Rook in Kubernetes

Overview

Rook is an open-source distributed filesystem designed for use under Kubernetes, and is only supported on Kubernetes 1.7 or higher.

This document serves as a high-level overview or introduction to some basic concepts and patterns surrounding Rook.

All of this is explained in much better detail by their official documentation: https://rook.github.io/docs/rook/master/

Source code is also available here: https://github.com/rook/rook

Prerequisites

Minimum Version: Kubernetes **v1.7** or higher is supported by Rook.

You will need to choose a `hostPath` for the `dataDirHostPath` (ensure that this has at least 5GB of free space).

You will also need to set up RBAC, and ensure that the Flex volume plugin has been configured.

Set the `dataDirHostPath`

If you are using `dataDirHostPath` to persist Rook data on Kubernetes hosts, make sure your host has at least 5GB of space available on the specified path.

Setting up RBAC

On Kubernetes 1.7+, you will need to configure Rook to use RBAC appropriately.

See https://rook.github.io/docs/rook/master/rbac.html

Flex Volume Configuration

The Rook agent requires setup as a Flex volume plugin to manage the storage attachments in your cluster. See the Flex Volume Configuration topic to configure your Kubernetes deployment to load the Rook volume plugin.

Getting Started

Now that we've examined each of the pieces, let's zoom out and see what we can do with the whole cluster.

For the quickest quick start, check out the Rook QuickStart guide: https://rook.github.io/docs/rook/master/quickstart.html

Getting Started without an Existing Kubernetes cluster

The easiest way to deploy a new Kubernetes cluster with Rook support on OpenStack (Nebula / SDSC) is to use the https://github.com/nds-org/kubeadm-terraform repository.

This may work for other cloud providers as well, but has not yet been thoroughly tested.

Getting Started on an Existing Kubernetes cluster

If you're feeling lucky, a simple Rook cluster can be created with the following kubectl commands. For the more detailed install, skip to the next section to deploy the Rook operator.

```
kubectl create -f https://raw.githubusercontent.com/rook/rook/master/cluster/examples/kubernetes/rook-operator.yaml
kubectl create -f https://raw.githubusercontent.com/rook/rook/master/cluster/examples/kubernetes/rook-cluster.yaml
```

After the cluster is running, you can create block, object, or file storage to be consumed by other applications in your cluster.

For a more detailed look at the deployment process, see below.

Deploy the Rook Operator

The first step is to deploy the Rook system components, which include the Rook agent running on each node in your cluster as well as Rook operator pod.

```
kubectl create -f https://raw.githubusercontent.com/rook/rook/master/cluster/examples/kubernetes/rook-operator.yaml

# verify the rook-operator and rook-agents pods are in the `Running` state before proceeding
kubectl -n rook-system get pod
```

You can also deploy the operator with the Rook Helm Chart.

Restart Kubelet (Kubernetes 1.7.x only)

For versions of Kubernetes prior to 1.8, the Kubelet process on all nodes will require a restart after the Rook operator and Rook agents have been deployed. As part of their initial setup, the Rook agents deploy and configure a Flexvolume plugin in order to integrate with Kubernetes' volume controller framework. In Kubernetes v1.8+, the dynamic Flexvolume plugin discovery will find and initialize our plugin, but in older versions of Kubernetes a manual restart of the Kubelet will be required.

Create a Rook Cluster

Now that the Rook operator and agent pods are running, we can create the Rook cluster. For the cluster to survive reboots, make sure you set the `dataDirHostPath` property. For more settings, see the documentation on configuring the cluster.

Save the cluster spec as `rook-cluster.yaml`:

```
apiVersion: v1
kind: Namespace
metadata:
  name: rook
---
apiVersion: rook.io/v1alpha1
kind: Cluster
metadata:
  name: rook
  namespace: rook
spec:
  dataDirHostPath: /var/lib/rook
  storage:
    useAllNodes: true
    useAllDevices: false
    storeConfig:
      storeType: bluestore
      databaseSizeMB: 1024
      journalSizeMB: 1024
```

Create the cluster:

```
kubectl create -f rook-cluster.yaml
```

Use `kubectl` to list pods in the `rook` namespace. You should be able to see the following pods once they are all running:

```
$ kubectl -n rook get pod
NAME                                READY     STATUS    RESTARTS   AGE
rook-ceph-mgr0-1279756402-wc4vt   1/1       Running   0          5m
rook-ceph-mon0-jflt5               1/1       Running   0          6m
rook-ceph-mon1-wkc8p               1/1       Running   0          6m
rook-ceph-mon2-p31dj               1/1       Running   0          6m
rook-ceph-osd-0h6nb                1/1       Running   0          5m
```

Monitoring Your Rook Cluster

A glimpse into setting up Prometheus for monitoring Rook: https://rook.github.io/docs/rook/master/monitoring.html

Advanced Configuration

Advanced Configuration options are also documented here: https://rook.github.io/docs/rook/master/advanced-configuration.html

- Log Collection
- OSD Information
- Separate Storage Groups
- Configuring Pools
- Custom ceph.conf Settings
- OSD CRUSH Settings
- Phantom OSD Removal

Debugging

For common issues, see https://github.com/rook/rook/blob/master/Documentation/common-issues.md

For more help debugging, see https://github.com/rook/rook/blob/master/Documentation/toolbox.md

Cluster Teardown

See https://rook.github.io/docs/rook/master/teardown.html for thorough steps on destroying / cleaning up your Rook cluster

Components

Rook runs a number of smaller microservices that run on different nodes in your Kubernetes cluster:

- The Rook Operator + API
- Ceph Managers / Monitors / OSDs
- Rook Agents

The Rook Operator

The Rook operator is a simple container that has all that is needed to bootstrap and monitor the storage cluster.

The operator will start and monitor ceph monitor pods and a daemonset for the Object Storage Devices (OSDs).

The operator manages custom resource definitions (CRDs) for pools, object stores (S3/Swift), and file systems by initializing the pods and other artifacts necessary to run the services.

The operator will monitor the storage daemons to ensure the cluster is healthy.

The operator will also watch for desired state changes requested by the api service and apply the changes.

The Rook operator also creates the Rook agents as a daemonset, which runs a pod on each node.

Ceph Managers / Monitors / OSDs

The operator will start and monitor ceph monitor pods and a daemonset for the OSDs, which provides basic Reliable Autonomic Distributed Object Store (RADOS) storage.

The operator will monitor the storage daemons to ensure the cluster is healthy.

Ceph monitors (aka "Ceph mons") will be started or failed over when necessary, and other adjustments are made as the cluster grows or shrinks.

Rook Agents

Each agent is a pod deployed on a different Kubernetes node, which configures a Flexvolume plugin that integrates with Kubernetes' volume controller framework.

All storage operations required on the node are handled such as attaching network storage devices, mounting volumes, and formatting the filesystem.

