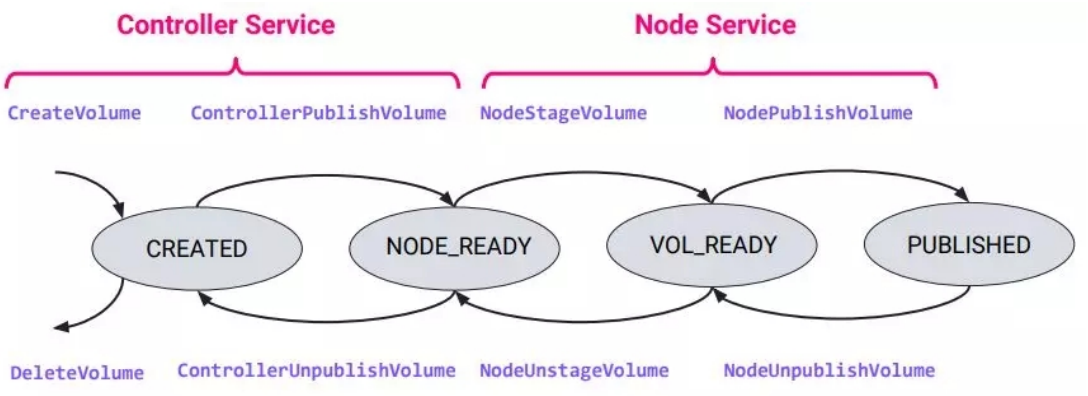
目录

CSI 设计思想

将 Pod 挂载 PV 扩展成 Provision、Attach 和 Mount 三个阶段。其中
Provision 等价于“创建磁盘”
Attach 等价于“挂载磁盘到虚拟机”
Mount 等价于“将该磁盘格式化后，挂载在 Volume 的宿主机目录上”

Volume 生命周期



CSI 设计方案

K8S 通过 gRPC 协议与 CSI 插件交互，每个 SP（存储供应商，实现 CSI 插件的）都必须提供两类插件：

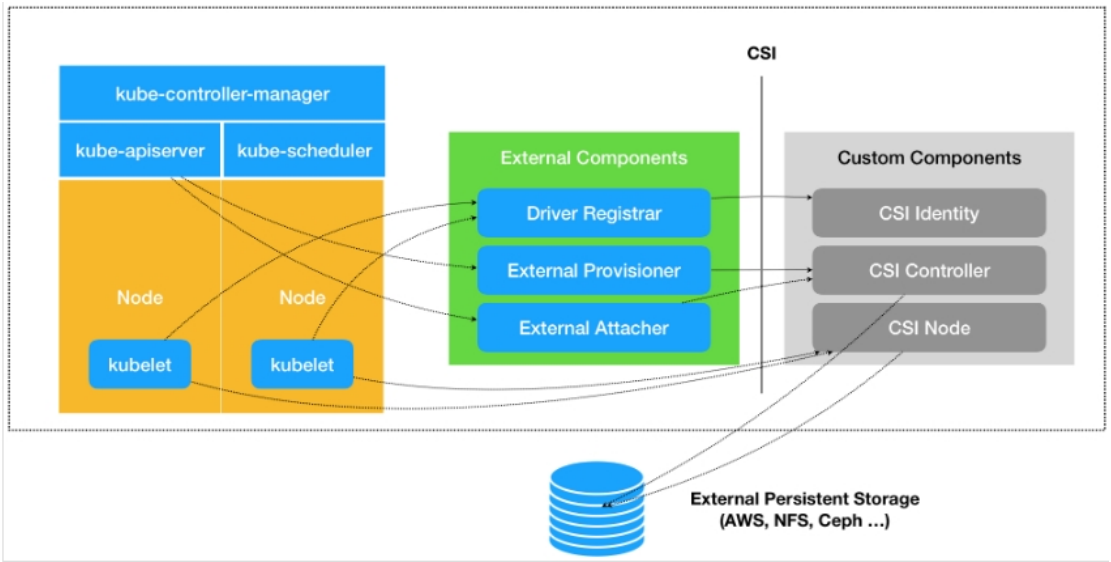
- Node plugin：在每个节点上运行,，作为一个 grpc 端点服务于 CSI 的 RPCs，执行具体的挂卷操作。
- Controller Plugin：可以在任何地方运行，一般执行全局性的操作，比如创建/删除卷。

