IBM Cloud Private Storage

IBM Cloud

Persistence and Kubernetes

IBM Cloud Private and storage

Making your storage decision

Kubernetes Persistent Storage

PV, PVC, Storage Class and Beyond

Persistent Storage Overview

Storage that persists beyond the lifecycle of the container allows workload to achieve state

Docker, containers and persistence

Kubernetes, pods and persistence

Terminology and the persistent lexicon

Value of the persistence solution

It is about STATE not STORAGE

We strive to simplify our configuration management by moving towards immutability

Containers

Containers are purposefully not Virtual Machines and do not carry the burden of a VMs resource management

Stateless

Applications that containerize the best tend to be stateless with a small number of configuration parameters required for personalization

State

Some services require backing storage to maintain the current operating state such as a database or an applications that persist transactions

Pervasive

As orchestration is used to place containerized workload throughout the private cloud it becomes important that the persistent storage is equally mobile

Docker and Storage

The **storage backend** in **Docker** handles reads and writes and supports key features such as **layering**

Backends

- The backend you chose (or that is chosen for you) depends heavily on the architecture of your nodes and version of Docker
- Understand your backend when troubleshooting
- Pay attention to backing filesystems

overlay2

- Built upon the original AUFS
- This is likely the first choice if available for your Linux distribution

Device Mapper

- May be preferred for CentOS (and derivatives)

https://success.docker.com/article/compatibility-matrix

Methods for persisting data

There are different methods for persisting data for a workload

Data can be stored using another cloud service hosted either in the private cloud or elsewhere in the enterprise (this then becomes a stateless workload)

Configuration and settings can be persisted using **ConfigMaps** and **Secrets**

A **Persistent Volume Claim** can be made by the developer against a **Persistent Volume** that a storage administrator has previously created

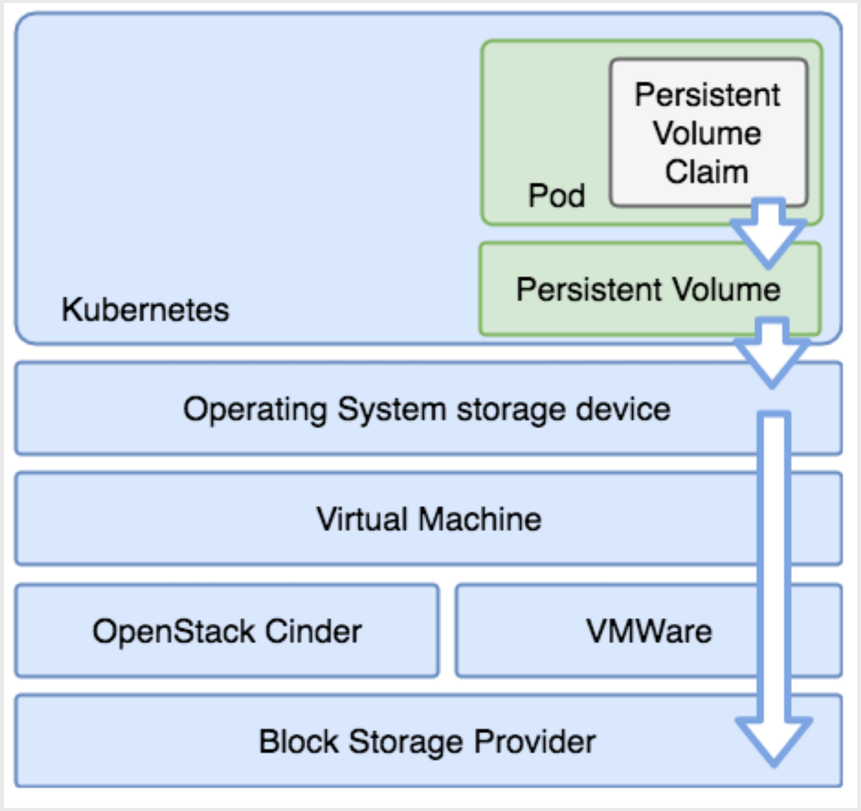
A **Persistent Volume Claim** make a dynamic request for storage via a previously defined **Storage Class** that supports a specific storage provider and set of use cases

Persistent Storage

Persistent Volume is a storage resource within the cluster. PVs have a lifecycle independent of any individual pod that uses it. This API object encapsulates the details of the storage implementation or cloud-provider-specific storage system.

