Alluxio K8S Operator 部署手册

1、 部署前环境检查

1) 操作系统环境检查

- 建议Linux的kernel在5.*，可以通过uname -a确认一下。
- 建议libfuse的版本在3.10以上，可以通过fusermount3 -V确认一下。

2) K8S环境检查

- 建议Kubernetes集群的版本1.19及以上，并enable gate feature。
- 确保集群的Kubernetes网络策略允许应用程序（Alluxio客户端）和Alluxio Pods之间通过定义的端口连接。
- Kubernetes集群安装了helm 3，版本建议为3.6.0及以上。
- 在Kubernetes集群中获得特定的RBAC权限以使Operator工作。
    - 在 Kubernetes 集群中获得特定的 RBAC 权限以使 Operator 工作
    - 创建 CRD（自定义资源）的权限
    - 允许 operator pod 创建 ServiceAccount、ClusterRole 和ClusterRoleBinding 的权限
    - 允许 operator 创建容纳自己的 namespace
    - 具体rbac.yaml文件位于operator部署包解压目录的 ./templates/rbac.yaml
    - 参考链接：使用 RBAC 鉴权
- Alluxio需要连接ETCD服务，如果已存在可用的ETCD服务，可以在后续配置中直接使用现有的ETCD服务。

  如果不存在现成可用的ETCD服务，Alluxio Operator中集成了ETCD官方提供的Operator，可以在启动Alluxio Operator时，同时启动ETCD服务。

  但K8S集群环境可能存在差异，需要进行如下操作：检查Kubernetes的storageclass，是否有名为gp2（一般是aws）或者standard的storageclass，并且确认该storageclass是否允许dynamic provision。

- 如果没有，方案1：创建对应storageclass并允许dynamic provision
- 如果没有，方案2：手动在Alluxio部署所在Namespace中创建pv，请注意hostpath的路径需要在启动ETCD的节点真实存在，并且具备被etcd pod访问的权限（或者直接777）

```
piVersion: v1
kind: PersistentVolume
metadata:
  name: alluxio

spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi
  hostPath:
    path: /tmp/host10/a
```

```
## 假设上面生成了一个etcd-pv.yaml文件
# create etcd pv
kubectl apply -f ./alluxio-operator/etcd-pv.yaml -n <namespace>

# delete pv pvc
kubectl delete pvc data-alluxio-etcd-{0..9}  -n <namespace>
kubectl delete -f ./alluxio-operator/etcd-pv.yaml  -n <namespace>
```

3) 服务器资源检查

- Alluxio缓存资源和路径准备
  - 准备Alluxio Worker部署节点的缓存盘，支持配置多块缓存盘，建议为高性能SSD盘。在所有Worker节点的缓存盘上创建相同路径的目录，如/data/alluxio
  - 如果要打开Alluxio FUSE的Local Cache，也需要进行上述操作，比如/data/fuse

- Alluxio FUSE挂载路径准备，需要提前创建好，并且确保权限是可以被K8S访问的。
- nodeSelector 标签：如果希望控制Alluxio的Worker、FUSE等的发布位置，可以提前准备好nodeSelector标签。
    - 如果Alluxio FUSE和Alluxio Worker并不在一起，需要准备两个nodeSelector标签。

2、 Alluxio资源和配置准备

1) Alluxio 相关镜像

Alluxio相关镜像			
镜像名	版本	用途	备注
alluxio-k8s-operator-<version>-docker.tar	根据情况选择对应版本	alluxio operator image	Alluxio提供
alluxio-csi-<version>-docker.tar	根据情况选择对应版本	alluxio csi image	Alluxio提供
alluxio-enterprise-AI-<version>-docker.tar	根据情况选择对应版本	alluxio images	Alluxio提供
公共镜像			
镜像名	版本	用途	备注
csi-node-driver-registrar	v2.0.0	csi driver registrar依赖	
csi-provisioner	v2.0.5	csi provisioner依赖	
etcd	3.5.9-debian-11-r24	etcd依赖	
os-shell	11-debian-11-r2	os-shell依赖	

grafana	11.1.0-ubuntu	监控展示	
prometheus	v2.52.0	指标采集	
operator charts			
charts name	版本	用途	备注
alluxio-operator-<version>-helmchart.tgz	根据情况选择对应版本	部署alluxio operator	Alluxio提供

