for Helm release automation

Priority: reliability

Solution
Script handling upgrades & automatically rollbacks failed releases

The need for Helm release automation

Priority: reliability

Solution
Script handling upgrades & automatically rollbacks failed releases

Problems

- Lack of automation
- Does not scale
- Additional server maintenance (Jenkins)

The need for Helm release automation

Requirements

- Automation
- Reliability
- Observability

The need for Helm release automation

Requirements

- Automation
- Reliability
- Observability

Solution

An in-cluster service to manage all incoming Helm releases

Choices:

- Own custom controller
- Flux Helm Operator

Building your own custom controller | operator

What is it?

- Your own Kubernetes controller
- Running your own logic

Building your own custom controller | operator

Pros	Cons
Your own code, with ability to add custom features and logic	Your own code, with responsibility to maintain and extend it
Can manage non cloud-native services	Initially, need to commit time to develop it
Automated rollback	
Control over delivery	

Installing Flux Helm Operator

What is it?

- Operator offered by WeaveWorks
- Weave Flux: CI/CD for Helm charts
- Flux Helm Operator: Helm release manager

Installing Flux Helm Operator

Pros	Cons
Someone else's code, benefitting from community inputs	Someone else's code
Open source & community-driven	In most companies, need to go through security review / approval process
Production-ready	Updates subjected to external PR review & approval
Regularly updated by Fluxcd + community	No control over delivery

Provide optional rollback support for HelmReleases #2006

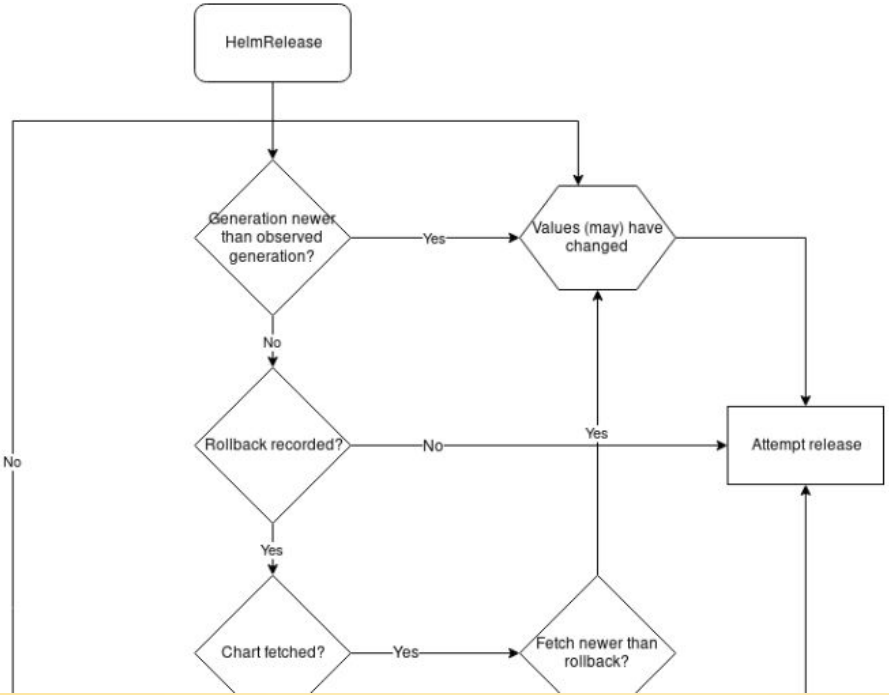
New issue

Merged **hiddeco** merged 7 commits into `master` from `helm/1960-rollback` on 1 Jul

Conversation 27 · Commits 7 · Checks 2 · Files changed 10 · +535 −197

hiddeco commented on 2 May • edited · Member

Fixes #1960



Reviewers
- 2opremio
- adrian ✓
- squaremo ✓

Assignees
No one assigned

Labels
enhancement
helm

Projects
None yet

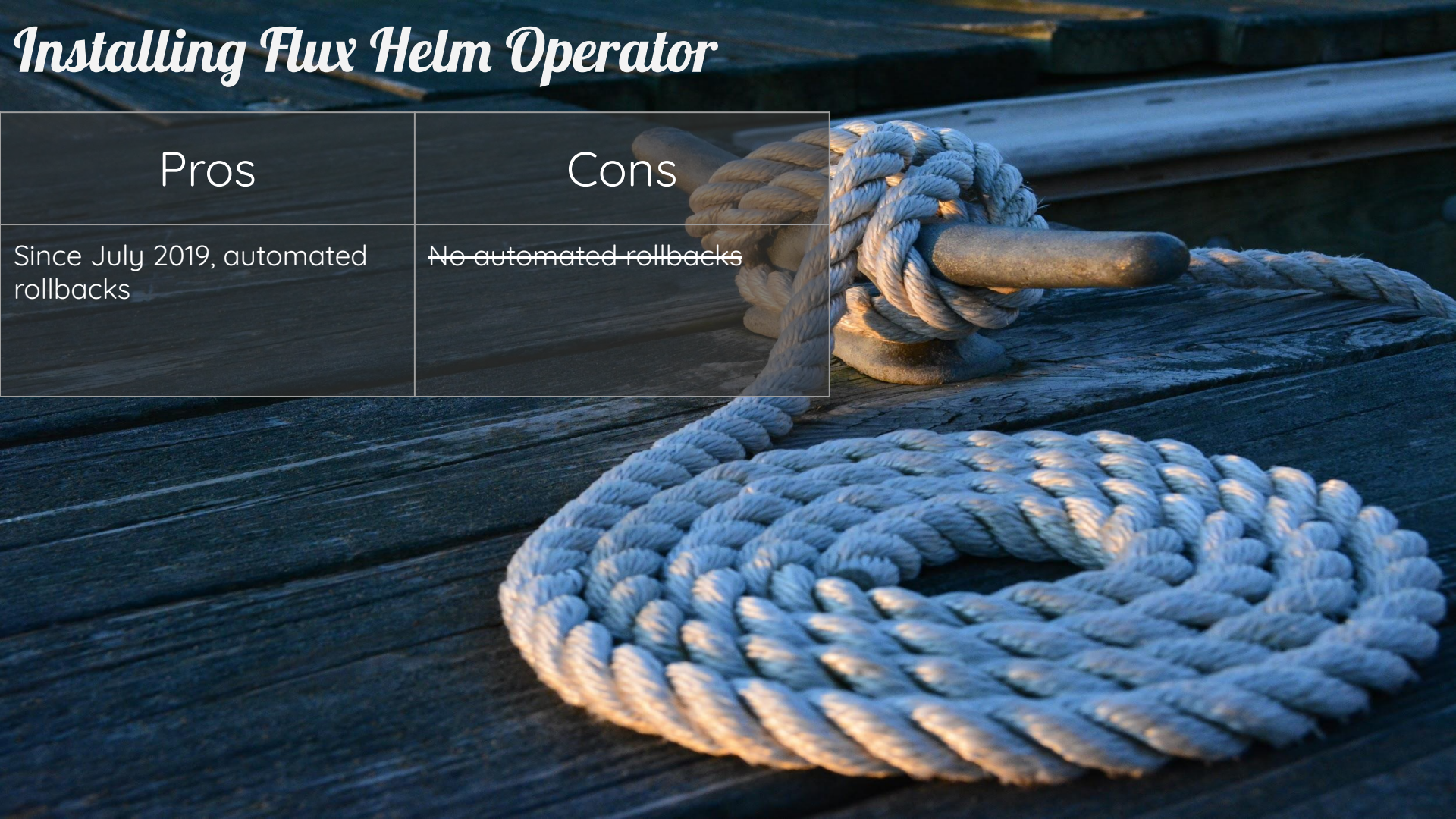
Milestone
No milestone

5 participants


Provide optional rollback support for HelmRele s #2006
New issue
Merged
hiddeco merged 7 commits into master from helm/19 ollback on 1 Jul
Conversation 27
Commits 7
Checks 2
+535 -197
hiddeco commented
Fixes #1960
2opremio
assigned
helm
Milestone
No milestone
5 participants
No
Yes
Fetch newer than rollback?
Thank you!
Fluxcd
hiddeco (Hidde Beydals)
2opremio (Alfonso Acosta)
squaremo (Michael Bridgen)
Workday
adrian (Adrian Smith)

Installing Flux Helm Operator

Pros	Cons
Since July 2019, automated rollbacks	~~No automated rollbacks~~



Custom resources & Custom controllers

Custom resources

A way to create custom objects that live within your cluster, and are handled by a custom controller running a logic of your own.