Storage

Rook provides three types of storage to the Kubernetes cluster:

- Block Storage: Mount storage to a single pod
- Object Storage: Expose an S3 API to the storage cluster for applications to put and get data that is accessible from inside or outside the Kubernetes cluster
- Shared File System: Mount a file system that can be shared across multiple pods

Custom Resource Definitions

Rook also allows you to create and manage your storage cluster through custom resource definitions (CRDs). Each type of resource has its own CRD defined.

- Cluster: A Rook cluster provides the basis of the storage platform to serve block, object stores, and shared file systems.
- Pool: A pool manages the backing store for a block store. Pools are also used internally by object and file stores.
- Object Store: An object store exposes storage with an S3-compatible interface.
- File System: A file system provides shared storage for multiple Kubernetes pods.

Shared Storage Example

Shamelessly stolen from https://rook.github.io/docs/rook/master/filesystem.html

Prerequisites

This guide assumes you have created a Rook cluster as explained in the main Kubernetes guide

Multiple File Systems Not Supported

By default only one shared file system can be created with Rook. Multiple file system support in Ceph is still considered experimental and can be enabled with the environment variable `ROOK_ALLOW_MULTIPLE_FILESYSTEMS` defined in `rook-operator.yaml`.

Please refer to cephfs experimental features page for more information.

Create the File System

Create the file system by specifying the desired settings for the metadata pool, data pools, and metadata server in the `Filesystem`CRD. In this example we create the metadata pool with replication of three and a single data pool with erasure coding. For more options, see the documentation on creating shared file systems.

Save this shared file system definition as `rook-filesystem.yaml`:

```
apiVersion: rook.io/v1alpha1
kind: Filesystem
metadata:
  name: myfs
  namespace: rook
spec:
  metadataPool:
    replicated:
      size: 3        <------------ this was eventually the problem
  dataPools:
    - erasureCoded:
       dataChunks: 2
       codingChunks: 1
  metadataServer:
    activeCount: 1
    activeStandby: true
```

The Rook operator will create all the pools and other resources necessary to start the service. This may take a minute to complete.

```
# Create the file system
$ kubectl create -f rook-filesystem.yaml


# To confirm the file system is configured, wait for the mds pods to start
$ kubectl -n rook get pod -l app=rook-ceph-mds
NAME                                READY     STATUS    RESTARTS   AGE
rook-ceph-mds-myfs-7d59fdfcf4-h8kw9       1/1       Running   0          12s
rook-ceph-mds-myfs-7d59fdfcf4-kgkjp       1/1       Running   0          12s
```

In this example, there is one active instance of MDS which is up, with one MDS instance in `standby-replay` mode in case of failover:To see detailed status of the file system, start and connect to the Rook toolbox. A new line will be shown with `ceph status` for the `mds` service.

```
$ ceph
status

  ...
  services:
    mds: myfs-1/1/1 up {[myfs:0]=mzw58b=up:active}, 1 up:standby-replay
```

Consume the Shared File System: Busybox + NGINX Example

As an example, we will start the kube-registry pod with the shared file system as the backing store. Save the following spec as `kube-registry.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: deployment-demo
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    demo: deployment
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: deployment-demo
spec:
  selector:
    matchLabels:
      demo: deployment
  replicas: 2
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    metadata:
      labels:
        demo: deployment
        version: v1
    spec:
      containers:
      - name: busybox
        image: busybox
        command: [ "sh", "-c", "while true; do echo $(hostname) v1 > /data/index.html; sleep 60; done" ]
        volumeMounts:
        - name: content
          mountPath: /data
      - name: nginx
        image: nginx
        volumeMounts:
          - name: content
            mountPath: /usr/share/nginx/html
            readOnly: true
      volumes:
      - name: content
        flexVolume:
          driver: rook.io/rook
          fsType: ceph
          options:
            fsName: myfs # name of the filesystem specified in the filesystem CRD.
            clusterNamespace: rook # namespace where the Rook cluster is deployed
            clusterName: rook # namespace where the Rook cluster is deployed
            # by default the path is /, but you can override and mount a specific path of the filesystem by
using the path attribute
            # path: /some/path/inside/cephfs
```

NOTE: I had to explicitly specify `clusterName` in the YAML above... newer versions of Rook will fallback to `clusterNamespace`

Kernel Version Requirement

If the Rook cluster has more than one filesystem and the application pod is scheduled to a node with kernel version older than 4.7, inconsistent results may arise since kernels older than 4.7 do not support specifying filesystem namespaces.

Testing Shared Storage

After creating our above example, we should now have 2 pods each with 2 containers running on 2 separate nodes:

```
ubuntu@mldev-master:~$ kubectl get pods -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP             NODE
deployment-demo-c59b896c8-8jph6   2/2       Running   0          33m       10.244.1.6     mldev-storage0
deployment-demo-c59b896c8-fnp49   2/2       Running   0          33m       10.244.3.7     mldev-worker0
rook-agent-8c74k                  1/1       Running   0          4h        192.168.0.3    mldev-storage0
rook-agent-bl5sr                  1/1       Running   0          4h        192.168.0.4    mldev-worker0
rook-agent-clxfl                  1/1       Running   0          4h        192.168.0.5    mldev-storage1
rook-agent-gll69                  1/1       Running   0          4h        192.168.0.6    mldev-master
rook-operator-7db5d7b9b8-svmfk    1/1       Running   0          4h        10.244.0.5     mldev-master
```

The nginx containers mount the shared filesystem read-only into /usr/share/nginx/html/

The busybox containers mount the shared filesystem read-write into /data/

To test that everything is working, we can exec into one busybox container and modify a file:

```
# Exec into one of our busybox containers in a pod
ubuntu@mldev-master:~$ kubectl exec -it deployment-demo-c59b896c8-fnp49 -c busybox -- sh


# Create a new file
/ # ls -al /data/
total 4
drwxr-xr-x    1 root     root             0 May 22 21:28 .
drwxr-xr-x    1 root     root          4096 May 22 21:26 ..
-rw-r--r--    1 root     root            35 May 22 22:03 index.html
/ # echo '<div>Hello, World!</div>' > /data/testing.html


# Ensure that the file was created successfully
/ # cat /data/testing.html
<div>Hello, World!</div>
```

Make sure that this appeared in the nginx container within the same pod:

```
# Exec into one of the same pod's nginx container (as a sanity check)
ubuntu@mldev-master:~$ kubectl exec -it deployment-demo-c59b896c8-fnp49 -c nginx -- sh


# Ensure that the file we created previously exists
/ # ls -al /usr/share/nginx/html/
total 5
drwxr-xr-x    1 root     root             0 May 22 21:28 .
drwxr-xr-x    1 root     root          4096 May 22 21:26 ..
-rw-r--r--    1 root     root            35 May 22 22:03 index.html
-rw-r--r--    1 root     root            25 May 22 21:28 testing.html


# Ensure that the file has the correct contents
/ # cat /usr/share/nginx/html/testing.html
<div>Hello, World!</div>
```

Perform the same steps in both containers in the other pod:

```
# Verify the same file contents in both containers of the other pod (on another node)
ubuntu@mldev-master:~$ kubectl exec -it deployment-demo-c59b896c8-8jph6 -c busybox -- cat /data/testing.html
<div>Hello, World!</div>
ubuntu@mldev-master:~$ kubectl exec -it deployment-demo-c59b896c8-8jph6 -c nginx -- cat /usr/share/nginx/html
/testing.html
<div>Hello, World!</div>
```

If all of the file contents match, then congratulations!

You have just set up your first shared filesystem under Rook!

Under the Hood

For more information on the low-level processes involved in the above example, see https://github.com/rook/rook/blob/master/design/filesystem.md

After running our above example, we can SSH into the storage0 and worker0 nodes to get a better sense of where Rook stores its data.

Our current confusion seems to come from some hard-coded values in Zonca's deployment of `rook-cluster.yaml`:

```
apiVersion: v1
kind: Namespace
metadata:
  name: rook
---
apiVersion: rook.io/v1alpha1
kind: Cluster
metadata:
  name: rook
  namespace: rook
spec:
  versionTag: v0.6.2
  dataDirHostPath: /var/lib/rook    <------ this seems to be important
  storage:
    useAllNodes: true
    useAllDevices: false
    storeConfig:
      storeType: bluestore
      databaseSizeMB: 1024
      journalSizeMB: 1024
    directories:
    - path: "/vol_b"
```

The `dataDirHostPath` above is the same one mentioned in the quick start and at the top of this document - this seems to tell Rook where on the host it should persist its configuration.

The `directories` section is supposed to list the paths that will be included in the storage cluster. (Note that using two directories on the same physical device can cause a negative performance impact.)

Investigating Storage `directories`

Checking the logs for one of the Rook agents, we can see a success message shows us where the data really lives:

```
# Check logs of rook-agent running on mldev-storage0
ubuntu@mldev-master:~$ kubectl logs -f rook-agent-8c74k
2018-05-22 17:37:46.340923 I | rook: starting Rook v0.6.2 with arguments '/usr/local/bin/rook agent'
2018-05-22 17:37:46.340990 I | rook: flag values: --help=false, --log-level=INFO
2018-05-22 17:37:46.341634 I | rook: starting rook agent
2018-05-22 17:37:47.963857 I | exec: Running command: modinfo -F parm rbd
2018-05-22 17:37:48.451205 I | exec: Running command: modprobe rbd single_major=Y
2018-05-22 17:37:48.945393 I | rook-flexvolume: Rook Flexvolume configured
2018-05-22 17:37:48.945572 I | rook-flexvolume: Listening on unix socket for Kubernetes volume attach commands.
2018-05-22 17:37:48.947679 I | opkit: start watching cluster resource in all namespaces at v1alpha1
2018-05-22 21:03:02.504533 I | rook-flexdriver: mounting ceph filesystem myfs on /var/lib/kubelet/pods/83ad3107-
5e03-11e8-b20b-fa163e9f32d5/volumes/rook.io~rook/content
2018-05-22 21:03:02.589501 I | rook-flexdriver: mounting ceph filesystem myfs on /var/lib/kubelet/pods/83b0ed53-
5e03-11e8-b20b-fa163e9f32d5/volumes/rook.io~rook/content
2018-05-22 21:03:03.084507 I | rook-flexdriver: mounting ceph filesystem myfs on /var/lib/kubelet/pods/83ad3107-
5e03-11e8-b20b-fa163e9f32d5/volumes/rook.io~rook/content
2018-05-22 21:03:03.196658 I | rook-flexdriver: mounting ceph filesystem myfs on /var/lib/kubelet/pods/83b0ed53-
5e03-11e8-b20b-fa163e9f32d5/volumes/rook.io~rook/content
2018-05-22 21:03:04.188182 I | rook-flexdriver: mounting ceph filesystem myfs on /var/lib/kubelet/pods/83ad3107-
5e03-11e8-b20b-fa163e9f32d5/volumes/rook.io~rook/content
...   ...   ...   ...   ...   ...   ...   ...   ...   ...
2018-05-22 21:26:46.887549 I | cephmon: parsing mon endpoints: rook-ceph-mon0=10.101.163.226:6790,rook-ceph-
mon1=10.107.75.148:6790,rook-ceph-mon2=10.101.87.159:6790
2018-05-22 21:26:46.887596 I | op-mon: loaded: maxMonID=2, mons=map[rook-ceph-mon0:0xc42028e980 rook-ceph-mon1:
0xc42028e9c0 rook-ceph-mon2:0xc42028ea20]
2018-05-22 21:26:46.890128 I | rook-flexdriver: WARNING: The node kernel version is 4.4.0-31-generic, which do
not support multiple ceph filesystems. The kernel version has to be at least 4.7. If you have multiple ceph
filesystems, the result could be inconsistent
2018-05-22 21:26:46.890236 I | rook-flexdriver: mounting ceph filesystem myfs on 10.101.163.226:
6790,10.107.75.148:6790,10.101.87.159:6790:/ to /var/lib/kubelet/pods/d40d5673-5e06-11e8-b20b-fa163e9f32d5
/volumes/rook.io~rook/content
2018-05-22 21:26:49.446669 I | rook-flexdriver:
2018-05-22 21:26:49.446832 I | rook-flexdriver: ceph filesystem myfs has been attached and mounted


# SSH into mldev-storage0, elevate privileges, and check the specified sub-folder
ubuntu@mldev-master:~$ ssh ubuntu@192.168.0.3
ubuntu@mldev-storage0:~$ sudo su
root@mldev-storage0:/home/ubuntu# ls -al /var/lib/kubelet/pods/d40d5673-5e06-11e8-b20b-fa163e9f32d5/volumes
/rook.io~rook/content
total 5
drwxr-xr-x 1 root root    0 May 22 21:28 .
drwxr-x--- 3 root root 4096 May 22 21:26 ..
-rw-r--r-- 1 root root   35 May 22 22:33 index.html
-rw-r--r-- 1 root root   25 May 22 21:28 testing.html
```

Obviously this is not where we want the shared filesystem data stored long-term, so I'll need to figure out why these files are persisted into `/var/lib/kubelet` and not into the `directories` specified in the Cluster configuration.

Digging Deeper into `dataDirHostPath`

Checking `/var/lib/rook` directory, we see a few sub-directories:

```
root@mldev-master:/var/lib/rook# ls -al
total 20
drwxr-xr-x  5 root root 4096 May 22 17:40 .
drwxr-xr-x 46 root root 4096 May 22 17:38 ..
drwxr--r--  3 root root 4096 May 22 17:41 osd1
drwxr--r--  2 root root 4096 May 22 17:40 rook
drwxr--r--  3 root root 4096 May 22 17:38 rook-ceph-mon0


root@mldev-master:/var/lib/rook# ls -al rook
total 16
drwxr--r-- 2 root root 4096 May 22 17:40 .
drwxr-xr-x 5 root root 4096 May 22 17:40 ..
-rw-r--r-- 1 root root  168 May 22 17:40 client.admin.keyring
-rw-r--r-- 1 root root 1123 May 22 17:40 rook.config

root@mldev-master:/var/lib/rook# ls -al rook-ceph-mon0/
total 24
drwxr--r-- 3 root root 4096 May 22 17:38 .
drwxr-xr-x 5 root root 4096 May 22 17:40 ..
drwxr--r-- 3 root root 4096 May 22 17:38 data
-rw-r--r-- 1 root root  248 May 22 17:38 keyring
-rw-r--r-- 1 root root  210 May 22 17:38 monmap
-rw-r--r-- 1 root root 1072 May 22 17:38 rook.config
srwxr-xr-x 1 root root    0 May 22 17:38 rook-mon.rook-ceph-mon0.asok


root@mldev-master:/var/lib/rook# ls -al osd1
total 7896
drwxr--r-- 3 root root        4096 May 22 17:41 .
drwxr-xr-x 5 root root        4096 May 22 17:40 ..
lrwxrwxrwx 1 root root          34 May 22 17:40 block -> /var/lib/rook/osd1/bluestore-block
lrwxrwxrwx 1 root root          31 May 22 17:40 block.db -> /var/lib/rook/osd1/bluestore-db
lrwxrwxrwx 1 root root          32 May 22 17:40 block.wal -> /var/lib/rook/osd1/bluestore-wal
-rw-r--r-- 1 root root           2 May 22 17:40 bluefs
-rw-r--r-- 1 root root 74883726950 May 22 22:13 bluestore-block
-rw-r--r-- 1 root root  1073741824 May 22 17:41 bluestore-db
-rw-r--r-- 1 root root   603979776 May 22 22:14 bluestore-wal
-rw-r--r-- 1 root root          37 May 22 17:41 ceph_fsid
-rw-r--r-- 1 root root          37 May 22 17:40 fsid
-rw-r--r-- 1 root root          56 May 22 17:40 keyring
-rw-r--r-- 1 root root           8 May 22 17:40 kv_backend
-rw-r--r-- 1 root root          21 May 22 17:41 magic
-rw-r--r-- 1 root root           4 May 22 17:40 mkfs_done
-rw-r--r-- 1 root root           6 May 22 17:41 ready
-rw-r--r-- 1 root root        1693 May 22 17:40 rook.config
srwxr-xr-x 1 root root           0 May 22 17:41 rook-osd.1.asok
drwxr--r-- 2 root root        4096 May 22 17:40 tmp
-rw-r--r-- 1 root root          10 May 22 17:40 type
-rw-r--r-- 1 root root           2 May 22 17:41 whoami
```

As you can see, these metadata files do not appear to be readable on disk and would likely need to be un-mangled by Rook to properly perform a backup.

Checking the `kubelet` logs...

Digging into the `systemctl` logs for `kubectl`, we can see it's complaining about the volume configuration:

```
ubuntu@mldev-master:~$ ssh ubuntu@192.168.0.4
ubuntu@mldev-worker0:~$ sudo systemctl status kubelet
 kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
           10-kubeadm.conf
   Active: active (running) since Tue 2018-05-22 17:36:36 UTC; 21h ago
     Docs: http://kubernetes.io/docs/
 Main PID: 4607 (kubelet)
    Tasks: 18
   Memory: 50.8M
      CPU: 27min 25.718s
   CGroup: /system.slice/kubelet.service
           4607 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc
/kubernetes/kubelet.conf --pod-manifest-path=/etc/kubernetes/manifests --allow-privileged=true --network-
plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin

May 23 15:00:57 mldev-worker0 kubelet[4607]: E0523 15:00:57.308626    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:03:57 mldev-worker0 kubelet[4607]: E0523 15:03:57.344832    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:06:57 mldev-worker0 kubelet[4607]: E0523 15:06:57.376002    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:09:57 mldev-worker0 kubelet[4607]: E0523 15:09:57.424020    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:12:57 mldev-worker0 kubelet[4607]: E0523 15:12:57.476398    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:15:57 mldev-worker0 kubelet[4607]: E0523 15:15:57.511746    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:18:57 mldev-worker0 kubelet[4607]: E0523 15:18:57.545539    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:21:57 mldev-worker0 kubelet[4607]: E0523 15:21:57.602083    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:24:57 mldev-worker0 kubelet[4607]: E0523 15:24:57.660774    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
May 23 15:27:57 mldev-worker0 kubelet[4607]: E0523 15:27:57.717357    4607 reconciler.go:376] Could not
construct volume information: no volume plugin matched
```

This is likely a reference to the Flex Volume Configuration steps that I breezed over, assuming they had been handled already by Terraform.

Examining our kubelet configuration on the worker node, we see that there is no reference to the `--volume-plugin-dir` mentioned in the Rook prerequisites:

```
root@mldev-worker0:/etc/systemd/system/kubelet.service.d# cat 10-kubeadm.conf
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=
/etc/kubernetes/kubelet.conf"
Environment="KUBELET_SYSTEM_PODS_ARGS=--pod-manifest-path=/etc/kubernetes/manifests --allow-privileged=true"
Environment="KUBELET_NETWORK_ARGS=--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin"
Environment="KUBELET_DNS_ARGS=--cluster-dns=10.96.0.10 --cluster-domain=cluster.local"
Environment="KUBELET_AUTHZ_ARGS=--authorization-mode=Webhook --client-ca-file=/etc/kubernetes/pki/ca.crt"
Environment="KUBELET_CADVISOR_ARGS=--cadvisor-port=0"
Environment="KUBELET_CERTIFICATE_ARGS=--rotate-certificates=true --cert-dir=/var/lib/kubelet/pki"
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_SYSTEM_PODS_ARGS $KUBELET_NETWORK_ARGS
$KUBELET_DNS_ARGS $KUBELET_AUTHZ_ARGS $KUBELET_CADVISOR_ARGS $KUBELET_CERTIFICATE_ARGS $KUBELET_EXTRA_ARGS
```

The following GitHub issues appear to confirm that the error message above seems to stem from a misconfigured `--volume-plugin-dir`:

- https://github.com/kubernetes/kubernetes/issues/16585
- https://github.com/rook/rook/issues/1162

Sadly, even setting this value explicitly did not fix my immediate issue.

Hacking Terraform

At this point, I decided to start hacking the Terraform deployment to get Rook working to the level we'll need for Workbench.

Out of the box, it seems to have everything we would need for Block storage, but this does not give us the ReadWriteMany capabilities that Workbench will require.

As a lesson from GlusterFS, the first stage in debugging a broken deployment process for a shared filesystem is to upgrade to the newest version as soon as humanly possible.

Instead of pulling directly from Zonca's GitHub, I opted to hard-code the entire YAML into the `assets/deploy-rook.sh` script for testing purposes:

```
#!/bin/bash
git clone https://github.com/groundnuty/k8s-wait-for.git

sudo helm repo add rook-alpha https://charts.rook.io/alpha
sudo helm install rook-alpha/rook --name rook --version 0.7.1


   ...    ...    ...    ...


