ucloud storage 2.0

Swift 설치 및 이용 가이드

2017. 1

문서 버전 및 이력

버전	일자	이력사항
v1	2016.12	최초 작성

목차

1 개요

'swift'는 openstack 커뮤니티에서 개발하는 클라우드 스토리지 프로젝트의 코드명이다. KT는 해당 프로젝트를 기반으로 ucloud storage 2.0을 구축했으며 따라서 swift 툴을 이용해서 KT ucloud storage 2.0 서비스를 이용 할 수 있다.
swift 프로젝트 내부의 파일 중에 swift 명령이 존재하는데 swift 패키지에 포함된 커맨드 라인 유틸리티로써, 기본 파일 이름은 swift이다. 사용자는 이 툴을 이용해 손쉽게 swift 기반 클라우드 스토리지에 에 파일을 업로드, 다운로드 및 기타 관리를 수행할 수 있다.

(참고 - swift의 파일 관리 구조)

swift는 파일을 관리하기 위해 내부적으로 각 이용자 계정에 대응되는 '어카운트(account)' 아래 컨테이너라(container)는 중간 계층을 제공한다. KT 클라우드 스토리지 서비스를 사용하는 경우에 ucloudbiz.olleh.com 포탈에서 storage service를 조회하면 파일박스와 파일들이 나오는데 컨테이너는 파일박스에 해당한다.
컨테이너는 파일을 모아두는 논리적 폴더라고 생각할 수 있다. 단 주의할 점은 컨테이너 안에 다시 컨테이너를 계층적으로 만들 수는 없다. 일반 OS의 디렉터리(directory)나 폴더 개념과 비교해 보면 일반 OS에서는 기본 루트 아래 여러 개의 하위 디렉터리를 만들고 파일을 저장할 수는 있으나, 컨테이너는 하부에 다시 서브 컨테이너를 만들 수 없다. (이를 가리켜 평평한 flat 구조라고 하고 swift와 유사 서비스인 Amazon S3의 bucket도 유사한 개념으로 동작한다.) 실제로 컨테이너는 디렉터리를 구분하는 용도 보다는 저장, CDN 연동, 공개 등의 관리 측면에서 사용 용도를 구분하기 위해 활용된다. 이때 기존 파일시스템에서 유지하던 계층적 디렉터리를 클라우드 스토리지에서 관리하는 것이 문제가 될 수 있는데 이를 어떻게 지원하고 사용할 수 있는지 뒤에서 설명한다.
이후 설명에서 swift는 실제 저장되는 파일들을 오브젝트(object)라고 부른다. 그러나 별도의 구분이 없는 한 오브젝트 대신 파일이라는 용어를 사용하며 두 용어는 동일하게 간주한다. swift에서 파일을 논리적으로 관리하는 계층은 어카운트 - 컨테이너(flat구조) - 파일(오브젝트) 라고 생각할 수 있다.

swift 툴에 익숙하지 않은 경우에는 이어지는 2. 설치 및 3. 기본사용 내용을 따라 한번 툴을 이용해 보고, 이후에 필요한 세부 동작이나 명령을 파악하기 위해 뒤의 내용들을 참고하기를 바란다.

이후 내용에서 설명상 편의를 위해 직접 명령을 수행하고 그 결과를 나타내었는데 '#' 로 시작하는 줄은 이용자가 직접 입력하는 내용을 표시한다. '<enter>' 는 엔터를 입력하라는 의미이다. 또한 명령 사용 예시는 리눅스를 기반으로 되어 있다.

2 설치

가) 설치 환경 준비

OpenStack

swift 툴 명령어 사용을 위해서는 인증 클라이언트인 python-keystoneclient의 설치도 필요하다. Openstack에서 제공하는 swift 명령어와 keystone 클라이언트는 파이썬 2.7, 3.4 그리고 3.5 버전에서 구동되므로 해당 툴을 설치 및 이용하기 이전에 파이썬 환경을 확인하고 설정하는 것이 중요하다. 설치는 크게 리눅스 – 데비안 계열(Debian, Ubuntu), 레드햇 계열(Fedora, CentoOS), 그리고 윈도우 계열로 구분되므로 사용자 환경에 맞는 설명을 참고한다. Openstack 커뮤니티에서 개발하는 swift 툴에 대한 설정 문서는 해당 커뮤니티 (http://www.openstack.org)를 참고한다.

나) 데비안(Debian) 계열 – Debian, Ubuntu

아래 설명은 Debian 7 및 Ubuntu 12/14배포판의 32bit, 64bit 버전에서 각각 테스트 된 내용이다.

