



BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

DYNAMIC STORAGE PROVISIONING IN KUBERNETES

Divya Vijayakumar, Arun Kandasamy
MSys Technologies, LLC



About MSys Technologies

MSYS TECHNOLOGIES

PRODUCT ENGINEERING SERVICES
AND DIGITAL TRANSFORMATION
PARTNER

BAY AREA UNICORNS



ENTERPRISES



AWARD



TOP FASTEST GROWING
STORAGE COMPANIES' FOR
TWO CONSECUTIVE YEARS



OUR WW STRENGTH 800 AND GROWING



MSYS OFFICES - USA | INDIA | VIETNAM

Agenda

- 01 CSI Storage Implementation in Kubernetes
- 02 Leveraging Persistent Volume in Containers
- 03 Volume Management and Monitoring
- 04 What next for Storage Provisioning in Orchestrators



DYNAMIC VOLUME PROVISIONING

Concept of Dynamic Provisioning

- Dynamic Provisioning of Volume allows storage volumes to be created on demand.
- Previously, cluster management was a manual process by contacting storage provider to create new storage volumes. Subsequently, Persistent Volume objects were created to represent them in Kubernetes.
- Dynamic Provisioning eliminated the need for cluster managers to pre-provision storage. Instead, it automatically provisions storage when requested by users.
- Persistent Volume types are implemented as plugins. Kubernetes currently supports many plugins like: AWSElasticBlockStore, AzureDisk, CSI, FlexVolume, NFS, Cinder, Glusterfs, etc.



