Container Storage Interface (CSI) Present and Future

Jie Yu (jie@mesosphere.com)

MESOSPHERE

2018

About Me

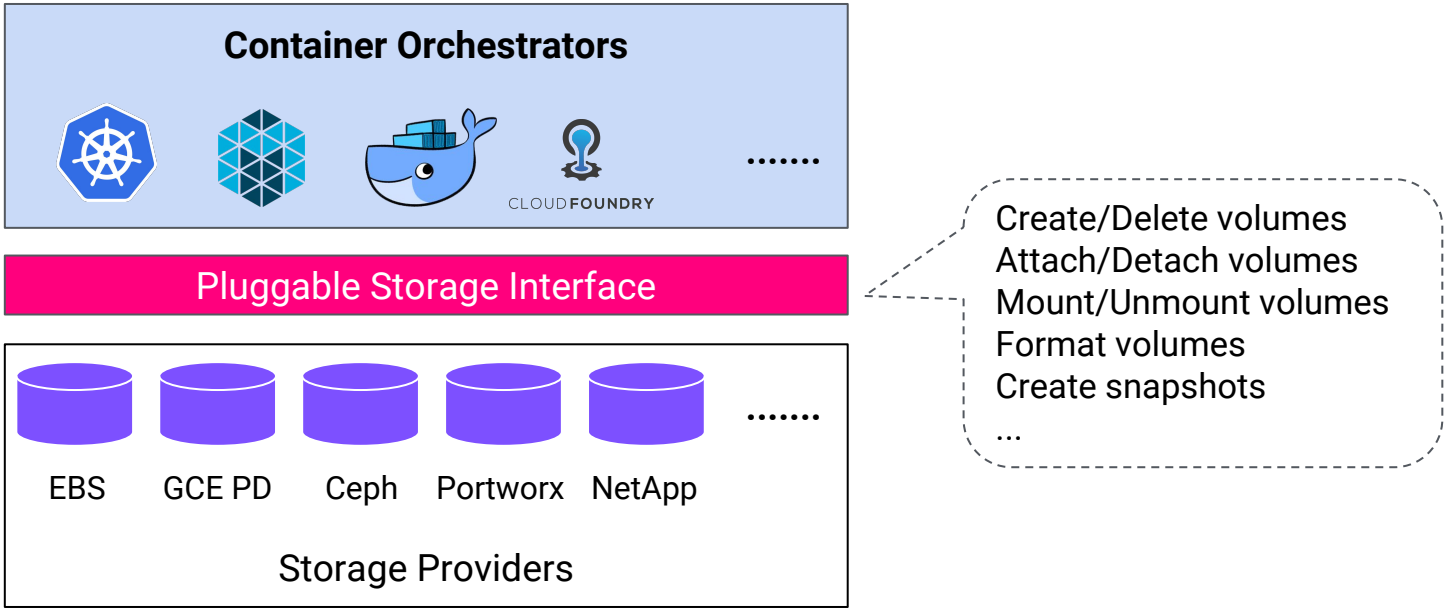
- Tech Lead @Mesosphere
- Co-author of CSI spec
- Apache Mesos PMC & Committer
- CSE PhD from U. of Michigan
- Former Twitter Engineer

Outline

- Motivations and Background
- Goals
- Design Choices
- Spec Overview
- CO Integrations
- Governance Model
- Future

Why CSI? &

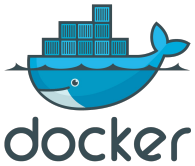
Storage in Cloud Native Environment



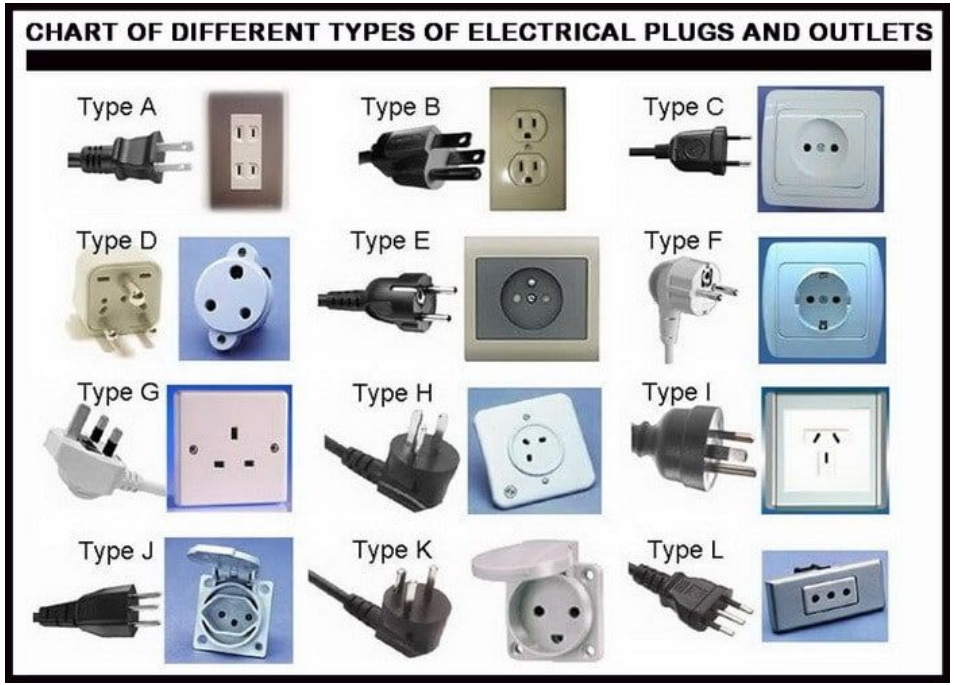
Storage Interfaces - Container Orchestrators (CO)