ㅇ 파이썬 버전 확인

먼저 파이썬 환경을 확인한다.

```
~ # python
Python 2.7.3 (default, Dec 18 2014, 19:10:20)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

파이썬 버전은 2.7, 3.4 그리고 3.5를 지원한다.

ㅇ 설치 준비

python-swiftclient 패키지를 설치하기 전에 필요한 패키지를 설치한다.

```
~ # apt-get install python-setuptools python-pip git <enter>
```

ㅇ python-swiftclient 패키지 다운로드

```
git clone https://github.com/openstack/python-swiftclient.git
```

ㅇ python-swiftclient 패키지 설치

```
~# cd python-swiftclient <enter>
```

파일 중에 requirements.txt 라는 파일을 아래와 같이 수정해 준다.

```
~# vi requirements.txt <enter>
futures>=3.0
requests>=1.1
six>=1.5.2
```

이제 필요한 패키지를 설치하고 python-swiftclient 패키지까지 설치한다.

```
~# pip install –r requirements.txt <enter>
~# python setup.py develop <enter>
```

이후에 swift 명령을 실행해서 아래와 같이 도움말이 출력되면 정상으로 설치된 것으로 간주한다.

```
# swift --
usage: swift [--version] [--help] [--os-help] [--snet] [--verbose]
            [--debug] [--info] [--quiet] [--auth <auth_url>]
            [--auth-version <auth_version> |
                --os-identity-api-version <auth_version> ]
            [--user <username>]
            [--key <api_key>] [--retries <num_retries>]
            [--os-username <auth-user-name>] [--os-password <auth-password>]
< 생 략 >
```

python-swiftclient 는 정상적으로 설치가 되었지만 ucloud storage 2.0에 인증을 위해서는 python-keystoneclient가 설치 되어야 한다.

ㅇ 설치 준비

```
~# apt-get install python-dev <enter>
~# pip install --upgrade setuptools <enter>
```

ㅇ python-keystoneclient 패키지 다운로드

```
~# git clone  https://github.com/openstack/python-keystoneclient.git
```

ㅇ python-keystoneclient 패키지 설치

```
~# cd python-keystoneclient <enter>
```

이제 필요한 패키지를 설치하고 python-keystoneclient 패키지까지 설치한다.

```
~# pip install –r requirements.txt <enter>
~# python setup.py develop <enter>
```

정상적으로 설치가 되었다면, keystone 인증을 통해 정상적으로 swift 명령어가 동작하는지 확인한다.

ㅇ 서비스 확인

아래와 같이 인증을 위한 사용자 정보를 파일로 만들어 놓으면 더욱 편리하게 swift CLI를 이용할 수 있다. Domain Id, Project ID 등에 관한 정보는 클라우드 콘솔 포탈 → ucloud storage → ucloud storage 2.0 → API key에서 확인할 수 있다.

```
~# vi openrc <enter>
export OS_PROJECT_DOMAIN_ID=<Domain ID>
export OS_USER_DOMAIN_ID=<Domain ID>
export OS_PROJECT_ID=<Project ID>
export OS_USERNAME=<Access Key ID>
export OS_PASSWORD=<Secret Key>
export OS_AUTH_URL=https://ssproxy2.ucloudbiz.olleh.com:5000/v3
export OS_IDENTITY_API_VERSION=3

~# source openrc <enter>
```

정상적으로 설치가 되었다면, swift 명령어가 아래와 같이 실행 가능하다.

```
~# swift stat
Account: AUTH_ e40e31fb1f9e02c4f531e2a17d4b8bc5
                      Containers: 7
                         Objects: 32
                           Bytes: 60416027249
Containers in policy "erasurecode": 2
   Objects in policy "erasurecode": 13
     Bytes in policy "erasurecode": 26624000000
      Containers in policy "3copy": 5
         Objects in policy "3copy": 19
           Bytes in policy "3copy": 33792027249
                 Meta Quota-Bytes: 2000000000000
                Meta Temp-Url-Key: testleeefef
                      X-Trans-Id: tx52989495c9c66bfb3ac26-005873181a
       X-Account-Project-Domain-Id: 9ae064419a 4c272b2058c2f2dac3fc90
                      Connection: close
                     X-Timestamp: 1482311682.66941
                    Content-Type: text/plain; charset=utf-8
                   Accept-Ranges: bytes
```

다) 레드햇(Redhat) 계열 – Fedora, CentOS

아래 설명은 Redhat Fedora 및 CentOS6,7 배포판의 32bit, 64bit 버전에서 각각 테스트 된 내용이다.

o 파이썬 버전 확인

먼저 파이썬 환경을 확인한다.

