参考文献：https://blog.csdn.net/networken/article/details/86697018
本篇文章绝大部分（几乎完全）复制粘贴，他的文章给了我很大帮助，现在在这里写这篇文章，一是为了记录，CSDN有时候文章会损坏或者服务器正在维修，二是我觉得这是一个很好的文章，自己收藏；三是自己遇到了问题，记录一下

NFS简介
NFS是网络文件系统Network File System的缩写，NFS服务器可以让PC将网络中的NFS服务器共享的目录挂载到本地的文件系统中，而在本地的系统中来看，那个远程主机的目录就好像是自己的一个磁盘分区一样。

kubernetes使用NFS共享存储有两种方式：

1.手动方式静态创建所需要的PV和PVC。
2.通过创建PVC动态地创建对应PV，无需手动创建PV。
下面对这两种方式进行配置并进行演示。

搭建NFS服务器
k8s集群准备，以这篇文章为例：https://blog.csdn.net/networken/article/details/84991940

我的集群是一主一从，master的IP是192.168.88.111,node1的IP是192.168.88.114

这里作为测试，临时在master节点上部署NFS服务器。

```
#master节点安装nfs
yum -y install nfs-utils

#创建nfs目录
mkdir -p /nfs/data/

#修改权限
chmod -R 777 /nfs/data

#编辑export文件,这个文件就是nfs默认的配置文件
vim /etc/exports
/nfs/data *(rw,no_root_squash,sync)

#配置生效
exportfs -r
#查看生效
exportfs

#启动rpcbind、nfs服务
systemctl restart rpcbind && systemctl enable rpcbind
systemctl restart nfs && systemctl enable nfs

#查看 RPC 服务的注册状况
rpcinfo -p localhost
```

```
#showmount测试
showmount -e 192.168.88.111
```

所有node节点安装客户端，开机启动

```
yum -y install nfs-utils
systemctl start nfs && systemctl enable nfs
```

作为准备工作，我们已经在 k8s-master(192.168.88.111) 节点上搭建了一个 NFS 服务器，目录为 /nfs/data

静态申请PV卷
添加pv卷对应目录,这里创建2个pv卷，则添加2个pv卷的目录作为挂载点。

```
#创建pv卷对应的目录
mkdir -p /nfs/data/pv001

#配置exportrs(我觉得可以不用这步，因为父目录/nfs/data，已经设为共享文件夹)
vim /etc/exports
/nfs/data *(rw,no_root_squash,sync)
/nfs/data/pv001 *(rw,no_root_squash,sync)

#配置生效
exportfs -r
#重启rpcbind、nfs服务
systemctl restart rpcbind && systemctl restart nfs
```

创建PV

下面创建名为pv001的PV卷，配置文件 nfs-pv001.yaml 如下：

```
[centos@k8s-master ~]$ vim nfs-pv001.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
name: nfs-pv001
labels:
  pv: nfs-pv001
spec:
capacity:
  storage: 1Gi
accessModes:
  - ReadWriteOnce
persistentVolumeReclaimPolicy: Recycle
storageClassName: nfs
nfs:
  path: /nfs/data/pv001
  server: 192.168.88.111
```

配置说明：
① capacity 指定 PV 的容量为 1G。
② accessModes 指定访问模式为 ReadWriteOnce，支持的访问模式有：

2.1ReadWriteOnce – PV 能以 read-write 模式 mount 到单个节点。
2.2ReadOnlyMany – PV 能以 read-only 模式 mount 到多个节点。
2.3ReadWriteMany – PV 能以 read-write 模式 mount 到多个节点。
③ persistentVolumeReclaimPolicy 指定当 PV 的回收策略为 Recycle，支持的策略有：

3.1Retain – 需要管理员手工回收。
3.2Recycle – 清除 PV 中的数据，效果相当于执行 rm -rf /thevolume/*。
3.3Delete – 删除 Storage Provider 上的对应存储资源，例如 AWS EBS、GCE PD、Azure Disk、OpenStack Cinder Volume 等。
④ storageClassName 指定 PV 的 class 为 nfs。相当于为 PV 设置了一个分类，PVC 可以指定 class 申请相应 class 的 PV。
⑤ 指定 PV 在 NFS 服务器上对应的目录。