Popular container orchestrators or middleware have independently evolved storage interfaces

- Docker volume plugins [link]
- K8s FlexVolume [link]
- K8s In-tree volume plugins [link]
- Libstorage storage drivers [link]
- OpenSDS volume drivers [link]



Storage Interfaces - Storage Providers (SP)



Source: https://www.yuadon.com/category/news/

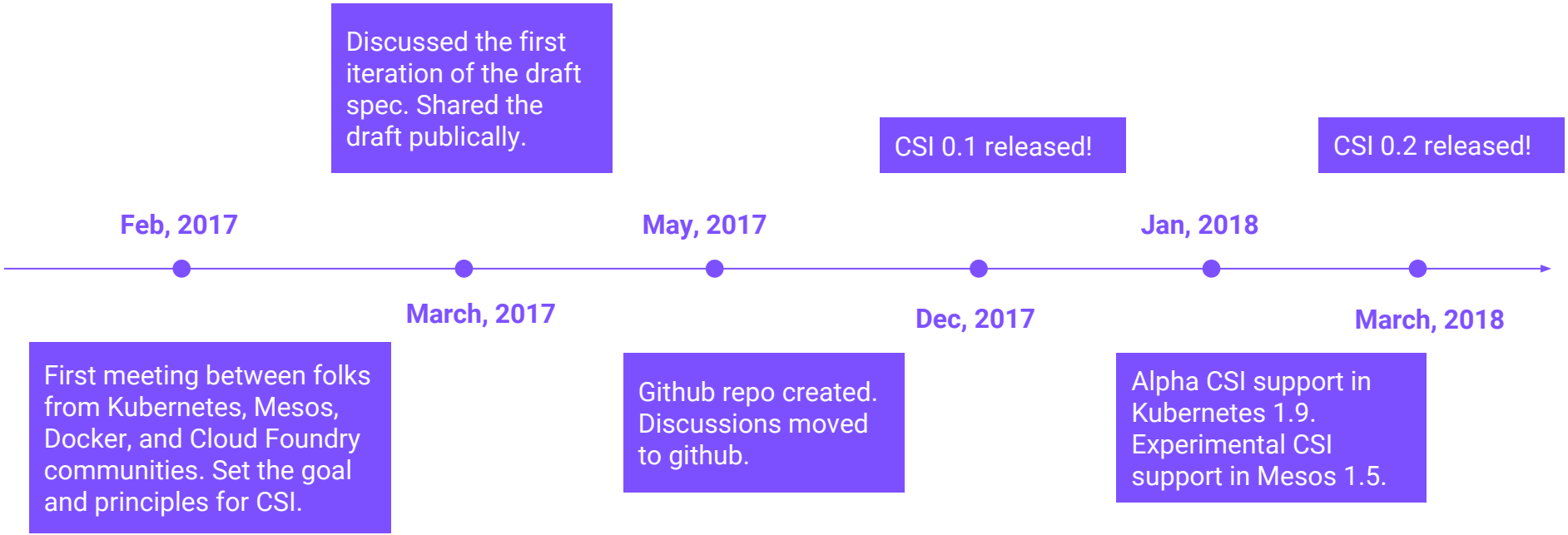
Problems with Existing Storage Interfaces

- CLI based interface
- Lack of Idempotency on APIs
- In-tree interface
- Tightly coupled with an implementation
- Too heavyweight

Container Storage Interface (CSI)

- Goals
  - Interoperability
  - Vendor neutral
  - Focus on specification
  - Control plane only
  - Keep it simple and boring

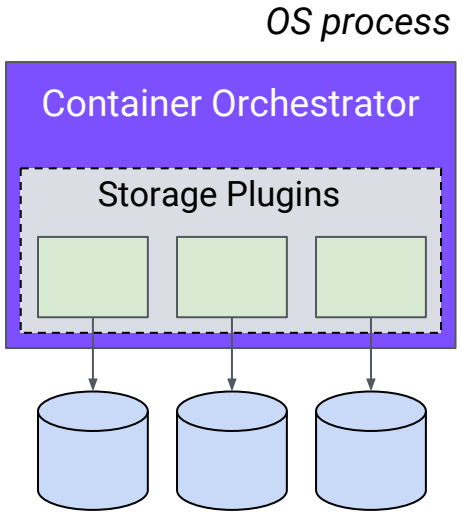
Timeline


Feb, 2017
First meeting between folks from Kubernetes, Mesos, Docker, and Cloud Foundry communities. Set the goal and principles for CSI.
March, 2017
Discussed the first iteration of the draft spec. Shared the draft publically.
May, 2017
Github repo created. Discussions moved to github.
Dec, 2017
CSI 0.1 released!
Jan, 2018
Alpha CSI support in Kubernetes 1.9. Experimental CSI support in Mesos 1.5.
March, 2018
CSI 0.2 released!