```
~# python
Python 2.7.12 (default, Jul  1 2016, 15:12:24)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.>>> quit() <enter>
~# quit()
```

파이썬 버전은 2.7, 3.4 그리고 3.5를 지원한다. CentOS6에서는 python 버전이 2.6.x 버전이므로 적합한 python 패키지를 다운받아 설치해 주어야 한다.(python 버전을 확인해서 지원하는 버전이라면 파이썬 설치는 그냥 넘어가도록 한다.) 또한 CentOS6 버전에서는 setuptools와 pip 패키지도 별도로 설치해 주어야 한다. 따라서 본격적인 클라이언트 설치 전에 준비되는 환경은 CentOS6과 그 외 버전을 따로 나누어 놓았으며, python-swiftclient 및 python-keystoneclient 설치는 공통적으로 진행하도록 한다.

< CentOS6 전용 설치 전 환경 준비 과정>

○ python 설치하기

python 설치를 위해 필요한 패키지들을 설치해 주어야 한다.(만약 이미 python을 설치 한 상태라면, 다시 python을 재설치 해 주어야 한다.- ./configure 부터 진행한다.)

```
~# yum -y install openssl-devel bzip2-devel sqlite-devel zlib-devel gcc git <enter>
```

Python-2.7.9 패키지를 다운로드 한다.

```
~# wget https://www.python.org/ftp/python/2.7.9/Python-2.7.9.tgz <enter>
~# tar xvzf Python-2.7.9.tgz <enter>
~# cd Python-2.7.9 <enter>
```

이제 새로운 버전의 Python을 설치해 준다. 여기에서 경로는 임의로 정하였다.

```
~# ./configure --prefix=/usr/local <enter>
~# make && make altinstall <enter>
```

Python-2.7.9 버전이 잘 설치 되었는지 아래와 같이 확인한다.

```
~# python2.7 –version <enter>
Python 2.7.9
```

○ setuptools 및 pip 설치하기

```
~# wget --no-check-certificate <enter>
<enter>https://pypi.python.org/packages/source/s/setuptools/setuptools-12.0.4.tar.gz
~# tar xvzf setuptools-12.0.4.tar.gz <enter>
~# cd setuptools-12.0.4 <enter>
~# python2.7 setup.py install <enter>
~# cd .. <enter>

~# wget --no-check-certificate  https://pypi.python.org/packages/source/p/pip/pip-
6.0.6.tar.gz <enter>
~# tar xvzf pip-6.0.6.tar.gz <enter>
~# cd pip-6.0.6 <enter>
~# python2.7 setup.py install <enter>
~# cd .. <enter>
```

< CentOS7 전용 설치 전 환경 준비 과정>

별도의 python 버전이 필요하지 않다. 환경 준비를 위해 기본적인 패키지를 다음과 같이 설치해준다.

```
~# wget --no-check-certificate  https://pypi.python.org/packages/source/p/pip/pip-
6.0.6.tar.gz <enter>
~# tar xvzf pip-6.0.6.tar.gz <enter>
~# cd pip-6.0.6 <enter>
```

```
~# python2.7 setup.py install <enter>
~# cd .. <enter>

~# yum –y install gcc
```

< 공통: python-swiftclient 및 python-keystoneclient 설치 >

여기부터는 Fedora, CentOS 6과 7 공통적으로 수행한다.

ㅇ python-swiftclient 패키지 다운로드

```
git clone https://github.com/openstack/python-swiftclient.git
```

ㅇ python-swiftclient 패키지 설치

```
~# cd python-swiftclient <enter>
```

이제 필요한 패키지를 설치하고 python-swiftclient 패키지까지 설치한다.

```
~# pip install –r requirements.txt <enter>
~# python setup.py develop <enter>
```

이후에 swift 명령을 실행해서 아래와 같이 도움말이 출력되면 정상으로 설치된 것으로 간주한다.

```
# swift --
usage: swift [--version] [--help] [--os-help] [--snet] [--verbose]
            [--debug] [--info] [--quiet] [--auth <auth_url>]
            [--auth-version <auth_version> |
                --os-identity-api-version <auth_version> ]
            [--user <username>]
            [--key <api_key>] [--retries <num_retries>]
            [--os-username <auth-user-name>] [--os-password <auth-password>]
< 생 략 >
```

python-swiftclient 는 정상적으로 설치가 되었지만 ucloud storage 2.0에 인증을 위해서는 python-keystoneclient가 설치 되어야 한다.

ㅇ 설치 준비

```
~# yum -y install python-devel <enter>
```

ㅇ python-keystoneclient 패키지 다운로드

```
~# git clone  https://github.com/openstack/python-keystoneclient.git
```

ㅇ python-keystoneclient 패키지 설치

```
~# cd python-keystoneclient <enter>
```

이제 필요한 패키지를 설치하고 python-keystoneclient 패키지까지 설치한다.

