Building Blocks

How Raw Block PersistentVolumes Changed the Way We Look at Storage

Rohan Gupta, Red Hat
@rohan47
Jose A. Rivera, Red Hat
@jarrpa

WARNING

The following presentation may contain opinions, speculations, and bad jokes. These are entirely the responsibility and fault of the presenters, and do not reflect the values of Red Hat, IBM, or the Rook project.

Introductions and Agenda

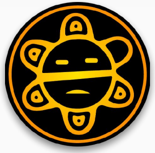
Introductions

Rohan Gupta

Associate Software Engineer, **Red Hat**

- Graduated from college in 2018.
- Did GSoC with CNCF and worked on adding NFS operator in Rook.
- Working on OpenShift Container Storage (OCS) focusing on Rook upstream.
- Loves watching anime and riding motorbikes.

Jose A. Rivera



Senior Software Engineer, **Red Hat**

- In and around storage for over 10 years.
- Works on OpenShift Container Storage (OCS), focusing on Rook and Ceph
- Project lead for the OCS Operator.
- Participates in SIG Storage.
- Likes hitting things, mostly drums.

Agenda

0. Introductions and Agenda ← you are here

1. Setting the Stage
   - Storage in Kubernetes
   - Raw Block PVs
   - Rook and Rook-Ceph

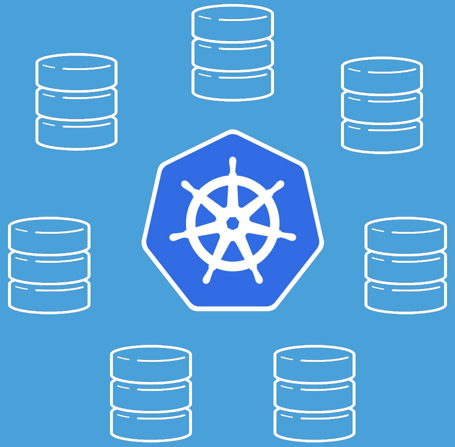
2. Developing the Characters
   - OSDs: Then and Now
   - Bumps in the Road