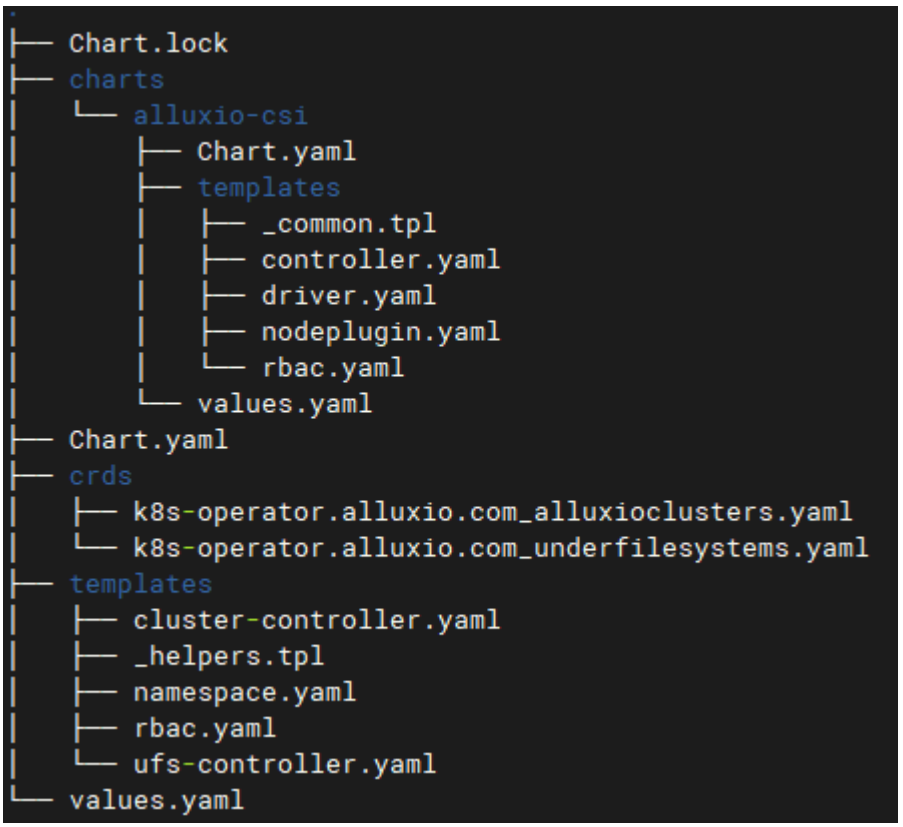
2) 资源下载

基于版本需要及商业授权原因，沟通一致后，由Alluxio提供对应版本及商业授权的下载链接。

3) Alluxio yaml配置

Alluxio yaml配置模板

见压缩包alluxio-operator-<version>-helmchart.tgz，解压后为alluxio-operator，首层目录架构如下：



- 其中**values.yaml**为**alluxio-operator.yaml**

alluxio-cluster.yaml

1) 把Alluxio提供的镜像推送到私有仓库，并修改alluxio-cluster.yaml中的image和imageTag
    a. docker load ***
    b. docker tag ***
    c. docker push ***
2) 绑定SSD Path（建议使用SSD盘），hostPath: /data/alluxio/***
3) yaml中包含了alluxio master、worker、fuse、etcd等配置信息
4) 其他Alluxio配置根据需要调整
    a. Alluxio的page size等根据需要配置
    b. Alluxio FUSE的fuse相关配置，比如Kernel cache、timeout参数等根据需要决定是否启用

```
apiVersion: k8s-operator.alluxio.com/v1
kind: AlluxioCluster
metadata:
  name: alluxio
spec:
  image: ****/****
  imageTag: *****
## 对应的是Alluxio EE 的 docker image
  imagePullPolicy: IfNotPresent

## pod的用户/组
  user: 0
  group: 0
  fsGroup: 0

# 提前将ufs mount需要的hdfs-site、core-site等配置通过configMap上传，示例
  configMaps:
    master:
      hdfs-site: /opt/alluxio/conf/hdfs-site.xml #mountPath指的是容器内的路径
      core-sitel: /opt/alluxio/conf/core-site.xml
    worker:
      hdfs-site: /opt/alluxio/conf/hdfs-site.xml
      core-site: /opt/alluxio/conf/core-site.xml
#hostPaths:示例
  hostPaths:
    worker:
      /data/alluxio_data/data: /ufs/data
    fuse:
      /data/alluxio_data/data: /ufs/data
    master:
      /data/alluxio_data/data: /ufs/data
# worker的pv、pvc配置，示例
  pvcMounts:
    worker:
      gluster-pvc2-volume: /nfs/volume-115-1
```

```
  fuse:
    gluster-pvc2-volume: /nfs/volume-115-1
  master:
    gluster-pvc2-volume: /nfs/volume-115-1

# Whether using hostNetwork for all Alluxio pods.
hostNetwork: false
# If hostNetwork is false, dnsPolicy defaults to ClusterFirst.
# If hostNetwork is true, dnsPolicy defaults to ClusterFirstWithHostNet.

properties:
  # Common properties
  alluxio.cluster.name: "default-alluxio"
  alluxio.dora.enabled: "true"
  alluxio.network.netty.heartbeat.timeout: "5min"
  alluxio.license: ""

  #ETCD
  alluxio.mount.table.source: "ETCD"
  # 如果要对接已有的etcd，需要修改此部分的访问地址/域名
  alluxio.etcd.endpoints: "http://alluxio-etcd:2379"
  alluxio.user.dynamic.consistent.hash.ring.enabled: "true"

  #Master&Job
  alluxio.master.scheduler.restore.job.from.journal: "false"
  alluxio.master.scheduler.initial.wait.time: "10s"
  alluxio.master.lost.worker.detection.interval: "30sec"
  alluxio.master.journal.type: "NOOP"
  alluxio.job.batch.size: "2000"
  alluxio.worker.network.grpc.reader.threads.max: "10000"
  alluxio.worker.network.ufs.reader.threads.max: "12000"
  alluxio.master.web.hostname: "alluxio-master-0"

  #Security
  alluxio.security.authentication.type: "NOSASL"
  alluxio.security.authorization.permission.enabled: "false"
```

```
# Connecting with Secure-HDFS
#alluxio.security.underfs.hdfs.kerberos.client.keytab.file: ""
#alluxio.security.underfs.hdfs.kerberos.client.principal: ""

#Worker
alluxio.worker.http.server.enabled: "false"
alluxio.worker.network.netty.backlog: "500"
alluxio.worker.network.netty.channel: "epoll"
alluxio.dora.ufs.list.status.cache.nr.files: "0"
alluxio.dora.ufs.list.status.cache.ttl: "0"
## Page Storage
alluxio.worker.page.store.page.size: "1MB"
## Metastore
# 如果底层文件会频繁发生修改，ttl需要设置的短一些
alluxio.dora.ufs.file.status.cache.ttl: "96h"
alluxio.dora.ufs.file.status.cache.size: "20000000"
alluxio.dora.worker.metastore.rocksdb.ttl: "96h"

#Client
alluxio.fuse.web.enabled: "true"
alluxio.dora.client.ufs.fallback.enabled: "true"
alluxio.underfs.io.threads: "50"
alluxio.user.netty.data.transmission.enabled: "true"
alluxio.user.network.netty.timeout: "120s"
alluxio.user.file.writetype.default: "THROUGH"

## meta about performance
alluxio.client.list.status.from.ufs.enabled: "true"
