Kubernetes中的存储

邢舟　IBM开放技术研究院

2017.11.26

“Kubenetes”系列公开课

- 每周四晚8点档

1. Kubernetes 初探
2. 上手 Kubernetes
3. Kubernetes 的资源调度
4. Kubernetes 的运行时
5. Kubernetes 的网络管理
6. **Kubernetes 的存储管理**
7. Kubernetes 的日志与监控
8. Kubernetes 的应用部署
9. 扩展 Kubernetes 生态
10. Kubernetes 的企业实践

课程Wiki：http://ibm.biz/opentech-ma

议程

- K8s存储概览
  - 应用场景
  - 设计与基本架构
  - 目前社区所实现和维护的存储插件一览
  - 目前存储所存在的问题
- K8s Volume Provisioner部分的代码实现
  - 扩展K8s存储的几种方式
  - Persistent Volume与Persistent Volume Claim
  - PV Controller的实现
  - Out-of-Tree Provisioner的实现
- 一个简单的K8s存储示例

K8s存储的主要应用场景

- 应用程序／服务存储状态、数据存取等
- 应用程序／服务配置文件读取、秘钥配置等
- 不同应用程序间或者应用程序内进程间共享数据

K8s存储的主要应用场景——一个例子（1/4）

- 需求
  - 部署一个Nginx服务，并在/var/nginx-data目录下存储用户上传的数据
- 解决：
  - 使用AWS的弹性存储卷并将其挂载至K8s容器指定目录

K8s存储的主要应用场景——一个例子（2/4）

- Step 1
  - 创建一个AWS存储卷
  - 获取创建好的存储卷ID

- aws ec2 create-volume –availability-zone xxx –size xx …

```
{
        "AvailabilityZone": ”xxx”,
        "Encrypted": false,
        "VolumeType": "gp2”,
        "VolumeId": "vol-xxxxxxxxxxxxxxxxxx”,
        "State": "creating”,
        "Iops": xxx,
        "SnapshotId": "”,
        "CreateTime": ”xxxxxxxxxxxx”,
        "Size": xx
}
```

K8s的主要应用场景——一个例子（3/4）

- Step 2
  - 创建一个包含nginx pod定义的yaml文件
  - 文件中包含存储卷在pod容器中的挂载位置
  - 以及K8s存储卷定义，其中使用到了Step1中所创建的存储卷的ID

```
apiVersion: v1
kind: Pod
metadata:
   name: nginx
spec:
   containers:
   - image: nginx
    name: nginx-server
    volumeMounts:
    - mountPath: /var/nginx-data
      name: data-volume
   volumes:
   - name: data-volume
     awsElasticBlockStore:
        volumeID: vol-xxxxxxxxxxxxxxxxx
        fsType: ext4
```

K8s存储的主要应用场景——一个例子（4/4）

- Step 3
  - 基于Step2中创建的yaml文件在K8s集群中创建nginx pod
  - 登录运行中的pod, 验证/var/nginx-data已经正确挂载

Credit: http://leebriggs.co.uk/blog/2017/03/12/kubernetes-flexvolumes.html

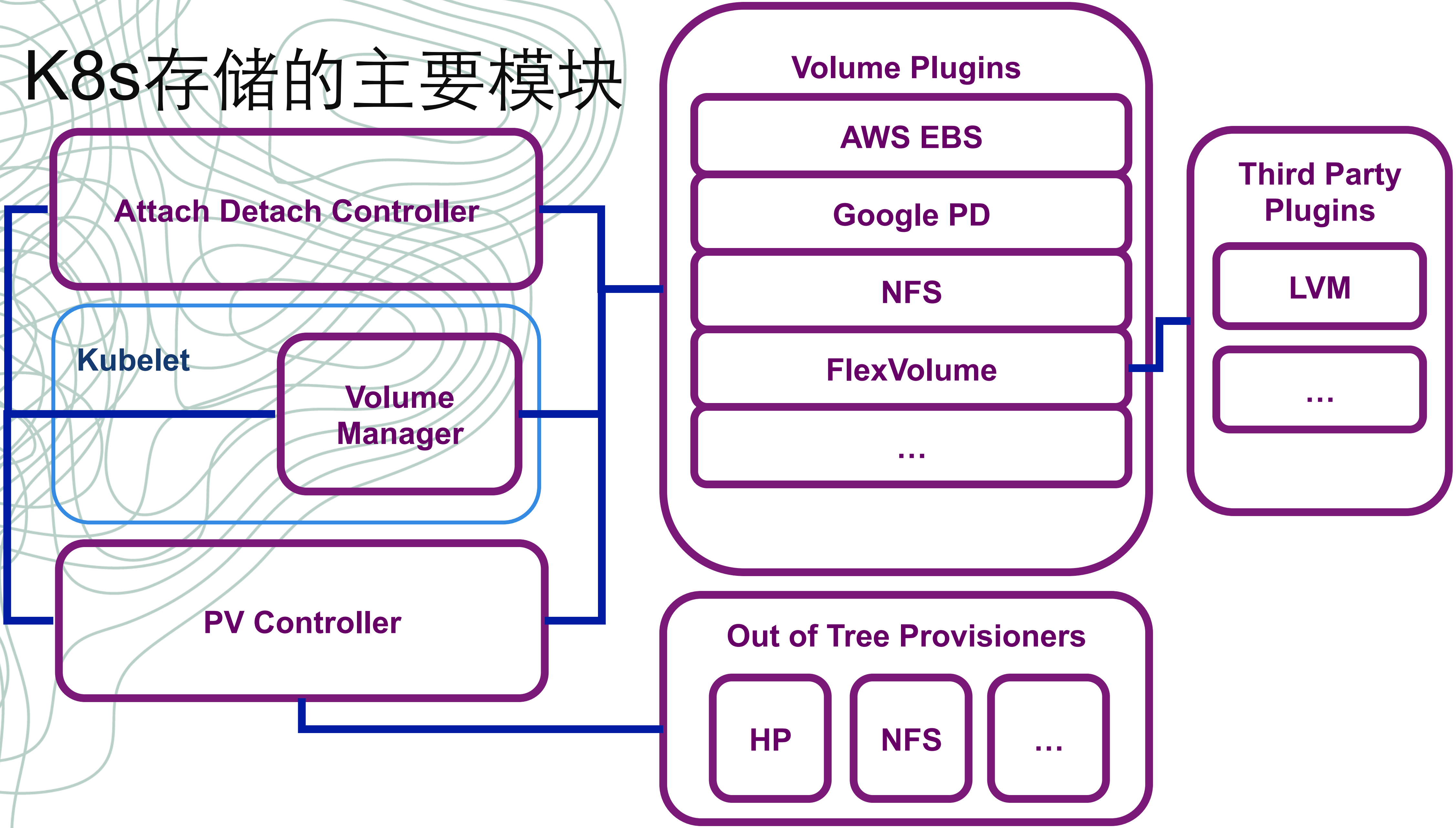
```
kubectl create –f nginx_pod.yaml

aws ec2 describe-volumes --volume-ids vol-
xxxxxxxxxxxxxx
{
    "Volumes": [
          {"Attachments": [
             {
               "VolumeId": "vol-xxxxxxxxxxxxxxxxx",
                "State": "attached”,
              "Device": "/dev/xvdba"
              }
           ]}
    ]
}
```

K8s存储的主要设计原则

- 遵循K8s整体架构
  - 声明式架构
- 易用性，尽可能多地兼容各种存储平台
  - 相比较于Docker Volume而言
  - 插件化
  - 兼容用户自定制插件
- 安全性
  - 数据安全性
  - 生命周期

K8s存储的主要模块


Attach Detach Controller
Kubelet
Volume Manager
PV Controller
Volume Plugins
AWS EBS
Google PD
NFS
FlexVolume
…
Third Party Plugins
LVM
…
Out of Tree Provisioners
HP
NFS
…

K8s存储的主要模块——Attach_Detach_Controller

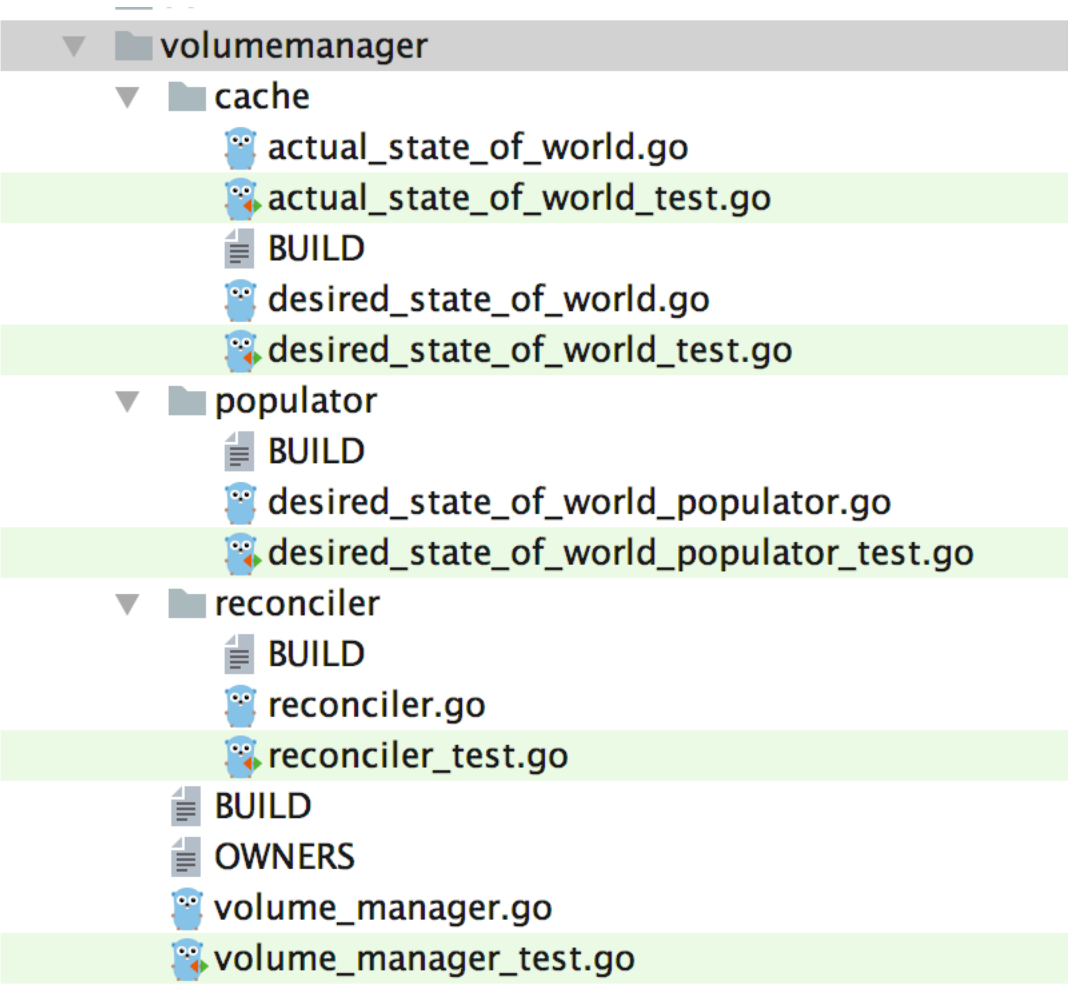
- 负责将远程网络块存储设备挂载到某一个K8s节点的Controller
    - 两个存储结构
        - Actual State of World
        - Desired State of World
    - 三个线程
        - PopulateActualStateofWorld
        - PopulateDesiredStateofWorld
        - Reconcile
    - 与Volume Plugin的交互
        - Attach/Detach
        - Attachable Plugins

