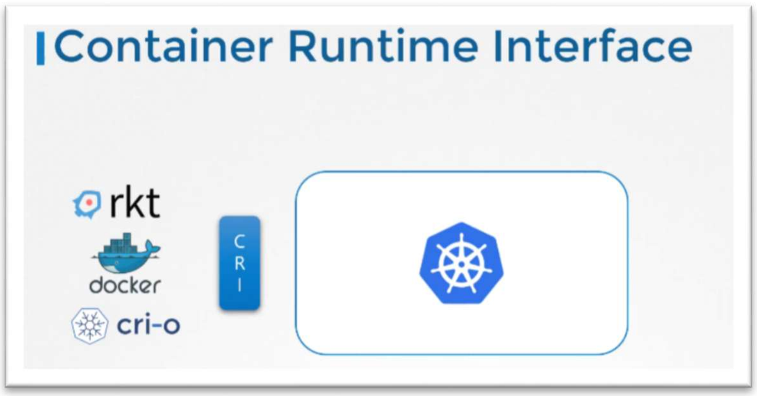
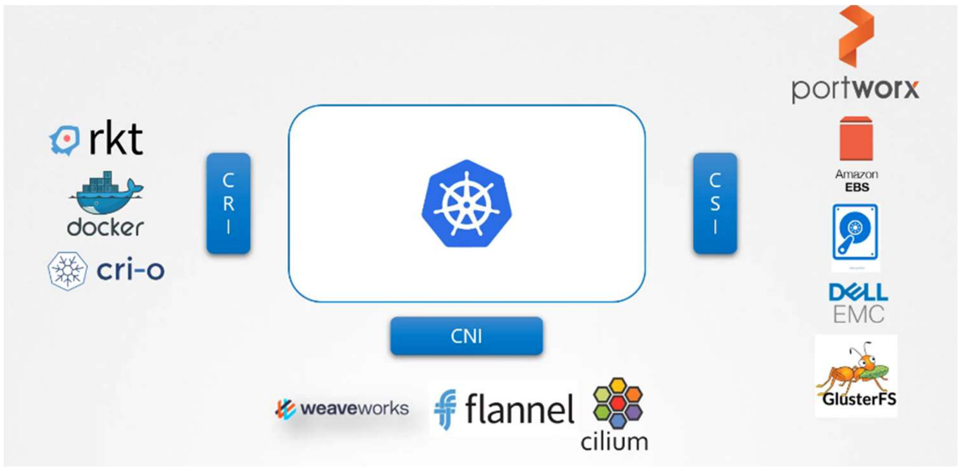
Let us now look at Container Storage Interface. In the past, Kubernetes used Docker alone as the container runtime engine, and all the code to work with Docker was embedded within the Kubernetes source code. With other container run times coming in, such as Rocket and CRI-O, it was important to open up and extend support to work with different container run times, and not be dependent on the Kubernetes source code. And that's how container runtime interface came to be.



The Container Runtime Interface is a standard that defines how an orchestration solution like Kubernetes would communicate with container run times like Docker. So in the future, if any new container runtime interface is developed, they can simply follow the CRI standards and that new container runtime would work with Kubernetes without really having to work with the Kubernetes team of developers or touch the Kubernetes source code.

Similarly, as we saw in the networking lectures to extend support for different networking solutions the container networking interface was introduced. Now any new networking vendors could simply develop their plugin based on the CNI standards and make their solution work with Kubernetes.

And as you can guess, the Container Storage Interface was developed to support multiple storage solutions. With CSI, you can now write your own drivers for your own storage to work with Kubernetes. Port Works, Amazon EBS, Azure Desk, Dell EMC Isilon, PowerMax Unity, XtremIO, NetApp, Nutanix, HPE, Hitachi, Pure Storage.Everyone's got their own CSI drivers.

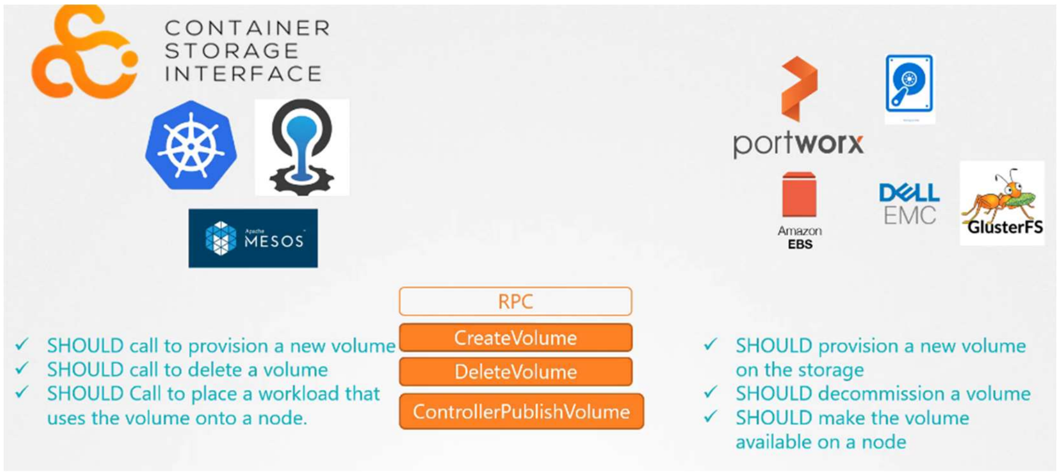


Note that CSI is not a Kubernetes specific standard. It is meant to be a universal standard, and if implemented allows any container orchestration tool to work with any storage vendor with a supported plugin. Currently, Kubernetes Cloud Foundry and Mesos are on board with CSI.

So here's what the CSI kind of looks like. It defines a set of RPCs, or remote procedure calls, that will be called by the container orchestrator, and these must be implemented by the storage drivers.

For example, CSI says that when a pod is created and requires a volume, the container orchestrator, in this case Kubernetes, should call the create volume RPC and pass a set of details such as the volume name. The storage driver should implement this RPC and handle that request and provision a new on the storage array, and return the results of the operation.

Similarly, container orchestrator should call the delete volume RPC when a volume is to be deleted, and the storage driver should implement the code to decommission the volume from the array when that call is made. And the specification details exactly


CONTAINER STORAGE INTERFACE
MESOS
portworx
Amazon EBS
DELL EMC
GlusterFS
RPC
CreateVolume
DeleteVolume
ControllerPublishVolume
✓ SHOULD call to provision a new volume
✓ SHOULD call to delete a volume
✓ SHOULD Call to place a workload that uses the volume onto a node.
✓ SHOULD provision a new volume on the storage
✓ SHOULD decommission a volume
✓ SHOULD make the volume available on a node