EOS Integration into OpenStack Manila

AUGUST 2019

AUTHOR(S):
Elisabeth Petit - Bois
Kennesaw State University

SUPERVISOR(S):
Andreas-Joachim Peters
CERN

ACKNOWLEDGEMENTS

First and foremost, thank you to CERN's Information Technology Storage (IT-ST) team for creating a very welcoming and nurturing environment for this openlab Summer Student Program. Individuals like Arne Wiebalck and Mihai Patrascoiu helped make all of my progress possible by supporting my efforts and being available for any questions I may have had. For that, I am very appreciative.

Finally, a very, very special thank you to my supervisor, Andreas-Joachim Peters, for his constant encouragement and flexibility during the nine weeks of my stay at CERN. I could not have asked for a better team or a better mentor.

PROJECT SPECIFICATION

OpenStack is a popular open-source cloud computing platform used widely at CERN. It services many uses through several particularly-named components. For example, OpenStack features “Keystone” for authentication, “Nova” for virtual machine management, and “Manila” for shared file system capabilities. For the purposes for this project, we focus on OpenStack Manila, an OpenStack component meant to deliver shared file systems across an organization.

EOS, a disk-based, low-latency storage service, was originally created to host experimental data, but it now has expanded to many more use cases across CERN. Today, EOS powers user, project and experiment data on services such as CERNBox.

In its current state, OpenStack and EOS are very distinctive, hardly ever making any interaction between the two services. For CERN’s nearly 12,000 users, this configuration can prove to be inconvenient and inefficient. While users are able to quickly access files and folders on CERNBox, users are unable to effectively manage this storage space. Management may involve controlling the size of storage available, access rights, and much, much more.

To introduce a better workflow and options for users, we strive to connect OpenStack and EOS Services by implementing a OpenStack Manila driver, effectively linking OpenStack to CERN’s EOS file system. By doing interfacing the two platforms, users will be able to and the introduction of EOS GRPC invites future opportunities for EOS to expand throughout more services within the Organization.

ABSTRACT

The purpose of this report is to provide a brief overview of what OpenStack is, focusing on the advantages of the integration of its Manila component at CERN. Furthermore, this document briefly describes the EOS file service, its history, and its potential to introduce a much more cohesive cloud environment for users by porting its services to OpenStack Manila.

CERN currently uses OpenStack heavily, employing more than five of its components to perform authentication, manage virtual machines, and more. OpenStack Manila, a shared file service, is one of the newest additions to the organization's deployments. OpenStack Manila allows for storage management and simultaneous file access.

EOS, a file service, hosts more than 270 petabytes worth of experiment, project, and user data. Through services like CERNBox, users are able to access their files easily through the EOS file system. While users are able to quickly retrieve files, users lack the ability to flexibly manage their storage space.

The scope of this project is to bridge the gap between services like CERNBox and OpenStack Manila through the EOS file system in order to build a much more efficient, user-friendly system. This report will analyze OpenStack Manila's architecture and operations. It will also provide detailed descriptions on how to navigate configuration of an OpenStack Manila installation using DevStack, a software meant to emulate a production OpenStack environment.

TABLE OF CONTENTS

1. INTRODUCTION

With more than 12,000 users worldwide, CERN is inherently required to provide quality service to its users. Server uptime and software responsiveness are all examples of software experience components that have the potential to frustrate and deter users from engaging with the organization's facilities. Since the creation of CERN openlab, officials have done their best to pinpoint ICT research challenges for the upcoming five years. Being able to effectively support CERN's growing number of users has not missed this list, especially when it comes to cloud computing.

Cloud computing is a key component for CERN's ability to quickly and effectively communicate and collaborate with individuals within and outside of the organization. OpenStack is the primary private cloud computing platform at CERN. OpenStack ships with multiple components including, but not limited to, networking, virtual machine deployment, and block storage. In total, CERN has deployed eight OpenStack components.

To address the potential issue of user support, OpenStack must be able to deliver sublime service to its users, especially when it comes to file and storage management through its component, Manila.