```
~# pip install –r requirements.txt <enter>
~# python setup.py develop <enter>
```

정상적으로 설치가 되었다면, keystone 인증을 통해 정상적으로 swift 명령어가 동작하는지 확인한다.

ㅇ 서비스 확인

아래와 같이 인증을 위한 사용자 정보를 파일로 만들어 놓으면 더욱 편리하게 swift CLI를 이용할 수 있다. Domain Id, Project ID 등에 관한 정보는 클라우드 콘솔 포탈 → ucloud storage → ucloud storage 2.0 → API key에서 확인할 수 있다.

```
~# vi openrc <enter>
export OS_PROJECT_DOMAIN_ID=<Domain ID>
export OS_USER_DOMAIN_ID=<Domain ID>
export OS_PROJECT_ID=<Project ID>
export OS_USERNAME=<Access Key ID>
export OS_PASSWORD=<Secret Key>
export OS_AUTH_URL=https://ssproxy2.ucloudbiz.olleh.com:5000/v3
export OS_IDENTITY_API_VERSION=3

~# source openrc <enter>
```

정상적으로 설치가 되었다면, swift 명령어가 아래와 같이 실행 가능하다.

```
~# swift stat
Account: AUTH_ e40e31fb1f9e02c4f531e2a17d4b8bc5
                      Containers: 7
                         Objects: 32
                           Bytes: 60416027249
Containers in policy "erasurecode": 2
   Objects in policy "erasurecode": 13
     Bytes in policy "erasurecode": 26624000000
      Containers in policy "3copy": 5
         Objects in policy "3copy": 19
           Bytes in policy "3copy": 33792027249
                 Meta Quota-Bytes: 2000000000000
                Meta Temp-Url-Key: testleeefef
                      X-Trans-Id: tx52989495c9c66bfb3ac26-005873181a
       X-Account-Project-Domain-Id: 9ae064419a 4c272b2058c2f2dac3fc90
                      Connection: close
                     X-Timestamp: 1482311682.66941
                    Content-Type: text/plain; charset=utf-8
                   Accept-Ranges: bytes
```

3 기본 사용법 - 기본 명령 및 옵션

가) 커맨드 라인에서 swift 명령을 실행 및 도움말 확인

swift 를 그냥 실행하면 하면 다음과 같은 도움말이 출력된다.

```
Usage: swift [--version] [--help] [--os-help] [--snet] [--verbose]
            [--debug] [--info] [--quiet] [--auth <auth_url>]
            [--auth-version <auth_version> |
                --os-identity-api-version <auth_version> ]
            [--user <username>]
            [--key <api_key>] [--retries <num_retries>]
            [--os-username <auth-user-name>] [--os-password <auth-password>]
            [--os-user-id <auth-user-id>]
            [--os-user-domain-id <auth-user-domain-id>]
            [--os-user-domain-name <auth-user-domain-name>]
            [--os-tenant-id <auth-tenant-id>]
            [--os-tenant-name <auth-tenant-name>]
            [--os-project-id <auth-project-id>]
            [--os-project-name <auth-project-name>]
            [--os-project-domain-id <auth-project-domain-id>]
            [--os-project-domain-name <auth-project-domain-name>]
            [--os-auth-url <auth-url>] [--os-auth-token <auth-token>]
            [--os-storage-url <storage-url>] [--os-region-name <region-name>]
            [--os-service-type <service-type>]
            [--os-endpoint-type <endpoint-type>]
            [--os-cacert <ca-certificate>] [--insecure]
            [--no-ssl-compression]
            <subcommand> [--help] [<subcommand options>]

Command-line interface to the OpenStack Swift API.

Positional arguments:
  <subcommand>
    delete              Delete a container or objects within a container.
    download            Download objects from containers.
    list                Lists the containers for the account or the objects
                        for a container.
    post                Updates meta information for the account, container,
```

```
                      or object; creates containers if not present.
    stat              Displays information for the account, container,
                      or object.
    upload            Uploads files or directories to the given container.
    capabilities       List cluster capabilities.
    tempurl           Create a temporary URL.
    auth              Display auth related environment variables.