A **Persistent Volume Claim** is a storage request, or claim, made by the developer. Claims request specific sizes of storage, as well as other aspects such as access modes.

A **StorageClass** describes an offering of storage and allow for the dynamically provisioning of PVs and PVCs based upon these controlled definitions.



Volume and Claim Lifecycle

The **Reclaim** policy informs the cluster what to do with the volume after it has been released from its claim
- **Retain** allows for the manual reclamation of the storage asset. The PVC is deleted but the PV remains.
- **Delete** reclaim policy removes both the objects from within the cluster, as well as the associated storage asset from the external infrastructure.
- **Recycle** has been deprecated and should no longer be used.

Access Modes define how volumes can be mounted in the manner supported by the storage provider

ReadWriteOnce (RWO) can be mounted as read-write by a single node and pod
- **ReadOnlyMany (ROX)** can be mounted read-only by many nodes and pods
- **ReadWriteMany (RWX)** can be mounted as read-write by many nodes and pods

NOTE: Reclaim policy and Access Modes may be defined differently by each storage provider implementation.

Storage Class Example

Dynamic Provisioning

Storage classes can map to a "provisioner" to dynamically provision persistent volumes based on the volume claim requests coming in as users deploy workloads and services.

Map to Storage

Some storage providers support the dynamic provisioning and abstract details so the developer doesn't need to take multiple steps to acquire, bind, and claim storage for their services.

Default Storage Class

A default storage class can dynamically provision storage when a storage class is not specified.

Claiming 'gold' storage when deploying an app

```
apiVersion: v1
kind:
PersistentVolumeClaim
metadata:
 name: mypvc
 namespace: testns
spec:
 accessModes:
 - ReadWriteOnce
 resources:
   requests:
     storage: 100Gi
 storageClassName: gold
```

Create 'gold' storage class, mapped to glusterfs

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
 name: gold
provisioner: kubernetes.io/glusterfs
parameters:
 resturl: "http://glusterIP:8080"
```

Change default storage class to GlusterFS

```
#get the names, see which is default
kubectl get storageclass

#set current default to "false"
kubectl patch storageclass default-class-name -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'

#set your desired default to "true"
kubectl patch storageclass gold -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Container Storage Interface (Beta as of K8s 1.10)

The Kubernetes **Volume Plugin System** makes it possible to connect persistent storage to the containers that require it and providing this function dynamically in some cases. **Container Storage Interface** (CSI) makes installing new plugins “as easy as deploying a pod”.

CSI enables third-party storage providers to develop solutions without the need to add to the core or “in-tree” of the Kubernetes codebase.

The CSI specification was developed with collaboration across Container Orchestration Systems (including K8s, Mesos, Docker and Cloud Foundry).

CSI plugin providers provide implementation instructions for deploying via their plugin.

Current in-tree volume plugins (including those based upon FlexVolume) will eventually migrate to CSI.

Persistent Storage and ICP

Storage Providers

IBM Cloud Private Storage Providers

Kubernetes and IBM Cloud Private offer many options for managing persistent storage within the cluster. ICP features the following:

GlusterFS enterprise grade of storage to K8s pods offering ease of configuration, scaling, encryption support, replication, striping and dynamic provisioning.

vSphere Cloud Provider (vSphereVolume Plugin) gives access to enterprise grade storage (vSAN, VMFS, Vvol) that is native to and already supported by the VMware infrastructure.

IBM Spectrum Scale for solutions not hosted in VMware provides direct access to IBM block storage via dynamic provisioning.
(See http://www.redbooks.ibm.com/redpapers/pdfs/redp5533.pdf)

NFS provides a versatile and easy to use method of getting persistent storage to pods that is already available in most customer environments.

HostPath is ideal for testing persistence in non-production environments.

