협업 연구자원 제공을 위한
OpenStack Swift 서비스 설치 가이드

일자: 2015년 11월 9일

부서: 첨단연구망센터 첨단연구망응용지원실

제출자: 홍원택, 권윤주

KiSTi www.kisti.re.kr 한국과학기술정보연구원 Korea Institute of Science and Technology Information

목차

1. 서론

본 문서는 최근 공개 소프트웨어 클라우드 운영체제로 활발히 이용되고 있는 OpenStack의 주요 프로젝트 중에서 Swift 서비스에 대해 소개하고, 설치 매뉴얼을 제공한다.
Swift 서비스는 OpenStack의 다른 서비스들과 달리 독립적으로 설치되어 운영될 수 있고, Keystone 서비스 정도만이 인증을 위해 선택적으로 연동될 필요가 있다. 본 문서에서는 Swift 서비스의 기본 기능을 확인하기 위한 설치를 중심으로 기술되고, 인증 서비스를 위해 Keystone 서비스의 설치 및 기본 환경 구성을 포함한다. 2014년 10월에 발표된 Juno 버전을 기준 버전으로 하였고, 사용된 리눅스 OS 버전은 Ubuntu 14.04이다.

2. Swift 서비스 및 컴포넌트 소개

2.1 주요 특징

OpenStack을 구성하는 여러 세부 프로젝트들 중에서 Nova와 Swift 프로젝트는 OpenStack의 중심을 이루는 프로젝트로서 최초 Austin 배포판에서부터 제공되어 오고 있다. 그 중에서 Swift 프로젝트는 Amazon의 Simple Storage Service(S3) 서비스와 유사한 오브젝트 스토리지를 제공하기 위한 수단으로 활용될 수 있다. Swift 서비스의 주요 특징들은 다음과 같이 분류할 수 있다.

- Highly available

Swift 서비스의 가장 중요한 특징으로, 복수개의 리플리카를 보유함으로써 H/W 오류 등의 문제로부터 데이터를 안전하게 유지할 수 있다. 최소 3개 이상의 리플리카를 기본으로 한다.

- Scalable

최소 2개부터 수천 노드까지 스토리지 노드를 확장할 수 있고, 이러한 특징은 “sing points of failure” 문제를 근본적으로 해결한다.

- Distributed

Region, Zone, Partitions 등의 개념을 통해 리플리카의 배치를 효율적으로 함으로써 Swift의 분산 메커니즘을 충실히 구현한다. 또한 “unique-as-possible” 알고리즘에 기반하여 클러스터 내에서 가장 덜 이용된 위치에 파티션을 배치시킴으로써 데이터를 균등하게 저장한다.

- Eventually consistent

Swift는 Block storage, Filesystems에 적합한 “strongly consistency”를 제공하지는 않는다. “Strongly consistency”는 Databases, 실시간처리 데이터 등에 요구되는 특성으로 scalability를 제한하고, H/W 오류 등의 문제 발생 시 가용성을 감소시킨다.
반면, Swift는 백업 파일, 로그 파일, 비정형 데이터와 같은 부류의 데이터에 대해서 “strongly consistency”가 아닌 “eventually consistency”를 제공하면서, “highly distributed” 인프라를 기반으로 “massive scalability”, “high availability”을 제공한다.

- Metadata
Object는 Object에 대한 정보, 즉 metadata를 갖는다. Swift 서비스가 관련된 metadata를 유지하게 하고, 그 정보를 다른 어플리케이션에서 사용할 수 있게 함으로써 Swift 시스템을 확장가능하게 한다.

- REST API
Object들에 대해 RESTful HTTP API를 통하여 저장 및 삭제 가능하다. 모든 어플리케이션들은 RESTful API를 통해 소통 가능하다.

2.2 주요 컴포넌트
- Proxy server
Swift 서비스의 외부 인터페이스 역할을 담당하고, 클라이언트로부터 업로드/다운로드 등의 요구사항을 받아 처리하고 결과를 넘겨준다. Keystone 서비스와 같은 Authentication 기능을 포함한 컨트롤러 서버에 함께 설치될 수 있다. 반면, 보다 큰 규모의 설치를 위해서 여러 대의 서버에 Proxy 서버들과 Load balancer들을 독립적으로 설치하여 운영할 수 있다.

- Account/Container/Object server
Account 서버는 각각의 account에 대해 관련된 metadata 요구를 처리하고, container들에 대한 리스트를 유지한다. Container 서버는 container에 대해 관련된 metadata 요구를 처리하고, object들에 대한 리스트를 유지한다. 이러한 정보는 SQLite database의 형태로 저장된다. Object 서버는 object들의 저장, 변경, 삭제 등에 직접 관여하고, 리플리카들의 무결성 검사를 위한 Auditor 서비스와도 밀접한 관련이 있다. 기본적으로 Swift 서비스는 5GB 이하로 object 크기를 제한한다. 추가적으로 object의 metadata는 파일의 xattrs(extended attributes)에 저장된다. 이것은 데이터와 메타데이터가 함께 저장됨을 의미한다.

