CSI Driver Developer Guide

kubernetes-csi.github.io/docs

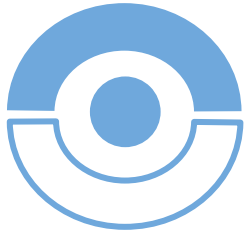
Cloud-Native Security India Community Group

27 April 2024

@parthyadav3105

@parthyadav3105

@parthyadav3105 👤

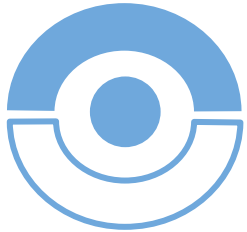


Extending k8s 🔌

Cloud Computing | Extending Kubernetes | All things Cloud-Native & Clean Code

LFX'20 mentee @ LFN-Anuket | CKA'24 (scored: 98) | SDE-II @ Coredge.io

Prerequisite

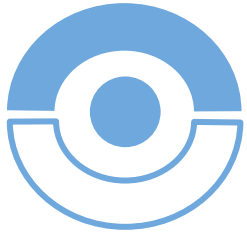


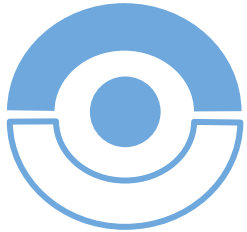
What you should know as a Kubernetes user?

- PersistentVolumeClaim
- PersistentVolume
- Dynamic Volume Provisioning
- StorageClass

Let us take a small recap: parthyadav.netlify.app/slides/storage-in-kubernetes

Table of Contents





Understanding CSI Standard

Let us first start with the CSI spec

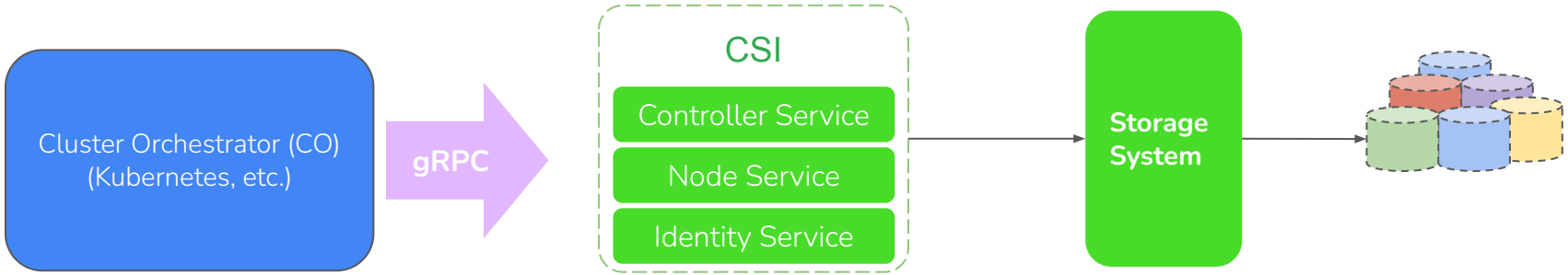
@parthyadav3105

CSI (Container Storage Interface)

Standard gRPC interface, storage vendor/provider implements.

Made of 3 services:

- Controller plugin
- Node plugin
- Identity plugin



Control flow Architecture

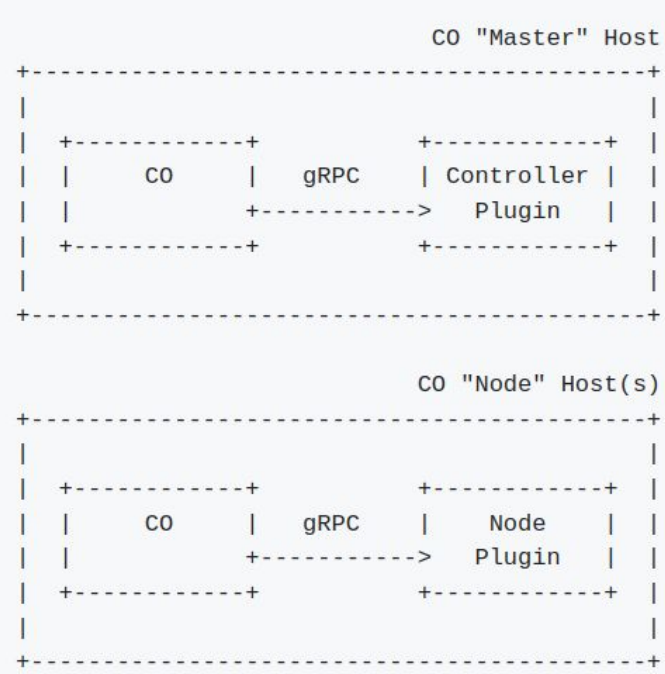
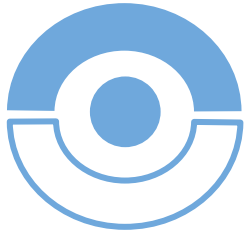


Figure 1: The Plugin runs on all nodes in the cluster: a centralized Controller Plugin is available on the CO master host and the Node Plugin is available on all of the CO Nodes.

Controller plugin do most of the work like create, delete, get, list, attach, expand volumes.

Node plugin do some node level operation like volume mount, usage stats, bind mount, etc. which controller-plugin cannot perform remotely.
(runs on each node)

Standard gRPC Interface of CSI: Overview



Identity Service

```
service Identity {
  rpc GetPluginInfo(GetPluginInfoRequest)
    returns (GetPluginInfoResponse) {}

  rpc GetPluginCapabilities(GetPluginCapabilitiesRequest)
    returns (GetPluginCapabilitiesResponse) {}

  rpc Probe (ProbeRequest)
    returns (ProbeResponse) {}
}
```

Standard gRPC Interface of CSI: Overview

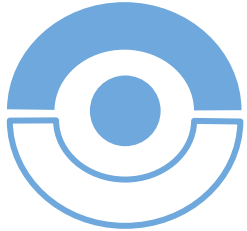
Controller Service

```
service Controller {
  rpc CreateVolume (CreateVolumeRequest)
    returns (CreateVolumeResponse) {}

  rpc DeleteVolume (DeleteVolumeRequest)
    returns (DeleteVolumeResponse) {}

  rpc ControllerPublishVolume (ControllerPublishVolumeRequest)
    returns (ControllerPublishVolumeResponse) {}

  rpc ControllerUnpublishVolume (ControllerUnpublishVolumeRequest)
    returns (ControllerUnpublishVolumeResponse) {}

  rpc ValidateVolumeCapabilities (ValidateVolumeCapabilitiesRequest)
    returns (ValidateVolumeCapabilitiesResponse) {}

  rpc ListVolumes (ListVolumesRequest)
    returns (ListVolumesResponse) {}

  rpc GetCapacity (GetCapacityRequest)
    returns (GetCapacityResponse) {}

  rpc ControllerGetCapabilities (ControllerGetCapabilitiesRequest)
    returns (ControllerGetCapabilitiesResponse) {}

  rpc CreateSnapshot (CreateSnapshotRequest)
    returns (CreateSnapshotResponse) {}

  rpc DeleteSnapshot (DeleteSnapshotRequest)
    returns (DeleteSnapshotResponse) {}

  rpc ListSnapshots (ListSnapshotsRequest)
    returns (ListSnapshotsResponse) {}

  rpc ControllerExpandVolume (ControllerExpandVolumeRequest)
    returns (ControllerExpandVolumeResponse) {}

  rpc ControllerGetVolume (ControllerGetVolumeRequest)
    returns (ControllerGetVolumeResponse) {
        option (alpha_method) = true;
    }

  rpc ControllerModifyVolume (ControllerModifyVolumeRequest)
    returns (ControllerModifyVolumeResponse) {
        option (alpha_method) = true;
    }
}
```

Standard gRPC Interface of CSI: Overview

Node Service

```
service Node {
  rpc NodeStageVolume (NodeStageVolumeRequest)
    returns (NodeStageVolumeResponse) {}

  rpc NodeUnstageVolume (NodeUnstageVolumeRequest)
    returns (NodeUnstageVolumeResponse) {}

  rpc NodePublishVolume (NodePublishVolumeRequest)
    returns (NodePublishVolumeResponse) {}

  rpc NodeUnpublishVolume (NodeUnpublishVolumeRequest)
    returns (NodeUnpublishVolumeResponse) {}

  rpc NodeGetVolumeStats (NodeGetVolumeStatsRequest)
    returns (NodeGetVolumeStatsResponse) {}


  rpc NodeExpandVolume(NodeExpandVolumeRequest)
    returns (NodeExpandVolumeResponse) {}


  rpc NodeGetCapabilities (NodeGetCapabilitiesRequest)
    returns (NodeGetCapabilitiesResponse) {}

  rpc NodeGetInfo (NodeGetInfoRequest)
    returns (NodeGetInfoResponse) {}
}
```



Understanding Storage

@parthyadav3105

What does storage mean?

So many options:

Object Stores

Amazon S3
Google Cloud Storage (GCS)
MinIO

Time series Databases

InfluxDB
Prometheus
Graphite

SQL Databases

MySQL
PostgreSQL
SQL Server

Message Queues

Apache Kafka
RabbitMQ
Google Cloud Pub/Sub
Amazon SQS

NoSQL Databases

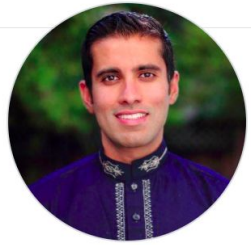
MongoDB
Redis
Etcd
Cassandra

File Storage

NFS
SMB
GlusterFS
CephFS

Block Storage

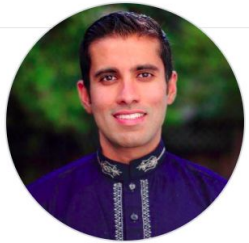
iSCSI
Fibre Channel
GCE Persistent Disks
Amazon EBS
Local Disks



original author: Saad Ali

Storage is complicated

So many options: Data Services vs Block/File



original author: Saad Ali

Data Services

Object Stores

Amazon S3
Google Cloud Storage (GCS)
MinIO

Time series Databases

InfluxDB
Prometheus
Graphite

SQL Databases

MySQL
PostgreSQL
SQL Server

Message Queues

Apache Kafka
RabbitMQ
Google Cloud Pub/Sub
Amazon SQS

NoSQL Databases

MongoDB
Redis
Etcd
Cassandra

Block/File

File Storage

NFS
SMB
GlusterFS
CephFS

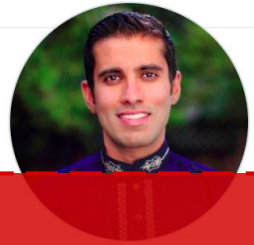
Block Storage

iSCSI
Fibre Channel
GCE Persistent Disks
Amazon EBS
Local Disks

What to focus?

