KubeCon | CloudNativeCon

North America 2017

Extending Kubernetes 101

Travis Nielsen, Principal SDE, *Quantum Corp, Rook*

Extensibility Schedule

11:10	Extending Kubernetes 101
11:55	Kubernetes Feature Prototyping with External Controllers and CRDs
2:00	Extending the Kubernetes API: What the Docs Don’t Tell You
2:45	client-go: The Good, The Bad and The Ugly
3:50	Using Custom Resources to Provide Cloud Native API Management
4:35	Extending Kubernetes: Our Journey & Roadmap

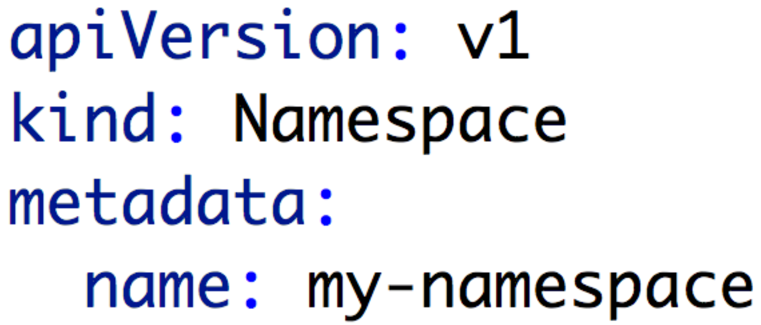
Agenda

- Why to extend Kubernetes
- How to extend Kubernetes
  - Understand Kubernetes Patterns
- Code Walkthrough!
- Q&A

Resource Declaration

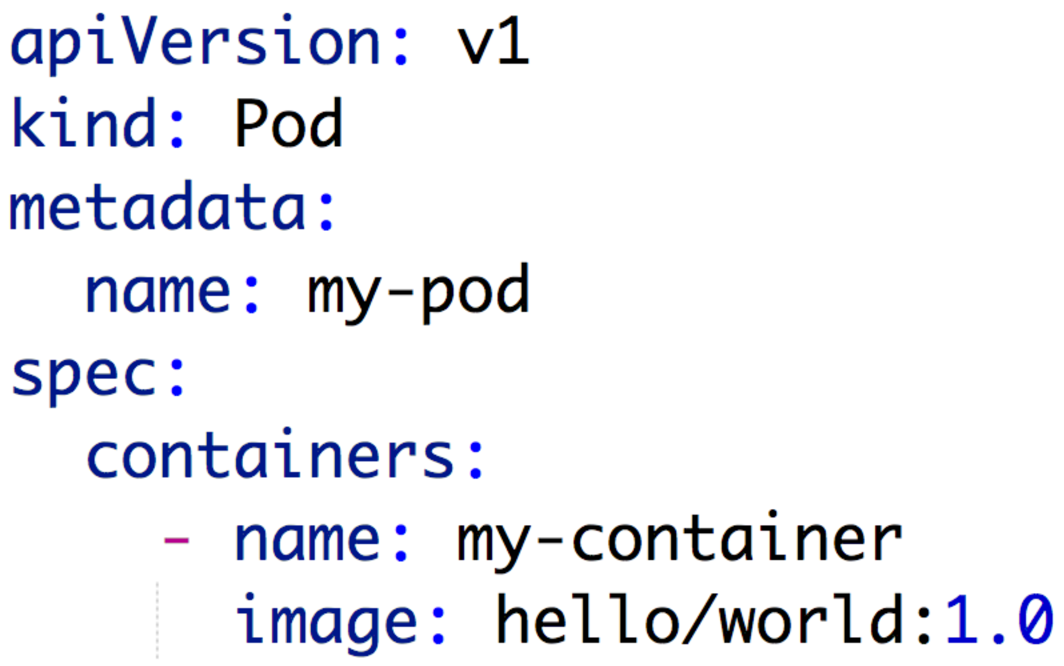
- Kubernetes resources are declarative
- Define resources and their properties in yaml
- Kubernetes handles their creation

Declarative Namespace



```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
```