Ceph (Rook) is an industry proven option that can provide several storage options along with persistent volumes for Kubernetes

NFS Persistent Storage Architecture

Easy to implement and versatile solution that can leverage existing NAS storage services within the enterprise

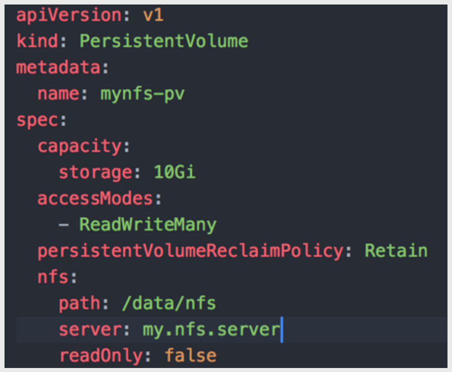
NFS PVs support the following access modes:

- **ReadWriteOnce** - the volume can be mounted as read-write by a single node
- **ReadOnlyMany** - the volume can be mounted read-only by many nodes
- **ReadWriteMany** - the volume can be mounted as read-write by many nodes

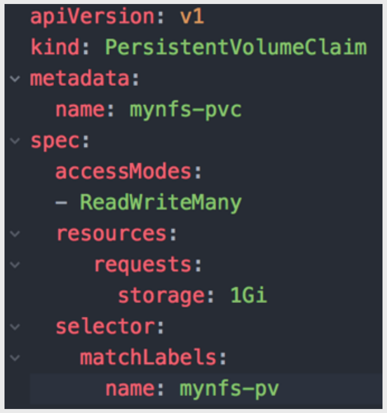
Supports all reclamation policies

Does not natively support dynamic provioning**

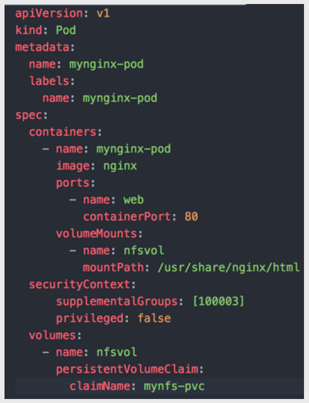
- Create the NFS share and authorize access
- Define your Persistent Volume
- Make the Persistent Volume Claim
- Access the Persistent Volume Claim from your pod(s)



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mynfs-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /data/nfs
    server: my.nfs.server
    readOnly: false
```



```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mynfs-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      name: mynfs-pv
```



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mynginx-pod
  labels:
    name: mynginx-pod
spec:
  containers:
    - name: mynginx-pod
      image: nginx
      ports:
        - name: web
          containerPort: 80
      volumeMounts:
        - name: nfsvol
          mountPath: /usr/share/nginx/html
  securityContext:
      supplementalGroups: [100003]
      privileged: false
  volumes:
    - name: nfsvol
      persistentVolumeClaim:
        claimName: mynfs-pvc
```

