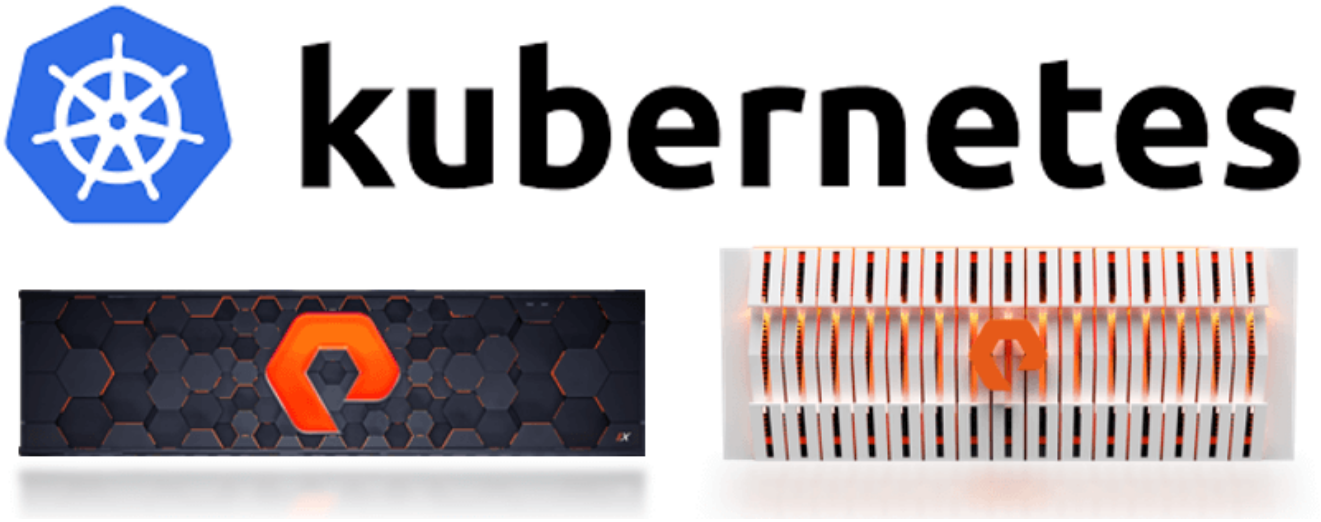
The Kubernetes Storage Ecosystem Explained

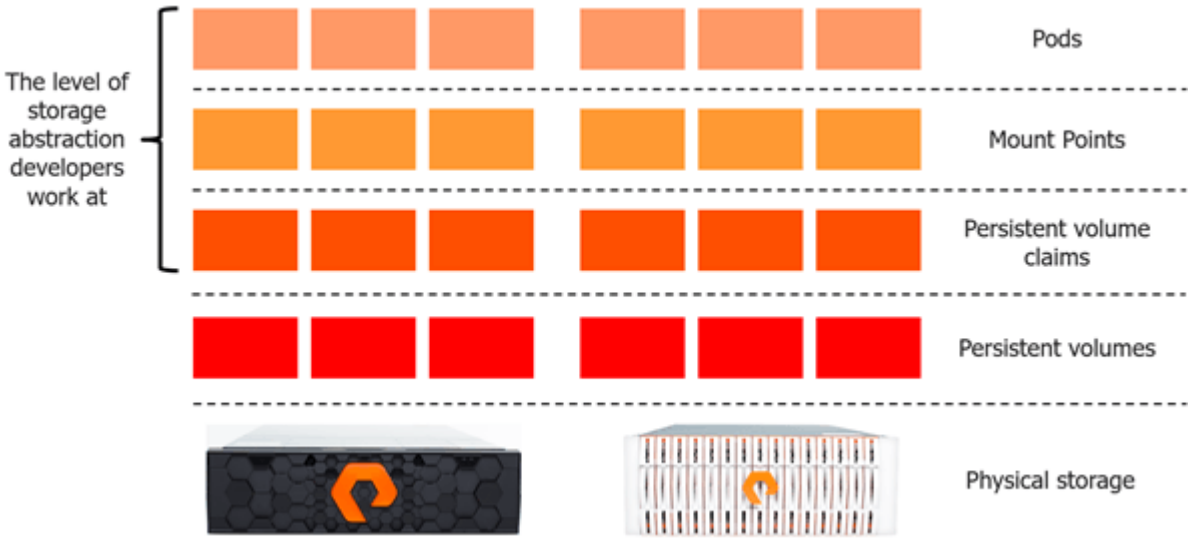


Pods, Volumes, Persistent Volume Claims, Persistent Volumes and Storage Classes

Containers run inside pods which are always co-scheduled on the same host. Pods are immutable, if a pod dies it is never resurrected. To make a VMware comparison; in an ESXi cluster, the unit of scheduling is a virtual machine, in a Kubernetes cluster the unit of scheduling is a pod. The “Touch point” for storage in a pod is a volume, and volume(s) always have associated mount point(s). First and foremost Kubernetes is a platform for developers, that is designed to decouple the application from the infrastructure it runs on.