K8s存储的主要模块——PV Controller

- PV Controller
  - 监控和管理集群中的PV和PVC
    - 实现PV/PVC生命周期管理
      - PVC: Pending, Bound, Lost
      - PV: Pending, Available, Bound, Released, Failed
    - 实现PV/PVC绑定
    - 实现Dynamic Provision
      - Direct Access
      - Dynamic Provision

K8s存储的主要模块——Volume Manager(1/2)

- 运行在每个Kubelet上的核心模块
  - 用于协调attach/detach controller, PV controller和各个Volume Plugin
  - 最终实现将块设备从创建到挂载到K8s上指定目录的过程



```
▼ volumemanager
  ▼ cache
      actual_state_of_world.go
      actual_state_of_world_test.go
      BUILD
      desired_state_of_world.go
      desired_state_of_world_test.go
  ▼ populator
      BUILD
      desired_state_of_world_populator.go
      desired_state_of_world_populator_test.go
  ▼ reconciler
      BUILD
      reconciler.go
      reconciler_test.go
    BUILD
    OWNERS
    volume_manager.go
    volume_manager_test.go
```

K8s存储的主要模块——Volume Manager(2/2)

- K8s挂载卷的基本过程
  - 用户创建Pod包含一个PVC
  - Pod被分配到节点NodeA
  - Kubelet等待Volume Manager准备设备
  - PV controller调用相应Volume Plugin(in-tree或者out-of-tree)创建持久化卷并在系统中创建PV对象以及其与PVC的绑定(Provision)
  - Attach/Detach controller或者Volume Manager通过Volume Plugin实现块设备挂载(Attach)
  - Volume Manager等待设备挂载完成，将卷挂载到节点指定目录(mount)
    - /var/lib/kubelet/plugins/kubernetes.io/aws-ebs/mounts/vol-xxxxxxxxxxxxxxxxx
  - Kubelet在被告知设备准备好后启动Pod中的容器，利用Docker –v等参数将已经挂载到本地的卷映射到容器中(volume mapping)

K8s存储的主要模块——Volume Plugin(1/2)

- Volume Plugin的主要接口：
  - Init
  - NewProvisioner(Provision)
  - NewDeleter(Delete)
  - NewAttacher(Attach)
  - NewDetacher(Detach)
  - NewMounter(Mount)
  - NewUnmounter(Unmount)
  - Recycle

Provisioner/Deleter Plugin

Attachable Plugin

Recyclable Plugin

K8s存储的主要模块——Volume Plugin(2/2)

- 持久化存储（网络）
  - Google Persistent Disk
  - AWS Elastic Block Store
  - Azure File Storage
  - Azure Data Disk
  - iSCSI
  - Flocker
  - NFS
  - vShpere
  - GlusterFS
  - Ceph File and RBD
  - Cinder
  - Quobyte Volume
  - FibreChannel
  - VMWare Photon PD
  - Portworx
  - Dell EMC ScaleIO

- 临时存储（本地）
  - Empty Dir(tmpfs)
  - K8s API
    - Secret
    - ConfigMap
    - Downward API
    - ProjectedVolume

- 其他
  - Flex Volume
  - Host Path
  - Local Persistent Storage

K8s存储目前存在的一些问题

- Resource Limitation & Separation
- Data Replication & Snapshot
- Volume Resize and Autoscaling
- Monitoring and QoS
- AccessMode
- PV Lost

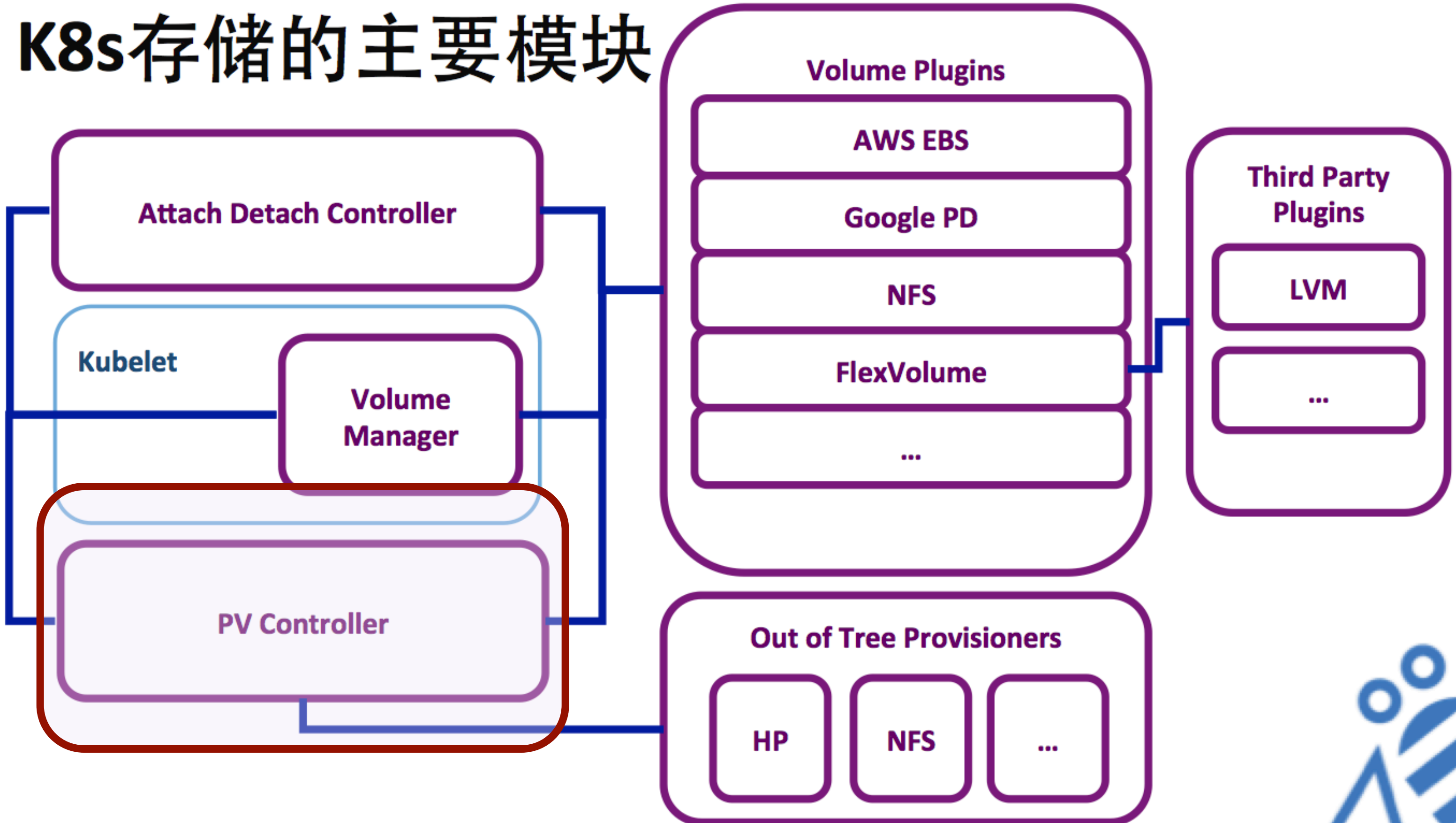
议程

- ❖ K8s存储概览
  - ❖ 应用场景
  - ❖ 设计与基本架构
  - ❖ 目前社区所实现和维护的存储插件一览
  - ❖ 目前存储所存在的问题
- ❖ K8s Volume Provisioner部分的代码实现
  - ❖ 扩展K8s存储的几种方式
  - ❖ Persistent Volume与Persistent Volume Claim
  - ❖ PV Controller的实现
  - ❖ Out-of-Tree Provisioner的实现
- ❖ 一个简单的K8s存储示例

扩展K8s存储的几种方式

- 扩展K8s存储的基本需求
- 扩展方式
  - 新的Volume Plugin
  - FlexVolume
  - Out-of-Tree Provisioner

K8s存储的主要模块



Persistent Volume与Persistent Volume Claim

- Persistent Volume/Persistent Volume Claim
  - 从Storage Admin与用户的角度看PV与PVC
    - Admin创建和维护PV
    - 用户只需要使用PVC(size & access mode)
  - PVC与PV的绑定
  - 用户级别的逻辑对象，将Volume实现与Pod解耦
  - PVC与Volume
  - PV与Volume

Storage Class

- StorageClass
  - StorageClass将说明Volume由哪种Volume Provisioner创建、创建时参数以及从其他功能性 / 非功能性角度描述的后台volume的各种参数
    - Static Provisioning
    - Dynamic Provisioning
      - In-tree provisioner
      - Out-of-tree provisioner

PV、PVC & StorageClass

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: slow
```

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  awsElasticBlockStore:
    volumeID: vol-xxxxxx
     fsType: ext4
```

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-eb
parameters:
  parameters:
  type: io1
  zone: us-east-1d
  iopsPerGB: "10"
```

利用PVC重写前面例子中的Volume定义

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx-server
    volumeMounts:
    - mountPath: /var/nginx-data
      name: data-volume
  volumes:
  - name: data-volume
    awsElasticBlockStore:
      volumeID: vol-xxxxxxxxxxxxxxxxx
      fsType: ext4
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx-server
    volumeMounts:
    - mountPath: /var/nginx-data
      name: data-volume
  volumes:
  - name: data-volume
    persistentVolumeClaim:
      claimName: myclaim
```

PV Controller——PersistentVolumeController

- 管理PV生命周期 & 与PVC绑定

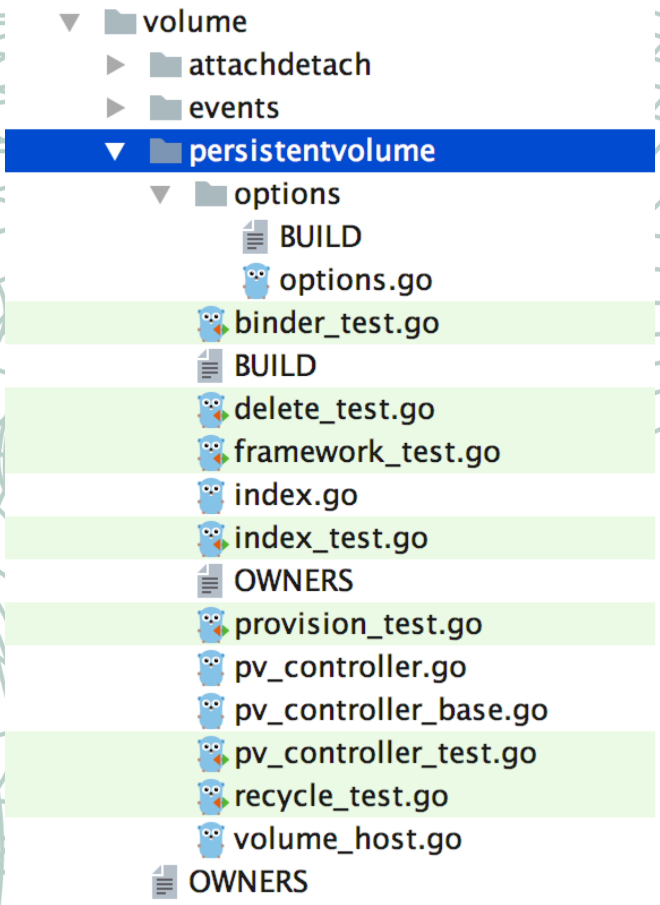
  - 主要数据结构
    - volumeQueue
      - workqueue.Type
    - volumes
      - persistentVolumeOrderIndex(cache.Store)
      - corelisters.PersistentVolumeLister
  - 运行时刻框架
    - volumeWorker
      - PV add/update/sync/delete
      - volumeQueue

