Rethinking Shared Storage

Overview

The purpose of this document is to outline the current method for exposing shared data to all Pods within the Workbench, and to propose a more scalable and maintainable solution.

How It Works Right Now

Currently, Workbench is configurable for shared data which **assumes that the data is already mounted somewhere on the host**. (see Option #1)

Workbench then asks Kubernetes to mount this data into each Pod that is started from Workbench using a `hostPath` volume mount.

Users can then access this shared data via some pre-set path (e.g. `/shared`) to run analysis on large dataset within the shared folder.

The data can be either *read-only* or *read-write*, depending on how Workbench is configured.

How Did We Get Here?

Previous courseware required only very small datasets on a small-scale cluster. This may have been a single-node installation of Workbench.

It was therefore trivial to assume that the data would lie on the host, as there was only a single host to manage.

This obviously does not scale well to a multi-node cluster, as this requires us to manually mount an NFS on each host.

Note that we could have Terraform deploy the NFS server for us when the cluster is first deployed, but using `hostPath` still feels somewhat hacky (See Option #2)

What We Would Like to See Instead

In an ideal world, we should not make assumptions about where data might live on a particular host.

Instead, we could expose the shared data as a mountable volume within Kubernetes that Workbench can access.

This would provide us access to the data in a manner that can (theoretically) scale to as many nodes as we need.

Options

Since Workbench already uses PVCs for its ReadWriteMany mounting of the user's Home folder, using PVCs seems like an obvious choice.

PVCs don't tie us to a particular volume provisioner, for which there are many configurable options to allow us to easily swap out the underlying volume implementation.

Our typical pattern for using PVCs is to have Workbench create the PVC, and allow an external provisioner (NFS, GLFS, etc) to create the PV and bind it to the PVC.

This presents a fundamental problem, however: PVCs are namespaced resources, meaning that Pods outside of the PVC's own namespace will not be able to reference or mount it.

Since Workbench users each run in their own namespace, no users would have access to this data because they cannot access the PVC.

No Need for NFS: Copy the Data Manually to the Same Location on Each Node

No changes, Workbench will assume the data is already located on the host at some pre-specified path.

Cluster Admin will be responsible for ensuring that the data is properly copied to all nodes and accessible to containers (with correct permissions).

No need for NFS in this case, because the data is small (~500MB) and read-only

Pros:

- Scales better than NFS - no concurrency problems

Cons:

- Manual steps still needed to scale-up cluster (copying data, changing permissions, etc)

Suck it Up and Just Create Mount the NFS Manually On Each Host

No changes, Workbench will assume the data is already mounted on the host at some pre-specified path.

Cluster Admin will be responsible for ensuring that the data is properly mounted to all nodes (with correct permissions) and resilient to node restarts.

In practice, this will likely be an external NFS export or globus endpoint mounted manually to each node.

Pros:

- We know it works and how to set it up (manually)

Cons:

- Obvious inscalability

Have Terraform Deploy the NFS server for us

Automating the manual process would remove the problem.. wouldn't it?

Our Terraform deployment's goal is to provision OpenStack one or more VMs and install/run Kubernetes on them as a cluster.

This NFS deployment is not included in the Kubernetes objective, but where else would this live?

Should Terraform run Ansible to get this extra stuff set up? Is there value in this?

Pros:

- No more manual NFS deploy steps

Cons:

- Underlying Pods still rely on a pre-populated `hostPath` , which feels wrong
- This feels distinctly separate from the objective of deploying Kubernetes

Manually Create a PV and Bind new PVC each time a new Namespace is Created

One way to get around the namespace-exclusivity of the PVC would be to create the PV ourselves.

Unlike PVCs, PVs themselves do not have a namespace, and are therefore accessible by PVCs in all namespaces.

Workbench would need to be modified to accept (as a config option) the name/ID of a PV.

Anytime a new user is approved during the registration process, a new namespace is created and Workbench creates the user's Home folder PVC there.

In addition to this Home folder PVC, Workbench would need to create another PVC the binds to the configured PV.

Pros:

- Mountable across namespaces
- PVC-friendly interface, albeit in reverse

Cons:

- Workbench now needs to be PV-aware (For some reason, it feels wrong to embed more PV/PVC logic here)

Why Use PVCs at All? Run a small NFS externally and mount the export directly to Pods

This would probably work quite well for read-only data and/or a small number of concurrent users.

We run an NFS export outside of Kubernetes that is exposed to the cluster.

When Workbench starts up Pods, the NFS export is mounted directly to each Pod in the same location.

The downside is that the configuration for the volumes in Workbench is now starting to get quite complex.

Pros:

- Mountable across namespaces

Cons:

- Custom interface (NFS mounts) that will likely be unused elsewhere - seems annoying to need to support NFS mounting externally for one case
- Testing locally will now require NFS

Run Workbench in a Single-Namespace

Run Workbench in a single namespace, and no longer create a namespace per-user.

For small-scale installations, we could offer a configuration option to run all user services in a single namespace.

This is similar to how JupyterHub and other products already work, so it may feel more familiar.

That way, it would be trivial remount the same PVC within that namespace to multiple Pods.

Pros:

- Simplifies Workbench installation, makes it easier for users to understand (e.g. similar to JupyterHub, other related products)
- Mountable across namespaces
- PVC-friendly interface
- Closes a very old issue on GitHub: https://github.com/nds-org/ndslabs/issues/250

Cons:

- Loss of network / resource isolation - particularly concerning in CHEESE, where security and hacking principles are taught