Openstack Swift Multi Server 한글 매뉴얼

By 2011.11.29 MNL SOLUTIN 장현정 책임

모든 서버에 다음과 같이 설치합니다.

1. 우선 가장 최신 버전인 Ubuntu11.10 64bit Server를 가지고 있는 두 대의 서버에 설치를 합니다. (Multi server니 최소 두 대는 있어야겠죠~!! ^^ 더 많이 있으면 더 좋구요~!!)

2. 그리고 다음과 같은 명령어를 따라 입력합니다.

```
$sudo apt-get install python-software-properties
$sudo add-apt-repository ppa:swift-core/ppa
$sudo apt-get update
$sudo apt-get install swift openssh-server
```

3. 다음과 같이 swift 폴더를 생성하고 소유권을 swift로 변경합나다.

```
mkdir -p /etc/swift
chown -R swift:swift /etc/swift/
```

4. 위에서 생성한 /etc/swift/ 폴더에 swift.conf 파일을 만들어 줍니다.

```
[swift-hash]
# 변경하기 곤란한 유일한 값으로 suffix를 입력하시면 됩니다.
swift_hash_path_suffix = changeme
```

Auth Node를 설치해 봅시다.

1. 우선 git을 설치합니다.

```
$sudo apt-get install git-core
```

2. Git이 설치가 되면 home디렉토리에 swauth를 받아와 설치합니다.

```
$git clone https://github.com/gholt/swauth.git
$cd swauth
$python setup.py install
```

Proxy Node 설치 및 환경설정

1. Swift-proxy service를 설치합니다.

```
$sudo apt-get install swift-proxy memcached
```

2. SSL을 위한 cert 파일을 생성 해야 하나, 하지 않아도 됩니다.

```
$cd /etc/swift
$openssl req -new -x509 -nodes -out cert.crt -keyout cert.key
```

3. /etc/memcached.conf의 다음 라인을 변경합니다.

```
-l 127.0.0.1
to
-l <PROXY_LOCAL_NET_IP>
```

4. 이제 memcached server를 재시작합니다.

```
$service memcached restart
```

5. /etc/swift/proxy-server.conf 파일을 생성합니다.

```
[DEFAULT]
# Enter these next two values if using SSL certifications
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
bind_port = 8080
workers = 8
user = swift

[pipeline:main]
# keep swauth in the line below if you plan to use swauth for
authentication
pipeline = healthcheck cache swauth proxy-server

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

[filter:swauth]
# the line below points to swauth as a separate project from swift
use = egg:swauth#swauth
super_admin_key = swauthkey
# 어형부형님의 Tip~!! Swauth가 설치되어 있는 주소를 꼭 입력해야 합니다.
default_swift_cluster = local#http://<SWAUTH_IP>:8080/v1

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

[filter:cache]
use = egg:swift#memcache
memcache_servers = <PROXY_LOCAL_NET_IP>:11211
```

** 이때 memcache server가 여러대라면 다음과 같이 환경설정을 합니다.

```
10.1.2.3:11211,10.1.2.4:11211
```

6. 이제 account, container, object ring을 생성합니다.

```
cd /etc/swift
swift-ring-builder account.builder create 18 1 1
swift-ring-builder container.builder create 18 1 1
swift-ring-builder object.builder create 18 1 1
```

** 아마도 Original 매뉴얼에는 create 18 3 1로 되어 있을 것 입니다. 하지만 우리는 서버가 두 대 뿐이므로 18 1 1 이라고 입력해야 합니다. 한대는 Proxy서버이고, 한대는 Storage서버이기 때문이죠~!!

7. Ring-builder를 이용하여 storage device를 추가합니다.

```
swift-ring-builder account.builder add z<ZONE>-
<STORAGE_LOCAL_NET_IP>:6002/<DEVICE> 100
swift-ring-builder container.builder add z<ZONE>-
<STORAGE_LOCAL_NET_IP_1>:6001/<DEVICE> 100
swift-ring-builder object.builder add z<ZONE>-
<STORAGE_LOCAL_NET_IP_1>:6000/<DEVICE> 100
```

** 예를 들면 다음과 같이 입력할 수 있습니다.

```
swift-ring-builder account.builder add z1-10.0.0.1:6002/sdb1 100
swift-ring-builder container.builder add z1-10.0.0.1:6001/sdb1 100
swift-ring-builder object.builder add z1-10.0.0.1:6000/sdb1 100
```

