

Cloudrity Solution

Rados gateway - s3 - openstack swift

hungdh9

2019

CONTENT

[Chương 1. Giới thiệu về Object Storage 2](#_Toc16502422)

[1.1. Object Storage là gì? 2](#_Toc16502423)

[1.2. So sánh giữa Object Storage và Block Storage. 2](#_Toc16502424)

[Chương 2. Giới thiệu S3 và Openstack Swift 2](#_Toc16502425)

[2.1. S3 Interface là gì? 2](#_Toc16502426)

[2.2. Module Openstack Swift 2](#_Toc16502427)

[2.3. So sánh giữa S3 API và Swift API 2](#_Toc16502428)

[Chương 3. Cung cấp Object Storage với Ceph Backend 2](#_Toc16502429)

[3.1. Giới thiệu Rados gateway 2](#_Toc16502430)

[3.2. Cài đặt Rados Gateway với docker image 2](#_Toc16502431)

[3.3. Tích hợp Rados Gateway với Keystone 4](#_Toc16502432)

[3.3.1. Xác thực người dùng của Openstack với keystone 4](#_Toc16502433)

[3.3.2. Thiết lập kênh truyền SSL giữa Rados gateway và Keystone 8](#_Toc16502434)

[3.4. Cài đặt Rados Gateway trên cụm cloud HHT với Ansible 9](#_Toc16502435)

[Chương 4. Sử dụng Ceph Object Storage thông qua Rados gateway 9](#_Toc16502436)

[4.1. Truy vấn Rados gateway thông qua S3 interface 9](#_Toc16502437)

[4.2. Truy vấn Rados gateway thông qua Openstack Swift interface 9](#_Toc16502438)

[4.2.1. Swift client CLI 10](#_Toc16502439)

[4.2.2. Swift Dashboard 11](#_Toc16502440)

# Giới thiệu về Object Storage

## Object Storage là gì?

Object Storage là một cách thức lưu trữ dữ liệu bên cạnh Block storage. Định nghĩa object storage dựa trên cách thức tổ chức dữ liệu lưu trữ của nó.



Hình bên trên thể hiện cách tổ chức dữ liệu lưu trữ của object storage trên thiết bị lưu trữ vật lý. Với một file dữ liệu cần lưu trữ, kích thước có thể là từ vài MB cho tới hàng GB thì đều được lưu tới thiết bị lưu trữ như một thực thể duy nhất. Đơn vị lưu trữ này được gọi là một object. Mỗi object sẽ bao gồm 3 thành phần:

* Dữ liệu lưu trữ của chính nó.
* Metadata: các cặp key-value chứa thông tin của object, có thể là thông tin mô tả, thông tin xác thực, bảo mật ,…
* UID: một ID định danh duy nhất trên toàn thể hệ thống object storage. Với UID này hệ thống có thể xác định được ví trị của object.

Tất cả các object trong hệ thống được chứa trong một không gian lưu trữ phẳng, không có tính cấu trúc hay phân tầng.

## So sánh giữa Object Storage và Block Storage.

Trước khi đi vào so sánh, chúng ta sẽ xem lại cách tổ chức dữ liệu lưu trữ của Block storage.



Hình phía trên thể hiện cách thức tổ chức dữ liệu lưu trữ của block storage. Đơn vị lưu trữ ở đây là các block (có kích thước cố định). Với một file dữ liệu cần lưu trữ, file này sẽ được phân thành các mảnh nhỏ với kích thước cố định từ trước (các mảnh này được gọi là block). Filesystem có nhiệm vụ gắn các block này với các địa chỉ phân vùng lưu trữ trên thiết bị lưu trữ, đồng thời tạo cấu trúc, tính phân cấp cho file lưu trữ. Mỗi khi người dùng truy cập tới một file, filesystem sẽ lấy tất cả các block lưu trữ của file đó và thể hiện thành file dữ liệu tới người dùng.

Bảng dưới đây là thông tin so sánh giữa Object storage và Block storage:

|  |  |  |
| --- | --- | --- |
|  | **Block Storage** | **Object Storage** |
| **Metadata** | No | Metadata của Object storage chứa:   * Thông tin mô tả * Thông tin bảo mật, xác thực * vv… |
| **Accessibility** | Có thể truy cập tới và sửa đổi một phần của dữ liệu lưu trữ (block cần thiết sửa đổi) | Phải truy cập tới toàn bộ object mới có thể thay đổi dữ liệu.   * với object lớn sẽ gặp vấn đề về hiệu năng |
| **Storage data type** | Những dữ liệu có cấu trúc, được truy cập và thay đổi thường xuyên, liên tục.  VD: database… | Dữ liệu tĩnh, không cấu trúc, thường không bị thay đổi.  VD: ảnh, video, web content, backup data,… |
| **High Availability** | No | Yes |
| **Scalability** | Low | High |
| **Structure** | Yes | No |

# Giới thiệu S3 và Openstack Swift

## S3 là gì?

S3 (viết tắt của Simple Storage Service) là một dịch vụ object storage được cung cấp bởi Amazon. S3 cung cấp cho người dùng một tập API để phục vụ cho cho phát triển các ứng dụng, phần mềm của riêng mình. Hình dưới đây thể hiện cách thức quản lý object với S3 interface:



Với mỗi một S3 account, người dùng sẽ quản lý các Bucket phía dưới. Trong các bucket này chứa object hoặc folder với object bên trong. Như đã nói phía trên thì Object storage không có tính cấu trúc hay phân tầng, nên về bản chất không tồn tại thực thể folder. Folder ở đây được hiển thị tới người dùng chỉ thông qua cách đặt tên cho object.

Ví dụ: người dùng muốn một object có tên là meo\_tam\_the.txt chứa trong thư mục cac\_loai\_meo thì object này sẽ được đặt tên là cac\_loai\_meo/meo\_tam\_the.txt (dấu “/” cũng tồn tại trong tên object).

## Module Openstack Swift

Hoàn toàn tương tự như S3, Swift là một module cung cấp dịch vụ object storage được phát triển cho môi trường openstack cloud. Swift cũng có riêng một tập API cung cấp cho người dùng sử dụng. Hình dưới đây là cách mà Swift quản lý các object trong hệ thống:



Thay vì sử dụng khái niệm Bucket thì Swift sử dụng khái niệm Container để chứa các folder/object.

# Cung cấp Object Storage với Ceph Backend

## Giới thiệu Rados gateway

Như chúng ta đã biết, Ceph có thể cung cấp cho người dùng 3 loại dịch vụ lưu trữ là:

* Rados Block Device (rbd)
* Object Storage
* Ceph File System (CephFS)

Với Ceph Object Storage, người dùng có thể sử dụng dịch vụ lưu trữ này thông qua thành phần Rados Gateway(radosgw). Rados gateway có vai trò là một HTTP server cung cấp interface cho người dùng sử dụng dịch vụ object storage với storage backend là Ceph.



Về phía Ceph cluster, radosgw sử dụng tập thư viện Librados để giao tiếp với cụm. Còn ở phía người dùng, radosgw tích hợp 2 tập API của S3 và Swift cho phép tùy chọn sử dụng.