alluxio.user.update.file.accesstime.disabled: "true"
alluxio.fuse.update.check.enabled: "false"
alluxio.underfs.listing.length: "1000"
alluxio.underfs.xattr.change.enabled: "false"
alluxio.fuse.max.reader.concurrency: "1024"
alluxio.user.fuse.sync.close.enabled: "true"

alluxio.user.streaming.reader.chunk.size.bytes: "2MB"
```

```
  alluxio.user.streaming.writer.chunk.size.bytes: "2MB"
  alluxio.user.network.netty.reader.buffer.size.packets: "128"

  #client meta cache
  alluxio.user.file.metadata.sync.interval: "-1"
  alluxio.user.metadata.cache.expiration.time: "60s"
  alluxio.user.metadata.cache.max.size: "1000000"

  # Async prefetch related
  alluxio.user.position.reader.streaming.async.prefetch.enable: "true"
  alluxio.user.position.reader.streaming.async.prefetch.thread: "256"
  alluxio.user.position.reader.streaming.async.prefetch.part.length: "4MB"
  alluxio.user.position.reader.streaming.async.prefetch.max.part.number: "8"
  alluxio.user.position.reader.streaming.async.prefetch.file.length.threshold: "4MB"

  # multi replica
  alluxio.user.replica.selection.policy: "CLIENT_FIXED"
  alluxio.user.file.replication.min: "1"

  # File Segmentation
  alluxio.dora.file.segment.read.enabled: "false"
  alluxio.dora.file.segment.size: "1GB"

## Alluxio Master ##
master:
  resources:
    limits:
      cpu: "4"
      memory: "10Gi"
    requests:
      cpu: "0.4"
      memory: "1Gi"
  jvmOptions:
    - "-Xmx8g"
    - "-Xms2g"
 # Additional nodeSelector only for scheduling Alluxio masters
```

```
  nodeSelector:
    alluxio-node: "true"

## Alluxio Worker ##
 worker:
  # Number of workers to launch
  count: 16
  resources:
    limits:
      cpu: "10"
      memory: "40Gi"
    requests:
      cpu: "4"
      memory: "16Gi"
  jvmOptions:
    - "-Xmx22g"
    - "-Xms8g"
    - "-XX:MaxDirectMemorySize=10g"
    -  "-XX:+UseG1GC"
  # Additional nodeSelector only for scheduling Alluxio workers
  nodeSelector:
    alluxio-node: "true"

## worker data cache path ,可以多个path
 pagestore:
  # Type of the volume for Alluxio worker page store.
  type: hostPath
  quota: 1536Gi,1536Gi,1536Gi
  hostPath: /data1/alluxio/data,/data2/alluxio/data,/data3/alluxio/data

 metastore:
  # Type of the volume for Alluxio worker Metastore.
  type: hostPath
  hostPath: /data/alluxio/meta

## Alluxio FUSE##
```

```
 fuse:
   # fuse mount type, default csi ,or daemonSet
   type: csi
   # The path on the host machine for mount Alluxio Fuse (for daemonset type)
   hostPathForMount: /mnt/alluxio-system/fuse
   resources:
     requests:
       cpu: "10"
       memory: "32Gi"
     limits:
       cpu: "32"
       memory: "40Gi"
   mountOptions:
     - allow_other
     - entry_timeout=60
     - attr_timeout=60
     - max_idle_threads=256
     - max_background=256
   jvmOptions:
     - "-Xmx22g"
     - "-Xms8g"
     - "-XX:MaxDirectMemorySize=10g"
     -  "-XX:+UseG1GC"
   # Additional nodeSelector only for scheduling Alluxio daemonSet Fuse
   nodeSelector:
     alluxio-node: "true"

# 如果要对接已有的etcd，请将enable设置为false
 etcd:
   enabled: true
   replicaCount: 1
   resources:
     requests:
       cpu: "0.5"
       memory: "0.5Gi"
     limits:
```

```
      cpu: "2"
      memory: "4Gi"
  # 如果无法从公网下载etcd镜像，需要使用自有镜像
  image:
    registry: **
    repository: **/**
    tag: **
  volumePermissions:
    image:
      registry: **
      repository: **/**
      tag: **
  nodeSelector:
    alluxio-etcd: "true"

# 如果要对接已有的Prometheus&Grafana，请将enable设置为false
 alluxio-monitor:
  enabled: true
 # 如果无法从公网下载Prometheus+Grafana镜像，需要使用自有镜像
  prometheus:
    imageInfo:
      image: **/**
      imageTag: **
  grafana:
    imageInfo:
      image: **/**
      imageTag: **
```