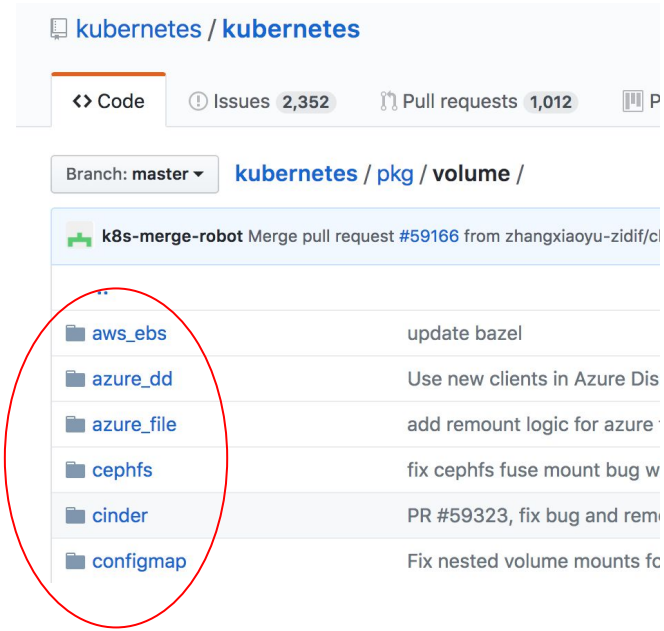
Design Choices

- In-tree vs. Out-of-tree
- Service vs. CLI
- Controller and Node services
- Idempotency
- Wire protocol: gRPC
- Async vs. Sync APIs
- Plugin packaging and deployment

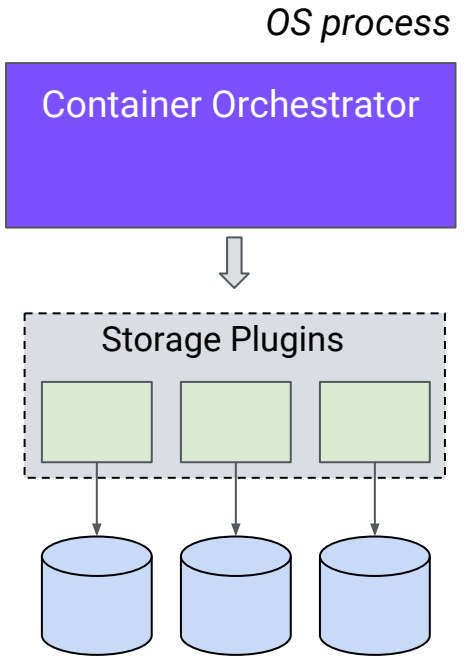
Plugin Model: In-tree



In-tree plugins



Plugin Model: Out-of-tree



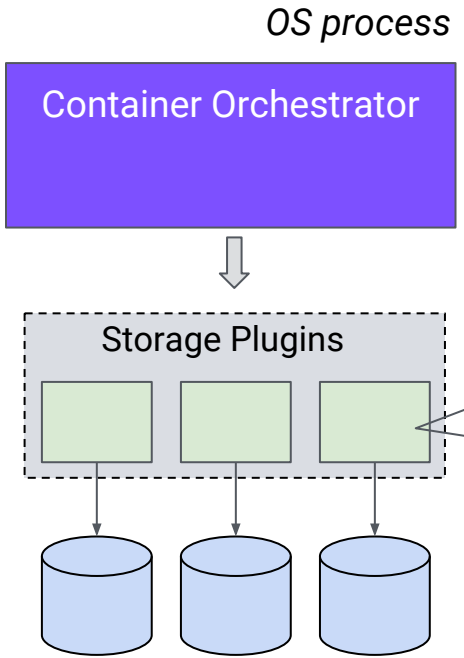
Examples: FlexVolume

Plugin Model: In-tree vs. Out-of-tree

- **Decision**: Out-of-tree

- Drawbacks of in-tree plugin model
  - Coupled release cycles
  - Testing burden
  - Force a choice on the language
  - Force plugins to be open source
  - Security (same privilege as the CO)

Service vs. CLI



- CLI?
  - Vendor deploys binaries on hosts
  - CO invokes the binary with args
- Long running service?
  - Vendor deploys services on hosts
  - CO makes requests to the service

Service vs. CLI

- **Decision**: Service

- Reasoning
  - Services are much easier to deploy
  - Root access required to install CLI binaries
  - Deploying CLI binary dependencies is not easy
  - Fuse based backends require long running processes

Controller and Node services

- Volume operations that have to be executed on the node
  - OS mount/unmount (e.g., mount(2) on Linux)
  - iSCSI initiator