(Ideally) CRDs responds to CRUD events (Create, Read, Update, Delete) and allow you to implement your own declarative API.

Custom resource definitions

Standalone CRDs

- Custom object with their own API endpoint
- Store / retrieve structured data

CRDs + Custom controllers

- Declarative API

Example: custom resources

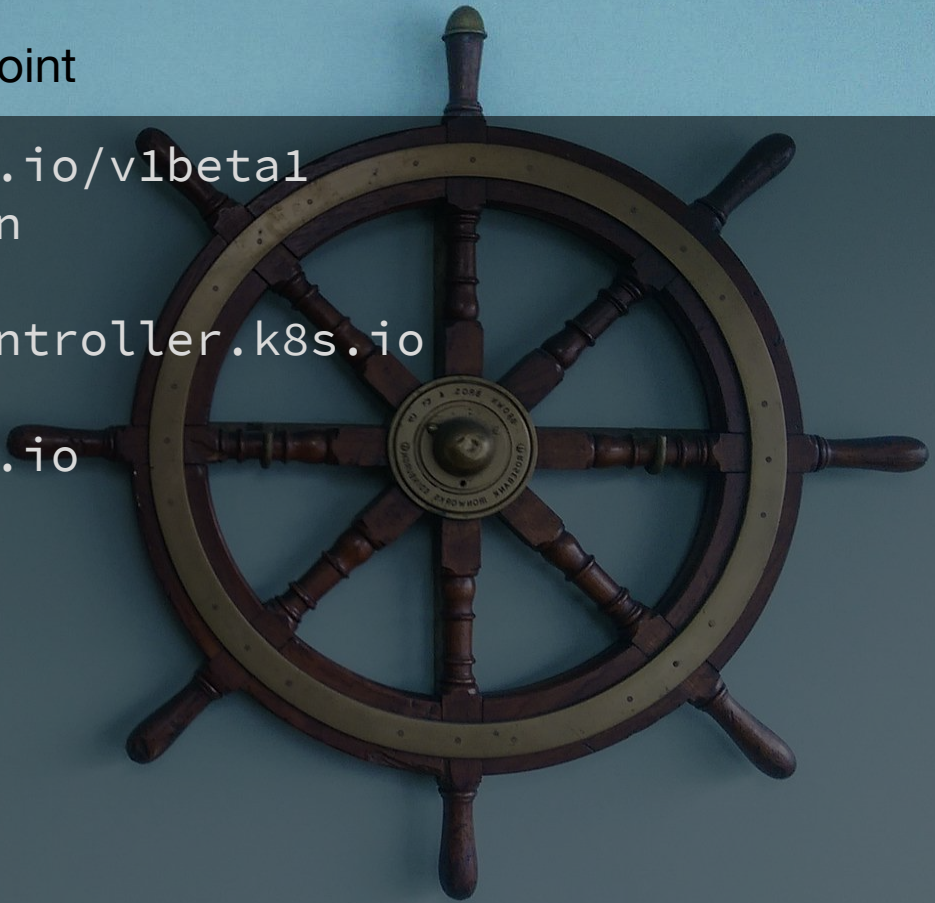
Helm Releases

- Object type: HelmRelease
- Object definition:
  - Release name
    The name of the application
  - Release version
    The version of application

Custom resource definitions

Custom object with their own API endpoint

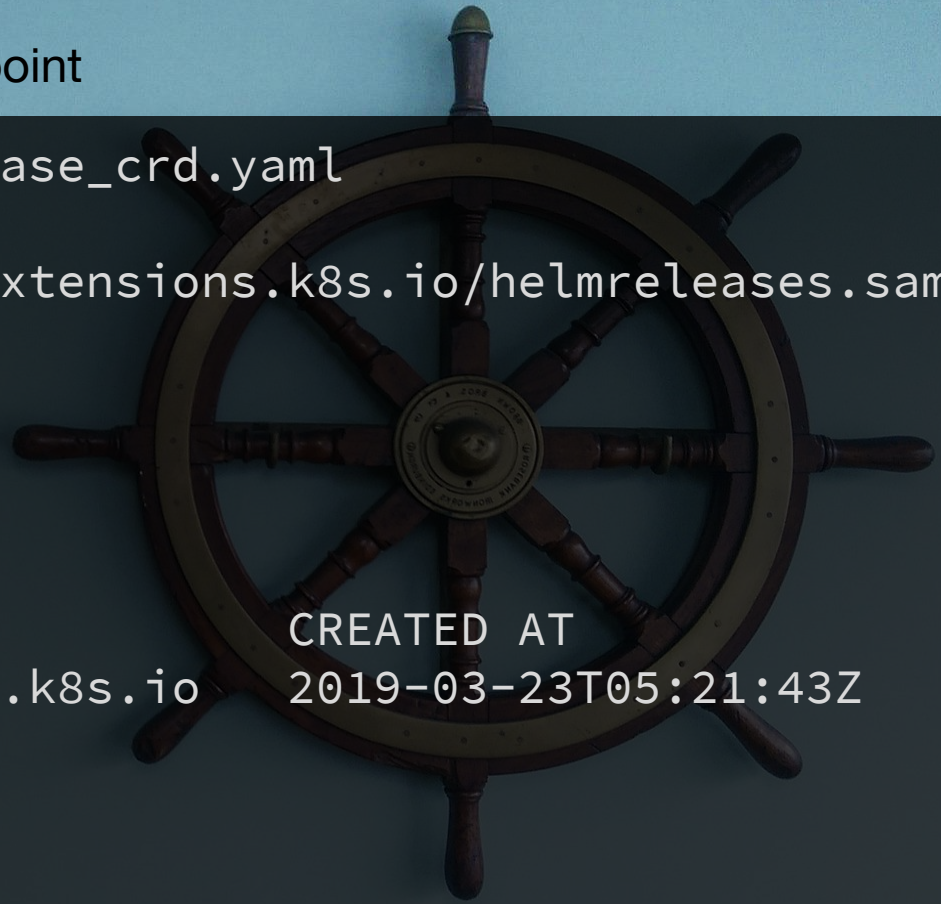
```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: helmreleases.samplecontroller.k8s.io
spec:
  group: samplecontroller.k8s.io
  version: v1alpha1
  names:
    kind: HelmRelease
    plural: helmreleases
  scope: Namespaced
```



Custom resource definitions

Custom object with their own API endpoint

```
$ kubectl create -f helm_release_crd.yaml

customresourcedefinition.apiextensions.k8s.io/helmreleases.samplecontroller.k8s.io created


$ kubectl get crd

NAME                                    CREATED AT
helmreleases.samplecontroller.k8s.io    2019-03-23T05:21:43Z
```