alluxio-ufs.yaml

目前支持直接基于Operator方式进行ufs的挂载，也支持基于后续章节介绍的基于mount add命令实现的ufs挂载。

以挂载S3为UFS示例：

```
apiVersion: k8s-operator.alluxio.com/v1
kind: UnderFileSystem
metadata:
```

```
  name: alluxio-s3
spec:
  alluxioCluster: alluxio
  path: s3://**
  mountPath: /s3
  mountOptions:
    s3a.accessKeyId: "**"
    s3a.secretKey: "**"
    alluxio.underfs.s3.endpoint: "**"
    alluxio.underfs.s3.region: **
```

alliuxio-operator.yaml（./values.yaml）

请将image和imageTag修改为对应的镜像私有仓库地址：

```
nameOverride: alluxio-operator

image: xxxxxxxxxxxxxxxx
imageTag: *****
## 对应的是Alluxio k8s operator 的 docker image
imagePullPolicy: IfNotPresent

alluxio-csi:
  enabled: false
  image: ****
  imageTag: ***
  # 如果kubelet的目录不是/var/lib/kubelet，需要修改此处，可以ps aux|grep kubelet确认 root dir
  kubeletPath: /pathtokubelet
  # 如果希望控制alluxio fuse csi的可用范围
  nodePlugin:
    nodeSelector:
      label-a: value-a
## 对应的是Alluxio CSI的docker image
```

alluxio-namespace.yaml（参考）

```
apiVersion: v1
```

```
kind: Namespace
metadata:
  name: alluxio-test
  labels:
    name: alluxio-test
```

grafana template

Alluxio会提供监控模板：Alluxio-Monitor.json。

3、 Alluxio部署操作

1) 配置修改

a) 镜像信息

- 提前将Alluxio image、Alluxio operator image、Alluxio csi image上传到本地image仓库，并修改 alluxio-cluster.yaml、alliuxio-operator.yaml中的image、imageTag

相关命令：

```
# load alluxio image
docker load < alluxio-enterprise-**-docker.tar

# load operator image
docker load < alluxio-operator-**-helmchart.tgz

# load csi image
docker load < alluxio-csi-**-docker.tar
# tag image
docker tag alluxio/alluxio-enterprise:**
<your.registry>/<your.repository>/alluxio-enterprise:**
docker tag alluxio/k8s-operator:** <your.registry>/<your.repository>/k8s-operator:**
docker tag alluxio/csi:** <your.registry>/<your.repository>/csi:**

# push alluxio image to repo
docker push <your.registry>/<your.repository>/alluxio-enterprise:**
```

```
# push operator image to repo
docker push <your.registry>/<your.repository>/k8s-operator:**
# push csi image to repo
docker push <your.registry>/<your.repository>/k8s-operator:**
```

b) 进程配置信息

- 根据需要调整都部署哪些服务，调整enable为true/false
    - etcd、Prometheus、Grafana建议使用客户环境已有的服务
- 根据需要修改 master、worker 、fuse 的 cpu、内存及节点数量等。
- 修改 nodeSelector 指定 pod 的启动位置
    - master、etcd对于cpu、内存、存储要求不高，并且一般启动一个节点即可，可以设定一个独立的标签
    - Worker对于cpu、内存、缓存存储要求高，可以基于SSD等缓存资源所在设立标签
    - FUSE需要跟业务pod在一起，可以共享标签

c) alluxio配置信息

- metastore的hostPath，会转化为Alluxio Master/Worker 的hostpath mount，存储 Alluxio 的 meta 等信息，建议此目录对应的磁盘是 SSD 盘；
- pagestore的hostPath，会转化为Alluxio Worker 的hostpath mount，存储 alluxio 缓存数据，建议使用高性能 SSD 盘；
- 其他Hostpath配置：
    - 如果是NAS 的挂载，需要先在 hostPaths 的 Worker 中加入对应的 mount 路径。
    - FUSE 的mount path，在宿主机的什么目录中查看被挂载的 ufs 存储文件列表。
- configMaps
    - 如果需要依赖外部的配置文件，需要通过configMaps将这些配置文件上传，并且配置挂载到对应的pod内，比如hdfs mount需要依赖的hdfs-site、core-site，以及可能需要的Kerberos Keytab文件等
- 一些Meta缓存的TTL