- 管理PVC生命周期 & 与PV绑定

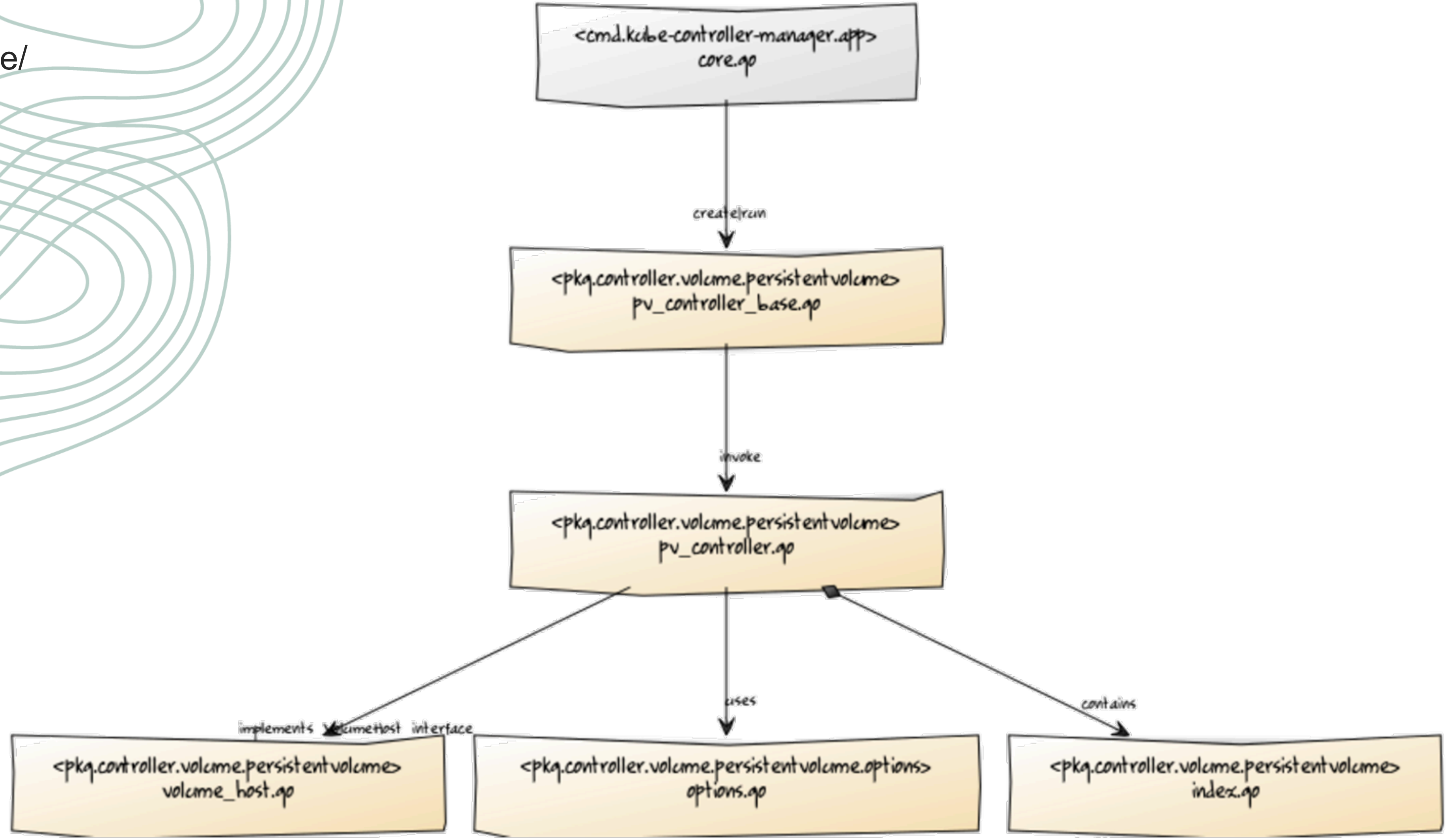
  - 主要数据结构
    - claimQueue
      - workqueue.Type
    - Claims
      - cache.Store
      - corelisters.PersistentVolumeClaimLister
  - 运行时刻框架
    - claimWorker
      - PVC add/update/sync/delete
      - claimQueue

PV Controller的基本实现

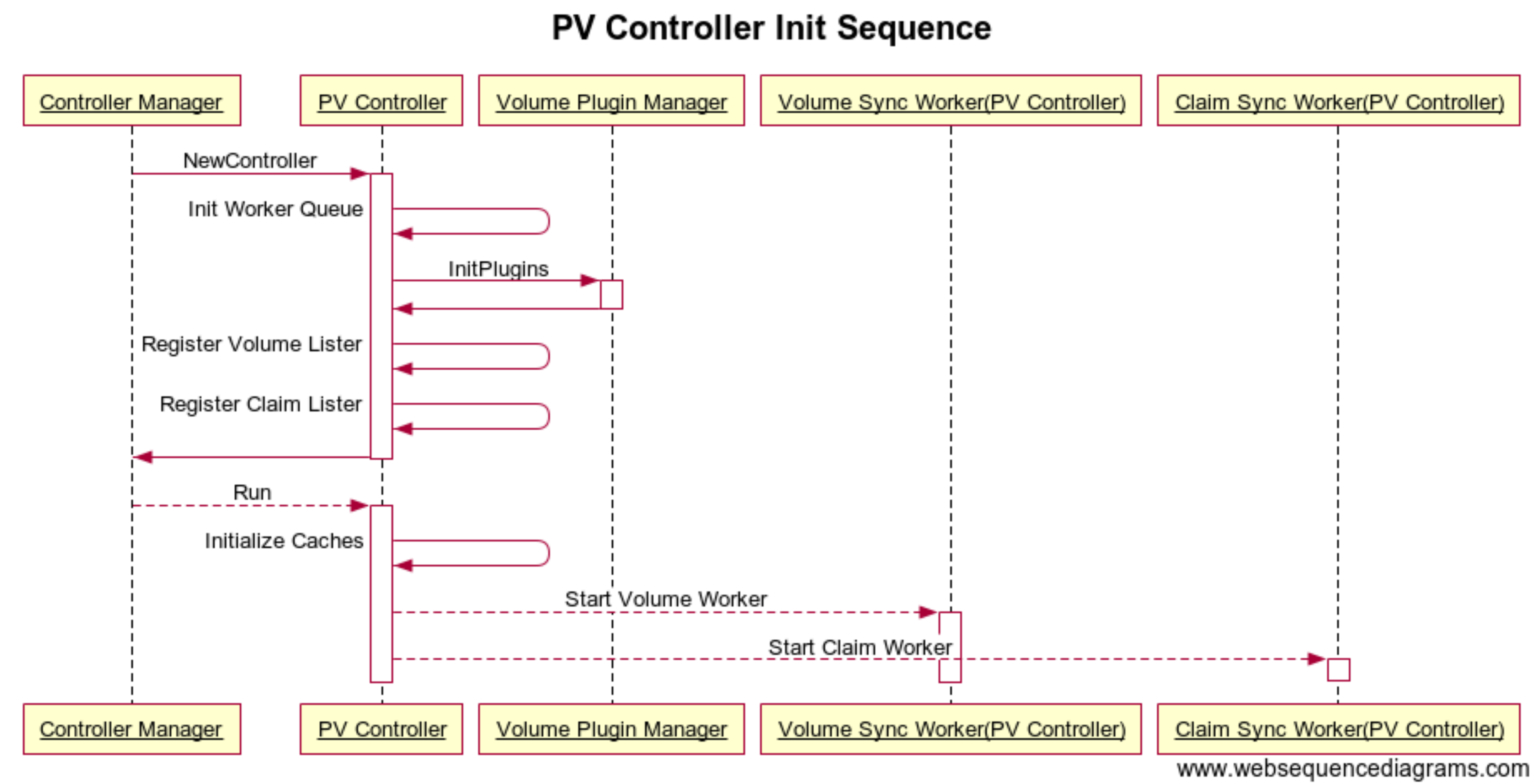
- kubernetes/pkg/controller/volume/persistentvolume



```
▼ volume
    ▶ attachdetach
    ▶ events
    ▼ persistentvolume
        ▼ options
            BUILD
            options.go
        binder_test.go
        BUILD
        delete_test.go
        framework_test.go
        index.go
        index_test.go
        OWNERS
        provision_test.go
        pv_controller.go
        pv_controller_base.go
        pv_controller_test.go
        recycle_test.go
        volume_host.go
    OWNERS
```


<cmd.kube-controller-manager.app>
core.go
create/run
<pkg.controller.volume.persistentvolume>
pv_controller_base.go
invoke
<pkg.controller.volume.persistentvolume>
pv_controller.go
implements VolumeHost interface
uses
contains
<pkg.controller.volume.persistentvolume>
volume_host.go
<pkg.controller.volume.persistentvolume.options>
options.go
<pkg.controller.volume.persistentvolume>
index.go