- Rings
{account, container, object} 서버는 각각의 ring을 유지하고 있고, 이러한 ring은 consistent hash ring 데이터 구조를 따른다. 이러한 ring 구조는 다수의 리플리카, Partition, 디스크 weight에 따른 각 리플리카의 partition 배치 및 처리와 관련하여 정보를 제공한다.

- Auditor
스토리지 노드에서 백그라운드로 실행되고, 저장된 데이터가 “bit-rot” 또는 파일 시스템 오류 등의 문제로 영향을 받는지를 확인하기 위해 디스크를 스캔한다. 각 서버 프로세스를 지원하기 위해 구동되는 {account, container, object} auditor 등이 있다. 에러가 발견되면, auditor는 오류가 있는 오브젝트를 “quarantine” 영역으로 이동시킨다.

- Replicator
{account, container, object} replicator 프로세스 형태로 모든 스토리지 노드들의 백그라운드에서 실행되고, 다른 노드들에 있는 복제본에 대해 로컬본을 비교한다. 만약, 다른 노드들 중의 하나가 오래된 버전의 복제본을 갖고 있으면, replicator는 로컬본의 복제본을 해당 노

드로 보낸다. replicator는 로컬본을 다른 노드들로 내보낸다. 만약 로컬본이 오래된 경우, 외부의 복제본을 가져오지 않는다.
replicator는 object와 container의 삭제를 처리한다. object 삭제는 해당 object에 대한 zero-byte의 tombstone 파일을 생성함으로써 시작되고, 다른 노드들로 복제되고, 해당 object는 삭제된다. container 삭제는 비어 있는 container에 대해서 발생한다.

- Account Reaper
account reaper는 “deleted”로 표시된 account를 발견하고, 관련된 objects와 containers를 제거하기 시작한다. account reaper는 delay value를 갖고 있고, 데이터를 지우기 시작하기 전에 기다린다. 이러한 절차는 잘못된 삭제를 방지하기 위함이다.

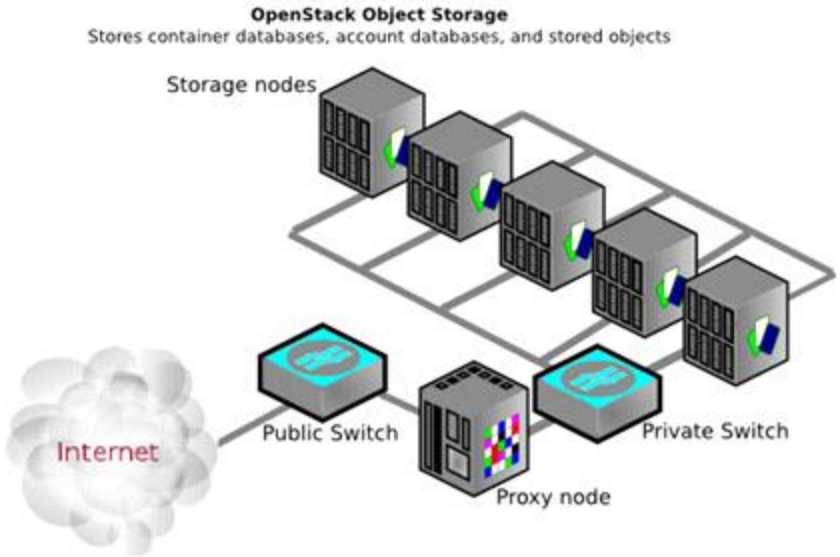

그림 2 Swift 서비스 구성

3. Swift 서비스 설치 및 설정

3.1 기본 환경 구성

3.1.1 Networking

Swift 서비스를 구성하기 위해서는 아래 그림에서 Controller 노드, Object Storage 노드들에 대한 네트워크 설정을 수행한다. 본 문서에서는 Management 네트워크, Storage 네트워크를 같은 망으로 구성한다.

