Rook：

一个自我管理的分布式存储编排系统，它本身并不是存储系统，在存储和k8s之前搭建了一个桥梁，存储系统的搭建或者维护变得特别简单，Rook支持CSI，CSI做一些PVC的快照、PVC扩容等操作。

Operator：主要用于有状态的服务，或者用于比较复杂应用的管理。

Helm：主要用于无状态的服务，配置分离。

Rook：

Agent：在每个存储节点上运行，用于配置一个FlexVolume插件，和k8s的存储卷进行集成。挂载网络存储、加载存储卷、格式化文件系统。

Discover：主要用于检测链接到存储节点上的存储设备。

Ceph：

OSD：直接连接每一个集群节点的物理磁盘或者是目录。集群的副本数、高可用性和容错性。

MON：集群监控，所有集群的节点都会向Mon汇报。他记录了集群的拓扑以及数据存储位置的信息。

MDS：元数据服务器，负责跟踪文件层次结构并存储ceph元数据。

RGW：restful API接口。

MGR：提供额外的监控和界面。

忽略1.3的视频

Rook官方文档：<https://rook.io/docs/rook/v1.3/ceph-quickstart.html>

git clone --single-branch --branch release-1.3 https://github.com/rook/rook.git

OSD配置：<https://rook.io/docs/rook/v1.3/ceph-cluster-crd.html#osd-configuration-settings>

git clone --single-branch --branch release-1.2 https://github.com/rook/rook.git

apiVersion: ceph.rook.io/v1

kind: CephBlockPool

metadata:

name: replicapool

namespace: rook-ceph

spec:

failureDomain: host

replicated:

size: 3

apiVersion: apps/v1

kind: StatefulSet

metadata:

creationTimestamp: "2020-04-03T13:49:26Z"

generation: 1

labels:

app: test-block-sts

name: test-block-sts

namespace: ratel-test1

resourceVersion: "5517946"

selfLink: /apis/apps/v1/namespaces/ratel-test1/statefulsets/test-block-sts

uid: 6b4de943-ad58-4d03-8602-50cc81056b58

spec:

podManagementPolicy: OrderedReady

replicas: 1

revisionHistoryLimit: 10

selector:

matchLabels:

app: test-block-sts

serviceName: test-block-sts

template:

metadata:

creationTimestamp: null

labels:

app: test-block-sts

spec:

containers:

- command:

- sh

- -c

- sleep 3600000

env:

- name: TZ

value: Asia/Shanghai

- name: LANG

value: C.UTF-8

image: nginx

imagePullPolicy: Always

lifecycle: {}

name: test-block-sts

ports:

- containerPort: 8080

name: web

protocol: TCP

resources:

limits:

cpu: 100m

memory: 100Mi

requests:

cpu: 10m

memory: 10Mi

securityContext:

privileged: false

runAsNonRoot: false

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

volumeMounts:

- mountPath: /usr/share/zoneinfo/Asia/Shanghai

name: tz-config

- mountPath: /etc/localtime

name: tz-config

- mountPath: /etc/timezone

name: timezone

- mountPath: /mnt

name: test-block

dnsPolicy: ClusterFirst

restartPolicy: Always

schedulerName: default-scheduler

securityContext: {}

terminationGracePeriodSeconds: 30

volumes:

- hostPath:

path: /usr/share/zoneinfo/Asia/Shanghai

type: ""

name: tz-config

- hostPath:

path: /etc/timezone

type: ""

name: timezone

updateStrategy:

rollingUpdate:

partition: 0

type: RollingUpdate

volumeClaimTemplates:

- metadata:

creationTimestamp: null

name: test-block

spec:

accessModes:

- ReadWriteOnce

resources:

limits:

storage: 10Gi

requests:

storage: 1Gi

storageClassName: rook-ceph-block

volumeMode: Filesystem

使用pvc动态申请pv

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: rook-ceph-test-pvc

spec:

accessModes:

- ReadWriteOnce

volumeMode: Filesystem

resources:

requests:

storage: 1Gi

storageClassName: rook-ceph-block

<https://rook.io/docs/rook/v1.2/ceph-filesystem.html>

<https://rook.io/docs/rook/v1.2/ceph-filesystem-crd.html>

PVC的扩容、PVC的快照和回滚。

<https://rook.io/docs/rook/v1.2/ceph-csi-drivers.html>

开启PVC的扩容和快照功能

tail -4 /etc/kubernetes/kubelet-conf.yml

featureGates:

EphemeralContainers: true

VolumeSnapshotDataSource: true

ExpandCSIVolumes: true

[root@k8s-master01 rook]# cat /usr/lib/systemd/system/kube-proxy.service

[Unit]

Description=Kubernetes Kube Proxy

Documentation=https://github.com/kubernetes/kubernetes

After=network.target

[Service]

ExecStart=/usr/local/bin/kube-proxy \

--config=/etc/kubernetes/kube-proxy.conf \

--feature-gates=EphemeralContainers=true,ExpandCSIVolumes=true,VolumeSnapshotDataSource=true \

