**加我微信可进群学习交流：**

**微信号：**

luckylucky421302

也可通过扫描下面二维码添加



**课程更新的知识点会通过微信公众号免费分享给大家，可以关注我的公众号**

****

官网地址：

<https://kubernetes.io/docs/concepts/storage/storage-classes/>

参考github地址：

<https://github.com/kubernetes-incubator/external-storage>

**1.storageclass（存储类）概念**

storageclass是一个存储类，k8s集群管理员通过创建storageclass可以动态生成一个存储卷供k8s用户使用。

**2.storageclass资源定义**

每个StorageClass都包含字段provisioner，parameters和reclaimPolicy，当需要动态配置属于该类的PersistentVolume时使用这些字段。

StorageClass对象的名称很重要，是用户可以请求特定类的方式。 管理员在首次创建StorageClass对象时设置类的名称和其他参数，并且在创建对象后无法更新这些对象。

管理员可以为不请求任何特定类绑定的PVC指定默认的StorageClass

**（1）yaml文件示例说明：**

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: standard

provisioner: kubernetes.io/aws-ebs

parameters:

type: gp2

reclaimPolicy: Retain

mountOptions:

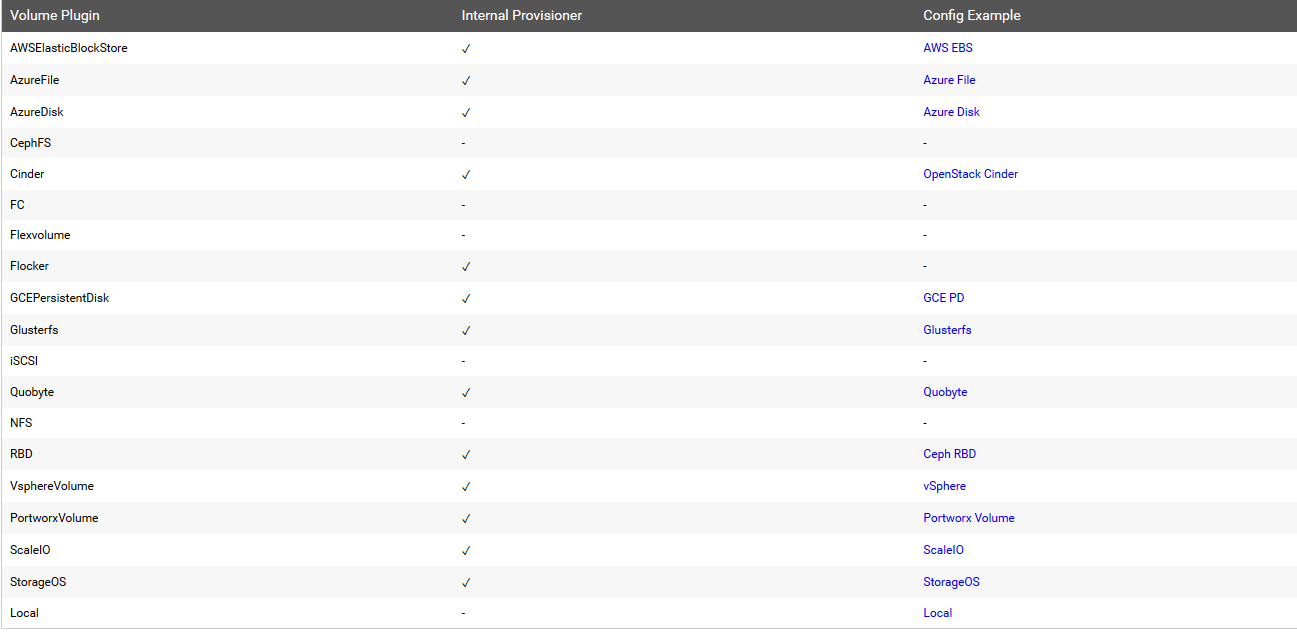
- debug

volumeBindingMode: Immediate

* **Provisioner**

storageclass需要有一个供应者，用来确定我们使用什么样的存储来创建pv

，常见的provisioner供应者如下：



provisioner既可以是内部供应程序，也可以由外部供应商提供，如果是外部供应商可以参考<https://github.com/kubernetes-incubator/external-storage/>下提供的方法创建storageclass的provisioner，例如，NFS不提供内部配置程序，但可以使用外部配置程序。 一些外部供应商列在存储库[https://github.com/kubernetes-incubator/external-storage](https://github.com/kubernetes-incubator/external-storage/tree/master/nfs)下。

nfs的provisioner：

<https://github.com/kubernetes-incubator/external-storage/tree/master/nfs/deploy/kubernetes>

* **Reclaim Policy（回收策略）**

由存储类动态创建持久化存储卷（pv）时可以指定reclaimPolicy字段，这个字段中指定的回收策略可以是Delete或Retain。 如果在创建StorageClass对象时未指定reclaimPolicy，则默认为Delete。

