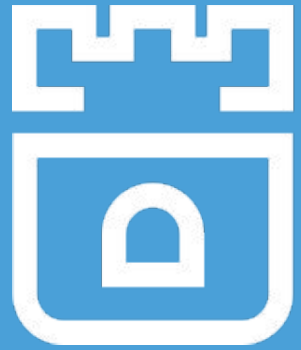

Rook Deep Dive

Tony Allen, Rook Contributor, Upbound Engineer
Alexander Trost, Rook Contributor, System Administrator

https://rook.io/
https://github.com/rook/rook

What is Rook?

- Cloud-Native Storage Orchestrator
- Extends Kubernetes with custom types and controllers
- Automates deployment, bootstrapping, configuration, provisioning, scaling, upgrading, migration, disaster recovery, monitoring, and resource management
- Framework for many storage providers and solutions
- Open Source (Apache 2.0)
- Hosted by the Cloud-Native Computing Foundation (CNCF)

Storage Challenges

- Reliance on external storage
  - Requires these services to be accessible
  - Deployment burden
- Reliance on cloud provider managed services
  - Vendor lock-in
- Day 2 operations - who is managing the storage?

Possible Solutions

- Deploy storage systems INTO the cluster
- Portable abstractions for all storage needs
  - Database, message queue, cache, object store, etc.
- Power of choice: cost, features, resiliency, compliance
- Automated management by smart software

Custom Resource Definitions (CRDs)

- Teaches Kubernetes about new first-class objects
- Custom Resource Definition (CRDs) are arbitrary types that extend the Kubernetes API
  - look just like any other built-in object (e.g. Pod)
  - Enabled native `kubectl` experience
- A means for user to describe their desired state

Rook Operators

- Implements the **Operator Pattern** for storage solutions
- User defines *desired state* for the storage cluster
- The Operator runs reconciliation loops
  - Observe - Watches for changes in desired state and cluster
  - Analyze - Determine differences between desired and actual
  - Act - Applies changes to the cluster to drive it towards desired

Rook Framework for Storage Solutions

- Rook is more than just a collection of Operators and CRDs
- **Framework** for storage providers to integrate their solutions into cloud-native environments
  - Storage resource normalization
  - Operator patterns/plumbing
  - Common policies, specs, logic
  - Testing effort
- Ceph, CockroachDB, Minio, Nexenta, and more...

Developer Deep Dive:
New Minio Operator

Minio ObjectStore CRD

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: objectstores.minio.rook.io
spec:
  group: minio.rook.io
  names:
    kind: ObjectStore
    listKind: ObjectStoreList
    plural: objectstores
    singular: objectstore
  scope: Namespaced
  version: v1alpha1
```

Minio ObjectStore Custom Object

Using the Object Store CRD

```
>> kubectl create -f object-store-crd.yaml
customresourcedefinition "objectstores.minio.rook.io" created

>> kubectl get crds
NAME                          AGE
objectstores.minio.rook.io    9s

>> kubectl create -f object-store.yaml
objectstore "my-store" created

>> kubectl get objectstores
NAME       AGE
my-store   19s
```

Using the Object Store CRD

Revisiting the ObjectStore

```yaml
apiVersion: minio.rook.io/v1alpha1
kind: ObjectStore
metadata:
  name: my-store
  namespace: rook-minio
spec:
  scope:
    nodeCount: 4
  resources:
  - name: objectserver
    limits:
      cpu: "500m"
      memory: "2Gi"
  network:
    hostNetwork: false
    port: 9000
  credentials:
    accessKey: "TEMP_DEMO_ACCESS_KEY"
    secretKey: "TEMP_DEMO_SECRET_KEY"
```

- Rook knows how to work with common information in storage object specs (networking, node counts, etc.)
- Only the credentials are Minio-specific.
- We can use this information to deploy a Minio cluster.

Minio Operator

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: rook-minio-operator
  namespace: rook-minio-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: rook-minio-operator
    spec:
      serviceAccountName: rook-minio-operator
      containers:
      - name: rook-minio-operator
        image: rook/minio:master
        args: ["minio", "operator"]
```

- We specify the container that the Minio operator will reside in.
- Args are provided to inform the Rook binary that it needs to operate on Minio.
- We would include the CRD in the same file as this operator description.

Minio Operator Container Image

```
FROM BASEIMAGE
ADD https://dl.minio.io/server/minio/release/linux-amd64/minio /usr/bin/
RUN chmod +x /usr/bin/minio

ADD rook rookflex /usr/local/bin/

ENTRYPOINT ["/tini", "--", "/usr/local/bin/rook"]
CMD [""]
```

- Contains both Minio server/tools and Rook libraries.
- Optimized docker build to collapse layers and minify image.
- Base image is Ubuntu Xenial.