2. WHAT IS EOS?

EOS is the disk storage system at CERN for large physics data. Gradually, EOS has expanded its use cases beyond physics to include user and project data as well. As of the time of this writing, EOS hosts more than 270 petabytes worth of data across all of its domains.

3. WHAT IS OPENSTACK?

OpenStack is an open source cloud computing tool kit. It allows organizations to install and implement cloud services according to business need. OpenStack has 32 components within its overall package geared towards sectors like networking, orchestration, and storage.

Table 1 lists the major OpenStack services, service components, and their functions:

Table 1
OpenStack Services and Components

Service	Components	Description
Compute	Nova, Zun, Qinling	Provides services to access compute resources, manage resources, and support serverless functions.
Hardware Lifecycle	Ironic, Cyborg	Provides access to compute resources and management for hardware accelerators.
Storage	Swift, Cinder, Manila	Allows for object storage, block storage, and shared storage.
Networking	Neutron, Octavia. Designate	Delivers networking-as-a-service, DNS-as-a-service, and load balancing.
Shared Services	Keystone, Placement, Glance, Barbican, Karbor, Searchlight	Provides a variety of services including authentication, cloud tracking, virtual image retrieval, searching, etc.

Orchestration	Heat, Senlin, Mistral, Zaqar, Blazar, AODH	Allows for orchestration of cloud applications, homogeneous objects, workflows, messaging, and security.
Workload Provisioning	Magnum, Sahara, Trove	Handles OpenStack prioritization of background services, accessing data processing frameworks, and implementing database-as-a-service.
Application Lifestyle	Masakari, Murano, Solum, Freezer	Provides services for disaster recovery, backup, application installation, and imaging.
API Proxies	EC2API	Provides an EC2-compatible API to OpenStack Nova.
Web Frontend	Horizon	Provides a GUI for users to access OpenStack services.

a. OPENSTACK ARCHITECTURE

OpenStack is quite unique in that it has multiple “plug-and-play” components connected to its core. While there are many options available as seen in Table 1, not all are necessary to configure to create a fully functional environment.

Figure 1, for example, illustrates an OpenStack environment in which there are five components configured for use: Keystone, Nova, Manila, Neutron, and Cinder. While there may exist some interdependence between modules, all five components largely operate separate from one another, allowing developers to pick and choose which modules are most essential to pending business needs.

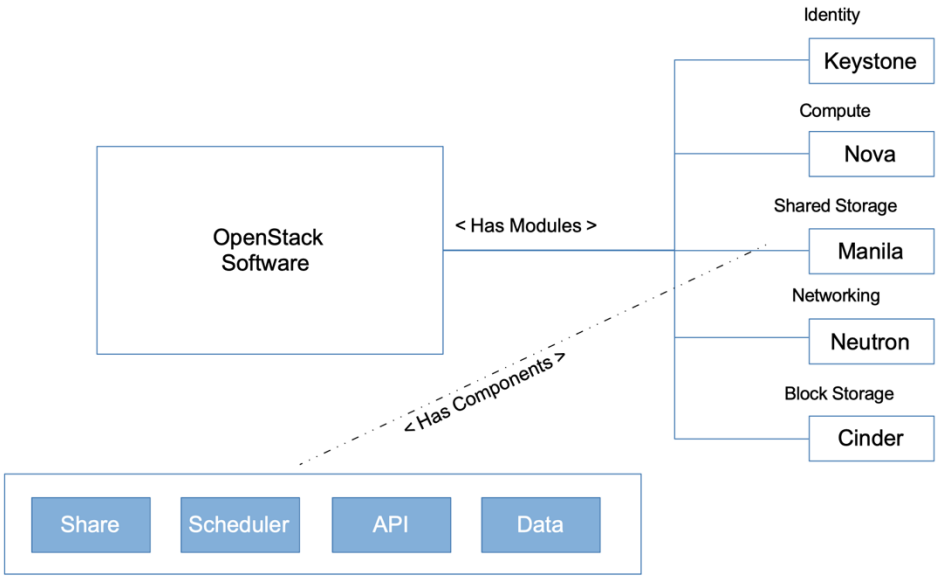

Figure 1 -- OpenStack Architecture

To keep each module independent, the module hosts a subsystem. In Figure 1, this concept is illustrated by the Manila component which features a subsystem containing “Share,” “Scheduler,” “API,” and “Data.” Together, these pieces allow OpenStack Manila to operate independently.