So many options:

original author: Saad Ali



Object Stores

Amazon S3
Google Cloud Storage (GCS)
MinIO

Time series Databases

InfluxDB
Prometheus
Graphite

SQL Databases

MySQL
PostgreSQL
SQL Server

Messaging Systems

Apache Kafka
RabbitMQ
Google Cloud Pub/Sub
Amazon SQS

Key-Value Databases

MongoDB
Redis
Etcd
Cassandra

Not Standardized yet

Data Services

File Storage

NFS
SMB
GlusterFS
CephFS

Block Storage

iSCSI
Fibre Channel
GCE Persistent Disks
Amazon EBS
Local Disks

Standardized (Posix, SCSI)

Block/File

Storage topology

How storage reach us?

Physical Storage→ (Linux)Device Drivers→ /dev/dba Block device→ Format & create filesystem(ext4)→ ls /path/to/file

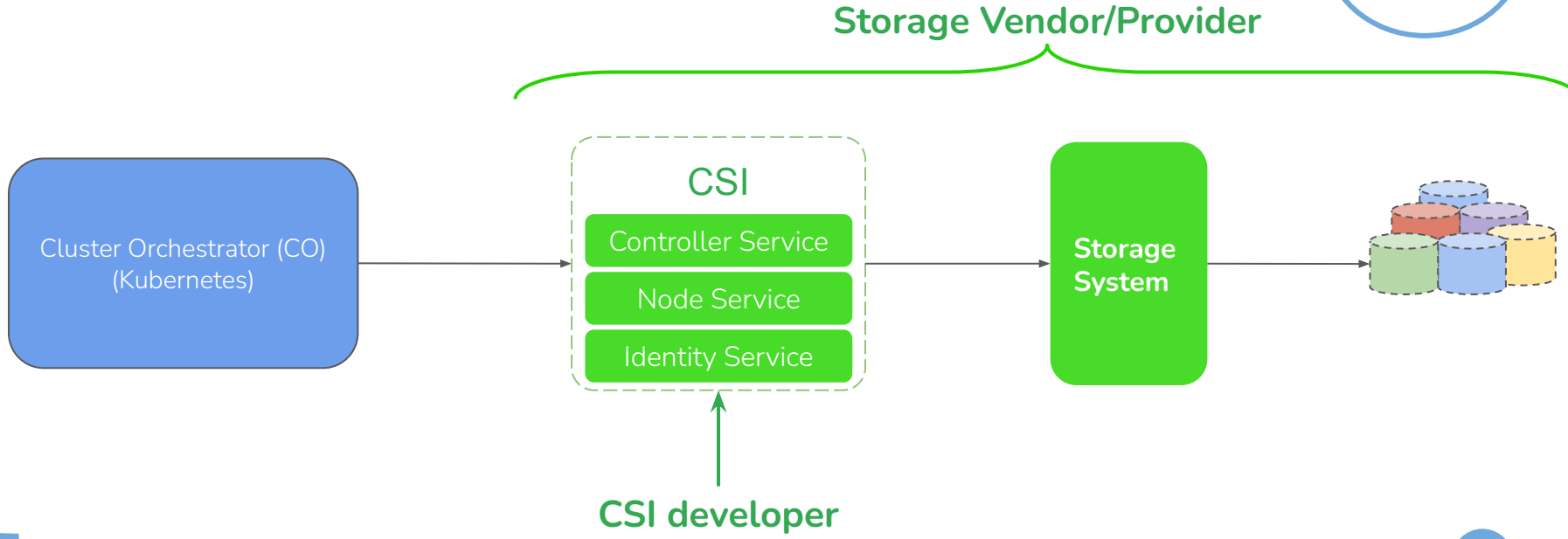
Storage over network

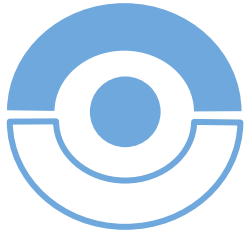
Standard Protocols like NFS, iSCSI, Fc, etc.

How storage reach us?

What does CSI developer do?

Develop CSI spec based gRPC server that talks to a third-party storage service.





Storage in Kubernetes

Supporting external storage provider

@parthyadav3105

Old: **In-Tree** approach for PV and Volume Plugins in k8s

Local

hostPath
Local

Ephemeral Storage

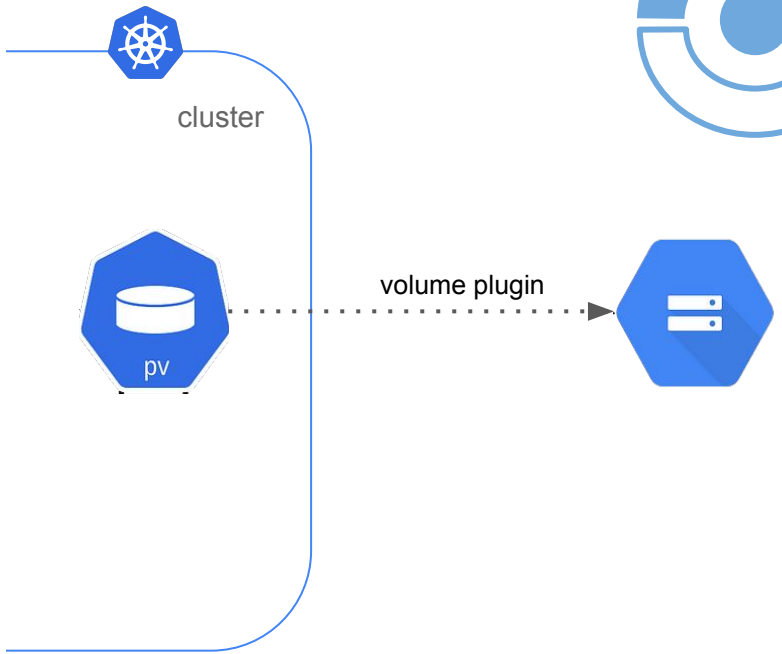
emptyDir
configMap
secret
downwardAPI

Out-of-Tree

flexVolume
csi

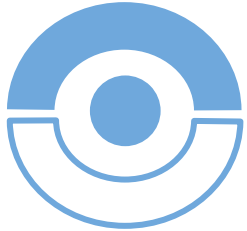
In-Tree

awsElasticBlockStore
azureDisk
azureFile
cephfs
cinder
fc
flocker
gcePersistentDisk
glusterfs
iscsi
nfs
photonPersistentDisk
portworxVolume
quobyte
rdb
scaleIO
storageos
vsphereVolume



Notes:
- many volume plugins.
- In-tree plugins are the one maintained by k8s teams.
- out-of-tree are for vendors to implement and support themself.
- maintaining in-tree plugins for 3rd party vendors was tough process.
- k8s is migrating to all in-tree plugins to csi
- this migration will not change the actual APIs, but internally the code for those in-tree api calls will be delegated to respective csi driver.

Out-of-Tree Volume Plugins



GOAL:
Allow storage vendors to support their storage in k8s independent of Kubernetes releases and codebase.

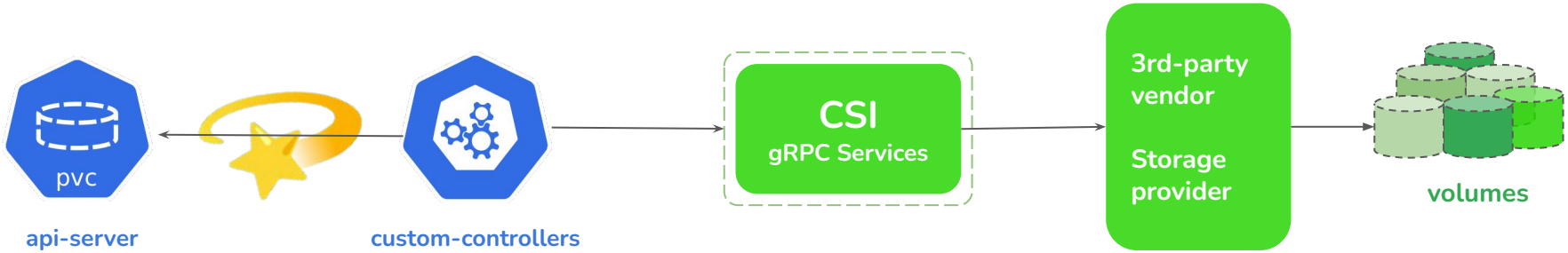
HOW:
Standard **gRPC** Interface called **Container Storage Interface(CSI)** defined by CSI spec

IMPLEMENTATION:
Kubernetes pvc, pv controllers skips volume provisioning when they find pvc/volume requested an out-of-tree volume plugin(i.e. CSI, etc). Now, external controllers can watch cluster and implement the logic for volume provisioning and update the status of PVC accordingly.