    - meta缓存可以有效的提升性能，不过为了权衡数据一致性，需要根据需要设定对应的ttl。如果底层文件会发生变化，那么ttl不能设置的过大，如果文件只会新增不会发生修改、删除，那么ttl可以设置的较大一些。
- 缓存的pagesize
    - 如果是小文件的顺序整读的场景，建议将pagesize设置的比小文件较大一些，这样可以确保每次读一个完整小文件对应的是读一个完成page。
    - 如果是大文件的顺序读或者随机读的场景，建议将pagesize设置的跟每次读的blocksize一样，比如每次读取1MB/2MB/4MB，当然如果每次读取的blocksize很小，那么建议将pagesize设置成1MB。

2) Alluxio 部署操作

部署Alluxio operator

```
# the last parameter is the directory to the helm chart
$ helm install operator -f alluxio-operator/alluxio-operator.yaml alluxio-operator
NAME: operator
LAST DEPLOYED: Wed May 15 17:32:34 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

# verify if the operator is running as expected
$ kubectl get pod -n alluxio-operator
NAME                                          READY   STATUS    RESTARTS AGE
alluxio-controller-6b449d8b68-njx7f           1/1     Running   0        45s
operator-alluxio-csi-controller-765f9fd65-drjm4   2/2     Running   0        45s
operator-alluxio-csi-nodeplugin-ks262         2/2     Running   0        45s
operator-alluxio-csi-nodeplugin-vk8r4         2/2     Running   0        45s
ufs-controller-65f7c84cbd-kll8q               1/1     Running   0        45s
```

部署Alluxio

```
$ kubectl create -f alluxio-operator/alluxio-cluster.yaml -n <namespace>
alluxiocluster.k8s-operator.alluxio.com/alluxio created
# the cluster will be starting$ kubectl get pod
```

```
NAME                                  READY   STATUS              RESTARTS   AGE
alluxio-etcd-0                        0/1     ContainerCreating   0          7s
alluxio-etcd-1                        0/1     ContainerCreating   0          7s
alluxio-etcd-2                        0/1     ContainerCreating   0          7s
alluxio-master-0                      0/1     Init:0/1            0          7s
alluxio-monitor-grafana-847fd46f4b-84wgg      0/1     Running           0          7s
alluxio-monitor-prometheus-778547fd75-rh6r6   1/1     Running           0          7s
alluxio-worker-76c846bfb6-2jkmr               0/1     Init:0/2          0          7s
alluxio-worker-76c846bfb6-nqldm               0/1     Init:0/2          0          7s

# check the status of the cluster
$ kubectl get alluxiocluster
NAME      CLUSTERPHASE   AGE
alluxio   Ready          2m18s

# and check the running pods after the cluster is ready
$ kubectl get pod -n <namespace>
NAME                                  READY   STATUS    RESTARTS   AGE
alluxio-etcd-0                        1/1     Running   0          2m3s
alluxio-etcd-1                        1/1     Running   0          2m3s
alluxio-etcd-2                        1/1     Running   0          2m3s
alluxio-master-0                      1/1     Running   0          2m3s
alluxio-monitor-grafana-7b9477d66-mmcc5       1/1     Running   0          2m3s
alluxio-monitor-prometheus-78dbb89994-xxr4c   1/1     Running   0          2m3s
alluxio-worker-85fd45db46-c7n9p               1/1     Running   0          2m3s
alluxio-worker-85fd45db46-sqv2c               1/1     Running   0          2m3s
```

挂载底层存储到**Alluxio**

```
$ kubectl create -f alluxio-operator/ufs.yaml -n <namespace>
underfilesystem.k8s-operator.alluxio.com/alluxio-oss created

# verify the status of the storage
$ kubectl get ufs
NAME        PHASE   AGE
```

```
alluxio-oss   Ready   46s


# also check the mount table via Alluxio command line
$ kubectl exec -it alluxio-master-0  -n <namespace> -- alluxio mount list 2>/dev/null
Listing all mount points
oss://my-bucket/path/to/mount  on  /oss/ properties={fs.oss.accessKeyId=xxx, fs.oss.accessKeySecret=xxx, fs.oss.endpoint=xxx}
```

集群删除

```
#删除ufs
kubectl delete -f ufs.yaml -n <namespace>

##删除alluxio集群
kubectl delete -f alluxio-cluster.yaml -n <namespace>

# helm uninstall operator，默认是alluxio-operator namespace
helm uninstall operator -n alluxio-operator
```

