YARN-881: Support Container Storage Interface(CSI) in YARN

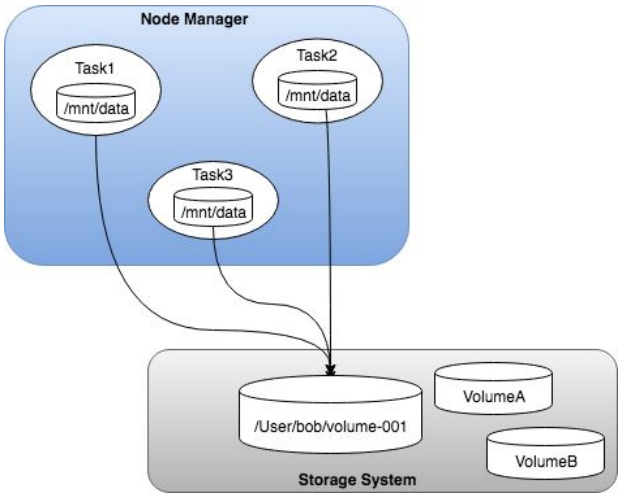
Author: Weiwei Yang, Sunil Govindan

1. Introduction

The Container Storage Interface (CSI) is a vendor neutral interface to bridge Container Orchestrators (CO) and Storage Providers (SP). COs who adopt CSI can leverage any storage provider, cloud or otherwise. Likewise, CSI enables the SPs to provide storage services to any CO that implements the specification.

Once YARN supports CSI, it will be simple to access various of storage systems in YARN docker containers as long as they are compatible with CSI. With this feature, it enables YARN to fit following scenarios:

- A general solution to tie to external storage systems
- Pre-populated some data in persistent storage for durable access
- Dynamically provision persistent volumes for containers and maintain task states in the storage, so the state can be retained upon restart



CSI defines the protocol of 3 gRPC services: controller, node and identity. SP just needs to implement a driver binary according to these protocols and then COs are able to consume these APIs to access SP's storage services. The detail of the specification can be found here. From high level,

- **Controller**: responsible of controlling and managing the volumes, such as: create, delete, attach/detach, snapshot, etc. This plugin can run anywhere but just single copy.
- **Node**: it formats and mount volumes to the underneath storage. This plugin needs to run on the node where the volume will be provisioned.

- **Identity**: allows a CO to query a plugin for capacities, health, and other metadata. This plugin needs to run with both controller and node plugins.

In this proposal, we introduce a solution for YARN to incorporate these services in response to those requests that ask for persisted volumes. We propose to make this two phase efforts, phase 1 focus on implementing YARN internal modules and protocols to work with CSI, only support pre-provisioned volume in order to simplify the volume lifecycle management; phase 2 we extend to support dynamical provisioned volumes and add security model.

