Internals of Docking Storage with Kubernetes Workloads

Dennis Chen
Staff Software Engineer
Agenda

• Background
• What’s CSI
• CSI vs FlexVolume
• How CSI works
• FlexVolume Driver Part
• CSI Driver Part
Background

1. Kubernetes has supported a long list of volume types such as:
   - awsElasticBlockStore
   - fc(fibre channel)
   - scaleIO
   - list to be continued...

   Those are so-called `In-tree` volume plugins.

2. Even k8s has do a lot for you, but sometimes you still need to write a new one.

   In this case, FlexVolume and CSI can help you well 😊 which is also the focus of our today’s topic: Out-of-Tree volume plugin interface.
Background

1. **In-tree Volume Plugins**
   - Those are linked, compiled, built and shipped with the core k8s binaries
   - Development is tightly coupled and dependent on k8s releases
   - Bugs in volume plugin can crash critical k8s components, instead of just the plugin
     - Will not be accepted since k8s 1.8

2. **Out-of-Tree Volume Plugins** (customized plugins by storage providers)
   - FlexVolume driver
   - CSI driver (*)
What’s CSI

• Container Storage Interface (CSI) is a standardized mechanism for Container Orchestration Systems (COs), including Kubernetes, to expose arbitrary storage systems to containerized workloads. Storage Provider (SP) develops once and this works across a number of COs.

• The goal of CSI is to become the primary volume plugin system for k8s in the future.

• k8s 1.9 release has already included the alpha feature of CSI implementation, then beta in Kubernetes v1.10

• The CSI spec can be found at:
  
  https://github.com/container-storage-interface/spec/blob/master/spec.md
CSI vs FlexVolume

Two Out-of-Tree Volume Plugin mechanisms in K8s – FlexVolume and CSI

1. FlexVolume plugin framework:
   - Makes the 3rd party storage providers’ plugin as “Out-of-Tree” (same as CSI does)
   - exec based API for external volume plugins
   - Needs to access the root filesystem of node and master machines when deploying
   - Doesn’t address the pain point of dependencies.

2. CSI overcomes the limitations of FlexVolume listed above. CSI is the preferred solution, for now CSI and FlexVolume can co-exist.
How CSI works

- A new in-tree CSI Volume plugin (K8s) + out-of-tree CSI Volume driver (3rd party)
- Communication channel via a Unix Domain Socket (UDS) created by 3rd Volume Driver

The socket file also called a ‘EndPoint’ in form of like:
/var/lib/kubelet/plugins/rook-ceph/csi.sock
How CSI works

Recommended Mechanism for Deploying CSI Drivers on k8s

https://github.com/kubernetes/community/blob/master/contributors/design-proposals/storage/container-storage-interface.md
A CSI deployment in real world

A diagram showing a Kubernetes cluster with multiple nodes and pods, including a driver, registrar, provisioner pod, attacher pod, and UDS. The diagram illustrates the architecture of a CSI deployment in a real-world scenario.
FlexVolume Driver Part (Take Rook as an example)
FlexVolume Driver -- rookflex

• `rookflex` exists in form of a binary file and has been deployed into `volume-plugin-dir` by Rook Agent on each node.

• `rookflex` implements ‘mount’ and ‘umount’ methods required by FlexVolume Spec

• For a specific YAML file of a workload, the storage related part looks like:

```yaml
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-block
provisioner: ceph.rock.io/block
defaults:
  parameters:
    pool: replicapool
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
spec:
  storageClassName: rook-ceph-block
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
```
A practical FlexVolume driver -- rookflex

- When that workload pod is scheduled to one node and begin to run, the kubelet will interacts with the driver to mount the volume into the `mountPath` specified by the YAML. To do so, kubelet needs to:

1. Lookup the right FlexVolume driver.

   The look up flow is: PVC name $\rightarrow$ StorageClass $\rightarrow$ provisioner name: ceph.rook.io/block $\rightarrow$ Flex volume vendor name: "ceph.rook.io“ $\rightarrow$ figure out the driver folder and driver name: rookflex

2. Call `mount` method of rookflex like: `$(volume-plugin-dir)/rookflex mount`

3. The above `mount` will call the corresponding function in Rook Agent via UDS.

4. Local Rook Agent will attach the volume into its node(a ‘rbd map’ operation).
Flexvolume-based volume operations

1. Provisioning part.
`rbd create` a volume in Ceph cluster.

2. Attach and Mount part.
`rbd map` the volume to a specified node as a block device then mount to the dir path in workload pod.
CSI Volume Driver Part
CSI: Zoom into the volume driver

Identity Routines
- GetPluginInfo()
- GetPluginCapabilities()
- Probe()

Controller Routines
- CreateVolume()
- DeleteVolume()
- ControllerPublishVolume()

Node Routines
- NodePublishVolume()
- NodeUnpublishVolume()
- ControllerPublishVolume()

3rd party Volume Driver

UDS

Identity Service

Node Service

Controller Service

Sidecar Containers
- driver registrar
- external-provisioner
- external-attacher

API Server
CSI: external-provisioner

1. A cluster admin creates a StorageClass pointing to the CSI driver’s external-provisioner.

2. A user creates a PersistantVolumeClaim referring to the new StorageClasses.

3. The persistent volume controller realizes that dynamic provisioning is needed.

4. The external-provisioner for the CSI driver sees the PersistentVolumeClaim so it stats dynamic volume provisioning:
   - It deferences the StorageClass to collect the opaque parameters to use for provisioning.
   - It calls CreateVolume() against the CSI driver container with parameters from the StorageClass and PersistentVolumeClaim objects.

5. Once the volume is successfully created, the external-provisioner creates a PersistentVolume object to represent the newly create volume and binds it to the PersistentVolumeClaim.
CSI: external-attacher

1. k8s attach/detach controller sees that a pod referencing a CSI volume plugin is scheduled to a node → call in-tree volume plugin’s attach()
2. The in-tree volume plugin creates a new `VolumeAttachment` object in the k8s API
3. The external-attacher sees the `VolumeAttachment` object and triggers a `ControllerPublish` again the CSI volume driver to fulfil it.
Ceph-CSI based volume operations

1. external-provisioner watches **PersistentVolumeClaim** objects and triggers Create/DeleteVolume against CSI volume driver.

2. external-attacher watches **VolumeAttachment** objects and triggers ControllerPublish/Unpublish against a CSI volume driver.
Thank You!
Danke!
Merci!
谢谢!
ありがとうございます!
Gracias!
Kiitos!