创建 pv：

```
[centos@k8s-master ~]$ kubectl apply -f nfs-pv001.yaml
persistentvolume/nfs-pv001 created

[centos@k8s-master ~]$ kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   A
nfs-pv001   1Gi        RWO            Recycle          Available           nfs
```

STATUS 为 Available，表示 pv就绪，可以被 PVC 申请。
创建PVC
接下来创建一个名为pvc001的PVC，配置文件 nfs-pvc001.yaml 如下：

```
[centos@k8s-master ~]$ vim nfs-pvc001.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc001
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: nfs
  selector:
    matchLabels:
      pv: nfs-pv001
```

执行yaml文件创建 pvc：

```
[centos@k8s-master ~]$ kubectl apply -f nfs-pvc001.yaml
persistentvolumeclaim/nfs-pvc001 created

[centos@k8s-master ~]$ kubectl get pvc
NAME      STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfs-pvc001   Bound    pv001    1Gi        RWO            nfs            6s

[centos@k8s-master ~]$ kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM            STORAGECLASS   REAS
nfs-pv001   1Gi        RWO            Recycle          Bound    default/pvc001   nfs
```

从 kubectl get pvc 和 kubectl get pv 的输出可以看到 pvc001绑定到pv001，申请成功。注意pvc绑定到对应pv通过labels标签方式实现，也可以不指定，将随机绑定到pv。
接下来就可以在 Pod 中使用存储了，Pod 配置文件 nfs-pod001.yaml 如下：

```
[centos@k8s-master ~]$ vim nfs-pod001.yaml
kind: Pod
apiVersion: v1
metadata:
  name: nfs-pod001
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: nfs-pv001
  volumes:
    - name: nfs-pv001
      persistentVolumeClaim:
        claimName: nfs-pvc001
```

与使用普通 Volume 的格式类似，在 volumes 中通过 persistentVolumeClaim 指定使用nfs-pvc001申请的 Volume。
执行yaml文件创建nfs-pdo001：

```
[centos@k8s-master ~]$ kubectl apply -f nfs-pod001.yaml
pod/nfs-pod001 created

[centos@k8s-master ~]$ kubectl get pod
NAME                                     READY   STATUS    RESTARTS   AGE
nfs-client-provisioner-75bf876d88-sqqpv   1/1     Running   0          25m
nfs-pod001                               1/1     Running   0          12s
```

验证 PV 是否可用:

```
[centos@k8s-master ~]$ kubectl exec nfs-pod001 touch /var/www/html/index001.html

[centos@k8s-master ~]$ ls /nfs/data/pv001/
index001.html
```

进入pod查看挂载情况

```
[centos@k8s-master ~]$ kubectl exec -it nfs-pod001 /bin/bash
root@nfs-pod001:/# df -h
......
192.168.92.56:/nfs/data/pv001   47G  5.2G   42G  11% /var/www/html
......
root@nfs-pod001:/#
```

删除pv
删除pod，pv和pvc不会被删除，nfs存储的数据不会被删除。

```
[centos@k8s-master ~]$ kubectl delete -f nfs-pod001.yaml
pod "nfs-pod001" deleted

[centos@k8s-master ~]$ kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM            STORAGECLASS   REAS
nfs-pv001   1Gi        RWO            Recycle          Bound    default/pvc001   nfs

[centos@k8s-master ~]$ kubectl get pvc
NAME     STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfs-pvc001   Bound    pv001    1Gi        RWO            nfs            25m

[centos@k8s-master ~]$ ls /nfs/data/pv001/
index001.html
```

继续删除pvc，pv将被释放，处于 Available 可用状态，并且nfs存储中的数据被删除。

```
[centos@k8s-master ~]$ kubectl delete -f nfs-pvc001.yaml
persistentvolumeclaim "nfs-pvc001" deleted

[centos@k8s-master ~]$ kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM            STORAGECLASS   R
nfs-pv001   1Gi        RWO            Recycle          Available                    nfs

[centos@k8s-master ~]$ ls /nfs/data/pv001/
[centos@k8s-master ~]$
```

继续删除pv

```
[centos@k8s-master ~]$ kubectl delete -f nfs-pv001.yaml
persistentvolume "pv001" deleted
```