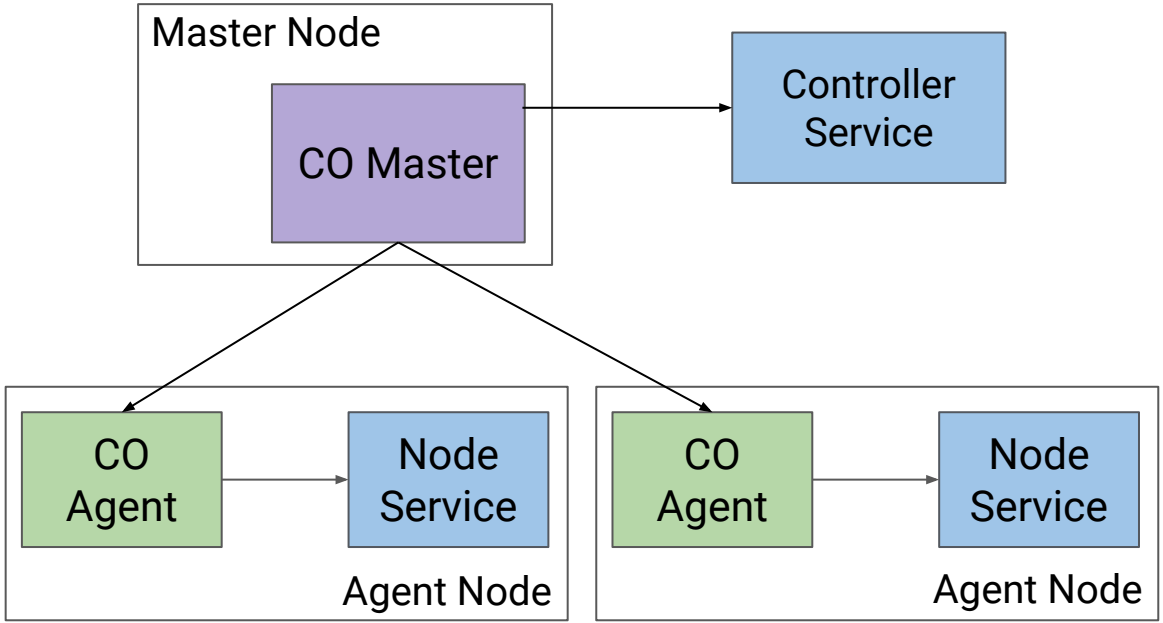
- Volume operations that can be executed on any node
  - Volume attach/detach (e.g., EBS)
  - Volume creation/deletion

Controller and Node services

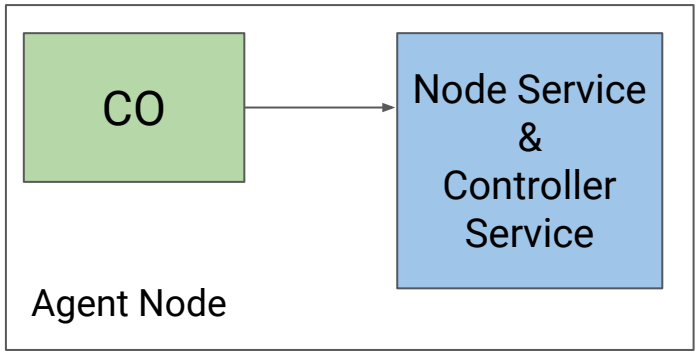
- **Decision**: Two sets of APIs
  - Controller service
  - Node service

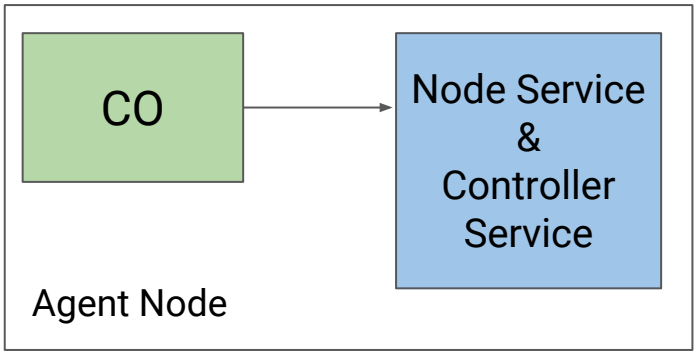
- Node services have to run on the *node*
  - *node* is where the volume will be used.