* **Mount Options（挂载选项）**

如果Volume Plugin不支持这个挂载选项，但是指定了，就会使provisioner创建失败

* **Volume Binding Mode（卷绑定模式）**

这个字段用来说明什么时候进行卷绑定和动态配置；

默认情况下，立即模式表示一旦创建了PersistentVolumeClaim，就会发生卷绑定和动态配置。对于受拓扑约束且无法从群集中的所有节点全局访问的存储后端，将在不知道Pod的调度要求的情况下绑定或配置PersistentVolumes。这可能导致不可调度的Pod。

集群管理员可以通过指定WaitForFirstConsumer模式来解决此问题，该模式将延迟绑定和配置PersistentVolume，直到创建使用PersistentVolumeClaim的Pod。将根据Pod的调度约束指定的拓扑选择或配置PersistentVolumes。这些包括但不限于资源需求，节点选择器，pod亲和力和反亲和力，以及污点和容忍度。

**允许卷扩展**

PersistentVolume 可以配置为可扩展。将此功能设置为 true 时，允许用户通过编辑相应的 PVC 对象来调整卷大小。

当基础存储类的 allowVolumeExpansion 字段设置为 true 时，以下类型的卷支持卷扩展。



**注意：**

此功能仅可用于扩容卷，不能用于缩小卷。

**允许卷拓扑结构**

当集群操作人员使用了 WaitForFirstConsumer 的卷绑定模式，在大部分情况下就没有必要将配置限制为特定的拓扑结构。 然而，如果还有需要的话，可以使用 allowedTopologies。这个例子描述了如何将分配卷的拓扑限制在特定的区域

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: standard

**provisioner**: kubernetes.io/gce-pd

**parameters**:

**type**: pd-standard

**volumeBindingMode**: WaitForFirstConsumer

**allowedTopologies**:

- **matchLabelExpressions**:

- **key**: failure-domain.beta.kubernetes.io/zone

**values**:

- us-central1-a

- us-central1-b

**参数**

Storage class 具有描述属于该存储类的卷的参数。可以接受的不同的参数取决于provisioner。 例如，参数 type 的值 io1 和参数 iopsPerGB 特定于 EBS PV。当参数被省略时，会使用默认值。一个 StorageClass 最多可以定义 512 个参数。这些参数对象的总长度不能超过 256 KiB, 包括参数的键和值。

### AWS EBS

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: slow

**provisioner**: kubernetes.io/aws-ebs

**parameters**:

**type**: io1

**iopsPerGB**: "10"

**fsType**: ext4

* type：io1，gp2，sc1，st1。详细信息参见 [AWS 文档](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html)。默认值：gp2。
* zone(弃用)：AWS 区域。如果没有指定 zone 和 zones，通常卷会在 Kubernetes 集群节点所在的活动区域中轮询调度分配。zone 和 zones 参数不能同时使用。
* zones(弃用)：以逗号分隔的 AWS 区域列表。如果没有指定 zone 和 zones，通常卷会在 Kubernetes 集群节点所在的活动区域中轮询调度分配。zone和zones参数不能同时使用。
* iopsPerGB：只适用于 io1 卷。每 GiB 每秒 I/O 操作。AWS 卷插件将其与请求卷的大小相乘以计算 IOPS 的容量，并将其限制在 20 000 IOPS（AWS 支持的最高值，请参阅 [AWS 文档](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html)。 这里需要输入一个字符串，即 "10"，而不是 10。
* fsType：受 Kubernetes 支持的文件类型。默认值："ext4"。
* encrypted：指定 EBS 卷是否应该被加密。合法值为 "true" 或者 "false"。这里需要输入字符串，即 "true", 而非 true。
* kmsKeyId：可选。加密卷时使用密钥的完整 Amazon 资源名称。如果没有提供，但 encrypted 值为 true，AWS 生成一个密钥。关于有效的 ARN 值，请参阅 AWS 文档。

**注意：**

zone 和 zones 已被弃用并被 [允许的拓扑结构](https://kubernetes.io/zh/docs/concepts/storage/storage-classes/#allowed-topologies) 取代。

### GCE PD

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: slow

**provisioner**: kubernetes.io/gce-pd

**parameters**:

**type**: pd-standard

**replication-type**: none

* type：pd-standard 或者 pd-ssd。默认：pd-standard
* zone(弃用)：GCE 区域。如果没有指定 zone 和 zones，通常卷会在 Kubernetes 集群节点所在的活动区域中轮询调度分配。zone 和 zones 参数不能同时使用。
* zones(弃用)：逗号分隔的 GCE 区域列表。如果没有指定 zone 和 zones，通常卷会在 Kubernetes 集群节点所在的活动区域中轮询调度（round-robin）分配。zone 和 zones 参数不能同时使用。
* fstype: ext4 或 xfs。 默认: ext4。宿主机操作系统必须支持所定义的文件系统类型。
* replication-type：none 或者 regional-pd。默认值：none。