动态申请PV卷

External NFS驱动的工作原理

K8S的外部NFS驱动，可以按照其工作方式（是作为NFS server还是NFS client）分为两类：

1.nfs-client:

也就是我们接下来演示的这一类，它通过K8S的内置的NFS驱动挂载远端的NFS服务器到本地目录；然后将自身作为storage provider，关联storage class。当用户创建对应的PVC来申请PV时，该provider就将PVC的要求与自身的属性比较，一旦满足就在本地挂载好的NFS目录中创建PV所属的子目录，为Pod提供动态的存储服务。

2.nfs:

与nfs-client不同，该驱动并不使用k8s的NFS驱动来挂载远端的NFS到本地再分配，而是直接将本地文件映射到容器内部，然后在容器内使用ganesha.nfsd来对外提供NFS服务；在每次创建PV的时候，直接在本地的NFS根目录中创建对应文件夹，并export出该子目录。

利用NFS动态提供Kubernetes后端存储卷

本文将介绍使用nfs-client-provisioner这个应用，利用NFS Server给Kubernetes作为持久存储的后端，并且动态提供PV。前提条件是有已经安装好的NFS服务器，并且NFS服务器与Kubernetes的Slave节点都能网络连通。将nfs-client驱动做一个deployment部署到K8S集群中，然后对外提供存储服务。

nfs-client-provisioner 是一个Kubernetes的简易NFS的外部provisioner，本身不提供NFS，需要现有的NFS服务器提供存储

部署nfs-client-provisioner

(在master上操作，即192.168.88.111)

首先克隆仓库获取yaml文件

```
git clone https://github.com/kubernetes-incubator/external-storage.git
cp -R external-storage/nfs-client/deploy/ $HOME
cd deploy
```

修改deployment.yaml文件

这里修改的参数包括NFS服务器所在的IP地址（192.168.88.111），以及NFS服务器共享的路径（/nfs/data），两处都需要修改为你实际的NFS服务器和共享目录。

```
[root@K8S-M1 deploy]# cat deployment.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nfs-client-provisioner
---
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: nfs-client-provisioner
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
    spec:
      serviceAccountName: nfs-client-provisioner
      containers:
        - name: nfs-client-provisioner
          image: quay.io/external_storage/nfs-client-provisioner:latest
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
```

```
          env:
            - name: PROVISIONER_NAME
              value: fuseim.pri/ifs
            - name: NFS_SERVER
              value: 192.168.88.111
            - name: NFS_PATH
              value: /nfs/data
      volumes:
        - name: nfs-client-root
          nfs:
            server: 192.168.88.111
            path: /nfs/data
```

这里我就有点疑惑了，这里既部署了deployment，也部署了ServiceAccount，在后面的rbac.yaml里也有同样的，所以我认为是应该是现在不需要这个ServiceAccount的创建。可能就是因为这个，所以我的pvc动态创建就出现了问题

部署deployment.yaml
（修改部分信息，参考上面）

```
kubectl apply -f deployment.yaml
```

查看创建的POD

```
[centos@k8s-master deploy]$ kubectl get pod -o wide
NAME                                     READY   STATUS              RESTARTS   AGE   IP
nfs-client-provisioner-75bf876d88-578lg   1/1     Running             0          51m   10.244.2.1
```

创建StorageClass

storage class的定义，需要注意的是：provisioner属性要等于驱动所传入的环境变量PROVISIONER_NAME的值。否则，驱动不知道知道如何绑定storage class。
此处可以不修改，或者修改provisioner的名字，需要与上面的deployment的PROVISIONER_NAME名字一致。
（此yaml无需修改）

```
[centos@k8s-master deploy]$ cat class.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: managed-nfs-storage
provisioner: fuseim.pri/ifs # or choose another name, must match deployment's env PROVISIONER_NA
parameters:
  archiveOnDelete: "false"
```

部署yaml文件

```
kubectl apply -f class.yaml
```

查看创建的storageclass

```
[centos@k8s-master deploy]$ kubectl get sc
NAME                  PROVISIONER      AGE
managed-nfs-storage   fuseim.pri/ifs   95m
[centos@k8s-master deploy]$
```