## Cài đặt Rados Gateway với docker image

Trước khi cài đặt Rados gateway, chúng ta cần phải tải về docker image của ceph-daemon với phiên bản mong muốn:

* Phiên bản luminous:

|  |
| --- |
| $ docker pull ceph/daemon:latest-luminous |

* Phiên bản mimic:

|  |
| --- |
| $ docker pull ceph/daemon:latest-mimic |

* Phiên bản nautilus:

|  |
| --- |
| $ docker pull ceph/daemon |

Sau khi đã tải về docker image với phiên bản mong muốn, tiếp theo sẽ là bước cấu hình cho rados gateway. Rados gateway có thể được cấu hình thông qua hai cách như sau:

* Sử dụng trực tiếp file cấu hình được truyền vào docker container chạy rados gateway (thường là /etc/ceph/ceph.conf) .
* Rados gateway container lấy các thông số cấu hình từ service etcd. (Cách này được khuyến khích dùng trong môi trường triển khai).

Trong tài liệu này, chúng ta chỉ thực hiện cấu hình cho Rados gateway thông qua dịch vụ etcd. Để cấu hình thông qua etcd, thực hiện theo các bước sau:

* Truy cập tới hệ thống chạy dịch vụ etcd (thông qua SSH, hoặc docker container)
* Sử dụng giao diện dòng lệnh điều khiển của etcd thực hiện cấu hình:

|  |
| --- |
| $ etcdctl rm /ceph-config/ceph/client\_host  $ etcdctl mkdir /ceph-config/ceph/client\_host  $ etcdctl mkdir /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_frontends "civetweb port=192.168.30.166:7482" |

Trong đó:

* rgw\_frontends: cấu hình địa chỉ, port của radosgw front end.
* Các trường cấu hình sẵn có dùng cho Rados gateway có thể tìm tại đây: <https://docs.ceph.com/docs/master/radosgw/config-ref/>

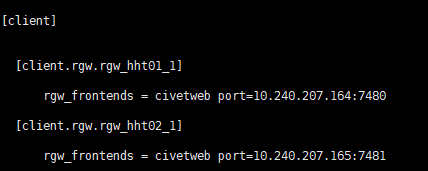
Khởi chạy rados gateway container với tùy chọn sử dụng etcd làm dịch vụ lưu trữ thông tin cấu hình:

|  |
| --- |
| $ docker run --name <radosgw\_container\_name> -d --net=host \  -e KV\_TYPE=etcd \  -e KV\_IP=<etcd\_service\_ip\_address> \  -e RGW\_NAME=<name\_of\_radosgw\_in\_ceph\_cluster> \  -v /var/lib/ceph/:/var/lib/ceph \  ceph/daemon:<version\_tag> rgw |

Trong đó:

* radosgw\_container\_name: tên của container chạy dịch vụ radosgw.
* etcd\_service\_ip\_address: địa chỉ IP của dịch vụ etcd.
* name\_of\_radosgw\_in\_ceph\_cluster: tên của client mà radosgw sẽ sử dụng để giao tiếp với ceph cluster.: tên của client mà radosgw sẽ sử dụng để giao tiếp với ceph cluster.
* version\_tag: tag của image tương ứng với phiên bản mong muốn cài đặt.

Sau khi docker container của rados gateway hoạt động, truy cập vào trong container và kiểm tra file cấu hình, /etc/ceph/ceph.conf, cấu hình đúng sẽ như sau:



## Build lại docker image cho Ceph-daemon

Trong trường hợp cần sửa đổi Ceph-daemon docker image cho phù hợp với môi trường cài đặt, chúng ta sẽ cần phải build lại image này. Phần này là một ví dụ cho việc build lại docker image ceph-daemon phiên bản luminous.

Tạo thư mục build docker image và di chuyển tới thư mục đó:

|  |
| --- |
| $ mkdir ceph\_luminous\_docker\_image && cd ceph\_luminous\_docker\_image |

Sau đó tải về source của ceph-container với branch latest-version( lastest luminous):

|  |
| --- |
| $ git clone <https://github.com/ceph/ceph-container.git> -b latest-version |

Tải về file forego-stable-linux-amd64.tgz và đặt ở thư mục hiện tại:

|  |
| --- |
| $ wget –O forego-stable-linux-amd64.tgz 'https://bin.equinox.io/c/ekMN3bCZFUn/forego-stable-linux-amd64.tgz' |

Tạo file Dockerfile với nội dung như sau:

|  |
| --- |
| # CEPH DAEMON IMAGE  # CEPH VERSION: Luminous  # CEPH VERSION DETAIL: 12.x.x  FROM ubuntu:16.04  MAINTAINER hungdh9 "hungdh9@mail.viettel.com"  ENV CEPH\_VERSION luminous  ENV CONFD\_VERSION 0.10.0  ENV KUBECTL\_VERSION v1.6.0  # Install neccessary package  RUN apt-get update  RUN apt-get install -y ca-certificates  RUN apt-get install -y vim net-tools git python-pip  COPY forego-stable-linux-amd64.tgz /forego.tgz  RUN mkdir -p /opt/ceph-container/bin/  # Packages list  ARG PACKAGES="ceph-mon ceph-osd ceph-mds ceph-mgr ceph-base ceph-common radosgw rbd-mirror sharutils etcd s3cmd nfs-ganesha nfs-ganesha-ceph nfs-ganesha-rgw lvm2"  ARG PURGES="/var/lib/apt/lists/\* /tmp/\* /var/tmp/\* /usr/lib/{dracut,locale,systemd,udev} /usr/bin/hyperkube /usr/bin/etcd /usr/bin/systemd-analyze /etc/{udev,selinux} /usr/lib/{udev,systemd}"  # Add bootstrap script, ceph defaults key/values for KV store  #RUN cd ceph-container && git checkout -b lastest-version && cd ..  ADD ./ceph-container/src/daemon/[s3cfg](https://github.com/ceph/ceph-container/blob/latest-version/src/daemon/s3cfg) /root/.s3cfg  ADD ./ceph-container/src/daemon/\*.sh /  ADD ./ceph-container/src/daemon/ceph.defaults /  ADD ./ceph-container/src/daemon/check\_zombie\_mons.py /  ADD ./ceph-container/src/daemon/osd\_scenarios/\* /  ADD ./ceph-container/src/daemon/entrypoint.sh.in /  ADD ./ceph-container/src/daemon/disabled\_scenario /  # install prerequisites  RUN DEBIAN\_FRONTEND=noninteractive apt-get update && apt-get install -y wget unzip uuid-runtime python-setuptools udev dmsetup && \  # Install ceph, ganesha and etcd  wget -q -O- 'https://download.ceph.com/keys/release.asc' | apt-key add - && \  echo "deb http://download.ceph.com/debian-$CEPH\_VERSION/ xenial main" | tee /etc/apt/sources.list.d/ceph-$CEPH\_VERSION.list && \  echo "deb http://download.ceph.com/nfs-ganesha/deb-V2.5-stable/luminous/ xenial main" | tee /etc/apt/sources.list.d/nfs-ganesha.list && \  apt-get update && apt-get install -y --no-install-recommends --force-yes $PACKAGES && \  dpkg -s $PACKAGES && \  apt-get clean && rm -rf $PURGES && \  # Download & install confd  wget -O /usr/local/bin/confd "https://github.com/kelseyhightower/confd/releases/download/v${CONFD\_VERSION}/confd-${CONFD\_VERSION}-linux-amd64" && \  chmod +x /usr/local/bin/confd && mkdir -p /etc/confd/conf.d && mkdir -p /etc/confd/templates && \  # Install forego  cd /usr/local/bin && tar xfz /forego.tgz && chmod +x /usr/local/bin/forego && rm /forego.tgz && \  # Install kubectl  wget -O /usr/local/bin/kubectl "https://storage.googleapis.com/kubernetes-release/release/${KUBECTL\_VERSION}/bin/linux/amd64/kubectl" && \  chmod +x /usr/local/bin/kubectl && \  # Cleaning container  #bash /clean\_container.sh && rm /clean\_container.sh  echo hungdh9 bypass clean\_container.sh  # Modify the entrypoint  RUN bash "/generate\_entrypoint.sh" && \  rm -f /generate\_entrypoint.sh && \  bash -n /\*.sh  # Add templates for confd  ADD ./ceph-container/src/daemon/confd/templates/\* /etc/confd/templates/  ADD ./ceph-container/src/daemon/confd/conf.d/\* /etc/confd/conf.d/  # Add volumes for Ceph config and data  VOLUME ["/etc/ceph","/var/lib/ceph", "/etc/ganesha"]  # Execute the entrypoint  WORKDIR /  ENTRYPOINT ["/entrypoint.sh"]  LABEL head=d0e6104 |