Examples:
  swift download --help

  swift -A https://auth.api.rackspacecloud.com/v1.0 -U user -K api_key stat -v

  swift --os-auth-url https://api.example.com/v2.0 --os-tenant-name tenant \
      --os-username user --os-password password list

  swift --os-auth-url https://api.example.com/v3 --auth-version 3\
      --os-project-name project1 --os-project-domain-name domain1 \
      --os-username user --os-user-domain-name domain1 \
      --os-password password list

  swift --os-auth-url https://api.example.com/v3 --auth-version 3\
      --os-project-id 0123456789abcdef0123456789abcdef \
      --os-user-id abcdef0123456789abcdef0123456789 \
      --os-password password list

  swift --os-auth-token 6ee5eb33efad4e45ab46806eac010566 \
      --os-storage-url https://10.1.5.2:8080/v1/AUTH_ced809b6a4baea7aeab61a \
      list

  swift list --lh

Options:
  --version           show program's version number and exit
  -h, --help          show this help message and exit
  --os-help           Show OpenStack authentication options.
  -s, --snet          Use SERVICENET internal network.
```

```
  -v, --verbose         Print more info.
  --debug               Show the curl commands and results of all http queries
                        regardless of result status.
  --info                Show the curl commands and results of all http queries
                        which return an error.
  -q, --quiet           Suppress status output.
  -A AUTH, --auth=AUTH  URL for obtaining an auth token.
  -V AUTH_VERSION, --auth-version=AUTH_VERSION, --os-identity-api-version=AUTH_VERSION
                        Specify a version for authentication. Defaults to
                        env[ST_AUTH_VERSION], env[OS_AUTH_VERSION],
                        env[OS_IDENTITY_API_VERSION] or 1.0.
  -U USER, --user=USER  User name for obtaining an auth token.
  -K KEY, --key=KEY     Key for obtaining an auth token.
  -R RETRIES, --retries=RETRIES
                        The number of times to retry a failed connection.
  --insecure            Allow swiftclient to access servers without having to
                        verify the SSL certificate. Defaults to
                        env[SWIFTCLIENT_INSECURE] (set to 'true' to enable).
  --no-ssl-compression  This option is deprecated and not used anymore. SSL
                        compression should be disabled by default by the
                        system SSL library.
```

swift 를 이용할 때 기본 형식은

~# swift [기본 옵션] subcommand [command_args]

형태이다. 이 때 subcommand 위치에 수행하려는 명령을 지정하는데, 사용량 정보를 보여주는 stat 명령, 내부 컨테이너 목록이나 특정 컨테이너의 파일 목록을 보여주는 list 명령, 파일을 업로드하는 upload 명령, 메타 데이터를 기록하는 post 명령, 파일을 다운로드 하는 download 명령, 파일이나 컨테이너를 을 삭제하는 delete 명령을 제공한다. 각 명령은 부가적으로 세부적인 인자(args) 를 지정할 수 있다.

기본 옵션으로 기타 정보 - 인증을 위한 계정정보, 키 값 등을 지정하게 되고, 버전 정보 상세 출력 등을 지정할 수 있다.

인증 정보를 지정하기 위해서 인증 url정보는

ucloud storage 2.0: https://ssproxy2.ucloudbiz.olleh.com:5000/v3

을 사용하고, 사용자 ID 는 포탈에 등록한 email ID, Key는 스토리지 서비스에서 확인한 API Key를 사용한다.

(참고로, -s 혹은 --snet 옵션은 현재 KT에서 적용되는 값이 아니므로 무시한다.)

나) 기본 사용법

먼저 사용량 정보를 조회하면서 기본 옵션 지정 방법을 확인해 본다.

```
~# swift --os-auth-url http://ssproxy2.ucloudbiz.olleh.com:5000/v3 --auth-version 3 --os-project-name
leeseul --os-project-id 7d1bbbb2fc4b49f2b741f9585ce3f890 --os-username leeseul --os-user-domain-name
default --os-password leeseul stat
                           Account: AUTH_7d1bbbb2fc4b49f2b741f9585ce3f890
                        Containers: 7
                           Objects: 38
                             Bytes: 45943
    Containers in policy "policy-0": 5
       Objects in policy "policy-0": 29
         Bytes in policy "policy-0": 43022
Containers in policy "erasurecoding": 2
   Objects in policy "erasurecoding": 9
     Bytes in policy "erasurecoding": 2921
                     Accept-Ranges: bytes
        X-Account-Project-Domain-Id: 90dca96ea6114242a18eef7beb595dfb
                       X-Timestamp: 1479714696.55854
                        X-Trans-Id: tx9f026d2714444095ac356-00584e5c44
                      Content-Type: text/plain; charset=utf-8
```

위와 유사하게 -U 옵션으로 자신의 ID를 지정하고, -K 옵션으로 자신의 key를 지정한 후에 stat 커맨드를 수행하면 아래와 같은 정보가 출력된다. 위에서 -U 및 -K 옵션으로 지정한 값 들은 이용자가 각자 자신이 사용하는 값들로 바꾸어서 입력해야 한다.
정보 중 Account는 계정에 1:1로 대응되는 고유 식별자이다. Containers는 계정에 존재하는 컨테이너(파일박스)의 총 개수, Objects는 전체 컨테이너들에 포함된 파일들의 개수의 총 합이며, Bytes는 전체 파일들에 대한 크기의 합이다. 처음에 계정을 발급받아서 명령을 수행하면 위와 같이 다 0으로 출력될 것이다. 혹시 이전에 포탈이나 3rd party tool 을 이용해서 파일을 업로드 한 경우 해당 정보가 반영되어 출력된다.

이후에 반복해서 swift 명령을 사용하게 될 텐데 매번 -A, -U, -K 옵션을 지정할 수도 있겠지만 조금 더 간편하게 사용하기 위해 OS_PROJECT_DOMAIN_NAME, OS_USERNAME 등으로로 환경 변수를 지정해 두면 옵션을 입력하는 수고를 덜 수 있다.

리눅스 환경에서는 다음과 같이 환경변수를 지정할 수 있다.

