OpenStack Swift with Galera on CentOS 6.5 Installation Manual

Created by 2015.01.27 OpenStack Korea Community Nalee Jang

본 문서는 CentOS 6.5에 MariaDB를 설치하고, 인증 서비스인 Keystone을 이용한 OpenStack Swift IceHouse 설치 문서입니다. 또한 CentOS 6.5 서버가 이미 설치되어있다고 가정하고 서버 한대에 모든 컴포넌트를 설치하고 테스트 할 수 있는 문서입니다.

Step 1: Installation NTP

싱글 노드 설치를 제외한 모든 멀티 노드 설치에서는 반드시 NTP를 이용하여 각각의 서버 시각을 동기화해야 합니다. 그렇지 않으면, 운영 시 데이터를 저장한다던가 삭제할 경우, 서버 시각 차이로 인해 문제가 발생합니다. 그럼, 지금부터 NTP를 설치해 보도록 하겠습니다.

1. Yum install을 이용해서 ntp를 설치합니다.

```
[root@nalee1 ~]# yum install ntp
Loaded plugins: fastestmirror, security
base
| 3.7 kB        00:00
base/primary_db
...
Setting up Install Process
Package ntp-4.2.6p5-1.el6.centos.x86_64 already installed and latest version
Nothing to do
```

2. ntp 설치가 완료되면 ntp.conf에 동기화할 컨트롤러 노드 IP를 추가합니다. 이때 기존에 설정되어 있는 server들은 모두 주석 처리합니다.

```
[root@nalee2 ~]# vi /etc/ntp.conf
#server 0.centos.pool.ntp.org iburst
#server 1.centos.pool.ntp.org iburst
#server 2.centos.pool.ntp.org iburst
#server 3.centos.pool.ntp.org iburst
server 127.127.1.0
```

*** 이때 ntp 설치 노드가 마스터이면 자기자신을 나타내는 127.127.1.0을 입력하고, 클러스터 노드라면 마스터 노드 IP를 입력합니다.

3. 환경설정이 끝나면 ntp 서비스를 시작하고 부팅 시 자동 시작될 수 있도록 설정합니다.

```
[root@nalee2 ~]# service ntpd start
Starting ntpd:                                             [  OK  ]
# chkconfig ntpd on
```

ㄴ

4. 동기화가 제대로 되었는지 다음과 같은 명령어로 확인합니다.

```
[root@nalee2 ~]# ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*LOCAL(0)        .LOCL.          5 l   43   64  377    0.000    0.000   0.004
```

이렇게 해서 NTP를 모두 설치해 보았습니다. NTP는 모든 노드에 동일하게 설치하며 마스터 노드와 클러스터 노드의 환경설정만 변경해 주면 됩니다. 마스터 노드는 오픈스택에서는 컨트롤러 노드라 불리며, 오픈스택 Swift만 설치된다고 가정했을 경우에는 프록시 노드에 해당합니다.
클러스터 노드는 오픈스택에서는 컴퓨트 노드에 해당하면 Swift에서는 스토리지 노드에 해당합니다.

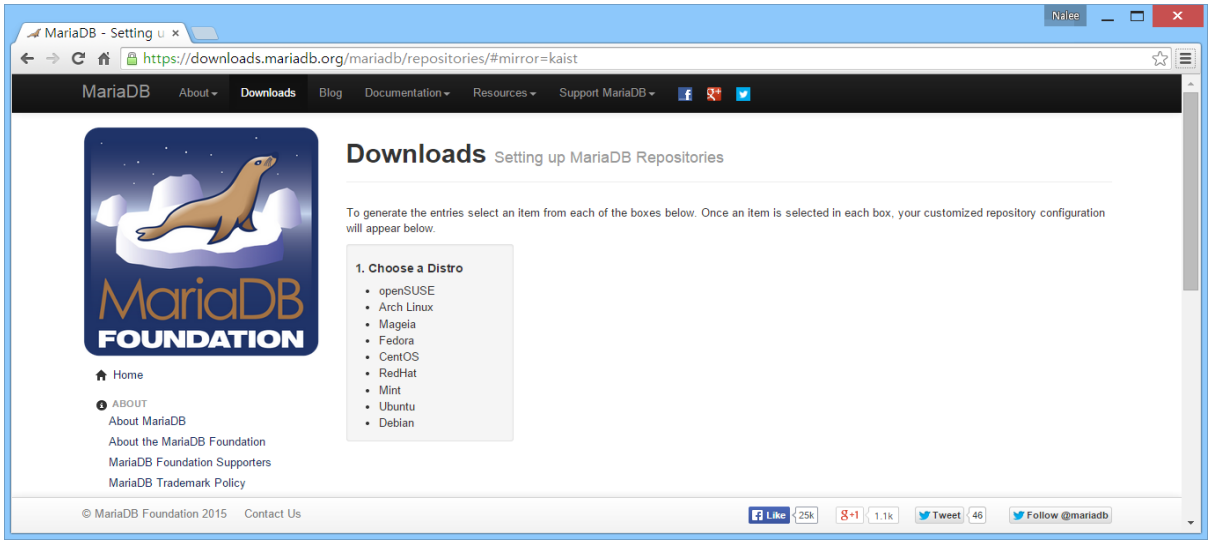
Step 2: Installation MariaDB

이번 단계에서는 MariaDB를 설치해 보도록 하겠습니다.