Custom resource definitions

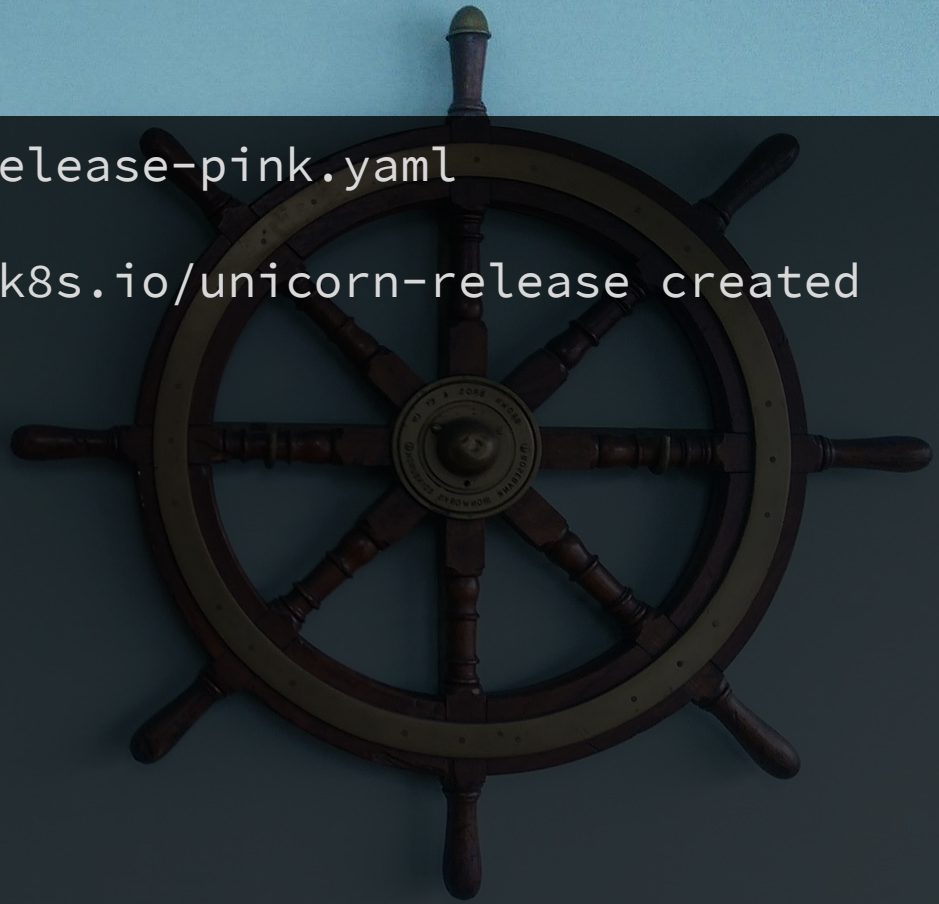
Store / retrieve structured data

```
apiVersion: samplecontroller.k8s.io/v1alpha1
kind: HelmRelease
metadata:
  name: unicorn-release
spec:
  releaseVersion: pink
  releaseName: unicorn
```

Custom resource definitions

Store / retrieve structured data

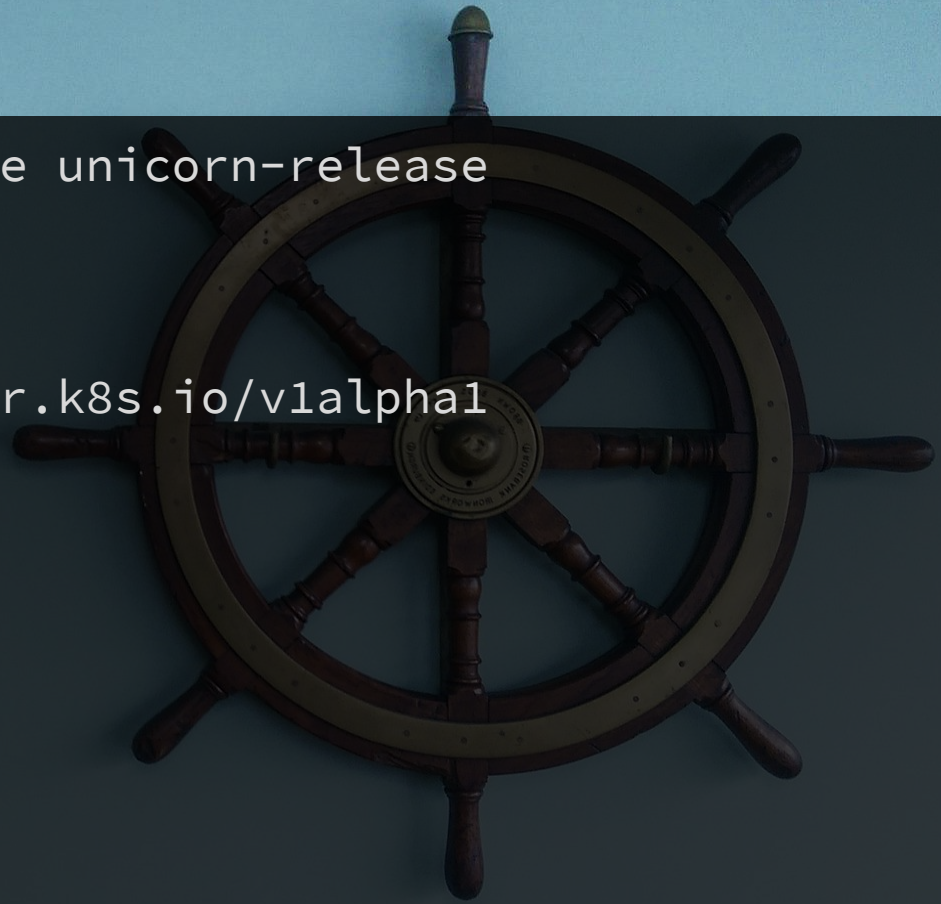
```
$ kubectl create -f unicorn-release-pink.yaml

helmrelease.samplecontroller.k8s.io/unicorn-release created



$ kubectl get helmreleases

NAME              AGE
unicorn-release   36s
```



Custom resource definitions

Store / retrieve structured data

```
$ kubectl describe helmrelease unicorn-release

Name:         unicorn-release
Namespace:    default
API Version:  samplecontroller.k8s.io/v1alpha1
Kind:         HelmRelease
Metadata:
  ...
Spec:
  Release Name:     unicorn
  Release Version:  pink
Events:             <none>
```



Custom controllers

Controller pattern

- Watches the current state of the cluster
- Ensure desired state of cluster = current state of cluster
- If desired state ≠ current state, will take action to make them match

Custom controllers

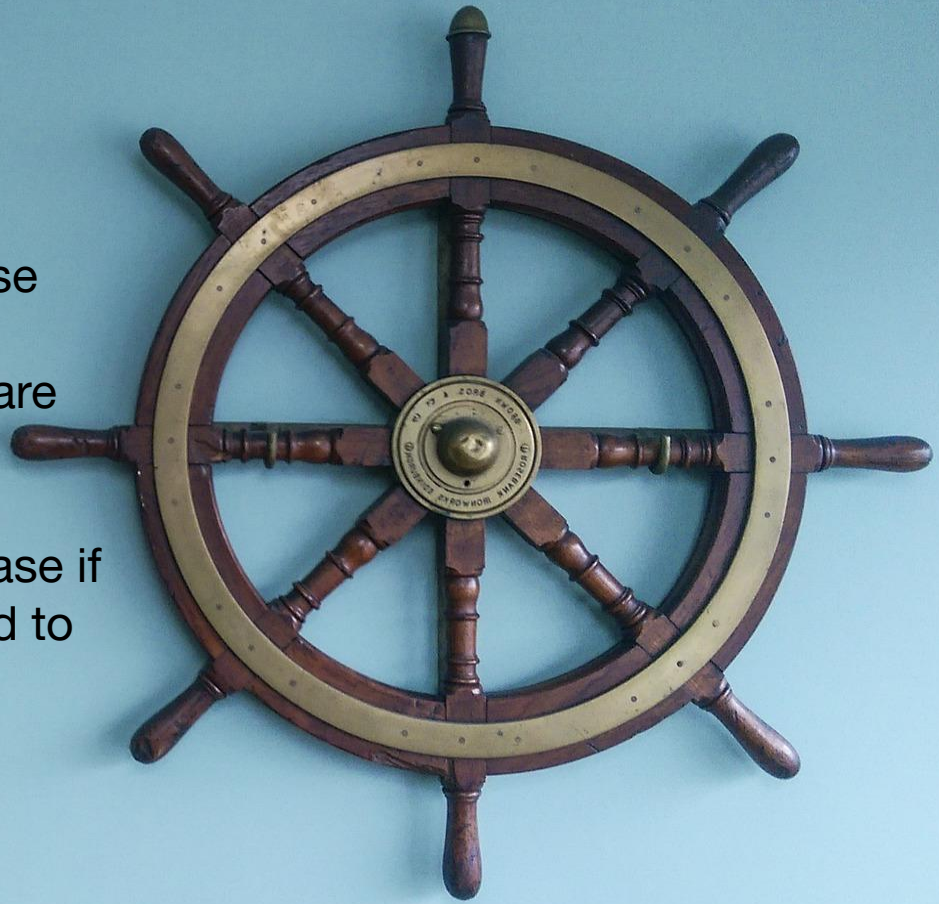
- Listen to any resource type
- Ensure existing state of resource type = desired state of resource type
- If desired state ≠ existing state, will take action to make existing state = desired state
- **This is implemented using your own logic!**