CSI 有三种 RPC：

- 身份服务：Node Plugin 和 Controller Plugin 都必须实现这些 RPC 集。
- 控制器服务：Controller Plugin 必须实现这些 RPC 集。
- 节点服务：Node Plugin 必须实现这些 RPC 集。

为了方便管理，建议将 3 个服务部署在一个 Container 中，使用 DaemonSet 方式部署到所有存储节点

CSI 架构



External Components

是从 Kubernetes 项目里面剥离出来的那部分存储管理功能，由 Kubernetes 社区来开发和维护
Driver Registrar：用于向 kubelet 注册 Custom Components
External Provisioner ：用于创建 PV
External Attacher：用于将 PV attache 到节点

Custom Components（csi-driver）

3 个 RPC 服务，是由第三方负责开发和维护
Identity Service：负责对外暴露这个插件本身的信息
Controller Service：用于创建、删除以及管理 CSI Volume
Node Service：用于将 Volume 存储卷挂载到指定的目录中

CSI 流程总结

1. Driver Registrar 调用 Identity Service 的 GetPluginInfo 方法后向 kubelet 注册 csi-driver，便于后面 kubelet 调用
2. 用户创建 PVC，引用 storageClass，storageClass 所指向的 External Provisioner 监听到对应的 PVC 事件，开始调用 Controller Service 的 CreateVolume/DeleteVolume 方法完成 PV 的创建/删除。
3. Pod 使用该 PVC 和 PV，并被调度到 Node 节点

4. Node 节点上 Kubelet 看到该 Pod 使用了 PV，其 volume Manager（通过 CSI-Plugin）会创建一个 volumeattachments，并等待其 attached 状态变化
5. External Attacher 监听 volumeattachments，调用 Controller Service 的 ControllerPublishVolume/ ControllerUnpublishVolume 方法完成 Attach/Detach
6. Node 节点上 Kubelet 的 volume Manager 观察到 attached 状态变为 true 后，调用 Node Service 的 NodeStageVolume 和 NodePublishVolume 方法完成 Mount/Unmount

node-driver-registrar

https://github.com/kubernetes-csi/driver-registrar 已被废弃
https://github.com/kubernetes-csi/node-driver-registrar 使用的

作用

- 向 Kubelet 注册 CSI 驱动程序。因为 kubelet 调用 CSI 的（NodeGetInfo，NodeStageVolume，NodePublishVolume）等方法时需要知道向哪个套接字发出调用
- 将 CSI 驱动程序自定义 NodeId 添加到 Kubernetes Node API 对象上的标签。便于后面 ControllerPublishVolume 调用能够获取到 nodeid 与 csi-driver 的映射关系

逻辑

通过 CSI driver socket 调用 csi-driver/identity 的 GetPluginInfo 函数，获取 CSI 驱动的名称

通过 Registration socket，向 kubelet 注册 CSI 驱动程序

源码

```
cmd/csi-node-driver-registrar/main.go
131         csiDriverName, err := csirpc.GetDriverName(ctx, csiConn)

github.com/kubernetes-csi/csi-lib-utils/rpc/common.go
43         rsp, err := client.GetPluginInfo(ctx, &req)

cmd/csi-node-driver-registrar/node_register.go
37          registrar           :=          newRegistrationServer(csiDriverName,
*kubeletRegistrationPath, supportedVersions)
62          grpcServer := grpc.NewServer()
64          registerapi.RegisterRegistrationServer(grpcServer, registrar)

k8s.io/kubernetes/pkg/kubelet/apis/pluginregistration/v1alpha1/api.pb.go
214 func RegisterRegistrationServer(s *grpc.Server, srv RegistrationServer) {
```

```
215         s.RegisterService(&_Registration_serviceDesc, srv)
216 }
```

注：

打标签有两种模式，一个模式是自己给 node 打上这个 annotation，并且在退出的时候把这个 annotation 去掉。另一个模式是交给 kubelet 的 pluginswatcher 来管理，kubelet 自己会根据 node-driver-registrar 提供的 socket 然后调用 gRPC 从 registrar 获取 NodeId 和 DriverName 自己把 annotation 打上。

