UNIVERSITY OF PISA

Cloud Computing

Year 2020/2021

Option #2 – Ceph-based file manager implementation

Project specification

MORTEZA AREZOUMANDAN

FRANCESCO DEL TURCO

AHMED SALAH TAWFIK IBRAHIM

Link GitHub: https://github.com/FranDT/CloudComputingProjects/tree/master/CephProject

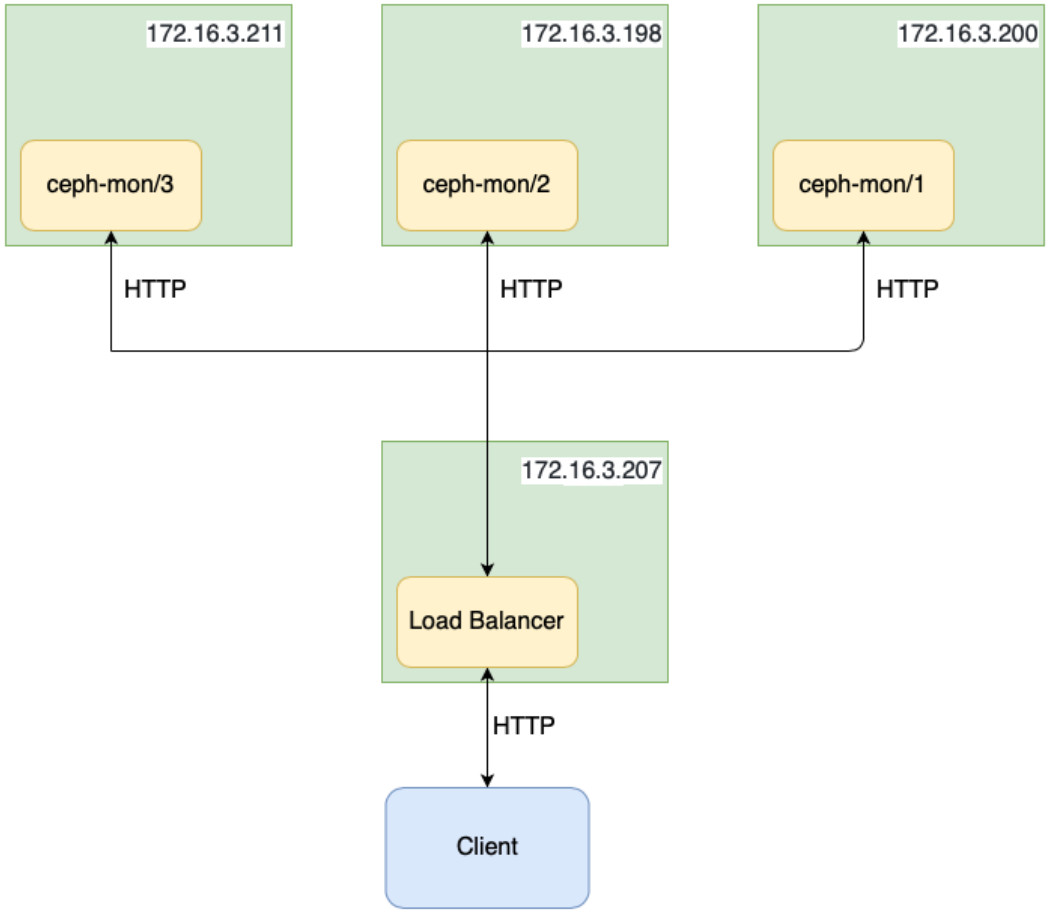
1.0 Introduction

The application we have developed is a distributed file storage system based on Ceph. The file storage exports a REST interface through which external users can perform the following operations:

- Retrieve the list of files currently stored in the system;
- Delete a file;
- Upload / Download a file;
- Shows current statistics on the status of the cluster.

2. 0 System Architecture

The application is composed by two layers: a frontend layer and a backend layer, that interacts through a load balancer, like in the following picture:

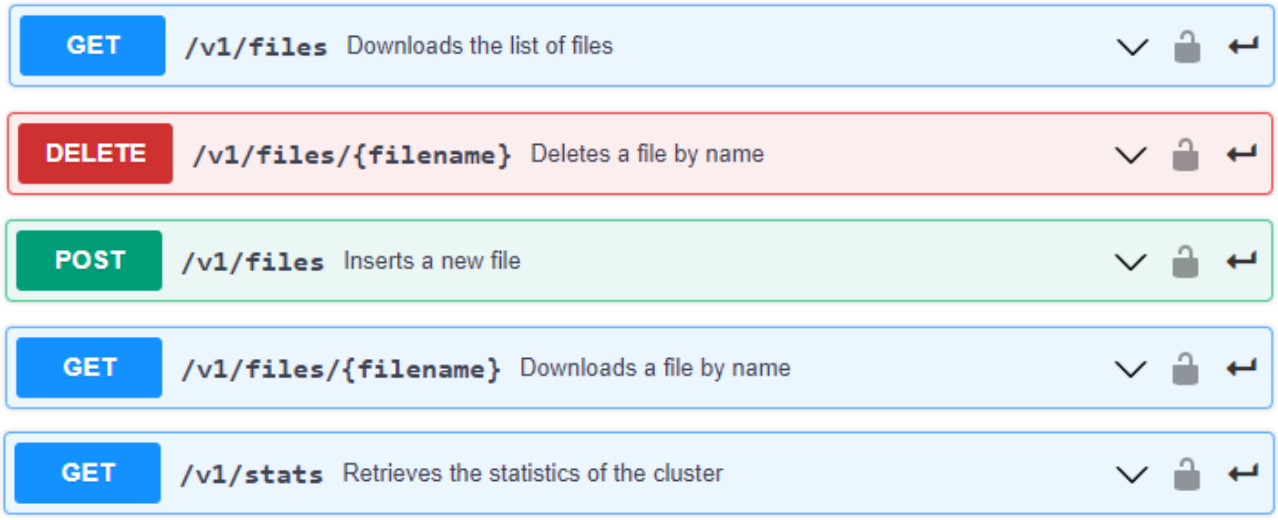


The HTTP requests from the users are sent to the load balancer which is in charge of distributing them efficiently across the three servers: this is performed by hashing the timestamp as the request is received by the load balancer, that will choose from a list of 3 IP addresses corresponding to the IPs of the machines on which the Ceph monitors are

running. When the destination is chosen, the request is forwarded to the designated Ceph-mon instance by exploiting the REST interface. The Ceph monitor instances are responsible to perform the requested operation.

2.1 Endpoint definition

As mentioned above, we provide a set of REST APIs, that are used to standardize the communication between the two layers. The methods provided are the following (they are showed in the order presented in paragraph 1.0):



In the POST, the data of the file in binary form are passed inside the body of the request: they are then decoded by the REST endpoint in the servers to extract the name of the file and the content of the file.

3.0 Implementation

To interact with Ceph, a module exploiting the librados python library has been written: this module is used in the REST endpoint on the server side and is therefore deployed on each Ceph-mon instance.

On another machine, a docker container is deployed for the load balancer. The load balancer is in charge of forwarding client requests to the servers: the load balancing is done by hashing the timestamp of the request. After that, in order for the load balancer and the server to communicate with each other, we need to activate port forwarding for each container hosting a Ceph-mon instance. Finally, a python client with a simple command line user interface has been implemented in order to ease the life of the user.

4.0 How to run the system

To run the system, there are a series of steps that must be performed. First of all, it is necessary to clone the git repository specified in the index page locally. After this, we need to move the server-related files to the containers hosting the monitors, that can be performed with the following command:

```
lxc file push <filePath> <containerName>/home/ubuntu
```

The files to move are “server_api.py” and “Backend.py”, while the container names are:

```
juju-f7b247-1-lxd-0
juju-f7b247-2-lxd-1
juju-f7b247-3-lxd-1
```

The first container is hosted on machine 172.16.3.200 and has IP equal to 252.3.200.9; the second container is hosted on machine 172.16.3.198 and has IP equal to 252.3.198.183; the third container is hosted on machine 172.16.3.211 and has IP equal to 252.3.211.113. Those IPs are needed since we need to implement port-forwarding on the juju containers hosting the Ceph-mon instances: this can be done with the command:

```
iptables -t nat -A PREROUTING -p tcp -i eth0 –dport 8080 -j DNAT –to-destination
<containerIP>:8080
```

This step has already been performed on our cluster.
After this, it is necessary to move the load balancer file named “LB.py” to the machine at IP 172.16.3.207 and we can do this with the following command:

```
scp <filePath> root@172.16.3.207:
```

After this command, we should create a new directory and move the file inside the directory, that will be used to deploy the Docker instance running the load balancer.

4.1 Backend

For each VM used as an OpenStack compute node, open the container shell for each ceph-mon instance as below:

1. For machine “172.16.3.200”:

   ```
   lxc exec juju-f7b247-1-lxd-0 /bin/bash
   ```

2. For machine “172.16.3.198”:

```
lxc exec juju-f7b247-2-lxd-1 /bin/bash
```

3. For machine “172.16.3.211”:

```
lxc exec juju-f7b247-3-lxd-1 /bin/bash
```

From there, run *server_api.py* in each node using the following command:

```
python3 /home/ubuntu/server_api.py
```

4.2 Load Balancer

The load balancer is run inside a Docker container, for which we need to create the Dockerfile and the requirements.txt file that will be used by the Dockerfile. We also need to create a monitors.txt file, containing the IP of the machines hosting the Ceph-mon instances: this file is used by the load balancer to decide to which machine it should forward each request.
The requirements.txt file is used by the Dockerfile to retrieve the list of libraries to be imported by the load balancer and must include “Flask” and “Werkzeug==0.16.1”.
Finally, the Dockerfile has the following format:

```
FROM python:3-alpine

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY requirements.txt /usr/src/app/

RUN pip3 install --no-cache-dir -r requirements.txt

COPY . /usr/src/app

EXPOSE 8080

ENTRYPOINT ["python3"]

CMD ["LB.py"]
```

We can at this point build the container with the following command:

```
docker build -t <dirName> .
```

We can then run the container exposing the port 8080 (which is hard coded inside the file as the port on which to receive the requests from the client):

```
docker run -p 8080:8080 -d load-balancer
```

4.3 Client

The client can be simply run using the following command in the folder containing the client file:

```
python3 CephClient.py
```

The client will start showing the list of commands the user can run, that are:

- commands -> prints the list of available commands;
- ls -> prints the list of files available in the storage;
- delete <filename> -> deletes the file with the corresponding filename;
- upload <filepath> -> uploads the file from the corresponding file path, that is then used as filename;
- download <filename> -> downloads the file with the corresponding filename;
- stats -> shows the current statistics on the status of the cluster;
- exit -> closes the client.