集群重启

```
## 当通过 kubectl edit cm alluxio-alluxio-conf -n <namespace>修改了一些配置后，可以通过以下方式重启服务

# master:
kubectl rollout restart statefulset alluxio-master -n <namespace>

# worker
kubectl rollout restart deployment alluxio-worker -n <namespace>

# daemonSet fuse:
kubectl rollout restart daemonset alluxio-fuse -n <namespace>

# proxy:
kubectl rollout restart daemonset alluxio-proxy  <namespace>
```

观察pod状态，查看pod日志是否异常，部署完成

4、 Alluxio使用

1) 应用挂载Alluxio FUSE

DaemonSet FUSE

- alliuxio-operator.yaml中type设置为daemonSet
- 可以通过nodeSelector选择启动Alluxio FUSE的宿主机
    - 确保Alluxio FUSE在宿主机上的Mount Path的访问权限。
- 需要使用Alluxio FUSE的应用Pod需要启动在有Alluxio FUSE Pod的宿主机上，建议使用同一个nodeSelector
    - 应用Pod可以通过hostpath的方式，将Alluxio FUSE在宿主机上形成的挂载路径挂载到应用pod内
    - 也可以通过基于Alluxio FUSE在宿主机上形成的挂载路径形成PV（需要额外创建），然后应用Pod通过PVC挂载

CSI FUSE

- alliuxio-operator.yaml中type设置为csi
- 应用Pod通过PVC的方式挂载Alluxio FUSE
    - pvc名称 alluxio-alluxio-csi-fuse-pvc

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: fuse-test-0
  labels:
    app: alluxio
spec:
  containers:
    - image: busybox:stable
      imagePullPolicy: IfNotPresent
      name: fuse-test
      command: ["sleep", "infinity"]
      volumeMounts:
        - mountPath: /data
```

```
        name: alluxio-pvc
        mountPropagation: HostToContainer
  volumes:
    - name: alluxio-pvc
      persistentVolumeClaim:
        claimName: alluxio-alluxio-csi-fuse-pvc
```

2) UFS 挂载

HDFS作为**UFS**:

```
#提前将依赖的hdfs 连接需要的hdfs-site、core-site已经Kerberos keytab等放到configmap内
#如果连接的是开启了Kerberos的hdfs，alluxio-cluster.yaml中相应的Kerberos配置需要打开和调整
#mount 操作，进入某个worker pod执行
alluxio mount add --path /path1 --ufs-uri hdfs:///hdfs-path1/

alluxio mount add --path /path2 --ufs-uri hdfs:///hdfs-path2/

alluxio mount add --path /path3  --ufs-uri hdfs://hdfs-path3/

# 查看挂载
alluxio mount list
# umount 操作
alluxio mount remove --path /path1
alluxio mount remove --path /path2
alluxio mount remove --path /path3
```

S3作为**UFS**:

```
alluxio mount add --option s3a.accessKeyId=xxxxx --option s3a.secretKey=xxxxx  --option alluxio.underfs.s3.endpoint=xxxx --option alluxio.underfs.s3.inherit.acl=false  --path /<bucket-name> --ufs-uri s3://<bucket name>/<dir>
```

NFS作为**UFS**:

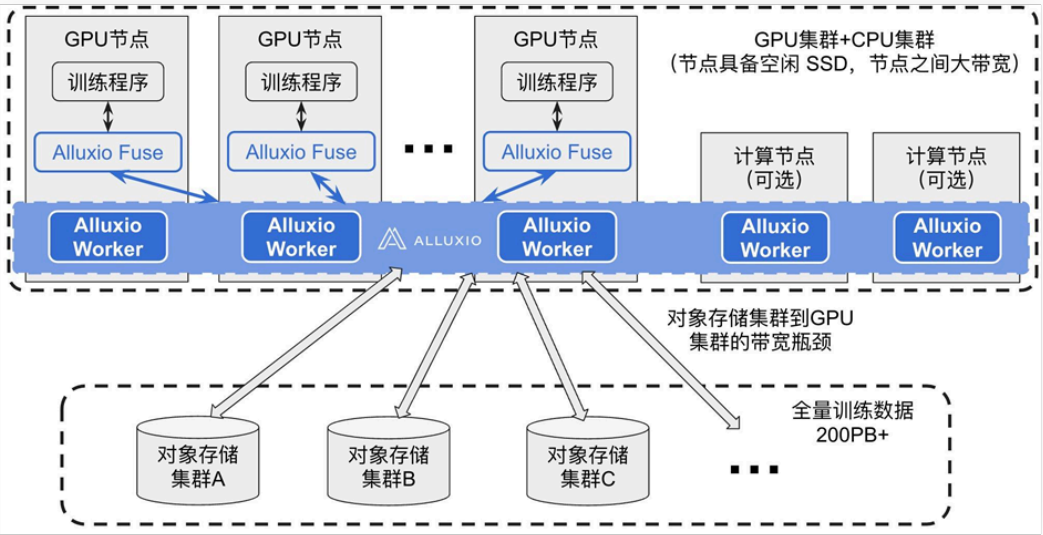
```
#提前将依赖的NAS/NFS通过hostPaths、pvcMounts挂载到FUSE、Master、Worker内。
alluxio mount add --path /{业务对应的子目录} --ufs-uri file:///nfs/volume-115-1/{业务对应的子目录}
```

3) 缓存预热

进入某一个Alluxio Master/Worker Pod：