Clone kubernetes/sample-controller from GitHub for an example of a sample controller

Example: custom controllers

Helm Release Controller

- Listen to CRDs of type HelmRelease
- Ensures all desired Helm releases are installed / upgraded
- Will install / upgrade the Helm release if not already installed / not upgraded to desired version



Helm Release Controller

- Cluster logic remains within the cluster
- Declarative API: let the cluster manage itself
- No need for additional script / Jenkins job

Helm Release Controller

- Automated rollback according to a logic of our own
- Allow for custom business logic
- No need to install / maintain the Helm Client on different servers

Introducing unicorns

A very simple app!

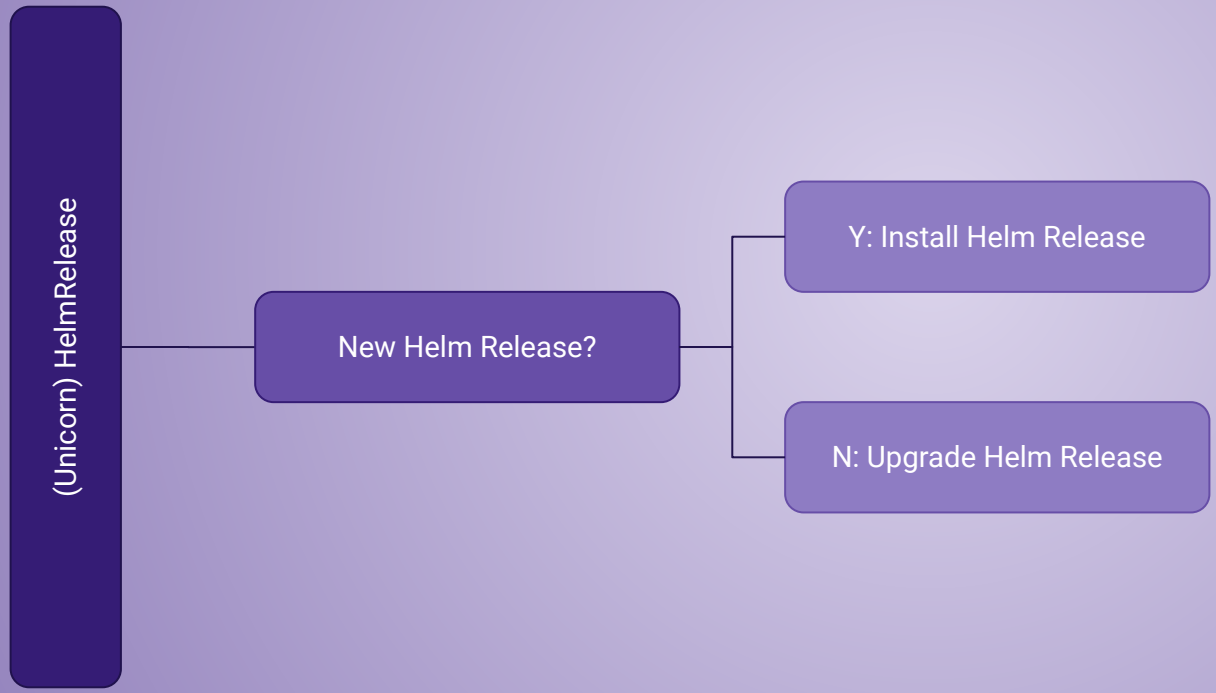
The app
One single HTML page showing a unicorn, serviced by Python's SimpleHTTPServer

Kubernetes resources
One deployment, with only 1 container containing the Unicorn app

3 Helm charts

- Pink unicorn
- Blue unicorn
- Green unicorn

Unicorns CRD'd



Unicorns CRD'd : the demo

Helm Release Controller: the implementation

Clone of the existing Sample Controller from Kubernetes

No update done to listeners, informers, event handlers, etc.

Focus on `SyncHandlers()` which is responsible for ensuring that desired state = existing state

kubernetes/sample-controller: https://github.com/kubernetes/sample-controller

Helm Release Controller: the implementation

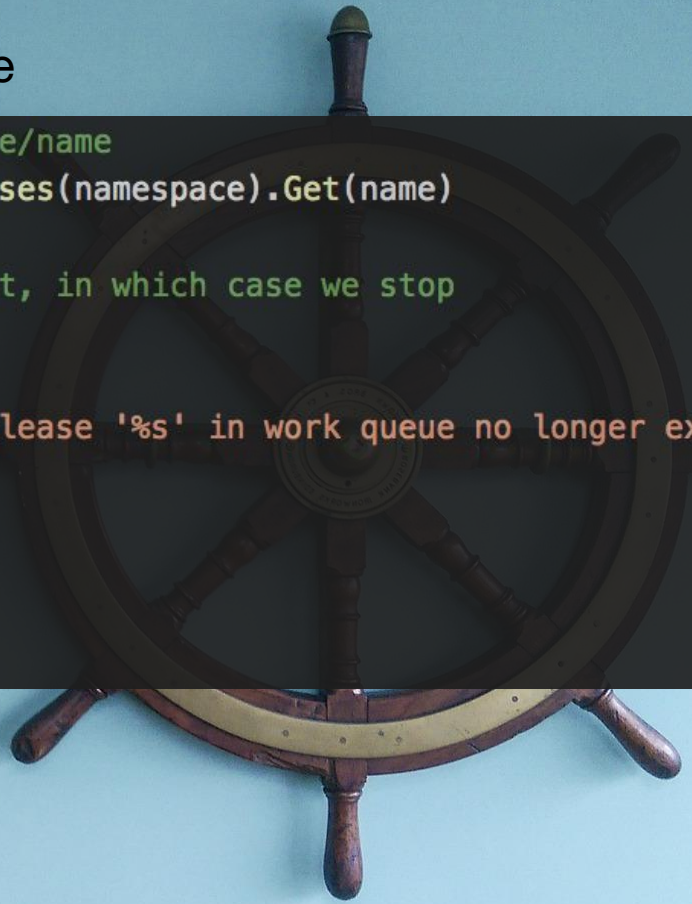
Receive a CRD of type Helm Release

```
// syncHandler compares the actual state with the desired, and attempts to
// converge the two. It then updates the Status block of the HelmRelease resource
// with the current status of the resource.
func (c *Controller) syncHandler(key string) error {
    // Convert the namespace/name string into a distinct namespace and name
    namespace, name, err := cache.SplitMetaNamespaceKey(key)
    if err != nil {
        utilruntime.HandleError(fmt.Errorf("invalid resource key: %s", key))
        return nil
    }
```