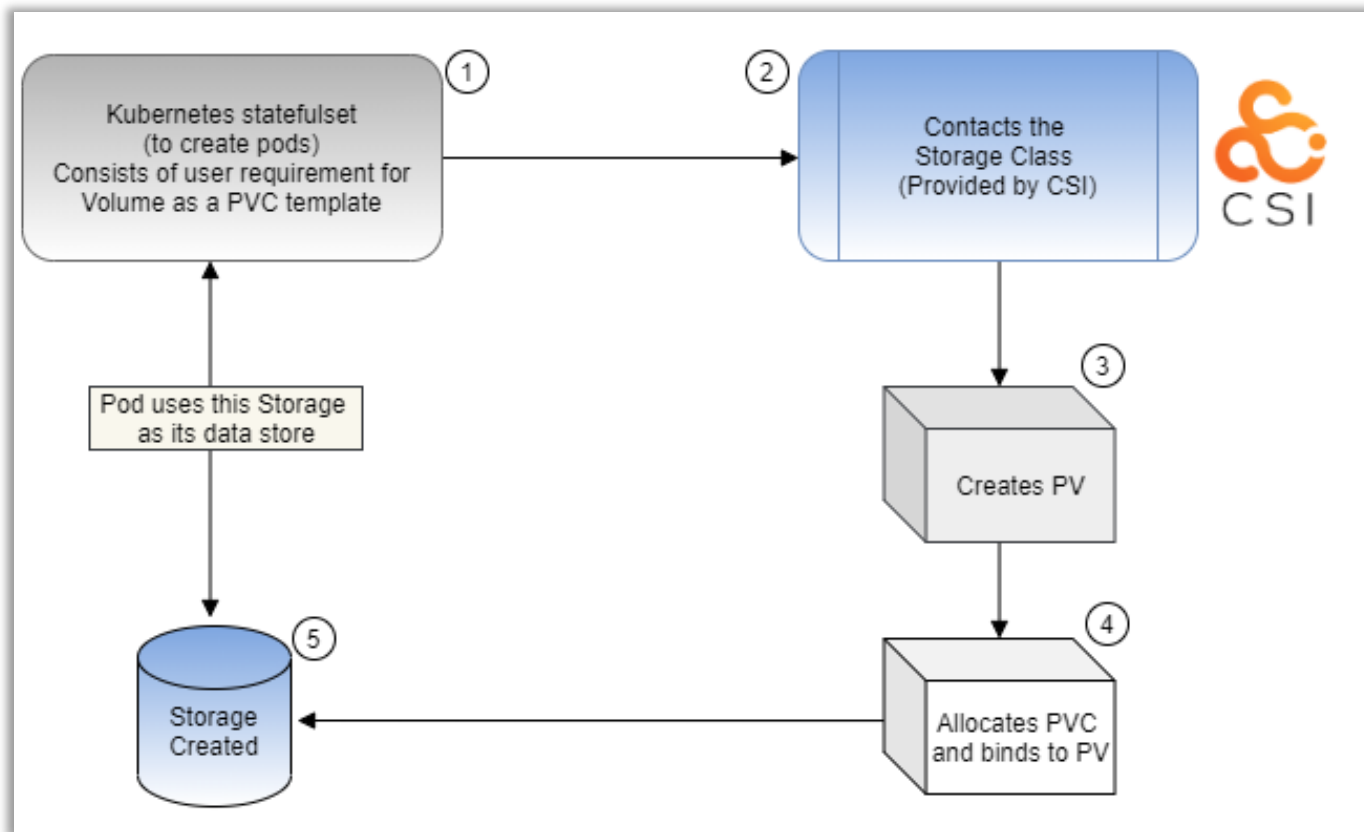
Concept of Dynamic Provisioning

Volume Type	Purpose
Storage Class	A way of provisioning different classes of storage to the admins. Each SC contains the fields ' <i>provisioner</i> ' and ' <i>parameters</i> ', which are used when a PV needs to be dynamically provisioned.
Persistent Volume (PV)	The kind storage which is either provisioned by an Admin or dynamically provisioned using Storage Classes (SC).
Persistent Volume Claim (PVC)	A request for storage by a user. Claims can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

SC-PV-PVC Relationship

- We have implemented the concept of statefulsets in Kubernetes for data-generating applications, which will contain a PVC request.
- This will contact the Storage Class, which in turn, will provision a PV and the PVC gets bound to this PV, inorder to use this storage.
- This storage will, ultimately, be used by the pod/deployment which requested for the PVC for its data.
- This is explained in a simple relationship diagram next.