配置授权

如果集群启用了RBAC，则必须执行如下命令授权provisioner。(k8s1.6+默认开启)
此yaml无需修改

```
[centos@k8s-master deploy]$ cat rbac.yaml
kind: ServiceAccount
apiVersion: v1
metadata:
  name: nfs-client-provisioner
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - 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
    namespace: default
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
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
subjects:
```

```
    - kind: ServiceAccount
      name: nfs-client-provisioner
      # replace with namespace where provisioner is deployed
      namespace: default
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io
```

部署yaml文件

```
kubectl create -f rbac.yaml
```

测试

创建测试PVC

```
kubectl create -f test-claim.yaml
```

这里指定了其对应的storage-class的名字为managed-nfs-storage，如下：

```
[centos@k8s-master deploy]$ cat test-claim.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-claim
  annotations:
    volume.beta.kubernetes.io/storage-class: "managed-nfs-storage"
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Mi
```

查看创建的PVC
可以看到PVC状态为Bound，绑定的volume为pvc-a17d9fd5-237a-11e9-a2b5-000c291c25f3。

```
[centos@k8s-master deploy]$ kubectl get pvc
NAME         STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORA
test-claim   Bound    pvc-a17d9fd5-237a-11e9-a2b5-000c291c25f3   1Mi        RWX            manag
```

查看自动创建的PV

```
[centos@k8s-master deploy]$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   C
pvc-a17d9fd5-237a-11e9-a2b5-000c291c25f3   1Mi        RWX            Delete           Bound    d
[centos@k8s-master deploy]$
```

然后，我们进入到NFS的export目录，可以看到对应该volume name的目录已经创建出来了。
其中volume的名字是namespace，PVC name以及uuid的组合：

```
[root@k8s-master ~]# cd /nfs/data/
[root@k8s-master data]# ll
total 0
drwxrwxrwx 2 root root 21 Jan 29 12:03 default-test-claim-pvc-a17d9fd5-237a-11e9-a2b5-000c291c25
```

创建测试Pod
指定该pod使用我们刚刚创建的PVC：test-claim，另外注意这里将镜像改为dockerhub镜像。
完成之后，如果attach到pod中执行一些文件的读写操作，就可以确定pod的/mnt已经使用了NFS的存储服务了。

```
[centos@k8s-master deploy]$ vim test-pod.yaml
kind: Pod
apiVersion: v1
metadata:
  name: test-pod
spec:
  containers:
  - name: test-pod
    image: willdockerhub/busybox:1.24
    command:
      - "/bin/sh"
    args:
      - "-c"
      - "touch /mnt/SUCCESS && exit 0 || exit 1"
    volumeMounts:
      - name: nfs-pvc
        mountPath: "/mnt"
  restartPolicy: "Never"
  volumes:
    - name: nfs-pvc
      persistentVolumeClaim:
        claimName: test-claim
```

执行yaml文件

```
kubectl create -f test-pod.yaml
```

查看创建的测试POD

```
[centos@k8s-master ~]$ kubectl get pod -o wide
NAME                                     READY   STATUS             RESTARTS   AGE   IP
nfs-client-provisioner-75bf876d88-578lg  1/1     Running            0          51m   10.244.2.1
test-pod                                 0/1     Completed          0          41m   10.244.1.
```

在NFS服务器上的共享目录下的卷子目录中检查创建的NFS PV卷下是否有"SUCCESS" 文件。

```
[root@k8s-master ~]# cd /nfs/data/
[root@k8s-master data]# ll
total 0
drwxrwxrwx 2 root root 21 Jan 29 12:03 default-test-claim-pvc-a17d9fd5-237a-11e9-a2b5-000c291c25
[root@k8s-master data]#

[root@k8s-master data]# cd default-test-claim-pvc-a17d9fd5-237a-11e9-a2b5-000c291c25f3/
[root@k8s-master default-test-claim-pvc-a17d9fd5-237a-11e9-a2b5-000c291c25f3]# ll
total 0
-rw-r--r-- 1 root root 0 Jan 29 12:03 SUCCESS
```