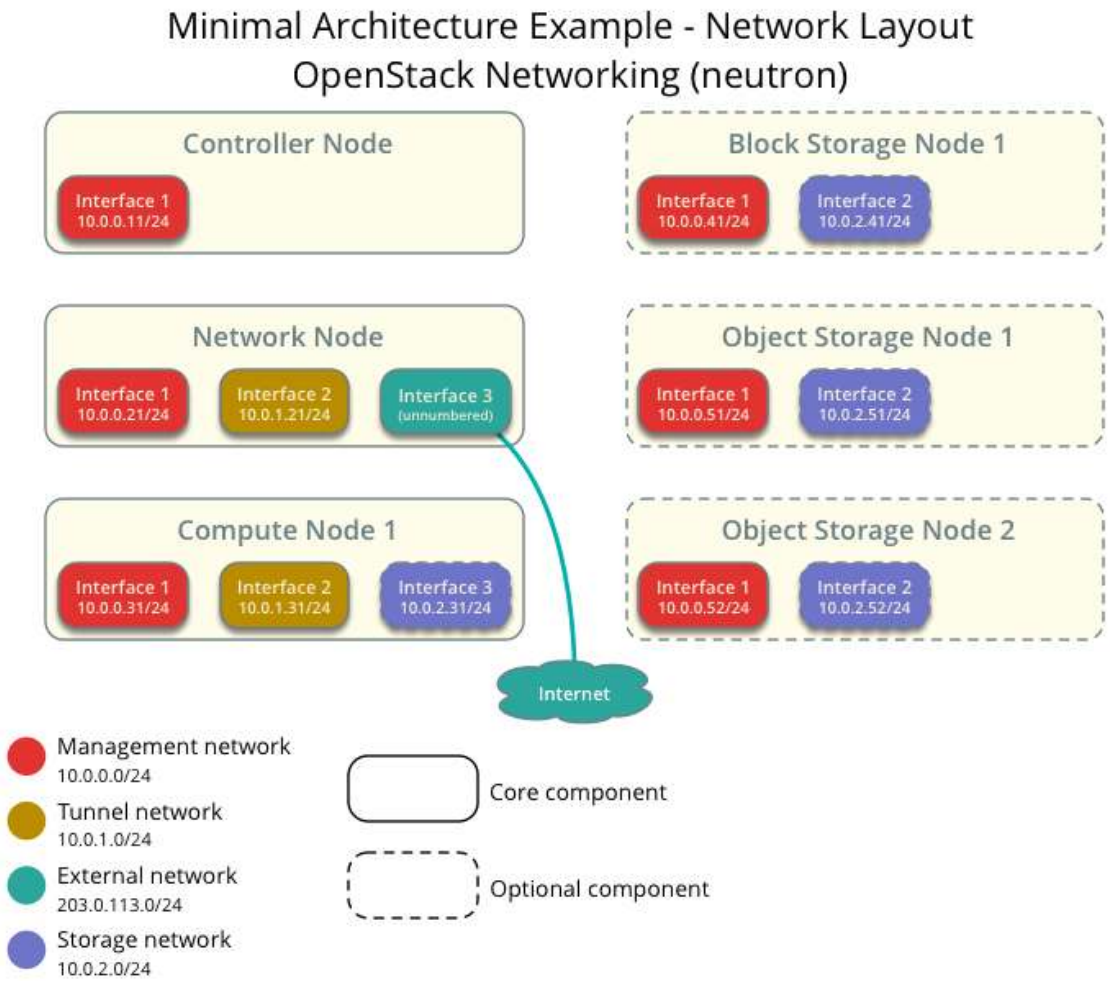

그림 3 네트워크 레이아웃

3.1.2 Network Time Protocol (NTP)

. Controller node

- NTP 설치

 # apt-get install ntp

- NTP 설정

1. /etc/ntp.conf 편집

```
# Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
# on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
# more information.
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org

# Use Ubuntu's ntp server as a fallback.
server ntp.ubuntu.com

# Access control configuration; see /usr/share/doc/ntp-doc/html/accopt.html for
# details.  The web page <http://support.ntp.org/bin/view/Support/AccessRestrictions>
# might also be helpful.
#
# Note that "restrict" applies to both servers and clients, so a configuration
# that might be intended to block requests from certain clients could also end
# up blocking replies from your own upstream servers.

# By default, exchange time with everybody, but don't allow configuration.
#restrict -4 default kod notrap nomodify nopeer noquery
#restrict -6 default kod notrap nomodify nopeer noquery
restrict -4 default kod notrap nomodify
restrict -6 default kod notrap nomodify
```

2. Restart the NTP service

 # service ntp restart

. Other nodes

- NTP 설치

 # apt-get install ntp

- NTP 설정

1. /etc/ntp.conf 편집

```
server controller iburst
```

2. Restart the NTP service

 # service ntp restart

. Verify operation

Controller node에서

 # ntpq -c peers

```
root@kisti84:/etc/apt/sources.list.d# ntpq -c peers
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+106.247.248.106 211.115.194.21   3 u  549 1024  377    4.931    6.021   2.137
+dadns.cdnetwork 131.107.13.100   2 u  943 1024  377    3.786   -0.300   0.664
*send.mx.cdnetwo 133.100.8.2      2 u  897 1024  377    3.762    0.245   6.588
 golem.canonical .INIT.          16 u    - 1024    0    0.000    0.000   0.000
```

 # ntpq -c assoc

```
root@kisti84:/etc/apt/sources.list.d# ntpq -c assoc

ind assid status  conf reach auth condition  last_event cnt
===========================================================
  1 32500  9424   yes   yes  none candidate   reachable  2
  2 32501  941a   yes   yes  none candidate    sys_peer  1
  3 32502  961a   yes   yes  none  sys.peer    sys_peer  1
  4 32503  8011   yes    no  none    reject    mobilize  1
```

All other nodes에서

 # ntpq -c peers

```
root@object1:/etc# ntpq -c peers
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*kisti84         211.233.84.186   3 u  746 1024  377    0.176   -0.937   0.486
```

 # ntpq -c assoc

```
root@object1:/etc# ntpq -c assoc

ind assid status  conf reach auth condition  last_event cnt
===========================================================
  1 13064  963a   yes   yes  none  sys.peer    sys_peer  3
```