Cuối cùng là thực hiện build và push image tới docker registry:

|  |
| --- |
| $ docker build -t <image\_name>:<image\_tag> [--build-arg http\_proxy=<proxy\_address>:<proxy\_port>] [--build-arg https\_proxy=<proxy\_address>:<proxy\_port>] .  $ docker push <image\_name>:<image\_tag> |

## Tích hợp Rados Gateway với Keystone

### Xác thực người dùng của Openstack với keystone

Thông thường, rados gateway sẽ tự quản lý user và thực hiện xác thực user:password cho các truy vấn được gửi tới. Để phù hợp với môi trường openstack cloud, Rados gateway có hỗ trợ tùy chọn tích hợp với dịch vụ Keyston. Khi tích hợp với Keystone, mỗi project/tenant của openstack sẽ tương ứng với một user trên Rados gateway. Quá trình xác thực sẽ diễn ra như sau:



* 1: trước khi thực hiện truy vấn tới rados gateway thông qua swift api, user’s client sẽ lấy về token từ Keystone.
* 2: client gửi truy vấn kèm theo token nhận được tới Rados gateway.
* 3: Rados gateway sử dụng API của Keystone để xác thực token mới nhận được.
* 4: xác thực token thành công, Rados gateway kiểm tra quyền truy cập của người dùng. Thiết lập, cấu hình cho quyền truy cập tài nguyên object storage của người dùng sẽ được trình bày ở phần sau của tài liệu. Cũng tại bước này nếu người dùng tương ứng với project của user chưa tồn tại thì Rados gateway sẽ tạo mới một user với id là id của project.
* 5: quyền truy cập được chấp thuận, Rados gateway thực hiện các tác vụ được chỉ thị với Ceph backend.

Sau đây là các bước thiết lập cấu hình cho Rados gateway tích hợp với dịch vụ Keystone của Openstack:

* Tạo người dùng với role admin cho dịch vụ object storage, nếu dịch vụ Swift đã được cài đặt từ trước thì có thể bỏ qua bước này.

|  |
| --- |
| $ openstack user create --project admin --password secret rgw-sync  $ openstack role add --user rgw-sync --project admin admin |

* Tạo service cho dịch vụ object storage, nếu dịch vụ Swift đã được cài đặt từ trước thì có thể bỏ qua bước này:

|  |
| --- |
| $ openstack service create --name=swift --description="Swift RadosGW Service" object-store |

* Nếu dịch vụ Swift đã được cài đặt từ trước thì cần phải xóa bỏ những endpoint cũ:

|  |
| --- |
| $ openstack endpoint list | grep swift | awk ‘{print $2}’ | xargs openstack endpoint delete |

* Tạo endpoint mới cho dịch vụ object storage trỏ tới frontend của Rados gateway:

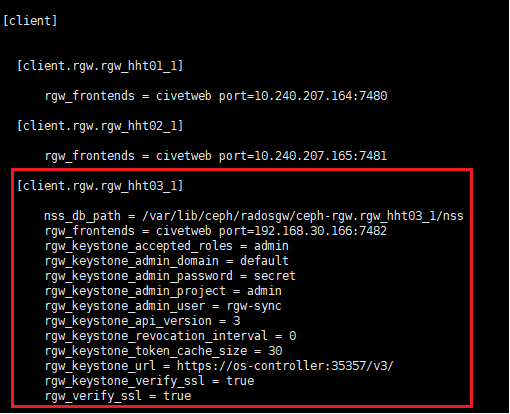
|  |
| --- |
| $ openstack endpoint create --region RegionOne swift public http://192.168.30.166:7482/swift/v1  $ openstack endpoint create --region RegionOne swift internal http://192.168.30.166:7482/swift/v1  $ openstack endpoint create --region RegionOne  admin http://192.168.30.166:7482/swift/v1 |

* Thêm thông số cấu hình cho Rados gateway thông qua etcd:

|  |
| --- |
| $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_api\_version 3  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_url https://os-controller:35357/v3/  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_admin\_user rgw-sync  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_admin\_password secret  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_accepted\_roles admin  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_admin\_domain default  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_admin\_project admin  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_token\_cache\_size 30  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_revocation\_interval 0  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_verify\_ssl true  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_verify\_ssl true |

Trong đó:

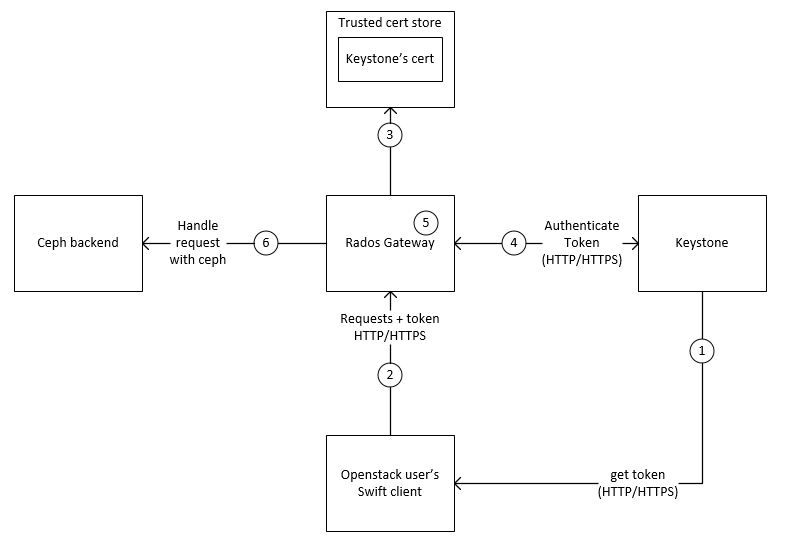
* rgw\_keystone\_api\_version: phiên bản API của keystone được radosgw sử dụng.
* gw\_keystone\_url: đường dẫn truy cập dịch vụ keystone.
* rgw\_keystone\_admin\_user: tên người dùng openstack với role admin. Người dùng nãy đã được tạo ở bước phía trên.
* rgw\_keystone\_admin\_password: mật khẩu tương ứng với người dùng trên.
* rgw\_keystone\_accepted\_roles: người dùng với các role được khai báo ở trường này sẽ được phép truy cập Rados gateway thông qua Swift API.
* rgw\_keystone\_admin\_domain: thông tin domain của người dùng.
* rgw\_keystone\_admin\_project: project/tenant mà người dùng phía trên thuộc về.
* rgw\_keystone\_revocation\_interval: khoảng thời gian định kỳ mà radosgw sẽ kiểm tra xem token đã bị thu hồi hay chưa.
* rgw\_keystone\_verify\_ssl: trường này xác định xem radosgw có thực hiện xác thực ssl certificate của keystone hay không.
* Sau khi cấu hình xong, thực hiện restart các container đang chạy rados gateway. Kiểm tra file cấu hình sau khi radosgw hoạt động trở lại, file cấu hình đúng sẽ như sau:



* Cuối cùng là thực hiện restart lại container đang chạy radosgw.