清理测试环境
删除测试POD

```
kubectl delete -f test-pod.yaml
```

删除测试PVC

```
kubectl delete -f test-claim.yaml
```

在NFS服务器上的共享目录下查看NFS的PV卷已经被删除。

ok,成功了

说一下我当时遇到的问题，当时创建pv后，pv显示pending

```
[root@K8S-M1 deploy]# kubectl describe pvc test-claim
Name:          test-claim
Namespace:     default
StorageClass:  managed-nfs-storage
Status:        Bound
Volume:        pvc-962a06fa-67db-11e9-963d-000c29619b15
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-class: managed-nfs-storage
               volume.beta.kubernetes.io/storage-provisioner: fuseim.pri/ifs
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Mi
Events:
  Type    Reason                Age   From                         Message
  ----    ------                ----  ----                         -------
  Normal  ExternalProvisioning  10s   persistentvolume-controller  waiting for a volume to be cr
```

即正在等待外部设置程序“fuseim.pri/ifs”或系统管理员手动创建卷
我参考了这个文章
https://github.com/kubernetes-incubator/external-storage/issues/754
试着执行此yaml

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: default-admin-rbac (or whatever)
subjects:
  - kind: ServiceAccount
    name: default
    namespace: default
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

试过，依旧不可以
我发现我的nfs-client-provisioner-xxx的pod会出现running后，过一阵子后出现CrashLoopBackOff，即说明容器曾经启动了，但又异常退出了。
通过kubectl logs -f 命令查看，发现

```
nfs-client-provisioner: dial tcp 192.168.88.114:10250: connect: connection refused
```

或者(我不记得是哪个了)

```
[root@K8S-M1 deploy]# kubectl get pod
NAME                                     READY   STATUS             RESTARTS   AGE
nfs-client-provisioner-648f9b65c6-snsc6   0/1     CrashLoopBackOff   6          14m
[root@K8S-M1 deploy]# kubectl logs -f nfs-client-provisioner-648f9b65c6-snsc6
F0427 02:58:21.988130       1 provisioner.go:180] Error getting server version: Get https://10.1
```

192.168.88.114是我的node，也是此pod所在的node。 所以我觉得是我的provisioner的原因，但是你删除pod，还是不可以，pod的状态和pvc的状态依旧。
或者第二种情况，不过这种我也找不出原因

当我把provisioner的deployment删除后，(我不记得是否重启了虚拟机)，重新 kubectl create -f ,创建deployment和pod，这次观察pod ，一直running，查看日志

```
[root@K8S-M1 deploy]# kubectl logs -f nfs-client-provisioner-c85fbcc5d-mtgb8
I0426 05:03:50.995699       1 leaderelection.go:185] attempting to acquire leader lease  default
I0426 05:03:51.083275       1 leaderelection.go:194] successfully acquired lease default/fuseim.
I0426 05:03:51.084006       1 controller.go:631] Starting provisioner controller fuseim.pri/ifs_
I0426 05:03:51.084350       1 event.go:221] Event(v1.ObjectReference{Kind:"Endpoints", Namespace
I0426 05:03:51.205879       1 controller.go:680] Started provisioner controller fuseim.pri/ifs_n
I0426 05:03:51.206648       1 controller.go:987] provision "default/test-claim" class "managed-n
```

这时再查看pvc，发现成功（或者你重新创建pvc）

```
[root@K8S-M1 deploy]# kubectl get pvc
NAME                    STATUS    VOLUME                                     CAPACITY   ACCESS M
test-claim              Bound     pvc-962a06fa-67db-11e9-963d-000c29619b15   1Mi        RWX

[root@K8S-M1 deploy]# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
pvc-962a06fa-67db-11e9-963d-000c29619b15   1Mi        RWX            Delete           Bound
```

发现成功创建了pv,并且绑定
再次查看provisioner的pod的日志，发现在上次的记录之外，多了下面的记录

```
I0426 05:03:51.213869       1 event.go:221] Event(v1.ObjectReference{Kind:"PersistentVolumeClaim
I0426 05:03:51.224187       1 controller.go:1087] provision "default/test-claim" class "managed-
I0426 05:03:51.224252       1 controller.go:1101] provision "default/test-claim" class "managed-
I0426 05:03:51.299834       1 controller.go:1108] provision "default/test-claim" class "managed-
I0426 05:03:51.299895       1 controller.go:1149] provision "default/test-claim" class "managed-
I0426 05:03:51.300609       1 event.go:221] Event(v1.ObjectReference{Kind:"PersistentVolumeClaim
```