3.1.3 OpenStack packages

. To enable the OpenStack repository

```
# apt-get install ubuntu-cloud-keyring
# echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu" \
  "trusty-updates/juno main" > /etc/apt/sources.list.d/cloudarchive-juno.list
```

. To finalize installation

```
# apt-get update && apt-get dist-upgrade
```

3.1.4 Database

Keystone이 설치되는 서버에 설치

. To install and configure the database server

- 패키지 설치

 # apt-get install mariadb-server python-mysqldb

- database root 계정에 대한 암호 설정

- /etc/mysql/my.cnf file 편집

```
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 0.0.0.0

default-storage-engine = innodb
innodb_file_per_table
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

. To finalize installation

 # service mysql restart

 # mysql_secure_installation

3.1.5 Messaging server

컨트롤러 노드 상에서 구동

. To install the RabbitMQ message broker service
 # apt-get install rabbitmq-server

. To configure the message broker service
세부적인 설정은 필요에 따라 설정하고, 서비스를 재시작한다.

 # service rabbitmq-server restart

참고로, RabbitMQ version을 확인하기 위해 “rabbitmqctl status”를 실행하면 아래와 같은 결과를 출력한다.

```
root@kisti84:/etc/rabbitmq# rabbitmqctl status | grep rabbit
Status of node rabbit@kisti84 ...
 {running_applications,[{rabbit,"RabbitMQ","3.2.4"},
```

3.2 Identity service 설치

OpenStack Identity 서비스는 다음의 두가지 중요한 기능을 수행한다.
- 사용자들의 권한 및 인증에 관여한다.
- API endpoints를 포함한 이용 가능한 서비스들의 목록을 제공한다.

3.2.1 Install and configure

. To configure prerequisites
1. 데이터베이스 생성
a. root user로 데이터베이스 서버 접근
```
$ mysql -u root -p
```

b. keystone DB 생성
```
CREATE DATABASE keystone;
```

c. keystone DB에 적절한 접근 허가
```
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' \
  IDENTIFIED BY 'KEYSTONE_DBPASS';
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
  IDENTIFIED BY 'KEYSTONE_DBPASS';
```

2. admin token으로 사용될 랜덤 값 생성
```
# openssl rand -hex 10
```

. To install and configure the components
1. 패키지 설치
```
# apt-get install keystone python-keystoneclient
```

2. /etc/keystone/keystone.conf 편집
```
[DEFAULT]
...
admin_token = ADMIN_TOKEN

[database]
...
connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone

[token]
...
provider = keystone.token.providers.uuid.Provider
driver = keystone.token.persistence.backends.sql.Token
```

[revoke]
...
driver = keystone.contrib.revoke.backends.sql.Revoke

[DEFAULT]
...
verbose = True

3. Identity service database 활성화
 # su -s /bin/sh -c "keystone-manage db_sync" keystone

. To finalize installation
1. Identity service 재시작
 # service keystone restart

2. SQLite database file 제거
 # rm -f /var/lib/keystone/keystone.db

3. expired tokens를 주기적으로 제거
*# (crontab -l -u keystone 2>&1 | grep -q token_flush) || *
*echo '@hourly /usr/bin/keystone-manage token_flush >/var/log/keystone/keystone-tokenflush.log 2>&1' *
>> /var/spool/cron/crontabs/keystone

3.2.2 Create tenants, users, and roles

. To configure prerequisites

1. admin token 설정

$ export OS_SERVICE_TOKEN=ADMIN_TOKEN

2. endpoint 설정

$ export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0

. To create tenants, users, and roles

1. admin tenant, user, and role 생성

a. admin tenant 생성

```
$ keystone tenant-create --name admin --description "Admin Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |           Admin Tenant           |
|   enabled   |               True               |
|      id     | 6f4c1e4cbfef4d5a8a1345882fbca110 |
|     name    |              admin               |
+-------------+----------------------------------+
```

b. admin user 생성

```
$ keystone user-create --name admin --pass ADMIN_PASS --email EMAIL_ADDRESS
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |        admin@example.com         |
| enabled  |               True               |
|    id    | ea8c352d253443118041c9c8b8416040 |
|   name   |              admin               |
| username |              admin               |
+----------+----------------------------------+
```

c. admin role 생성

```
$ keystone role-create --name admin
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|    id    | bff3a6083b714fa29c9344bf8930d199 |
|   name   |              admin               |
+----------+----------------------------------+
```

d. admin role을 admin tenant와 user에 더함

$ keystone user-role-add --user admin --tenant admin --role admin

2. demo tenant, user 생성

a. demo tenant 생성

```
$ keystone tenant-create --name demo --description "Demo Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |           Demo Tenant            |
|   enabled   |               True               |
|      id     | 4aa51bb942be4dd0ac0555d7591f80a6 |
|     name    |               demo               |
+-------------+----------------------------------+
```

b. demo tenant 하에 demo user 생성