# Now we can install the cluster
echo '
apiVersion: v1
kind: Namespace
metadata:
  name: rook
---
apiVersion: rook.io/v1alpha1
kind: Cluster
metadata:
  name: rook
  namespace: rook
spec:
  versionTag: v0.7.1
  dataDirHostPath: /var/lib/rook
  # cluster level storage configuration and selection
  storage:
    useAllNodes: false
    useAllDevices: false
    deviceFilter:
    metadataDevice:
    location:
    storeConfig:
      storeType: bluestore
      databaseSizeMB: 1024 # this value can be removed for environments with normal sized disks (100 GB or
larger)
      journalSizeMB: 1024  # this value can be removed for environments with normal sized disks (20 GB or
larger)
    nodes:
    - name: "mldev-storage0"
      directories:         # specific directories to use for storage can be specified for each node
      - path: "/vol_b"
    - name: "mldev-storage1"
      directories:         # specific directories to use for storage can be specified for each node
      - path: "/vol_b"
' | kubectl create -f -


# And create a storage class
wget -q https://raw.githubusercontent.com/zonca/jupyterhub-deploy-kubernetes-jetstream/master/storage_rook/rook-
storageclass.yaml
sudo kubectl create -f rook-storageclass.yaml
```

The embedded `rook-cluster.yaml` above differs from Zonca's original template in the following ways:

- Rook has been upgraded from **v0.6.2** to **v0.7.1**, in the helm install and in rook-cluster.yaml
- Expanded **`storage`** section to include a **`nodes`** subsection - this specifies which machines / directories should be part of the storage cluster
- Turn off **`useAllNodes`**

Checking the `rook-operator` logs...

Now, with the new version of rook up and running, I attempted to make a filesystem as before. This time, however, no pods were spawned following my filesystem's creation.

In the rook-operator logs, I noticed a pretty gnarly-sounding error message:

```
2018-05-23 17:30:50.277799 W | op-osd: failed to create osd replica set for node mldev-storage0. ReplicaSet.
apps "rook-ceph-osd-mldev-storage0" is invalid: [spec.template.spec.volumes[2].name: Invalid value: "vol_b": a
DNS-1123 label must consist of lower case alphanumeric characters or '-', and must start and end with an
alphanumeric character (e.g. 'my-name',  or '123-abc', regex used for validation is '[a-z0-9]([-a-z0-9]*[a-z0-
9])?'), spec.template.spec.containers[0].volumeMounts[2].name: Not found: "vol_b"]
2018-05-23 17:30:50.941464 W | op-osd: failed to create osd replica set for node mldev-storage1. ReplicaSet.
apps "rook-ceph-osd-mldev-storage1" is invalid: [spec.template.spec.volumes[2].name: Invalid value: "vol_b": a
DNS-1123 label must consist of lower case alphanumeric characters or '-', and must start and end with an
alphanumeric character (e.g. 'my-name',  or '123-abc', regex used for validation is '[a-z0-9]([-a-z0-9]*[a-z0-
9])?'), spec.template.spec.containers[0].volumeMounts[2].name: Not found: "vol_b"]
2018-05-23 17:30:51.171199 I | exec: Running command: ceph osd unset noscrub --cluster=rook --conf=/var/lib/rook
/rook/rook.config --keyring=/var/lib/rook/rook/client.admin.keyring --format json --out-file /tmp/479433030
2018-05-23 17:30:58.450482 I | exec: noscrub is unset
2018-05-23 17:30:58.450617 I | exec: Running command: ceph osd unset nodeep-scrub --cluster=rook --conf=/var/lib
/rook/rook/rook.config --keyring=/var/lib/rook/rook/client.admin.keyring --format json --out-file /tmp/253765613
2018-05-23 17:31:06.351987 I | exec: nodeep-scrub is unset
2018-05-23 17:31:06.352146 E | op-cluster: failed to create cluster in namespace rook. failed to start the
osds. 2 failures encountered while running osds in namespace rook: failed to create osd replica set for node
mldev-storage0. ReplicaSet.apps "rook-ceph-osd-mldev-storage0" is invalid: [spec.template.spec.volumes[2].name:
Invalid value: "vol_b": a DNS-1123 label must consist of lower case alphanumeric characters or '-', and must
start and end with an alphanumeric character (e.g. 'my-name',  or '123-abc', regex used for validation is '[a-
z0-9]([-a-z0-9]*[a-z0-9])?'), spec.template.spec.containers[0].volumeMounts[2].name: Not found: "vol_b"]
failed to create osd replica set for node mldev-storage1. ReplicaSet.apps "rook-ceph-osd-mldev-storage1" is
invalid: [spec.template.spec.volumes[2].name: Invalid value: "vol_b": a DNS-1123 label must consist of lower
case alphanumeric characters or '-', and must start and end with an alphanumeric character (e.g. 'my-name',  or
'123-abc', regex used for validation is '[a-z0-9]([-a-z0-9]*[a-z0-9])?'), spec.template.spec.containers[0].
volumeMounts[2].name: Not found: "vol_b"]
2018-05-23 17:31:06.352173 E | op-cluster: giving up creating cluster in namespace rook after 5m0s
```

Since Zonca's method would deploy JupyterHub (which doesn't have the same ReadWriteMany volume-mounting requirements), it is possible that the `path: /vol_b` identifier worked fine in the case of block storage, but for a Rook filesystem there are apparently other considerations.

Namely, in this case the path is somehow coerced into a volumeMount's name, which means that our path can't contain an underscore.

Changing **`/vol_b`** to **`/volb`** solved this problem - this must be adjusted both in the `deploy-rook.sh` script above, as well as the `bootstrap-rook.sh` script alongside of it.

Now we're getting somewhere...

After changing the volume path and redeploying (again), now **`myfs`** pods were being spawned after creating the filesystem in Kubernetes, as they should be:

```
ubuntu@mldev-master:~$ kubectl get pods -n rook
NAME                                READY     STATUS    RESTARTS   AGE
rook-api-6fc75cd555-rwcr9           1/1       Running   0          1h
rook-ceph-mds-myfs-6846dc79fb-5hmzx   1/1       Running   0          1h
rook-ceph-mds-myfs-6846dc79fb-jmdr9   1/1       Running   0          1h
rook-ceph-mgr0-5778b8b46b-2m4js     1/1       Running   0          1h
rook-ceph-mon0-bgzh7                1/1       Running   0          1h
rook-ceph-mon1-8bd9x                1/1       Running   0          1h
rook-ceph-mon2-cfq4f                1/1       Running   0          1h
rook-ceph-osd-mldev-storage0-pb6vs  1/1       Running   0          1h
rook-ceph-osd-mldev-storage1-62brv  1/1       Running   0          1h
```

We can also see that our data is now being stored in the correct place on **`/volb`** on **`storage0`**, instead of inside of in **`/var/lib/rook`**:

```
ubuntu@mldev-storage0:~$ sudo ls -al /volb/osd0
total 2316
drwxr--r-- 3 root root        4096 May 23 19:43 .
drwxr-xr-x 4 root root        4096 May 23 19:43 ..
lrwxrwxrwx 1 root root          26 May 23 19:43 block -> /volb/osd0/bluestore-block
lrwxrwxrwx 1 root root          23 May 23 19:43 block.db -> /volb/osd0/bluestore-db
lrwxrwxrwx 1 root root          24 May 23 19:43 block.wal -> /volb/osd0/bluestore-wal
-rw-r--r-- 1 root root           2 May 23 19:43 bluefs
-rw-r--r-- 1 root root 79132893184 May 23 19:43 bluestore-block
-rw-r--r-- 1 root root  1073741824 May 23 19:43 bluestore-db
-rw-r--r-- 1 root root   603979776 May 23 20:21 bluestore-wal
-rw-r--r-- 1 root root          37 May 23 19:43 ceph_fsid
-rw-r--r-- 1 root root          37 May 23 19:43 fsid
-rw-r--r-- 1 root root          56 May 23 19:43 keyring
-rw-r--r-- 1 root root           8 May 23 19:43 kv_backend
-rw-r--r-- 1 root root          21 May 23 19:43 magic
-rw-r--r-- 1 root root           4 May 23 19:43 mkfs_done
-rw-r--r-- 1 root root          24 May 23 19:43 path_block.db
-rw-r--r-- 1 root root          25 May 23 19:43 path_block.wal
-rw-r--r-- 1 root root           6 May 23 19:43 ready
-rw-r--r-- 1 root root        1689 May 23 19:43 rook.config
srwxr-xr-x 1 root root           0 May 23 19:43 rook-osd.0.asok
drwxr--r-- 2 root root        4096 May 23 19:43 tmp
-rw-r--r-- 1 root root          10 May 23 19:43 type
-rw-r--r-- 1 root root           2 May 23 19:43 whoami
```

Unfortunately my test case was still failing, as writing even the smallest amount of data to the shared filesystem would cause the container to lock up:

```
ubuntu@mldev-master:~$ kubectl exec -it deployment-demo-c59b896c8-2llsl -c busybox -- sh
/ # ls -al /data/
total 4
drwxr-xr-x    1 root     root             0 May 23 19:56 .
drwxr-xr-x    1 root     root          4096 May 23 19:54 ..
-rw-r--r--    1 root     root             0 May 23 19:54 index.html
-rw-r--r--    1 root     root             0 May 23 19:56 testing.html
/ # cat /data/testing.html
/ # echo '<div>Hello, World!</div>' > /data/testing.html
^C^C^C
^C
^C
^C
^C
^C^C^C^C

^C
^C^C^C^C^C


rm -rf /data/^C^C^C
^C
^C^C
rm -rf /data/testing.html
^C^C^C^C

^C
^C
^C^C^C^Cubuntu@mldev-master:~$
```

Fortunately, `kubectl describe pod` gave me some kind of lead:

```
ubuntu@mldev-master:~$ kubectl describe pod deployment-demo-c59b896c8-2llsl
Name:           deployment-demo-c59b896c8-2llsl
Namespace:      default
Node:           mldev-storage0/192.168.0.3
Start Time:     Wed, 23 May 2018 19:51:41 +0000
Labels:         demo=deployment
                pod-template-hash=715645274
                version=v1

   ...    ...    ...    ...Events:
  Type     Reason                 Age   From                     Message
  ----     ------                 ----  ----                     -------
  Normal   Scheduled              37m   default-scheduler        Successfully assigned deployment-demo-
c59b896c8-2llsl to mldev-storage0
  Normal   SuccessfulMountVolume  37m   kubelet, mldev-storage0  MountVolume.SetUp succeeded for volume
"default-token-hx8hk"
  Warning  FailedMount            35m   kubelet, mldev-storage0  Unable to mount volumes for pod "deployment-
demo-c59b896c8-2llsl_default(b602a2dc-5ec2-11e8-92e6-fa163e8f23f7)": timeout expired waiting for volumes to
attach/mount for pod "default"/"deployment-demo-c59b896c8-2llsl". list of unattached/unmounted volumes=[content]
  Warning  FailedMount            35m   kubelet, mldev-storage0  Unable to mount volumes for pod "deployment-
demo-c59b896c8-2llsl_default(b602a2dc-5ec2-11e8-92e6-fa163e8f23f7)": timeout expired waiting for volumes to
attach/mount for pod "default"/"deployment-demo-c59b896c8-2llsl". list of unattached/unmounted volumes=[content]
  Warning  FailedMount            35m   kubelet, mldev-storage0  MountVolume.SetUp failed for volume "content"
: mount command failed, status: Failure, reason: failed to mount filesystem myfs to /var/lib/kubelet/pods
/b602a2dc-5ec2-11e8-92e6-fa163e8f23f7/volumes/rook.io~rook/content with monitor 10.99.14.47:6790,10.104.100.195:
6790,10.110.166.39:6790:/ and options [name=admin secret=AQBRwgVbTxaqNxAA9tGopJpso/BU8xEQO1jNgQ==]: mount
failed: exit status 32
Mounting command: systemd-run
Mounting arguments: --description=Kubernetes transient mount for /var/lib/kubelet/pods/b602a2dc-5ec2-11e8-92e6-
fa163e8f23f7/volumes/rook.io~rook/content --scope -- mount -t ceph -o name=admin,
secret=AQBRwgVbTxaqNxAA9tGopJpso/BU8xEQO1jNgQ== 10.99.14.47:6790,10.104.100.195:6790,10.110.166.39:6790:/ /var
/lib/kubelet/pods/b602a2dc-5ec2-11e8-92e6-fa163e8f23f7/volumes/rook.io~rook/content
Output: Running scope as unit run-rbddb8b0550494bf5a46ecf0173cdced7.scope.
mount: 10.99.14.47:6790,10.104.100.195:6790,10.110.166.39:6790:/: can't read superblock
```

Bah! What is it now?

This could be a side-effect of one or more of the following changes:

- Upgrading to Rook 0.7.1
- The adjustments to the `directories` configuration in `rook-cluster.yaml` are now writing data to the correct drive (/volb), but that drive may be improperly formatted for use with Rook

Narrowing it down...

Adjusting my rook-cluster.yaml to only include the adjustments to the `directories` configuration, and to use `storeType: filestore` instead of `bluestore`.

This will hopefully prevent Rook from mangling our files on disk, and simplify our backups process:

```
#!/bin/bash
git clone https://github.com/groundnuty/k8s-wait-for.git

sudo helm repo add rook-alpha https://charts.rook.io/alpha
sudo helm install rook-alpha/rook --name rook --version 0.6.2


   ...    ...    ...    ...


