PolyFS - The Easiest to Use Distributed Filesystem

Katharos Technology

ABSTRACT: In the software community today, there is a shocking lack of easy-to-use, Open Source, distributed filesystems. Shared storage is crucial to any business beond the scale of single server, and a distributed filesystem is the most universal way to enable shared storage. PolyFS seeks to close the gap in shared filesystems and provide a solution that is easy to use, manage, and deploy while at the same time providing top-of-the-line performance. PolyFS, while not being limited to any particular application or usage, will be designed to include a Docker deployment and volume plugin and will provide an end-to-end solution for distributed volumes hosted by and for Docker Swarm.

NOTE: This is a work-in-progress design document outlining the plan for the design of PolyFS. This document will be updated as we discover more and change the design. The architecture in outlined in this paper has not been tested as of yet.

Introduction

As an organization with experience running stateful containers in the Cloud with Docker Swarm, we are well acquainted with the need for a distributed Docker storage solution. When running applications on a Docker cluster, you have to have a way to make sure that each application can get to its persisted data. Unlike running an app on a single server, you cannot use the local disc for storage because you never know which server a particular container may start up on. You could constrain your app to run only on a specific server, but that eliminates much of the advantages of having the Docker cluster in the first place. You want to be able to view your cluster as a pool of resources that you can run your applications in, without having to worry about where those resources are comming from.

The need for shared storage is not limited to Docker. Some applications *require* that you provide some form of a shared filesystem in order to cluster the application. The only way that you can run these applications in a fault tolerant way for high availability is to provide a highly available distributed filesystem.

Existing Solutions

There are a number of existing distributed filesystems, but so far we have not found one that statisfies our requirements:

Project	Problems
Ceph	Large and complicated to deploy and manage
Gluster	They say not to run databases on it and we have heard that it can be slow
LizardFS	Unresponsive dev team, and support is low while they rewrite it
MooseFS	Proprietary high availability system that is not Open Source
Portworx	Not Open Source
SeaweedFS	Afraid of the lack of data integrity assureance

Ceph

Ceph is a big name in distributed filesystems and was our first choice for attempting a distributed filesystem, but after trying to install and run it on Docker Swarm, we realized that it was very complicated setup. Running and managing Ceph on Swarm seemed to be difficult to impossible. Additionally other people also have stories to tell of how horrible Ceph was to try and stand up before they gave up and sought out a different solution.

Gluster

We have heard some report of Gluster's performance being low which made us unsure about trying it. Eventually we decided we would try it ourselves until we read the following in their documentation:

> Gluster does not support so called “structured data”, meaning live, SQL databases. Of course, using Gluster to backup and restore the database would be fine.

LizardFS and MooseFS

LizardFS is a fork of MooseFS, an easy to deploy distributed filesystem, but they have been developed separately for the last 6 years. LizardFS is developed more openly than MooseFS with the biggest difference being that MooseFS's HA deployment software is proprietary. LizardFS appeared for a long time to be the filesystem that we were looking for. It was super easy to deploy and manage, and we successfully ran it on a Swarm cluster and wrote a Docker volume driver for it. Everything seemed about right until we discovered that development was stopping on the current version in favor of a rewrite. This ocurred after about 7 months of silence on the part of the dev team. This served to remove our confidence in LizardFS, and, due to the closed source HA solution for MooseFS, left us without a filesystem.

Portworx

Portworx has an impressive offering that achieves most of what we are looking for, but the installation requires more setup on the Docker host machine that we would like and thier software is not Open Source.

SeaweedFS

SeaweedFS is an extemely interesting project that has gotten a lot of attention on GitHub. SeaweedFS seems to be the most promising Open Source project for distributed filesystems out right now. The issues that we have with SeaweedFS is the lack of assurance when it comes to data integrity, and its unstability as a quickly developing tool. There are still concerns that have yet to be addressed and SeaweedFS is just too immature and untested for us to trust our production data with it yet.

Summary Of Existing Solutions

None of these options have satisfied our requirement for an Open Source distributed filesystem that can run in Docker containers on Docker Swarm and provide shared Docker volumes for the Swarm. It is worth noting that there are promising projects for providing storage for Docker containers on Kubernetes, such as OpenEBS and Rook, but we are looking for a solution that can run in Docker Swarm, or anywhere else.

After months of searching for a satisfactory existing solution in vain, we have decided to atempt to build our own filesystem tailored to our specific goals.

PolyFS Goals

Ease of Use and Simplicity

One of the highest concerns for PolyFS is ease of use and, by extension, simplicity. Building a distributed filesystem is a difficult job, but running one should not have to be. We want PolyFS to be absolutely as easy to think about, deploy, and manage as possible. Our design seeks to eliminate as many unnecessary components and services as possible. The filesystem should do everything that it can to manage itself so that there is very little operational demand. You should not have to struggle to understand how to run PolyFS wherever you need it.

Performance and Reliability

Obviously no amount of ease of use is actually useful if the performance and reliability of the system is not adequate. PolyFS will need to perform as well or better than the other existing filesystems and you should have no fear that you data will be any less safe with it than with your local hard drive. Our design decisions should put no bottelneck on how fast the filesystem can perform or on how much storage you can add to the cluster.

Docker Swarm Integration

With PolyFS we will provide the documentation and configuration necessary to stand up PolyFS on Docker Swarm, along with a Docker volume plugin that can be used to mount Docker volumes on top of the fileystem. Kubernetes support may be added later. PolyFS will not be designed specific to any orchestrator, though, and should be able to be integrated into any system. Docker will not be required to use PolyFS.

PolyFS Design

PolyFS's overall design can be seen in **Figure 1**.

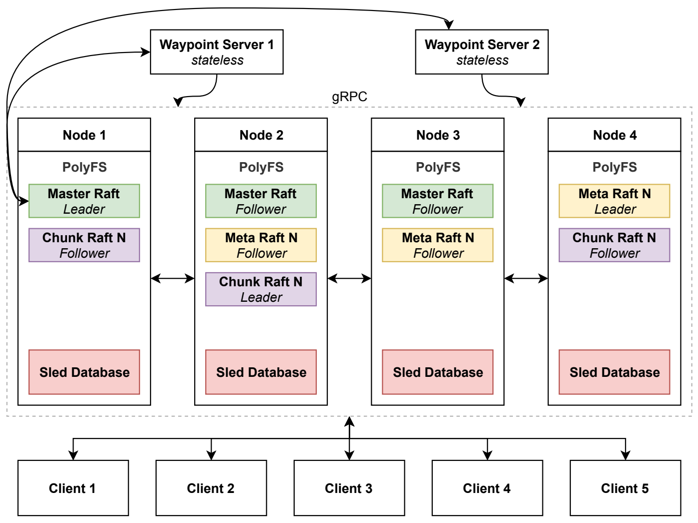

Figure 1: PolyFS Architecture

PolyFS is distributed as either a Docker image or a single binary that is deployed to all the servers in the cluster.