b. OPENSTACK AT CERN

OpenStack is CERN’s primary cloud service software. With it, CERN is able to deploy virtual machines through Nova, implement authentication via Keystone, and equip users with block storage with Cinder. Through these use cases and the user of many other OpenStack component deployments, CERN is able to effectively collaborate with researchers through the Worldwide LHC Computing Grid.

4. OPENSTACK MANILA

Today, OpenStack manages more than five services at CERN through its components. Components like Ironic, Oslo, Glance, Keystone, and Nova have found their way to deployment and, consequently, everyday use across the organization. Still, even now, the quest to further improve operations continues.

Since 2015, CERN has explored OpenStack Manila, a file sharing component in the cloud computing bundle. OpenStack Manila allows users to manage and access system storage simultaneously. Shared file system storage allows users to have more flexibility and control over their data and where it is stored.

This section provides a basic overview of OpenStack Manila terminology as well as its architecture.

a. OPENSTACK MANILA BASICS

Before delving into the details of OpenStack Manila, it is imperative to understand key terms essential to the system’s operation.

i. SHARES

A share is the primary resource unit in OpenStack Manila. A share represents the location of storage space on a readable and writable filesystem. Shares are created and managed by the end-user.

Because file systems may require different protocols to communicate, shares are protocol-specific. They are also configurable to have a custom name, description, and size.

ii. SHARE TYPES

A share type is a label placed on a share in order to identify its backend. Share types are important because they ultimately influence how a share responds to user share management requests. During creation, Manila shares must be assigned a share type.

iii. BACKENDS

A backend serves as the gateway between OpenStack Manila and a file storage system. Backends work with Manila drivers in order to fulfil share creation and share management requests. Multiple backends may be enabled at once on an OpenStack

instance; however, only one backend can be assigned to a share at a time. A Manila share must have an assigned backend.

iv. MANILA DRIVERS

A driver allows for communication between OpenStack Manila and external file services. By default, OpenStack Manila ships with twenty Manila drivers. These drivers include support for CephFS, GlusterFS, IBM GPFS, and many other file systems.

Manila is not restricted to the drivers listed; in fact, developers are encouraged to contribute to OpenStack Manila's driver repository.

b. OPENSTACK MANILA ARCHITECTURE

Looking closer at OpenStack Manila's backend, the API and Scheduler modules serve as the brain for the component. The API module processes REST requests through a messaging bus. The messaging bus works in sync with the API in order to delivers the requests to the appropriate Manila processes. Figure 2 provides a closer look at the relationship between these modules.

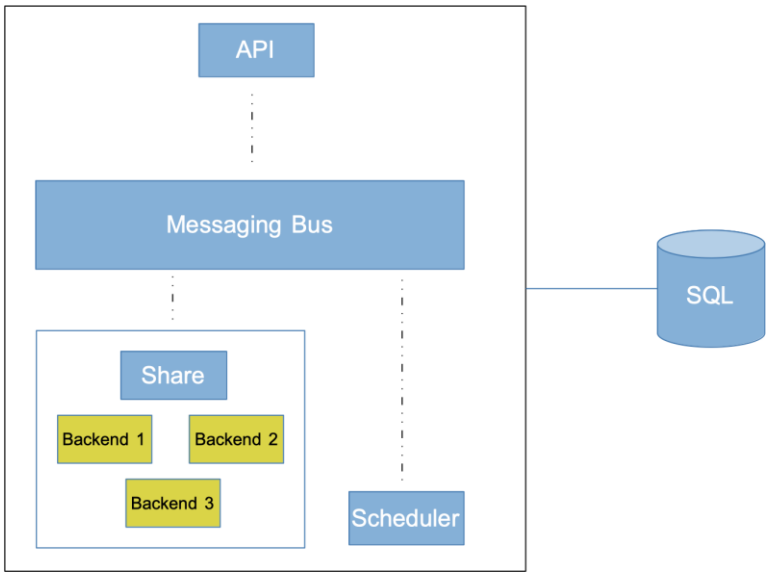

Figure 2 -- OpenStack Manila Architecture

When creating Manila shares, a user must make a request via the Manila API. From this point, the API routes the request to the scheduler to filter available backends according to the requested share type and share protocol. Once the scheduler has determined a suitable backend to process share creation, Manila stores the new share information in an internal SQL database.