**Dynamic provisioning may be supported in some implementations

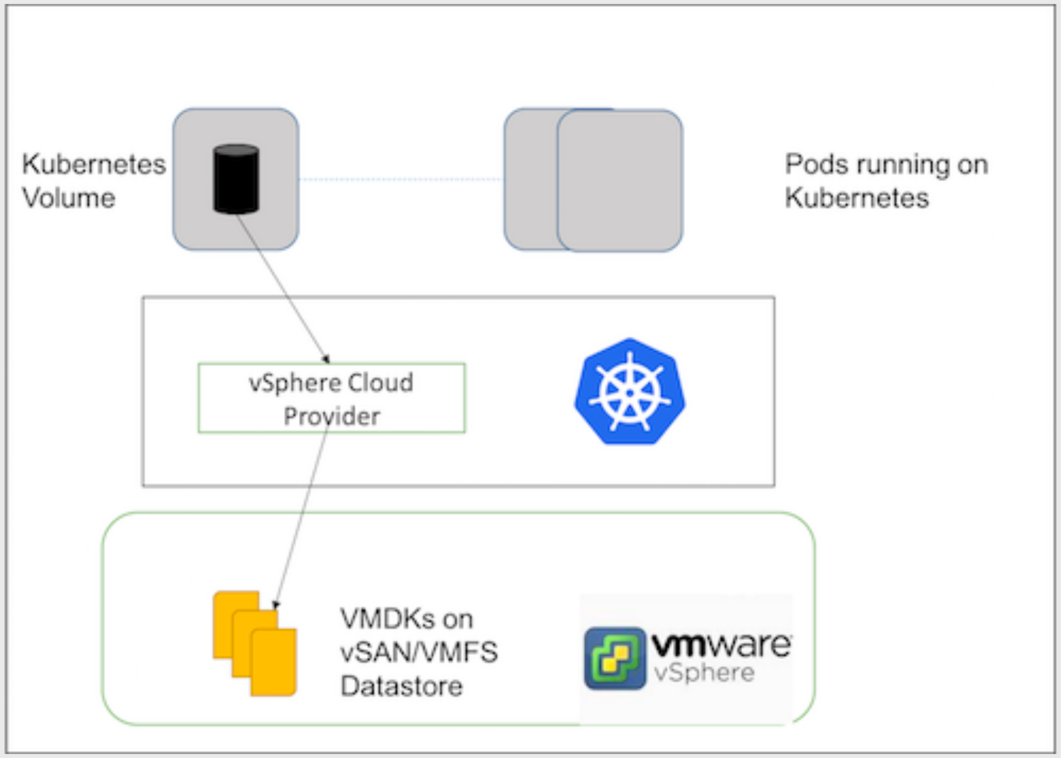
Storage via the vSphere Cloud Provider

Provides integration with the vSphere Software Defined Storage platform that integrates with block, file and hyperConverged storage offerings

Storage provided via vSAN, VMFS and VVol datastores depending on the backend storage used

Supports dynamic provisioning, encryption and de-duplication

Full support of RWO however does not support RWX / ROX (unless pods are collocated to a single node



GlusterFS and Heketi Architecture

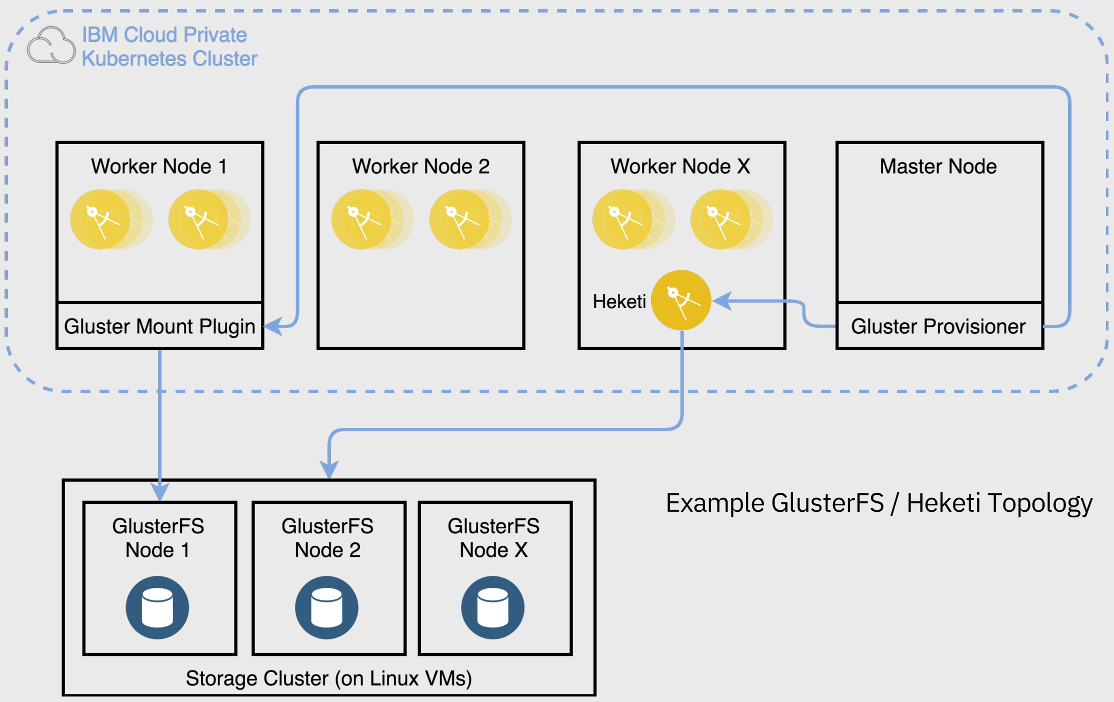
Heketi provides a Restful interface for the management of GlusterFS as well as the ability for. Full dynamic provisioning from ICP.