```
#  job submit
alluxio job load --path hdfs:///***/***/ --submit
alluxio job load --path s3://***/***/ --submit
#job progress
alluxio job load --path hdfs:///***/***/ --progess
#job Stop
alluxio job load --path hdfs:///***/***/ --stop
```

附录1：Alluxio部署架构



1) Alluxio组件说明

Alluxio Master

- Alluxio Master 目前用于Alluxio Job Load时的任务调度。
- 运行状态是K8s statefulset。
- 部署在资源有空闲、网络畅通的任意节点。

Alluxio Worker

- Alluxio Worker承载着Alluxio的缓存存储、请求处理。
- Alluxio Worker建议使用高性能SSD盘存储数据缓存、Meta缓存。
- Alluxio Worker可以无状态扩展，通过一致性hash机制实现分布式。
- 一般一个宿主机启动一个Alluxio Worker Pod;建议可以部署在有缓存资源的节点上。如果GPU服务器有空闲NVMe空间，则建议部署在GPU服务器，否则也可以选择其他服务器，但是建议在一个大的网段，确保网络不是瓶颈。

Alluxio Fuse

- Alluxio FUSE提供了POSIX访问接口，一般部署在需要使用FUSE POSIX访问的应用/计算节点。
- Alluxio FUSE会在部署的宿主机上创建一个FUSE Mount入口，用于后续的计算/应用访问。
- 一般一个宿主机启动一个Alluxio FUSE Pod。

Alluxio Proxy

- Alluxio Proxy提供了S3、RESTful访问接口。

ETCD

- ETCD用于Alluxio Worker注册，也用于向Alluxio Client、Master广播Alluxio Worker List。
- 部署在资源有空闲、网络畅通的任意节点。

2) 影响Alluxio服务性能因素

- 缓存盘的性能
  决定了缓存数据读写的性能。

建议使用高性能的NVMe盘，可以使用多块盘（也可以做raid0），提升整体吞吐能力，比如PCI-E 4.0的NVMe单盘吞吐在7GB/s左右。

- Alluxio FUSE和Alluxio Worker之间的网络

  决定了数据传输的性能。

  最好是40Gb/100Gb带宽，如果是40Gb网络，则每个Alluxio Worker最大的输出能力是5GB/s（Alluxio FUSE输入能力同理）。

- Alluxio FUSE的CPU、内存

  决定了Alluxio FUSE客户端的处理能力。

  CPU 以intel为例，建议至少是3代以上CPU，比如6326，越高越好。

  内存最好可以保证CPU的每个通道都有内存条，可以保障CPU和内存之间的带宽。

  并发数越高，传输的数据越多，对于CPU、内存要求越高，包括CPU和内存之间的通道数，最好是满配。

  如果需要开启Metadata Cache等，也需要内存支撑。

- Alluxio Worker的CPU、内存

  决定了Alluxio Worker进程的处理能力。

  CPU 以intel为例，建议至少是3代以上CPU，比如6326，越高越好。

  内存最好可以保证CPU的每个通道都有内存条，可以保障CPU和内存之间的带宽。

  并发数越高，传输的数据越多，对于CPU、内存要求越高。

  缓存的数据越多，缓存的Metadata Cache也需要内存支撑。

- UFS的IOPS、BW，以及Alluxio Worker和UFS之间的网络

  决定了job load的性能以及冷读的性能。

  一般为了保证快速数据预热，需要高并发的从UFS进行数据的load，对于UFS自身的服务能力和网络负载都是挑战。

附录2：Alluxio缓存说明

Alluxio的几个缓存

1) Alluxio FUSE 侧的元数据缓存
    a) 缓存生成方式
        i) 通过ls等方式手动触发
        ii) 通过冷读留存
    b) 缓存失效原因
        i) TTL到了会失效
        ii) 元数据缓存max size到了，会发生驱逐
        iii) 底层文件发生变更，元数据缓存失效
    c) 缓存作用

i) 提升元数据访问性能，特别是目录list。
目录list的缓存对于当前FUSE上面的各训练任务都会生效。
如果训练文件list的生成不依赖Alluxio FUSE获取，则缓存
alluxio.user.metadata.cache.max.size以及
alluxio.user.metadata.cache.expiration.time可以设置的小一些。

ii) 减少重复元数据访问，一次文件访问会触发3-4次元数据的操作，利用缓存可以大大提升除第一次之外的元数据访问性能。

2) Alluxio FUSE 侧的数据缓存

默认不开，因为计算调度很难满足每次都在同一个客户端读取同样的数据。
开启后等于读本地磁盘，受限于本地磁盘性能。

a) 缓存作用

i) 如果可以保证计算调度访问相同数据，另外本地盘性能较好，网络条件较差的情况下，FUSE 客户端侧缓存可以大大提升数据访问性能

3) Alluxio Worker 侧的元数据缓存

a) 缓存生成方式

i) 通过job load预热

ii) 通过冷读异步留存

b) 缓存失效原因

i) alluxio.dora.ufs.file.status.cache.ttl到了会失效

ii) alluxio.dora.ufs.file.status.cache.size到了，会发生驱逐

iii) 底层文件发生变更，元数据缓存失效

c) 缓存作用

i) 尽量避免向UFS发起元数据请求，一般使用对象存储的时候，高并发时向UFS发起元数据请求性能较差

4) Alluxio Worker 侧的数据缓存

a) 缓存生成方式

i) 通过job load预热

ii) 通过冷读异步留存

b) 缓存失效原因

i) 缓存空间满了，被驱逐

ii) 底层文件发生变更，缓存失效

c) 缓存作用

i) 尽量避免从UFS读取数据

附录3：K8S常用命令

