VOLUMES AND DATA

Volumes and Data

Volume Types

There are several types that you can use to define volumes, each with their pros and cons. Some are local, and many make use of network-based resources.

In GCE or AWS, you can use volumes of type **`GCEpersistentDisk`** or **`awsElasticBlockStore`**, which allows you to mount GCE and EBS disks in your Pods, assuming you have already set up accounts and privileges.

`emptyDir` and **`hostPath`** volumes are easy to use. As mentioned, **`emptyDir`** is an empty directory that gets erased when the Pod dies, but is recreated when the container restarts. The **`hostPath`** volume mounts a resource from the host node filesystem. The resource could be a directory, file socket, character, or block device. These resources must already exist on the host to be used. There are two types, **`DirectoryOrCreate`** and **`FileOrCreate`**, which create the resources on the host, and use them if they don't already exist.

NFS (Network File System) and iSCSI (Internet Small Computer System Interface) are straightforward choices for multiple readers scenarios.

rbd for block storage or CephFS and GlusterFS, if available in your Kubernetes cluster, can be a good choice for multiple writer needs.

Besides the volume types we just mentioned, there are many other possible, with more being added: **`azureDisk`**, **`azureFile`**, **`csi`**, **`downwardAPI`**, **`fc`** (fibre channel), **`flocker`**, **`gitRepo`**, **`local`**, **`projected`**, **`portworxVolume`**, **`quobyte`**, **`scaleIO`**, **`secret`**, **`storageos`**, **`vsphereVolume`**, **`persistentVolumeClaim`**, **`CSIPersistentVolumeSource`**, etc.

CSI allows for even more flexibility and decoupling plugins without the need to edit the core Kubernetes code. It was developed as a standard for exposing arbitrary plugins in the future.