Easy to configure and deploy in various topologies and environments

Flexible scaling with no central metadata server

Supports encryption, replication, striping and dynamic provisioning

Full support of RWO, RWX and RWO access modes



Example GlusterFS / Heketi Topology

HostPath Storage Provider

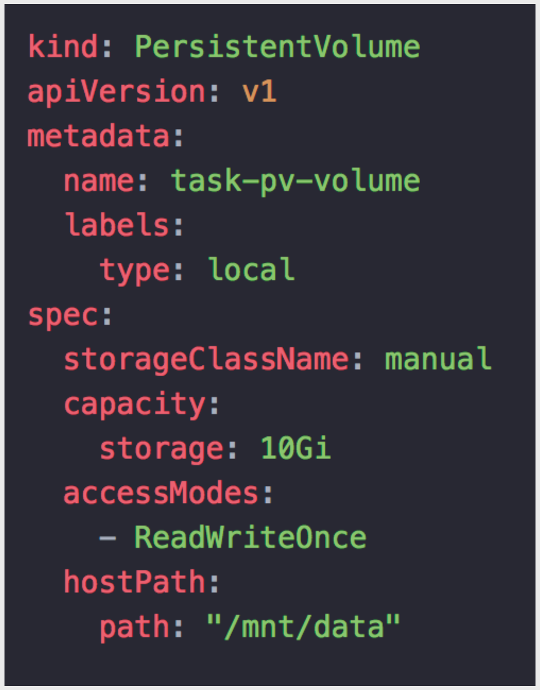
HostPath is a storage provider created for easer of development / testing and should not typically be considered for shared or production environments

Easy to configure and “get something working”

Data is not synced across multiple worker nodes thus can only persist if the pod is configured with affinity to a single node

Requires manual management

Not natively a secure method for managing storage



```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Making Storage Decisions

Finding the Best Solution for You

Which storage options are best?

Your solution likely lends itself to multiple storage options
Although you may find a single solution that fits the majority of your use cases, consider meeting outliers with a solution that isn't the MOST convenient, but perhaps allows you to make a better choice for the majority case

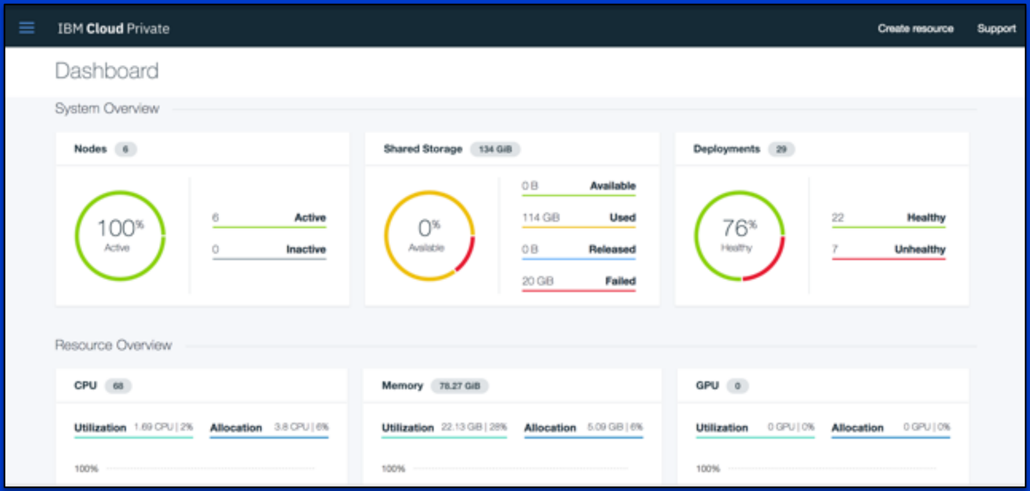
Consider your access policy requirements
Analyze the workload and its requirements (RWO, RWX, ROX) and determine how often each access mode is required. Compare this with the storage solutions ability to make storage provisioning dynamically and consider the ramifications to your deployment workflow.

Don't ignore the estate
Your operations team has storage solutions and preferences that may conflict with your instincts. They may or may not be willing to adopt yet-another-storage-solution. Don't ignore the effort required to manage the solution as it moves to steady-state. You may be required to bring along the entire operations model for the solution including support, monitoring, security, compliance, disaster recover, backup etc..

Try IBM Cloud Private today!

Guided and Proof of Technology demos

Free Community Edition!



http://ibm.biz/ICP-DTE