3. Putting on a Show
   - Demo Time!

Setting the Stage

Storage In Kubernetes

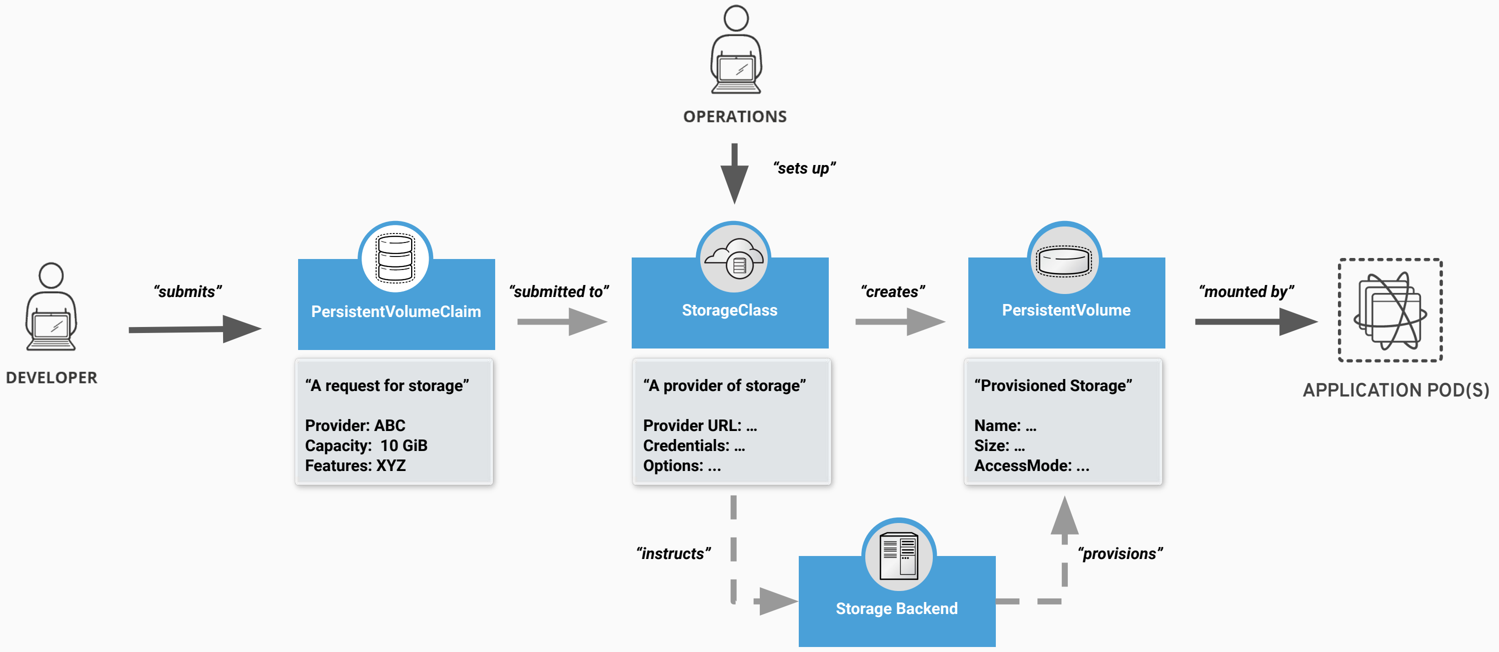
A primer



Storage Resource Types

- PersistentVolumes (PVs)
  - Represents a volume of storage
  - Different backends define what a "volume" represents
- PersistentVolumeClaims (PVCs)
  - Represents a request to use storage
- StorageClasses (SCs)
  - Provides a point PVCs can use for dynamic provisioning of PVs

Dynamic Provisioning



"A request for storage"

Provider: ABC
Capacity: 10 GiB
Features: XYZ

"A provider of storage"

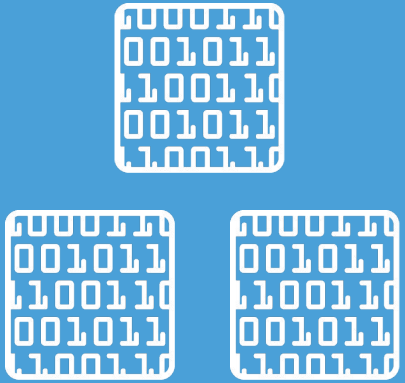
Provider URL: ...
Credentials: ...
Options: ...

"Provisioned Storage"

Name: ...
Size: ...
AccessMode: ...

Raw Block PVs

The new kid in town



Why Raw Block PVs?

- Allows Kubernetes to present storage to containers without a formatted filesystem
- Many applications, like databases (MongoDB, Cassandra), can leverage block storage directly, with no additional configuration
- Allows certain storage providers to provide more consistent I/O performance and lower latency

https://kubernetes.io/docs/concepts/storage/persistent-volumes/#raw-block-volume-support

VolumeMode: File vs Block

VolumeMode, a new field, is how you use the feature

- In Beta since Kubernetes 1.13
- Specifies how the storage will be accessed i.e., as a filesystem or raw block device
- **VolumeMode: Block** must be set on both the PV and the PVC
- **VolumeMode: File** is the backwards-compatible default

VolumeMode: File

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: file-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: File ← can omit

  ...
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: file-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: File ← can omit
  resources:
    requests:
      storage: 10Gi
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-file-volume
spec:
  containers:
    - name: busybox
      image: busybox
      command:
        - sleep
        - "3600"
      volumeMounts:
        - name: data
          mountPath: "/mnt/foo"
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: file-pvc
```

VolumeMode: Block

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: block-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  ...
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 10Gi
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-block-volume
spec:
  containers:
    - name: busybox
      image: busybox
      command:
        - sleep
        - "3600"
      volumeDevices:
        - name: data
          devicePath: /dev/vda
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: block-pvc
```

VolumeMode vs. AccessMode

These are not synonymous nor related

- **Access Modes** (i.e. RWX, RWO) denote how many Pods may attach a PVC at a time and whether or not they can write to it
- Certain storage drivers that provide raw block volumes may only support a subset of the Access Modes their file volumes provide
    - This is typically a limitation of the storage attachment technology

Rook and Rook-Ceph

Cloud-native, software-defined storage

What is Rook?

- Storage Operators for Kubernetes
- Automate
  - Deployment
  - Bootstrapping
  - Configuration
  - Upgrading

Rook Operators

- Implement the **Operator Pattern** for storage solutions
- Define *desired state* for the storage resource
  - Storage Cluster, Pool, Object Store, etc.
- Reconcile the *actual state* to match the desired state
  - Watch for changes in desired state
  - Watch for changes in the cluster
  - Apply changes to the cluster to make it match desired state