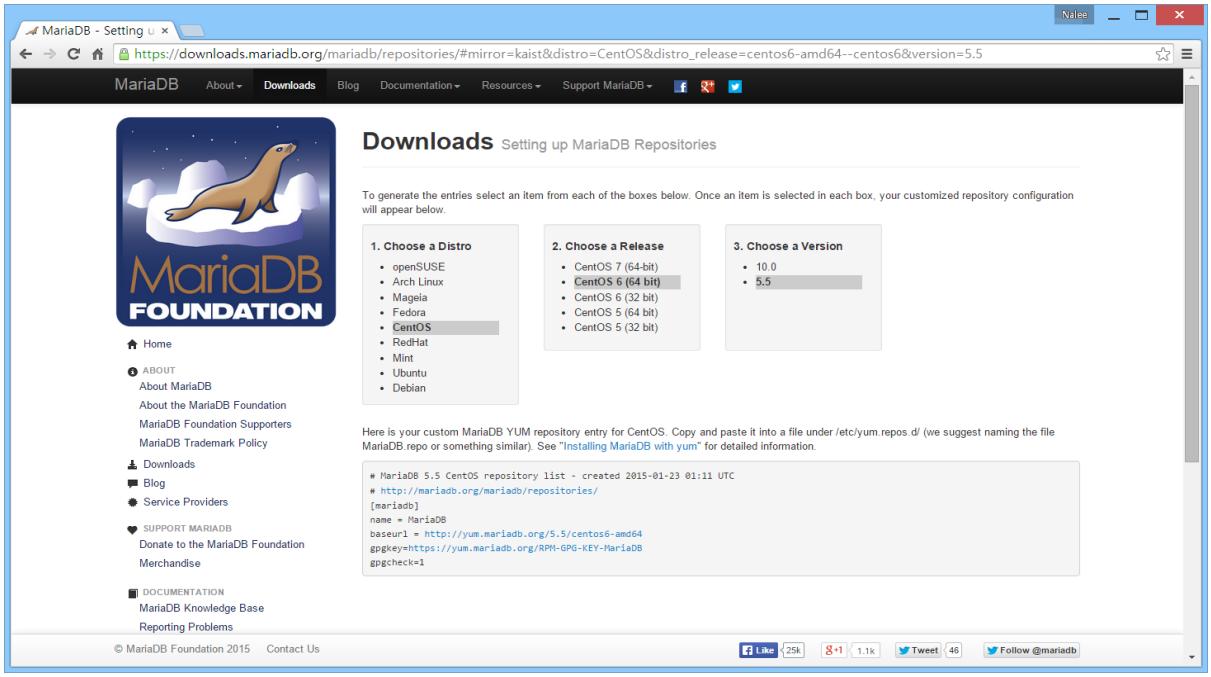
Add MariaDB Repositories

MariaDB를 설치하기 전에 반드시 설치하고자 하는 버전의 레파지토리를 설정해 주어야 합니다. 그럼, 지금부터 아래와 같은 방법으로 모든 노드에 레파지토리를 설정해 보겠습니다.

5. MariaDB 레파지토리를 설정해야 하기 위해 다음 사이트를 방문한다.
   URL: https://downloads.mariadb.org/mariadb/repositories/



6. MariaDB를 설치할 Host 노드의 OS 타입을 선택하고 OS의 버전을 선택하면 해당 레파지토리를 설정하는 방법이 나온다. 본 문서에서는 CentOS6.5를 사용할 것이므로 CentOS를 선택한 후 CentOS 6(64bit)를 선택하고 설치할 MariaDB 버전을 선택한다.



7. /etc/yum.repos.d/ 디렉터리 밑에 mariadb.repo 파일을 생성한 후 위 사이트에서 복사한 내용을 붙여 넣기 한 후 저장하고 빠져 나온다.

```
[root@nalee1 ~]# vi /etc/yum.repos.d/mariadb.repo
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/5.5/centos6-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

** 이때 CentOS가 32bit라면 MariaDB 레파지토리 사이트에서 CentOS 6 (32 bit)를 선택한 후 해당 내용을 붙여 넣기 해야 한다.

8. RedHat 기반의 리눅스에서는 Selinux를 이용한 보안정책을 사용하는데, MariaDB-Galera를 설치하고 서로 통신을 할 때는 이 Selinux 보안정책 때문에 서로 통신이 되질 않는다. 따라서, 아래와 같이 Selinux의 설정을 사용하지 않겠다고 변경해 주어야 한다.

```
[root@nalee1 ~]# cat /selinux/enforce
1
[root@nalee1 ~]# echo 0 > /selinux/enforce
[root@nalee1 ~]# cat /selinux/enforce
0
[root@nalee1 ~]# vi /etc/selinux/config
# This file controls the state of SELinux on the system.
```

```
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
# SELINUX=enforcing   -- 이 부분을 주석처리하고 아래와 같이 disabled로 변경한다.
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

Install MariaDB

레파지토리를 설정하고 SELINUX의 보안정책을 변경하면 이번에는 MariaDB-Server와 Client를 설치합니다.