# Now we can install the cluster
echo '
apiVersion: v1
kind: Namespace
metadata:
  name: rook
---
apiVersion: rook.io/v1alpha1
kind: Cluster
metadata:
  name: rook
  namespace: rook
spec:
  versionTag: v0.6.2
  dataDirHostPath: /var/lib/rook
  # cluster level storage configuration and selection
  storage:
    useAllNodes: false
    useAllDevices: false
    deviceFilter:
    metadataDevice:
    location:
    storeConfig:
      storeType: filestore
    nodes:
    - name: "mldev-storage0"
      directories:         # specific directories to use for storage can be specified for each node
      - path: "/rook"      # if changed, also adjust the mount path in bootstrap-rook.sh
    - name: "mldev-storage1"
      directories:         # specific directories to use for storage can be specified for each node
      - path: "/rook"      # if changed, also adjust the mount path in bootstrap-rook.sh
' | kubectl create -f -


# And create a storage class
wget -q https://raw.githubusercontent.com/zonca/jupyterhub-deploy-kubernetes-jetstream/master/storage_rook/rook-
storageclass.yaml
sudo kubectl create -f rook-storageclass.yaml
```

Sadly, changing to filestore did not unmangle the files:

```
ubuntu@mldev-storage0:~$ sudo ls -al /rook/osd1/
total 5242956
drwxr--r--   4 root root       4096 May 24 16:03 .
drwxr-xr-x   4 root root       4096 May 24 16:03 ..
-rw-r--r--   1 root root         37 May 24 16:03 ceph_fsid
drwxr-xr-x 508 root root      20480 May 24 16:11 current
-rw-r--r--   1 root root         37 May 24 16:03 fsid
-rw-r--r--   1 root root 5368709120 May 24 16:12 journal
-rw-r--r--   1 root root         56 May 24 16:03 keyring
-rw-r--r--   1 root root         21 May 24 16:03 magic
-rw-r--r--   1 root root          6 May 24 16:03 ready
-rw-r--r--   1 root root       1283 May 24 16:03 rook.config
srwxr-xr-x   1 root root          0 May 24 16:03 rook-osd.1.asok
-rw-r--r--   1 root root          4 May 24 16:03 store_version
-rw-r--r--   1 root root         53 May 24 16:03 superblock
drwxr--r--   2 root root       4096 May 24 16:03 tmp
```

```
-rw-r--r--   1 root root         10 May 24 16:03 type
-rw-r--r--   1 root root          2 May 24 16:03 whoami
ubuntu@mldev-storage0:~$ sudo ls -al /rook/osd1/current
total 4080
drwxr-xr-x 508 root root 20480 May 24 16:11 .
drwxr--r--   4 root root  4096 May 24 16:03 ..
drwxr-xr-x   2 root root  4096 May 24 16:04 1.0_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.10_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.10_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.11_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.11_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:06 1.12_head
drwxr-xr-x   2 root root  4096 May 24 16:06 1.12_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.13_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.13_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.14_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.14_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:06 1.15_head
drwxr-xr-x   2 root root  4096 May 24 16:06 1.15_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.16_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.16_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:06 1.17_head
drwxr-xr-x   2 root root  4096 May 24 16:06 1.17_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.18_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.18_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.19_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.19_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.1a_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.1a_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.1b_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.1b_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:06 1.1c_head
drwxr-xr-x   2 root root  4096 May 24 16:06 1.1c_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.1d_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.1d_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.1e_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.1e_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:06 1.1f_head
drwxr-xr-x   2 root root  4096 May 24 16:06 1.1f_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:06 1.1_head
drwxr-xr-x   2 root root  4096 May 24 16:06 1.1_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:06 1.20_head
drwxr-xr-x   2 root root  4096 May 24 16:06 1.20_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:04 1.21_head
drwxr-xr-x   2 root root  4096 May 24 16:04 1.21_TEMP
    ...    ...    ...    ...    ...    ...
drwxr-xr-x   2 root root  4096 May 24 16:09 3.45s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.46s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.46s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.47s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.47s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.49s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.49s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.4es0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.4es0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.4s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.4s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.50s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.50s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.51s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.51s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.52s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.52s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.55s0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.55s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.56s0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.56s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.57s0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.57s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.59s0_head
```

```
drwxr-xr-x   2 root root  4096 May 24 16:10 3.59s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.5as0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.5as0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.5ds0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.5ds0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.5es0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.5es0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.5fs0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.5fs0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.62s0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.62s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:10 3.63s0_head
drwxr-xr-x   2 root root  4096 May 24 16:10 3.63s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.6s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.6s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.7s0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.7s0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.bs0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.bs0_TEMP
drwxr-xr-x   2 root root  4096 May 24 16:09 3.es0_head
drwxr-xr-x   2 root root  4096 May 24 16:09 3.es0_TEMP
-rw-r--r--   1 root root     4 May 24 16:12 commit_op_seq
drwxr-xr-x   2 root root 12288 May 24 16:12 meta
-rw-r--r--   1 root root     0 May 24 16:03 nosnap
drwxr-xr-x   2 root root  4096 May 24 16:03 omap
```

But it did actively prevent me from using a shared filesystem:

```
2018-05-24 16:08:39.653246 I | exec: Error EINVAL: pool must only be stored on bluestore for scrubbing to work:
osd.1 uses filestore
2018-05-24 16:08:39.653337 W | cephmds: failed to set ec pool property. failed to set pool property
allow_ec_overwrites on pool myfs-data0, exit status 22
2018-05-24 16:08:39.653422 I | exec: Running command: ceph fs flag set enable_multiple true --yes-i-really-mean-
it --cluster=rook --conf=/var/lib/rook/rook/rook.config --keyring=/var/lib/rook/rook/client.admin.keyring --
format json --out-file /tmp/566066704
2018-05-24 16:08:43.644068 I | exec: Running command: ceph fs new myfs myfs-metadata myfs-data0 --cluster=rook
--conf=/var/lib/rook/rook/rook.config --keyring=/var/lib/rook/rook/client.admin.keyring --format json --out-
file /tmp/808516655
2018-05-24 16:08:45.443925 I | exec: Error EINVAL: pool 'myfs-data0' (id '3') is an erasure-coded pool, with no
overwrite support
2018-05-24 16:08:45.444031 E | op-mds: failed to create file system myfs. failed to create file system myfs:
failed enabling ceph fs myfs. exit status 22
```

Confirmed by this GitHub issue: https://github.com/rook/rook/issues/1604

Back to bluestore...

Switching back to `storeType: bluestore` on Rook v0.6.2 with the correct `nodes/directories` configuration:

```
#!/bin/bash
git clone https://github.com/groundnuty/k8s-wait-for.git

sudo helm repo add rook-alpha https://charts.rook.io/alpha
sudo helm install rook-alpha/rook --name rook --version 0.6.2


   ...    ...    ...    ...