SC-PV-PVC Relationship

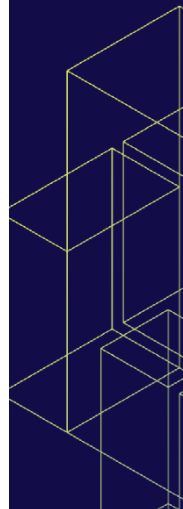




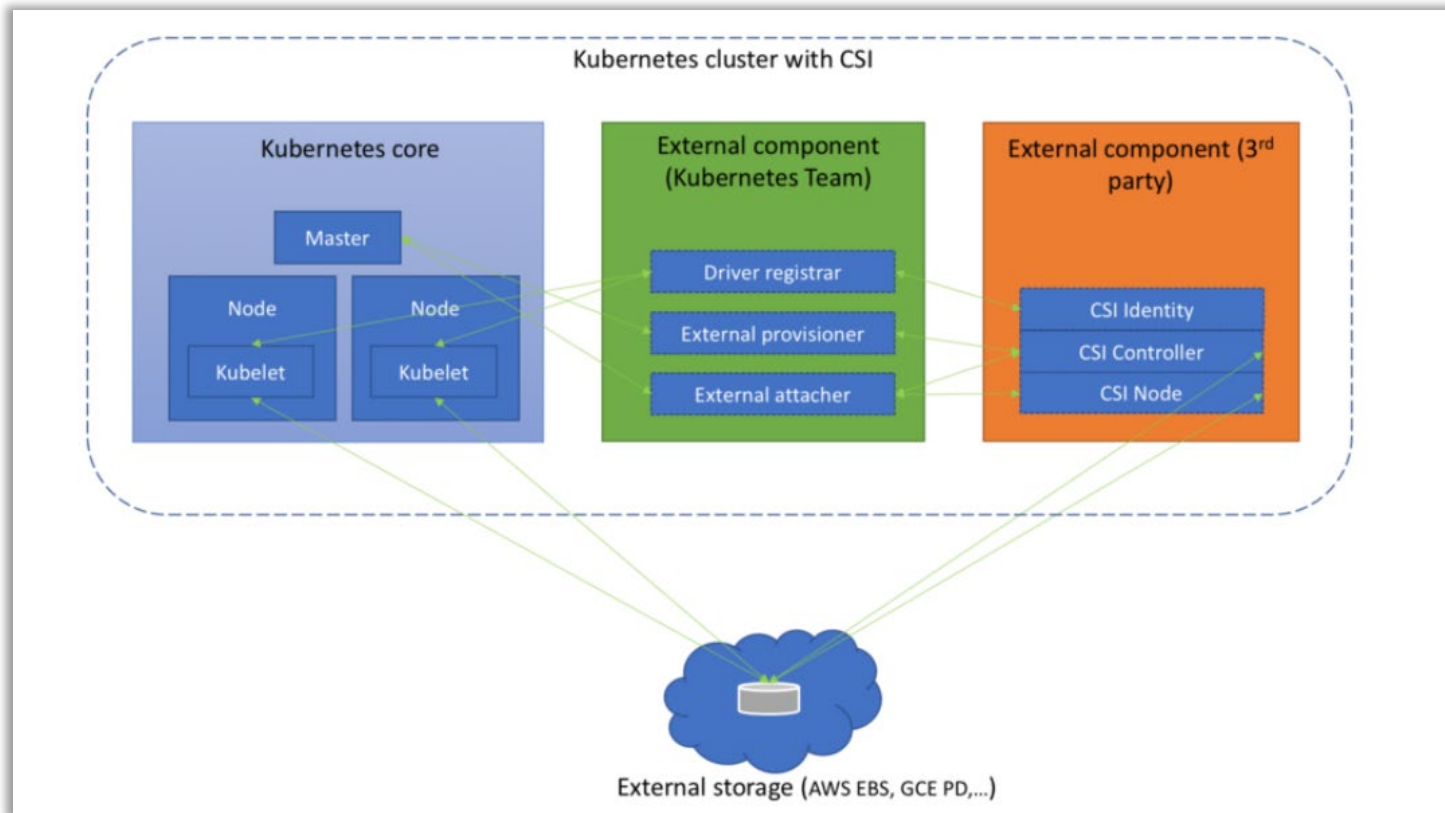
CSI PLUGIN

About CSI

- Container Storage Interface (CSI) is a plugin for container orchestrators to allow storage providers to expose their products as persistent storage in containerised applications.
- CSI is essentially an interface between container workloads and third-party storage. It supports the creation and configuration of persistent storage external to the orchestrator, its input/output and advanced functionalities.
- It is an initiative to unify the storage interface of Container Orchestrator Systems (COs) like Kubernetes, Mesos, Docker Swarm, Cloud Foundry, combined with storage vendors like Ceph, Portworx, NetApp.
- Implementing a single CSI for a storage vendor is guaranteed to work with all COs.



CSI Architecture





PV-PVC IMPLEMENTATION

PVC in Kubernetes Manifest

A Persistent Volume Claim is simply a Volume requirement from the user. There are different ways to create and deploy it in Kubernetes.

- One way is to do it via CLI command : *kubectl create -f pvc.yaml*
- Another way is to integrate it along with the Statefulset manifest file.