```
$ keystone user-create --name demo --tenant demo --pass DEMO_PASS --email EMAIL_ADDRESS
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |         demo@example.com         |
| enabled  |               True               |
|    id    | 7004dfa0dda84d63aef81cf7f100af01 |
|   name   |               demo               |
| tenantId | 4aa51bb942be4dd0ac0555d7591f80a6 |
| username |               demo               |
+----------+----------------------------------+
```

3. 다른 서비스들과 interaction을 위해 service tenant 생성

a. service tenant 생성

```
$ keystone tenant-create --name service --description "Service Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |          Service Tenant          |
|   enabled   |               True               |
|      id     | 6b69202e1bf846a4ae50d65bc4789122 |
|     name    |             service              |
+-------------+----------------------------------+
```

3.2.3 Create the service entity and API endpoint

. To configure prerequisites

OS_SERVICE_TOKEN와 OS_SERVICE_ENDPOINT 환경변수 설정

. To create the service entity and API endpoints

1. Identity service에 대한 서비스 엔터티 생성

```
$ keystone service-create --name keystone --type identity \
  --description "OpenStack Identity"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |        OpenStack Identity        |
|   enabled   |               True               |
|      id     | 15c11a23667e427e91bc31335b45f4bd |
|     name    |             keystone             |
|     type    |             identity             |
+-------------+----------------------------------+
```

2. Identity service API endpoints 생성

```
$ keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ identity / {print $2}') \
  --publicurl http://controller:5000/v2.0 \
  --internalurl http://controller:5000/v2.0 \
  --adminurl http://controller:35357/v2.0 \
  --region regionOne
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
|   adminurl  |   http://controller:35357/v2.0   |
|      id     | 11f9c625a3b94a3f8e66bf4e5de2679f |
| internalurl |   http://controller:5000/v2.0    |
|  publicurl  |   http://controller:5000/v2.0    |
|    region   |           regionOne              |
|  service_id | 15c11a23667e427e91bc31335b45f4bd |
+-------------+----------------------------------+
```

3.2.4 Verify operation

1. 임시 OS_SERVICE_TOKEN, OS_SERVICE_ENDPOINT 환경변수 unset

$ unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT

2. admin tenant, user로서 authentication token 요청

```
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS \
  --os-auth-url http://controller:35357/v2.0 token-get
+-----------+----------------------------------+
|  Property |              Value               |
+-----------+----------------------------------+
|  expires  |       2014-10-10T12:50:12Z       |
|     id    | 8963eb5ccd864769a894ec316ef8f7d4 |
| tenant_id | 6f4c1e4cbfef4d5a8a1345882fbca110 |
|  user_id  | ea8c352d253443118041c9c8b8416040 |
+-----------+----------------------------------+
```

3. admin tenant, user로서 tenant-list 요청

```
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS \
  --os-auth-url http://controller:35357/v2.0 tenant-list
+----------------------------------+----------+---------+
|                id                |   name   | enabled |
+----------------------------------+----------+---------+
| 6f4c1e4cbfef4d5a8a1345882fbca110 |  admin   |   True  |
| 4aa51bb942be4dd0ac0555d7591f80a6 |   demo   |   True  |
| 6b69202e1bf846a4ae50d65bc4789122 | service  |   True  |
+----------------------------------+----------+---------+
```

4. admin tenant, user로서 user-list 요청

```
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS \
  --os-auth-url http://controller:35357/v2.0 user-list
+----------------------------------+---------+---------+---------------------+
|                id                |   name  | enabled |        email        |
+----------------------------------+---------+---------+---------------------+
| ea8c352d253443118041c9c8b8416040 |  admin  |   True  | admin@example.com   |
| 7004dfa0dda84d63aef81cf7f100af01 |   demo  |   True  | demo@example.com    |
+----------------------------------+---------+---------+---------------------+
```

5. admin tenant, user로서 role-list 요청

```
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS \
  --os-auth-url http://controller:35357/v2.0 role-list
+----------------------------------+----------+
|                id                |   name   |
+----------------------------------+----------+
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_ |
| bff3a6083b714fa29c9344bf8930d199 |  admin   |
+----------------------------------+----------+
```

6. demo tenant, user로서 authentication token 요청

```
$ keystone --os-tenant-name demo --os-username demo --os-password DEMO_PASS \
  --os-auth-url http://controller:35357/v2.0 token-get
+-----------+----------------------------------+
|  Property |              Value               |
+-----------+----------------------------------+
|  expires  |       2014-10-10T12:51:33Z       |
|     id    | 1b87ceae9e08411ba4a16e4dada04802 |
| tenant_id | 4aa51bb942be4dd0ac0555d7591f80a6 |
|  user_id  | 7004dfa0dda84d63aef81cf7f100af01 |
+-----------+----------------------------------+
```

7. demo tenant, user로서 “admin-only” CLI command 수행되지 않는지 검증.