Declarative Pod



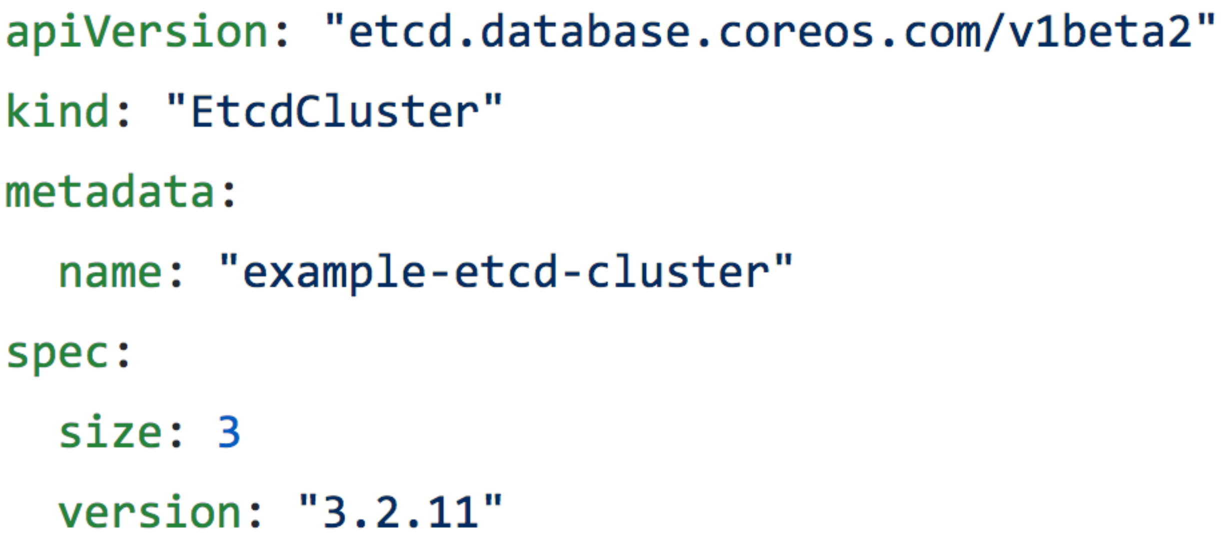
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: hello/world:1.0
```

Declarative Custom Resources

- Custom resources can also be defined
- Follow the same pattern as built-in resources

Example: Etcd

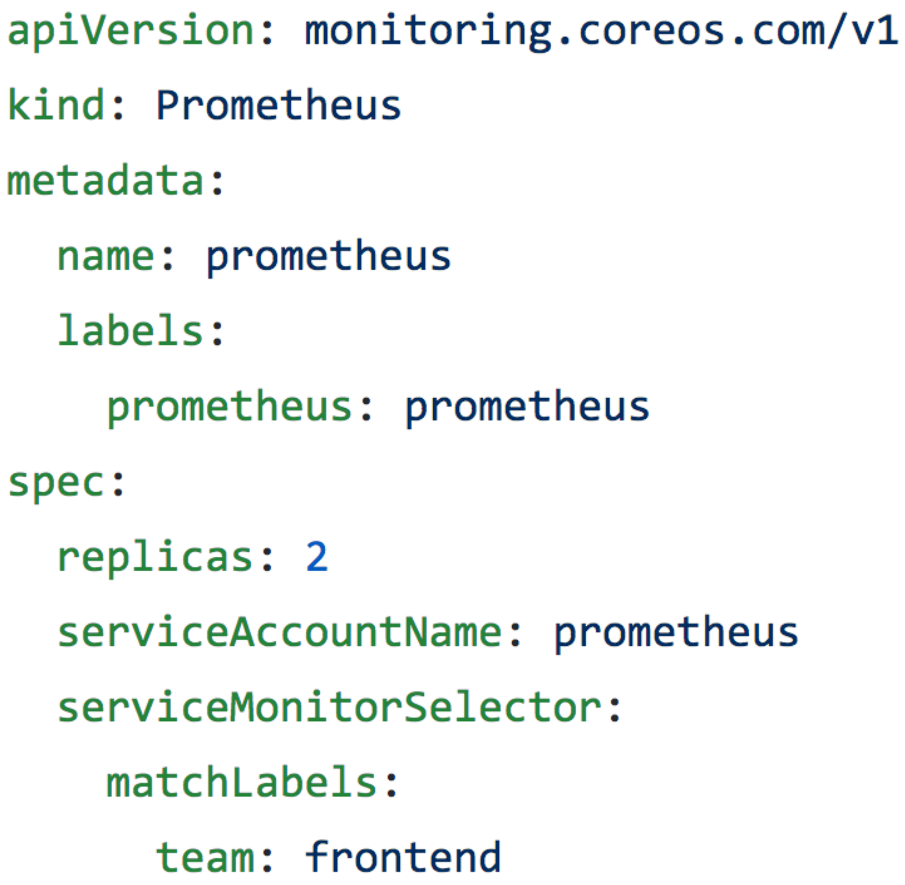
https://github.com/coreos/etcd-operator