如果 replication-type 设置为 none，会分配一个常规（当前区域内的）持久化磁盘。

如果 replication-type 设置为 regional-pd，会分配一个 [区域性持久化磁盘（Regional Persistent Disk）](https://cloud.google.com/compute/docs/disks/#repds)。在这种情况下，用户必须使用 zones 而非 zone 来指定期望的复制区域（zone）。如果指定来两个特定的区域，区域性持久化磁盘会在这两个区域里分配。如果指定了多于两个的区域，Kubernetes 会选择其中任意两个区域。如果省略了 zones 参数，Kubernetes 会在集群管理的区域中任意选择。

**注意：**

zone 和 zones 已被弃用并被 [allowedTopologies](https://kubernetes.io/zh/docs/concepts/storage/storage-classes/" \l "allowed-topologies) 取代。

### Glusterfs

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: slow

**provisioner**: kubernetes.io/glusterfs

**parameters**:

**resturl**: "http://127.0.0.1:8081"

**clusterid**: "630372ccdc720a92c681fb928f27b53f"

**restauthenabled**: "true"

**restuser**: "admin"

**secretNamespace**: "default"

**secretName**: "heketi-secret"

**gidMin**: "40000"

**gidMax**: "50000"

**volumetype**: "replicate:3"

* resturl：分配 gluster 卷的需求的 Gluster REST 服务/Heketi服务url。 通用格式应该是 IPaddress:Port，这是 GlusterFS 动态分配器的必需参数。 如果 Heketi 服务在 openshift/kubernetes 中安装并暴露为可路由服务，则可以使用类似于 http://heketi-storage-project.cloudapps.mystorage.com 的格式，其中 fqdn 是可解析的 heketi 服务网址。

Heketi可用于管理glusterfs

* restauthenabled：Gluster REST 服务身份验证布尔值，用于启用对 REST 服务器的身份验证。如果此值为 ‘true’，则必须填写 restuser 和 restuserkey 或 secretNamespace + secretName。此选项已弃用，当在指定 restuser，restuserkey，secretName 或 secretNamespace 时，身份验证被启用。
* restuser：在 Gluster 可信池中有权创建卷的 Gluster REST服务/Heketi 用户。
* restuserkey：Gluster REST服务/Heketi用户的密码将被用于对 REST 服务器进行身份验证。此参数已弃用，取而代之的是secretNamespace+secretName。
* secretNamespace，secretName：Secret 实例的标识，包含与 Gluster REST 服务交互时使用的用户密码。 这些参数是可选的，secretNamespace和secretName都省略时使用空密码。所提供的 Secret 必须将类型设置为“kubernetes.io/glusterfs”，例如以这种方式创建：

kubectl create secret generic heketi-secret --type="kubernetes.io/glusterfs" --from-literal=key='opensesame' --namespace=default

secret 的例子可以在 [glusterfs-provisioning-secret.yaml](https://github.com/kubernetes/examples/tree/master/staging/persistent-volume-provisioning/glusterfs/glusterfs-secret.yaml) 中找到。

* clusterid：630372ccdc720a92c681fb928f27b53f是集群的ID，当分配卷时，Heketi 将会使用这个文件。它也可以是一个 clusterid 列表，例如： "8452344e2becec931ece4e33c4674e4e,42982310de6c63381718ccfa6d8cf397"。这个是可选参数。
* gidMin，gidMax：storage class GID 范围的最小值和最大值。在此范围（gidMin-gidMax）内的唯一值（GID）将用于动态分配卷。这些是可选的值。如果不指定，卷将被分配一个 2000-2147483647 之间的值，这是 gidMin 和 gidMax 的默认值。
* volumetype：卷的类型及其参数可以用这个可选值进行配置。如果未声明卷类型，则由分配器决定卷的类型。例如：volumetype: replicate:3 其中 ‘3’ 是 replica 数量. ‘Disperse/EC volume’: volumetype: disperse:4:2 其中 ‘4’ 是数据，‘2’ 是冗余数量,也可以把volumetype设置成none，volumetype: none

有关可用的卷类型和管理选项，请参阅[管理指南](https://access.redhat.com/documentation/en-US/Red_Hat_Storage/3.1/html/Administration_Guide/part-Overview.html)。

更多相关的参考信息，请参阅[如何配置 Heketi](https://github.com/heketi/heketi/wiki/Setting-up-the-topology)。

当动态分配持久卷时，Gluster插件自动创建名为gluster-dynamic-<claimname> 的端点和 headless service。在 PVC 被删除时动态端点和 headless service 会自动被删除。

### OpenStack Cinder

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: gold

**provisioner**: kubernetes.io/cinder

**parameters**:

**availability**: nova

* availability：可用区域。如果没有指定，通常卷会在 Kubernetes 集群节点所在的活动区域中轮询调度分配。

**注意：**

**FEATURE STATE:** Kubernetes 1.11 [deprecated]

OpenStack 的内部驱动程序已经被弃用。请使用 [OpenStack 的外部驱动程序](https://github.com/kubernetes/cloud-provider-openstack)。