Minio ObjectStore Golang Types

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

type ObjectStoreSpec struct {
    // How to utilize the underlying storage resources of the cluster
    Scope rookv1alpha2.StorageScopeSpec `json:"scope"`

    // Resource utilization spec (CPU, memory)
    Resources rookv1alpha2.ResourceSpec `json:"resources"`

    // Networking configuration spec
    Network rookv1alpha2.NetworkSpec `json:"network"`

    // Credentials for minio client access (s3 protocol)
    Credentials CredentialConfig `json:"credentials"`
}

type CredentialConfig struct {
    AccessKey string `json:"accessKey"`
    SecretKey string `json:"secretKey"`
}
```

- It’s necessary to implement an ObjectStore struct that defines the config properties the user can edit in the ObjectStore yaml
- Notice the spec uses common types from the Rook framework.

Minio Operator Watching for Events

```go
var ObjectStoreResource = opkit.CustomResource{
    Name:    "objectstore",
    Plural:  "objectstores",
    Group:   "minio.rook.io",
    Version: "v1alpha1",
    Scope:   apiextensionsv1beta1.NamespaceScoped,
    Kind:    reflect.TypeOf(miniov1alpha1.ObjectStore{}).Name(),
}

func (c *MinioController) StartWatch(namespace string, stopCh chan struct{}) error {
    resourceHandlerFuncs := cache.ResourceEventHandlerFuncs{
        AddFunc:    c.onAdd,
        UpdateFunc: c.onUpdate,
        DeleteFunc: c.onDelete,
    }

    logger.Infof("start watching object store resources in namespace %s", namespace)
    watcher := opkit.NewWatcher(ObjectStoreResource, namespace, resourceHandlerFuncs,
        c.context.RookClientset.MinioV1alpha1().RESTClient())
    go watcher.Watch(&miniov1alpha1.ObjectStore{}, stopCh)
}
```

- We create a new watcher to watch for **add**, **update**, or **delete** events.
- Event handler functions are passed to the Rook operator-kit.

Watching with Informers

```
func (w *ResourceWatcher) Watch(objType runtime.Object, done <-chan struct{}) error {
    source := cache.NewListWatchFromClient(
        w.client,
        w.resource.Plural,
        w.namespace,
        fields.Everything())
    _, controller := cache.NewInformer(
        source,

        // The object type.
        objType,

        // resyncPeriod
        // Every resyncPeriod, all resources in the cache will retrigger events.
        // Set to 0 to disable the resync.
        0,

        // Your custom resource event handlers.
        w.resourceEventHandlers)

    go controller.Run(done)
    <-done
    return nil
}
```

- We use an Informer to watch for k8s events, which prevents excessive polling on the API server.
- The informer keeps a cache of objects to limit GETs.

ObjectStore Add Handler

```go
func (c *MinioController) onAdd(obj interface{}) {
    objectstore := obj.(*miniov1alpha1.ObjectStore).DeepCopy()

    // Create the headless service.
    _, err := c.makeMinioHeadlessService(objectstore.Name, objectstore.Namespace, objectstore.Spec)
    if err != nil {
        logger.Errorf("failed to create minio service: %v", err)
        return
    }

    // Create the stateful set.
    _, err = c.makeMinioStatefulSet(objectstore.Name, objectstore.Namespace, objectstore.Spec)
    if err != nil {
        logger.Errorf("failed to create minio stateful set: %v", err)
        return
    }

    // Create the nodeport service.
    svcName := objectstore.Name + "-service"
    _, err = c.makeMinioService(svcName, objectstore.Namespace, objectstore.Spec)
    if err != nil {
        logger.Errorf("failed to create minio service: %v", err)
        return
    }
}
```

- The add handler implementation uses the k8s API to create services, stateful sets, etc.
- We programmatically follow the deployment procedure for the Minio cluster.

ObjectStore Update Handler

```go
func (c *MinioController) onUpdate(oldObj, newObj interface{}) {
    oldStore := oldObj.(*miniov1alpha1.ObjectStore).DeepCopy()
    newStore := newObj.(*miniov1alpha1.ObjectStore).DeepCopy()

    // Analyze differences between old cluster and new cluster,
    // perform operations to make actual state match the desired state
}
```

Administrator Deep Dive

Monitoring

- Prometheus Metrics through Ceph MGR Module
- Dashboards available on Grafana.com

Example Alert Rules - **Coming soon!**
When I have polished mine ;)

Maintenance/Helpful Tips

- Monitoring:
  - Latency Check
    - Watchout for “anomalies”
  - (Working) Alert rules
- (Deep) Scrubbing of OSDs
  - Is enabled automatically => verify that it is done!

Demo

Specifications

OS: Container Linux
Container Runtime: CRI-O (FTW!)

- 3 x K8S Master (for HA)
- Total Nodes: 8

Node Hardware Specs:

- CPU: i7-6700 Quad HT (Skylake)
- Memory: 32GB DDR4
- Storage: 2 x 4TB
- Network: 1 GBit/s

(Price: ~34€ + 79€ Setup)

Specifications - Disk Layout

Disk 1:

- OS: ~80GB
- Rook Partition: ~3910GB

Disk 2:

- Rook: ~3979GB (+ WAL 605MB + DB 21.5GB)

“Raw” Storage Capacity:

8 Nodes * 2 Disks * 4TB

=> 64TB

“Real” (Available) Storage Capacity:

=> ~57.7TB

Demo

Production cluster running stateful applications

How to get involved?

- Contribute to Rook
  - https://github.com/rook/rook
  - https://rook.io/
- Slack - https://rook-io.slack.com/
  - #conferences now for Kubecon EU
- Twitter - @rook_io
- Forums - https://groups.google.com/forum/#!forum/rook-dev
- Community Meetings

More Sessions

- **Kubernetes Runs Anywhere, but Does your Data?**
  - Fri May 4th 14:45, Auditorium 10

Questions?

https://github.com/rook/rook

https://rook.io/

Thank you!

https://github.com/rook/rook

https://rook.io/