```yaml
apiVersion: "etcd.database.coreos.com/v1beta2"
kind: "EtcdCluster"
metadata:
  name: "example-etcd-cluster"
spec:
  size: 3
  version: "3.2.11"
```

Example: Prometheus

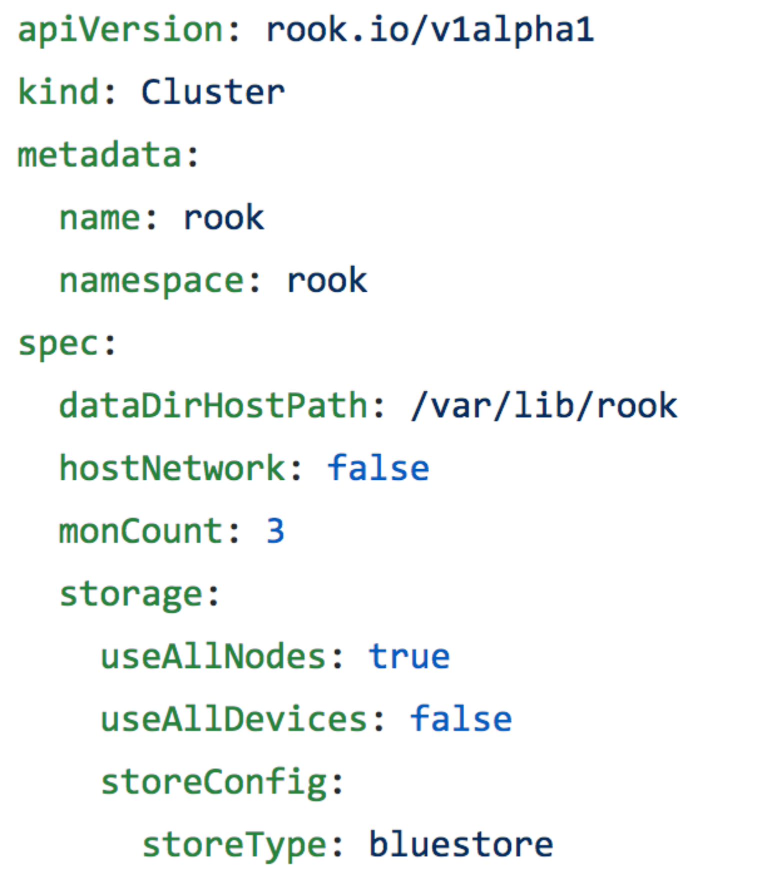
https://github.com/coreos/prometheus-operator/



```yaml
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
  labels:
    prometheus: prometheus
spec:
  replicas: 2
  serviceAccountName: prometheus
  serviceMonitorSelector:
    matchLabels:
      team: frontend
```

Example: Rook

https://github.com/rook/rook