### vSphere

使用用户指定的磁盘格式创建一个 StorageClass。

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: fast

**provisioner**: kubernetes.io/vsphere-volume

**parameters**:

**diskformat**: zeroedthick

`diskformat`: `thin`, `zeroedthick` 和 `eagerzeroedthick`。默认值: `"thin"`。

在用户指定的数据存储上创建磁盘格式的 StorageClass。

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: fast

**provisioner**: kubernetes.io/vsphere-volume

**parameters**:

**diskformat**: zeroedthick

**datastore**: VSANDatastore

`datastore`：用户也可以在 StorageClass 中指定数据存储。卷将在 storage class 中指定的数据存储上创建，在这种情况下是 `VSANDatastore`。该字段是可选的。如果未指定数据存储，则将在用于初始化 vSphere Cloud Provider 的 vSphere 配置文件中指定的数据存储上创建该卷。

Kubernetes 中的存储策略管理

使用现有的 vCenter SPBM 策略

vSphere 用于存储管理的最重要特性之一是基于策略的管理。基于存储策略的管理（SPBM）是一个存储策略框架，提供单一的统一控制平面的跨越广泛的数据服务和存储解决方案。 SPBM 使能 vSphere 管理员克服先期的存储配置挑战，如容量规划，差异化服务等级和管理容量空间。

SPBM 策略可以在StorageClass 中使用 `storagePolicyName` 参数声明。

Kubernetes 内的 Virtual SAN 策略支持

Vsphere Infrastructure（VI）管理员将能够在动态卷配置期间指定自定义 Virtual SAN 存储功能。你现在可以定义存储需求，例如性能和可用性，当动态卷供分配时会以存储功能的形式提供。存储功能需求会转换为Virtual SAN 策略，然后当 persistent volume（虚拟磁盘）在创建时，会将其推送到Virtual SAN 层。虚拟磁盘分布在 Virtual SAN 数据存储中以满足要求。

有几个 [vSphere 例子](https://github.com/kubernetes/examples/tree/master/staging/volumes/vsphere) 供你在 Kubernetes for vSphere 中尝试进行 persistent volume 管理。

### Ceph RBD

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: fast

**provisioner**: kubernetes.io/rbd

**parameters**:

**monitors**: 10.16.153.105:6789

**adminId**: kube

**adminSecretName**: ceph-secret

**adminSecretNamespace**: kube-system

**pool**: kube

**userId**: kube

**userSecretName**: ceph-secret-user

**userSecretNamespace**: default

**fsType**: ext4

**imageFormat**: "2"

**imageFeatures**: "layering"

* monitors：Ceph monitor，逗号分隔。该参数是必需的。
* adminId：Ceph 客户端 ID，用于在池 ceph 池中创建映像。默认是 “admin”。
* adminSecret：adminId 的 Secret 名称。该参数是必需的。 提供的 secret 必须有值为 “kubernetes.io/rbd” 的 type 参数。
* adminSecretNamespace：adminSecret 的命名空间。默认是 “default”。
* pool: Ceph RBD 池. 默认是 “rbd”。
* userId：Ceph 客户端 ID，用于映射 RBD 镜像。默认与 adminId 相同。
* userSecretName：用于映射 RBD 镜像的 userId 的Ceph Secret的名字。 它必须与PVC存在于相同的namespace 中。该参数是必需的。提供的 secret必须具有值为 “kubernetes.io/rbd” 的 type参数，例如以这样的方式创建：

kubectl create secret generic ceph-secret --type="kubernetes.io/rbd" **\**--from-literal=key='QVFEQ1pMdFhPUnQrSmhBQUFYaERWNHJsZ3BsMmNjcDR6RFZST0E9PQ==' **\** --namespace=kube-system

* userSecretNamespace：userSecretName 的命名空间。
* fsType：Kubernetes支持的fsType。默认："ext4"。
* imageFormat：Ceph RBD镜像格式，“1” 或者 “2”。默认值是 “1”。
* imageFeatures：这个参数是可选的，只能在你将 imageFormat 设置为 “2” 才使用。目前支持的功能只是 layering。默认是 “"，没有功能打开。

### Quobyte

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: slow

**provisioner**: kubernetes.io/quobyte

**parameters**:

**quobyteAPIServer**: "http://138.68.74.142:7860"

**registry**: "138.68.74.142:7861"

**adminSecretName**: "quobyte-admin-secret"

**adminSecretNamespace**: "kube-system"

**user**: "root"

**group**: "root"

**quobyteConfig**: "BASE"

**quobyteTenant**: "DEFAULT"

* quobyteAPIServer：Quobyte API 服务器的格式是 "http(s)://api-server:7860"
* registry：用于挂载卷的 Quobyte registry。你可以指定 registry 为 <host>:<port> 或者如果你想指定多个 registry，你只需要在他们之间添加逗号，例如 <host1>:<port>,<host2>:<port>,<host3>:<port>。 主机可以是一个 IP 地址，或者如果您有正在运行的 DNS，您也可以提供 DNS 名称。
* adminSecretNamespace：adminSecretName的 namespace。 默认值是 “default”。
* adminSecretName：保存关于 Quobyte 用户和密码的 secret，用于对 API 服务器进行身份验证。 提供的 secret 必须有值为 “kubernetes.io/quobyte” 的 type 参数 和 user 与 password 的键值， 例如以这种方式创建：