```
$ keystone --os-tenant-name demo --os-username demo --os-password DEMO_PASS \
  --os-auth-url http://controller:35357/v2.0 user-list
You are not authorized to perform the requested action, admin_required. (HTTP 403)
```

3.2.5 Create OpenStack client environment scripts

. To create the scripts

1. admin-openrc.sh 편집

export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_AUTH_URL=http://controller:35357/v2.0

2. demo-openrc.sh 편집

export OS_TENANT_NAME=demo
export OS_USERNAME=demo
export OS_PASSWORD=DEMO_PASS
export OS_AUTH_URL=http://controller:5000/v2.0

. To load client environment scripts

admin tenant, user credentials, Identity 서비스 위치를 로딩하기 위해
$ source admin-openrc.sh

3.3 Object storage 서비스 설치

3.3.1 Install and configure the controller node

. To configure prerequisites

1. Identity service credentials 생성

a. swift user 생성

```
$ keystone user-create --name swift --pass SWIFT_PASS
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |                                  |
| enabled  |               True               |
|    id    | d535e5cbd2b74ac7bfb97db9cced3ed6 |
|   name   |              swift               |
| username |              swift               |
+----------+----------------------------------+
```

b. swift user에 admin role 더함

$ keystone user-role-add --user swift --tenant service --role admin

c. swift service entity 생성

```
$ keystone service-create --name swift --type object-store \
  --description "OpenStack Object Storage"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |     OpenStack Object Storage     |
|   enabled   |               True               |
|      id     | 75ef509da2c340499d454ae96a2c5c34 |
|     name    |              swift               |
|     type    |           object-store           |
+-------------+----------------------------------+
```

2. Object Storage service API endpoints 생성

```
$ keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ object-store / {print $2}') \
  --publicurl 'http://controller:8080/v1/AUTH_%(tenant_id)s' \
  --internalurl 'http://controller:8080/v1/AUTH_%(tenant_id)s' \
  --adminurl http://controller:8080 \
  --region regionOne
+-------------+-----------------------------------------------------+
|   Property  |                        Value                        |
+-------------+-----------------------------------------------------+
|   adminurl  |            http://controller:8080/                  |
|      id     |          af534fb8b7ff40a6acf725437c586ebe           |
| internalurl | http://controller:8080/v1/AUTH_%(tenant_id)s        |
|  publicurl  | http://controller:8080/v1/AUTH_%(tenant_id)s        |
|    region   |                    regionOne                        |
|  service_id |          75ef509da2c340499d454ae96a2c5c34           |
+-------------+-----------------------------------------------------+
```

. To install and configure the controller node components

1. 패키지 설치

*# apt-get install swift swift-proxy python-swiftclient python-keystoneclient *
python-keystonemiddleware memcached

2. /etc/swift 생성

3. proxy service configuration file 받기

*# curl -o /etc/swift/proxy-server.conf *
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/proxy-server.conf-sample

4. /etc/swift/proxy-server.conf 편집

[DEFAULT]
...
bind_port = 8080
user = swift
swift_dir = /etc/swift

[pipeline:main]
pipeline = authtoken cache healthcheck keystoneauth proxy-logging proxy-server
 # 위와 같이 매뉴얼에 되어 있으나, 아래와 같이 "tempauth"가 있어서는 안됨.

```
[pipeline:main]
pipeline = authtoken keystoneauth catch_errors gatekeeper healthcheck proxy-logg
ing cache container_sync bulk tempurl ratelimit container-quotas account-quotas
slo dlo proxy-logging proxy-server
#pipeline = authtoken keystoneauth catch_errors gatekeeper healthcheck proxy-log
ging cache container_sync bulk tempurl ratelimit tempauth container-quotas accou
nt-quotas slo dlo proxy-logging proxy-server
```

[app:proxy-server]
...
allow_account_management = true
account_autocreate = true
[filter:keystoneauth]
use = egg:swift#keystoneauth
...
*operator_roles = admin,_member_*

[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = swift