Option 1: Split Controller and Node Services



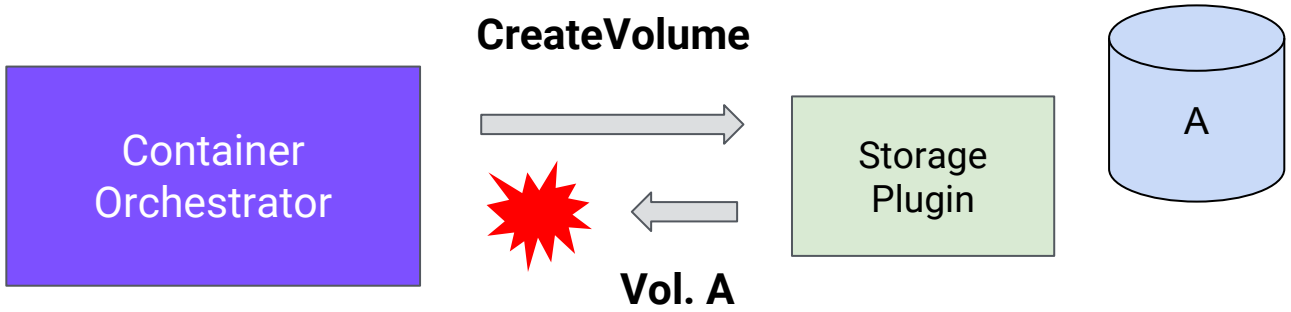
Option 2: Headless





Idempotency

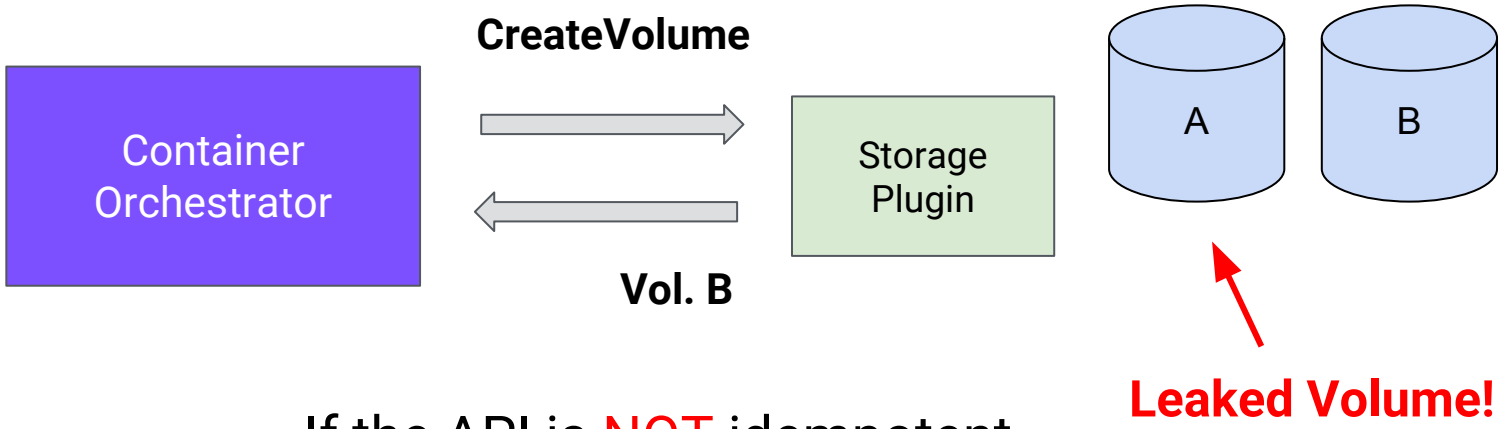
- Why this is important for a CO? Failure recovery!



If the API is NOT idempotent ...

Idempotency

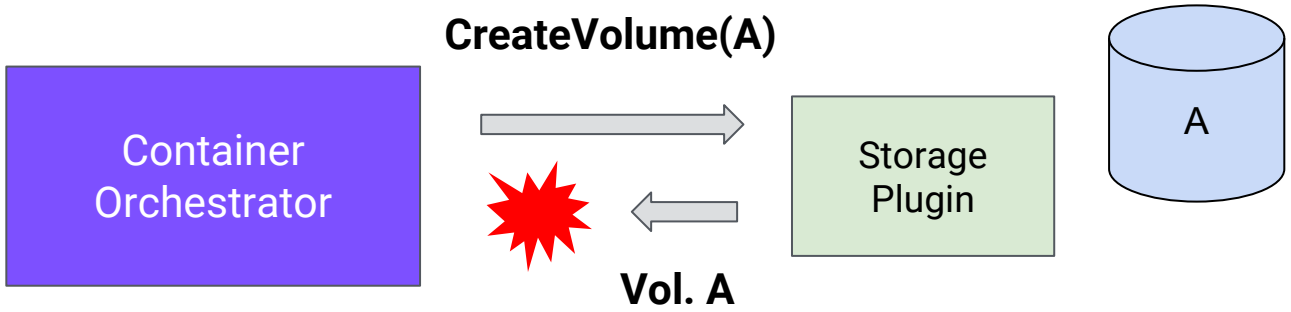
- Why this is important for a CO? Failure recovery!



If the API is NOT idempotent ...

Idempotency

- Why this is important for a CO? Failure recovery!

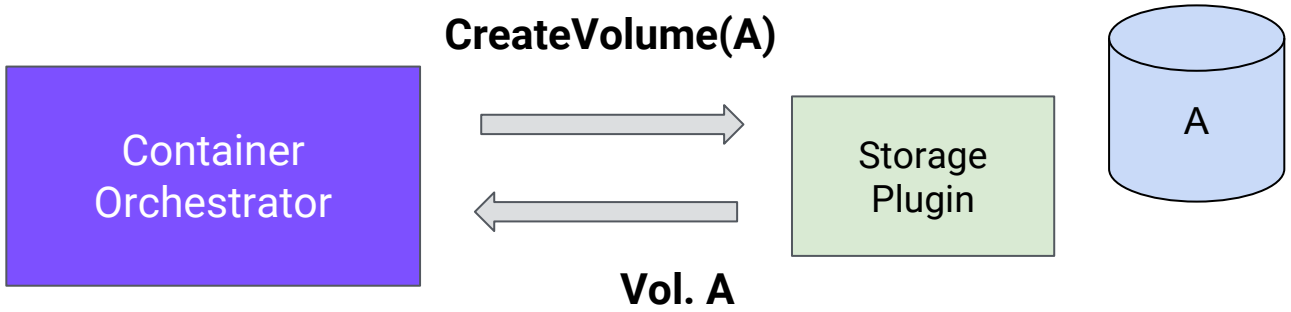


If the API is idempotent ...

Idempotency

Designing robust and predictable APIs with idempotency
https://stripe.com/blog/idempotency

- Why this is important for a CO? Failure recovery!



If the API is idempotent ...