### Thiết lập kênh truyền SSL giữa Rados gateway và Keystone

Để đảm bảo tính bảo mật cho kênh truyền xác thực giữa Rados gateway và Keystone, việc xác thực SSL được áp dụng.



Tương tự như sơ đồ các bước xác thực token giữa Rados gateway và Keystone phía trên, ở đây có thêm một bước số 3: xác thực certificate do keystone gửi tới trước khi mã hóa dữ liệu và gửi tới keystone xác thực.

Trong quá trình lab hiện tại, các trường cấu hình của radosgw phục vụ cho việc xác thực SSL certificate của keystone không hoạt động. Vì vậy người thực hiện đã viết thêm một webservice đứng giữa keystone và radosgw để thực hiện truy vấn https thay cho radosgw. Webservice này đã được tích hợp để đọc chung file cấu hình với radosgw và được build cùng với ceph-daemon docker image. Khi image được chạy thì cả radosgw và webservice đồng thời được khởi chạy.



Dưới đây là phần cấu hình cho radosgw xác thực SSL certificate của keystone thông qua dịch vụ etcd:

|  |
| --- |
| $ etcdctl mk /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/keystone\_cert\_path /var/lib/ceph/radosgw/openstack.crt  $ etcdctl mk /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/keystone\_endpoint\_for\_intermediate\_keystone <https://os-controller:35357/v3>  $ etcdctl update /ceph-config/ceph/client\_host/client.rgw.rgw\_hht03\_1/rgw\_keystone\_url http://192.168.30.166:3004/fake\_authen |

Trong đó:

* keystone\_cert\_path: đường dẫn tới file cert của keystone bên trong docker image.
* keystone\_endpoint\_for\_intermediate\_keystone: đường dẫn thực sự tới dịch vụ keystone.
* rgw\_keystone\_url: trường cấu hình này giờ đây sẽ trỏ tới địa chỉ truy cập của webservice trung gian. Phần uri hiện tại phải đặt cố định là fake\_authen. URL: http://<address>:<port>/fake\_authen

Sau đó thực hiện restart lại container đang chạy radosgw.

# Sử dụng dịch vụ Ceph Object Storage thông qua Rados gateway

## Người dùng Openstack cloud

Phần hướng dẫn sử dụng rados gateway đối với người dùng Openstack cloud sau đây chỉ có thể thực hiện được sau khi tích hợp Rados gateway với dịch vụ Keystone (hướng dẫn tại mục 3.4).

### Truy vấn Rados gateway thông qua S3 interface

Nội dung phần này sẽ hướng dẫn truy vấn Rados gateway với S3 interface bằng cách sử dụng tập thư viện Python được xây dựng dựa trên S3 API. Để sử dụng thư viện của các ngôn ngữ khác như C++, C#, Java, Perl, PHP, Ruby thì có thể tìm hiểu tại đây: <https://docs.ceph.com/docs/master/radosgw/s3/>

Cài đặt gói thư viện python sử dụng để truy vấn S3 API:

|  |
| --- |
| $ sudo pip install boto |

Vì là người dùng thuộc openstack cloud nên đầu tiên cần phải lấy về thông tin xác thực cho người dùng, bao gồm 2 giá trị là access-key và secret-key.

* Thiết lập thông tin tài khoản được sử dụng:

|  |
| --- |
| $ export OS\_PROJECT\_DOMAIN\_NAME=Default  $ export OS\_USER\_DOMAIN\_NAME=Default  $ export OS\_PROJECT\_NAME=admin  $ export OS\_TENANT\_NAME=admin  $ export OS\_USERNAME=hungdh9  $ export OS\_PASSWORD=1  $ export OS\_AUTH\_URL=https://os-controller:35357/v3  $ export OS\_IDENTITY\_API\_VERSION=3  $ export OS\_IMAGE\_API\_VERSION=2 |

* Thực hiện câu lệnh sau với openstack client để lấy về access-key và secret key:

|  |
| --- |
| $ openstack --os-interface public ec2 credentials create |

Kết quả trả về:



Với access-key và secret-key vừa lấy về, thực hiện truy vấn tới Rados gateway

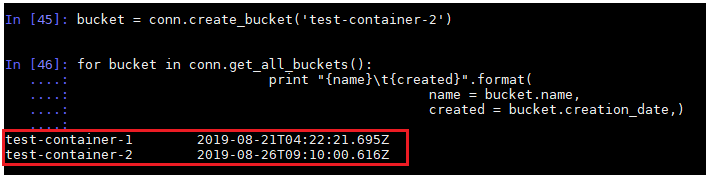
* Khởi tạo kết nối tới rados gateway với đoạn code sau:

|  |
| --- |
| import boto  import boto.s3.connection  access\_key = ‘2101bc8f13f0401e90a3712942b852c4’  secret\_key = ‘8c2a53d17f0948318710b930f9aac4e3’  conn = boto.connect\_s3(  aws\_access\_key\_id = access\_key,  aws\_secret\_access\_key = secret\_key,  host = '192.168.30.166',  is\_secure=False,  calling\_format = boto.s3.connection.OrdinaryCallingFormat(),  port=7482) |

* Tạo một bucket mới và liệt kê các bucket mà tài khoản đang sở hữu:

|  |
| --- |
| bucket = conn.create\_bucket('test-container-2')  for bucket in conn.get\_all\_buckets():  print "{name}\t{created}".format(  name = bucket.name,  created = bucket.creation\_date,) |

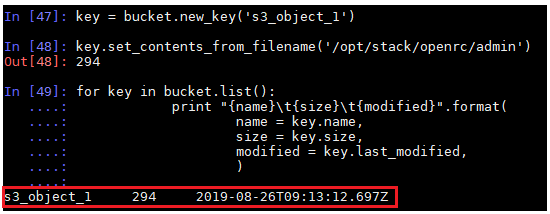
Kết quả trả về như sau:



* Tạo một object thuộc bucket mới tạo, dữ liệu của object lấy từ nội dung của file */opt/stack/openrc/admin*, sau đó liệt kê các object mà bucket sở hữu:

|  |
| --- |
| key = bucket.new\_key('s3\_object\_1')  key.set\_contents\_from\_filename('/opt/stack/openrc/admin')  for key in bucket.list():  print "{name}\t{size}\t{modified}".format(  name = key.name,  size = key.size,  modified = key.last\_modified,  ) |

Kết quả trả về:



Trên đây là một số ví dụ thao tác với bucket, object bằng cách sử dụng tập thư viện Python boto. Để tìm hiểu thêm về thư viện boto, người dùng có thể xem tại đây:

* Boto tutorial: <http://docs.pythonboto.org/en/latest/s3_tut.html>
* Boto document: <http://docs.pythonboto.org/en/latest/ref/s3.html>

### Truy vấn Rados gateway thông qua Openstack Swift Interface

#### Swift API

Nội dung phần này sẽ bao gồm một số ví dụ thực hiện truy vấn Rados gateway thông qua tập API của Swift đối với người dùng thuộc openstack cloud.

