使用nfs-storageClass

- storageClass （存储类）

---

k8s持久化存储有多种方式，当我们没有条件使用ceph时候，NFS存储则是理想的选择；通常的的作法有两种，但是两种方法都明显的弊端：

- 在每一台node上挂载主机nfs，然后在容器上映射到主机目录

  > 每一台主机的挂载路径必须完全一致；由于对用到主机磁盘路径完全依靠手工填写，需要每个容器、至少是namespace级别统一规划好路径，否则很容易发生文件夹冲突。

- 使用PV/PVC；先创建PV，映射到NFS,在创建PVC，将PVC绑定到PV

  > 项目或者POD数量较多时候，先创建PV、再创建PVC、然后附加数据卷步骤繁多，操作麻烦。特别是在rancher界面操作时，页面切换转来转去。

storageClass则是为了解决这个问题而诞生，他能够根据一组配置，自动生成PV/PVC

- 安装

- **00-nfs-storage-rabc.yaml**

```
apiVersion: v1
kind: Namespace
metadata:
  # replace with namespace where provisioner is deployed
  # 替换为你的namespace
  name: nfs-storage
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nfs-client-provisioner
  # replace with namespace where provisioner is deployed
```

```
  # □□□□namespace
  namespace: nfs-storage
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "update", "patch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    # replace with namespace where provisioner is deployed
    # □□□□namespace
    namespace: nfs-storage
roleRef:
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
---
kind: Role
```

```
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  # 替换为你的namespace
  namespace: nfs-storage
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  # 替换为你的namespace
  namespace: nfs-storage
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    # replace with namespace where provisioner is deployed
    # 替换为你的namespace
    namespace: nfs-storage
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io
```

- **01-nfs-storage-provisioner.yaml**

这其实是一个容器，他会进行一些列的API监视，然后根据给定的参数创建PV/PVC并进行绑定。

需要设置的参数：

```
# env:
#  - PROVISIONER_NAME：名称，这个地方在后面创建storageClass时候会用到
#  - NFS_SERVER：nfs服务器地址
#  - NFS_PATH：nfs路径
```

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nfs-client-provisioner
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nfs-client-provisioner
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
    spec:
      serviceAccountName: nfs-client-provisioner
      containers:
        - name: nfs-client-provisioner
          # image: registry.k8s.io/sig-storage/nfs-subdir-external-provisioner:v4.0.2
          image: k8s.dockerproxy.com/sig-storage/nfs-subdir-external-provisioner:v4.0.2
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME
              value: k8s-sigs.io/nfs-subdir-external-provisioner
            - name: NFS_SERVER
              # value: <YOUR NFS SERVER HOSTNAME>
              value: nfs-share.vpclub.io
            - name: NFS_PATH
              # value: /var/nfs
              value: /data/share
      volumes:
        - name: nfs-client-root
          nfs:
            # server: <YOUR NFS SERVER HOSTNAME>
            server: nfs-share.vpclub.io
            path: /data/share
```