Wire protocol: gRPC

- Why gRPC?
  - Language agnostic
  - Easy to write specification
  - Big community with lots of tooling
  - Real production users

Async vs. Sync APIs

- **Decision**: Synchronous API

- Reasoning:
  - Keep it simple. Async is significantly more complex
  - Async does not solve the long running operation problem
  - The key is to make the call idempotent for failure recovery
  - Plugin implementation can still be async

Plugin packaging and deployment

- **Decision**: Do not dictate
  - The only requirement is to provide gRPC endpoints (over unix socket for now)

- Possible options:
  - Containers deployed by CO (e.g., DaemonSet)
  - Systemd services deployed by cluster admin

API Overview

- 3 core gRPC services
  - Identity
  - Controller
  - Node

Identity Service

```
service Identity {
  rpc GetPluginInfo(...) ...
  rpc GetPluginCapabilities(...)...
  rpc Probe (...)...
}
```

Controller Service

```
service Controller {
  rpc CreateVolume (...) ...                          Optional
  rpc DeleteVolume (...) ...                          Optional
  rpc ControllerPublishVolume (...) ...               Optional
  rpc ControllerUnpublishVolume (...) ...             Optional
  rpc ValidateVolumeCapabilities (...) ...
  rpc ListVolumes (...) ...                           Optional
  rpc GetCapacity (...) ...                           Optional
  rpc ControllerGetCapabilities (...) ...
}
```

Node Service

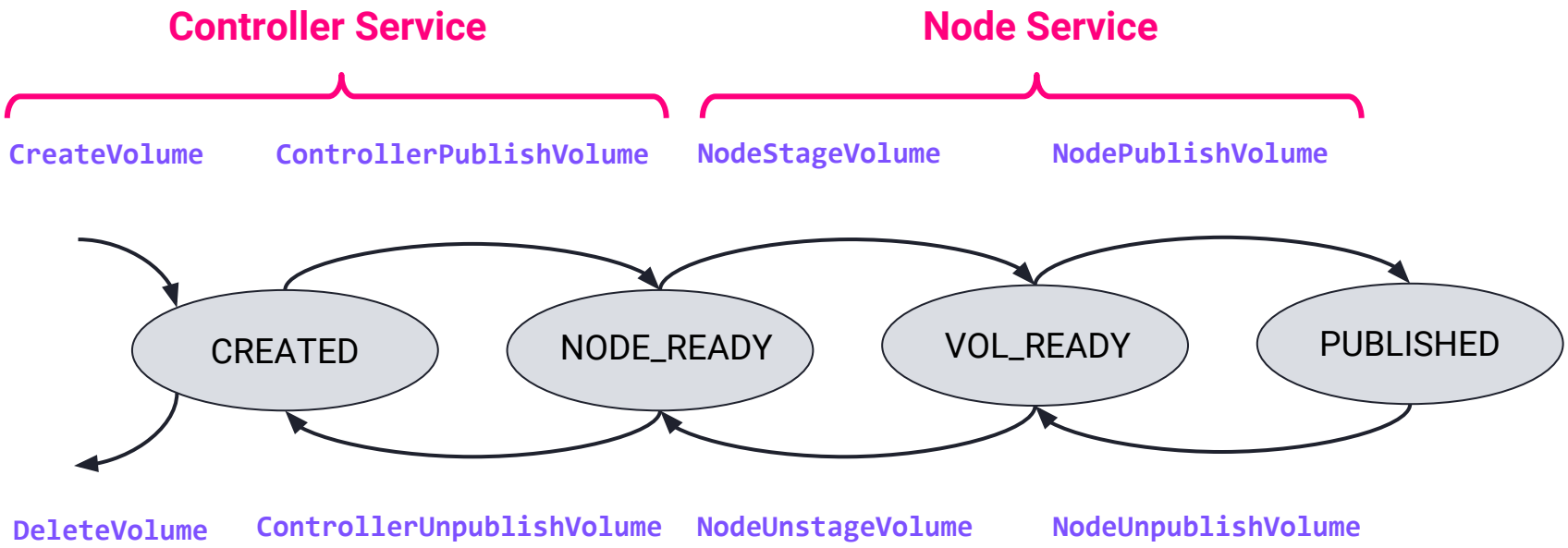
```
service Node {
  rpc NodeStageVolume (...) ...
  rpc NodeUnstageVolume (...) ...
  rpc NodePublishVolume (...) ...
  rpc NodeUnpublishVolume (...) ...
  rpc NodeGetId (...) ...
  rpc NodeGetCapabilities (...) ...
}
```

Optional

Optional

Optional

Volume Lifecycle



Plugin Case Study: NFS

- No need for Controller Service!
- Perform mount in NodePublishVolume

- Reference implementation
  - https://github.com/kubernetes-csi/drivers/tree/master/pkg/nfs

Plugin Case Study: GCE PD

- Need both Controller and Node services
- Create the persistent disk in CreateVolume
  - Using disks.get and disks.insert APIs
- Attach the disk in the ControllerPublishVolume
  - Using instances.get and instances.attachDisk APIs
- Format and mount the volume in NodeStageVolume
- Perform a bind mount in NodePublishVolume