Out-of-Tree Volume Plugins

The idea: CSI driver plugin and Storage Class

approx:

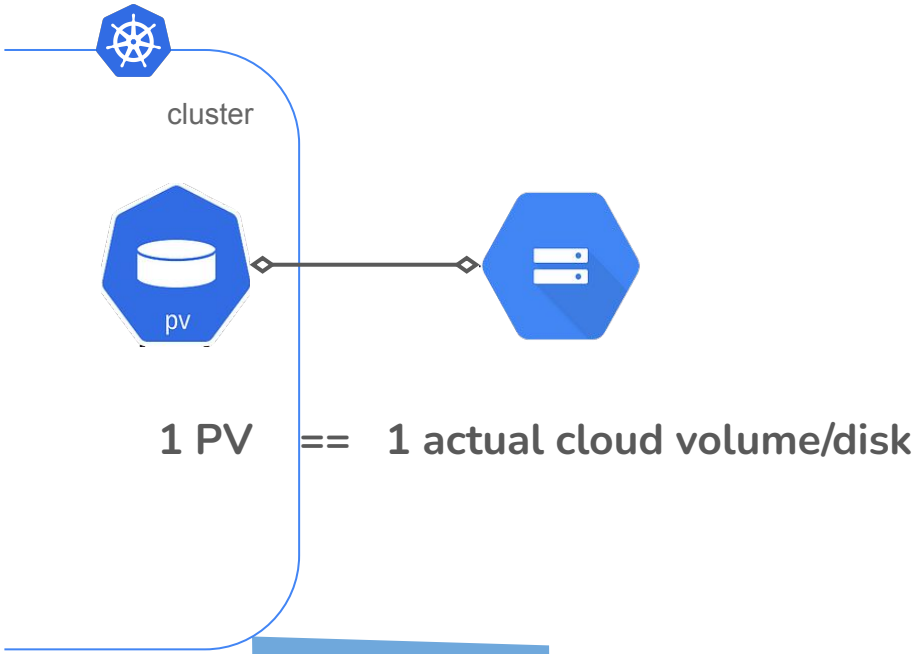


Implemented by third-party Storage Vendor

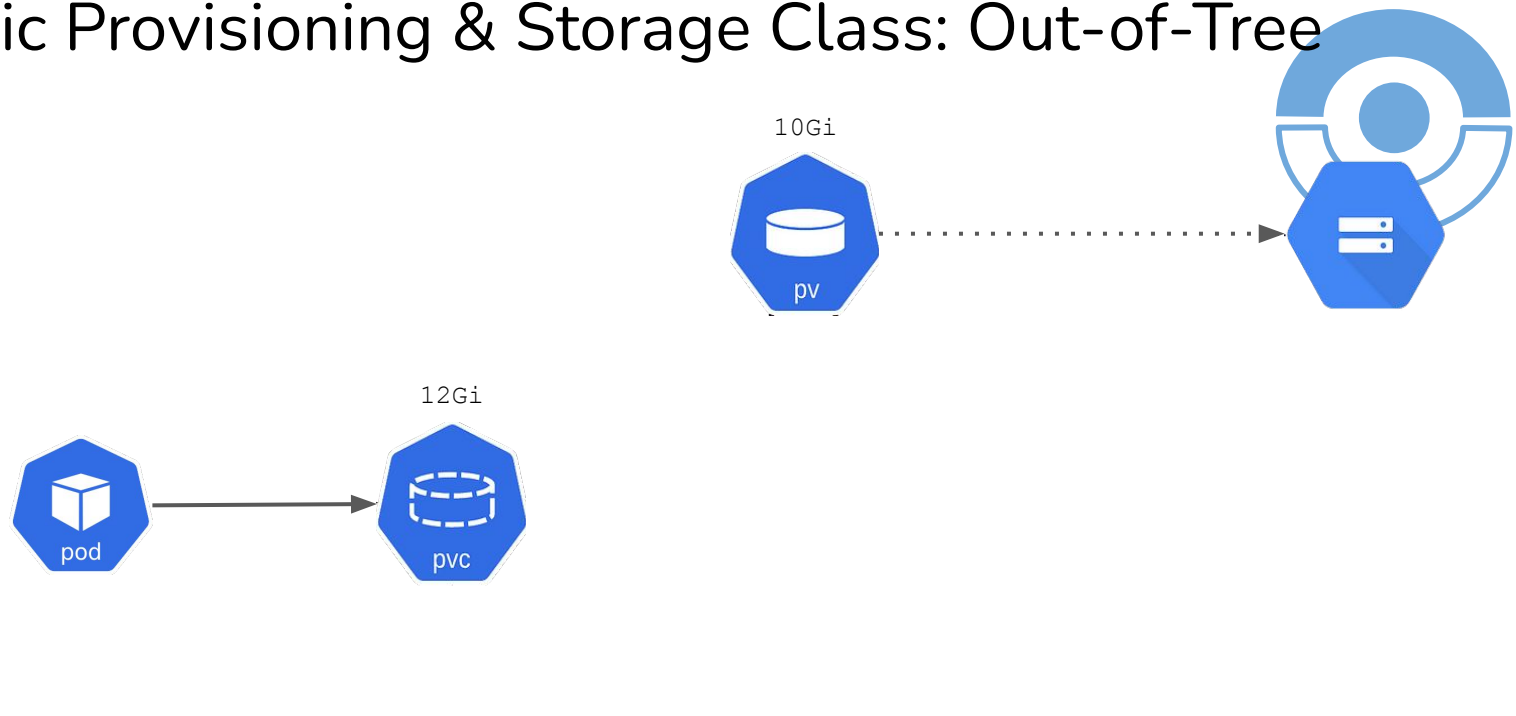
A PV represents one actual volume

PV are cluster level resource managed by admin or infra provider.

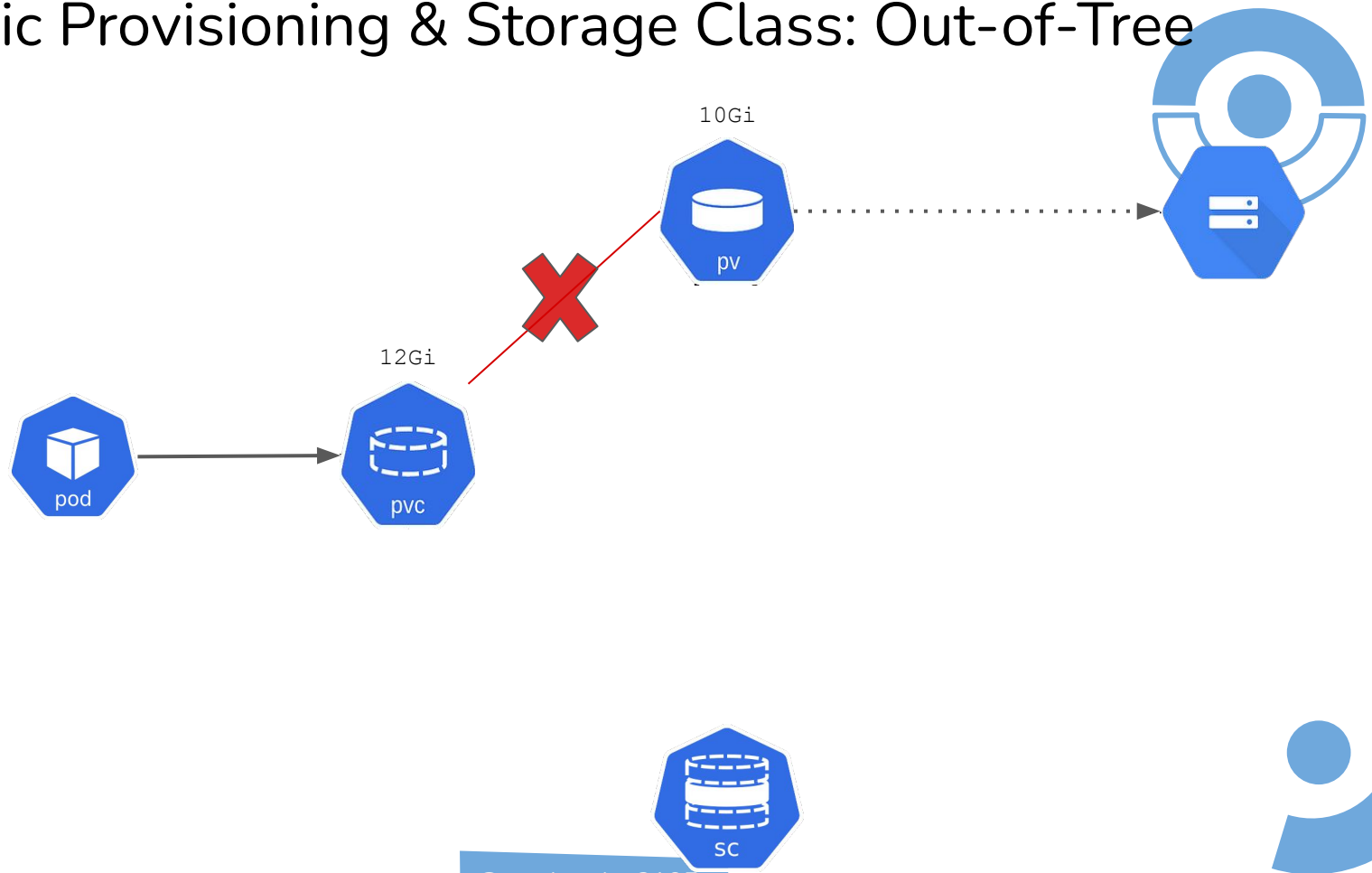
- A PersistentVolume represents an actual real volume externally in storage-cloud.