```
# load image
docker load -i < alluxio-enterprise-**-docker.tar
# tag image
docker tag alluxio/alluxio-enterprise:**
<your.registry>/<your.repository>/alluxio-enterprise:**
# push alluxio image to repo
docker push <your.registry>/<your.repository>/alluxio-enterprise:**

# create/show namespace
kubectl create -f ./alluxio-namespace.yaml
kubectl get namespaces | grep alluxio


#helm install/uninstall operator

helm install operator ./alluxio-operator  -f ./alluxio-operator/alliuxio-operator.yaml

helm uninstall operator -n alluxio-operator



#启动、删除、查看集群（pod）

kubectl create -f alluxio-cluster.yaml -n <namespace>

kubectl delete -f alluxio-cluster.yaml -n <namespace>

kubectl get alluxiocluster -n <namespace>



# 查看pod

kubectl get pods -n  <namespace>

kubectl get pods -o wide -n  <namespace>

kubectl describe pod <pod-name> -n <namespace>

# 进入pod(linux)命令行
```

```
 kubectl exec -i -t $POD_NAME -- /bin/bash


# pv/pvc

kubectl get pv  -n  <namespace>

kubectl get pvc  -n  <namespace>

kubectl describe pv <pv-name> -n  <namespace>

kubectl describe pvc <pvc-name> -n <namespace>

kubectl delete pv  -n  <namespace>

kubectl delete pvc  -n  <namespace>


# 查看、编辑configmap

kubectl get configmaps -n <namespace>

kubectl describe configmap <configmap-name> -n <namespace>

kubectl edit configmap <configmap-name> -n <namespace>


# 编辑/查看label

kubectl label nodes <nodename> <key>=<value>

kubectl get nodes --selector=<key>=<value>

kubectl label nodes <nodename> <key>-


# 查看pod日志，-p查看更靠前/完整的日志

kubectl logs <pod-name> -n <namespace>

kubectl logs <pod-name> -p -n <namespace>
```

```
#查看pod内container的日志

kubectl logs my-pod -c my-container -f


# 查看集群事件

kubectl get events


#查看deployment的log

kubectl logs deploy/alluxio-cluster-controller -n alluxio-operator
```

附录4：基准性能测试指引

单体文件测试

- 适用场景
  - 测试单个大文件的顺序读、随机读
- 示例代码
  - filename：Alluxio FUSE目录下的测试文件
  - bs：每次读取的blocksize
  - rw：读写模式，一般是顺序读read、随机读randread
  - ioengine：一般是psync、libaio
  - numjobs：并发数，可以测试1并发、3并发、10并发、30并发等，测试一下不同并发现的性能表现
  - direct：是否开启Kernel Cache（1 关闭，0 开启）。

```
fio --filename=/alluxio/fuse/alluxiofuse.test --bs=256k --size=10G --rw=read
--ioengine=psync --iodepth=30 --numjobs=1 --time_based --group_reporting
--runtime=30 --name=fio-test --direct=1
```

目录多文件测试

- 适用场景
  - 测试多个文件的顺序读、随机读
- 示例代码
  - directory：Alluxio FUSE下的测试目录
  - bs：每次读取的blocksize
  - rw：读写模式，一般是顺序读read、随机读randread
  - ioengine：一般是psync、libaio
  - numjobs：并发数，可以测试1并发、3并发、10并发、30并发等，测试一下不同并发现的性能表现
  - nrfiles：每个jobs读取的文件数，如果numjobs=128，nrfiles=1000，则该fio命令会读取128*1000个文件
  - filesize：设定文件大小
  - openfiles：每个job同时打开的文件数
  - direct：是否开启Kernel Cache（1 关闭，0 开启）。
- 备注：fio directory读取的文件是有规律的，所以需要先用fio write生成对应的文件（rw=wirte，同时去掉 readonly），建议直接在本地路径生成对应的文件，然后上传到ufs。

```
fio -engine=libaio -bs=256k --rw=read --group_reporting
--directory=/alluxio/fuse/local/multiple_files_100k --name=read_test_256k
--direct=1  --nrfiles=1000 --openfiles=16 --filesize=256K --readonly --numjobs=128
```