Vì người dùng thuộc môi trường openstack cloud nên mọi truy vấn tới rados gateway đều được xác thực một lần nữa với keystone. Việc xác thực đều dựa trên token mà keystone cấp cho người dùng, token này sẽ được gửi kem trong header của mọi truy vấn. Vì vậy việc đầu tiên cần làm là xác thực người dùng với keystone và lấy về token.

|  |
| --- |
| Keystone API: /v3/auth/tokens  Method: GET  Request’s body: { "auth": {  "identity": {  "methods": ["password"],  "password": {  "user": {  "name": "test\_user\_1",  "domain": { "id": "default" },  "password": "1"  }  }  },  "scope": {  "project": {  "name": "test\_project\_1",  "domain": { "id": "default" }  }  }  }  } |

Với truy vấn thành công thì status code trả về sẽ là 201 và token được chứa tại trường “X-Subject-Token” trong header của gói tin trả về:

|  |
| --- |
| Reponse’s header: {'Content-Length': '3861', 'X-Subject-Token': 'gAAAAABdY5D4y5yfWoVgygnCHgkHnTztwEd1BOBmANARKGo2PhmMGnSD7taLf9f8Z-WdfI\_7FpgNYLRLL\_IYKIhAkUPFF-rZx25dNUde1hb6seSN4\_zhasBuI\_eEhc1s3qTUlSeWz4Xd\_8SlSj5w1AxSa7uTq9poeQdLbIHOxXYtzLQhpKEq5CU', 'Vary': 'X-Auth-Token', 'Server': 'Apache/2.4.18 (Ubuntu)', 'Connection': 'close', 'Date': 'Mon, 26 Aug 2019 07:57:44 GMT', 'Content-Type': 'application/json', 'x-openstack-request-id': 'req-c98c6647-484f-4a19-81b8-d716b1d92d04'} |

Sau khi lấy được token xác thực, người dùng có thể sử dụng tập API của Swift để truy vấn tới Rados gateway. Thông tin về Swift API có thể tìm thấy tại đây: [https://docs.openstack.org/api-ref/object-store/index.html?expanded=show-account-details-and-list-containers-detail,get-object-content-and-metadata-detail#accounts](https://docs.openstack.org/api-ref/object-store/index.html?expanded=show-account-details-and-list-containers-detail,get-object-content-and-metadata-detail%23accounts)

#### Swift client CLI

Để thuận tiện truy vấn Rados gateway thông qua Openstack Swift interface, chúng ta sẽ cài đặt gói python client của Swift:

|  |
| --- |
| $ pip install python-swiftclient==3.8.0  $ apt-get install python-keystonemiddleware -y |

Sau đây là hướng dẫn sử dụng dịch vụ object storage trên môi trường Openstack cloud thông qua giao diện dòng lệnh của Swift:

* Trước tiên là thiết lập thông tin tài khoản được sử dụng để truy vấn tới Rados gateway:

|  |
| --- |
| $ export OS\_PROJECT\_DOMAIN\_NAME=Default  $ export OS\_USER\_DOMAIN\_NAME=Default  $ export OS\_PROJECT\_NAME=admin  $ export OS\_TENANT\_NAME=admin  $ export OS\_USERNAME=hungdh9  $ export OS\_PASSWORD=1  $ export OS\_AUTH\_URL=https://os-controller:35357/v3  $ export OS\_IDENTITY\_API\_VERSION=3  $ export OS\_IMAGE\_API\_VERSION=2 |

* Xác thực thông tin tài khoản với Rados gateway:

|  |
| --- |
| $ swift auth |



* Tạo container mới, sau đó kiểm tra các container của người dùng hiện đang tồn tại:

|  |
| --- |
| $ swift post <container\_name>  $ swift list |



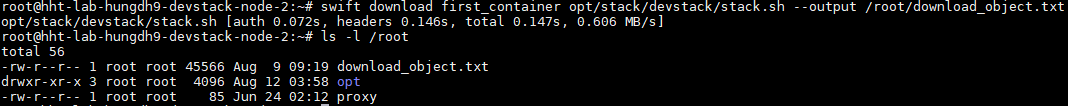
* Upload dữ liệu thành một object tới container vừa tạo, sau đó kiểm tra các object thuộc container đó:

|  |
| --- |
| $ swift upload <container\_name> <file\_path/directory\_path>  $ swift list <container\_name> |



* Tải về object vừa mới được upload:

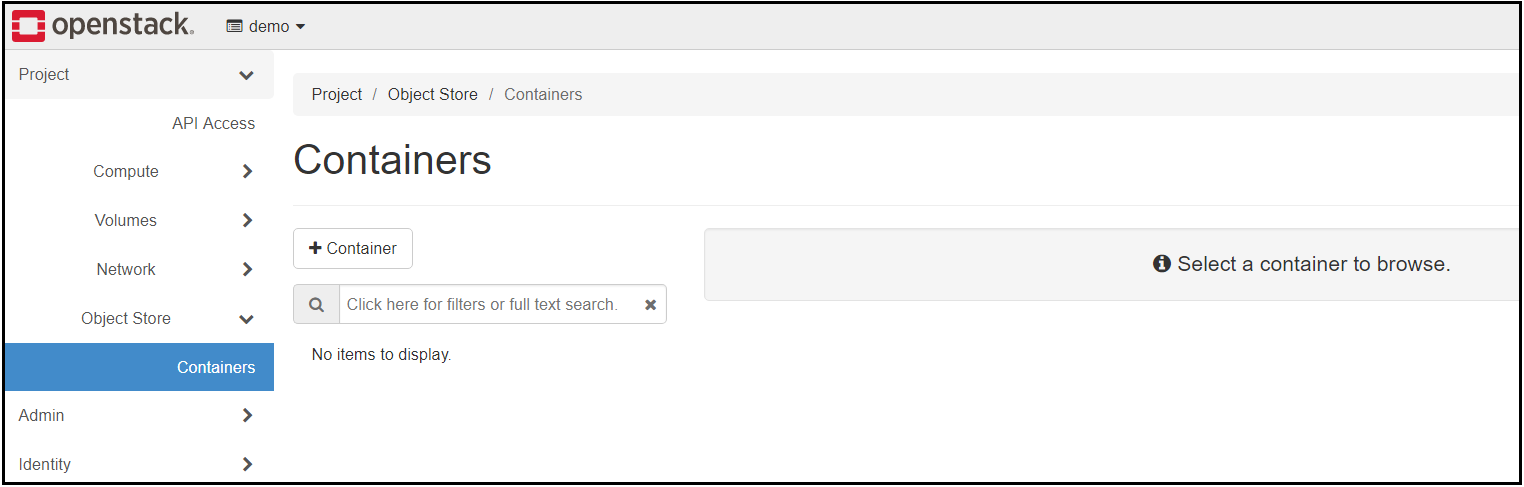
|  |
| --- |
| $ swift download <container\_name> <object\_name> --output <out\_put\_file\_path> |