```
admin_password = SWIFT_PASS
delay_auth_decision = true

[filter:cache]
...
memcache_servers = 127.0.0.1:11211
```

3.3.2 Install and configure the storage nodes

. To configure prerequisites
1. 1st 스토리지 노드 IP 설정
a. Configure the management interface
IP address: 10.0.0.51
Network mask: 255.255.255.0 (or /24)
Default gateway: 10.0.0.1

b. Set the hostname of the node to object1

2. 2nd 스토리지 노드 IP 설정
a. Configure the management interface
IP address: 10.0.0.52
Network mask: 255.255.255.0 (or /24)
Default gateway: 10.0.0.1

b. Set the hostname of the node to object2

3. 양 스토리지 노드들 상의 “shared items” 설정
a. 컨트롤러 노드로부터 /etc/hosts file 내용 복사 및 아래 내용 추가.
 # object1
10.0.0.51 object1
 # object2
10.0.0.52 object2

모든 다른 노드들 상의 /etc/hosts 파일에 위 내용 추가

b. NTP 설치 및 설정

c. utility packages 설치
 # apt-get install xfsprogs rsync

d. XFS로 /dev/sdb1, /dev/sdbc1 파티션 포맷
 # mkfs.xfs /dev/sdb1
 # mkfs.xfs /dev/sdc1

e. 마운트 포인트 디렉토리 생성
 # mkdir -p /srv/node/sdb1
 # mkdir -p /srv/node/sdc1

f. /etc/fstab 파일 편집 및 다음 내용 추가

/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 2

/dev/sdc1 /srv/node/sdc1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 2

g. 마운트

 # mount /srv/node/sdb1

 # mount /srv/node/sdc1

4. /etc/rsyncd.conf 편집 및 다음 내용 추가

uid = swift

gid = swift

log file = /var/log/rsyncd.log

pid file = /var/run/rsyncd.pid

address = MANAGEMENT_INTERFACE_IP_ADDRESS

망 분리를 할 때, 위 ip를 다른 망 ip로 분리할 필요.

[account]

max connections = 2

path = /srv/node/

read only = false

lock file = /var/lock/account.lock

[container]

max connections = 2

path = /srv/node/

read only = false

lock file = /var/lock/container.lock

[object]

max connections = 2

path = /srv/node/

read only = false

lock file = /var/lock/object.lock

5. /etc/default/rsync 편집 및 활성화

RSYNC_ENABLE=true

6. rsync 서비스 시작

 # service rsync start

. To install and configure storage node components
아래의 절차를 각 스토리지 노드 상에서 수행

1. 패키지 설치
 # apt-get install swift swift-account swift-container swift-object

2. accounting, container, and object service configuration files 내려받기
*# curl -o /etc/swift/account-server.conf *
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/account-server.conf-sample
*# curl -o /etc/swift/container-server.conf *
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/container-server.conf-sample
*# curl -o /etc/swift/object-server.conf *
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/object-server.conf-sample

3. /etc/swift/account-server.conf 편집 및 다음 내용 수행
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6002
user = swift
swift_dir = /etc/swift
devices = /srv/node

[pipeline:main]
pipeline = healthcheck recon account-server

[filter:recon]
...
recon_cache_path = /var/cache/swift

4. /etc/swift/container-server.conf 편집 및 다음 내용 수행
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6001
user = swift
swift_dir = /etc/swift
devices = /srv/node

[pipeline:main]
pipeline = healthcheck recon container-server

[filter:recon]
...
recon_cache_path = /var/cache/swift

5. /etc/swift/object-server.conf 편집 및 다음 내용 수행
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6000
user = swift
swift_dir = /etc/swift
devices = /srv/node

[pipeline:main]
pipeline = healthcheck recon object-server

[filter:recon]
...
recon_cache_path = /var/cache/swift

6. 마운트 디렉토리의 적합한 권한 부여
 # chown -R swift:swift /srv/node

7. recon 디렉토리 생성 및 권한 부여
 # mkdir -p /var/cache/swift
 # chown -R swift:swift /var/cache/swift

3.3.3 Create initial rings

(가정) one region/zone, 2^10(1024) 최대 파티션, 3 replicas, 1 hour(hold down value)

. Account ring (컨트롤러 노드 상에서 아래 절차 수행)

1. /etc/swift 디렉토리로 변경

2. 기본 account.builder 파일 생성

 # swift-ring-builder account.builder create 10 3 1

3. 링에 각 스토리지 노드 추가

```
# swift-ring-builder account.builder \
  add r1z1-STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS:6002/DEVICE_NAME DEVICE_WEIGHT

# swift-ring-builder account.builder add r1z1-10.0.0.51:6002/sdb1 100
```

(반복)

4. 링 contents 검증

```
# swift-ring-builder account.builder
account.builder, build version 4
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 4 devices, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port  replication ip   replication port      name weight partitions balance meta
             0       1     1       10.0.0.51  6002       10.0.0.51               6002      sdb1 100.00        768    0.00
             1       1     1       10.0.0.51  6002       10.0.0.51               6002      sdc1 100.00        768    0.00
             2       1     1       10.0.0.52  6002       10.0.0.52               6002      sdb1 100.00        768    0.00
             3       1     1       10.0.0.52  6002       10.0.0.52               6002      sdc1 100.00        768    0.00
```

5. 링의 rebalance

 # swift-ring-builder account.builder rebalance

. Container ring (컨트롤러 노드 상에서 아래 절차 수행)

1. /etc/swift 디렉토리로 변경

2. 기본 container.builder 파일 생성

 # swift-ring-builder container.builder create 10 3 1

3. 링에 각 스토리지 노드 추가

```
# swift-ring-builder container.builder \
  add r1z1-STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS:6001/DEVICE_NAME DEVICE_WEIGHT

# swift-ring-builder container.builder add r1z1-10.0.0.51:6001/sdb1 100
```

(반복)

4. 링 contents 검증

```
# swift-ring-builder container.builder
container.builder, build version 4
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 4 devices, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port  replication ip    replication port      name weight partitions balance meta
             0       1     1       10.0.0.51  6001       10.0.0.51                6001      sdb1 100.00        768    0.00
             1       1     1       10.0.0.51  6001       10.0.0.51                6001      sdc1 100.00        768    0.00
             2       1     1       10.0.0.52  6001       10.0.0.52                6001      sdb1 100.00        768    0.00
             3       1     1       10.0.0.52  6001       10.0.0.52                6001      sdc1 100.00        768    0.00
```

5. 링의 rebalance

 # swift-ring-builder container.builder rebalance

. Object ring (컨트롤러 노드 상에서 아래 절차 수행)

1. /etc/swift 디렉토리로 변경

2. 기본 object.builder 파일 생성

 # swift-ring-builder object.builder create 10 3 1

3. 링에 각 스토리지 노드 추가