```
# 因为PV创建以后不可以修改，非常麻烦，建议nfs主机使用域名方式
# 需要在每台主机增加hosts
# echo "192.168.5.10       nfs-share.vpclub.io" >> /etct/hosts
```

- **02-nfs-storage-class.yaml**

provisioner参数需要和PROVISIONER_NAME参数对应

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-client-root-ns-pvname
provisioner: k8s-sigs.io/nfs-subdir-external-provisioner
parameters:
  # 此处也可以使用各种规则构造nfs中真实目录名称
  # "${.PVC.namespace}/${.PVC.name}"
  # "${.PVC.namespace}/${.PVC.annotations.nfs.io/storage-path}"
  pathPattern: "${.PVC.namespace}/${.PVC.name}"
  # 删除PVC时候的策略,Delete ：删除目录,Retain保留目录
  onDelete: retain
  # 回收策略 Retain — 手动回收,Recycle — 需要擦除后才能再次使用,Delete — 当用户删除对应的 PVC 时，动态配置的 volume 将被自动删除。默认为 Delete
reclaimPolicy: Retain

---

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-client-root-ns-customer
 # or choose another name, must match deployment's env PROVISIONER_NAME'
 # 替换为 provisioner pod 的环境变量 PROVISIONER_NAME
provisioner: "k8s-sigs.io/nfs-subdir-external-provisioner"
parameters:
  # 此处也可以使用各种规则构造nfs中真实目录名称
  # "${.PVC.namespace}/${.PVC.name}"
  # "${.PVC.namespace}/${.PVC.annotations.nfs.io/storage-path}"
```

```
  pathPattern: "${.PVC.namespace}/${.PVC.annotations.nfs.io/storage-path}"
  # 删除PVC时候的策略,Delete ：删除目录,Retain保留目录
  onDelete: retain
  # 删除PVC时候的策略，如果存在本项，且值为false，删除目录；如果存在 onDelete 设置，则以 onDelete 设置为准，既本配置项可以不需要
  archiveOnDelete: "false"
# 回收策略 Retain — 手动回收,Recycle — 需要擦除后才能再次使用,Delete — 当用户删除对应的 PVC 时，动态配置的 volume 将被自动删除。默认为 Delete
reclaimPolicy: Retain

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-client-root-ns
 # or choose another name, must match deployment's env PROVISIONER_NAME'
 # 替换为 provisioner pod 的环境变量 PROVISIONER_NAME
provisioner: k8s-sigs.io/nfs-subdir-external-provisioner
parameters:
  # 此处也可以使用各种规则构造nfs中真实目录名称
  # "${.PVC.namespace}/${.PVC.name}"
  # "${.PVC.namespace}/${.PVC.annotations.nfs.io/storage-path}"
  pathPattern: "${.PVC.namespace}"
  # 删除PVC时候的策略,Delete ：删除目录,Retain保留目录
  onDelete: retain
# 回收策略 Retain — 手动回收,Recycle — 需要擦除后才能再次使用,Delete — 当用户删除对应的 PVC 时，动态配置的 volume 将被自动删除。默认为 Delete
reclaimPolicy: Retain

---

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-client-root
 # or choose another name, must match deployment's env PROVISIONER_NAME'
 # 替换为 provisioner pod 的环境变量 PROVISIONER_NAME
provisioner: k8s-sigs.io/nfs-subdir-external-provisioner
parameters:
  # 此处也可以使用各种规则构造nfs中真实目录名称
  # "${.PVC.namespace}/${.PVC.name}"
  # "${.PVC.namespace}/${.PVC.annotations.nfs.io/storage-path}"
  pathPattern: "${.PVC.annotations.nfs.io/storage-path}"
```

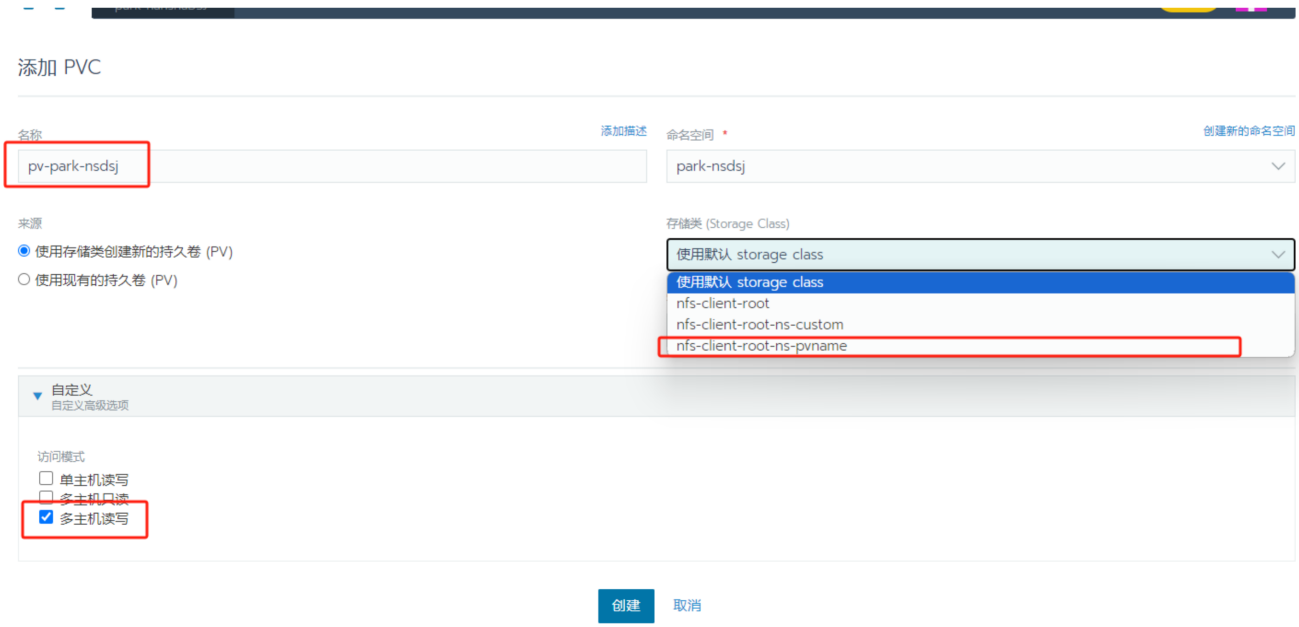
```
  # 删除PVC时候的策略,Delete ：删除目录,Retain保留目录
  onDelete: retain
# 回收策略 Retain — 手动回收,Recycle — 需要擦除后才能再次使用,Delete — 当用户删除对应的 PVC 时，动态配置的 volume 将被自动删除。默认为 Delete
reclaimPolicy: Retain
```