Helm Release Controller: the implementation

Get information on CRD of type HelmRelease

```
// Get the HelmRelease resource with this namespace/name
helmRelease, err := c.helmReleasesLister.HelmReleases(namespace).Get(name)
if err != nil {
    // The HelmRelease resource may no longer exist, in which case we stop
    // processing.
    if errors.IsNotFound(err) {
        utilruntime.HandleError(fmt.Errorf("helmRelease '%s' in work queue no longer exists", key))
        return nil
    }

    return err
}
```



Helm Release Controller: the implementation

Get information about the current CRD (= desired state)

```
// Get desired release name
releaseName := helmRelease.Spec.ReleaseName
if releaseName == "" {
    utilruntime.HandleError(fmt.Errorf("%s: Helm release name must be specified", key))
    return nil
}

// Get desired release version
releaseVersion := helmRelease.Spec.ReleaseVersion
if releaseVersion == "" {
    utilruntime.HandleError(fmt.Errorf("%s: Helm release version must be specified", key))
    return nil
}
```

Helm Release Controller: the implementation

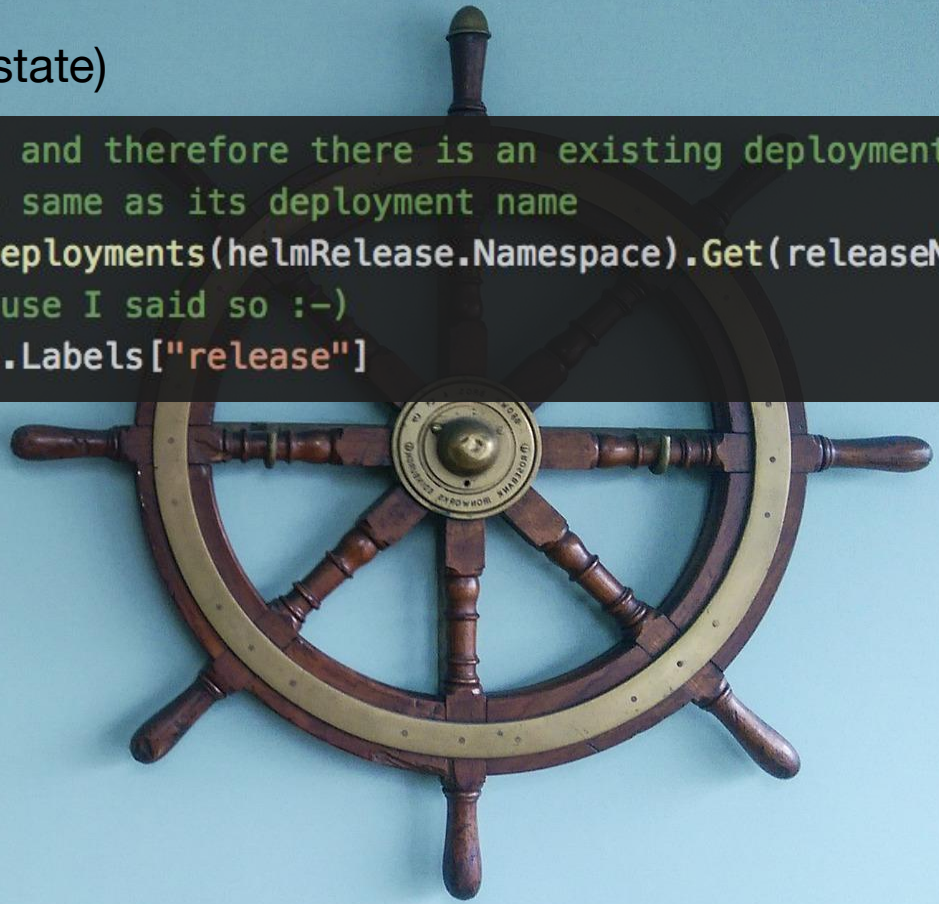
Install Helm release if it doesn't already exist (= match desired state)

```
// Check if this release already exists
helmClient := helm.NewClient(helm.Host("192.168.64.2:30522"))
_, err = helmClient.ReleaseHistory(releaseName)

if error.IsNotFound(err) {
    // Helm release was not found
    // We will created the named release
    _, err = helmClient.InstallRelease(
        // hardcoded local path for now
        // this will later be pulled from our Helm repositories
        fmt.Sprintf("/Users/pauline.lallinec/dev/unicorn-helm-charts/%s-%s.tgz", releaseName, releaseVersion),
        helmRelease.Namespace,
        helm.ReleaseName(releaseName),
    )
}
```

Helm Release Controller: the implementation

Check existing deployment (= existing state)

```
// At this stage we know a release exist and therefore there is an existing deployment
// By convention the release name is the same as its deployment name
deployment, err := c.deploymentsLister.Deployments(helmRelease.Namespace).Get(releaseName)
// We know we have a release label, because I said so :-)
deployedRelease := deployment.ObjectMeta.Labels["release"]
```



Helm Release Controller: the implementation

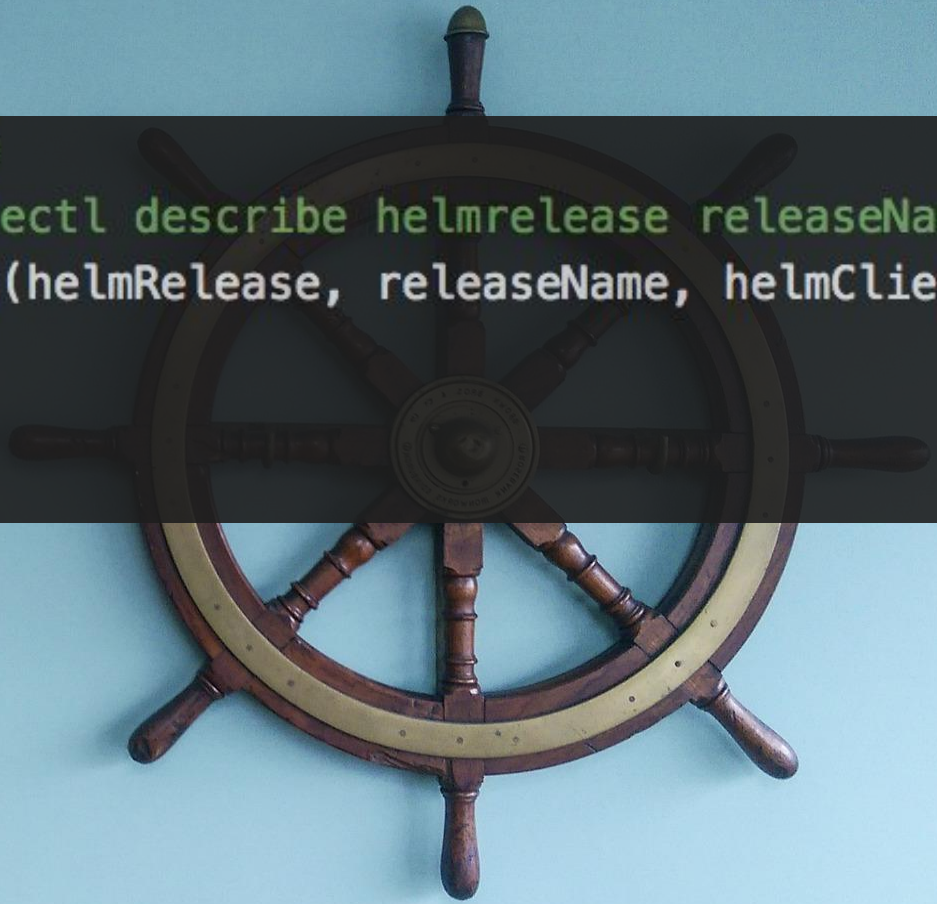
Upgrade existing release if necessary (= match desired state)

```
// Compare desired release with existing release and upgrade release if they do not match
desiredRelease := fmt.Sprintf("%s-%s", releaseVersion, releaseName)
if desiredRelease != deployedRelease {
    klog.V(4).Infof("Desired Release [%s] - Deployed released: [%s]", desiredRelease, deployedRelease)
    _, err = helmClient.UpdateRelease(
        releaseName,
        fmt.Sprintf("/Users/pauline.lallinec/dev/unicorn-helm-charts/%s-%s.tgz", releaseName, releaseVersion),
    )
}
```