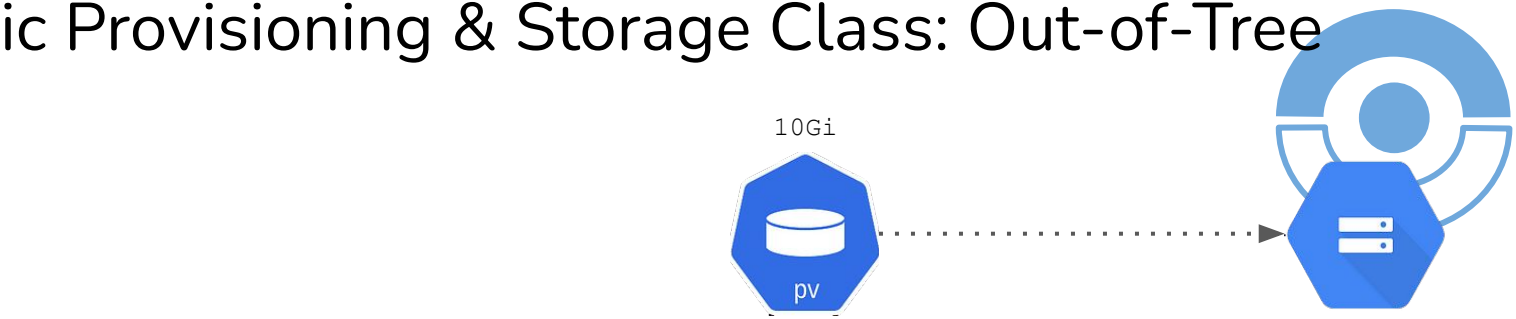
Dynamic Provisioning & Storage Class: Out-of-Tree



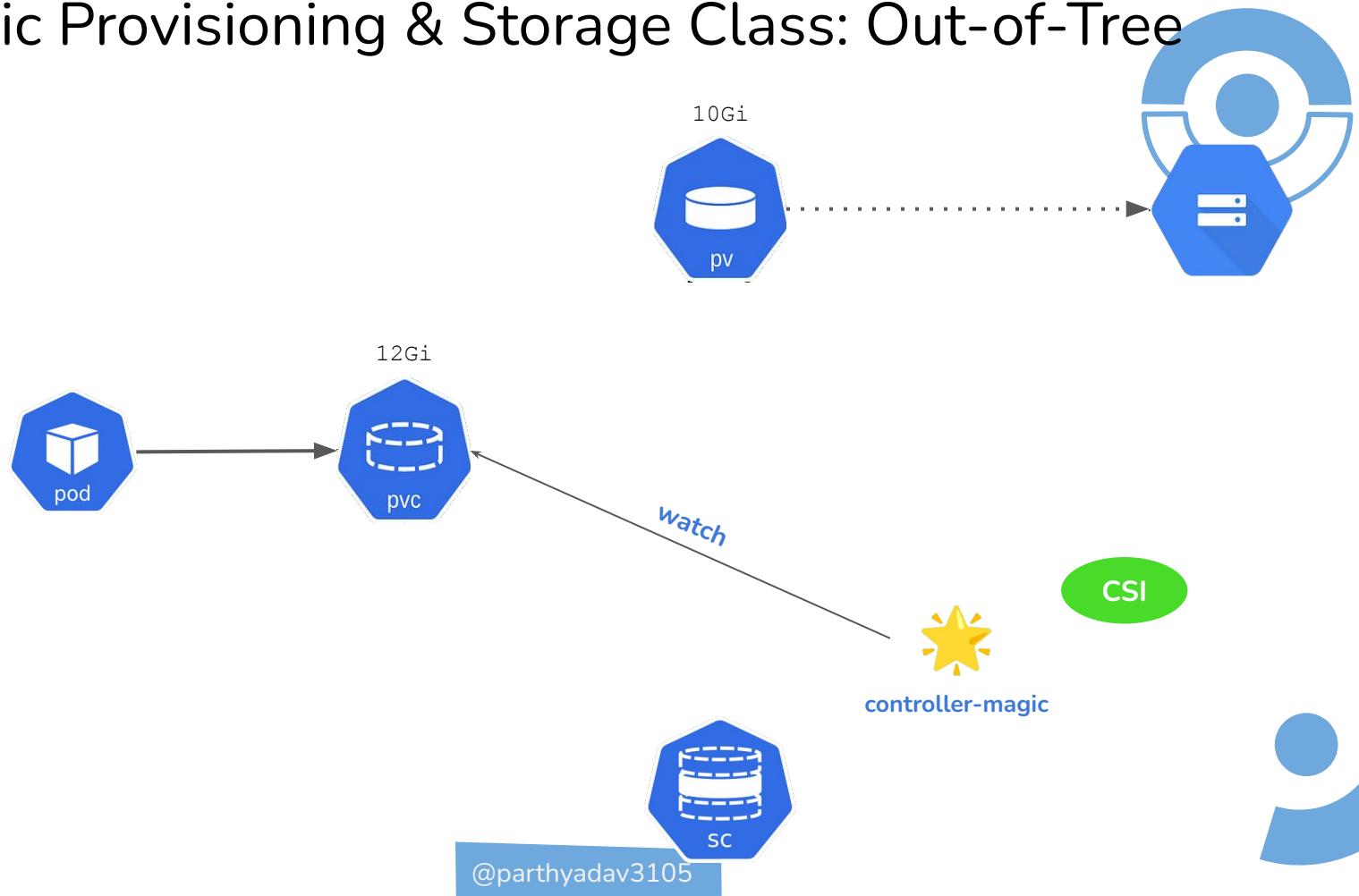
Dynamic Provisioning & Storage Class: Out-of-Tree



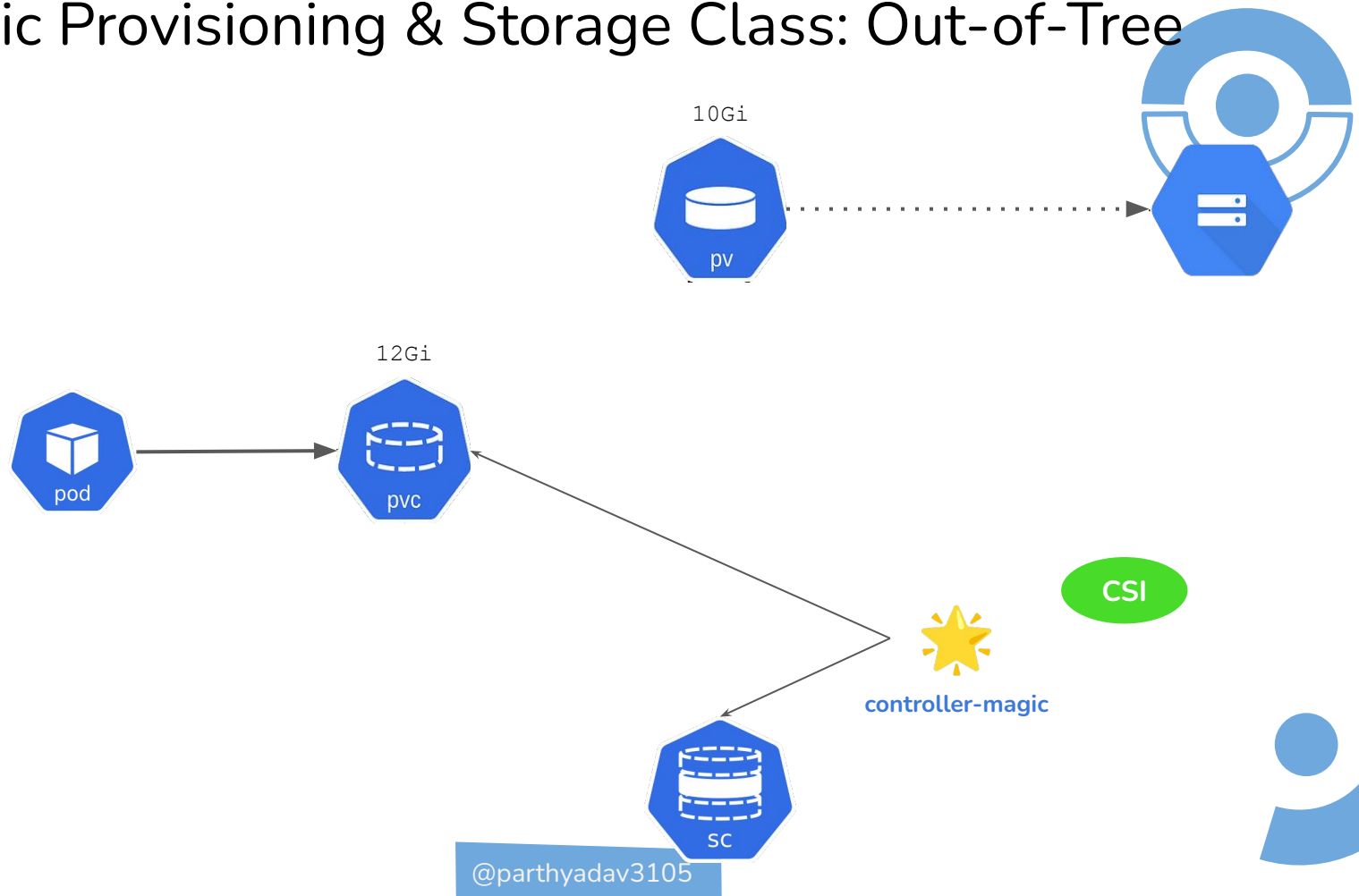
Dynamic Provisioning & Storage Class: Out-of-Tree