```
export OS_PROJECT_DOMAIN_NAME= <default>
export OS_USER_DOMAIN_NAME= <default>
export OS_PROJECT_NAME= <projectname>
export OS_USERNAME= <username>
export OS_PASSWORD= <password>
export OS_AUTH_URL=http://ssproxy2.ucloudbiz.olleh.com:5000/v3
export OS_IDENTITY_API_VERSION=3
```

* 윈도우 환경에서는 환경 변수를 아래와 같이 지정한다.

```
> set ST_AUTH=https://api.ucloudbiz.olleh.com/storage/v1/auth/ <enter>
> set ST_USER=myemailid@email.com <enter>
> set ST_KEY=MTMyMzMyMTg2MTEzMjMzMTg2NDYzNjUa <enter>
```

다시 언급하지만, 이용자들은 ST_USER 및 ST_KEY 값 들을 자신이 포탈에서 확인하거나 발급 받은 값들로 바꾸어 적용해야 한다.

이후에 swift 명령시, 옵션을 생략하고 다음과 같이 간단하게 사용할 수 있다.

```
~# swift stat
                          Account: AUTH_c88ed112dafe4517a9435425d02234b0
                       Containers: 7
                          Objects: 4665
                            Bytes: 1189060
    Containers in policy "policy-0": 6
       Objects in policy "policy-0": 4664
         Bytes in policy "policy-0": 1187293
Containers in policy "erasurecoding": 1
   Objects in policy "erasurecoding": 1
     Bytes in policy "erasurecoding": 1767
                    Accept-Ranges: bytes
        X-Account-Project-Domain-Id: 90dca96ea6114242a18eef7beb595dfb
                     X-Timestamp: 1477980875.18477
                      X-Trans-Id: tx131dc568ae254690a15ca-00584e5d41
                    Content-Type: text/plain; charset=utf-8
```

설명을 더 진행하기 위해 임시로 몇 개의 파일을 만들어서 무작정 업로드 해 본다.

업로드를 위해 먼저 임의의 파일 및 디렉터리를 생성하려고 하는데 이해를 돕기 위해 디렉터리 구조를 표시하면 다음과 같다.

< 디렉터리 구조 개요 – 입력하는 값은 아님 >

```
testdir/ +
         +- test.file1
         +- test.file2
test.file3
test.file4
```

위와 같은 형태로 디렉터리와 파일을 생성하기 위해 아래와 같이 입력한다. 윈도우 환경에서는 명령이 약간 다르므로 아래를 참고한다.

```
~# mkdir testdir <enter>
~# echo 'this is a test file 1' > testdir/test.file1 <enter>
~# echo 'this is a test file 2' > testdir/test.file2 <enter>
~# echo 'this is a test file 3' > test.file3 <enter>
~# echo 'this is a test file 4' > test.file4 <enter>
~# swift upload testcont1 testdir/* <enter>
testdir/test.file3
testdir/test.file1
testdir/test.file2
testdir/test.file4
~# swift upload testcont1 test.file3 <enter>
test.file3
~# swift upload testcont2 test.file4 <enter>
test.file4
```

* 윈도우 환경에서는 디렉터리 아래 모든 파일을 dir/* 로 사용하면 파이썬 라이브러리에서 오류가 발생하므로 6번째 명령에서 '*'를 넣지 않는다