2. Phase-1 Targets

Phase 1 targets are outlined as below

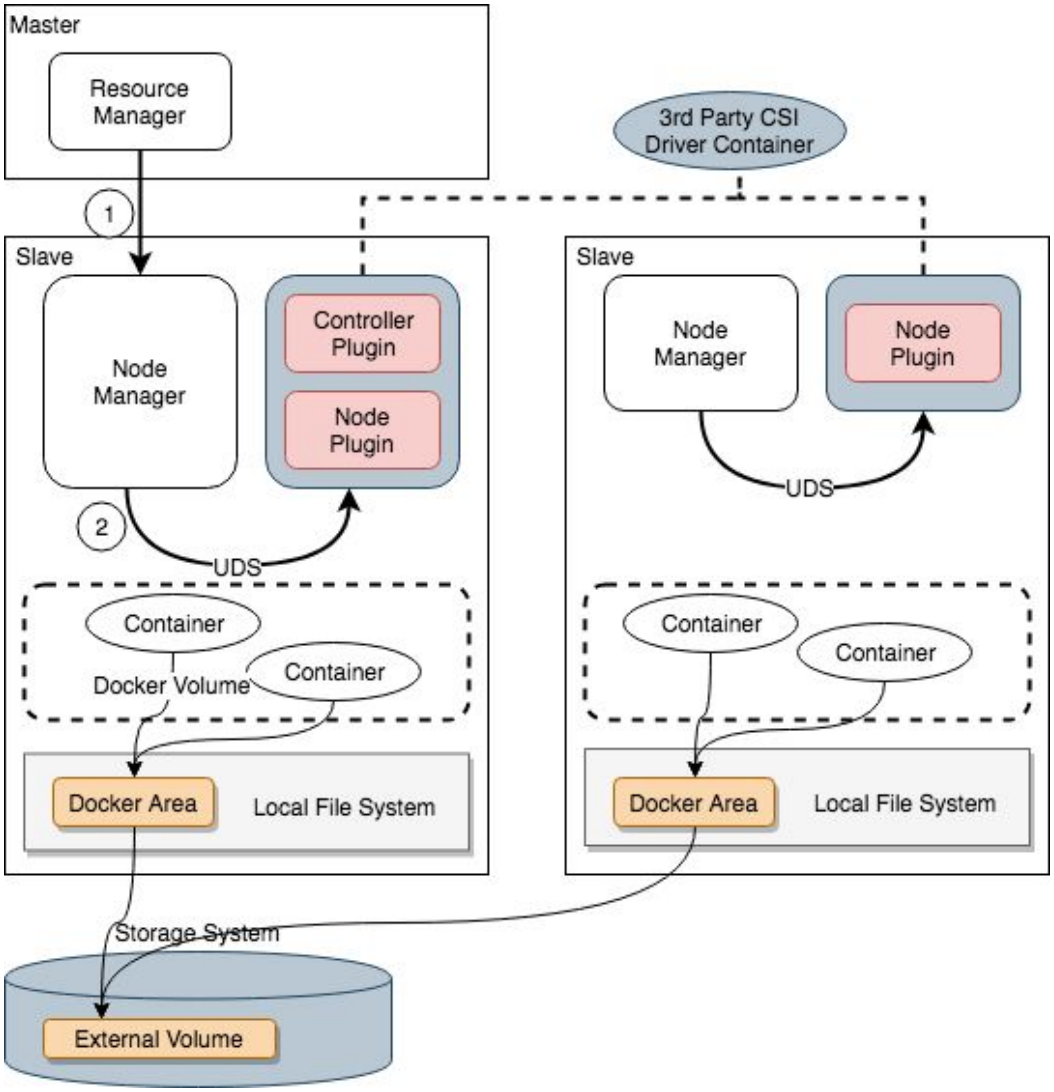
1. Define and implement YARN user level API for CSI volumes
2. Framework support for YARN to interact with CSI driver
3. Support to bind pre-provisioned CSI volumes[1] to docker containers
4. Provide a end-to-end native service example with volume attached

In phase 2, we will further explore how to support dynamical provisioned volumes and how to enhance the security model with external storage systems.

3. Proposal Overview

With CSI abstraction, we are able to add a clean interface in YARN to support external volumes, YARN exposes CSI volume in its public API for user to request (see more in CSI Volume). When process such requests, YARN internally translates them to corresponding CSI API calls, route them to CSI driver and process storage related operations. In YARN, we don't need to maintain any code directly related to a storage vendor, we only communicate with CSI driver using the common interface. An overview of the proposal can be illustrated as below,

[1] There are two volume provisioning pattern in CSI spec, 1) pre-provisioned; 2) dynamically provisioned. See more in *Volume Lifecycle* section of the CSI spec doc.



On every NM, there is one and only one csi-driver-container deployed[2], NM sets up a communication channel with the driver running on the same host, via unix domain socket. In NM process, there will be an CSI driver adaptor delegates all calls between YARN and a CSI driver, including both controller/node/identity API calls. Such adaptor needs to register with RM in order to tell RM which node runs the controller plugin. A typical workflow is,

1) App requests for some containers, and each container needs to mount a directory "/mnt/data" to a pre-provisioned volume "volume-001"
2) RM intercepts the volume info from the AM requests, for each volume request, RM calls the controller plugin (via NM adaptor) to ensure the volume is properly provisioned
3) Scheduler places the containers to certain NMs
4) AM asks NM to launch these containers

[2] How to deploy and manage CSI drivers will be discussed in a separate JIRA. In this proposal, we assume the driver containers are deployed on the cluster and endpoints are available for access.

5) Before NM launch a container, NM calls the node plugin to mount the given volume "volume-001" to a specific mount point on NM's local file system
6) NM launches the container and bind the local mount point to the docker container on "/mnt/data" via docker volume.
7) Container finished execution, NM disables the mount and properly handles the cleanup of the local mounts as well as the volume

4. The Volume Resource

We propose to add **Volume** as a resource in YARN. A volume represents a storage location with a certain capability that can be accessed by YARN containers, its life cycle is decoupled from container and application. Which means it won't be erased when a container or application is finished (this is very similar to the persistent volumes in k8s). A volume can be mounted to a container on a specific mount path, and containers are accessing the volume via this mount path. Volume resource can be provided by local disk or block device, or from an external storage system.