A small block of example is shown here.

```
volumeClaimTemplates:  
- metadata:  
  name: pvcvolume  
  spec:  
    accessModes: [ "ReadWriteOnce" ]  
    storageClassName: "my-sc-1"  
    resources:  
      requests:  
        storage: 10Gi
```

PVC in Kubernetes Manifest

- As explained before, a PVC has to be 'bound' to its PV from StorageClass. When the status is '**Bound**', it means the volume is good to go and ready to use.
- The Volume name mentioned can be checked on the storage array where our storage block remains. The size created and the size remaining can be viewed on GUI.

```
[kube@k8s-master1 ~]$ kubectl get pvc -n divya-vijay
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS
postgresdb-postgres-0	Bound	pvc-30a0c7c6-53d1-4168-81bb-ea14e3980e6c	30Gi	RWO	my-sc-1
pvcvolume-mq-0	Bound	pvc-3072fa60-8572-4087-8c60-438ca8c28cf9	10Gi	RWO	my-sc-1

```
[kube@k8s-master1 ~]$
```

Kubernetes Manifest

- It is a general practice to attach the array volume to data-generating applications.
- These compound applications which need external volume can be created as Statefulsets in Kubernetes, or a Deployment, rather than as a single pod. This helps in managing the lifecycle of the pods better.
- Some examples of such applications are a Database or a Processor.

```
[kube@k8s-master1 /]$ kubectl get sts -n divya-vijay
```

NAME	READY	AGE
mq	2/2	2m28s
postgres	1/1	4m50s



WHY CSI

Other Methodologies

- To avoid manual creation of PV and PVC, like in static method, we opted for dynamic provisioning.
- There are multiple approaches to this:
 - CSI drivers
 - Hostpath provisioner
 - NFS provisioner, etc.
- We have also tried and tested with Hostpath method and arrived at CSI as the best approach for our work.

Other Methodologies

- StorageClass, which runs in the cluster, keeps monitoring for any PVC creation.
- When a new PVC request comes in, StorageClass creates PV for matching PVC at run time, and binds the allocated volume for the pod.
- In our scenario, we pre-created storage class named “*hostpath*” to define hostpath provisioner for the use and *hostpath parameter* was set as */incoming_data/*.
- This approach is about storing data on the host directly, hence, it is quick and sought-after for dynamic provisioning. But, its inability to handle the huge amount of data getting stored in the hostpath, proved to be a great drawback.
- This caused the storage machine extremely lagging and hindered it to perform tasks to its full potential.

Why We Opted for CSI

- Owing to the drawback from other methods, we opted for external storage provided via CSI driver.
- When provisioning volume for a huge framework with vast amount of data and complexity that needs to be orchestrated using Kubernetes, CSI driver appears as the best approach to manage the data store.
- Storage provisioned via CSI drivers was external to the host and was easy to incorporate into the Kubernetes manifest file.
- This was like creating any other resource on K8s - like pod, service or volume, and linking it to the storage array present elsewhere.



ORCHESTRATOR FEATURES

Kubernetes Features

Self-Healing nature of Kubernetes cluster

A 5-node multi-master Kubernetes cluster setup (3M + 2N) was created.

If any one of the pods were to go down, Kubernetes will instantly re-deploy it, matching it to desired state of the applications.

Rolling Update

- A container image used for Deployment can be updated from one version to another without any downtime.
- Rolling update for image with zero downtime:
kubectrl set image deployment/postgres-deployment postgres=postgres:latest
- Similarly, if there is some problem with updated image, rollback to 2 revisions based on stability with:
kubectrl rollout history deployment/postgres-deployment --revision=2

Kubernetes-CSI Feature

Raw block volume support for CSI

- Kubernetes supports two volumeModes of PersistentVolumes: *Filesystem* and *Block*.
- The default volumeMode for a PersistentVolumeClaim is “Filesystem”. If a raw block volume is desired, volumeMode needs to be set to “Block”.