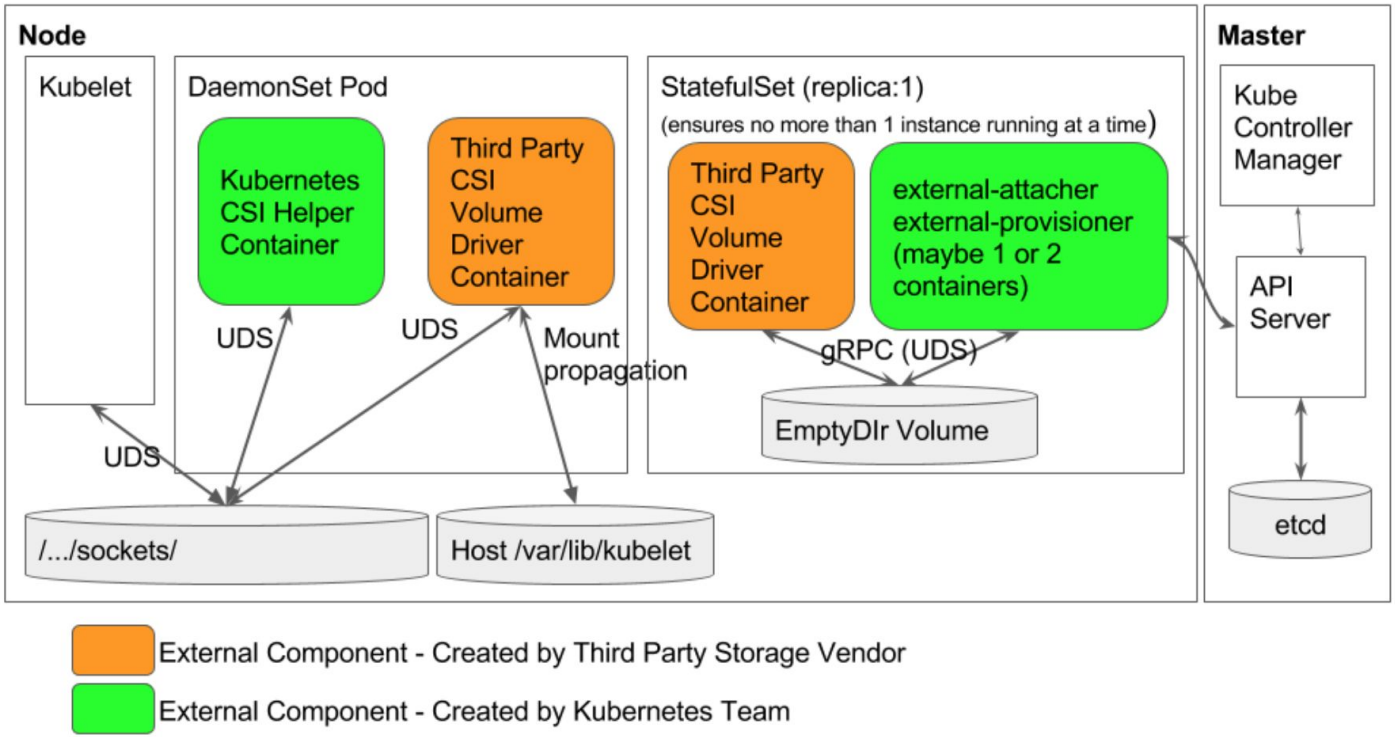
- Reference implementation
  - https://github.com/GoogleCloudPlatform/compute-persistent-disk-csi-driver

Plugin Case Study: LVM

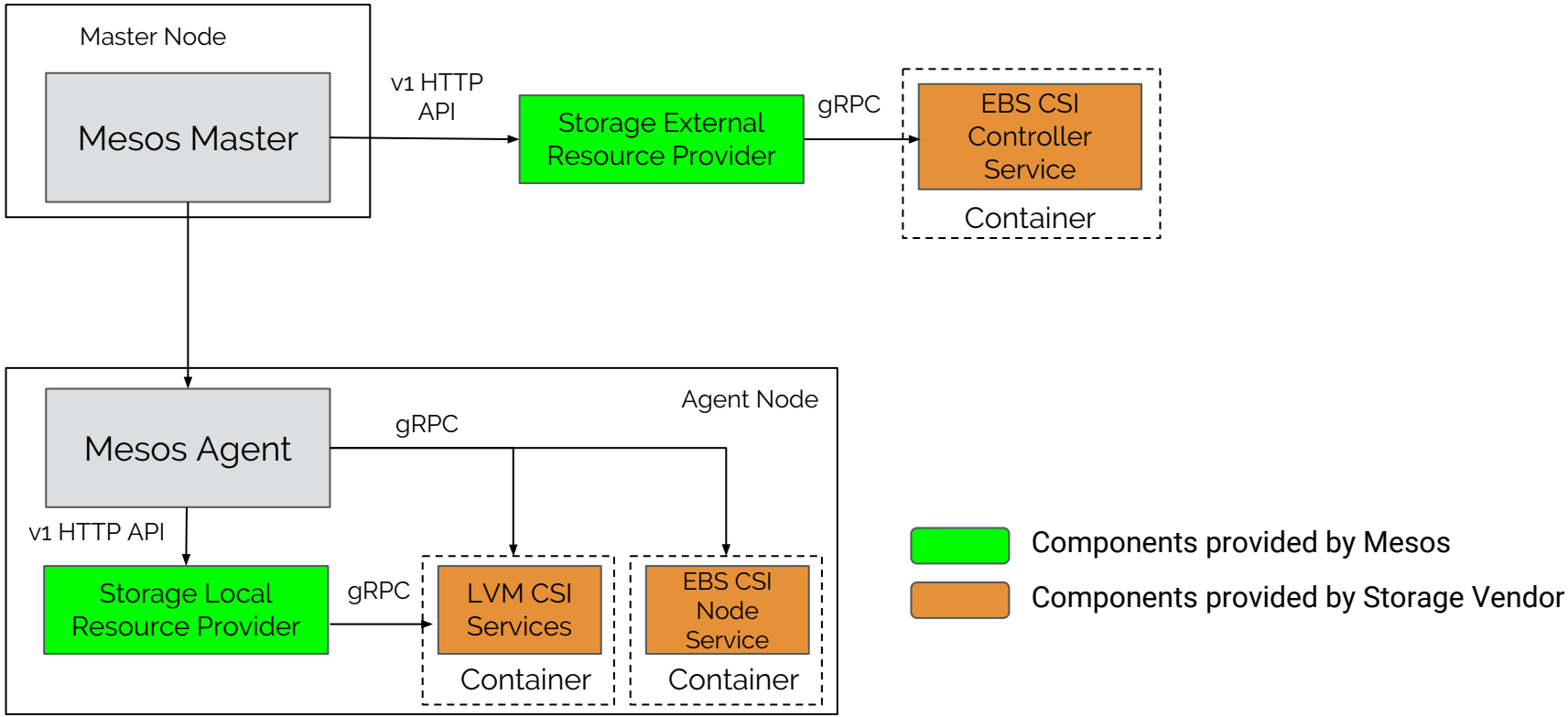
- Both Controller and Node services are deployed on the node
- Create logical volumes (`lvcreate`) in CreateVolume
- No need for ControllerPublishVolume
- Format and mount the volume in NodePublishVolume