9. 이제 모든 노드에 MariaDB-Galera와 Client를 설치한다.

```
[root@nalee1 ~]# yum install mariaDB-server mariaDB-client mariadb
```

10. /etc/my.cnf 파일을 열고 아래와 같이 수정합니다.

```
[root@nalee1 ~]# vi /etc/my.cnf
[mysqld]
...
bind-address = 10.0.100.32 # 설치할 노드의 IP를 입력합니다.
default-storage-engine = innodb
innodb_file_per_table
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

11. 이제 MariaDB-Galera 서비스를 아래와 같이 시작한다.

```
[root@nalee1 ~]# service mysql start
Starting MySQL..... SUCCESS!
```

12. 재부팅이 되어도 MariaDB가 자동으로 실행되길 원한다면 아래와 같이 chkconfig를 이용하여

runlevel을 on으로 설정한다.

```
[root@nalee1 ~]# chkconfig mysql on
```

Initial MariaDB Configuration

MariaDB를 설치했으면 이번에는 MariaDB에 접속을 하기 위한 환경설정을 해 보도록 하겠습니다.

13. MariaDB 설치가 완료되었다면 이제 mysql_secure_installation 명령어를 사용하여 root 패스워드를 설정한다.

```
[root@nalee1 ~]# mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user.  If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorisation.

You already have a root password set, so you can safely answer 'n'.

Set root password? [Y/n] y
New password: ***********
Re-enter new password: ***********
Password updated successfully!
Reloading privilege tables..
 ... Success!

By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
```

```
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] y
 ... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
 ... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

Step 3: Installation Openstack Package

데이터베이스 설치와 설정이 완료되면 Openstack를 설치하기 위한 패키지를 설치하게 됩니다. 본 문서에서는 CentOS 6.5를 사용하였기 때문에 IceHouse 버전으로 OpenStack를 설치합니다. 만일 Juno 버전의 오픈스택을 설치할 계획이라면 CentOS 7 이 설치된 노드를 이용해야 합니다.

14. Yum install을 이용해서 RDO 레파지토리를 사용하기 위한 yum-plugin-priorities를 설치한다.

```
[root@nalee1 ~]# yum install yum-plugin-priorities
```

15. 이번에는 다음과 같은 명령어로 rdo-release-icehouse를 내려 받고 설치한다.

```
[root@nalee1 ~]# yum install http://repos.fedorapeople.org/repos/openstack/openstack-icehouse/rdo-release-icehouse-4.noarch.rpm
```

*** http://repos.fedorapeople.org/repos/openstack/ 사이트에 방문하면 오픈스택 가장 최신 릴리즈 버전 3개가 존재한다. 설치하고자 하는 릴리즈 버전을 선택한 후 들어가면 또 다시 해당 버전의 rpm 파일들이 보인다. 이때 가장 최신 버전의 rpm 파일을 설치하면 된다. 그러나, CentOS6 버전에서 Juno를 설치하게 되면 정상적으로 기능이 동작하지 않으므로 IceHouse를 설치한다.

16. EPEL 패키지는 패키지 서명 및 저장소 정보에 대한 GPG(GNU Privacy Guard) 키가 포함되어 있다. EPEL 패키지는 레드햇, CentOS에 설치되어야 하며 설치 방법은 다음과 같다.

```
[root@nalee1 ~]# yum install http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
```

17. 이번에는 Openstack-utils를 설치한다. Openstack-utlis에는 설치 및 시스템 구성을 쉽게 할 수 있는 유틸리티 프로그램이 포함되어 있다.

```
[root@nalee1 ~]# yum install openstack-utils
```

18. 패키지 설치가 모두 완료되면 시스템을 업그레이드한다.

```
[root@nalee1 ~]# yum upgrade
```

Step 4: Installation Keystone

이번에는 Keystone을 설치해 보도록 하겠습니다.

19. yum install을 이용해서 openstack-keystone과 python-keystoneclient를 설치한다.

```
[root@nalee1 ~]# yum install openstack-keystone python-keystoneclient
```

20. Openstack-config 명령을 이용해서 다음과 같이 데이터베이스 정보를 변경한다.

```
[root@nalee1 ~]# openstack-config --set /etc/keystone/keystone.conf ₩
    database connection mysql://keystone:keystonedbpass@10.0.100.32/keystone
```

21. 이번에는 mysql에 접속해서 keystone 데이터베이스와 사용자 계정 및 패스워드 그리고 접속 권한을 함께 생성한다.

```
[root@nalee1 ~]# mysql -u root -p
mysql> CREATE DATABASE keystone;
Query OK, 1 row affected (0.01 sec)

mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' IDENTIFIED BY
'keystonedbpass';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' IDENTIFIED BY
'keystonedbpass';
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
```

22. Keystone 계정 생성이 완료되면 keystone에서 필요한 테이블을 만들어야 합니다. 테이블 생성은 keystone-manage db_sync 명령을 이용한다.

```
[root@nalee1 ~]# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

23. keystone의 admin token을 설정하기 위해 openssl rand 명령어를 이용해서 생성된 임의의 문자열을 ADMIN_TOKEN에 저장한다. 그리고 ADMIN_TOKEN에 저장된 문자열은 openstack_config 명령어로 /etc/keystone/keystone.conf 파일의 admin_token에 설정한다.