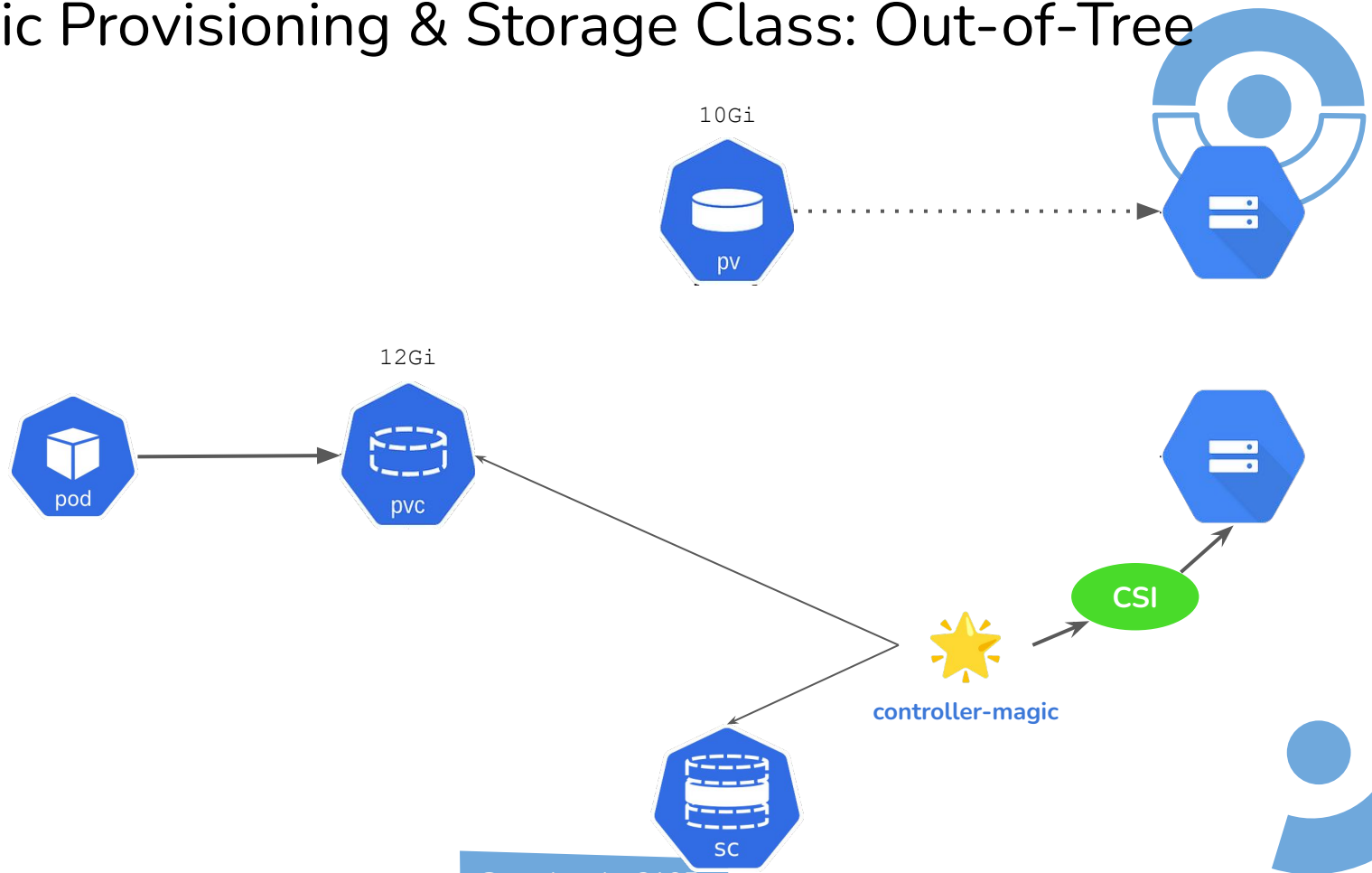
Dynamic Provisioning & Storage Class: Out-of-Tree



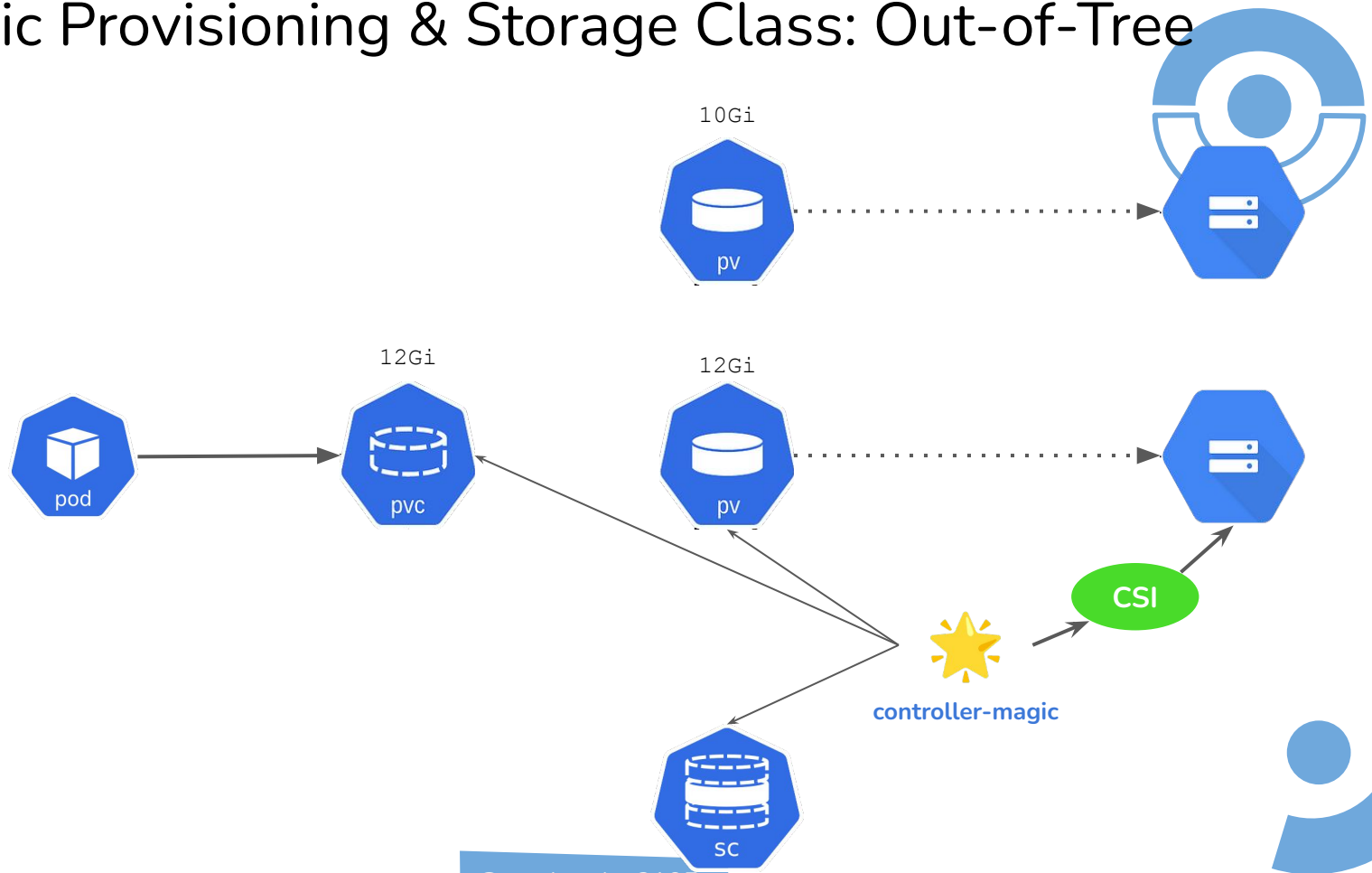
Dynamic Provisioning & Storage Class: Out-of-Tree



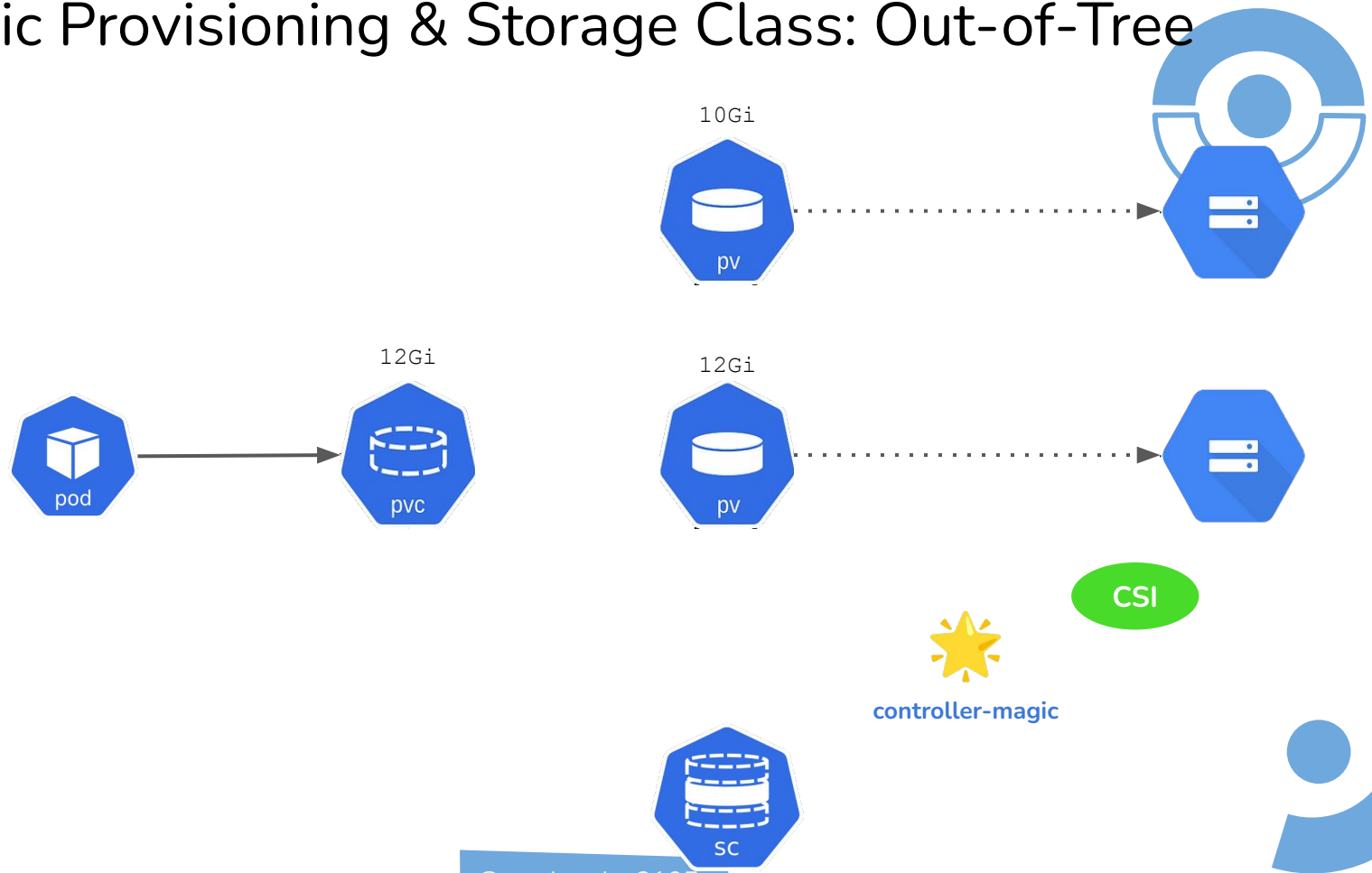
Dynamic Provisioning & Storage Class: Out-of-Tree