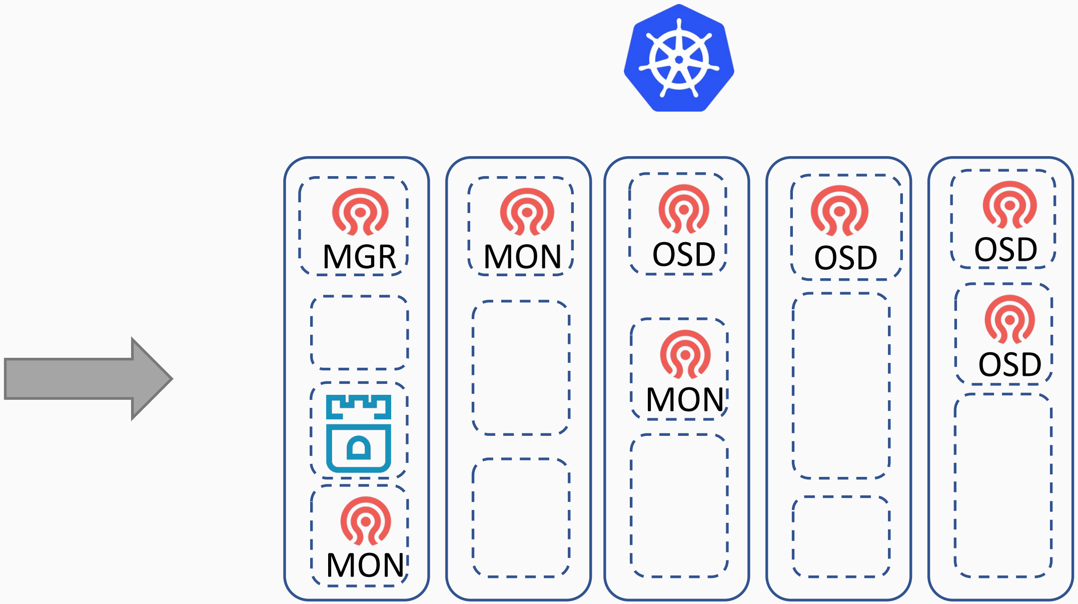
https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

Rook-Ceph

- Ceph in containers
- Resilient, distributed storage
  - Self-healing
- Highly scalable
- Runs on commodity hardware
- Fully open source!

Rook-Ceph

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
spec:
  cephVersion:
    image: ceph/ceph:v14
  mon:
    count: 3
  network:
    hostNetwork: false
  storage:
    useAllNodes: true
```



https://github.com/rook/rook/blob/master/Documentation/ceph-cluster-crd.md

Developing the Characters

OSDs:
Then and Now

Presenting devices to Ceph

Local Storage OSDs

- Define storage nodes
  - Names, labels, or all
- Define local devices
  - Manual or auto-discover
- Rook automation
  - Prepare devices
  - Start OSD Pod

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
spec:
  ...
  storage:
    useAllNodes: true
    useAllDevices: true
```

Local Storage OSDs

Pros:

- Easy to configure
- Familiar
- Supports any type of device/appliance that Linux supports

Cons:

- Rely on specialized nodes
- Rigid coupling between compute and storage

StorageClassDeviceSets

- Define storage nodes
  - Names, labels, or all
- Define desired amount of storage
- Rook automation
  - Prepare devices
  - Start OSD Pod

```
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
spec:
  ...
  storage:
    storageClassDeviceSets:
      ...
```

StorageClassDeviceSets

- SCDSs were designed to be a generic Rook struct
    - Some features not used in Rook-Ceph
- **name:** use for generating unique and consistent PVC names
- **count:** number of devices in this set

```
storageClassDeviceSets:
  - name: set1
    count: 3
    portable: true
    volumeClaimTemplates:
      - spec:
          resources:
            requests:
              storage: 10Gi
          storageClassName: gp2
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
```

StorageClassDeviceSets

- **portable:** PVCs are allowed to move between nodes
- **volumeClaimTemplates:** a list of PVC templates
  - Just a standard PVC spec
  - Only one is supported for Rook-Ceph
    - More may be supported for more advanced features later

```
storageClassDeviceSets:
  - name: set1
    count: 3
    portable: true
    volumeClaimTemplates:
      - spec:
          resources:
            requests:
              storage: 10Gi
          storageClassName: gp2
          volumeMode: Block
          accessModes:
            - ReadWriteOnce
```

StorageClassDeviceSets

Pros:

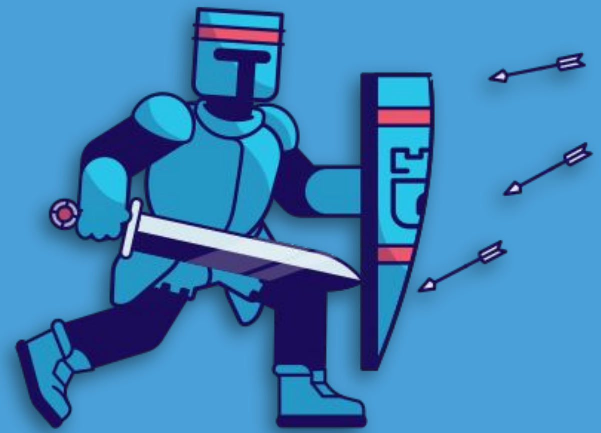
- Offload device distribution
- Device migration between nodes
- Works with any raw block PVs, regardless of driver
- Shiny and new 😀