- Reference implementation
  - https://github.com/mesosphere/csilvm

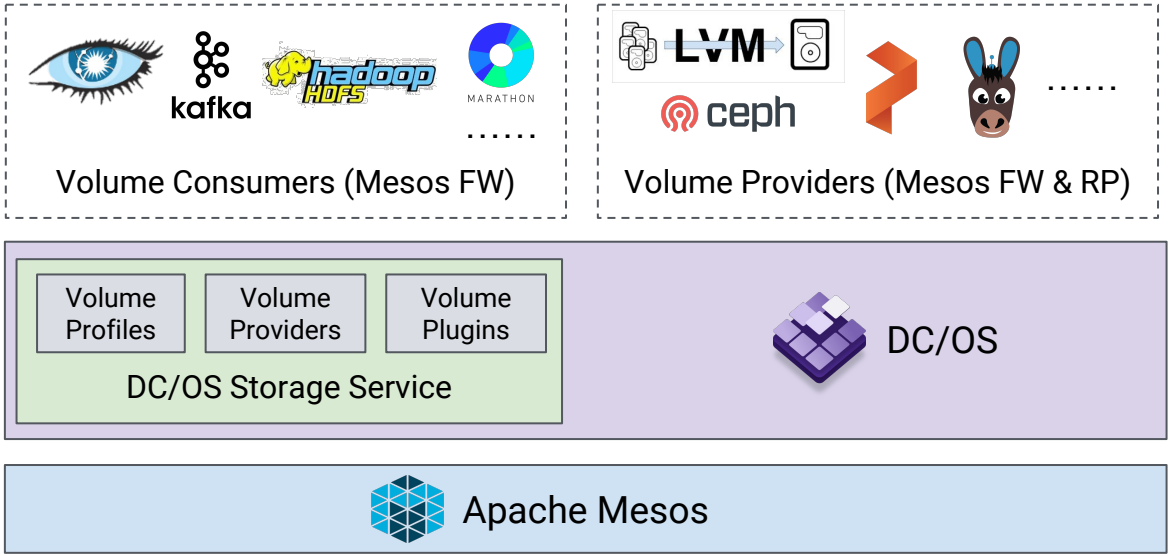
CO Integrations: Kubernetes



CO Integrations - Apache Mesos



DC/OS Storage Support Based on CSI



https://docs.mesosphere.com/services/beta-storage/0.1.0-beta/

Governance Model

- Goals
  - Inclusive and open
  - Independent of any single CO
  - Try to avoid a storage vendor war

Governance Model

Community Membership

Role	Responsibilities	Requirements	Defined by
member	Active contributor in the community.	Sponsored by 2 reviewers. Multiple contributions to the project.	CSI GitHub org member.
reviewer	Review contribution from other members.	History of review and authorship.	OWNERS file reviewer entry.
approver	Approve contributions for merge.	Core group of CO representatives.	OWNERS file approver entry.
janitor	Org and repo maintenance.	Minimal subset of approvers required for maintenance.	GitHub repo write access.

https://github.com/container-storage-interface/community/blob/master/governance.md

Future Work

- Topology aware
- Snapshot support
- Volume resizing
- Plugin registration
- Smoke test suite

Thanks!

- Q&A

- Resources
  - Spec: https://github.com/container-storage-interface/spec
  - Community: https://github.com/container-storage-interface/community
  - Mailing list: container-storage-interface-community@googlegroups.com
  - Community sync: Weekly on Wednesdays at 9 AM (PST)
  - Recordings: https://www.youtube.com/channel/UC2KKgeIo5x3W0wjEz0RbKyg