How Developers View Storage when Working With Kubernetes

In the world of Kubernetes, a developer’s view of the storage world is based around persistent volumes claims:



Persistent volume claims have three attributes:

- name
- storage class
- size

This yaml excerpt illustrates how a persistent volume claim is created:
[crayon-6514db26b8438518343863/]
A persistent volume is the entity that maps directly onto the storage platform. If a persistent volume exists that the persistent volume claim can be satisfied from, Kubernetes will use it to satisfy the volume claim. Otherwise, a persistent volume will be created if the storage class is configured for dynamic storage provisioning. Below is an example of the yaml for block and file storage classes:

[crayon-6514db26b8441587356447/]
The association of a persistent volume claim with a persistent volume is known as persistent volume claim binding.

The Storage Provisioner

The provisioner abstracts out the mechanism of creating/deleting volumes across the different storage types used in a Kubernetes cluster. One instance of the provisioner should exist per storage type. This is not to be confused with the FlexVolume driver which mounts the volume.

Storage Classes

A storage class in conjunction with a provisioner allows storage to be provisioned dynamically. The exact storage classes available will vary from storage platform to another. For the purposes of this blog post there are two types of storage class:

- A block storage class
- A file / unstructured data storage class

Storage classes are developer visible objects and abstract out the actual provision-er implementation.

Static Versus Dynamic Provisioning

Most storage provisioning mechanisms can be either static or dynamic. Under static provisioning schemes, the Kubernetes administrator creates persistent volumes for developers to consume via persistent volume claims. With dynamic provisioning schemes, persistent volumes are created dynamically in response to the persistent volume claims asked for. Rules associated with the storage class determine if and how persistent volumes are created for dynamic storage provisioning.

Statefulsets

StatefulSets were first introduced with Kubernetes 1.9 are a nuanced topic worthy of an entire blog post in its own right. Suffice it to say that StatefulSets provide an elegant means of addressing the challenges of scaling state-full applications:

- Scaling out applications via ReplicaSets results in the storage being replicated
- Some state-full multi pod applications require that pods are started and stopped in a specific order, namely applications with controller / worker pod architectures
- Assigning a unique identifier to a collection of pods that form an application

What Is Important in The Underlying Kubernetes Storage Platform?

In the days of yore when spinning disk-based storage platforms ruled the data center, the general rules of thumb were:

- Avoid Raid 5 for any workload that is more than 30% write intensive
- Avoid Raid 5 for database transaction / redo logs
- There was a whole raft of tricks around short stroking and elevator sort algorithms used to minimize disk head movement

Fast forward to the year 2018 and the role of spinning disk-based storage is now relegated to backup, recovery and tier 2 type use cases. However, the slew of flash storage platforms based on retrofit architectures designed for the spinning-disk era means that for certain legacy products, the lore around raid levels still applies. For retrofit platforms that provide integration with Kubernetes, LUNs can be created for different raid levels along with labels for exposing this detail to the users of the platform. Herein lies the problem, the *raison d'être* of Kubernetes, is that first and foremost it is a developers platform, a point made clear by Brendan Burns, one of Kubernetes founding engineers in an excerpt from this article:

> *From the beginning of the container revolution two things have become clear: First, the decoupling of the layers in the technology stack are producing a clean, principled layering of concepts with clear contracts, ownership and responsibility. Second, the introduction of these layers has enabled developers to focus their attention exclusively on the thing that matters to them — the application.*

As such, hardware details should be abstracted away from developers as much as possible. A provisioner that furnishes dynamic provisioning removes the need persistent volumes to be setup manually and provides the developers with the most friction-less storage-as-a-service experience possible.

The other main decision point to consider is hyper-converged versus converged dis-aggregated

infrastructure. The hyper-converged world of rigid compute and storage blocks works when the set of applications hosted all have the same compute and storage requirements. But when the application landscape is composed of applications with varying compute and storage requirements, this use case requires the flexibility of being able to scale compute and storage independently of one another. Simply put, Kubernetes requires the flexibility that only dis-aggregated infrastructure can offer.

In summary, what really matters for a storage platform that provides Kubernetes integration, is its ability to provide the pure developer experience that Kubernetes was designed for in the first place.

Further Reading

For further information on how Pure Storage can help deliver a developer-centric storage experience for Kubernetes, the following two articles are recommended reading:

- Delivering Container Storage-As-A-Service
- Announcing The General Availability of Pure Service Orchestrator