To manage the existing share, the user must send a request through the Manila API. Ultimately, the share's assigned backend determines how to respond and manipulate the share according to the user's request.

5. IMPLEMENTATION

This section details the outlines the technologies required to install, configure, and maintain OpenStack Manila as well as deploy the interfaces necessary to connect OpenStack and EOS.

It should be noted that, for the purposes of this project, two solutions were produced in order to provide a proof-of-concept before continuing with the production system. Therefore, at the end of this section, the report will describe the two solutions curated in order to successfully integrate EOS into OpenStack Manila.

a. TOOLS

Two requirements guided the development tool selection process. Firstly, it was essential to find an OpenStack environment separate from the one deployed at CERN in order to test new configurations. Secondly, it was important that the interface connecting EOS to OpenStack seamlessly integrate into the existing development stack.

i. DEVELOPMENT ENVIRONMENT

Devstack is a software package consisting of scripts that create a sample OpenStack environment. With it, developers are able to create and modify configurations without disrupting production. Devstack also gives developers the opportunity to explore the OpenStack platform and its various components without committing to the software.

For the purposes of this project, Devstack was used to create an environment complete with the Manila component, the Horizon component, and their respective dependencies. As a reminder, Manila enables file system sharing while Horizon provides a graphical user interface for OpenStack.

ii. MESSAGING

GRPC is a remote procedure call framework that allows for communication within any environment. For systems that have different backend implementations, GRPC is useful because it allows information to flow freely between backends through data structures called “protobufs.”

A protobuf universalizes communication by defining the messaging structure between two backends. The code snippet below provides a protobuf example:

```
enum MANILA_REQUEST_TYPE {
       ...
}

message ManilaRequest {
       MANILA_REQUEST_TYPE request_type = 1 ;
       string auth_key = 2 ;
       string protocol = 3 ;
       string share_name = 4 ;
       ...
       string share_location = 13 ;
}

message ManilaResponse {
       string msg = 1 ;
```

```
        string code = 2 ;
        ...
        string new_share_path = 6 ;
}

service Eos {
        rpc ManilaServerRequest(ManilaRequest) returns
  (ManilaResponse) {}
}
```

The snippet defines two message types: a ManilaRequest and a ManilaResponse. Each message type is defined by several attributes. For example, in the case of ManilaRequest, it is composed of four distinct strings *auth_key*, *protocol*, *share_name*, and *share_location*. It also may contain a MANILA_REQUEST_TYPE object.

Below the message definitions in the code snippet, the *Eos* (EOS GRPC) service is defined. The EOS GRPC service sends and receives messages between the GRPC server and the client. Message must be in the same format as those defined in the protobuf file.

The universal nature of protobufs is particularly useful for communication between OpenStack and EOS. The production EOS GRPC configuration supports a C++ backend. OpenStack ships with a Python backend; therefore, it is most intuitive to use GPRC for cross-backend messaging.

b. MOCK SYSTEM ARCHITECTURE

Before building a production system, a mock EOS GRPC server was developed to emulate the creation and deletion of shares. The mock EOS GRPC sever also emulates the increase and decrease of share size as well as the managing and unmanaging of shares. The shares exist on the local machine.