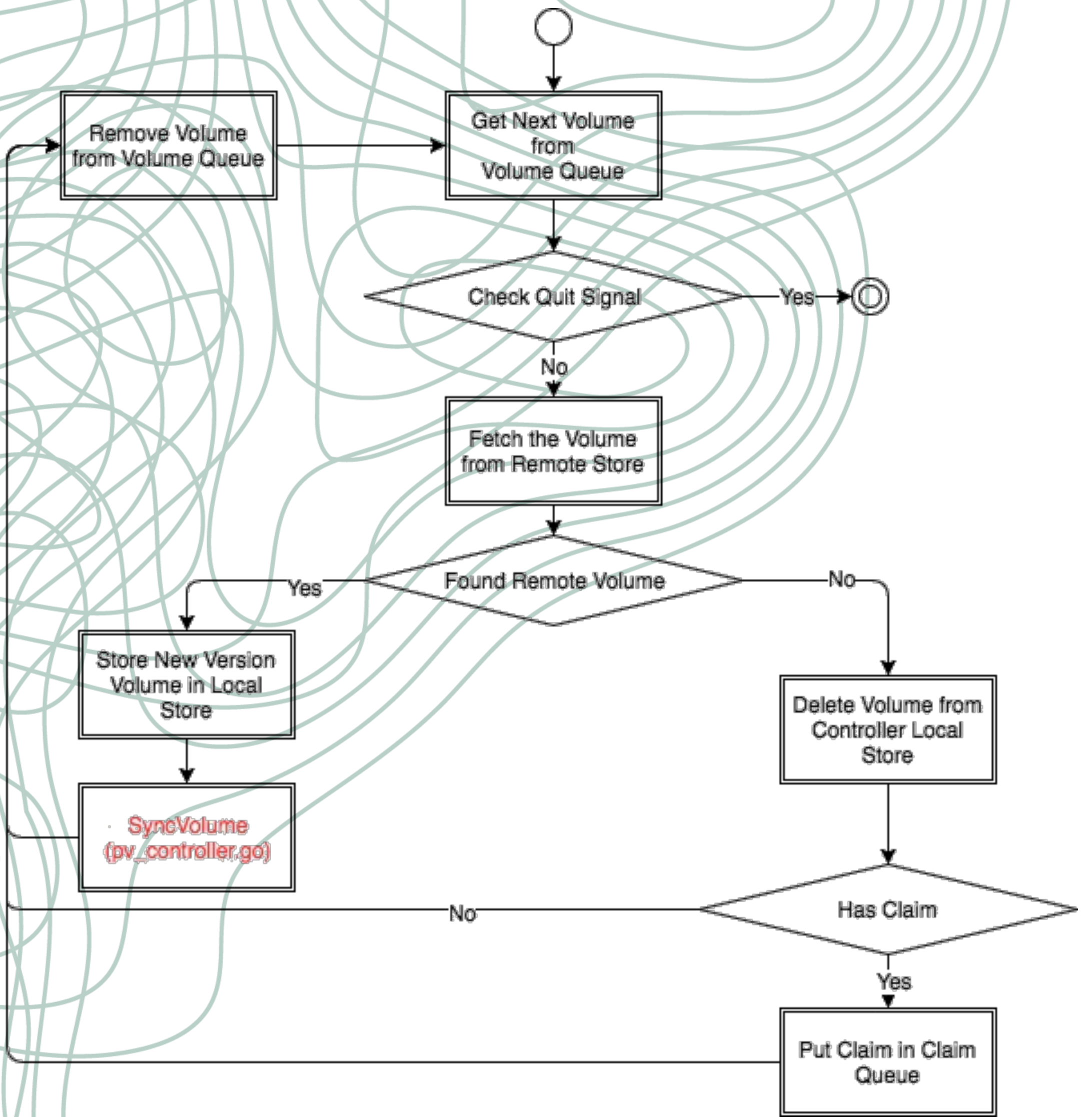
PV Controller的初始化过程



IBM developer.ibm.com/code

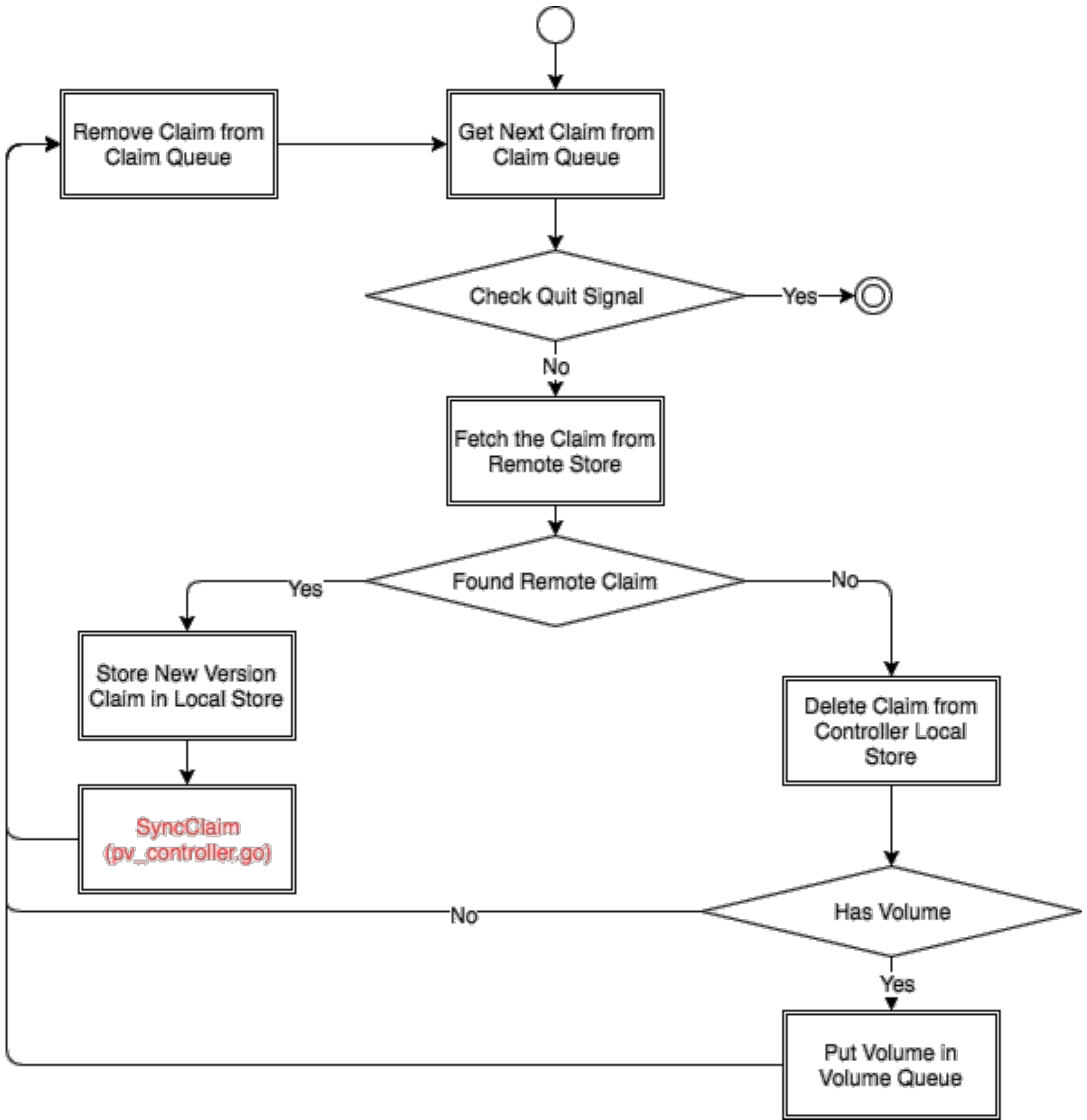
Volume Worker与Claim Worker的工作流程

- Volume Worker



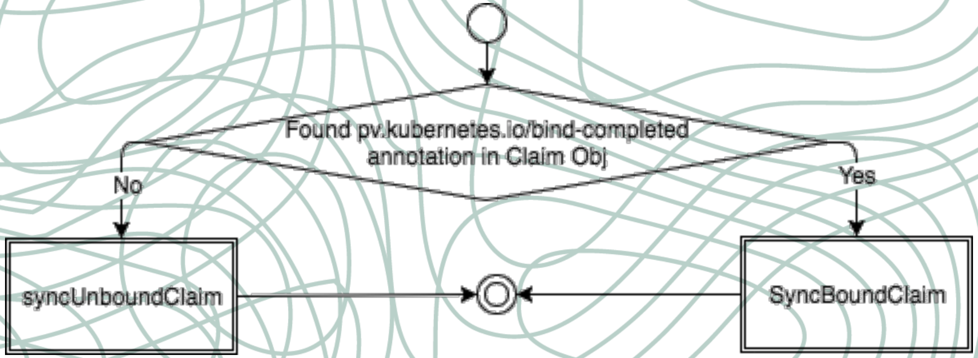
Get Next Volume from Volume Queue
Remove Volume from Volume Queue
Check Quit Signal
Yes
No
Fetch the Volume from Remote Store
Found Remote Volume
Yes
No
Store New Version Volume in Local Store
SyncVolume (pv_controller.go)
Delete Volume from Controller Local Store
Has Claim
No
Yes
Put Claim in Claim Queue

- Claim Worker

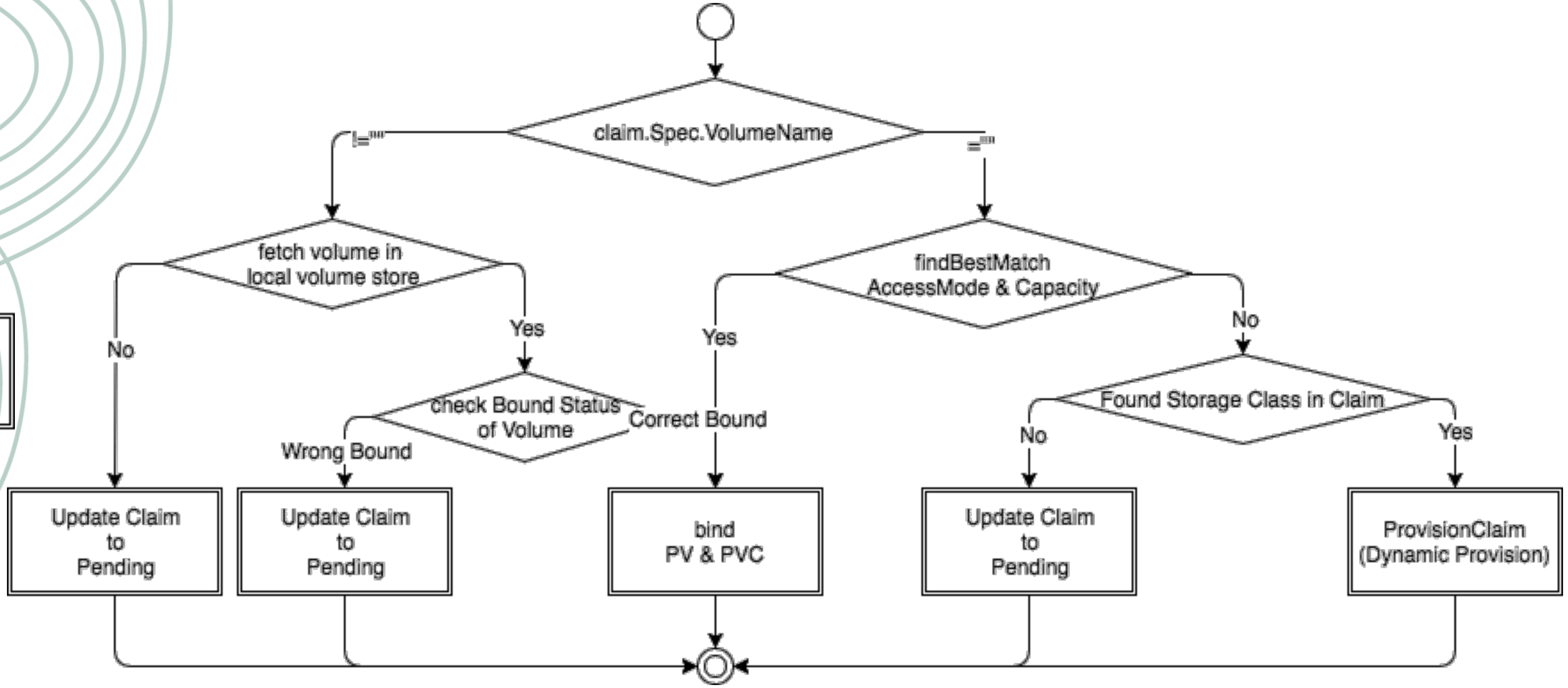


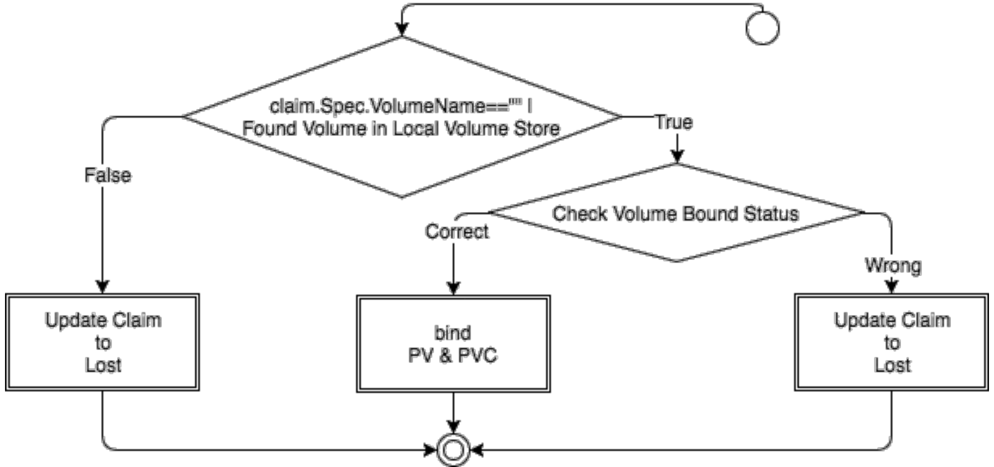
Remove Claim from Claim Queue
Get Next Claim from Claim Queue
Check Quit Signal
Yes
No
Fetch the Claim from Remote Store
Found Remote Claim
Yes
No
Store New Version Claim in Local Store
SyncClaim (pv_controller.go)
Delete Claim from Controller Local Store
Has Volume
No
Yes
Put Volume in Volume Queue