```
# swift-ring-builder object.builder \
  add r1z1-STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS:6000/DEVICE_NAME DEVICE_WEIGHT
```

```
# swift-ring-builder object.builder add r1z1-10.0.0.51:6000/sdb1 100
```

(반복)

4. 링 contents 검증

```
# swift-ring-builder object.builder
object.builder, build version 4
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 4 devices, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port  replication ip    replication port      name weight partitions balance meta
             0       1     1       10.0.0.51  6000       10.0.0.51                6000      sdb1 100.00        768    0.00
             1       1     1       10.0.0.51  6000       10.0.0.51                6000      sdc1 100.00        768    0.00
             2       1     1       10.0.0.52  6000       10.0.0.52                6000      sdb1 100.00        768    0.00
             3       1     1       10.0.0.52  6000       10.0.0.52                6000      sdc1 100.00        768    0.00
```

5. 링의 rebalance

 # swift-ring-builder object.builder rebalance

. Distribute ring configuration files

account.ring.gz, container.ring.gz, object.ring.gz 파일들을 각 스토리지 노드들의 /etc/swift 디렉토리로 복사

3.3.4 Finalize installation

. To configure hashes and default storage policy

1. /etc/swift/swift.conf 받기

*# curl -o /etc/swift/swift.conf *
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/swift.conf-sample

2. /etc/swift/swift.conf 편집 및 다음 내용 수행

[swift-hash]
...
swift_hash_path_suffix = HASH_PATH_PREFIX
swift_hash_path_prefix = HASH_PATH_SUFFIX

[storage-policy:0]
...
name = Policy-0
default = yes

3. swift.conf 파일을 각 스토리지 노드들의 /etc/swift 디렉토리로 복사

4. 모든 노드들에서, 적합한 권한 부여

 # chown -R swift:swift /etc/swift

5. 컨트롤러 노드 상에서, Object Storage proxy service 재시작(관련 모듈 포함)

 # service memcached restart
 # service swift-proxy restart

6. 스토리지 노드들 상에서, Object Storage 서비스 시작

 # swift-init all start

4. 기본 기능 테스트

위에서 설정된 Swift 서비스에 대해 컨트롤러 노드 상에서 다음 과정을 수행하여 기본 기능에 대한 검증을 수행한다.

- demo tenant credentials 소싱

$ source demo-openrc.sh

- 서비스 상태 보기

```
$ swift stat
Account: AUTH_11b9758b7049476d9b48f7a91ea11493
Containers: 0
   Objects: 0
     Bytes: 0
Content-Type: text/plain; charset=utf-8
X-Timestamp: 1381434243.83760
X-Trans-Id: txdcdd594565214fb4a2d33-0052570383
X-Put-Timestamp: 1381434243.83760
```

- 테스트 파일 업로드

```
juno@kisti84:~$ swift upload demo-container1 test.txt
test.txt
```

- containers 리스팅

$ swift list

demo-container1

```
juno@kisti84:~$ swift stat
      Account: AUTH_68962452c5dd4c34b96191530832c603
   Containers: 1
      Objects: 1
        Bytes: 19
Accept-Ranges: bytes
  X-Timestamp: 1439189732.46016
   X-Trans-Id: tx76bd1ed5b2084d7b82140-0055c84d78
 Content-Type: text/plain; charset=utf-8
```

- 테스트 파일 내려받기

```
juno@kisti84:~/temp$ swift download demo-container1 test.txt
test.txt [auth 0.217s, headers 0.403s, total 0.403s, 0.000 MB/s]
```

5. 결론 및 시사점

본 문서에서는 OpenStack에서 Object Storage 서비스를 제공하는 Swift 프로젝트의 주요 특징 및 컴포넌트들을 정리하고, Swift 서비스를 구동하기 위해 필요한 설치 과정 및 주요 파일들에 대한 설정을 기술하였다. OpenStack 기반의 클라우드 환경에서 대용량의 비정형 데이터를 효과적으로 서비스하기 위해 Swift 서비스는 필요하다. 또한, 본 문서에서는 구체적으로 다루어지지 않았지만, 실질적인 서비스 제공을 위해서는 Management 네트워크, Storage 네트워크, Replication 네트워크에 대한 분리 및 적절한 망 설계가 동반되어야 할 것으로 판단된다.

6. 참고문헌

[1] OpenStack project, http://docs.openstack.org/juno/install-guide/install/apt/content/index.html
[2] SwiftStack, https://swiftstack.com/openstack-swift/
[3] “Object Storage in OpenStack using Swift”, HP Education Student Workbook