GlusterFS as an Object Store

Aravinda Vishwanathapura
aravinda@kadalu.io

Draft v0.1 Apr 23, 2021
Draft v0.2 Apr 26, 2021
Draft v0.3 Apr 30, 2021

GlusterFS [1] is a Open source scalable network filesystem.

GlusterFS already provides rich APIs to interact with the file system other than Fuse. Libgfapi is used by many integrations like NFS Ganesha [2], Samba [3], Qemu integration [4] etc. This article provides a possible way to use GlusterFS as object storage.

Many existing solutions like Minio[5] [6] [7], OpenStack Swift[8] can provide Object storage on top of GlusterFS mount. Why this project instead of reusing the existing ones?

- Libgfapi is already providing APIs. The gap is tiny to complete the Story.
- As mentioned above, Libgfapi is a stable interface and used in production with many integrations.
- Avoiding the Fuse layer by using LibGfapi is a huge win! Better Performance.
- Get all the benefits of GlusterFS instead of using a new Object Storage.

1 Hello World!

`glfsio`[9] is a tool created by Kadalu team to provide Object interface to GlusterFS volumes by using `libgfapi`. Update the configuration file as required and run `glfsio` using,

```
# config.yaml
---
volume: "gvol1"
rootdir: "/records"
servers: ["node1.example.com",
↪  "node2.example.com",
↪  "node3.example.com"]
immutable: false
enable_deletes: true
enable_versions: false
enable_dedup: true
port: 4000
```

```
# ./glfsio --config=/etc/glfsio/config.yaml
glfsio server is ready at
↪  http://0.0.0.0:4000
```

Note: `libgfapi.so` should be available in the node where `glfsio` is running.

Now create a Python/Ruby/NodeJS script to interact with the service.

```
require "glfsio"

volume = GlusterFSVolume.new(
    GLFSIO_SERVER_URL, "gvol1"
)
volume.create_file("/hello.txt", "Hello
↪  World!")
puts volume.read_file("/hello.txt")  #
↪  Hello World!
```

[1] https://gluster.org
[2] https://github.com/nfs-ganesha/nfs-ganesha/tree/next/src/FSAL/FSAL_GLUSTER
[3] https://www.samba.org/samba/docs/current/man-html/vfs_glusterfs.8.html
[4] https://github.com/qemu/qemu/blob/266469947161aa10b1d36843580d369d5aa38589/block/gluster.c
[5] https://kadalu.io/docs/k8s-storage/latest/running-minio/
[6] https://vinayak-hariharmath.medium.com/how-to-export-gluster-volume-as-an-s3-compliant-object-store-fe4a7d31bf7a
[7] https://gist.github.com/harshavardhana/bda9b18d40f9b733e32eaa3b0736ca70
[8] https://github.com/gluster/gluster-swift
[9] Coming Soon..

2 Features

- **De-duplication** Content Hash is used as the file name in the backend. With this, we get de-duplication for free!
- **Immutability** Complete Volume can be made immutable or a selected bucket/directory/file can be made immutable depending on the requirement.
- **Versions** Since the content hash changes if the file content is edited, each edit can be stored as a new version. Retention period can be configured once Versions are enabled. Versions can be enabled or disabled at bucket/directory level.
- **Sub directory support** If the portion of Volume to be used as Object Store then configure that sub-directory as the root directory.
- **Replication and Data high availability** Underlaying GlusterFS Volume takes care of the high availability and data redundancy.
- **All GlusterFS features** All features of the GlusterFS like Geo-replication, Snapshots, Tiering, Quota etc are available with this project.

3 APIs

3.1 Create File

Creating a file using HTTP POST will calculate the content hash and stores the actual data with hash value as the file name. The Object metadata are stored in Db for quick access later.

Read the file content and then calculate the Hash value. Now copy the uploaded file inside the `.store` directory with the name as the hash value.

```
volume.create_file("/hello.txt", "Hello
↪  World!")
volume.create_directory("/images")
imagedata = File.read("/uploads/logo.png")
volume.create_file("/images/logo.png",
↪  imagedata)
volume.list_files("/images")  //
↪  ["logo.png"]
```

- Read the uploaded file content and calculate the file hash(`sha256`).
- Create a file in the GlusterFS volume with the name same as the Hash of the file.
- Create a hard link file with the name same as the uploaded file to the file created in the previous step.
- Add entry to `.meta.db` in the respective directory.