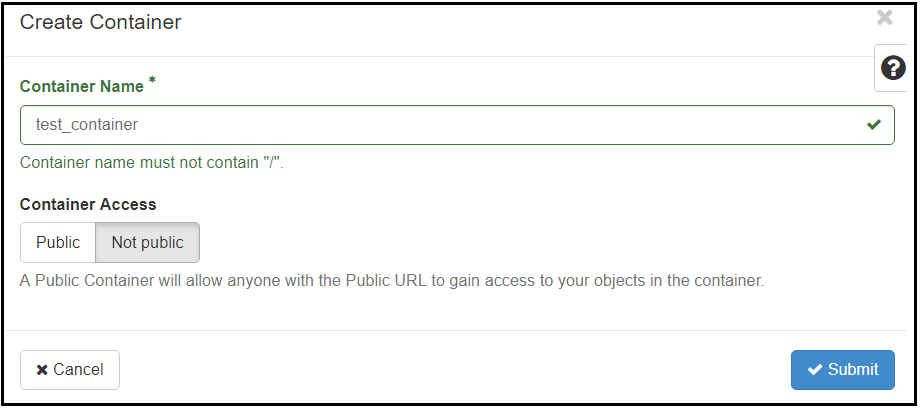
#### Swift Dashboard

Module swift có cung cấp giao diện quản lý cho dịch vụ object storage, được tích hợp với gói giao diện horizon. Sau đây là hướng dẫn sử dụng Swift dashboard:

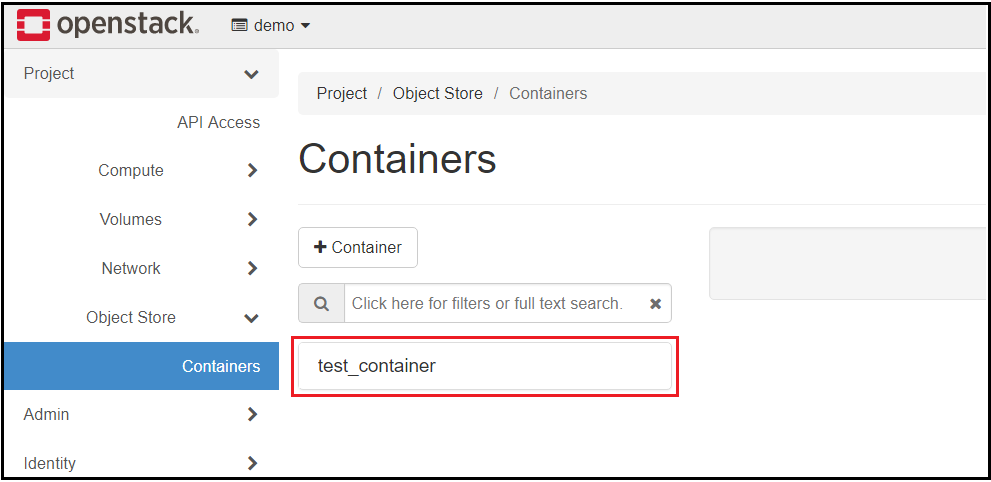
* Đăng nhập vào dashboard Horizon của openstack với người dùng hiện có, sau đó chọn tab Project > Object Store > Containers, giao diện quản lý của Swift như sau:



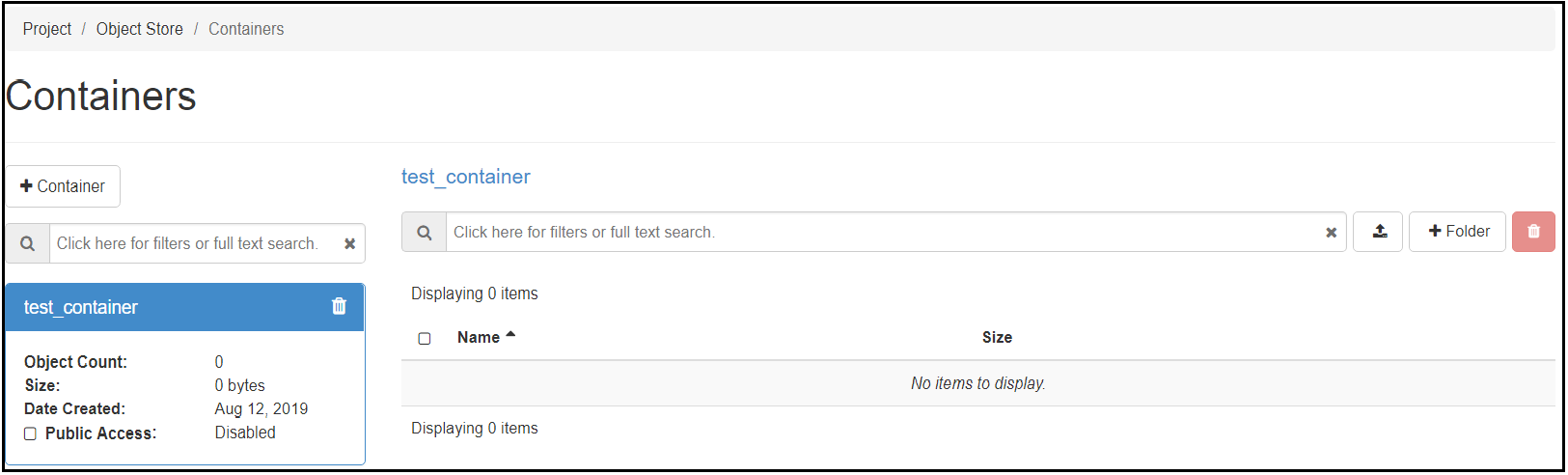
* Tạo mới container. Click chọn , điền tên container muốn tạo, thiết lập public/private cho container, sau đó chọn “Submit” để hoàn tất tạo container:



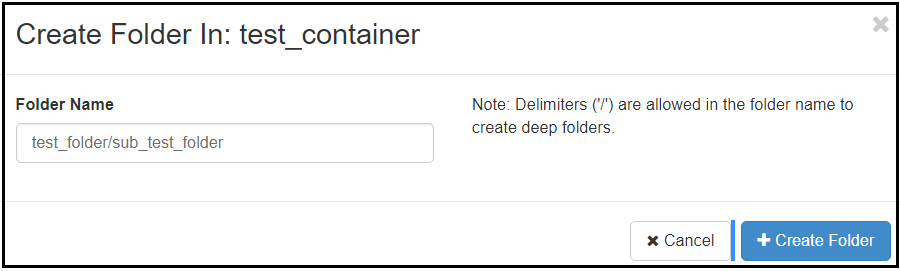
Kết quả tạo thành công, container mới sẽ xuất hiện ngay phía dưới giao diện quản lý:



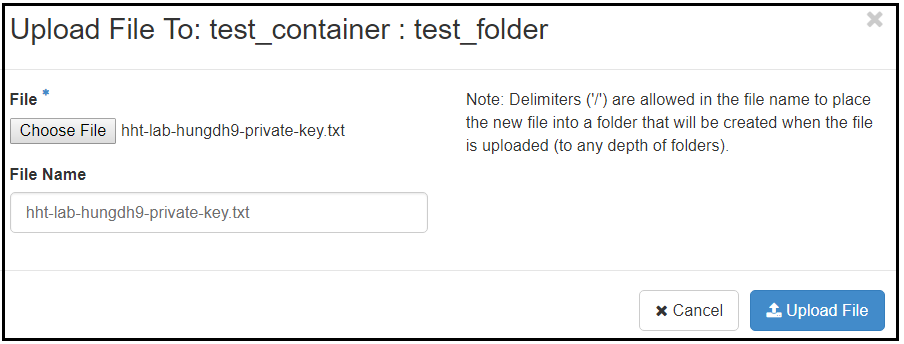
* Click chọn container vừa tạo để xem thông tin chi tiết:



* Trên giao diện cho phép tạo thư mục bên trong container. Chọn biểu tượng , nhập tên thư mục, có thể tạo them nhiều thư mục con bên trong bằng cách sử dụng dấu “/” làm ký hiệu phân cấp. Chọn “Create Folder” để hoàn tất:



* Di chuyển tới thư mục vừa tạo, click chọn để upload một object. Chọn file được upload, sau đó click “Upload File” để hoàn tất:



* Để tải về hay xóa một object, chỉ cần chọn action tương ứng với object đó:



## Người dùng thông thường

### Tạo người dùng với radosgw-admin

Phần này chỉ đề cập tới thao tác tạo người dùng của công cụ radosgw-admin, những kiểu thao tác khác không được đề cập. Chi tiết về sử dụng radosgw-admin của người dùng quản trị viên có thể xem thêm tại đây: [https://docs.ceph.com/docs/master/radosgw/admin/#](https://docs.ceph.com/docs/master/radosgw/admin/%23)

Với những người dùng không thuộc openstack cloud thì sẽ chỉ cần xác thực bởi radosgw. Những tài khoản người dùng này được admin tạo và cấp phát cho người dùng. Rados gateway quản lý hai kiểu người dùng thông thường là:

* User: tương ứng với một người dùng của S3 interface.
* Subuser: tương ứng với người dùng của Openstack Swift.

**Tạo một User**

Để tạo một User tương ứng với tài khoản S3, câu lệnh được sử dụng có cú pháp như sau:

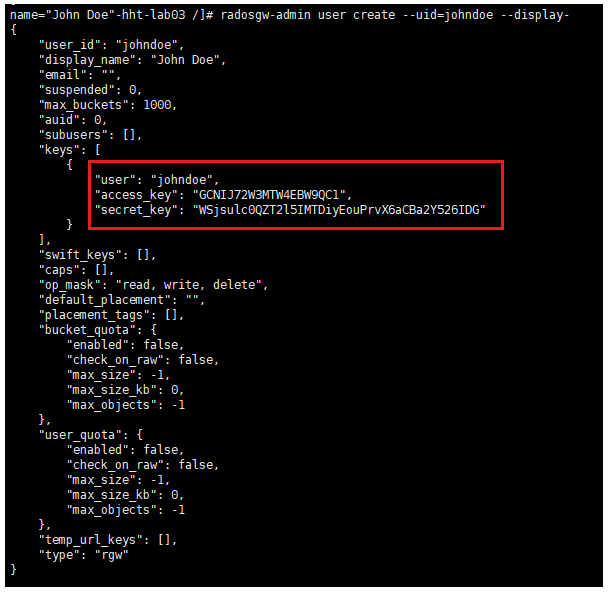
|  |
| --- |
| $ radosgw-admin user create --uid=<username> --display-name="< display-name >" [--email=<email\_address>] |

Ví dụ:

Kết nối tới docker container ceph-mon rồi thực hiện câu lệnh sau.

|  |
| --- |
| $ radosgw-admin user create --uid=johndoe --display-name="John Doe" |

Kết quả trả về:



Thông tin của user, access\_key và secret\_key sẽ được cung cấp cho người dùng để xác thực truy vấn tới radosgw.

**Tạo Subuser**

Để tạo một Subser tương ứng với tài khoản Swift, câu lệnh được sử dụng có cú pháp như sau:

|  |
| --- |
| $ radosgw-admin subuser create --uid=<user\_id> --subuser=<subuser\_id> --access=[read | write | readwrite | full] |

Ví dụ:

Kết nối tới docker container ceph-mon rồi thực hiện câu lệnh sau.

|  |
| --- |
| $ radosgw-admin subuser create --uid=hungdh9 --subuser=hungdh9:swift --access=full |

Kết quả trả về:



Thông tin người dùng trong trường “swift\_keys” được cung cấp cho người dùng để truy vấn tới rados gateway thông qua Swift interface. Có thể thấy thông tin người dùng còn bao gồm thông tin tài khoản tương ứng với S3 interface: access\_key và secret\_key. Người dùng hoàn toàn có thể sử dụng S3 interface để truy vấn tới tài nguyên mà tài khoản subuser đang sở hữu.

### Truy vấn Rados gateway thông qua S3 interface

Với thông tin về access\_key và secret\_key do quản trị viên cung cấp, người dùng hoàn toàn có thể truy vấn tới Rados gateway thông qua S3 interface tương tự như đã được giới thiệu tại phần 4.1.1.

### Truy vấn Rados gateway thông qua Openstack Swift Interface

Phần này sẽ sử dụng tập thư viện Python Swift client để giao tiếp với radosgw thông qua Swift interface.

Cài đặt gói python swift client:

|  |
| --- |
| $ pip install python-swiftclient==3.8.0  $ apt-get install python-keystonemiddleware -y |

Tutorial sử dụng thư viện python swift client có thể xem tại đây: <https://docs.ceph.com/docs/master/radosgw/swift/python/>

# Thao tác triển khai & vận hành dịch vụ Ceph Object Storage cho người dùng quản trị

## Triển khai Ceph Rados Gateway với Ansible

Hiện tại rados gateway đang được cài đặt trên cụm cloud HHT với mô hình triển khai như sau:



Trên mỗi node sẽ chạy một container của radosgw. Vì các radosgw có thể hoạt động active-active với nhau nên để đảm bảo HA cho dịch vụ chỉ cần sử dụng tới HAProxy. HAProxy sẽ thực hiện loadbalance cho các radosgw container trên 3 node.

Bộ cài ansible cho Ceph rados gateway hiện tại có hai kịch bản như sau:

* ceph\_rgw.yml: kịch bản này thực hiện xóa bỏ các radosgw instance hiện tại, sau đó tạo mới hoàn toàn các radosgw instance thuộc cụm Ceph.
* ceph-rgw-update-config.yml: kịch bản này thực hiện update cấu hình mới cho các radosgw instance đang chạy.

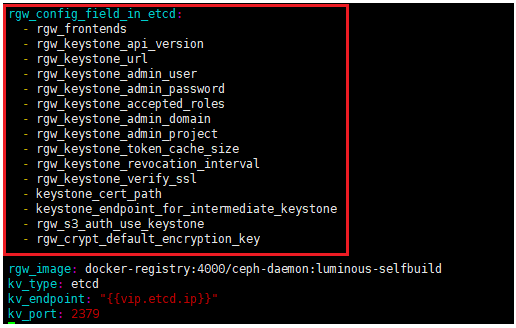
Sau đây là các bước cần thiết để cài đặt radosgw trên cụm hht với ansible.