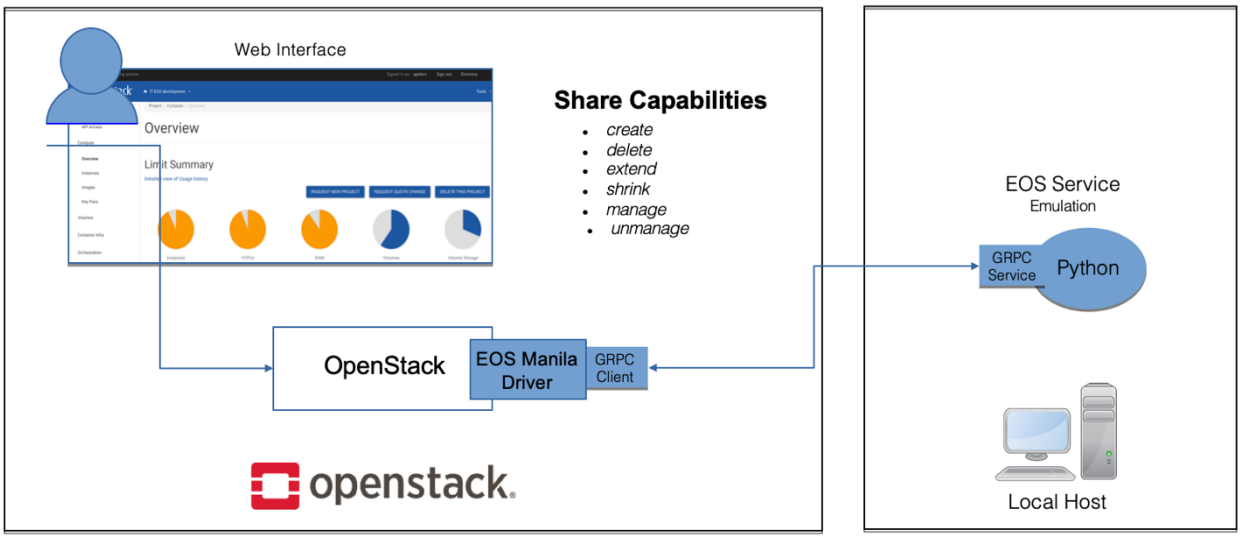

Figure 3 -- Mock System Architecture

As shown in Figure 3, when a user makes a request through the Mania API, OpenStack routes the request as explained in Figure 2. The backend communicates with the EOS Manila driver in order to deliver the request to the mock GRPC server. The mock GRPC server handles the request on the local machine and returns the response back through the initial pipeline.

The EOS GRPC server requires that the EOS Manila driver pass a valid authentication key with its request. The authentication key is adjustable through the OpenStack Manila configuration file. The GRPC server also requires that the share being manipulated uses the EOS protocol. This setting must be applied upon share creation. In the case that either of these requirements is not satisfied, the user's request will be denied.

c. PRODUCTION SYSTEM ARCHITECTURE

In place of the local machine originally in the mock systems, the production system instead interfaces directly with the EOS file system at CERN. In this scenario, shares are located in the logged in user's directory. Figure 4 illustrates the new workflow.