The following pseudo-code to understand the steps when `POST /gvol1/v1/file1` api is called.

```
content = File.read("/uploads/file1")  #
↪  Hello World!
hash = hash_sha256(content)  #
↪  03ba204e50d126e4674c005e04d82e84c213667
↪  80af1f43bd54a37816b6ab340
hash_file_path = ".store/#{hash[0,2]}/#{has
↪  h[2,2]}/#{hash}"  #
↪  .store/03/ba/03ba204e50d126e...

if !file_exists(hash_file_path)
    rename_file("/uploads/file1",
    ↪  hash_file_path)
end

create_hardlink(hash_file_path, "file1")

sqlite_add_file(".meta.db", "file1", ...)
```

3.2 Edit a File

This option is enabled only if `immutable=false`. Edit API will recalculate the content hash and does the necessary relinking. Keep the old content if `enable_versions=true`.

Calculate the new content hash and then check for that hash existence. If not exists, create a new hash file in the `.store` directory. Delete the old hash file if no other hard links exist (that is if link count is 1) and if version support is not required. Move to the .versions directory if all the versions of the file to be maintained. Refer Version support section.

```
volume.update_file("/hello.txt", "Hello
↪  World again!")
```

The following pseudo-code to understand the steps when `PUT /gvol1/v1/file1` api is called.

```
new_content = File.read("/uploads/file1")
↪  # Hello World!
new_hash = hash_sha256(content)  #
↪  03ba204e50d126e4674c005e04d82e84c213667
↪  80af1f43bd54a37816b6ab340
new_hash_file_path = ".store/#{hash[0,2]}/#
↪  {hash[2,2]}/#{hash}"  #
↪  .store/03/ba/03ba204e50d126e...

current_hash = get_current_hash("file1.txt")

if !file_exists(new_hash_file_path)
    rename_file("/uploads/file1.txt",
    ↪  new_hash_file_path)
end
```

```
create_hardlink(new_hash_file_path, "file1")

if number_of_hardlinks(current_hash) == 1
        delete_hardlink(current_hash)
end

sqlite_update_file(".meta.db", "file1", ...)
```

3.3 Read the content

A file can be read in two ways. One using the file name used while creating the file, and the other way is by using the File hash. Internally both are the same since these are hard links in the GlusterFS Volume.

```
volume.read_file("/hello.txt")  # Hello
↪  World!
volume.read_file("03ba204e50d126e4674c005e0
↪  4d82e84c21366780af1f43bd54a37816b6ab340
↪  ")  # Hello
↪  World!
```

3.4 Delete file or Directory

```
volume.delete_file("/hello.txt")
```

This option is enabled only if `immutable=false` and `enable_deletes=true`. Following steps will be run when a file is deleted.

- Get the Hash of the file from the xattr or db.
- Delete the file
- Delete the hash file only if the link count is 1
- Remove entry from respective directory's `.meta.db` file.
- Remove all versions of the file if exists.

3.5 Renaming the files and directories

```
volume.rename_file("/hello.txt",
↪  "/hello_world.txt")
```

3.6 Listing the directories

Directory listing is optimised by keeping the files list in a database(SQLite3) in the respective directory. Listing files is equivalent to the following command.

```
-- List of files
SELECT * FROM files;

-- Directory utilization
SELECT sum(size) AS utilization FROM files;
```

3.7 Storage Accounting

Each directory maintains its accounting in the `.meta.db` file. Aggregated information will be tracked in Project root directory.

4 Version Control

If version support is enabled, then maintain the hard links under the `.versions` directory. These versioned files are maintained by appending version information to the existing file paths.

5 Configurations

- **De-dupe support** This can be disabled if de-duplication is not required.
- **Version support** Version support is disabled by default, but This can be enabled when required.
- **Access optimisations** If the frequency of accessing files using hash more than the Path then the approach of creating the file can be optimised. For example, if the hash file is created first then if the file by path is hard-linked then if the file path hashes to a different distribution group then GlusterFS creates a link-to file in the hashed brick and the actual content is in another brick. This may increase the latency to access the file. If the files are accessed using Path more often than the hash then Create a file with the actual name first and then create a hard link with the name as a hash.
- **Root directory** If only one portion or directory of a GlusterFS volume needs to be converted as an Object store, then root dir can be configured as required.
- **Immutable** With these configurations, Edit and delete will not be available. Once created it will stay in the GlusterFS Volume.
- **ACL** TBD

6 What next?

- Even though this solution is usable in its current state, the S3 compatibility layer will benefit more users.
- Implement Volume and Directory level Access control.
- Caching layer for better read performance.