疑点：

打标签 annotation 的 csi.volume.kubernetes.io/nodeid 在哪里进行的？

external-provisioner

作用

根据用户的 PVC 请求创建/删除 PV，完成 Provision 和 Delete。

逻辑

用户创建 PVC 引用 storageclass，storageclass 的 provisioner 属性会让 kubernetes 在创建 PVC 时指定 annotations（volume.beta.kubernetes.io/storage-provisioner）。

provisioner 监听 Kube-API 中的 PVC 对象，执行其 Provision 函数，先后**调用 csi-driver/identity 的 GetPluginCapabilities、csi-driver/controller 的 ControllerGetCapabilities、csi-driver/identity 的 GetPluginInfo**，获取 csi-driver 的相关信息，再根据 PVC 和 Storageclass 的属性组成 CreateVolumeRequest 对象，传递并**调用 csi-driver/controller 的 CreateVolume/DeleteVolume** 函数返回 CreateVolumeResponse 对象，根据返回信息组成 pv 并返回，然后 kube-apiserver 中的 VolumeController 的 PersistentVolumeController 进行创建和绑定 PVC。

源码

```
github.com/kubernetes-incubator/external-storage/lib/controller/controller.go
1015         volume, err = ctrl.provisioner.Provision(options)
1042                                                 if _, err = ctrl.client.CoreV1().PersistentVolumes().Create(volume); err == nil || apierrs.IsAlreadyExists(err) {

external-provisioner/pkg/controller/controller.go
169         rsp, err := client.GetPluginInfo(ctx, &req)
```

```
232         rsp, err := client.GetPluginCapabilities(ctx, &req)
246         rsp, err := client.ControllerGetCapabilities(ctx, &req)
317 func makeVolumeName(prefix, pvcUID string, volumeNameUUIDLength int)
(string, error) {
354             driverState, err := checkDriverState(p.grpcClient, p.timeout,
needSnapshotSupport)
481                 rep, err = p.csiClient.CreateVolume(ctx, &req)
}
651         _, err = p.csiClient.DeleteVolume(ctx, &req)
```

注：可以部署多个 provisioner，但只能有一个 provisioner 领导者。可以在启动时指定--enable-leader-election 开启选举。
--volume-name-prefix 参数可以指定 PV 的名称前缀（默认 pvc-<uuid>）

Volume-Mnager

作用

调用 in-tree 的 CSI-Plugin 创建 VolumeAttachment，并使用 WaitForAttach 等待其状态变为 true 后再通过 in-tree 的 CSI-Plugin 调用 csi-driver/node 的 NodeStageVolume 和 NodePublishVolume 方法完成 Mount

逻辑

kubelet 有一个 volume manager 来管理 volume 的 mount/attach 操作。
desiredStateOfWorld：是从 podManager 同步的理想状态。
actualStateOfWorld：是目前 kubelet 的上运行的 pod 的状态。
每次 volume manager 需要把 actualStateOfWorld 中 volume 的状态同步到 desired 指定的状态。
volume Manager 有两个 goroutine ，一个是同步状态，一个 reconciler.reconcile。
reconcile 方法先后分别调用 in-tree 的 CSI plugin 的 Attach、WaitForAttach，当 WaitForAttach 满足后调用 in-tree 的 CSI plugin 的 MountDevice 和 SetUp 方法

源码

github.com/kubernetes/pkg/kubelet/volumemanager/volume_manager.go

```
240         go vm.desiredStateOfWorldPopulator.Run(sourcesReady, stopCh)
244         go vm.reconciler.Run(stopCh)
```

github.com/kubernetes/pkg/kubelet/volumemanager/reconciler/reconciler.go

```
156 func (rc *reconciler) reconcile() {
```

```
214                                     err := rc.operationExecutor.AttachVolume(volumeToAttach,
rc.actualStateOfWorld)
234                                 err := rc.operationExecutor.MountVolume(
235                                         rc.waitForAttachTimeout,
236                                         volumeToMount.VolumeToMount,
237                                         rc.actualStateOfWorld,
238                                         isRemount)

github.com/kubernetes/pkg/volume/util/operationexecutor/operation_executor.go
598 func (oe *operationExecutor) AttachVolume(
602                             oe.operationGenerator.GenerateAttachVolumeFunc(volumeToAttach,
actualStateOfWorld)
721 func (oe *operationExecutor) MountVolume(
731             if fsVolume {
734                     generatedOperations = oe.operationGenerator.GenerateMountVolumeFunc(
735                                     waitForAttachTimeout, volumeToMount, actualStateOfWorld,
isRemount)
736
737             } else {
740                                                             generatedOperations,    err    =
oe.operationGenerator.GenerateMapVolumeFunc(
741                                 waitForAttachTimeout, volumeToMount, actualStateOfWorld)
742             }

github.com/kubernetes/pkg/volume/util/operationexecutor/operation_generator.go
294 func (og *operationGenerator) GenerateAttachVolumeFunc(
348                         devicePath, attachErr := volumeAttacher.Attach(
349                                 volumeToAttach.VolumeSpec, volumeToAttach.NodeName)
520 func (og *operationGenerator) GenerateMountVolumeFunc(
595                                 devicePath, err = volumeAttacher.WaitForAttach(
620                                 err = volumeDeviceMounter.MountDevice(
662                         mountErr := volumeMounter.SetUp(fsGroup)
```