# Now we can install the cluster
echo '
apiVersion: v1
kind: Namespace
metadata:
  name: rook
---
apiVersion: rook.io/v1alpha1
kind: Cluster
metadata:
  name: rook
  namespace: rook
spec:
  versionTag: v0.6.2
  dataDirHostPath: /var/lib/rook
  # cluster level storage configuration and selection
  storage:
    useAllNodes: false
    useAllDevices: false
    deviceFilter:
    metadataDevice:
    location:
    storeConfig:
      storeType: bluestore
      databaseSizeMB: 1024 # this value can be removed for environments with normal sized disks (100 GB or
larger)
      journalSizeMB: 1024  # this value can be removed for environments with normal sized disks (20 GB or
larger)
    - name: "mldev-storage0"
      directories:         # specific directories to use for storage can be specified for each node
      - path: "/rook"      # if changed, also adjust the mount path in bootstrap-rook.sh
    - name: "mldev-storage1"
      directories:         # specific directories to use for storage can be specified for each node
      - path: "/rook"      # if changed, also adjust the mount path in bootstrap-rook.sh
' | kubectl create -f -


# And create a storage class
wget -q https://raw.githubusercontent.com/zonca/jupyterhub-deploy-kubernetes-jetstream/master/storage_rook/rook-
storageclass.yaml
sudo kubectl create -f rook-storageclass.yaml
```

With this configuration, the filesystem pods came up just fine, but this exhibited the same behavior as the case above, where writing to a file causes the container to hang indefinitely.

The only way to recover in this case is to Ctrl+P, Ctrl+Q to abandon the `kubectl exec` command, Ctrl+C-ing as necessary - this leaves a hanging exec process, but it is still unclear whether there are performance concerns about executing this case repeatedly without rebooting.

I have noticed that cluster with pods in an error state such as this one will fail to terraform destroy (the operation never completes even after waiting 15+ minutes)

Resolution

After pouring through the docs and GitHub issues and tediously reading the source code, we found a concerning comment in a GitHub issue: https://github.com/rook/rook/issues/1220#issuecomment-343342515

It turns out that this example shared filesystem requires 3 nodes to function - this is why it was mysteriously failing when only 2 OSDs were present.

All in all, as long as you have more OSDs than the `spec.metadataPool.replicated.size` you have configured in `rook-filesystem.yaml`

In our sample cluster we have 2 storage nodes, so setting this value to **2** fixed our test case.

The final working configuration can be found below:

rook-cluster.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: rook
---
apiVersion: rook.io/v1alpha1
kind: Cluster
metadata:
  name: rook
  namespace: rook
spec:
  versionTag: v0.6.2
  dataDirHostPath: /var/lib/rook
  # cluster level storage configuration and selection
  storage:
    useAllNodes: false
    useAllDevices: false
    deviceFilter:
    metadataDevice:
    location:
    storeConfig:
      storeType: bluestore
      databaseSizeMB: 1024 # this value can be removed for environments with normal sized disks (100 GB or
larger)
      journalSizeMB: 1024  # this value can be removed for environments with normal sized disks (20 GB or
larger)
    - name: "mldev-storage0"
      directories:         # specific directories to use for storage can be specified for each node
      - path: "/rook"      # if changed, also adjust the mount path in bootstrap-rook.sh
    - name: "mldev-storage1"
      directories:         # specific directories to use for storage can be specified for each node
      - path: "/rook"      # if changed, also adjust the mount path in bootstrap-rook.sh
```

rook-filesystem.yaml

```
apiVersion: rook.io/v1alpha1
kind: Filesystem
metadata:
  name: myfs
  namespace: rook
spec:
  metadataPool:
    replicated:
      size: 2
  dataPools:
    - erasureCoded:
       dataChunks: 2
       codingChunks: 1
  metadataServer:
    activeCount: 1
    activeStandby: true
```

Recovering from backup

This feature is currently in the planning stages: https://github.com/rook/rook/issues/1552

Unofficial Python script for creating / restoring backups from Rook: https://gitlab.com/costrouc/kubernetes-rook-backup

Edge Cases and Quirks

There are many pitfalls here, particularly surrounding my perceived fragility of the shared filesystem

DO NOT delete the filesystem before shutting down all of the pods consuming it

Deleting the shared filesystem out from under the pod will confuse the kubelet, and prevent it from being able to properly unmount and terminate your containers.

If you need to shut down your shared filesystem, please ensure that you first shut down any pods consuming it's storage.

This will hopefully be improved in later versions of Kubernetes (1.9+?)

You must follow these cleanup steps before `terraform destroy` will work

Expanding on the above topic, `terraform destroy` will hang on destroying your cluster if you fail to cleanup your filesystems properly:

1. Shut down any pods that are consuming the shared filesystem (e.g. any clients)
2. Shut down the filesystem pods (e.g. any servers)
3. Run `terraform destroy` to cleanup all other cluster resources

WARNING: this seems like a very tenuous/tedious process... I am hoping that later versions of terraform/rook will improve the stability of cleanup under these scenarios. Perhaps we can expand their cleanup to first drain all nodes of their running pods (if this is not already the case), although this would not fix the case of a user deleting the filesystem before a running pod that is consuming it - in this case, the pods will fail to terminate indefinitely, which I think is what is leading terraform to fail.

Kill (or hide) a hanging pod

There are a few ways to kill a pod with fire:

```
# Attempt to shorten the grace period
ubuntu@mldev-master:~$ kubectl delete pod deployment-demo-c59b896c8-8hgnm --grace-period=1
pod "deployment-demo-c59b896c8-8hgnm" deleted


# If that doesn't work, you can try to force it
ubuntu@mldev-master:~$ kubectl delete pod deployment-demo-c59b896c8-8hgnm --grace-period=1 --force
pod "deployment-demo-c59b896c8-8hgnm" deleted


# If your pod still doesn't terminate, you can change try it with --now instead:
ubuntu@mldev-master:~$ kubectl delete pod deployment-demo-c59b896c8-8hgnm --now
pod "deployment-demo-c59b896c8-8hgnm" deleted
ubuntu@mldev-master:~$ kubectl delete pod deployment-demo-c59b896c8-8hgnm --now --force
pod "deployment-demo-c59b896c8-8hgnm" deleted


# As a last resort, drop the grace period to zero
# WARNING: Only use this as a last resort, and never use it in production!
ubuntu@mldev-master:~$ kubectl delete pod deployment-demo-c59b896c8-8hgnm --grace-period=0 --force
warning: Immediate deletion does not wait for confirmation that the running resource has been terminated. The
resource may continue to run on the cluster indefinitely.
pod "deployment-demo-c59b896c8-8hgnm" deleted
```

Note that the last method may not actually cleanup all resources properly, as noted in the output above.

Only use this as a last resort on test clusters, and NEVER use **`--grace-period=0`** on a production cluster.

Cleaning up failed runs of `terraform destroy`

Here is a quick checklist of the items that you will need to manually if you are unable to `terraform destroy` your cluster:

1. Under Compute: Instances, then Volumes
2. Under Compute Access & Security: Security Groups, then Keypair
3. Under Network: Router, then Network

You will then need to manually delete your terraform state:

```
rm -rf terraform.tfstate*
```