```
[root@nalee1 ~]# ADMIN_TOKEN=$(openssl rand -hex 10)
[root@nalee1 ~]# echo $ADMIN_TOKEN
dc387ae9520f5b69f643
[root@nalee1 ~]# openstack-config --set /etc/keystone/keystone.conf DEFAULT ₩
    admin_token $ADMIN_TOKEN
```

24. Keystone은 기본적으로 PKI 토큰을 사용합니다. 그러므로 keystone-manage를 이용해서 PKI

Key와 인증서를 생성하고, 생성된 /etc/keystone/ssl은 아무나 접근할 수 없도록 액세스를 제한한다.

```
[root@nalee1 ~]# keystone-manage pki_setup --keystone-user keystone --keystone-group keystone
Generating RSA private key, 2048 bit long modulus
.....................................................................................+++
.....................................................+++
e is 65537 (0x10001)
Generating RSA private key, 2048 bit long modulus
..............+++
.............+++
e is 65537 (0x10001)
Using configuration from /etc/keystone/ssl/certs/openssl.conf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName               :PRINTABLE:'US'
stateOrProvinceName     :ASN.1 12:'Unset'
localityName              :ASN.1 12:'Unset'
organizationName        :ASN.1 12:'Unset'
commonName                :ASN.1 12:'www.example.com'
Certificate is to be certified until Jun 12 17:10:09 2024 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated
[root@nalee1 ~]# chown -R keystone:keystone /etc/keystone/ssl
[root@nalee1 ~]# chmod -R o-rwx /etc/keystone/ssl
```

25. SSL PKI Key 설정이 완료되면 keystone 서비스를 재시작하고, 부팅 시 서비스가 자동 실행될 수 있도록 설정한다.

```
[root@nalee1 ~]# service openstack-keystone start
Starting keystone:                                         [  OK  ]
[root@nalee1 ~]# chkconfig openstack-keystone on
```

keystone 사용자, 테넌트, 롤 그리고 endpoint 생성

keystone 설치가 완료되면 keystone를 접근하기 위한 사용자, 테넌트, 롤과 서비스, endpoint를 생성해야 합니다.

1. 우선 OS_SERVICE_TOKEN과 OS_SERVICE_ENDPOINT를 export한다.

```
[root@nalee1 ~]# export OS_SERVICE_TOKEN=$ADMIN_TOKEN
[root@nalee1 ~]# export OS_SERVICE_ENDPOINT=http://10.10.15.11:35357/v2.0
```

2. keystone user-create 명령어를 이용해서 다음과 같이 admin 사용자 계정, 패스워드, 이메일을 입력하고 admin 계정을 생성한다.

```
[root@nalee1 ~]# keystone user-create --name=admin --pass=adminpass ₩
--email=admin@email.com
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |         admin@email.com          |
| enabled  |               True               |
|    id    | 7bc1bda9d2444b9c92554ff9e0b71e22 |
|   name   |              admin               |
| username |              admin               |
+----------+----------------------------------+
```

3. admin 롤을 생성한다.

```
[root@nalee1 ~]# keystone role-create --name=admin
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|    id    | 02ff77d31ec841b3914db4c3daca4f1c |
|   name   |              admin               |
+----------+----------------------------------+
```

4. 이번에는 admin 테넌트를 생성한다.

```
[root@nalee1 ~]# keystone tenant-create --name=admin --description="Admin Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |           Admin Tenant           |
|   enabled   |               True               |
|      id     | dbd8df87e5c34536844d915b555061b6 |
|     name    |              admin               |
+-------------+----------------------------------+
```

5. Admin 테넌트와 member 테넌트에 admin 사용자 계정과 롤을 추가한다.

```
[root@nalee1 ~]# keystone user-role-add --user=admin --tenant=admin --role=admin
[root@nalee1 ~]# keystone user-role-add --user=admin --tenant=_member_ --
role=admin
```

6. 이번에는 demo 계정을 다음과 같이 생성한다.

```
[root@nalee1 ~]# keystone user-create --name=demo --pass=demopass ₩
--email=demo@email.com
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |          demo@email.com          |
| enabled  |               True               |
|    id    | 129a943bc94b43379a794ae011a87fe9 |
|   name   |               demo               |
| username |               demo               |
+----------+----------------------------------+
```

7. Demo 테넌트도 생성한다.

```
[root@nalee1 ~]# keystone tenant-create --name=demo --description="Demo Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |           Demo Tenant            |
|   enabled   |               True               |
|      id     | afa084dced1e4d70b479dac82ea659ac |
|     name    |               demo               |
+-------------+----------------------------------+
```

8. 생성한 Demo 계정을 Demo 테넌트에 추가하고 롤은 member를 설정한다.

```
[root@nalee1 ~]# keystone user-role-add --user=demo --role=_member_ --tenant=demo
```

9. 이번에는 오픈스택 서비스를 위한 서비스 테넌트를 다음과 같이 생성한다.

```
[root@nalee1 ~]# keystone tenant-create --name=service --description="Service Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |          Service Tenant          |
|   enabled   |               True               |
|      id     | 31303ac56ece4fbc85bb16074adfc9cb |
|     name    |             service              |
+-------------+----------------------------------+
```

10. 서비스 테넌트가 생성되면 이번에는 keystone 서비스를 다음과 같이 생성한다.