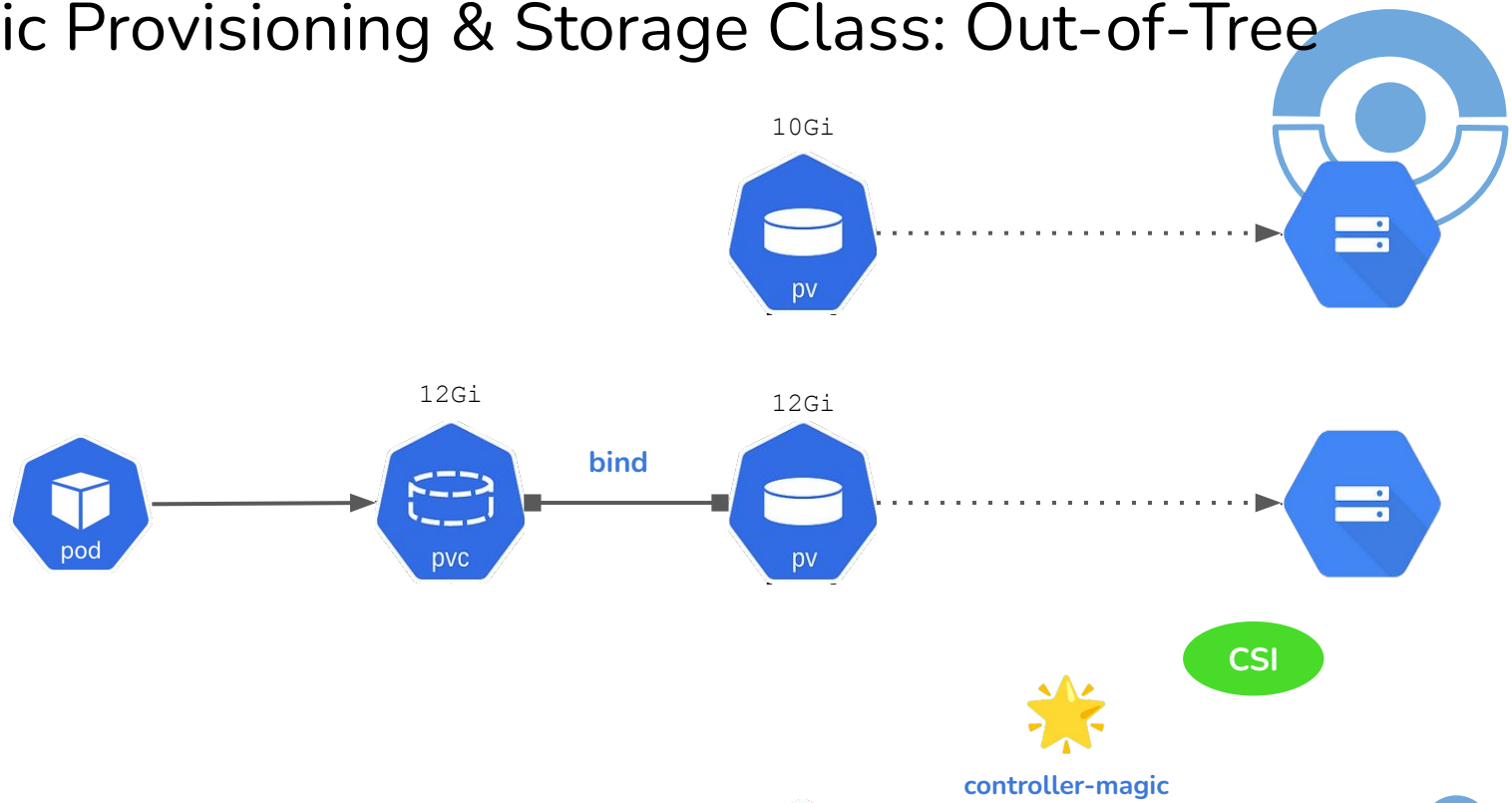
Dynamic Provisioning & Storage Class: Out-of-Tree



Dynamic Provisioning & Storage Class: Out-of-Tree

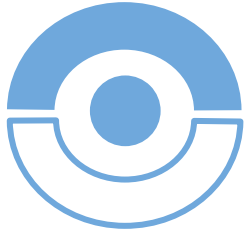


Dynamic Provisioning & Storage Class: Out-of-Tree



What is “ ” in Out-of-Tree

controller-magic



In case of Out-of-Tree, the PVC and PV controller skips volume provisioning (and other operations like volume attach).

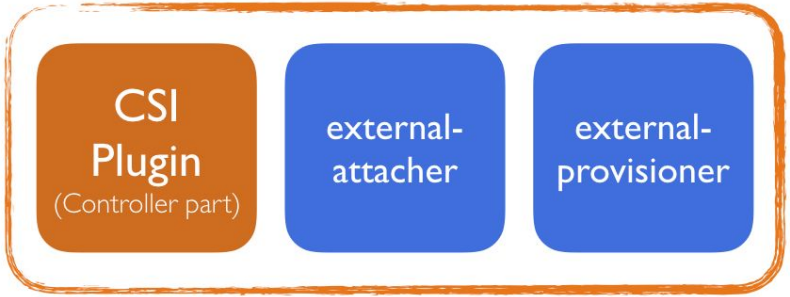
Custom controller must be deploy aside the CSI-driver to watch PVC, PV:

- external-provisioner
- external-attacher
- external-resizer
- external-snapshotter

These controllers are supported by the community and can be reused(as sidecar containers) while you develop your own csi driver.