SyncClaim的基本逻辑(pv_controller.go)


Found pv.kubernetes.io/bind-completed annotation in Claim Obj
No
Yes
syncUnboundClaim
SyncBoundClaim

- PVC的基本状态
  - Pending
  - Bound
  - Lost
- Access Mode
  - RWX
  - RWO
  - ROX

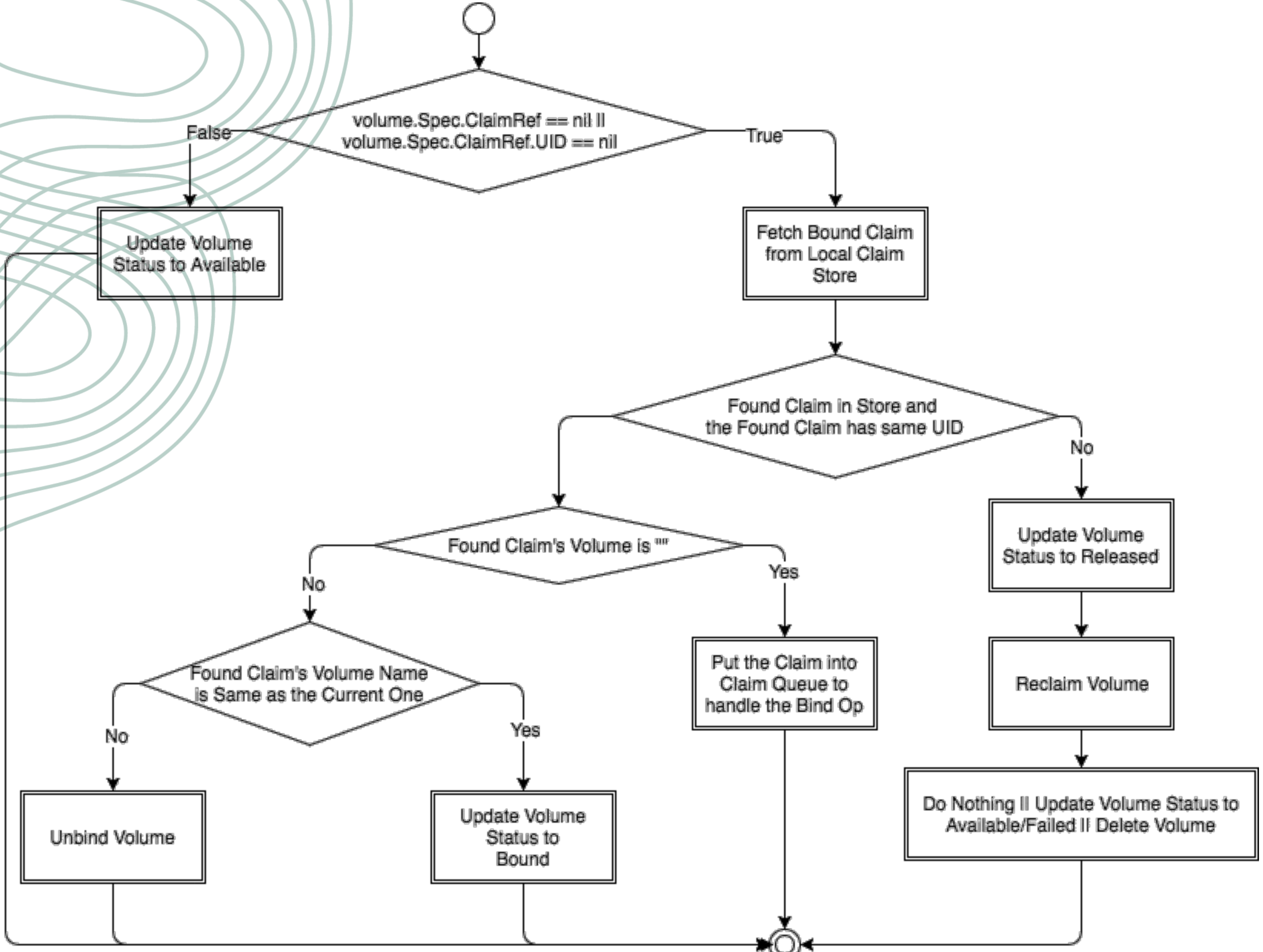

claim.Spec.VolumeName
!=""
==""
fetch volume in local volume store
No
Yes
check Bound Status of Volume
Wrong Bound
Correct Bound
findBestMatch AccessMode & Capacity
Yes
No
Found Storage Class in Claim
No
Yes
Update Claim to Pending
Update Claim to Pending
bind PV & PVC
Update Claim to Pending
ProvisionClaim (Dynamic Provision)


claim.Spec.VolumeName=="" |
Found Volume in Local Volume Store
True
False
Check Volume Bound Status
Correct
Wrong
Update Claim to Lost
bind PV & PVC
Update Claim to Lost

developer.ibm.com/**code**

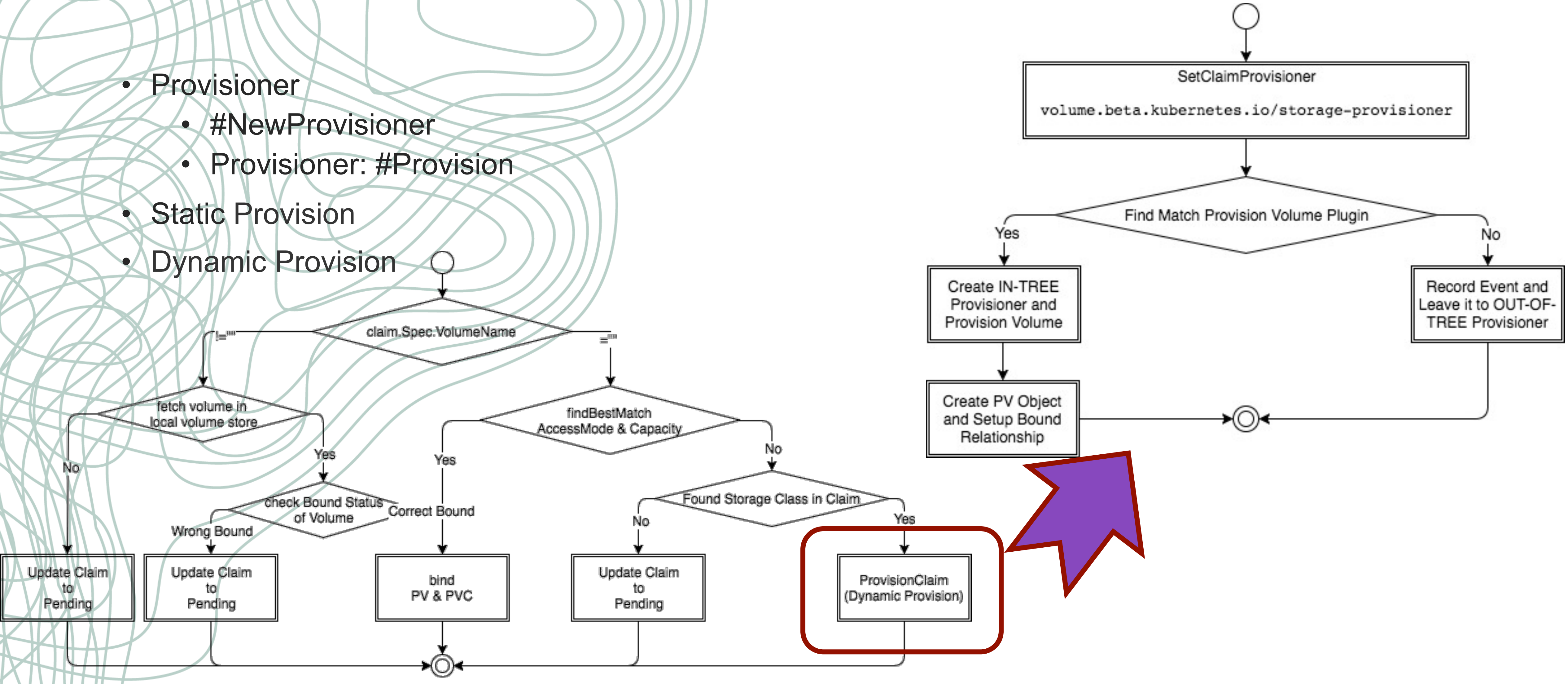
SyncVolume的基本逻辑(pv_controller.go)

- PV的基本状态
  - Pending
  - Available
  - Bound
  - Released
  - Failed
- Volume Recycle Policy
  - Retain
  - Recycle
  - Delete