```
volumeClaimTemplates:
- metadata:
  name: pvcvolume
  spec:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: "my-sc-1"
    resources:
      requests:
        storage: 10Gi
    volumeMode: "Block"
```



VOLUME MANAGEMENT

Volume Monitoring

Volume Management in Orchestrators can be done by implementing any one of the Monitoring tools along with the setup.







Prometheus

- This is a straight-forward monitoring tool widely used for Kubernetes designs. Any combinations of metrics and monitoring tools can be applied to orchestrator setup.
- *Prometheus* can be run as a Deployment and *node-exporters* can be as a DaemonSet along with *nsenter* implementation. This will start populating the metrics of the minion nodes.

Volume Monitoring

Volume Expansion

One other way to manage storage is to use the concept of Volume Expansion by patching up additional volume with existing PVC memory.

<input type="checkbox"/> NAME ^		APPLICATION	DATA	SIZE	IOPS	TOTAL MiB/s	LATENCY (MS)
<input type="checkbox"/>  pvc-24fb06c2-d1dc-4... default	● Offline	 SQL Server	48.8 MiB <div><div></div></div>	300 GiB	0	0	r 0 w 0
<input type="checkbox"/>  pvc-3072fa60-8572-4... default	● Online	 SQL Server	0 B <div><div></div></div>	10 GiB	0	0	r 0 w 0
<input type="checkbox"/>  pvc-30a0c7c6-53d1-4... default	● Online	 SQL Server	0 B <div><div></div></div>	30 GiB	0	0	r 0 w 0

Volume Monitoring

Volume Expansion

If the allotted Volume is about to be exhausted, the mentioned PVC can be “expanded” by mentioning this in storage class,

allowVolumeExpansion: true

Example: From 10Gi, increase to 64Gi by,

kubectl patch pvc/pvcvolume --patch '{"spec": {"resources": {"requests": {"storage": "64Gi"}}}}'

persistentvolumeclaim/pvcvolume patched



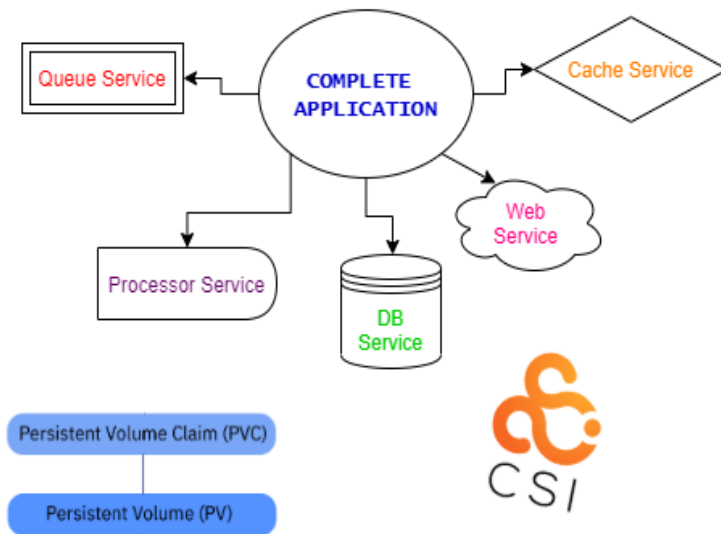
THE FUTURE

What next

Future Roadmap

The future of Volume Provisioning in Kubernetes lies in leveraging the key components of Storage Architecture.

Dissociating the applications from a traditional, Monolithic design and empowering them to run individually, but still bringing them together in a cluster to perform as **ONE** with some components even supporting externally like a Volume store, is the new vision of all Orchestrators.



References

1. <https://kubernetes.io/docs/concepts/>
2. <https://medium.com/google-cloud/understanding-the-container-storage-interface-csi-ddbeb966a3b>
3. <https://blog.argoproj.io/volume-monitoring-in-kubernetes-with-prometheus-3a185e4c4035>



About Authors



DIVYA VIJAYAKUMAR

Divya holds prominent experience in DevOps domain, specializing in Docker and Kubernetes. At Msys, she is responsible for orchestrating and managing compound applications. In her prior experience with Comcast communications, she has worked on crucial frameworks development with CICD pipeline and Python programming.



ARUN KANDASAMY

Arun holds over a decade of distinguished experience in Embedded and storage domain with strong exposure to Docker and Kubernetes. At Msys, he is responsible for containerizing complex application, which holds external/persistent storage as well as monitoring metrics using Prometheus.



QUESTIONS?





THANK YOU :)