Understanding deployment topology for CSI driver

Node plugin: runs on all nodes

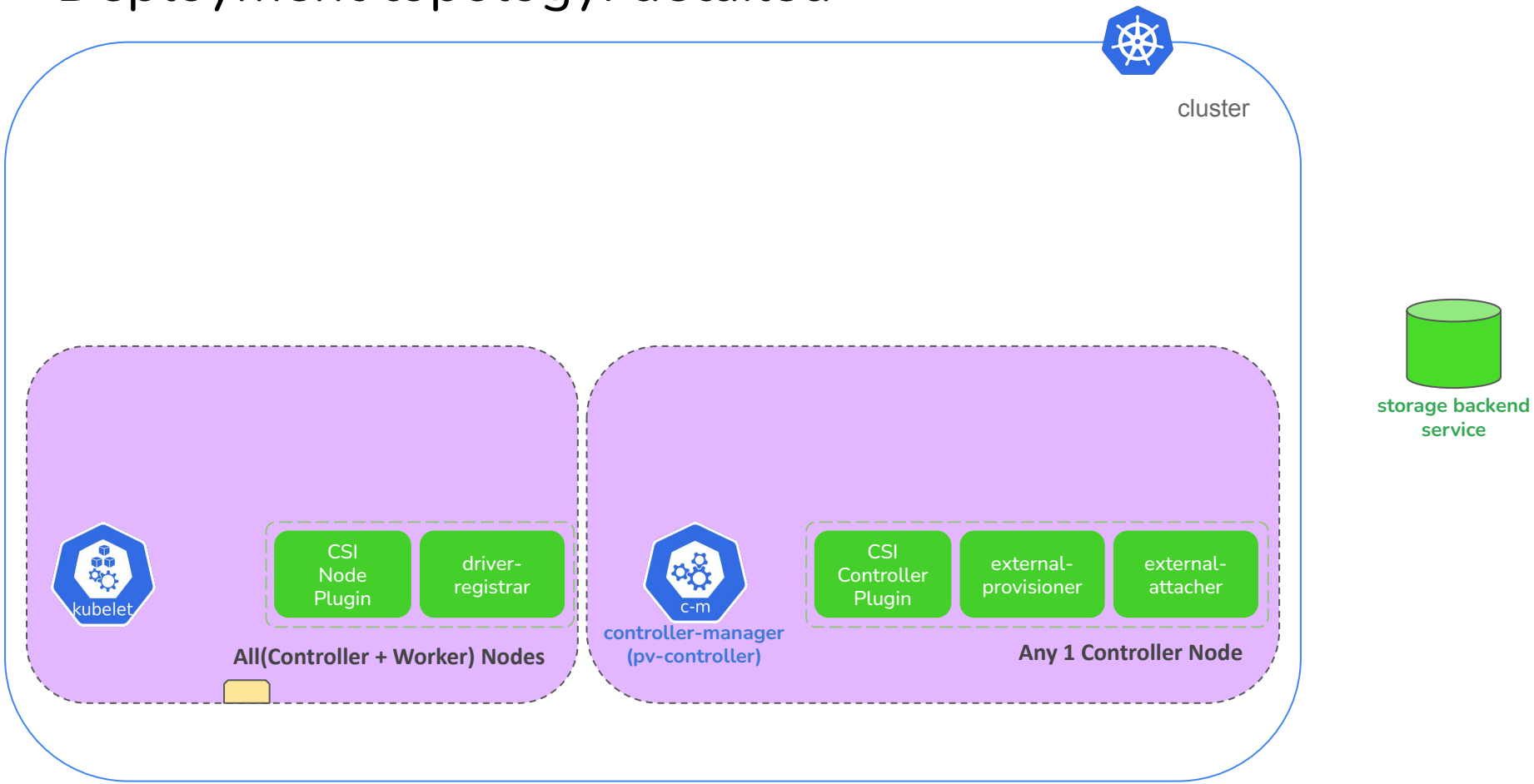


Controller plugin: can run on any node

Fatih Arslan

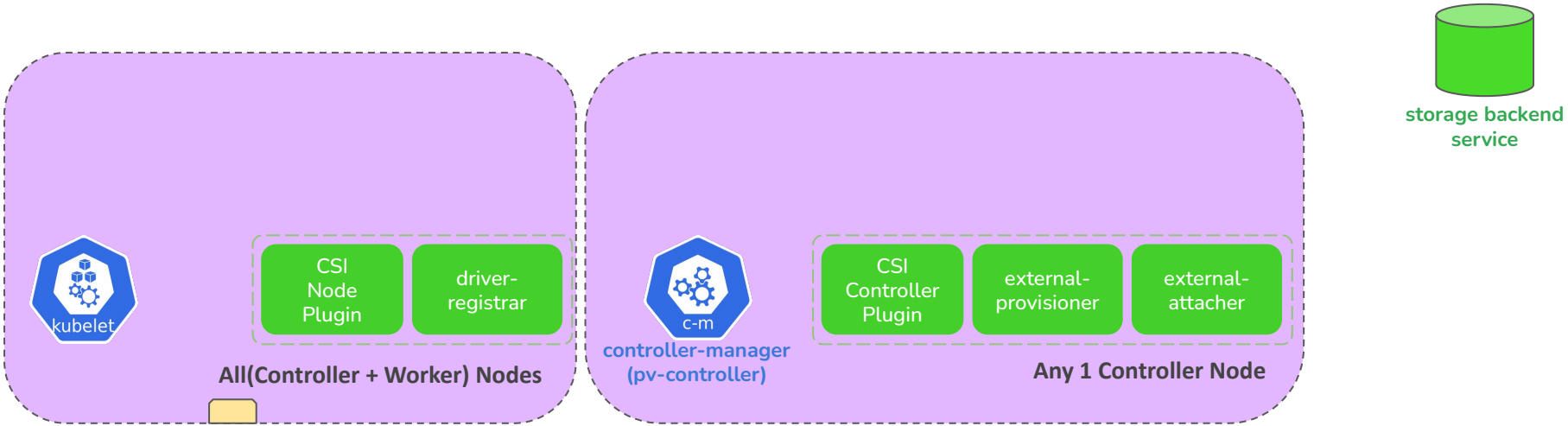
Jun 21, 2018 *author:* arslan.io/2018/06/21/how-to-write-a-container-storage-interface-csi-plugin/

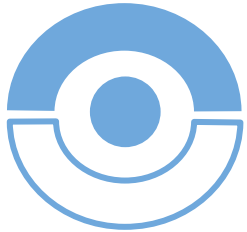
Deployment topology: detailed



Conclusion

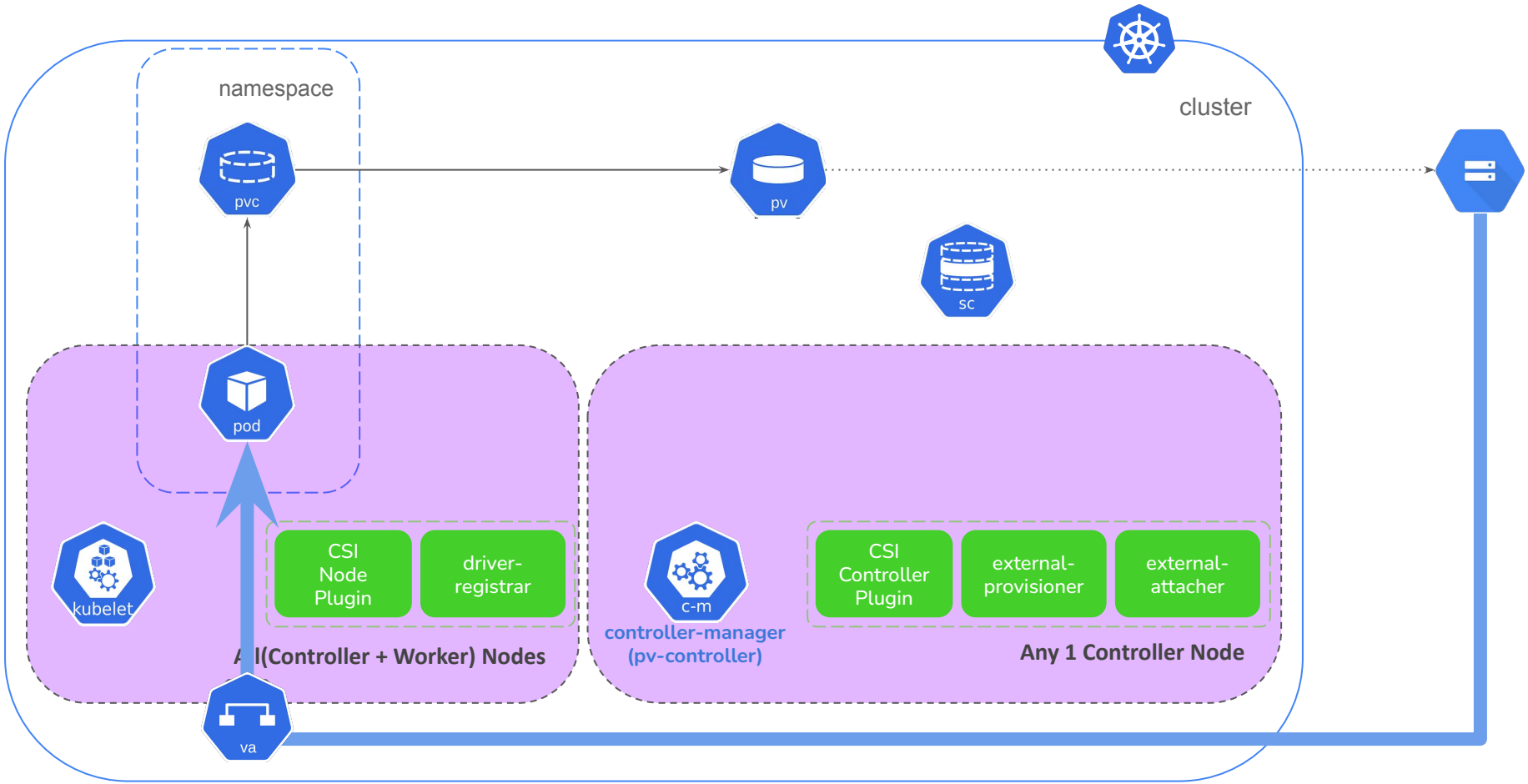
- Only CSI driver performs calls to storage backend provider.
- CSI Controller-plugin is called by controllers for create, delete, attach, list, etc of volumes.
- CSI Node plugin is called by kubelet for stage, bind mount, volume stats.
- PVC, PV bind and other PVC, PV logic is handled same as before by k8s controllers.



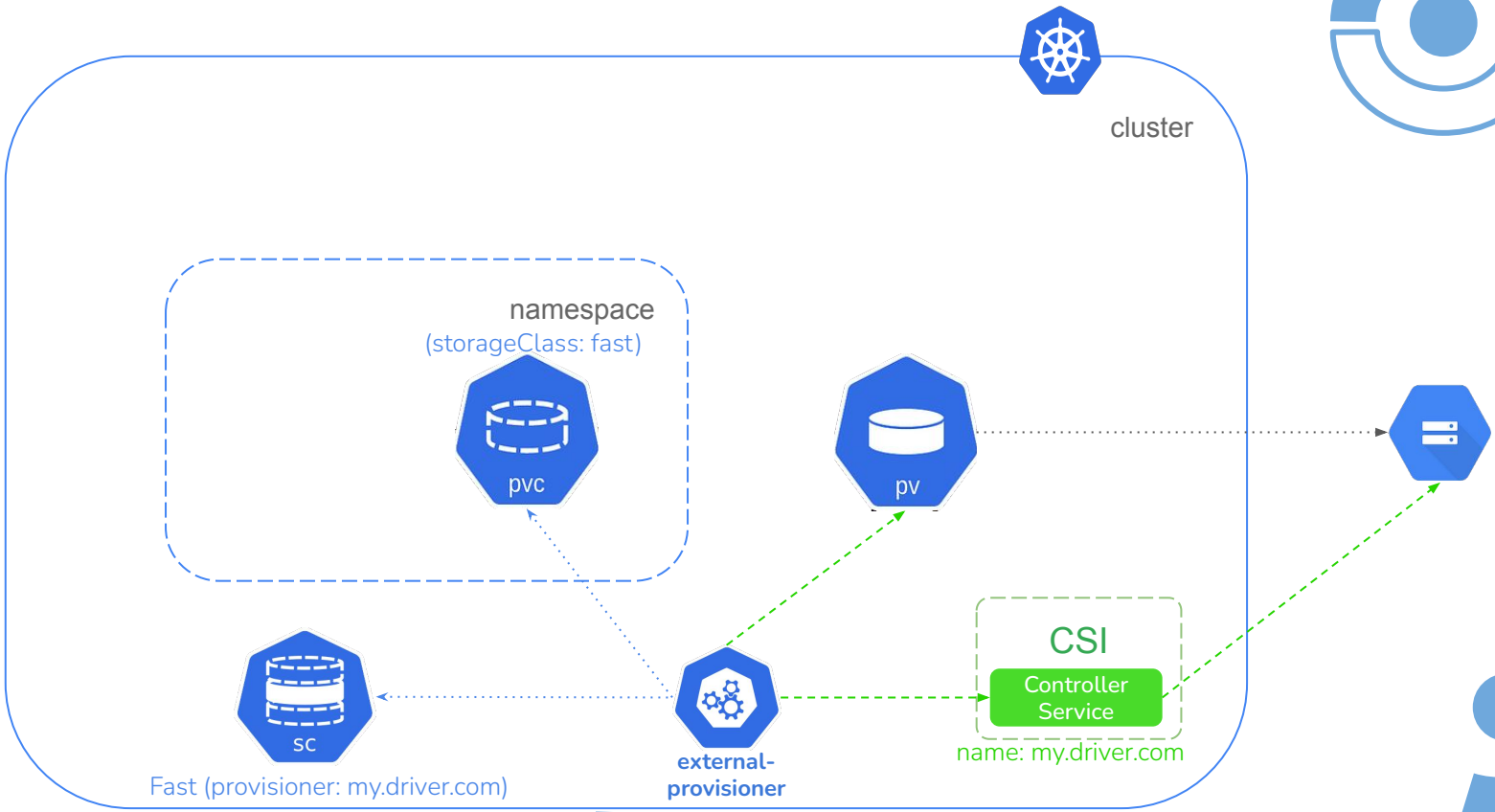


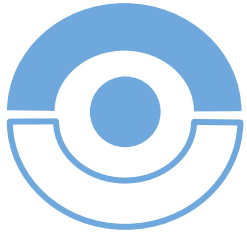
How it works?