```yaml
apiVersion: rook.io/v1alpha1
kind: Cluster
metadata:
  name: rook
  namespace: rook
spec:
  dataDirHostPath: /var/lib/rook
  hostNetwork: false
  monCount: 3
  storage:
    useAllNodes: true
    useAllDevices: false
    storeConfig:
      storeType: bluestore
```

Are Custom Resources Needed?

- What if Kubernetes resources do not satisfy your application's management requirements?
- What if you need to handle failover differently?
- What if you have dynamic components to deploy?
- What if you want to automate management beyond health checks?

Example: Distributed Data Platform

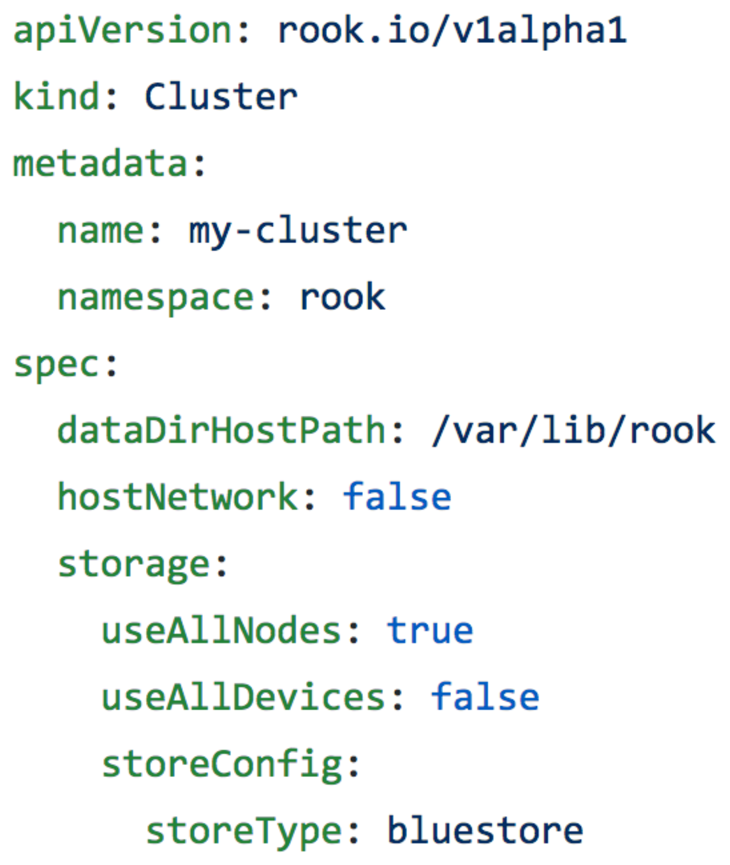
- Distributed Data platforms require special handling
- Deployment
- Monitoring
- Failover
- Upgrade
- Durability

The Traditional Approach

- Implement a management REST API
- Expose a service endpoint
- No integration with Kubernetes API or kubectl
- No RBAC security
- ☹

The Extension Approach

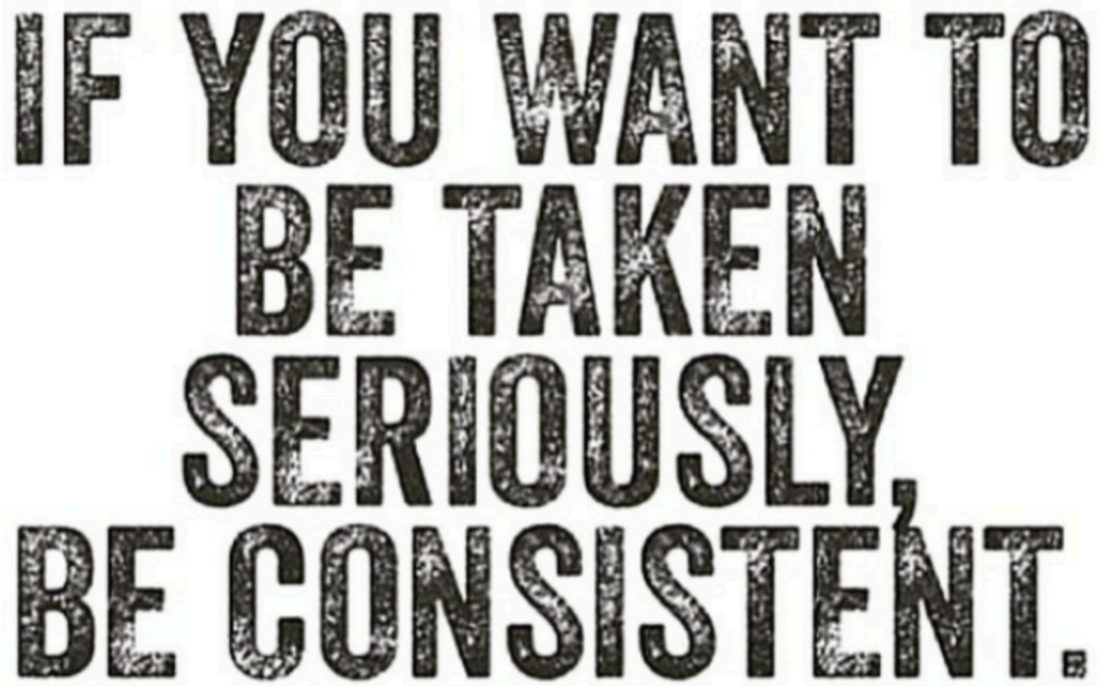
- Custom resources are designed to feel like built-in resources
- Custom Resource Definition (CRD)
  - Declarative state
- Resource manifests
  - kubectl create -f my-cluster.yaml
  - kubectl edit clusters.rook.io my-cluster
  - kubectl delete clusters.rook.io my-cluster