说明provisioner成功工作，nfs动态存储也成功了。我想了一下，可能是再创建provisioner的deployment时，也创建了ServiceAccount，但是这个ServiceAccount还没有绑定具有权限的ClusterRole,虽然后面的rbac.yaml里写入了，但是deployment已经把没有绑定权限的ClusterRole的ServiceAccount写入到环境里，它已经不受后面操作的影响了，所以pvc才说正在等待外部设置程序“fuseim.pri/ifs”或系统管理员手动创建卷，pod无法作为provisioner来创建pvc，所以pvc一直pending。

1.所以deployment就只需要创建provisioner的deployment，可以把创建ServiceAccount的代码删除，由rbac.yaml去统一创建绑定。
2.也可能需要需要先创建rbac.yaml,再创建deployment和storageClass。

3.也可能就是偶然，出现这种情况就删除deployment，重新创建
4.因为我是在笔记本上用虚拟机创建k8s集群，所以有时候node会noready，出现了很多次，需要重启node，错误过程中（这几个小时里）也出现了，我不知道是不是一直NotReady。所以可能是因为node NotReady，所以pod才会重启为CrashLoopBackOff,所以日志会显示nfs-client-provisioner: dial tcp 192.168.88.114:10250: connect: connection refused，无法作为provisioner来动态创建pv，这也是一种可能
5.虚拟机挂起，关机后启动出现某种缓存类似的问题，
也可能是都不对，哈哈，只是举了我的几个想法，不完全。

我刚刚又重新创建pvc，发现

```
[root@K8S-M1 deploy]# kubectl describe pvc test-claim
Name:          test-claim
Namespace:     default
StorageClass:  managed-nfs-storage
Status:        Bound
Volume:        pvc-0d94a6a4-6800-11e9-963d-000c29619b15
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-class: managed-nfs-storage
               volume.beta.kubernetes.io/storage-provisioner: fuseim.pri/ifs
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      1Mi
Access Modes:  RWX
Events:
  Type       Reason                 Age                From
  ----       ------                 ----               ----
  Normal     ExternalProvisioning   76s (x2 over 76s)  persistentvolume-controller
  Normal     Provisioning           76s                fuseim.pri/ifs_nfs-client-provisioner-c85
  Normal     ProvisioningSucceeded  76s                fuseim.pri/ifs_nfs-client-provisioner-c85
Mounted By:  <none>
```

也是出现了waiting for a volume to be created, either by external provisioner "fuseim.pri/ifs" or manually created by system administrator，后面External provisioner给他创建pv来提供。一会时间。但是之前那个是一直创建不了，一直pending，那就是External provisioner的pod(deployment)有问题。这只是补充情况

虚拟机挂起后启动（从挂起到启动）出现了下面的情况，pod重新创建了

```
[root@K8S-M1 deploy]# kubectl get pod
NAME                                     READY   STATUS             RESTARTS   AGE
nfs-client-provisioner-648f9b65c6-snsc6   0/1     CrashLoopBackOff   6          14m
[root@K8S-M1 deploy]# kubectl logs -f nfs-client-provisioner-648f9b65c6-snsc6
F0427 02:58:21.988130       1 provisioner.go:180] Error getting server version: Get https://10.1
```

不过如果虚拟机一直开着，那么不会出现这种情况，应该是虚拟机挂起到启动，k8s的某种缓存或者其他原因导致会出现这种情况。如果你装nfs-provisioner，出现这种情况，那么就重启虚拟机，不行就重启电脑，再重新部署deployment（可能也不需要吧，就删除pod）

OK，感觉已经解决了
在我挂起(可能关机)虚拟机后，启动（从挂起到启动）虚拟机后，k8s的provisioner的pod就出现了上面的情况，即使重新部署deployment，也不行。我重启k8s(master和node)，好像也不行。我后面重启电脑后，打开虚拟机后，再次部署provisioner的deployment，就OK了。说明这是偶然的，重启虚拟机的k8s就可能解决，起因可能是虚拟机挂起后重新启动，k8s就出现某种问题，解决方案 重启虚拟机或者直接重启电脑。说明应该是原因5