8. Device를 추가했으면 다음과 같은 명령어로 확인을 꼭 해야 합니다.

```
swift-ring-builder account.builder
swift-ring-builder container.builder
swift-ring-builder object.builder
```

9. Ring을 Rebalance합니다.

```
swift-ring-builder account.builder rebalance
swift-ring-builder container.builder rebalance
swift-ring-builder object.builder rebalance
```

10. /etc/swift 폴더에 생성된 account.ring.gz, container.ring.gz, object.ring.gz 파일을 다른 스토리지 서버의 /etc/swift 폴더에 복사합니다.

```
$scp -P 22 swift@<PROXY_LOCAL_NET_IP>:/etc/swift/*.ring.gz ./
```

11. /etc/swift 폴더내의 모든 파일들을 swift 소유로 변경합니다.

```
$sudo chown -R swift:swift /etc/swift
```

12. 이제 Proxy service를 시작합니다.

```
$sudo swift-init proxy start
```

Storage Node 설치 및 환경설정

1. Storage node 에 다음과 같은 package 를 설치합니다.

```
$sudo apt-get install swift-account swift-container swift-object
xfsprogs
```

2. Storage node 를 다음과 같은 xfs volume 으로 설정합니다.

```
$sudo fdisk /dev/sdb  (set up a single partition)
$sudo mkfs.xfs -i size=1024 /dev/sdb1
$sudo echo "/dev/sdb1 /srv/node/sdb1 xfs
noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
$sudo mkdir -p /srv/node/sdb1
$sudo mount /srv/node/sdb1
$sudo chown -R swift:swift /srv/node
```

3. /etc/rsyncd.conf 파일을 생성합니다.

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = <STORAGE_LOCAL_NET_IP>

[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
```

4. /etc/default/rsync 파일을 다음과 같이 수정합니다.

```
RSYNC_ENABLE = true
```

5. Rsync daemon 을 시작합니다.

```
$sudo service rsync start
```

6. /etc/swift/account-server.conf 파일을 생성합니다.

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP>
workers = 2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]
```

7. /etc/swift/container-server.conf 파일을 생성합니다.

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP>
workers = 2

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container

[container-replicator]

[container-updater]

[container-auditor]
```

8. /etc/swift/object-server.conf 파일을 생성합니다.

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP>
workers = 2

[pipeline:main]
pipeline = object-server

[app:object-server]
use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]
```

9. Storage serivice 를 시작합니다.

```
swift-init object-server start
swift-init object-replicator start
swift-init object-updater start
swift-init object-auditor start
swift-init container-server start
swift-init container-replicator start
swift-init container-updater start
swift-init container-auditor start
swift-init account-server start
swift-init account-replicator start
swift-init account-auditor start
```

Openstack Object Storage admin Account 생성 및 테스트

1. 다음과 같이 swauth-prep 를 실행합니다.

```
swauth-prep -K key -A http://<AUTH_HOSTNAME>:8080/auth/
```

2. 사용자를 추가합니다.

```
swauth-add-user -K key -A http://<AUTH_HOSTNAME>:8080/auth/ -a
system root testpass
```

3. X-Storage-Url 과 X-Auth_Token 를 구합니다.

```
curl -k -v -H 'X-Storage-User: system:root' -H 'X-Storage-Pass:
testpass' http://<AUTH_HOSTNAME>:8080/auth/v1.0
```

4. Account HEAD 를 확인합니다.

```
curl -k -v -H 'X-Auth-Token: <token-from-x-auth-token-above>' <url-
from-x-storage-url-above>
```

5. Swift Tool, swift 상태를 확인합니다.

```
swift -A http://<AUTH_HOSTNAME>:8080/auth/v1.0 -U system:root -K
testpass stat
```

6. Myfiles 라는 컨테이터에 파일을 하나 올려봅니다.

```
swift -A http://<AUTH_HOSTNAME>:8080/auth/v1.0 -U system:root -K
testpass upload myfiles bigfile1.tgz
swift -A http://<AUTH_HOSTNAME>:8080/auth/v1.0 -U system:root -K
testpass upload myfiles bigfile2.tgz
```

7. Myfiles 컨테이너의 모든 파일을 다운로드 받아봅니다.

```
swift -A http://<AUTH_HOSTNAME>:8080/auth/v1.0 -U system:root -K
testpass download myfiles
```

8. Swauth Web Admin 을 설치할 경우에는 아래와 같이 입력합니다.

```
$cd swauth
$swift -A http://<AUTH_HOSTNAME>:8080/auth/v1.0 -
U .super_admin:.super_admin -K swauthkey upload .webadmin .
```