```
> swift upload testcont1 testdir/ <enter>
```

* 또한 윈도우 환경에서는 경로명을 지정할 때 역슬래시 '\' 문자를 이용한다. 로컬 경로를 지정할 때는 testdir\test.file1 과 같이 역슬래시로 바꾸어 입력한다.

명령을 수행할 때 특별히 에러가 출력되지 않으면 성공한 것이다. 위의 명령을 수행함으로써 먼저 testdir 디렉터리를 생성하고 그 하부에 test.file1, test.file2를 생성하는 한편, 기본 디렉터리에 test.file3,

test.file4 를 생성함으로써 총 4개의 파일을 임시로 생성한다.
이후에 swift를 이용해 testcont1 에 파일 3개를 업로드, testcont2 에 한 개의 파일을 업로드 했다.
swift 툴은 특정 컨테이너로 파일을 업로드시 해당 컨테이너가 존재하지 않으면 내부적으로 먼저 컨테이너를 생성한 후에 업로드 동작을 수행하므로 위의 업로드 과정에서 두 개의 컨테이너 - testcont1, testcont2 가 자동으로 생성된다.

이제 용량을 다시 확인하면 다음과 같다.

```
~# swift stat <enter>
Account: AUTH_0ee03824681f4661a456edade051db3c
                   Containers: 4
                      Objects: 10
                        Bytes: 563
Containers in policy "policy-0": 4
   Objects in policy "policy-0": 12
     Bytes in policy "policy-0": 563
             Meta Temp-Url-Key: mykey
    X-Account-Project-Domain-Id: 90dca96ea6114242a18eef7beb595dfb
                  X-Timestamp: 1481172928.09677
                   X-Trans-Id: tx09a7ad8242a747fabc58b-00584e612f
                 Content-Type: text/plain; charset=utf-8
                Accept-Ranges: bytes
```

위와 같이 컨테이너가 2개 생성되었고, 총 파일(=오브젝트)의 개수가 4개로 출력된다.

실제 파일을 조회하기 위해서는 list 명령을 이용해 컨테이너 목록을 확인하고, 이후에 파일 목록을 조회할 수 있다.

```
~# swift list <enter>
testcon
testcon1
testcont1
testcont2
~# swift list testcont1 <enter>
test.file3
test.files3
testdir/test.file1
```

```
testdir/test.file2
testdir/test.file3
testdir/test.file4
~# swift list testcont2 <enter>
test.file4
```

끝으로, download 명령을 이용해 파일 하나를 다운로드 해 보면 다음과 같다. 파일을 저장하기 위해 -o 옵션으로 저장될 파일 이름을 지정한다.

```
~# swift download testcont2 test.file4 -o out.file <enter>
test.file4 [auth 0.710s, headers 0.856s, total 0.856s, 0.000 MB/s]
~# cat out.file <enter>
this is a test file 4
```

* 윈도우 환경에서는 cat 대신 type 명령을 이용한다.

```
> type out.file <enter>
this is a test file 4
```

지금까지 간단하게 swift 를 이용해 파일 정보 조회, 리스트 조회, 업로드, 다운로드 등을 수행했으며 이후에 각각의 명령을 세부적으로 살펴본다.

4 컨테이너/ 파일 목록 조회 - list 명령

list 명령은 하부의 컨테이너 목록이나 특정 컨테이너에 포함된 파일 목록을 출력한다.

가) 컨테이너 목록 조회

list 명령 뒤에 인자를 추가하지 않으면 기본으로 컨테이너 목록을 출력한다.

```
$ swift list <enter>
testcon
testcon1
testcont1
testcont2
```

나) 오브젝트 목록 조회

list 명령 뒤에 컨테이너 이름을 지정하면 컨테이너에 포함된 파일 목록을 출력한다.

```
$ swift list testcont1 <enter>
test.file3
test.files3
testdir/test.file1
testdir/test.file2
testdir/test.file3
testdir/test.file4
```

파일 목록을 출력해 보면 기존에 업로드한 디렉터리가 파일 이름 앞에 포함되어 저장된 것을 확인할 수 있다. 이처럼 swift 를 이용해서 특정 디렉터리에 포함된 파일을 업로드 할 때 해당 디렉터리 이름을 파일이름에 포함해서 저장하는 방식으로 디렉터리 정보를 포함하게 한다.
부족하나마 이용자가 파일 목록을 디렉터리 형태로 출력하기 위해 추가 옵션을 지정할 수 있다.
먼저 -p (또는 --prefix) 옵션으로 프리픽스 문자열을 지정하면, 해당 프리픽스가 포함된 파일 목록만 출력할 수 있다.
가령 testdir/ 이라는 디렉터리 이름을 프리픽스로 지정해서 파일 목록을 출력하면 다음과 같다.