```
[root@nalee1 ~]# keystone service-create --name=keystone --type=identity ₩
  --description="OpenStack Identity"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |        OpenStack Identity        |
|   enabled   |               True               |
|      id     | 4fdf89fb665d44f7bf0dd053e8dfadef |
```

```
|     name    |             keystone             |
|     type    |             identity             |
+-------------+----------------------------------+
```

11. 생성된 Keystone 서비스는 사용자나 다른 서비스가 접속할 수 있도록 Rest API 주소를 생성한다. 필요한 파라메터는 생성한 서비스의 ID, 외부 접속 Public URL, 내부 접속 Internal URL, 관리자가 접속할 Admin URL로 이루어진다.

```
[root@nalee1 ~]# keystone endpoint-create ₩
  --service-id=4fdf89fb665d44f7bf0dd053e8dfadef ₩
  --publicurl=http://10.0.100.32:5000/v2.0 ₩
  --internalurl=http://10.0.100.32:5000/v2.0 ₩
  --adminurl=http://10.0.100.32:35357/v2.0
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
|   adminurl  |  http://10.0.100.32:35357/v2.0   |
|      id     | 27fd1214abd24430b18045b835885f6b |
| internalurl |   http://10.0.100.32:5000/v2.0   |
|  publicurl  |   http://10.0.100.32:5000/v2.0   |
|    region   |            region0ne             |
|  service_id | 4fdf89fb665d44f7bf0dd053e8dfadef |
+-------------+----------------------------------+
```

12. endpoint URL 생성이 완료되면 admin 계정을 생성할 때 추가했던 OS_SERVICE_TOKEN과 OS_SERVICE_ENDPOINT를 환경변수에서 해제한다

```
[root@nalee1 ~]# unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT
```

13. 이제 keystone 서비스가 정상적으로 실행되는지 테스트한다. 이때 환경변수를 쉽게 편집하기 위해 vi 에디터로 파일에 저장해두면 좋다.

```
[root@nalee1 ~]# vi admin-openrc.sh
export OS_USERNAME=admin
export OS_PASSWORD=adminpass
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://10.0.100.32:35357/v2.0
```

14. Vi 에디터로 만든 환경변수 설정 파일은 다음과 같이 source로 export할 수 있다.

```
[root@nalee1 ~]# source admin-openrc.sh
```

15. 사용자 목록도 확인해 본다.

```
[root@nalee1 ~]# keystone user-list
+----------------------------------+-------+---------+-----------------+
```

```
|                id                |  name | enabled |      email      |
+----------------------------------+-------+---------+-----------------+
| 7bc1bda9d2444b9c92554ff9e0b71e22 | admin |   True  | admin@email.com |
| 129a943bc94b43379a794ae011a87fe9 |  demo |   True  |  demo@email.com |
+----------------------------------+-------+---------+-----------------+
```

16. 사용자가 가지고 있는 롤 목록도 확인해 본다.

```
[root@nalee1 ~]# keystone user-role-list
+----------------------------------+-------+----------------------------------+-----
-----------------------------+
|                id                |  name |              user_id             |
tenant_id            |
+----------------------------------+-------+----------------------------------+-----
-----------------------------+
|  02ff77d31ec841b3914db4c3daca4f1c  |  admin  |  7bc1bda9d2444b9c92554ff9e0b71e22  |
dbd8df87e5c34536844d915b555061b6 |
+----------------------------------+-------+----------------------------------+-----
-----------------------------+
```

Step 5: Installation Swift Components

Keystone 설치까지 완료되면 마지막으로 Swift 컴포넌트를 설치합니다. Swift 컴포넌트를 설치할 때는 우선 Keystone에 Swift 사용자 계정을 생성한 후, Swift를 설치해야 합니다. 그럼, 지금부터 오픈스택의 오브젝트 스토리지 서비스인 Swift를 설치해 보도록 하겠습니다.

Create swift user to keystone

인증을 Keystone으로 사용할 계획이라면, Swift를 설치하기 전에 Keystone에 사용자 계정 및 서비스 그리고, 앤드포인트를 생성해야 합니다.

17. Keystone 사용자를 아래와 같은 명령어로 생성한다.

```
[root@nalee1 ~]# source admin-openrc.sh
[root@nalee1 ~]# keystone user-create --name=swift --pass=swiftpass ₩n
--email=swift@email.com
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |         swift@email.com          |
| enabled  |               True               |
|    id    | 7acccaa638224f05beb74950f8d12da9 |
```

```
|   name   |              swift               |
| username |              swift               |
+----------+----------------------------------+
```

18. 사용자를 생성했으면 이번에는 생성한 swift 사용자에 service 테넌트와 admin 롤을 설정해 준다.

```
[root@nalee1 ~]# keystone user-role-add --user=swift --tenant=service --role=admin
```

19. 이번에는 swift 서비스를 아래와 같은 명령어로 생성한다.