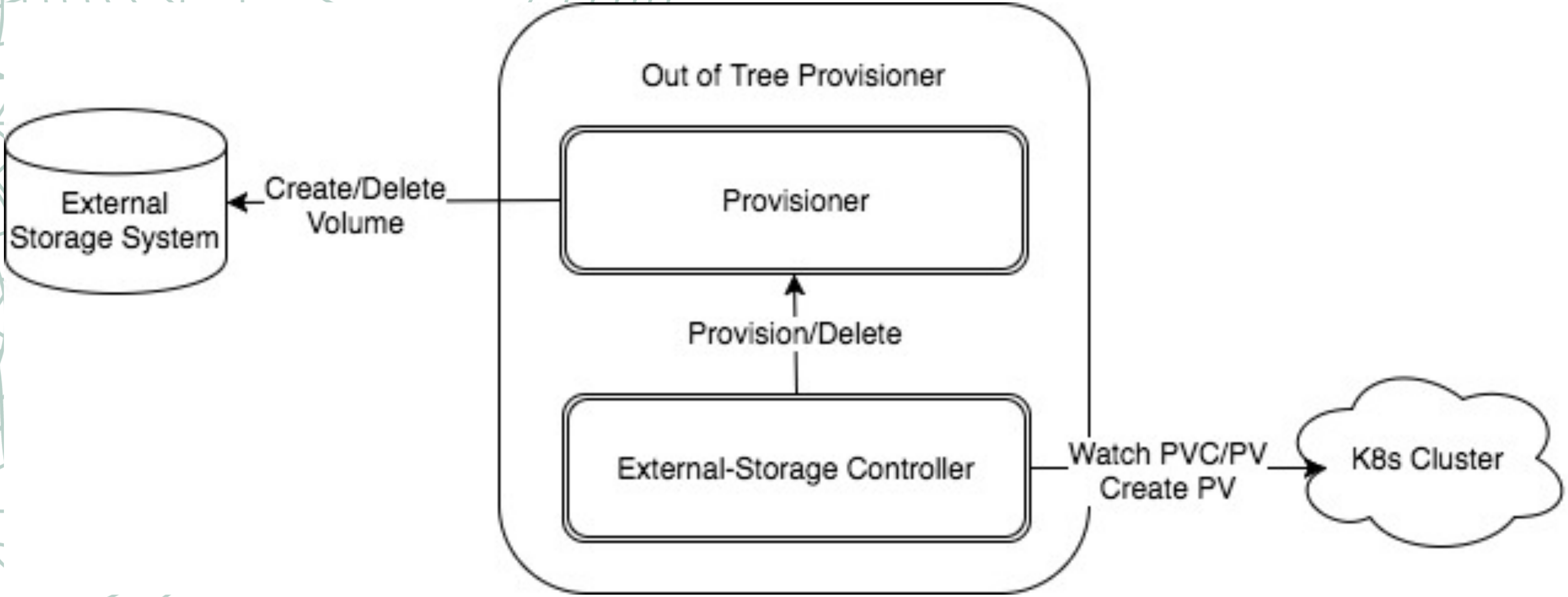

volume.Spec.ClaimRef == nil ll
volume.Spec.ClaimRef.UID == nil
False
True
Update Volume Status to Available
Fetch Bound Claim from Local Claim Store
Found Claim in Store and the Found Claim has same UID
No
Found Claim's Volume is ""
No
Yes
Update Volume Status to Released
Found Claim's Volume Name is Same as the Current One
No
Yes
Put the Claim into Claim Queue to handle the Bind Op
Reclaim Volume
Unbind Volume
Update Volume Status to Bound
Do Nothing ll Update Volume Status to Available/Failed ll Delete Volume

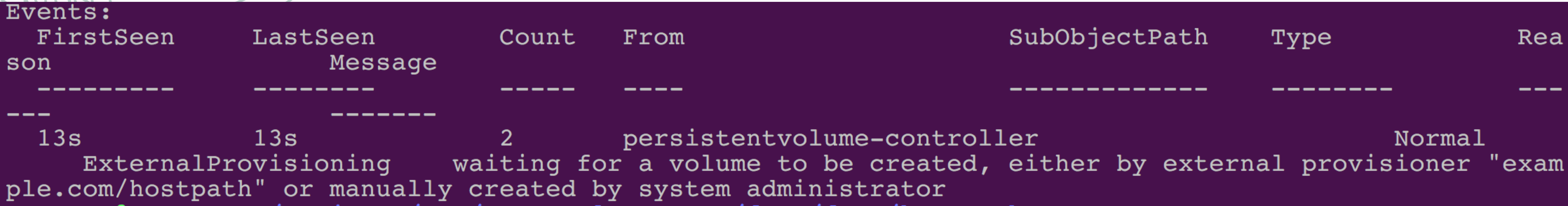
PV Controller中的Provision

- Provisioner
  - #NewProvisioner
  - Provisioner: #Provision
- Static Provision
- Dynamic Provision


claim.Spec.VolumeName
!=""
=""
fetch volume in local volume store
No
Yes
check Bound Status of Volume
Wrong Bound
Correct Bound
findBestMatch AccessMode & Capacity
Yes
No
Found Storage Class in Claim
No
Yes
Update Claim to Pending
Update Claim to Pending
bind PV & PVC
Update Claim to Pending
ProvisionClaim (Dynamic Provision)
SetClaimProvisioner
volume.beta.kubernetes.io/storage-provisioner
Find Match Provision Volume Plugin
Yes
No
Create IN-TREE Provisioner and Provision Volume
Record Event and Leave it to OUT-OF-TREE Provisioner
Create PV Object and Setup Bound Relationship

Out-of-Tree Provisioner的基本工作场景

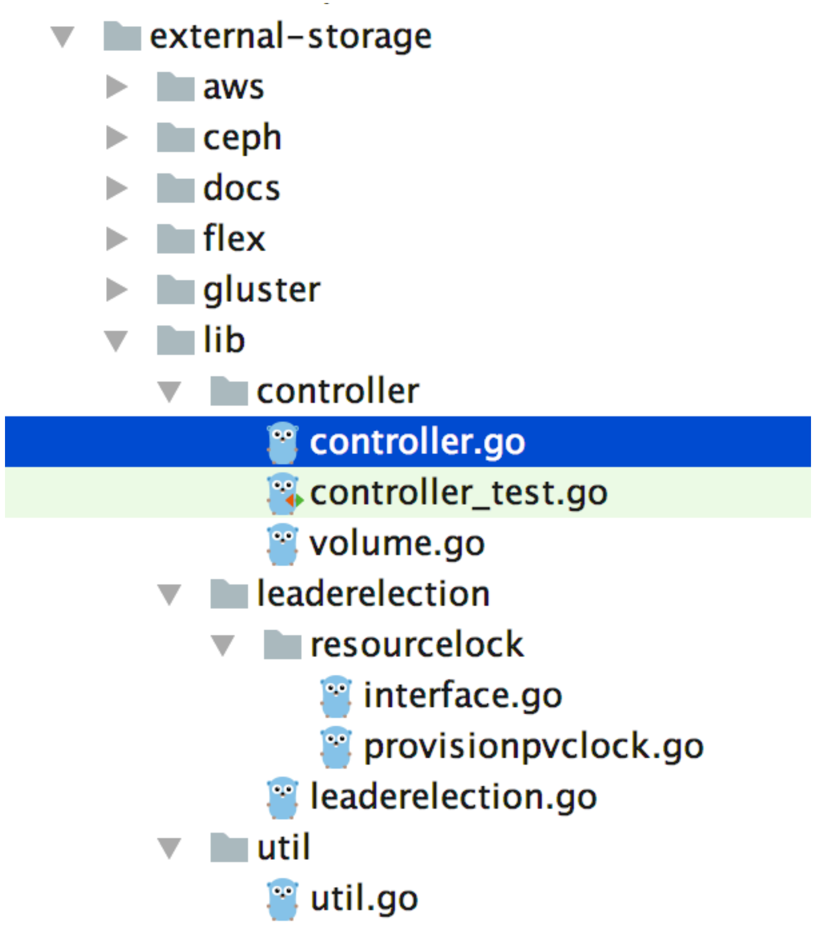




```
Events:
  FirstSeen     LastSeen        Count   From                            SubObjectPath   Type            Rea
son                     Message
  ---------     ---------       -----   ----                            -------------   --------        ---
---                     -------
  13s           13s             2       persistentvolume-controller                     Normal
    ExternalProvisioning    waiting for a volume to be created, either by external provisioner "exam
ple.com/hostpath" or manually created by system administrator
```

Out of Tree Provisioner的基本实现

- Incubator Project
  - https://github.com/kubernetes-incubator/external-storage
- 基本模块
  - Provisioner部分（Samples）
  - Controller部分
    - Watch PVC/PV
    - Create PV
    - Delete PV
    - Multi-Controller Lock
    - 代码位于 external-storage/lib/controller



```
▼ external-storage
    ▶ aws
    ▶ ceph
    ▶ docs
    ▶ flex
    ▶ gluster
    ▼ lib
        ▼ controller
              controller.go
              controller_test.go
              volume.go
        ▼ leaderelection
            ▼ resourcelock
                  interface.go
                  provisionpvclock.go
              leaderelection.go
        ▼ util
              util.go
```

Out of Tree Provisioner之多controller实例

- 利用锁机制在多个不同controller实例时间共享锁信息
- 代码位于external-storage/libs/leaderelection
- 定义了一个Provision锁以及锁的操作机制。锁信息将作为annotation存储于PVC中
- acquire->run->renew

```
type LeaderElectionRecord struct {
   HolderIdentity       string      `json:"holderIdentity"`
   LeaseDurationSeconds int
`json:"leaseDurationSeconds"`
   AcquireTime          metav1.Time
`json:"acquireTime"`
   RenewTime            metav1.Time
`json:"renewTime"`
   LeaderTransitions    int
`json:"leaderTransitions"`
}
```

IBM

developer.ibm.com/code

议程

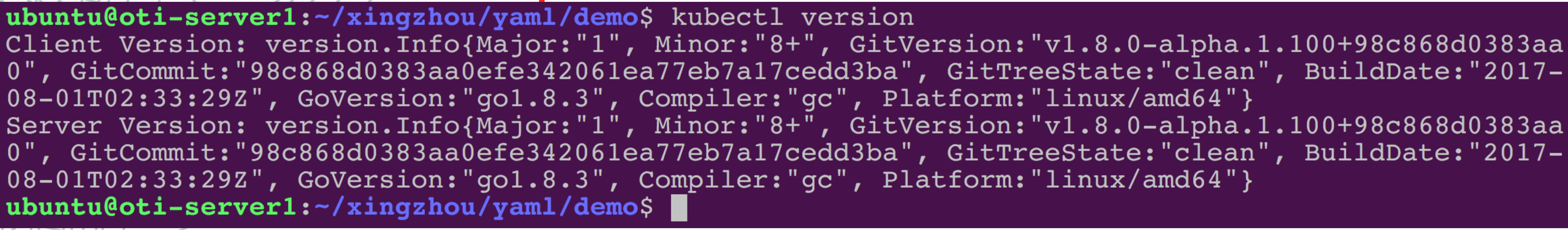
- K8s存储概览
  - 应用场景
  - 设计与基本架构
  - 目前社区所实现和维护的存储插件一览
  - 目前存储所存在的问题
- K8s Volume Provisioner部分的代码实现
  - 扩展K8s存储的几种方式
  - Persistent Volume与Persistent Volume Claim
  - PV Controller的实现
  - Out-of-Tree Provisioner的实现
- 一个简单的K8s存储示例

一个简单的例子

- 在虚拟机中启动一个all-in-one K8s集群以及配置一个NFS服务器
- 通过NFS暴露一个共享目录
- 创建PV,PVC和Pod, Pod将挂载NFS所暴露的目录