```
apiVersion: rook.io/v1alpha1
kind: Cluster
metadata:
  name: my-cluster
  namespace: rook
spec:
  dataDirHostPath: /var/lib/rook
  hostNetwork: false
  storage:
    useAllNodes: true
    useAllDevices: false
    storeConfig:
      storeType: bluestore
```

Consistent Tools for Extensions

- Tools
  - kubectl
  - Helm
- API
  - client-go
- Security
  - RBAC



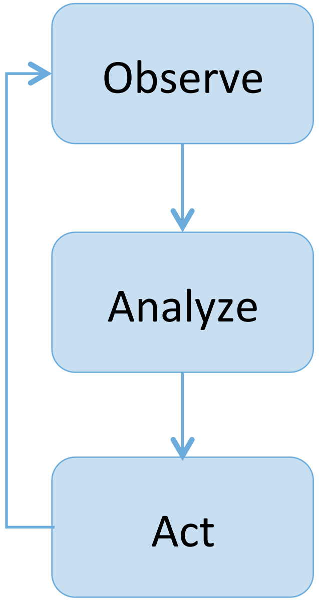
IF YOU WANT TO BE TAKEN SERIOUSLY, BE CONSISTENT.

Resource Patterns

- Kubernetes resources follow a pattern
- Declarative
    - kubectl create -f my-resource.yaml
    - kubectl edit deployment my-resource
    - kubectl delete deployment my-resource
- Handled by a controller

Controllers

- Controllers act on the resource metadata
  - Create, update, delete
- Control loop
  - Observe
    - Watch for a desired state, triggered by Kubernetes events
  - Analyze
    - Calculate changes
  - Act
    - Add, update, or remove a resource



Developing Custom Resources

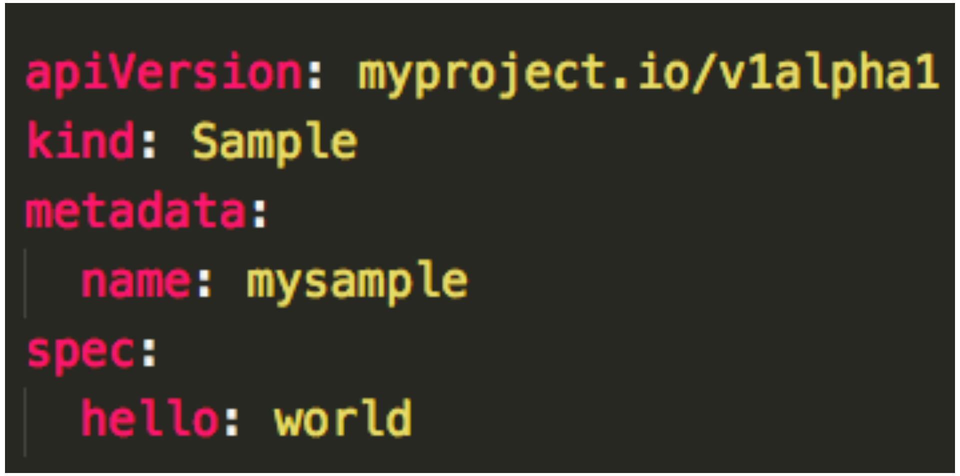
- Design your custom resource
  - Define the CRD properties
- Make your resource available to clients
  - Run the code generation tools
- Develop your custom controller (operator)
  - Simplified with the Operator Kit (https://github.com/rook/operator-kit)
  - Register the CRD
  - Implement Add(), Update(), and Delete()
  - Start watching the CRD
- Build

Custom Resources at Runtime

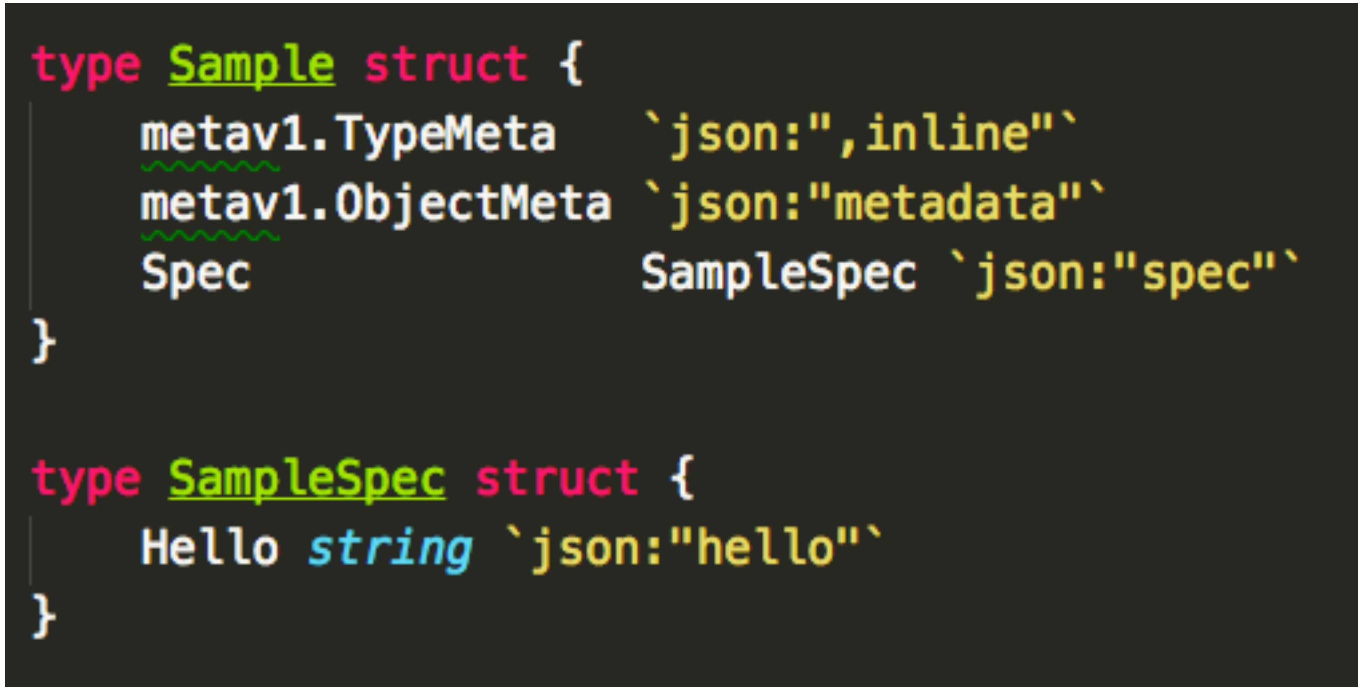
- Define operator manifest
  - RBAC rules
  - Role bindings
  - Deployment for the operator
- Run the operator
  - kubectl create -f sample-operator.yaml
- Create a custom resource
  - kubectl create -f sample-resource.yaml

Sample CRD

https://github.com/rook/operator-kit/tree/master/sample-operator