Helm Release Controller: the implementation

Update Helm Release status

```
// Update status of the release
// This is accessible from `kubectl describe helmrelease releaseName`
err = c.updateHelmReleaseStatus(helmRelease, releaseName, helmClient)
if err != nil {
    return err
}
```



Helm Release Controller: the implementation

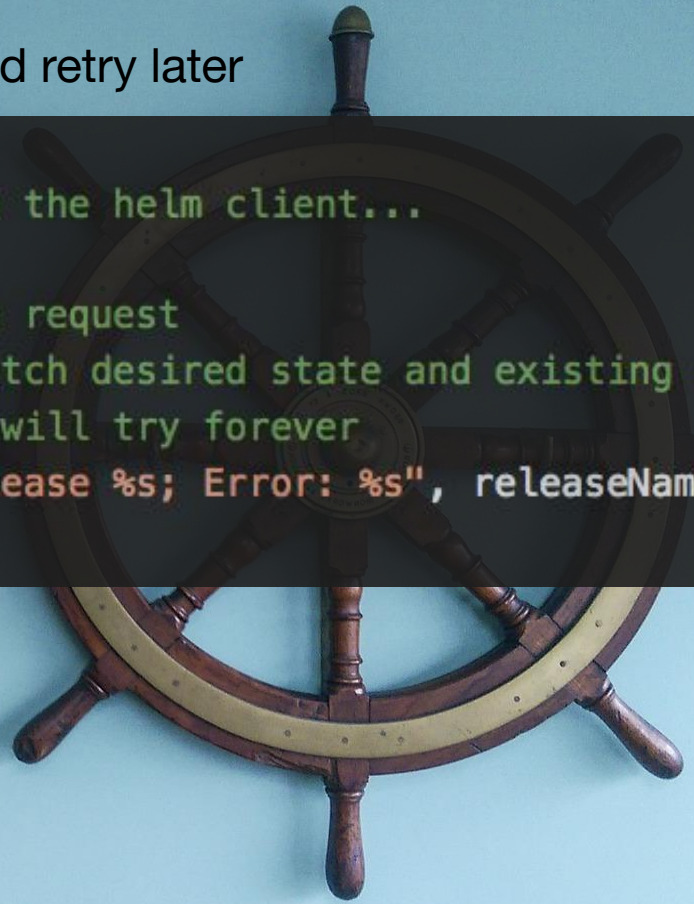
Update Helm Release status

```go
func (c *Controller) updateHelmReleaseStatus(helmRelease *samplev1alpha1.HelmRelease, releaseName string, helmClient *helm.Client) error {
    // Create copy of object
    helmReleaseCopy := helmRelease.DeepCopy()

    // Get release Status
    releaseStatus, err := helmClient.ReleaseStatus(releaseName)
    if err != nil {
        return err
    }
    helmReleaseCopy.Status.ReleaseStatus = releaseStatus.String()

    // Similarly as above, get history of release and content of release

    _, err = c.sampleclientset.SamplecontrollerV1alpha1().HelmReleases(helmRelease.Namespace).Update(helmReleaseCopy)
    return err
}
```

Helm Release Controller: the implementation

If an error happens, re-enqueue the event and retry later

```
// This is everywhere in the code
// After every install / upgrade / call to the helm client...
if err != nil {
    // if any error, we will reenqueue the request
    // the controller will try again to match desired state and existing state
    // if the error happens everytime, it will try forever
    return fmt.Errorf("Error upgrading release %s; Error: %s", releaseName, err)
}
```



Helm Release Controller: the implementation

Finally, return successful sync event

```
c.recorder.Event(helmRelease, corev1.EventTypeNormal, SuccessSynced, MessageResourceSynced)
return nil
```

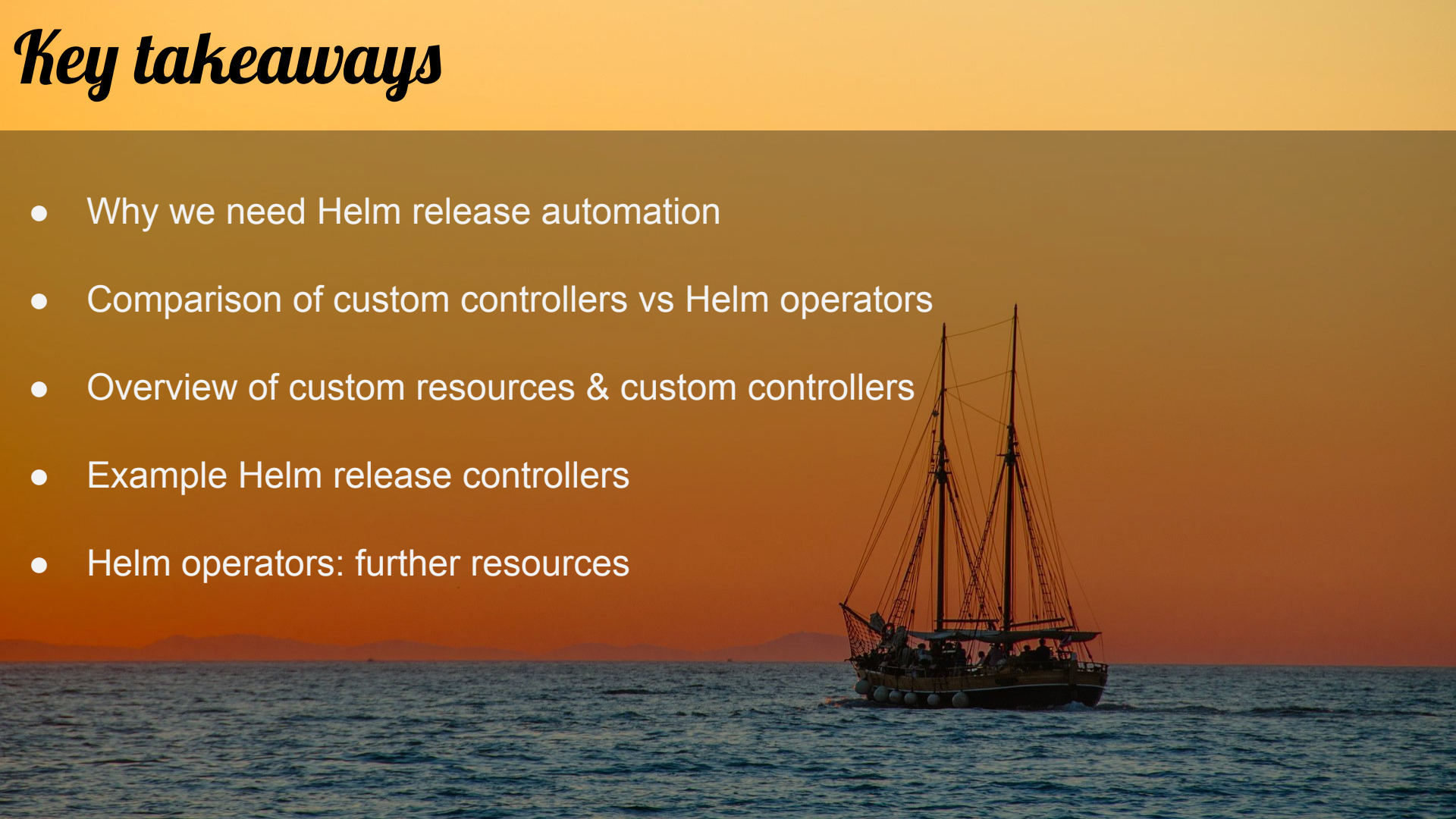
CRDs + Custom controllers: Other benefits