准备和启动环境

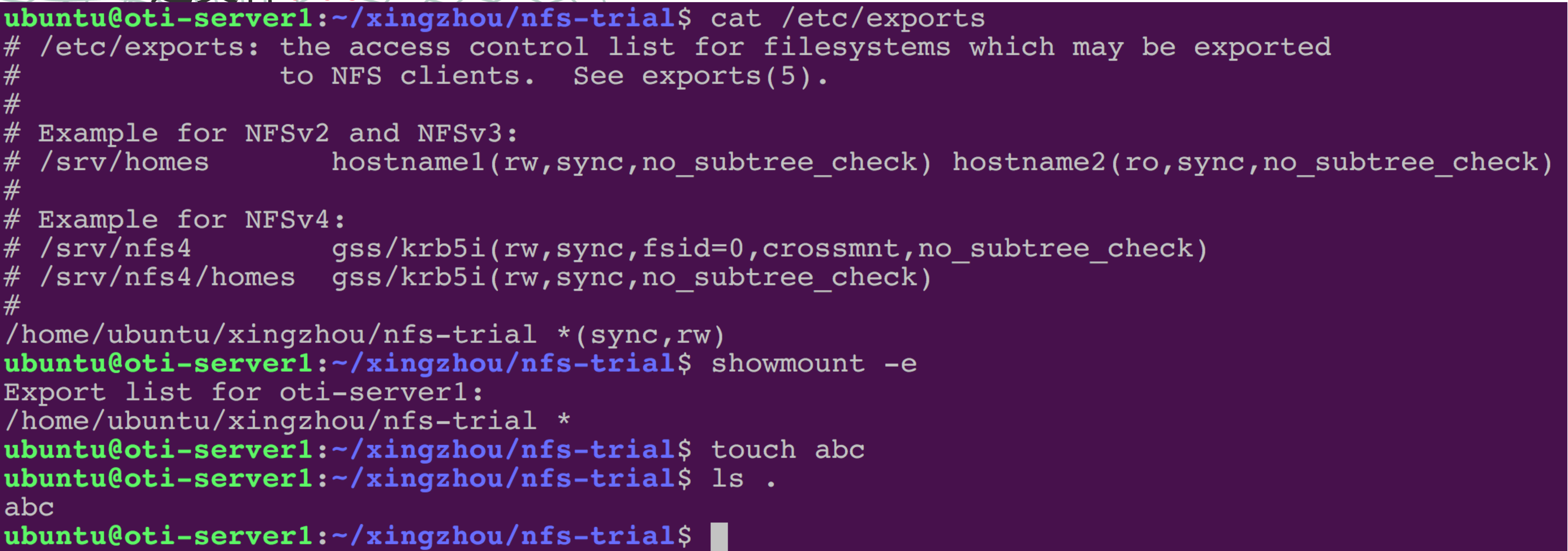
- 基本环境
  - ubuntu 16.04
  - Docker 1.12.6
  - NFS Server
  - Kubernetes Master Branch code
- 编译K8s源代码，启动K8s集群
  - ***sudo hack/local-up-cluster.sh***



```
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl version
Client Version: version.Info{Major:"1", Minor:"8+", GitVersion:"v1.8.0-alpha.1.100+98c868d0383aa
0", GitCommit:"98c868d0383aa0efe342061ea77eb7a17cedd3ba", GitTreeState:"clean", BuildDate:"2017-
08-01T02:33:29Z", GoVersion:"go1.8.3", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"8+", GitVersion:"v1.8.0-alpha.1.100+98c868d0383aa
0", GitCommit:"98c868d0383aa0efe342061ea77eb7a17cedd3ba", GitTreeState:"clean", BuildDate:"2017-
08-01T02:33:29Z", GoVersion:"go1.8.3", Compiler:"gc", Platform:"linux/amd64"}
ubuntu@oti-server1:~/xingzhou/yaml/demo$ 
```

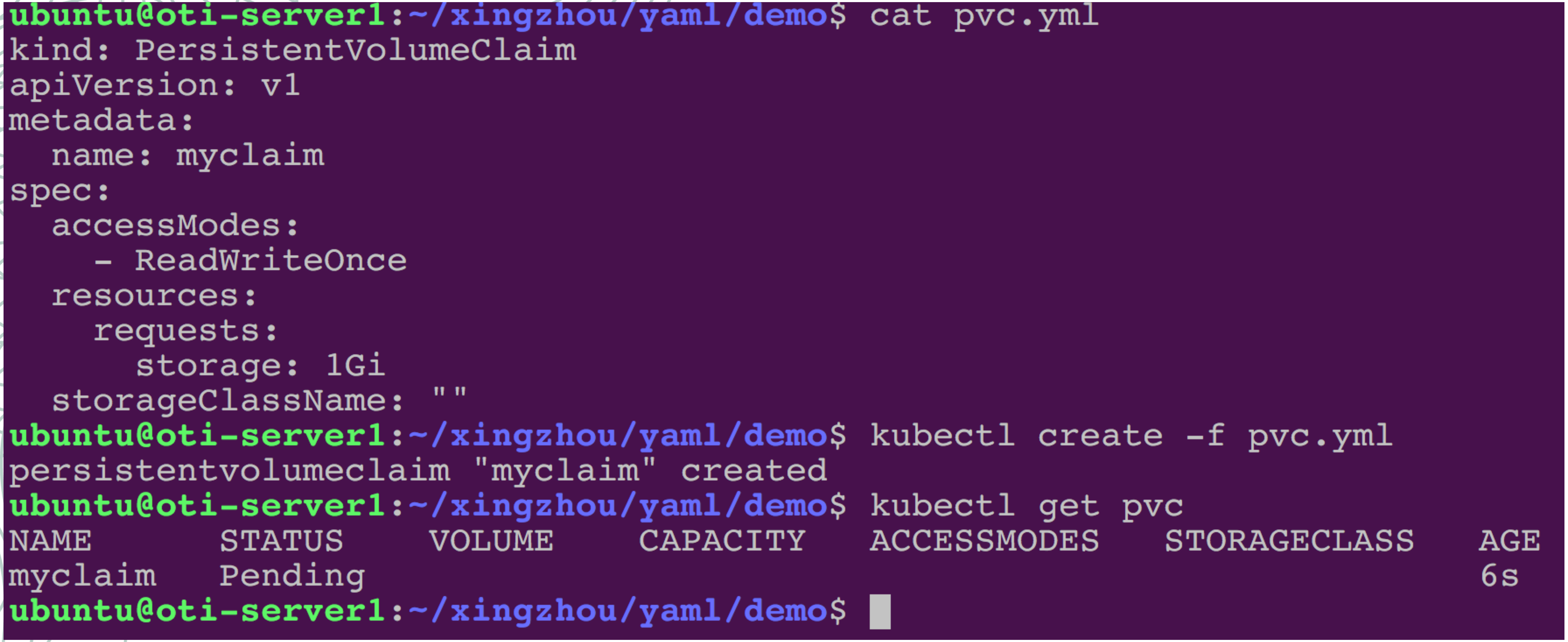
设置NFS Server

- 对外暴露*/home/ubuntu/xingzhou/nfs-trial*目录，并在目录下创建文件*abc*



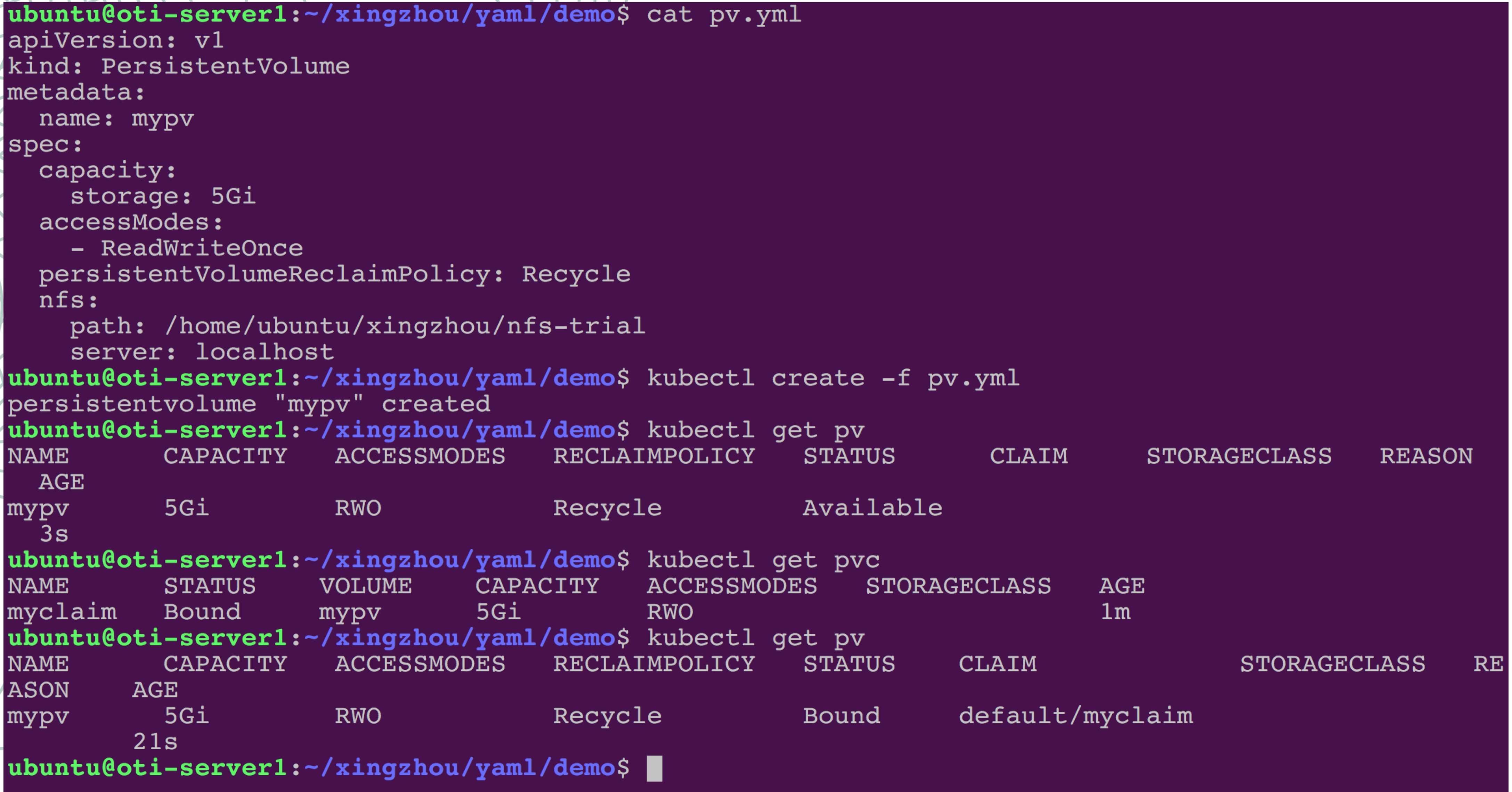
```
ubuntu@oti-server1:~/xingzhou/nfs-trial$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/home/ubuntu/xingzhou/nfs-trial *(sync,rw)
ubuntu@oti-server1:~/xingzhou/nfs-trial$ showmount -e
Export list for oti-server1:
/home/ubuntu/xingzhou/nfs-trial *
ubuntu@oti-server1:~/xingzhou/nfs-trial$ touch abc
ubuntu@oti-server1:~/xingzhou/nfs-trial$ ls .
abc
ubuntu@oti-server1:~/xingzhou/nfs-trial$ 
```