For the initial version, we limit our scope within:

- Only to support volumes from the storage system that compatible with CSI specification
- Only to support mount volume[3] (a volume mounted with file system and appears as a directory inside the container)
- Only to support volumes that resource is not managed by YARN

4.1 Sample Volume Resource Information

A sample of volume resource information is like below:

```
yarn.io/volume {
  size : 100,
  unit: Gi,
  ResourceType: ignorable,
  Attributes : {
    driver: "vendor-csi-driver",
    mountPoint: "/mnt/amt",
    accessMode: "readonly",
    volumeId: "vol-01" // preprovisioned
```

[3] This is majorly because we concern about the testing effort, the design doesn't limit us only to support mount volumes. It won't have much more work to support block volumes, perhaps no changes at all. How to consume the block device will be handled inside of the app's container.

```
  }
}
```

Upon spec defines a volume resource, that connects to an external storage system through "vendor-csi-driver", that mounts a pre-provisioned volume to a local dir "/mnt/amt" inside of the docker container. The spec contains two new concept, "**IGNORABLE**" resource type and "**Attributes**", we'll explain this in 4.3 and 4.4.

4.3 IGNORABLE Resource Type

To support volume resource, more specifically their resource is not managed by YARN for the 1st phase. Resources such volumes consumed are managed by a 3rd party storage system, not directly by YARN. So their resource will not be participated into the scheduling process like other resources, e.g memory and cpu, when the scheduler calculates resource shares, capacity and limits. They will be intercepted by a volume manager module and handled before entering the scheduling process. Therefore, we propose to add a new resource type **IGNORABLE**.

For such resource type, NM doesn't need to collect resource info and report back to RM, such resource will be ignored during scheduling phase, it is handled by a customized pluggable AMS processor, see more in Deep Dive section.

4.4 Resource Attributes

Currently a resource in YARN consists following information:

- Name (string)
- Units (string)
- ResourceType (COUNTABLE)
- Value (long)
- MinAllocation (long)
- MaxAllocation (long)

This describes a single dimensional resource with single countable value. However for a volume resource, this is not enough, it needs to carry more info such as volume name, access mode, mount path etc. Therefore, we propose to add optional "**Attributes**" field

- Attributes (string-string-map)

The attributes can be used to carry all required info for a resource provider (usually an external resource provider like a storage system) in order to make the resource available for consumption.

5. Deep Dive

5.1 RM Side

According to the deployment architecture, we need to develop some internal components in YARN to handle ResourceRequest/SchedulingRequest that carried volume resource, manage the lifecycle of volumes, local mounts, and translate these operations according to CSI protocol.

- CSI Volume Processor

  An application master processor that implemented for CSI, it intercepts the csi-volume spec from resource/scheduling requests and properly handles the request, then forward the requests to next processor in the chain.

- CSI Volume Manager

  The CSI volume manager reads the csi-volume spec and handles volume operations by connecting to the controller service on a csi-driver-container. This module manages the lifecycle of CSI volumes in YARN. Note, if there are multiple csi-driver instances running on the cluster, we need same number instance of volume managers to co-work with them.

- CSI Driver Client

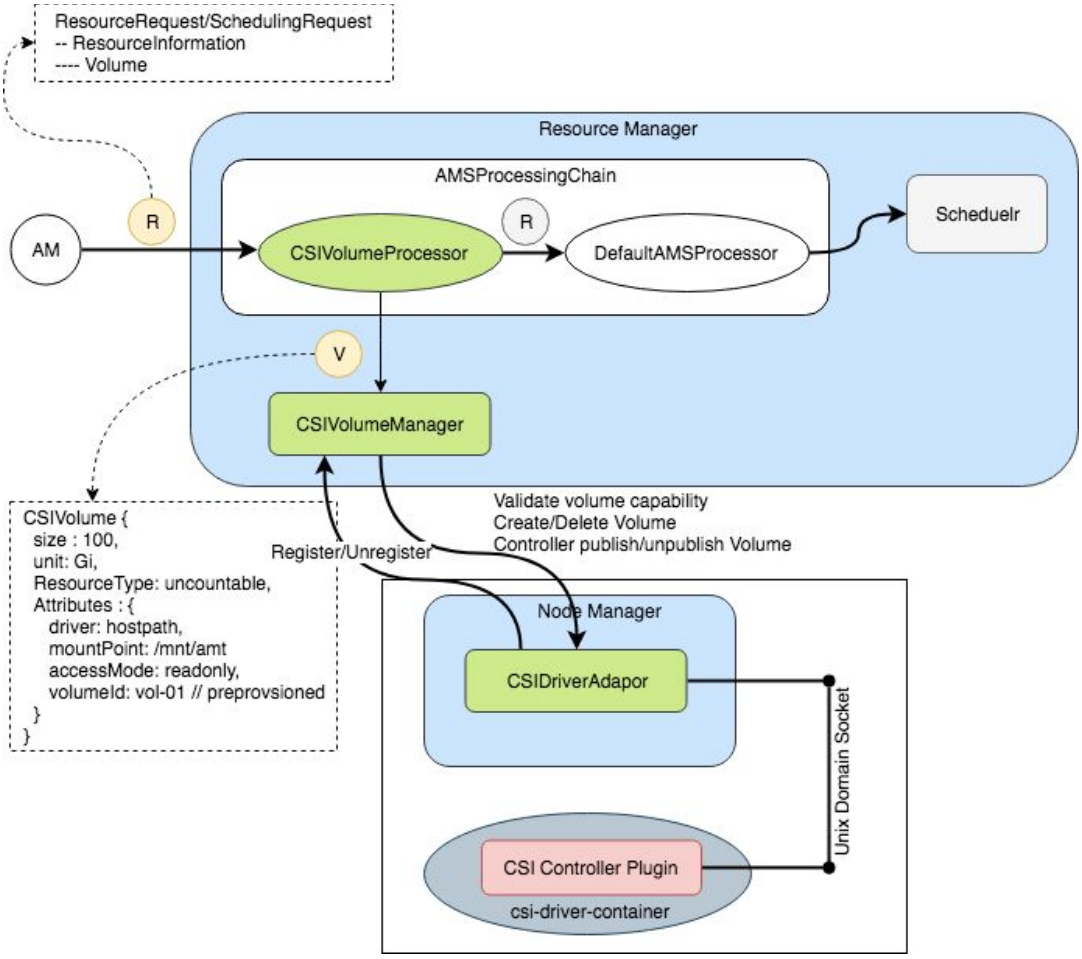
  A client library to connect to a certain CSI driver endpoint to handle CSI related operations, this client should be pluggable[4].