external-attacher

作用

将 PV 附加/分离 Node 节点，完成 Attach 和 Detach

这在云环境中很常见，在云环境中，云 API 能够将卷附加到节点上，而不需要在节点上运行任何代码。

逻辑

用户创建的 Pod 在调度到 Node 节点之后，kubelet 会创建一个 VolumeAttachment 资源

Attacher 监听 Kube-API 中的 volumeattachments 对象，先判断 csi-driver 是否支持 ControllerPublishVolume 功能，使用两种不同的 handler，调用 SyncNewOrUpdatedVolumeAttachment ，最终将 volumeattachment 的 attached 置为 true。

- CSIHandler：先**调用 csi-driver/controller 的 ControllerPublishVolume/ ControllerUnpublishVolume** 函数，再调用 markAsAttached，将 VolumeAttachment. Status. Attached 重置为 true
- TrivialHandler：直接调用 markAsAttached，将 VolumeAttachment. Status. Attached 重置为 true

源码

```
external-attacher/ cmd/csi-attacher/main.go
130         supportsService, err := csiConn.SupportsPluginControllerService(ctx)
140             supportsAttach, err := csiConn.SupportsControllerPublish(ctx)
151             handler  =  controller.NewCSIHandler(clientset,  csiClientset,
attacher, csiConn, pvLister, nodeLister, nodeInfoLister, vaLister, timeout)
154             handler = controller.NewTrivialHandler(clientset)

pkg/controller/csi_handler.go
90      func      (h      *csiHandler)      SyncNewOrUpdatedVolumeAttachment(va
*storage.VolumeAttachment) {
119         va, metadata, err := h.csiAttach(va)
134         if _, err := markAsAttached(h.client, va, metadata); err != nil {
321           publishInfo, _, err := h.csiConnection.Attach(ctx, volumeHandle,
readOnly, nodeID, volumeCapabilities, attributes, secrets)

pkg/connection/connection.go
195 func (c *csiConnection) Attach(ctx context.Context, volumeID string, readOnly
bool,    nodeID    string,    caps    *csi.VolumeCapability,    context,    secrets
map[string]string) (metadata map[string]string, detached bool, err error) {
207         rsp, err := client.ControllerPublishVolume(ctx, &req)
223         _, err = client.ControllerUnpublishVolume(ctx, &req)

pkg/controller/trivial_handler.go
47      func      (h      *trivialHandler)      SyncNewOrUpdatedVolumeAttachment(va
*storage.VolumeAttachment) {
51                  if _, err := markAsAttached(h.client, va, nil); err != nil {
```

注：
1：可以部署多个 attacher，但只能有一个 attacher 领导者。可以在启动时指定--leader-election 开启选举
2：文件存储就不需要 attache，因为不需要绑定设备到节点上，直接使用网络接口就可以了

CSI-Plugin（In-tree）

作用

真正调用 csi-driver 的方法，完成 Volume 的 mount/unmount。

逻辑

1：Attach 方法创建一个 volumeattachment 资源
2：WaitForAttach 方法等待 volumeattachment 的 attached 状态变化
3：MountDevice 方法调用 csi-driver/node 的 NodeStageVolume 方法完成临时挂载
4：SetUp 方法调用 csi-driver/node 的 NodePublishVolume 方法完成挂载

源码

```
github.com/kubernetes/pkg/volume/csi/csi_attacher.go
60 func (c *csiAttacher) Attach(spec *volume.Spec, nodeName types.NodeName) (string, error) {
76              attachment := &storage.VolumeAttachment{
77                      ObjectMeta: meta.ObjectMeta{
78                              Name: attachID,
79                      },
80                      Spec: storage.VolumeAttachmentSpec{
81                              NodeName: node,
82                              Attacher: pvSrc.Driver,
83                              Source: storage.VolumeAttachmentSource{
84                                      PersistentVolumeName: &pvName,
85                              },
86                      },
87              }
88
89              _, err = c.k8s.StorageV1().VolumeAttachments().Create(attachment)
269 func (c *csiAttacher) MountDevice(spec *volume.Spec, devicePath string, deviceMountPath string) (err error) {
370              err = csi.NodeStageVolume(ctx,
371                      csiSource.VolumeHandle,
```

```
372                     publishContext,
373                     deviceMountPath,
374                     fsType,
375                     accessMode,
376                     nodeStageSecrets,
377                     csiSource.VolumeAttributes)
```