```
[root@nalee1 ~]# keystone service-create --name=swift --type=object-store ₩
--description="OpenStack Object Storage"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |     OpenStack Object Storage     |
|   enabled   |               True               |
|      id     | 2e69a88e0b7445919bf04bd277afa317 |
|     name    |              swift               |
|     type    |           object-store           |
+-------------+----------------------------------+
```

20. 서비스 생성이 되면 Endpoint를 생성해야 하는데 이때 사용할 URL은 프록시 노드의 URL을 입력해 주면 된다.

```
[root@nalee1 ~]# keystone endpoint-create ₩
  --service-id=2e69a88e0b7445919bf04bd277afa317 ₩
  --publicurl='http://10.0.100.32:8080/v1/AUTH_%(tenant_id)s' ₩
  --internalurl='http://10.0.100.32:8080/v1/AUTH_%(tenant_id)s' ₩
  --adminurl=http://10.0.100.32:8080
+-------------+-----------------------------------------+
|   Property  |                  Value                  |
+-------------+-----------------------------------------+
|   adminurl  |          http://10.0.100.32:8080        |
|      id     |       1d24ab0b9340418ba8059130e4b103ee  |
| internalurl | http://10.0.100.32:8080/v1/AUTH_%(tenant_id)s |
|  publicurl  | http://10.0.100.32:8080/v1/AUTH_%(tenant_id)s |
|    region   |                  region0ne              |
|  service_id |       2e69a88e0b7445919bf04bd277afa317  |
+-------------+-----------------------------------------+
```

Create swift configuration

Keystone 인증정보 설정이 완료되면 모든 swift 노드에 아래와 같이 swift.conf 만들고, 같은 그룹이라면 표현을 해 주어야 합니다.

21. /etc/swift 디렉터리를 하나 생성한다.

```
[root@nalee1 ~]# mkdir -p /etc/swift
```

22. 생성한 /etc/swift 디렉터리에 swift.conf 파일을 생성하고 아래와 같은 내용을 입력한다.

```
[root@nalee1 ~]# vi /etc/swift/swift.conf
[swift-hash]
# random unique string that can never change (DO NOT LOSE)
swift_hash_path_prefix = xrfuniounenqjnw
swift_hash_path_suffix = fLIbertYgibbitZ
```

Installation Swift Proxy

프록시 노드에 Proxy-server 컴포넌트들을 설치하고, 환경설정을 합니다. 이번 설치에서는 각각의 서버에 모두 Swift 컴포넌트를 설치할 예정이므로 각각의 노드에 모두 프록시 컴포넌트를 설치합니다.

23. 프록시 노드에 아래와 같은 컴포넌트들을 설치한다.

```
[root@nalee1 ~]# yum install openstack-swift-proxy memcached python-swiftclient python-keystone-auth-token
```

24. Memcached 의 서버 IP 정보를 변경한다. Memcached를 프록시 노드에 같이 설치하였으므로 프록시 노드의 IP를 아래와 같이 입력해 주면 된다.

```
[root@nalee1 ~]# vi /etc/sysconfig/memcached
OPTIONS="-l 10.0.100.32"
```

25. IP 정보를 변경후에 memcached 서비스를 시작한다. 그리고, 재부팅 후에도 서비스가 계속 실행될 수 있도록 chkconfig를 이용하여 서비스를 on 시킨다.

```
[root@nalee1 ~]# service memcached start
Starting memcached:                                        [  OK  ]
[root@nalee1 ~]# chkconfig memcached on
```

26. /etc/swift/proxy-server.conf 파일을 열어 keystone 정보를 아래와 같이 빨간색 부분으로 되어 있는 부분을 수정하거나 추가한다.

```
[root@nalee1 ~]# vi /etc/swift/proxy-server.conf
[DEFAULT]
bind_port = 8080
workers = 8
user = swift

[pipeline:main]
pipeline = healthcheck cache authtoken keystone proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true

[filter:cache]
use = egg:swift#memcache
memcache_servers = 10.0.100.32:11211

[filter:catch_errors]
use = egg:swift#catch_errors

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:keystone]
use = egg:swift#keystoneauth
operator_roles = admin, SwiftOperator
is_admin = true
cache = swift.cache

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
```

```
admin_tenant_name = service
admin_user = swift
admin_password = swiftpass
auth_host = 10.0.100.32
auth_port = 35357
auth_protocol = http
signing_dir = /tmp/keystone-signing-swift
```

Installation Swift Storage

프록시 노드 설치가 끝나면 이번에는 스토리지 노드용 컴포넌트들을 설치합니다. 일반적인 멀티 노드에서는 스토리지용 노드에 아래와 같은 컴포넌트를 설치하면 되고, 서버 한대에 설치하여 테스트 할 경우에는 프록시 노드에 설치하면 됩니다.

27. 스토리지 노드에 아래와 같은 컴포넌트들을 설치한다. 이때 한 노드에 모든 컴포넌트 설치 테스트를 할 예정이라면 해당 노드에 함께 설치하면 된다.

```
[root@nalee1 ~]# yum install openstack-swift-account openstack-swift-container ₩
  openstack-swift-object xfsprogs xinetd
```

28. 이번에는 스토리지 저장 공간으로 쓰일 디바이스를 설정해 보자. 아래 같은 경우에는 추가 디바이스가 있을 경우의 디바이스 설정방법이다. Fdisk를 이용하여 사용할 만큼의 디바이스 파티션을 생성하는데 대체적으로 전부 사용할 수 있도록 설정한다.