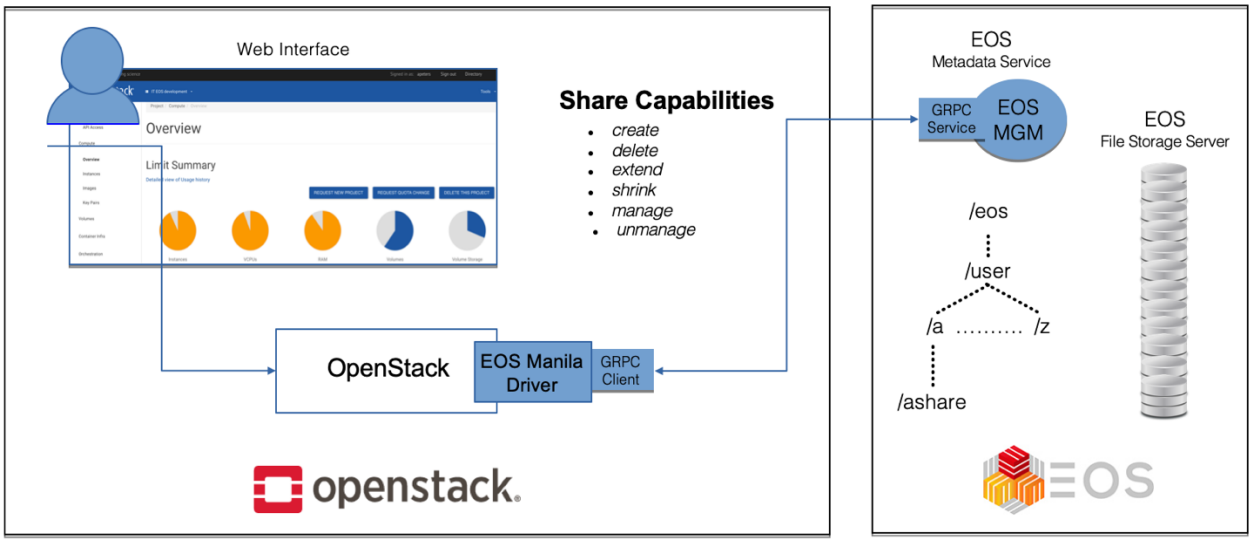

Figure 4 -- Production System Architecture

6. SYSTEM CONFIGURATION

The purpose of the EOS Manila driver is to directly connect CERN users with the OpenStack interface in order to request and configure in a self-service approach storage space at CERN. Every CERN user has a dashboard in OpenStack where one can request virtual machines and space in storage systems. The driver aims to connect the two systems using their APIs.

The EOS Manila driver is hosted on GitHub, and it is freely accessible for users to download and manipulate. The driver is generic baseline to understand how OpenStack Manila handles calls made to drivers in order to communicate with outside interfaces.

a. OPENSTACK MANILA INSTALLATION WITH DEVSTACK

To begin using the EOS Manila driver with OpenStack, it is first necessary to install OpenStack with Manila support using Devstack. After building a compatible Linux machine dedicated to OpenStack, using a root Linux account, run the following commands:

1. Create a "stack" user with sudo privileges.

```
$ sudo useradd -s /bin/bash -d /opt/stack -m stack
$ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
```

2. Switch to the "stack" user.

```
$ sudo su - stack
```

3. Clone the DevStack repository and change directories into the newly downloaded folder.

```
$ git clone https://opendev.org/openstack/devstack
$ cd devstack
```

4. Copy "local.conf" file from /devstack/samples into the devstack folder.

```
$ cp samples/local.conf ./
```

5. Add the following lines at the bottom of the local.conf file just copied into the root directory of the devstack folder.

```
enable_plugin manila https://github.com/openstack/manila
enable_plugin manila-ui https://github.com/openstack/manila-ui
```

6. Start the install.

```
$ ./stack.sh
```

The installation will take between 30 to 40 minutes, depending on the speed of the available internet connection. After it has finished, Devstack will supply a sample admin and demo accounts to use freely.

b. ENABLING EOS AS A SHARE PROTOCOL

As mentioned previously, the EOS driver will only communicate with Manila drivers that employ the EOS protocol. In any other case, the driver will deny permisson to the request. Therefore, it is imperative to enable EOS as a share protocol in OpenStack Manila.

1. Modify the Manila configuration file to add EOS as a share protocol.

```
$ vi /etc/manila/manila.conf

...
#modify the other enabled share protocols as neccesary
enabled_share_protocols = NFS,CIFS,EOS
```

2. Modify Manila UI to enable EOS as a share protocol.

```
$ vi ~/manila-ui/manila_ui/local/local_settings.d/_90_manila_shares.py

OPENSTACK_MANILA_FEATURES = {
    'enable_share_groups': True,
    'enable_replication': True,
    'enable_migration': True,
    'enable_public_share_type_creation': True,
    'enable_public_share_group_type_creation': True,
    'enable_public_shares': True,
    'enabled_share_protocols': ['NFS', 'CIFS', 'GlusterFS', 'HDFS', 'CephFS',
                                'MapRFS', 'EOS'],
}
```

3. Register the configuration options for the EOS Manila Driver.

```
$ vi ~/manila/manila/opts.py

import manila.share.drivers.eos-manila.driver
...
_global_opt_lists = [
    ...
    manila.share.drivers.eos-manila.driver.eos_opts
    ...
]
```