github.com/kubernetes/pkg/volume/csi/csi_mounter.go

```
 95 func (c *csiMountMgr) SetUp(fsGroup *int64) error {
 96         return c.SetUpAt(c.GetPath(), fsGroup)
 97 }
 98
 99 func (c *csiMountMgr) SetUpAt(dir string, fsGroup *int64) error {
243         err = csi.NodePublishVolume(
244                     ctx,
245                     volumeHandle,
246                     readOnly,
247                     deviceMountPath,
248                     dir,
249                     accessMode,
250                     publishContext,
251                     volAttribs,
252                     nodePublishSecrets,
253                     fsType,
254                     mountOptions,
255         )
```

注：

文件系统类型的存储（NFS/GlusterFS 等），只需要 NodePublishVolume 一步即可

csi-driver

★★★

https://github.com/container-storage-interface/spec/blob/master/spec.md#rpc-interface

需要定义三个 service（RPC 集合）：identity、controller、node
根据不同的 Capability 实现其对应的接口功能，然后其他程序调用 identity/GetPluginCapabilities 就可以知道该 csi-driver 能够做什么了

公共部分

https://github.com/kubernetes-csi/drivers/tree/master/pkg/csi-common

k8s 实现了一个官方的公共代码，公共代码实现了 CSI 要求的 RPC 方法，我们自己开发的插件可以继承官方的公共代码，然后把自己要实现的部分方法进行覆盖即可

identity

GetPluginInfo
Probe
GetPluginCapabilities

controller

CreateVolume
DeleteVolume
ControllerPublishVolume
ControllerUnpublishVolume
ValidateVolumeCapabilities
ListVolumes
GetCapacity
ControllerGetCapabilities
CreateSnapshot
DeleteSnapshot
ListSnapshots

node

NodePublishVolume
NodeUnpublishVolume
NodeGetInfo
NodeGetCapabilities
NodeGetVolumeStats

能力列表

github.com/container-storage-interface/spec/lib/go/csi/csi.pb.go

UNKNOWN

CONTROLLER_SERVICE

ControllerServiceCapability_RPC_UNKNOWN

ControllerServiceCapability_RPC_CREATE_DELETE_VOLUME
ControllerServiceCapability_RPC_PUBLISH_UNPUBLISH_VOLUME
ControllerServiceCapability_RPC_LIST_VOLUMES
ControllerServiceCapability_RPC_GET_CAPACITY
ControllerServiceCapability_RPC_CREATE_DELETE_SNAPSHOT
ControllerServiceCapability_RPC_LIST_SNAPSHOTS
ControllerServiceCapability_RPC_CLONE_VOLUME
ControllerServiceCapability_RPC_PUBLISH_READONLY
ControllerServiceCapability_RPC_EXPAND_VOLUME

VOLUME_ACCESSIBILITY_CONSTRAINTS

VolumeCapability_AccessMode_UNKNOWN
VolumeCapability_AccessMode_SINGLE_NODE_WRITER
VolumeCapability_AccessMode_SINGLE_NODE_READER_ONLY
VolumeCapability_AccessMode_MULTI_NODE_READER_ONLY
VolumeCapability_AccessMode_MULTI_NODE_SINGLE_WRITER
VolumeCapability_AccessMode_MULTI_NODE_MULTI_WRITER

NODE_SERVICE

NodeServiceCapability_RPC_UNKNOWN
NodeServiceCapability_RPC_STAGE_UNSTAGE_VOLUME
NodeServiceCapability_RPC_GET_VOLUME_STATS
NodeServiceCapability_RPC_EXPAND_VOLUME
但好像暂时不支持 NodeServiceCapability 的增加

增加能力

```
lvm.driver.AddControllerServiceCapabilities([]csi.ControllerServiceCapability_RPC_Type{csi.ControllerServiceCapability_RPC_CREATE_DELETE_VOLUME})
lvm.driver.AddVolumeCapabilityAccessModes([]csi.VolumeCapability_AccessMode_Mode{csi.VolumeCapability_AccessMode_SINGLE_NODE_WRITER})
```

注：添加能力后就必须实现对应的接口功能