- Choice of programming language
- Can enforce validation
- Can support /status and /scale subresources (and maybe /exec and /log in the future*)

* https://github.com/kubernetes/kubernetes/issues/72637

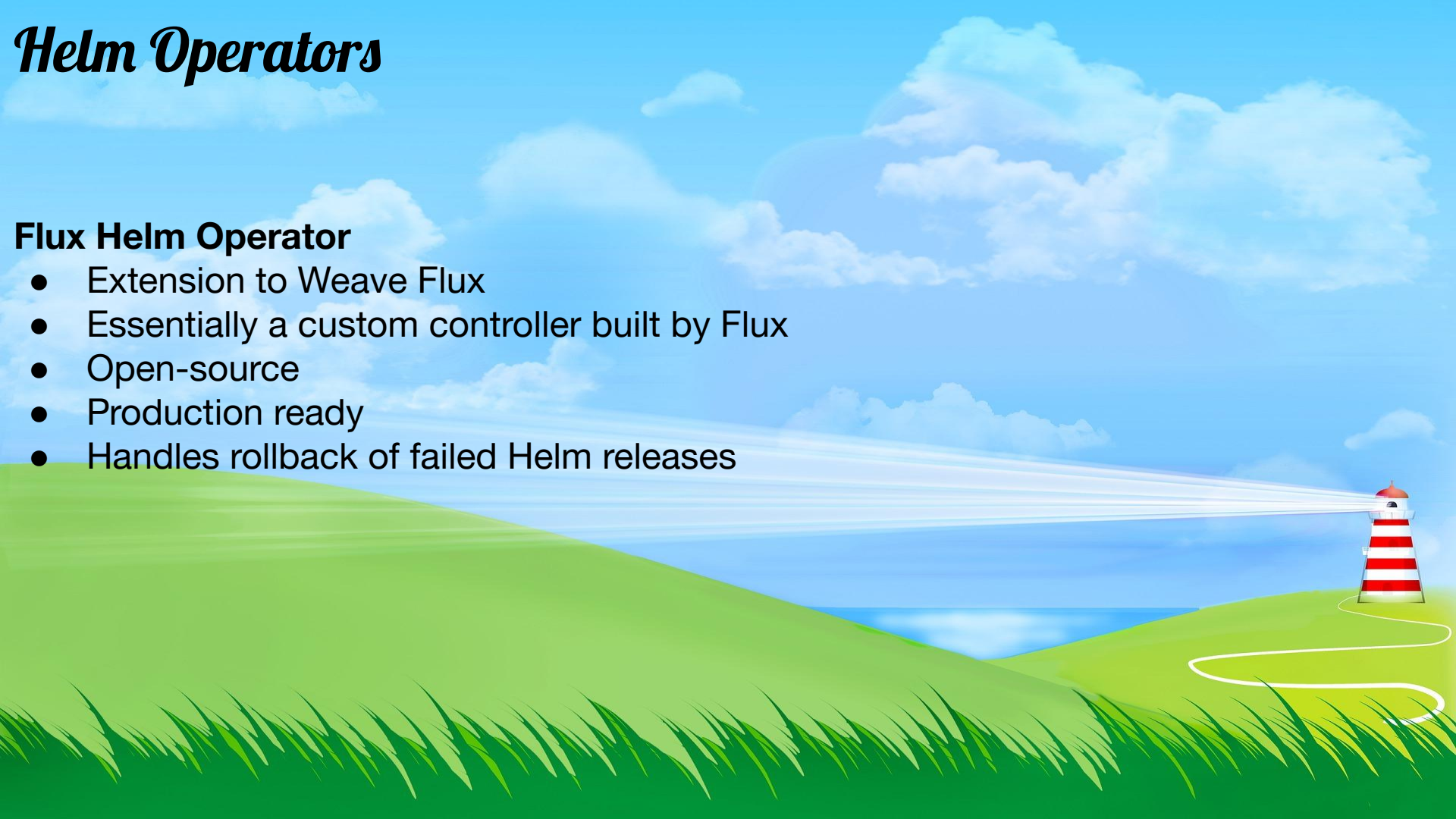
Key takeaways

- Why we need Helm release automation
- Comparison of custom controllers vs Helm operators
- Overview of custom resources & custom controllers
- Example Helm release controllers
- Helm operators: further resources



Helm Operators

Helm Operators



Flux Helm Operator

- Extension to Weave Flux
- Essentially a custom controller built by Flux
- Open-source
- Production ready
- Handles rollback of failed Helm releases

Helm Operators

How to setup and use Weave Flux + Flux Helm Operator?

"GitOps Continuous Delivery with Helm Operator"

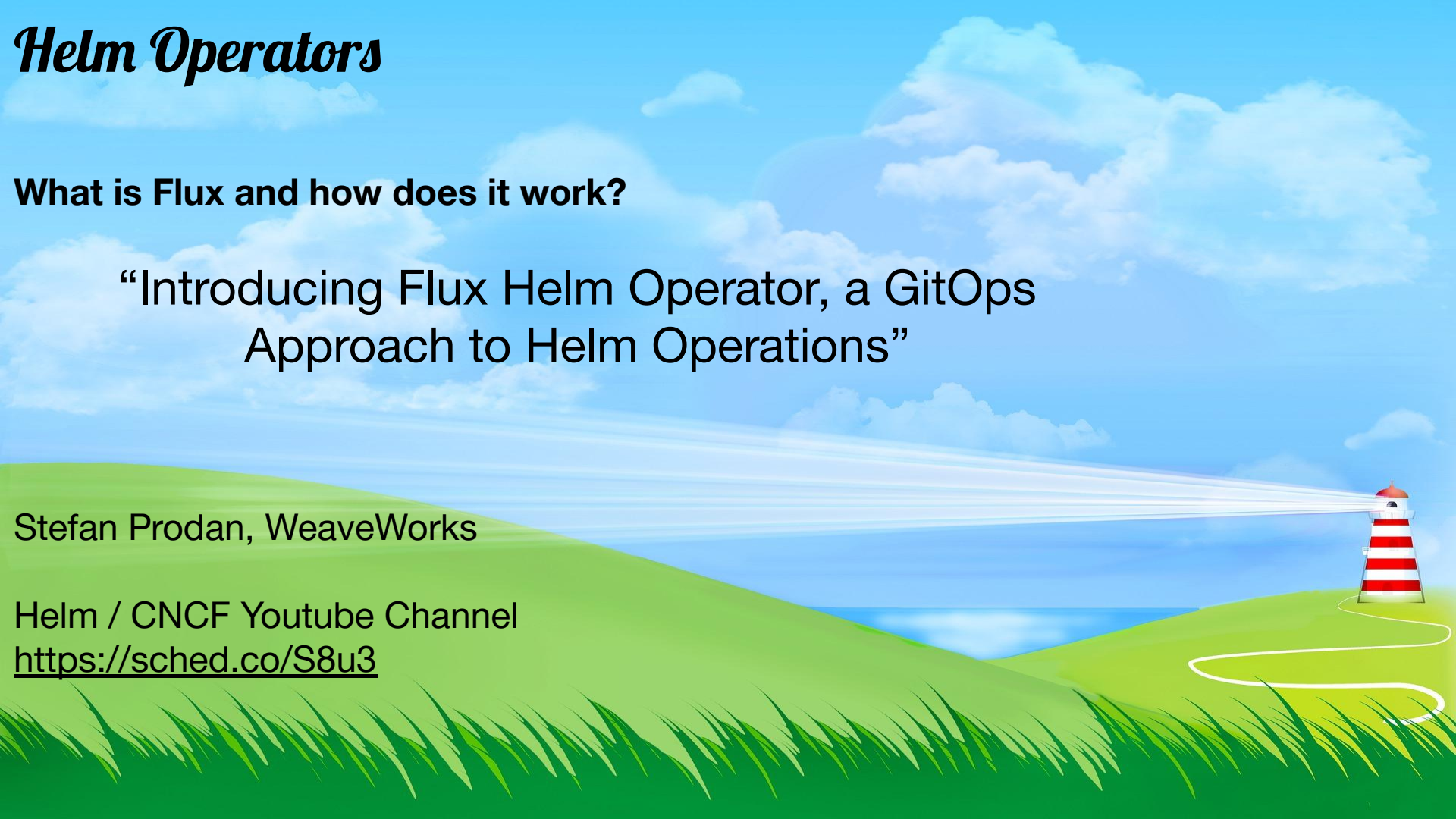
Kingdon Barrett, University of Notre Dame
Stefan Prodan, WeaveWorks

Thu. 12th Sept., 1.25pm, IJ Zaal
https://sched.co/S8ti

Helm Operators

What is Flux and how does it work?