4. Define EOSException.

```
$ vi ~/manila/manila/exception.py

class EOSException(ManilaException):
    message = _("EOS exception occurred: %(msg)s")
```

5. Add "EOS" as a share protocol constant.

```
vi ~/manila/manila/common/constants.py

SUPPORTED_SHARE_PROTOCOLS = (
```

```
    'NFS', 'CIFS', 'GLUSTERFS', 'HDFS', 'CEPHFS', 'MAPRFS', 'EOS')
```

6. Restart all Manila services.

```
$ sudo systemctl restart devstack@m*
```

c. CONFIGURING THE EOS MANILA DRIVER

Before beginning these series of steps, ensure that the user logged into OpenStack is recognized as an admin user. Additionally, EOS must be integrated as a share protocol as described in the previous section.

1. Navigate to the OpenStack Manila drivers folder and clone the repository.

```
$ cd ~/manila/manila/share/drivers/eos-manila
$ git clone https://github.com/cern-eos/eos-manila.git
```

2. Modify the bottom of Manila configuration file.

```
$ vi /etc/manila/manila.conf
```

3. Add a new stanza to manila.conf for the EOS Manila driver configuration:

```
[eos]
driver_handles_share_servers = False
share_backend_name = EOS
share_driver = manila.share.drivers.eos-manila.driver.EOSDriver
auth_key = BakTIcB08XwQ7vNvagi8 # arbitrary authentication key defined in
server
```

4. Enable the EOS Manila driver in the [DEFAULT] stanza of the manila.conf file and, if you have not aready done so, enable the EOS protocol.

```
#modify the other enabled share backends/protocols as neccesary
enabled_share_backends = eos
...
enabled_share_protocols = NFS,CIFS,EOS
```

5. Create a new share type for EOS.

```
$ manila type-create eos False --extra-specs share_backend_name=EOS
```

6. Restart all Manila services.

```
$ sudo systemctl restart devstack@m*
```

d. RUNNING THE SAMPLE GRPC SERVER

In the case that one would like to experiment with the capabilities of the EOS Manila driver without a dedicated server, the GitHub repository comes equipped with a sample GRPC server.

The GRPC server hosts "shares" in stack's home directory in a folder called 'eos_shares.' Here, the shares are organized according to which user created it. Each time a share is created, a share folder appears in the user's folder with a file indicating information about the share.

To run the server:

```
$ cd grpc_eos
$ python server.py
```

With each request, the server will print the parameters of the request as well as if the requests were successful or not.

7. CONCLUSION

OpenStack Manila is a powerful platform for shared file system management. Now that users are able to manage storage spaces hosted on the EOS file service through the platform, ease of use will increase across the organization. The addition of EOS to OpenStack also introduces a new realm of possibilities for EOS. With EOS GRPC, the file service has the potential to expand to new projects and services outside of the OpenStack platform. This preposition is exciting, foreshadowing a much more unified software toolkit for CERN researchers.

8. REFERENCES

[1] Alberto Di Meglio. CERN openlab white paper on future ICT challenges in scientific research, 2015. URL cds.cern.ch/record/2301895

[2]: CERN EOS Service Main Page. EOS Service, 2019. URL http://information-technology.web.cern.ch/services/eos-service

[3]: EOS Manila Driver GitHub Repository. CERN EOS, 2019. URL https://github.com/cern-eos/eos-manila

[4]: EOS Manila Development Branch. CERN EOS, 2019. URL https://gitlab.cern.ch/dss/eos/tree/wip-manila

[5]: Google. GRPC Documentation, 2019. URL https://grpc.io/docs

[6] Mihai Patrascoiu. Manila – OpenStack File Sharing Service, 2015. URL https://zenodo.org/record/33192#.XU182HUzYeN

[7] OpenStack community. Open source software for creating private and public clouds, 2019. URL http://www.openstack.org

[8] OpenStack community. OpenStack Manilla, 2019. URL https://wiki.openstack.org/wiki/Manila