创建PVC



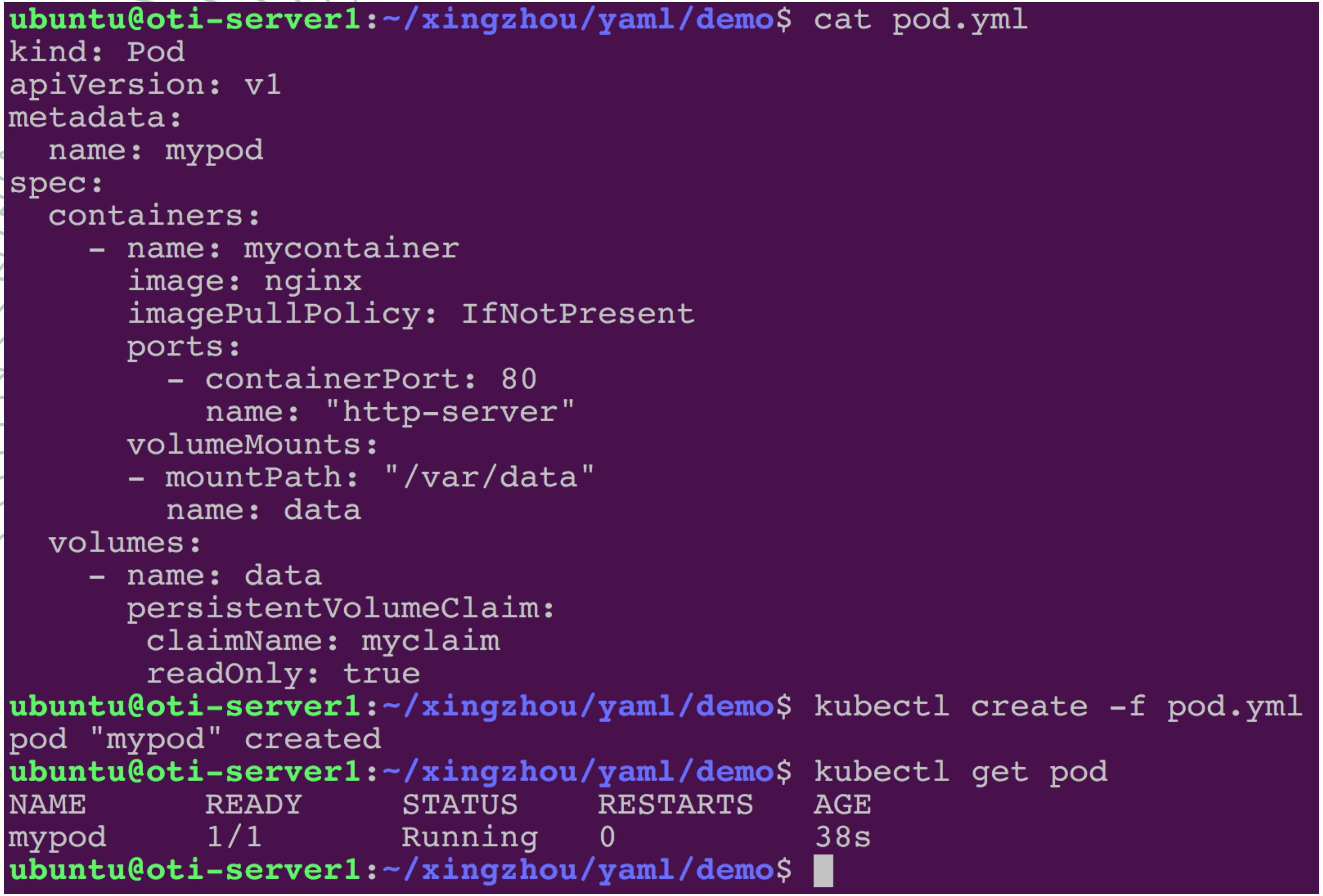
```
ubuntu@oti-server1:~/xingzhou/yaml/demo$ cat pvc.yml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: ""
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl create -f pvc.yml
persistentvolumeclaim "myclaim" created
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl get pvc
NAME      STATUS    VOLUME    CAPACITY   ACCESSMODES   STORAGECLASS   AGE
myclaim   Pending                                                   6s
ubuntu@oti-server1:~/xingzhou/yaml/demo$ 
```

创建PV



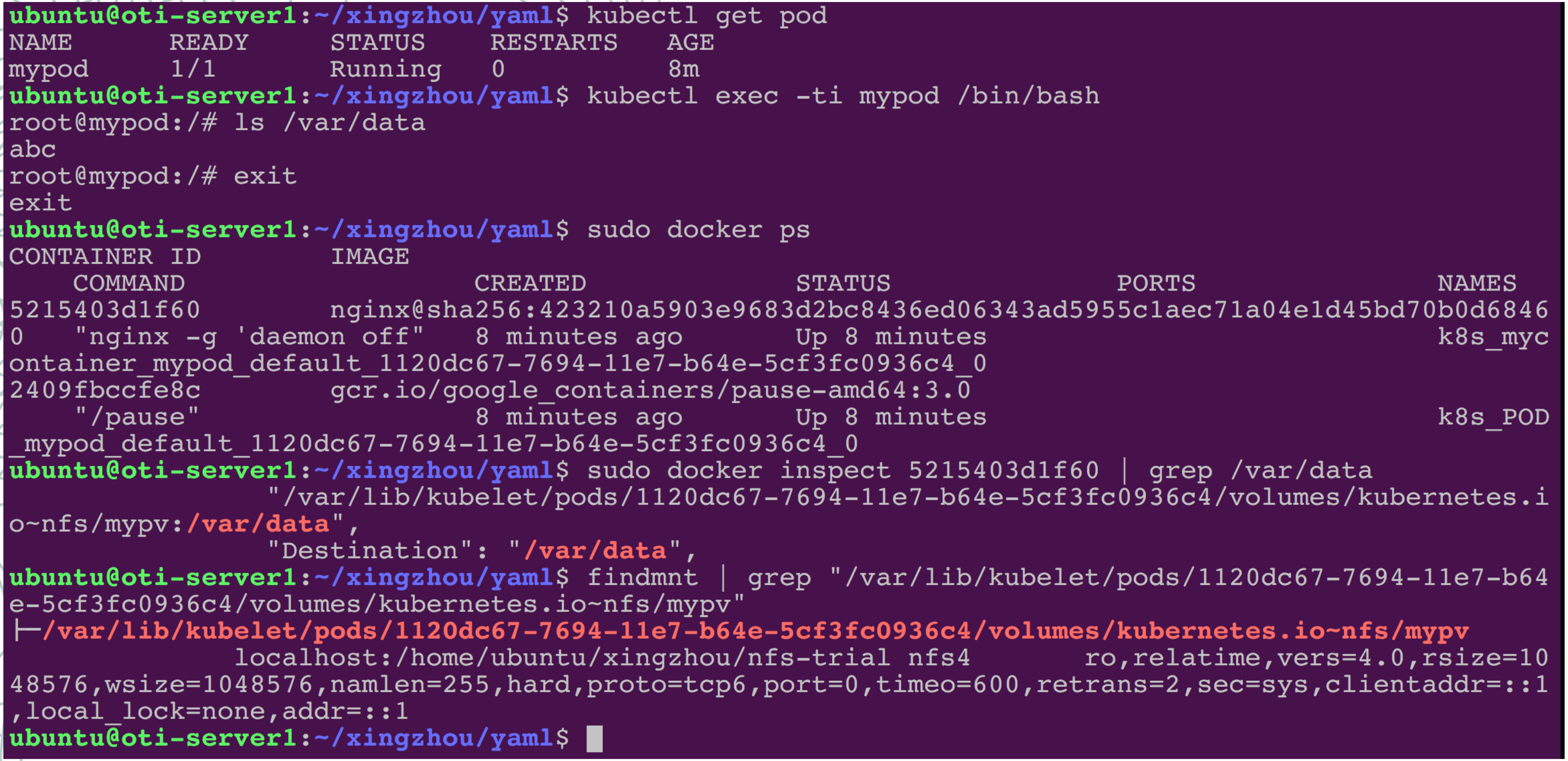
```
ubuntu@oti-server1:~/xingzhou/yaml/demo$ cat pv.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mypv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /home/ubuntu/xingzhou/nfs-trial
    server: localhost
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl create -f pv.yml
persistentvolume "mypv" created
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl get pv
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS      CLAIM     STORAGECLASS   REASON
  AGE
mypv      5Gi        RWO           Recycle         Available
  3s
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl get pvc
NAME      STATUS    VOLUME    CAPACITY   ACCESSMODES   STORAGECLASS   AGE
myclaim   Bound     mypv      5Gi        RWO                          1m
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl get pv
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM             STORAGECLASS   RE
ASON     AGE
mypv      5Gi        RWO           Recycle         Bound     default/myclaim
         21s
ubuntu@oti-server1:~/xingzhou/yaml/demo$ █
```

创建Pod



```
ubuntu@oti-server1:~/xingzhou/yaml/demo$ cat pod.yml
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: mycontainer
      image: nginx
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
      - mountPath: "/var/data"
        name: data
  volumes:
    - name: data
      persistentVolumeClaim:
       claimName: myclaim
       readOnly: true
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl create -f pod.yml
pod "mypod" created
ubuntu@oti-server1:~/xingzhou/yaml/demo$ kubectl get pod
NAME      READY     STATUS    RESTARTS   AGE
mypod     1/1       Running   0          38s
ubuntu@oti-server1:~/xingzhou/yaml/demo$
```

验证挂载目录



```
ubuntu@oti-server1:~/xingzhou/yaml$ kubectl get pod
NAME      READY     STATUS    RESTARTS   AGE
mypod     1/1       Running   0          8m
ubuntu@oti-server1:~/xingzhou/yaml$ kubectl exec -ti mypod /bin/bash
root@mypod:/# ls /var/data
abc
root@mypod:/# exit
exit
ubuntu@oti-server1:~/xingzhou/yaml$ sudo docker ps
CONTAINER ID        IMAGE
    COMMAND                  CREATED             STATUS              PORTS               NAMES
5215403d1f60        nginx@sha256:423210a5903e9683d2bc8436ed06343ad5955c1aec71a04e1d45bd70b0d6846
0   "nginx -g 'daemon off"   8 minutes ago       Up 8 minutes                            k8s_myc
ontainer_mypod_default_1120dc67-7694-11e7-b64e-5cf3fc0936c4_0
2409fbccfe8c        gcr.io/google_containers/pause-amd64:3.0
    "/pause"                 8 minutes ago       Up 8 minutes                            k8s_POD
_mypod_default_1120dc67-7694-11e7-b64e-5cf3fc0936c4_0
ubuntu@oti-server1:~/xingzhou/yaml$ sudo docker inspect 5215403d1f60 | grep /var/data
                "/var/lib/kubelet/pods/1120dc67-7694-11e7-b64e-5cf3fc0936c4/volumes/kubernetes.i
o~nfs/mypv:/var/data",
                "Destination": "/var/data",
ubuntu@oti-server1:~/xingzhou/yaml$ findmnt | grep "/var/lib/kubelet/pods/1120dc67-7694-11e7-b64
e-5cf3fc0936c4/volumes/kubernetes.io~nfs/mypv"
├─/var/lib/kubelet/pods/1120dc67-7694-11e7-b64e-5cf3fc0936c4/volumes/kubernetes.io~nfs/mypv
              localhost:/home/ubuntu/xingzhou/nfs-trial nfs4       ro,relatime,vers=4.0,rsize=10
48576,wsize=1048576,namlen=255,hard,proto=tcp6,port=0,timeo=600,retrans=2,sec=sys,clientaddr=::1
,local_lock=none,addr=::1
ubuntu@oti-server1:~/xingzhou/yaml$
```
]

一个简单的Dynamic-Provisioner例子

- 在虚拟机中启动一个all-in-one K8s集群

  https://github.com/kubernetes-incubator/external-storage/tree/master/docs/demo/hostpath-provisioner

- 创建PV,PVC和Pod, Pod将挂载external-storage所暴露的HostPath目录

谢谢大家