```yaml
apiVersion: myproject.io/v1alpha1
kind: Sample
metadata:
  name: mysample
spec:
  hello: world
```



```go
type Sample struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata"`
    Spec              SampleSpec `json:"spec"`
}

type SampleSpec struct {
    Hello string `json:"hello"`
}
```

Demo: Custom Resource

- Start the operator
  - kubectl create -f sample-operator.yaml
- Create the custom resource
  - kubectl create -f sample.yaml
- Update the resource
  - kubectl edit samples my-sample
- Delete the resource
  - kubectl delete samples my-sample
- View the actions in the operator log
  - kubectl logs -l app=sample-operator

Code Walkthrough

Key Takeaways

- CRDs make Kubernetes extensible
- CRDs follow the same patterns as all K8s resources
  - Custom controller applies desired state
- CRDs have low overhead to implement
  - Simple patterns with well-documented examples
  - Majority of your time will be spent on business logic

Rook

- Block, File, and Object storage for Kubernetes
  - Built on Ceph
  - Open to other storage platforms
- CRDs + Operator + Volume Plugin = Fully integrated storage
- CRDs
  - Cluster, Pool, ObjectStore, Filesystem, VolumeAttachment
- Submitted to CNCF

Links

- CRD Docs:
  - https://kubernetes.io/docs/concepts/api-extension/custom-resources/
- Operator kit: Library to create a custom controller
  - Includes the “hello world” sample
  - https://github.com/rook/operator-kit
- Etcd: https://github.com/coreos/etcd-operator
- Prometheus: https://github.com/coreos-prometheus-operator
- Rook: https://github.com/rook/rook

Questions?

- Travis Nielsen
    - travis.nielsen@quantum.com
    - github: @travisn
    - twitter: @STravisNielsen
- Rook
    - https://github.com/rook/rook
    - We’re hiring!