--v=2

[root@k8s-master01 rook]# cat /etc/systemd/system/kubelet.service.d/10-kubelet.conf

[Service]

Environment="KUBELET\_KUBECONFIG\_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.kubeconfig --kubeconfig=/etc/kubernetes/kubelet.kubeconfig"

Environment="KUBELET\_SYSTEM\_ARGS=--network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin"

Environment="KUBELET\_CONFIG\_ARGS=--config=/etc/kubernetes/kubelet-conf.yml"

Environment="KUBELET\_EXTRA\_ARGS=--node-labels=node.kubernetes.io/node='' --pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google\_containers/pause-amd64:3.1" --feature-gates="EphemeralContainers=true,ExpandCSIVolumes=true,VolumeSnapshotDataSource=true"

删除集群的文档：<https://rook.io/docs/rook/v1.2/ceph-teardown.html>

创建一个storageClass

rook/cluster/examples/kubernetes/ceph/csi/rbd

XFS文件系统报错

Normal Scheduled <unknown> default-scheduler Successfully assigned default/csirbd-demo-pod-restore to k8s-master02

Normal SuccessfulAttachVolume 17s attachdetach-controller AttachVolume.Attach succeeded for volume "pvc-d0fb7ae3-b987-402a-8d2d-2392d7bd1966"

Warning FailedMount <invalid> (x4 over 3s) kubelet, k8s-master02 MountVolume.MountDevice failed for volume "pvc-d0fb7ae3-b987-402a-8d2d-2392d7bd1966" : rpc error: code = Internal desc = 'xfs\_repair' found errors on device /dev/rbd0 but could not correct them: Phase 1 - find and verify superblock...

Phase 2 - using internal log

- zero log...

ERROR: The filesystem has valuable metadata changes in a log which needs to

be replayed. Mount the filesystem to replay the log, and unmount it before

re-running xfs\_repair. If you are unable to mount the filesystem, then use

the -L option to destroy the log and attempt a repair.

Note that destroying the log may cause corruption -- please attempt a mount

of the filesystem before doing this.

K8s的问题

1. 1.18解决了这个问题（可以自己试一下）
2. 用xfs\_repair修复这个RBD
3. Ceph-csi 1.2.1 1.2.3（不推荐回滚版本）
4. 用ext4去代替xfs

Rook官方工具修复xfs\_repair

<https://rook.io/docs/rook/v1.2/direct-tools.html>

创建direct-tools：

[root@k8s-master01 rook]# kubectl create -f cluster/examples/kubernetes/ceph/direct-mount.yaml

Dashboard文档：<https://rook.io/docs/rook/v1.2/ceph-dashboard.html>

获取dashboard密码：

kubectl -n rook-ceph get secret rook-ceph-dashboard-password -o jsonpath="{['data']['password']}" | base64 --decode && echo

[root@k8s-master02 /]# rbd map replicapool/csi-vol-04df7d56-7a74-11ea-a4ed-7e6741ead7e8

/dev/rbd0

[root@k8s-master02 /]# ls /dev/rbd0

/dev/rbd0

[root@k8s-master02 /]# ls /dev/sda

sda sda1 sda2

[root@k8s-master02 /]# mount /dev/rbd0 /mnt/

[root@k8s-master02 /]# cd /mnt/

[root@k8s-master02 mnt]# ls

1 10 2 3 4 5 6 7 8 9

[root@k8s-master02 mnt]# cd

[root@k8s-master02 ~]# ls

anaconda-ks.cfg

[root@k8s-master02 ~]# umount /mnt/

[root@k8s-master02 ~]#

[root@k8s-master02 ~]# ls

anaconda-ks.cfg

[root@k8s-master02 ~]# xfs\_repair /dev/rbd0

Phase 1 - find and verify superblock...

Phase 2 - using internal log

- zero log...

- scan filesystem freespace and inode maps...

- found root inode chunk

Phase 3 - for each AG...

- scan and clear agi unlinked lists...

- process known inodes and perform inode discovery...

- agno = 0

- agno = 1

- agno = 2

- agno = 3

- agno = 4

- agno = 5

- agno = 6

- agno = 7

- process newly discovered inodes...

Phase 4 - check for duplicate blocks...

- setting up duplicate extent list...

- check for inodes claiming duplicate blocks...

- agno = 0

- agno = 1

- agno = 2

- agno = 3

- agno = 4

- agno = 5

- agno = 6

- agno = 7

Phase 5 - rebuild AG headers and trees...

- reset superblock...

Phase 6 - check inode connectivity...

- resetting contents of realtime bitmap and summary inodes

- traversing filesystem ...

- traversal finished ...

- moving disconnected inodes to lost+found ...

Phase 7 - verify and correct link counts...

done

[root@k8s-master02 ~]# xfs\_repair -L /dev/rbd0

卸载：rbd unmap /dev/rbd0