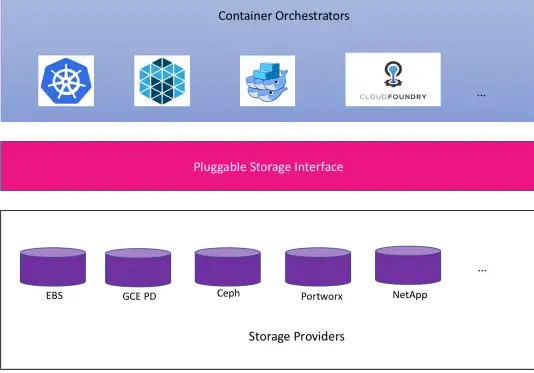
**Container Storage Interface**

**Introduction:**

Container Storage Interface (CSI) is an initiative to unify the storage interface of Container Orchestrator Systems (COs) like Kubernetes, Mesos, Docker swarm, cloud foundry, etc. combined with storage vendors like Ceph, Portworx, NetApp etc. This means, implementing a single CSI for a storage vendor is guaranteed to work with all COs.



Source of the image:

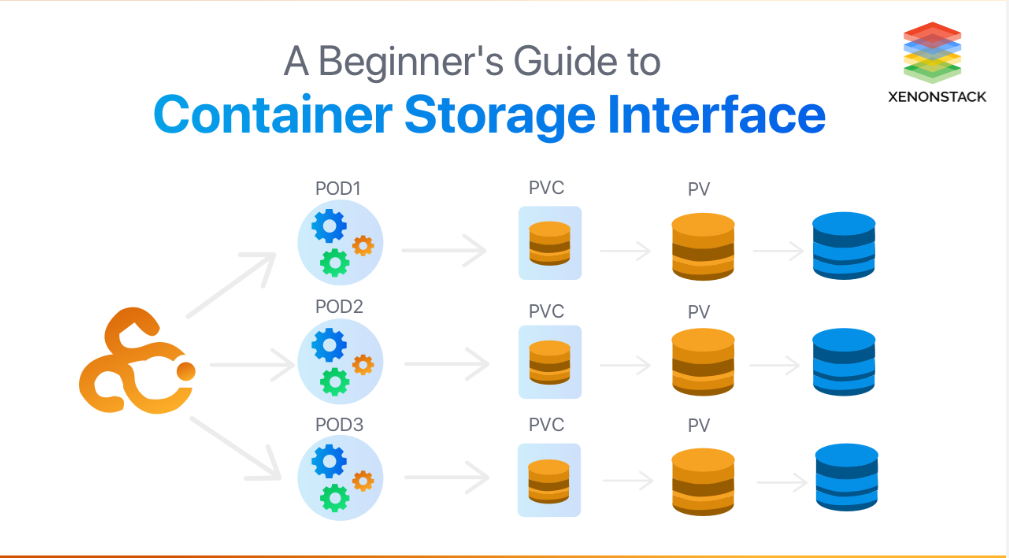
https://schd.ws/hosted\_files/kccnceu18/fb/CloudNativeCon%20EU%202018%20CSI%20Jie%20Yu.pdf

Container are sweeping the datacentre. They’re a lightweight method of virtualising applications and allow for rapid scaling and contain all that’s needed to run processes in a multitude of environments with few dependencie

Project Summary:

Organization: Google  
license: Apache 2.0

**Basic Terminologies related to Container Storage Interface (CSI)**



* **COS -** Container Orchestrator Systems is all about managing the life cycle of Containers, especially in large dynamic environments. Software teams use Container Orchestrator to control and automate many tasks.
* **Volume -** A unit of space that will be made available inside of managed containers via the CSI.
* **Node -** A host where a workload will be running.
* **Workload -** The atomic unit of “work” anticipated by a COS.
* **Plugin -** Plugin points to a service that exposes gRPC endpoints.

**Container Storage Interface Goals:**

* **Interoperability -** The storage vendor can build once a plugin, and that plugin can be used by all the CO that support CSI.
* **Vendor-neutral -** Any CO and storage provider should not terminate it.
* **Focus on specification**
* **Control plane only**

A method of observing, reviewing, and managing the workflow in a cloud-based infrastructure. Click to explore about, Cloud Management and Monitoring Tools

**CSI Connects Kubernetes to 60+ storage products:**

CSI is the container storage interface It is a plugin for Kubernetes and other container orchestrators that allows storage suppliers to expose their products to containerised applications as persistent storage.

At the time of writing, for a wide range of file, block and object storage in hardware and cloud formats.