```
[root@nalee1 ~]# fdisk /dev/vdb

Command (m for help): n
Partition type:
   p   primary (1 primary, 1 extended, 2 free)
   l   logical (numbered from 5)
p
Partition number (1-4, default 3): 1
First sector (0-20805, default 0):
Using default value 0
Last sector, +sectors or +size{K,M,G} (20805- 20805, default 20805):
Using default value 488397167

Command (m for help): w
```

29. 파티셔닝 된 디바이스는 xfs 형태로 아래와 같이 포맷을 한다.

```
[root@nalee1 ~]# mkfs.xfs /dev/vdb1
meta-data=/dev/vdb1      isize=256    agcount=4, agsize=983040 blks
         =               sectsz=512   attr=2, projid64bit=1
         =               crc=0
data     =               bsize=4096   blocks=39322169, imaxpct=25
         =               sunit=0      swidth=0 blks
naming   =version 2      bsize=4096   ascii-ci=0 ftype=0
log      =internal log   bsize=4096   blocks=2560, version=2
         =               sectsz=512   sunit=0 blks, lazy-count=1
realtime =none           extsz=4096   blocks=0, rtextents=0
```

30. 리눅스에서 포맷 된 디바이스는 마운트를 해야 사용을 할 수 있다. 마운트를 하기 위해 /etc/fstab에 마운트 정보를 추가한다. 그리고, 연결한 디렉터리를 생성하고 생성한 디렉터리를 이용하여 디바이스 마운트를 한다. 마운트 한 디바이스는 swift가 데이터를 저장하고 읽을 때 사용할 예정이므로 사용자 접속권한을 swift로 변경한다.

```
[root@nalee1 ~]# echo "/dev/vdb1 /srv/node/sdb1 xfs
noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
[root@nalee1 ~]# mkdir -p /srv/node/sdb1
[root@nalee1 ~]# mount /srv/node/sdb1
[root@nalee1 ~]# chown -R swift:swift /srv/node
```

31. 마운트 된 스토리지용 디렉터리는 rsyncd.conf에 해당 정보를 입력해 주어야 한다. 우선, /etc/rsyncd.conf 파일을 생성하고, 아래와 같은 내용을 추가한다.

```
[root@nalee1 ~]# vi   /etc/rsyncd.conf
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 10.0.100.32

[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
```

32. 서버 동기화를 위한 rsync를 사용하기 위해서는 rsync를 활성화시켜야 한다. 그러기 위해서, /etc/xinetd.d/rsync의 disable의 yes를 아래와 같이 no로 변경해야 한다.

```
[root@nalee1 ~]# vi /etc/xinetd.d/rsync
# default: off
# description: The rsync server is a good addition to an ftp server, as it ₩
#       allows crc checksumming etc.
service rsync
{
        disable   = no
        flags           = IPv6
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/bin/rsync
        server_args     = --daemon
        log_on_failure  += USERID
}
```

33. Rsync를 활성화로 변경했으면 이번에는 xinted를 재시작한다. 그리고, 재부팅시에도 계속 실행이 될 수 있도록 chkconfig를 이용해 xinetd 서비스를 on 시킨다.

```
[root@nalee1 ~]# service xinetd start
Starting xinetd:                                           [  OK  ]
```

```
[root@nalee1 ~]# chkconfig xinetd on
```

34. Swift recon 디렉토리를 생성하고, 사용자 권한을 swift로 변경한다.

```
[root@nalee1 ~]# mkdir -p /var/swift/recon
[root@nalee1 ~]# chown -R swift:swift /var/swift/recon
```

Create Ring builder

프록시 노드와 스토리지 노드 설정이 끝나면 이제 링 빌더를 생성해야 합니다. 멀티노드로 구성을 할 경우 프록시 노드에서 링 빌더를 생성한 후 반드시 각각의 스토리지 노드로 생성한 링 빌더를 복사하여야 합니다. 본 문서는 서버 한대에 모든 컴포넌트를 다 설치하는 테스트 문서로써 해당 부분은 빠져 있습니다.

35. /etc/swift 디렉터리로 이동하여 account, container, object 링을 생성한다.

```
[root@nalee1 ~]# cd /etc/swift
[root@nalee1 swift]# swift-ring-builder account.builder create 18 1 1
[root@nalee1 swift]# swift-ring-builder container.builder create 18 1 1
[root@nalee1 swift]# swift-ring-builder object.builder create 18 1 1
```

*** 이때 스토리지 노드가 4대 이상이라면 create 18 3 1로 설정하고, 스토리지 노드가 3대 이상이라면 create 18 2 1로 설정하며, 스토리지 노드가 2대 이하라면 create 18 1 1로 설정한다.

36. 생성한 링 빌더에 각각 디바이스 정보를 추가한다.