"Introducing Flux Helm Operator, a GitOps Approach to Helm Operations"



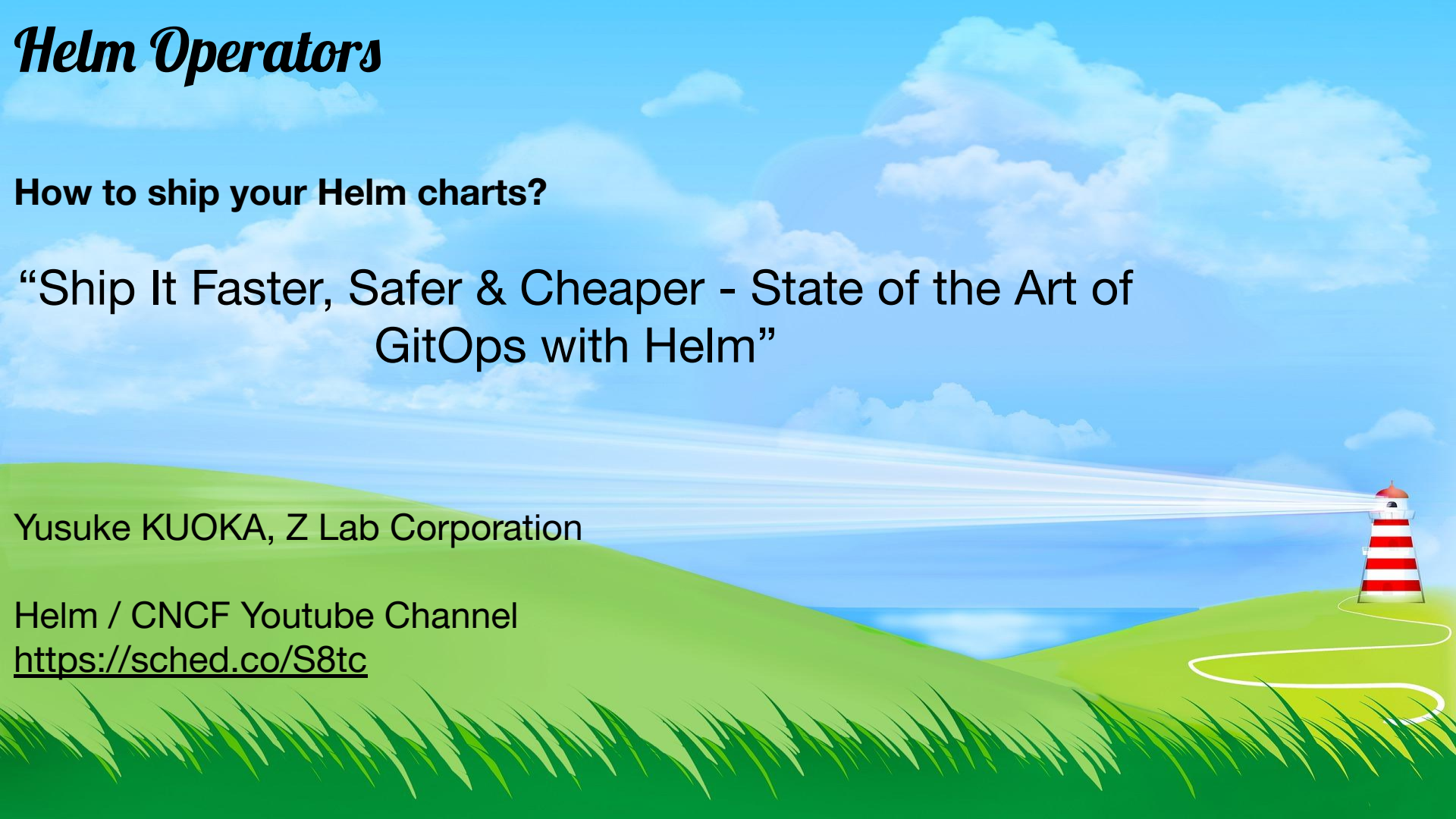
Stefan Prodan, WeaveWorks

Helm / CNCF Youtube Channel
https://sched.co/S8u3

Helm Operators

How to ship your Helm charts?

"Ship It Faster, Safer & Cheaper - State of the Art of GitOps with Helm"



Yusuke KUOKA, Z Lab Corporation

Helm / CNCF Youtube Channel
https://sched.co/S8tc

This presentation features not only my work, but my entire team's work, and therefore I would like to recognize their contribution :-)

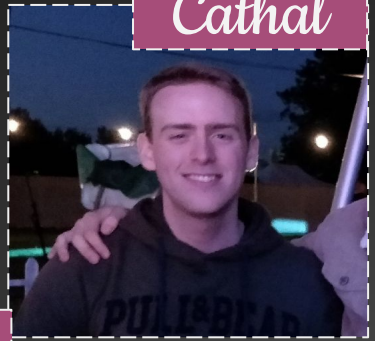
Thank you

Scylla + Fabrication Team

Slide not included in the presentation

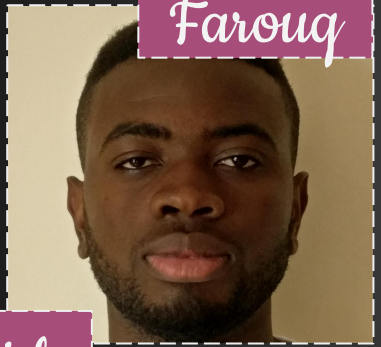
Adrian

Cathal

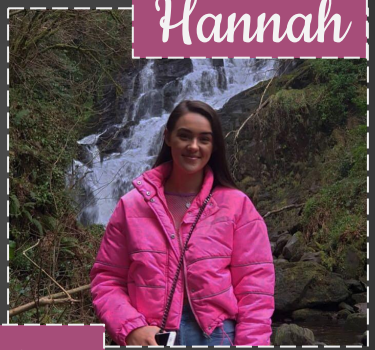


David

Farouq



Hannah



John

Sathish

Rob

Pauline

Thank you!

Learn more more about engineering at Workday!
medium.com/workday-engineering

Learn more about opportunities at Workday!
workday.com/careers

Learn more about me!
@plallin

Resources!

Deep dive in Kubernetes controllers https://engineering.bitnami.com/articles/a-deep-dive-into-kubernetes-controllers.html

Writing a Blue/Green deployment CRD https://www.youtube.com/watch?v=YmRem5IWaEc

Official Kubernetes sample controller https://github.com/kubernetes/sample-controller

“CRD’s aren’t just for add-ons anymore” https://www.youtube.com/watch?v=ji0FWzFwNhA

Flux helm operator https://github.com/fluxcd/

Cool PR https://github.com/fluxcd/flux/pull/2006

Operator pattern https://kubernetes.io/docs/concepts/extend-kubernetes/operator/