- 执行安装

```
kubectl apply -f 00-nfs-storage-rabc.yaml
kubectl apply -f 01-nfs-storage-provisioner.yaml
kubectl apply -f 02-nfs-storage-class.yaml
```

- 使用存储类

- 预创建**PVC**



- 快捷创建

  也可以在创建时指定storageClass，配置选项与预创建一致

- 存储驱动创建**pvc**规则

pathPattern: "${.PVC.namespace}/${.PVC.name}"

自动生成与PVC名称对应的路径

`/data/share/<namespace>/<pvname>`

对nfs-client-root-ns-pvname

pathPattern: "${.PVC.namespace}/${.PVC.annotations.nfs.io/storage-path}"

项目内(namespac`nfs.io/storage-path` ,由于rancher创建PVC时并不支持填写注释，且PVC创建后不可以给更改，需要使用导入YML方式创建

`/data/share/<namespace>/<storage-path>`

对nfs-client-root-ns-custom

pathPattern: "${.PVC.namespace}"

自动生产从当前项目(amespace)对应的路径

`/data/share/<namespace>`

对nfs-client-root-ns

pathPattern: "${.PVC.annotations.nfs.io/storage-path}"

全完使`nfs.io/storage-path`注釋于rancher创建PVC时并不支持填写注释，且PVC创建后不可以给更改，需要使用导入YML方式创建,也可以不指定stor

`/data/share/<storage-path>`，如果storage-path不填写，则直接从nfs更目录开始。

对nfs-client-root

- **03-nfs-storage-custome.yaml**

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-park-nsdsj
  # namespace: nfs-storage
```

```
  annotations:
    nfs.io/storage-path: "park-tianwei" # not required, depending on whether this annotation was shown in the storage class description
spec:
  storageClassName: nfs-client-root
  # storageClassName: nfs-client-root-ns-customer
  # storageClassName: nfs-client-root-ns-pvname
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi



# 非必要
# 因为前文的storage-class路径规则为 "${.PVC.namespace}/${.PVC.annotations.nfs.io/storage-path}"
# 由于Rancher界面使用快捷创建、预创建PVC，不支持增加注释，需要复制到导入YAML进行执行，如果你想指定到不同的路径下，可以执行这个
# kubectl apply -f 03-nfs-storage-customer.yaml
```

- 其他注意事项
  - pv创建后不支持修改nfs地址，对于nfs主机应该使用域名
  - 如果PVC有项目在使用，则PVC不可以不删，可以用一个centos镜像，将需要用到的PVC生成并挂载，避免误删除，同时可以可以进入到该容器查看磁盘结构和文件信息

版本号 #27
由 董列涛 创建于 12 八月 2023 16:39:06
由 董列涛 更新于 30 九月 2025 04:34:55