kubectl create secret generic quobyte-admin-secret **\**

--type="kubernetes.io/quobyte" --from-literal=key='opensesame' **\**

--namespace=kube-system

* user：对这个用户映射的所有访问权限。默认是 “root”。
* group：对这个组映射的所有访问权限。默认是 “nfsnobody”。
* quobyteConfig：使用指定的配置来创建卷。您可以创建一个新的配置，或者，可以修改 Web console 或 quobyte CLI 中现有的配置。默认是 “BASE”。
* quobyteTenant：使用指定的租户 ID 创建/删除卷。这个 Quobyte 租户必须已经于 Quobyte。 默认是 “DEFAULT”。

### Azure 磁盘

### Azure Unmanaged Disk Storage Class（非托管磁盘存储类）

**kind**: StorageClass

**apiVersion**: storage.k8s.io/v1

**metadata**:

**name**: slow

**provisioner**: kubernetes.io/azure-disk

**parameters**:

**skuName**: Standard\_LRS

**location**: eastus

**storageAccount**: azure\_storage\_account\_name

* skuName：Azure 存储帐户 Sku 层。默认为空。
* location：Azure 存储帐户位置。默认为空。
* storageAccount：Azure 存储帐户名称。如果提供存储帐户，它必须位于与集群相同的资源组中，并且 location 是被忽略的。如果未提供存储帐户，则会在与群集相同的资源组中创建新的存储帐户。

### Azure 磁盘 Storage Class（从 v1.7.2 开始）

**kind**: StorageClass

**apiVersion**: storage.k8s.io/v1

**metadata**:

**name**: slow

**provisioner**: kubernetes.io/azure-disk

**parameters**:

**storageaccounttype**: Standard\_LRS

**kind**: Shared

* storageaccounttype：Azure 存储帐户 Sku 层。默认为空。
* kind：可能的值是 shared（默认）、dedicated 和 managed。 当 kind 的值是 shared 时，所有非托管磁盘都在集群的同一个资源组中的几个共享存储帐户中创建。 当 kind 的值是 dedicated 时，将为在集群的同一个资源组中新的非托管磁盘创建新的专用存储帐户。
* resourceGroup: 指定要创建 Azure 磁盘所属的资源组。必须是已存在的资源组名称。若未指定资源组，磁盘会默认放入与当前 Kubernetes 集群相同的资源组中。
* Premium VM 可以同时添加 Standard\_LRS 和 Premium\_LRS 磁盘，而 Standard 虚拟机只能添加 Standard\_LRS 磁盘。
* 托管虚拟机只能连接托管磁盘，非托管虚拟机只能连接非托管磁盘。

### Azure 文件

**kind**: StorageClass

**apiVersion**: storage.k8s.io/v1

**metadata**:

**name**: azurefile

**provisioner**: kubernetes.io/azure-file

**parameters**:

**skuName**: Standard\_LRS

**location**: eastus

**storageAccount**: azure\_storage\_account\_name

* skuName：Azure 存储帐户 Sku 层。默认为空。
* location：Azure 存储帐户位置。默认为空。
* storageAccount：Azure 存储帐户名称。默认为空。 如果不提供存储帐户，会搜索所有与资源相关的存储帐户，以找到一个匹配 skuName 和 location 的账号。 如果提供存储帐户，它必须存在于与集群相同的资源组中，skuName 和 location 会被忽略。
* secretNamespace：包含 Azure 存储帐户名称和密钥的密钥的名称空间。 默认值与 Pod 相同。
* secretName：包含 Azure 存储帐户名称和密钥的密钥的名称。 默认值为 azure-storage-account-<accountName>-secret
* readOnly：指示是否将存储安装为只读的标志。默认为 false，表示 读/写 挂载。 该设置也会影响VolumeMounts中的 ReadOnly 设置。

在存储分配期间，为挂载凭证创建一个名为 secretName 的 secret。如果集群同时启用了 [RBAC](https://kubernetes.io/docs/admin/authorization/rbac/) 和 [Controller Roles](https://kubernetes.io/docs/admin/authorization/rbac/#controller-roles)， 为 system:controller:persistent-volume-binder 的 clusterrole 添加 secret 资源的 create 权限。

在多租户上下文中，强烈建议显式设置 secretNamespace 的值，否则其他用户可能会读取存储帐户凭据。

### Portworx 卷

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: portworx-io-priority-high

**provisioner**: kubernetes.io/portworx-volume

**parameters**:

**repl**: "1"

**snap\_interval**: "70"

**io\_priority**: "high"

* fs：选择的文件系统：none/xfs/ext4（默认：ext4）。
* block\_size：以 Kbytes 为单位的块大小（默认值：32）。
* repl：同步副本数量，以复制因子 1..3（默认值：1）的形式提供。 这里需要填写字符串，即，"1" 而不是 1。
* io\_priority：决定是否从更高性能或者较低优先级存储创建卷 high/medium/low（默认值：low）。
* snap\_interval：触发快照的时钟/时间间隔（分钟）。快照是基于与先前快照的增量变化，0 是禁用快照（默认：0）。 这里需要填写字符串，即，是 "70" 而不是 70。
* aggregation\_level：指定卷分配到的块数量，0 表示一个非聚合卷（默认：0）。 这里需要填写字符串，即，是 "0" 而不是 0。
* ephemeral：指定卷在卸载后进行清理还是持久化。 emptyDir 的使用场景可以将这个值设置为 true ， persistent volumes 的使用场景可以将这个值设置为 false（例如 Cassandra 这样的数据库）true/false（默认为 false）。这里需要填写字符串，即，是 "true" 而不是 true。