Cons:

- Requires pre-defined StorageClasses
- Device support limited by what's in Kubernetes
- Not as simple to configure
- New and different 😒

Bumps in the Road

Gotchas and caveats



Check Your Privilege

Problem: OSD Pods run as **privileged** Pods

- Host's `/dev` is bind-mounted into the container
- Prevents Kubernetes from presenting the block device at the desired path

Solution: Use a non-privileged init container to copy the device (it's just a file!) to an emptyDir shared between the init container and the privileged container (hat tip to **John Strunk**)

Check Your Privilege

```
apiVersion: v1
kind: Pod
spec:
  ...
  containers:
  - command: ["/rook/tini"]
    args:
    - --
    - /rook/rook
    - ceph
    - osd
    - start
    ...
    name: osd
    volumeMounts:
    - mountPath: /mnt
      name: set1-dev0-bridge
    ...
```

```
initContainers:
- command: ["cp"]
  args: ["-a","/set1-dev0","/mnt/set1-dev0"]
  name: blkdevmapper
  volumeDevices:
  - devicePath: /set1-dev0
    name: set1-dev0
  volumeMounts:
  - mountPath: /mnt
    name: set1-dev0-bridge
  ...
volumes:
- name: set1-dev0
  persistentVolumeClaim:
    claimName: set1-dev0
- emptyDir:
    medium: Memory
    name: set1-dev0-bridge
...
```

Virtually Lost

Problem: When spinning up multiple OSDs on the same node, some OSDs would be unable to find their storage devices

- Rook-Ceph uses LVM for the OSD devices
- Kubernetes creates a loopback device for the storage device
- Because `/dev` is mounted, this led to the LVM LV having two PV references, which confused `ceph osd start` command

Solution: Pass the exact path to the LV (e.g. `/dev/<vg_name>/<lv_name>`) that was used by the OSD prepare Job to the OSD daemon

Proper Distribution

Problem: OSDs were clustering on few nodes

- Reduces data resiliency
- Potentially increases volume recovery time

Solution: Use placement affinities

```yaml
name: set1
count: 3
portable: true
...
placement:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: app
                operator: In
                values:
                  - rook-ceph-osd
            topologyKey: kubernetes.io/hostname
```

Putting on a Show

Demo Time!

The moment of truth

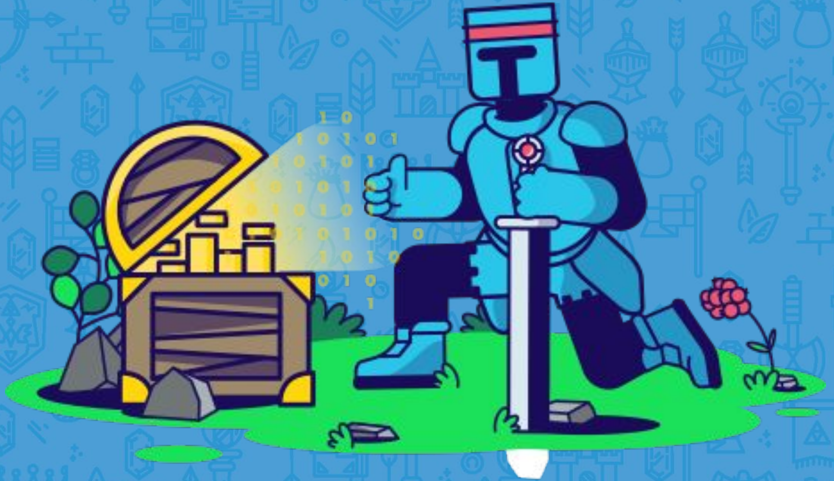
Thanks!

https://github.com/rook/rook

https://rook.io/

@rohan47

@jarrpa



But wait, there's more!

But wait, there's more!
What about on-premises??

Local Block PVs

Allows Kubernetes to access a local volume via the PVC/PV interface.

Create a PV with a reference to a **StorageClass**

Specify **node affinity**

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-block-pv
spec:
  capacity:
    storage: 500Gi
  accessModes:
  - ReadWriteOnce
  volumeMode: Block
  storageClassName: local-storage
  local:
    path: /mnt/disks/vol1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - my-node
```

Local Block PVs

Create a StorageClass that uses **no-provisioner** and **topology-aware provisioning**, which will allow the Pod scheduler to take the locality of the PV into account.

Create PVC and Pod as normal.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-block-pvc
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Block
  resources:
    requests:
      storage: 500Gi
```

Thanks, again!

https://github.com/rook/rook

https://rook.io/

@rohan47

@jarrpa