How it works e2e



Dynamic Provisioning: out-of-tree



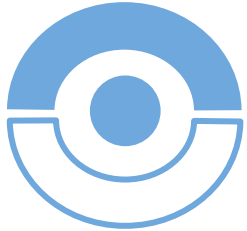


Developing CSI

Developer view point

@parthyadav3105

Your Matras



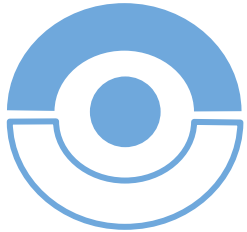
- CSI spec is **source-of-truth**
- kubernetes-csi.github.io/docs/ is your documentation, reference & features checklist.
- Get overview from: arslan.io/2018/06/21/how-to-write-a-container-storage-interface-csi-plugin/
- Follow other csi driver for reference:
  - github.com/kubernetes/cloud-provider-openstack/cmd/cinder-csi-plugin
  - github.com/digitalocean/csi-digitalocean
  - github.com/kubernetes-csi/csi-driver-nfs
  - more list: kubernetes-csi.github.io/docs/drivers.html
- Note: Everyone's implementation is slightly different and they do not implement all the features.

Well known directory structure

It is not recommend, but you will find common directory structure everyone follows:

```
.
├── main.go
└── pkg
    └── driver
        ├── controllerserver.go
        ├── driver.go
        ├── identityserver.go
        ├── nodeserver.go
        ├── server.go
        └── utils.go
```

Capabilities



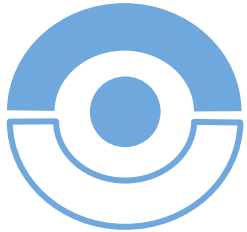
Capabilities are way for CSI driver to tell it's users(CO) that what features are supported.

- `CONTROLLER_SERVICE` (`PluginCapability`)
- `VOLUME_ACCESSIBILITY_CONSTRAINTS` (`PluginCapability`)
- `VolumeExpansion` (`PluginCapability`)
- `CREATE_DELETE_VOLUME` (`ControllerServiceCapability`)
- `PUBLISH_UNPUBLISH_VOLUME` (`ControllerServiceCapability`)
- `CREATE_DELETE_SNAPSHOT` (`ControllerServiceCapability`)
- `CLONE_VOLUME` (`ControllerServiceCapability`)
- `STAGE_UNSTAGE_VOLUME` (`NodeServiceCapability`)
- ..... more

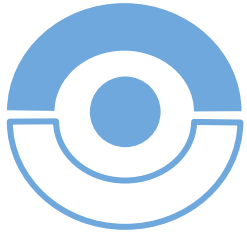
Plugin can optionally expose some of these capabilities based on what it wants to support. Also, some features will also require us to add new sidecar containers or new flags to existing sidecar containers like external-provisioner, attacher, snapshotter, etc.

Code walkthrough

github.com/digitalocean/csi-digitalocean



Debugging your CSI while development.



Container Storage Client(CSC): github.com/rexray/gocsi/csc

Helper csi tools to test CSI driver's gRPC functions.

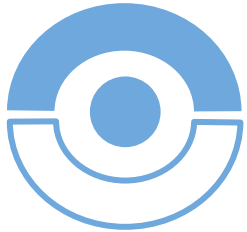
```
$ csc identity plugin-info --endpoint tcp://127.0.0.1:5000

$ csc controller create-volume --endpoint tcp://127.0.0.1:5000 myvolname

$ csc node get-id --endpoint tcp://127.0.0.1:5000

$ csc --help #for more
```

CSIDriver Object

- Simplify driver discovery in k8s.
- Change behaviour of k8s/kubelet towards csi:

```yaml
apiVersion: storage.k8s.io/v1
kind: CSIDriver
metadata:
  name: mycsidriver.example.com
spec:
  attachRequired: true
  podInfoOnMount: true
  volumeLifecycleModes:
    - Persistent
    - Ephemeral
#  more: ... skipped
```

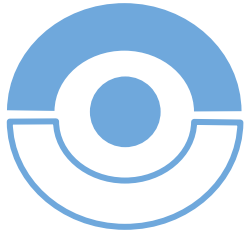
CSINode Object



- Kubectl get csinodes
- Maps csinodes to k8s node with same name.
- CSINode tracks status of node, This is a way for kubelet to tell k8s-controllers the status(availability) of the driver on node. On node register, kubelet calls GetNodeInfo() on csi node plugin
- NodeID from GetNodeInfo() is also stored in CSINode object. This is used in Volume topology aware scheduling by kubernetes if csi implements `VOLUME_ACCESSIBILITY_CONSTRAINTS` .
- Automatically created:

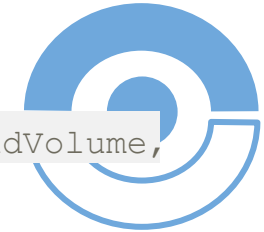
```yaml
apiVersion: storage.k8s.io/v1
kind: CSINode
metadata:
  name: node1
spec:
  drivers:
  - name: mycsidriver.example.com
    nodeID: node-id-82nc9-18cn3-2mcs
    topologyKeys: ['mycsidriver.example.com/regions', "mycsidriver.example.com/zones"]

```

Topology Feature



- Some storage systems have constraints that the volume is not equally accessible from all nodes.
- Topology constraints can be anything racks, regions, zones, profile groups, etc.
- To support topology:
  - `VOLUME_ACCESSIBILITY_CONSTRAINTS` (`PluginCapability`) must be defined by csi driver.
  - Plugin must set `accessible_topology` in `NodeGetInfoResponse`.
  - During `CreateVolume`, csi driver must use the topology information passed through `CreateVolumeRequest.accessibility_requirements`.
- Note: In storage class the user will also want to use `WaitForFirstConsumer`.

Volume Expansion Feature



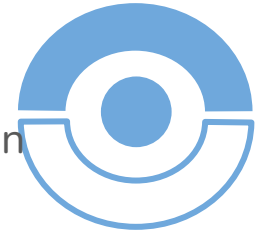
If you plugin support volume resize(expand), i.e. implements `ControllerExpandVolume`, `NodeExpandVolume` then it needs to has `VolumeExpansion` plugin capability.

A volume can be expanded in two modes:

- ONLINE (while mounted and in use)
- OFFLINE (only when not is use/mounted)

ControllerExpandVolume() rpc for performing expansion logic with storage provider. While NodeExpandVolume() rpc is needed if you need to form some node level resize on the mounted path.

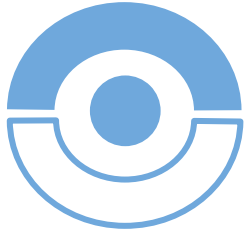
Feature: Volume as Data Source



If CSi driver is capable of creating volume from another volume as source, then Controller plugin can have `CLONE_VOLUME` to tell kubernetes about it.

`CLONE_VOLUME` Capability tell that PVC can be created using another volume as source.

Feature: Snapshots



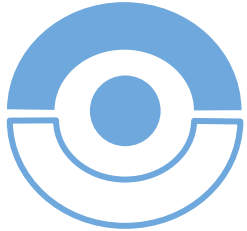
If CSI driver supports Snapshots, i.e. create, delete, list, get, etc for snapshot and create new volume from existing snapshot then:

CSI driver can have `CREATE_DELETE_SNAPSHOT, LIST_SNAPSHOTS` to tell Kubernetes that it is capable of managing Snapshot.

To enable snapshot you will also have to deploy sidecar external-snapshotter along with controller plugin. To will also have to deploy snapshot CRDs, they are not part of core k8s.

Note: When deleting any volume, volume should be independent of Snapshot. Storage Provider should allow deleting volume without deleting any snapshot created from it.

Volumes per Node limit



Sometimes the Node may have limit on number of attachment it can have per Node.

CSI driver can share this info in `max_volumes_per_node` response parameter to . This way Kubernetes/CO can know that there is max limit per node and the scheduler will not schedule pods on that node if the limit is exhausted.

Total Storage Capacity limit at storage provider

Any resource cannot be infinite.

If Storage provider has limit on max storage available in the cloud or you want you keep any such max limit then you can share this information to CO/Kubernetes.

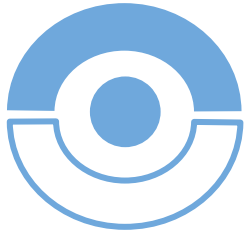
To share storage capacity, controller plugin can have `GET_CAPACITY` (optionals)capability and implement rpc which returns this information.

This capacity is used by Kubernetes to track storage limit and is used during scheduling.

Testing CSI driver

- Unit Testing(sanity tests) github.com/kubernetes-csi/csi-test/pkg/sanity
- Function testing (e2e testing) kubernetes-csi.github.io/docs/functional-testing

CSI drivers are not limited to Kubernetes



- CSI spec is a standard
- The gRPC interface defined in CSI-driver are agnostic to Kubernetes. They do not have to understand Kubernetes. (although there is a way to pass pod information if needed).
- A CSI-driver can be used by any CO(Container Orchestrator), provided:
  - the CO understand how to call CSI driver and
  - CO follows CSI spec.

What we did not covered:

- We did not covered considerations for implementing CSI driver for windows.
- We did not covered Ephemeral volumes.
- Volume Health monitoring
- Metrics from csi driver
- Token requests
- Passing Secrets & credentials from k8s
- Node registration mechanism
- Group controller and Snapshot_Metadata gRPC service in CSI driver.
- How Access Modes in k8s and respective Volume Capability in CSI are handled.
- FSGroup support
- Bind mount
- Retain, Delete
- etc.

Thank You

Fetch latest version of this doc at:
parthyadav.netlify.app/slides/csi-developer-guide

Thank You

@parthyadav3105