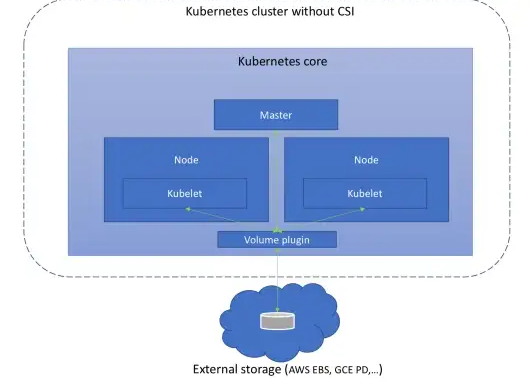
CSI is essentially an interface between container workloads and third-party storage that supports the creation and configuration of persistent storage external to the orchestrator, its input/output (I/O) and advanced functionality such as snapshots and cloning.

CSI replaces plugins developed earlier in the Kubernetes evolution, such as in-tree volume plugins and FlexVolume plugins. Without going into detail, the advantage of CSI is that it has been designed to provide a simplified set of specifications to which storage suppliers can write their plugins, and that they are not dependent on the Kubernetes release cycle.

# Before CSI

The first release of CSI v0.1 was in December 2017. Before CSI was introduced, different COS have their interface that the storage vendor has to implement so that the COS can talk to that SP during the life cycle of volume. The First release of the CSI version is 0.1 was introduced in December 2017. In the case of Kubernetes, volume plugins were serving the storage need for container workload. The first problem was in Kubernetes itself, we need to write a driver as a source vendor and put it inside the Kubernetes code, and what that meant is that as a storage vendor you could not one fix any issues

Which you have to wait for kubernetes release you are bound for release process of kubernetes.



**Source of the image:**

**https://miro.medium.com/max/720/1\*xsxMjNGNd6C\_L9Vd3zYMWA.webp**

The volume plugins are part of the COs core as shown in the picture above. Due to this, it has the following drawbacks mentioned in the CSI design document:

* Volume plugin development is tightly coupled and dependent on Kubernetes releases.
* Kubernetes developers/community are responsible for testing and maintaining all volume plugins, instead of just testing and maintaining a stable plugin API.
* Bugs in volume plugins can crash critical Kubernetes components, instead of just the plugin.
* Volume plugins get full privileges of kubernetes components (kubelet and kube-controller-manager).
* Plugin developers are forced to make plugin source code available, and can not choose to release just a binary.

**Object storage:**

Object storage also called object-based storage. Which describe an approach to addressing and manipulating discrete unit of storage called objects like files objects contain data, but unlike files, objects are not organized in a hierarchy that means every object exists at the same level in a flat address space called the storage pool and one object can not be placed inside another object. An object is designed for unstructured data such as media, documents, logs, backup, etc. The most popular cloud object storage is AWS S3. Useful for automating and streamlining data storage. Use cases -

* Storage for unstructured data like music, image, etc.
* Storage for backup files, log files.
* Hybrid cloud storage.
* Disaster Recovery.

**Benefits of object storage:**

* Scalability: Adding data forever, there is no limit. Security and compliance
* Flexible management
* Efficiency

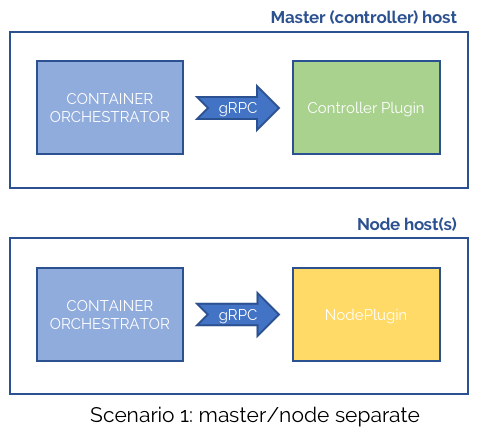
Cloud Governance is a set of rules. It applies specific policies or principles to the use of cloud computing services. Click to explore about, Cloud Governance Challenges.

**Storage vendor/3rd-party external component:**

This is a vendor specific implementation. Each vendor should implement their respective APIs into gRPC service functions. E.g. Implementation of GCE PD,Ceph, etc. They too consists of three sub-component.

Architecture:

CSI is divided up into a number of architectural designs that determine how plugins are deployed. The architectures match typical CO implementations that have master and node hosts. The three scenarios are:



Source of the Image:

https://content.architecting.it/wp-images/post\_3cad\_image3.png

* Master/Node with separate plugins for both controller and node functions.
* “Headless” where the plugins still run separately for controller and node but only run on the node hosts.
* Headless combined, where a single plugin provides the controller and node capabilities together.

**Reference:**

https://medium.com/google-cloud/understanding-the-container-storage-interface-csi-ddbeb966a3b

https://kubernetes.io/blog/2023/01/20/security-behavior-analysis/

https://www.architecting.it/blog/container-storage-interface/