- CSI Driver Adaptor

  The CSI driver adaptor runs on all node managers, side by side with csi-driver-container. It delegates all invocations between YARN and CSI driver through Unix Domain Socket on the node manager host. The adaptor also needs to register itself to the volume manager and one of them will be the primary talking to the controller plugin.

A more detail view is like following:

[4] We can leverage existing CSI client library from gocsi project, but we need to develop our own client library for production.



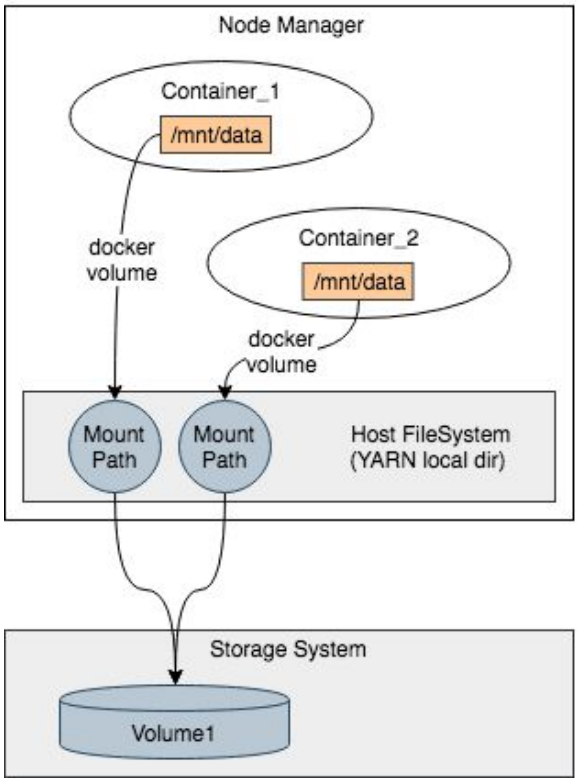
Since this is added in the main AM-RM allocate workflow, it's necessary to make the handling of volume operations in CSIVolumeManager to use *async manner*, in order not to block the entire process.

5.2 NM Side

Before we launch a container, in order to make a volume visible and accessible for a docker container, it must be properly formatted and mounted to a directory. Per CSI spec, this is handled by the node-service of the csi-driver. Related API calls including

- NodeStageVolume/Unstage
- NodePublishVolume/Unpublish

When NM starts the container, it firstly extracts the CSI-Volume info from resource, and call the csi-driver on the same node (via UDS) to mount the volume to a *container-csi-mount* directory under YARN local directory, then we mount this *container-csi-mount* to the container in path where user specified in the volume request.




```
Container_1
   yarn.io/csi-volume : {
      value : "10",
      unit : "Gi",
      attributes: {
          volumeName : "test-volume",
          driverName: "hostpath",
          mountPath: "/mnt/data"
      }
   }
```

```
Container_2
   yarn.io/csi-volume : {
      value : "5",
      unit : "Gi",
      attributes: {
          volumeName : "test-volume",
          driverName: "hostpath",
          mountPath: "/mnt/data"
      }
   }
```

Note, depending on the capability of different storage systems, it may not allow a volume to be mounted onto multiple places, or it allows multiple mounts but doesn't allow concurrency writes. YARN needs to do sanity checks to make sure the volume would not be abused.

When container is finished, the local mount on NM host needs to be disabled and removed. If container is crashed or killed, the cleanup also needs to be handled.