### ScaleIO

**kind**: StorageClass

**apiVersion**: storage.k8s.io/v1

**metadata**:

**name**: slow

**provisioner**: kubernetes.io/scaleio

**parameters**:

**gateway**: https://192.168.99.200:443/api

**system**: scaleio

**protectionDomain**: pd0

**storagePool**: sp1

**storageMode**: ThinProvisioned

**secretRef**: sio-secret

**readOnly**: **false**

**fsType**: xfs

* provisioner：属性设置为 kubernetes.io/scaleio
* gateway 到 ScaleIO API 网关的地址（必需）
* system：ScaleIO 系统的名称（必需）
* protectionDomain：ScaleIO 保护域的名称（必需）
* storagePool：卷存储池的名称（必需）
* storageMode：存储提供模式：ThinProvisioned（默认）或 ThickProvisioned
* secretRef：对已配置的 Secret 对象的引用（必需）
* readOnly：指定挂载卷的访问模式（默认为 false）
* fsType：卷的文件系统（默认是 ext4）

ScaleIO Kubernetes 卷插件需要配置一个 Secret 对象。 secret 必须用 kubernetes.io/scaleio 类型创建，并与引用它的 PVC 所属的名称空间使用相同的值 如下面的命令所示：

kubectl create secret generic sio-secret --type="kubernetes.io/scaleio" **\**

--from-literal=username=sioadmin --from-literal=password=d2NABDNjMA== **\**

--namespace=default

### StorageOS

**apiVersion**: storage.k8s.io/v1

**kind**: StorageClass

**metadata**:

**name**: fast

**provisioner**: kubernetes.io/storageos

**parameters**:

**pool**: default

**description**: Kubernetes volume

**fsType**: ext4

**adminSecretNamespace**: default

**adminSecretName**: storageos-secret

* pool：分配卷的 StorageOS 分布式容量池的名称。如果未指定，则使用通常存在的 default 池。
* description：分配给动态创建的卷的描述。所有卷描述对于 storage class 都是相同的， 但不同的 storage class 可以使用不同的描述，以区分不同的使用场景。 默认为 Kubernetas volume。
* fsType：请求的默认文件系统类型。请注意，在 StorageOS 中用户定义的规则可以覆盖此值。默认为 ext4
* adminSecretNamespace：API 配置 secret 所在的命名空间。如果设置了 adminSecretName，则是必需的。
* adminSecretName：用于获取 StorageOS API 凭证的 secret 名称。如果未指定，则将尝试默认值。

StorageOS Kubernetes 卷插件可以使 Secret 对象来指定用于访问 StorageOS API 的端点和凭据。 只有当默认值已被更改时，这才是必须的。 secret 必须使用 kubernetes.io/storageos 类型创建，如以下命令：

kubectl create secret generic storageos-secret **\**

--type="kubernetes.io/storageos" **\**

--from-literal=apiAddress=tcp://localhost:5705 **\**

--from-literal=apiUsername=storageos **\**

--from-literal=apiPassword=storageos **\**

--namespace=default

用于动态分配卷的 Secret 可以在任何名称空间中创建，并通过 adminSecretNamespace 参数引用。 预先配置的卷使用的 Secret 必须在与引用它的 PVC 在相同的名称空间中。

### 本地

**FEATURE STATE:** Kubernetes v1.14 [stable]

**kind**: StorageClass

**apiVersion**: storage.k8s.io/v1

**metadata**:

**name**: local-storage

**provisioner**: kubernetes.io/no-provisioner

**volumeBindingMode**: WaitForFirstConsumer

本地卷还不支持动态分配，然而还是需要创建 StorageClass 以延迟卷绑定，直到完成 pod 的调度。这是由 WaitForFirstConsumer 卷绑定模式指定的。

延迟卷绑定使得调度器在为 PersistentVolumeClaim 选择一个合适的 PersistentVolume 时能考虑到所有 pod 的调度限制。

**3.实现nfs做存储类的动态供给**

**参考**<https://jimmysong.io/kubernetes-handbook/practice/using-nfs-for-persistent-storage.html>

（1）创建运行nfs-provisioner的sa账号

cat serviceaccount.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: nfs-provisioner

kubectl apply -f serviceaccount.yaml

（2）对sa账号做rbac授权

cat rbac.yaml

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: nfs-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"]

- apiGroups: [""]

resources: ["services", "endpoints"]

verbs: ["get"]