```
[root@nalee1 swift]# swift-ring-builder account.builder add r1z1-10.0.100.32:6002/sdb1 100
Device d0r1z1-10.0.100.32:6002R10.0.100.32:6002/sdb1_"" with 100.0 weight got id 0
[root@nalee1 swift]# swift-ring-builder container.builder add r1z1-10.0.100.32:6001/sdb1 100
Device d0r1z1-10.0.100.32:6001R10.0.100.32:6001/sdb1_"" with 100.0 weight got id 0
[root@nalee1 swift]# swift-ring-builder object.builder add r1z1-10.0.100.32:6000/sdb1 100
Device d0r1z1-10.0.100.32:6000R10.0.100.32:6000/sdb1_"" with 100.0 weight got id 0
```

37. 추가한 디바이스 정보가 정상적으로 추가되었는지 아래와 같은 명령어를 이용하여 확인한다.

```
[root@nalee1 swift]# swift-ring-builder account.builder
account.builder, build version 1
262144 partitions, 1.000000 replicas, 1 regions, 1 zones, 1 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  10.0.100.32  6002      10.0.100.32              6002 sdb1 100.00     262144    0.00
[root@nalee1 swift]# swift-ring-builder container.builder
container.builder, build version 1
262144 partitions, 1.000000 replicas, 1 regions, 1 zones, 1 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  10.0.100.32  6001      10.0.100.32              6001 sdb1 100.00     262144    0.00
[root@nalee1 swift]# swift-ring-builder object.builder
object.builder, build version 1
262144 partitions, 1.000000 replicas, 1 regions, 1 zones, 1 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  10.0.100.32  6000      10.0.100.32              6000 sdb1 100.00     262144    0.00
```

38. 생성한 링 빌더를 각각 리밸런싱한다.

```
[root@nalee1 swift]# swift-ring-builder account.builder rebalance
[root@nalee1 swift]# swift-ring-builder container.builder rebalance
[root@nalee1 swift]# swift-ring-builder object.builder rebalance
```

39. 마지막으로 /etc/swift 디렉터리 내의 모든 파일의 소유권을 swift로 변경한다.

```
[root@nalee1 ~]# chown -R swift:swift /etc/swift
```

Start Swift Service

모든 서비스 설치와 환경설정을 하고, 링 빌더까지 생성을 했으면 이번에는 Swift 서비스들을 시

작해 주어야 합니다. 아래와 같이 프록시 서버, 어카운트 서버, 컨테이너 서버, 오브젝트 서버들을 시작해 보도록 하겠습니다.

40. Swift를 실행하기 위한 모든 환경설정이 끝났으면 이제 proxy 서비스를 시작해보자. 아래와 같은 명령으로 swift-proxy 서비스를 시작하고, 재부팅시 서비스가 계속 실행될 수 있도록 chkconfig를 이용하여 on 시킨다.

```
[root@nalee1 ~]# service openstack-swift-proxy start
Starting openstack-swift-proxy:                            [  OK  ]
[root@nalee1 ~]# chkconfig openstack-swift-proxy on
```

41. Proxy 서비스를 실행했다면 이번에는 스토리지 노드의 서비스들을 아래와 같은 명령어로 실행시킨다.

```
[root@nalee1 ~]# for service in ₩
   openstack-swift-object openstack-swift-object-replicator openstack-swift-object-
updater openstack-swift-object-auditor openstack-swift-container openstack-swift-
container-replicator openstack-swift-container-updater openstack-swift-container-
auditor openstack-swift-account openstack-swift-account-replicator openstack-swift-
account-reaper openstack-swift-account-auditor; do ₩
service $service start; chkconfig $service on; done
Starting openstack-swift-object:                           [  OK  ]
Starting openstack-swift-object-replicator:                [  OK  ]
Starting openstack-swift-object-updater:                   [  OK  ]
Starting openstack-swift-object-auditor:                   [  OK  ]
Starting openstack-swift-container:                        [  OK  ]
Starting openstack-swift-container-replicator:             [  OK  ]
Starting openstack-swift-container-updater:                [  OK  ]
Starting openstack-swift-container-auditor:                [  OK  ]
Starting openstack-swift-account:                          [  OK  ]
Starting openstack-swift-account-replicator:               [  OK  ]
Starting openstack-swift-account-reaper:                   [  OK  ]
Starting openstack-swift-account-auditor:                  [  OK  ]
```

Test Swift Service

마지막으로 Swift가 제대로 설치가 되었는지 테스트를 해 보아야 합니다. 아래와 같이 rc 파일을 이용하여 쉽게 Swift 테스트를 해 봅니다.

42. 테스트를 위해 swift 서비스 설치 시 생성해 두었던 openrc 파일을 export 시킨다.

```
[root@nalee1 ~]# source admin-openrc.sh
```

43. Admin 계정의 swift 상태를 확인해 본다.

```
[root@nalee2 ~]# swift stat
          Account: AUTH_12a3ef369ba94a7ebcfc17113b2feaa6
       Containers: 0
          Objects: 0
            Bytes: 0
   Content-Type: text/plain; charset=utf-8
    X-Timestamp: 1422327355.92825
     X-Trans-Id: tx55cc97ecbd4f4d18bcef0-0054c6fe3b
X-Put-Timestamp: 1422327355.92825
```

44. myfiles라는 컨테이너를 만들고 그 안에 test.txt를 올려보자.

```
[root@nalee2 ~]# swift upload myfiles test.txt
test.txt
[root@nalee2 ~]# swift upload myfiles test2.txt
test2.txt
```