Bước 1: Thực hiện truy cập tới node hht-lab04 và di chuyển tới thư mục chứa bộ cài ansible:

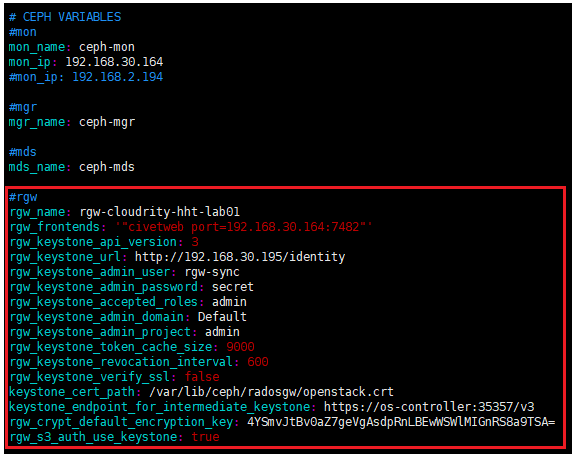
|  |
| --- |
| $ ssh [cloud@10.240.207.169](mailto:cloud@10.240.207.169)  $ cd /data/hungdh9-ansible-openstack-infra |

Bước 2: Thiết lập thông tin cấu hình cho radosgw. Ngoài những trường cấu hình đã được trình bày tại Chương 3, thì người dùng quản trị có thể tham khảo thêm các trường cấu hình cho Ceph rados gateway tại đây: <https://docs.ceph.com/docs/master/radosgw/config-ref/>

* Thêm tên trường cấu hình cho radosgw tới biến *rgw\_config\_field\_in\_etcd* trong file */data/hungdh9-ansible-openstack-infra/group\_vars/ceph\_rgw.yml*:

****

* Thiết lập giá trị cho các trường cấu hình của radosgw tương ứng trên từng node. Các giá trị được thiết lập trong file */data/hungdh9-ansible-openstack-infra/host\_vars/cloudrity-hht-lab0{X}.yml* (X là {1,2,3} tương ứng với từng node):

****

* Thêm lệnh cấu hình cho radosgw tại file */data/hungdh9-ansible-openstack-infra/roles/ceph-rgw/tasks/config\_in\_etcd.yml*:



Bước 3: Cài đặt radosgw trên 3 node. Câu lệnh phía dưới thực hiện tại thư mục */data/hungdh9-ansible-openstack-infra*.

|  |
| --- |
| $ ansible-playbook ceph-rgw.yml |

Bước 4: Cập nhật cấu hình mới cho rados gateway. Thực hiện lại bước 2 để thêm, sửa xóa các trường cấu hình cho rados gateway. Sau đó chạy câu lệnh phía dưới tại thư mục */data/hungdh9-ansible-openstack-infra* để cập nhật cấu hình:

|  |
| --- |
| $ ansible-playbook ceph-rgw-update-config.yml |

## Một số thao tác vận hành giành cho quản trị viên

Trong phần này sẽ cung cấp hướng dẫn một số thao tác thường dùng cho quản trị viên. Hướng dẫn đầy đủ cho người dùng quản trị có thể xem thêm tại đây: <https://docs.ceph.com/docs/master/radosgw/admin/>

### Quản lý người dùng

Chi tiết xem tại đây: [https://docs.ceph.com/docs/master/radosgw/admin/#user-management](https://docs.ceph.com/docs/master/radosgw/admin/%23user-management)

Các câu lệnh trong phần hướng dẫn này đều được thực hiện tại container chạy ceph-mon instance của cụm ceph. Đứng trên node 1, 2, 3 của cụm HHT Lab và thực hiện lệnh sau để truy cập vào ceph-mon container:

|  |
| --- |
| $ sudo docker exec -ti ceph-mon bash |

**Tạo tài khoản cho người dùng**

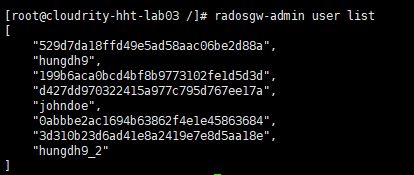
Hướng dẫn tạo tài khoản cho người dùng đã được trình bày phía trên, tại mục 4.2.1.

**Xem thông tin người dùng**

Trước khi xem thông tin từng người dùng cụ thể, sử dụng câu lệnh sau để liệt kê ra những người dùng đang được quản lý bởi rados gateway, kết quả trả về sẽ là danh sách UID của người dùng:

|  |
| --- |
| $ radosgw-admin user list |

Kết quả trả về:



Sử dụng câu lệnh sau để hiện thị thông tin một người dùng nhất định:

|  |
| --- |
| $ radosgw-admin user info --uid=<user\_id> |

Trong đó:

* user\_id: ID được khai báo tại trường --uid cho tài khoản tại bước tạo user.

Kết quả trả về như sau:



### Quản lý Quota

Chi tiết xem tại đây: [https://docs.ceph.com/docs/master/radosgw/admin/#quota-management](https://docs.ceph.com/docs/master/radosgw/admin/%23quota-management)

### Quản lý ACL cho container/bucket

Nội dung phần này chỉ áp dụng cho những người dùng thuộc môi trường Openstack cloud. Mặc định mỗi project trên openstack sẽ tương ứng với một account của rados gateway. Hay nói cách khác tất cả người dùng openstack thuộc cùng project sẽ sử dụng chung một account trên rados gateway và có quyền sử dụng, truy cập tới tài nguyên của account đó.



Swift interface hỗ trợ việc cấu hình ACL để cấp quyền truy cập tới một tài nguyên nhất định của project cho người dùng thuộc project khác.



Các bước cấu hình cấp quyền sử dụng tài nguyên tới một người dùng như sau:

* Truy cập vào docker container đang chạy openstack client:

|  |
| --- |
| $ docker exec -ti openstack-client bash |

* Sử dụng câu lệnh với cú pháp sau:

|  |
| --- |
| $ swift post <container\_name> --read-acl "<project\_id>:<user\_id>[,<project\_id>:<user\_id>]" --write-acl "<project\_id>:<user\_id>[,<project\_id>:<user\_id>]" |

Ví dụ:

$ swift post container\_2 --read-acl "d427dd970322415a977c795d767ee17a:48e771682a2b41b28f79b496f5c43129" --write-acl "d427dd970322415a977c795d767ee17a:48e771682a2b41b28f79b496f5c43129"

**Thiết lập quyền truy cập tới container/bucket cho một người dùng duy nhất**

Như đã nói ở trên, tất cả người dùng trong cùng openstack project sẽ được sử dụng chung tài nguyên object storage. Vì vậy việc thiết lập truy cập tới tài nguyên cho một dùng duy nhất là không thể.

Nhưng với cách thức cấp quyền truy cập như trên, chúng ta vẫn có thể thực hiện được việc này như sau:

* Tài nguyên object storage(container/bucket) thuộc về một project không chứa người dùng thông thường. Ví dụ: project admin, project service, hay có thể tạo ra một project riêng dùng cho việc cấp phát private container/bucket.
* Admin sẽ cấp quyền truy cập tới container/bucket cho một người hay nhiều người dùng. Như vậy vấn đề tài nguyên bị chia sẻ với các người dùng trong cùng project đã được giải quyết. Giờ đây container/bucket chỉ được truy cập tới bởi những người dùng được admin cấp quyền sử dụng.