- apiGroups: ["extensions"]

resources: ["podsecuritypolicies"]

resourceNames: ["nfs-provisioner"]

verbs: ["use"]

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: run-nfs-provisioner

subjects:

- kind: ServiceAccount

name: nfs-provisioner

namespace: default

roleRef:

kind: ClusterRole

name: nfs-provisioner-runner

apiGroup: rbac.authorization.k8s.io

---

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: leader-locking-nfs-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-provisioner

subjects:

- kind: ServiceAccount

name: nfs-provisioner

namespace: default

roleRef:

kind: Role

name: leader-locking-nfs-provisioner

apiGroup: rbac.authorization.k8s.io

kubectl apply -f rbac.yaml

**（3）通过deployment创建pod用来运行nfs-provisioner程序（用来划分pv的程序）**

cat nfs-deployment.yaml

kind: Deployment

apiVersion: apps/v1

metadata:

name: nfs-provisioner

spec:

selector:

matchLabels:

app: nfs-provisioner

replicas: 1

strategy:

type: Recreate

template:

metadata:

labels:

app: nfs-provisioner

spec:

serviceAccount: nfs-provisioner

containers:

- name: nfs-provisioner

image: registry.cn-hangzhou.aliyuncs.com/open-ali/nfs-client-provisioner:latest

volumeMounts:

- name: nfs-client-root

mountPath: /persistentvolumes

env:

- name: PROVISIONER\_NAME

value: example.com/nfs

- name: NFS\_SERVER

value: 192.168.0.6

- name: NFS\_PATH

value: /data/nfs\_pro

volumes:

- name: nfs-client-root

nfs:

server: 192.168.0.6

path: /data/nfs\_pro

kubectl apply -f nfs-deployment.yaml

注：（1）、（2）、（3）这三个步骤是用来创建nfs外部供应商程序的，我们storageclass要想使用nfs作为外部供应者，必须执行这三个步骤

**（4）创建storageclass**

cat nfs-storageclass.yaml

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: nfs

provisioner: example.com/nfs

kubectl apply -f nfs-storageclass.yaml

显示如下，说明创建成功了：

NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE

nfs example.com/nfs Delete Immediate false 17m

（5）创建pvc

cat claim.yaml

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: test-claim1

spec:

accessModes: [“ReadWriteMany”]

resources:

requests:

storage: 1Gi

storageClassName: nfs

kubectl apply -f claim.yaml

（6）创建pod，使用storageclass动态生成pv

cat read-pod.yaml

kind: Pod

apiVersion: v1

metadata:

name: read-pod

spec:

containers:

- name: read-pod

image: nginx

volumeMounts:

- name: nfs-pvc

mountPath: /usr/share/nginx/html

restartPolicy: "Never"

volumes:

- name: nfs-pvc

persistentVolumeClaim:

claimName: test-claim1

kubectl apply -f read-pod.yaml

kubectl get pods

显示如下，说明pod创建成功

NAME READY STATUS RESTARTS AGE

nfs-provisioner-764b7db9c5-wptrk 1/1 Running 0 27m

（7）创建statefulset，动态生成存储

想要使用下面的volumeClaimTemplate，需要我上面的第（1）、（2）、（3）、（4）都部署成功才可以

cat statefulset-storage.yaml

apiVersion: v1

kind: Service

metadata:

name: storage

labels:

app: storage

spec:

ports:

- port: 80

name: web

clusterIP: None

selector:

app: storage

---

apiVersion: apps/v1beta1

kind: StatefulSet

metadata:

name: storage

spec:

serviceName: "storage"

replicas: 2

template:

metadata:

labels:

app: storage

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

name: web

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html

volumeClaimTemplates:

- metadata:

name: www

annotations:

volume.beta.kubernetes.io/storage-class: "nfs"

spec:

accessModes: [ "ReadWriteOnce" ]

resources:

requests:

storage: 2Gi

kubectl apply -f statefulset-storage.yaml

**验证，我自己的操作步骤：**

[root@master1 manifests]# kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 2d9h

my-nginx NodePort 10.110.201.60 <none> 80:32348/TCP 2d6h

my-nginx-headless ClusterIP None <none> 80/TCP 2d1h

my-service ClusterIP 10.108.54.235 <none> 80/TCP 2d2h

storage ClusterIP None <none> 80/TCP 2m4s

You have new mail in /var/spool/mail/root

[root@master1 manifests]# kubectl get storageclass

NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE

nfs example.com/nfs Delete Immediate false 53m

[root@master1 manifests]# kubectl get pods

NAME READY STATUS RESTARTS AGE

nfs-provisioner-764b7db9c5-wptrk 1/1 Running 0 58m

pod-taint 1/1 Running 0 24h

read-pod 1/1 Running 0 44m

storage-0 1/1 Running 0 32s

storage-1 0/1 ContainerCreating 0 13s

test-hostpath 2/2 Running 0 47h

test-nfs-volume 2/2 Running 0 46h

test-pod 1/1 Running 0 2d

[root@master1 manifests]# kubectl get pods

NAME READY STATUS RESTARTS AGE

nfs-provisioner-764b7db9c5-wptrk 1/1 Running 0 58m

pod-taint 1/1 Running 0 24h

read-pod 1/1 Running 0 44m

storage-0 1/1 Running 0 48s

storage-1 0/1 ContainerCreating 0 29s

test-hostpath 2/2 Running 0 47h

test-nfs-volume 2/2 Running 0 46h

test-pod 1/1 Running 0 2d

[root@master1 manifests]# kubectl get storageclass

NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE

nfs example.com/nfs Delete Immediate false 53m

[root@master1 manifests]# kubectl get pv

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE

pvc-0f4a099e-8614-4ad6-b48e-aaac3cbeec10 1G RWX Delete Bound default/test-claim1 nfs 49m

pvc-3e499430-3b6e-4dc2-90bd-9c3e69ccf6af 2Gi RWX Delete Bound default/www-storage-0 nfs 57s

pvc-c704f398-2d67-4ef4-b7b8-ea671b48fce1 2Gi RWX Delete Bound default/www-storage-1 nfs 33s

v1 1G RWO Retain Available 31h

v10 10G RWO,RWX Retain Available 31h

v2 2G RWX Retain Available 31h

v3 3G ROX Retain Available 31h

v4 4G RWO,RWX Retain Bound default/www-web-0 31h

v5 5G RWO,RWX Retain Bound default/www-web-1 31h

v6 6G RWO,RWX Retain Bound default/www-web-2 31h

v7 7G RWO,RWX Retain Available 31h

v8 8G RWO,RWX Retain Available 31h

v9 9G RWO,RWX Retain Available 31h

[root@master1 manifests]# kubectl get pvc

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

test-claim1 Bound pvc-0f4a099e-8614-4ad6-b48e-aaac3cbeec10 1G RWX nfs 50m

www-storage-0 Bound pvc-3e499430-3b6e-4dc2-90bd-9c3e69ccf6af 2Gi RWX nfs 77s

www-storage-1 Bound pvc-c704f398-2d67-4ef4-b7b8-ea671b48fce1 2Gi RWX nfs 58s

www-web-0 Bound v4 4G RWO,RWX 23h

www-web-1 Bound v5 5G RWO,RWX 23h

www-web-2 Bound v6 6G RWO,RWX 23h

You have new mail in /var/spool/mail/root

[root@master1 manifests]# vim st

statefulset-storage.yaml statefulset.yaml storageclass.yaml

[root@master1 manifests]# vim st

statefulset-storage.yaml statefulset.yaml storageclass.yaml

[root@master1 manifests]# vim statefulset-storage.yaml

You have new mail in /var/spool/mail/root

[root@master1 manifests]#

[root@master1 manifests]#

[root@master1 manifests]# kubectl get pv

NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORAGECLASS REASON AGE

pvc-0f4a099e-8614-4ad6-b48e-aaac3cbeec10 1G RWX Delete Bound default/test-claim1 nfs 52m

pvc-3e499430-3b6e-4dc2-90bd-9c3e69ccf6af 2Gi RWX Delete Bound default/www-storage-0 nfs 3m33s

pvc-c704f398-2d67-4ef4-b7b8-ea671b48fce1 2Gi RWX Delete Bound default/www-storage-1 nfs 3m9s

v1 1G RWO Retain Available 32h

v10 10G RWO,RWX Retain Available 32h

v2 2G RWX Retain Available 32h

v3 3G ROX Retain Available 32h

v4 4G RWO,RWX Retain Bound default/www-web-0 31h

v5 5G RWO,RWX Retain Bound default/www-web-1 32h

v6 6G RWO,RWX Retain Bound default/www-web-2 32h

v7 7G RWO,RWX Retain Available 32h

v8 8G RWO,RWX Retain Available 32h

v9 9G RWO,RWX Retain Available 32h

[root@master1 manifests]# kubectl get pvc

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

test-claim1 Bound pvc-0f4a099e-8614-4ad6-b48e-aaac3cbeec10 1G RWX nfs 52m

www-storage-0 Bound pvc-3e499430-3b6e-4dc2-90bd-9c3e69ccf6af 2Gi RWX nfs 3m54s

www-storage-1 Bound pvc-c704f398-2d67-4ef4-b7b8-ea671b48fce1 2Gi RWX nfs 3m35s

www-web-0 Bound v4 4G RWO,RWX 23h

www-web-1 Bound v5 5G RWO,RWX 23h

www-web-2 Bound v6 6G RWO,RWX 23h

[root@master1 manifests]# kubectl get pods

NAME READY STATUS RESTARTS AGE

nfs-provisioner-764b7db9c5-wptrk 1/1 Running 0 61m

pod-taint 1/1 Running 0 24h

read-pod 1/1 Running 0 47m

storage-0 1/1 Running 0 4m3s

storage-1 1/1 Running 0 3m44s

test-hostpath 2/2 Running 0 47h

test-nfs-volume 2/2 Running 0 46h

test-pod 1/1 Running 0 2d