```
$ swift list testcont1 -p testdir/ <enter>
testdir/test.file1
testdir/test.file2
```

그러나 파일 개수가 많아지면 어떤 디렉터리가 있는지 확인하기도 쉽지 않을 수 있다. 이때는 -d (또는 --delimiter) 옵션으로 특정 문자를 구분자로 지정해 준다. 이 경우에는 파일 목록을 출력할 때 해당 구분자가 나올때 까지 파일 이름을 출력한다. 가령 구분자로 '/' 문자를 지정하면 해당 디렉토에 포함된 파일 및 하위 디렉터리 이름까지만 출력한다.

```
$ swift list testcont1 –d / <enter>
test.file3
testdir/
```

위의 예제에서는 보여지지 않았지만 디렉터리가 중첩 되는 경우에 특정 디렉터리를 기준으로 하위 디렉터리 목록 및 파일 목록을 출력하려면 '-p dir1/dir2/ -d /' 와 같이 옵션을 동시에 지정해 줌으로써 비슷한 효과를 줄 수 있다.

5 정보/사용량 조회 - stat 명령

stat 명령은 전체 어카운트에 대한 사용량 조회 및 특정 컨테이너에 대한 사용량 조회, 특정 파일에 대한 세부정보 등을 출력한다. 이때 조회할 대상을 인자(arguement)로 지정하는데 컨테이너 이름, 파일 이름을 지정할 수 있다.

가) 어카운트 정보 조회

stat 명령에 아무 인자를 주지 않으면 어카운트에 대한 정보를 출력한다. 주의할 점은 파일을 업로드 하거나 삭제한 직후에 어카운트에 대한 정보를 출력하면 최근 추가/삭제된 내용이 바로 반영되지는 않는다. Swift 구현 특성상 최신에 갱신된 내용은 최대 2~3분까지 지연되어서 반영되므로 참고한다. 혹은 어카운트가 아닌 컨테이너 단위로 정보를 조회하면 정확한 정보를 확인할 수 있다.

```
$ swift stat <enter>
Account: AUTH_0ee03824681f4661a456edade051db3c
                    Containers: 4
                       Objects: 16
                         Bytes: 651
Containers in policy "policy-0": 4
   Objects in policy "policy-0": 16
     Bytes in policy "policy-0": 651
              Meta Temp-Url-Key: mykey
    X-Account-Project-Domain-Id: 90dca96ea6114242a18eef7beb595dfb
                  X-Timestamp: 1481172928.09677
                   X-Trans-Id: tx4751d79a7186456d99279-00584e664a
                 Content-Type: text/plain; charset=utf-8
                Accept-Ranges: bytes
```

나) 컨테이너 정보 조회

stat 명령 뒤에 인자로 컨테이너 이름을 지정하면 해당 컨테이너에 대한 정보를 출력한다.

```
$ swift stat testcont1 <enter>
Account: AUTH_0ee03824681f4661a456edade051db3c
       Container: testcont1
         Objects: 6
           Bytes: 132
        Read ACL:
       Write ACL:
         Sync To:
        Sync Key:
```

```
   Accept-Ranges: bytes
X-Storage-Policy: Policy-0
     X-Timestamp: 1481531397.33871
      X-Trans-Id: tx27034e446f004cf599b10-00584e6684
    Content-Type: text/plain; charset=utf-8
```

출력되는 값 중 Read ACL, Write ACL은 (ACL: Access Control List) 해당 컨테이너에 파일을 읽거나 쓸 수 있는 권한을 지정한다. 기본으로는 인증 받은 사용자만 토큰 값을 획득하고 이후 읽기/ 쓰기 요청에 토큰을 포함해서 요청을 보냄으로써 권한을 확인하고 해당 동작이 수행된다. 그러나 가령 불특정 다수가 해당 컨테이너를 읽기를 바란다면 Read ACL에 *:r 과 같이 아무나 읽도록 지정함으로써 인증을 받지 않은 이용자도 직접 url을 이용해 컨테이너 읽기가 가능하다. 이를 확인하기 위해 curl 명령으로 직접 컨테이너 읽기를 시도할 수 있다. 이때 해당 url은 도메인 + '/v1/Account/Container' 형태로 path가 지정된다.

```
$ curl https://ssproxy2.ucloudbiz.olleh.com/v1/AUTH_1497b65d-d50a-4bfa-bc05-6267d5ff6713/testcont1
<enter>
<html>
 <head>
  <title>401 Unauthorized</title>
 </head>
 <body>
  <h1>401 Unauthorized</h1>
  This server could not verify that you are authorized to access the document you requested. Either
you supplied the wrong credentials (e.g., bad password), or your browser does not understand how to
supply the credentials required.<br /><br />


 </body>
</html>
```

이와 같이 접근을 해도 401 에러가 난다. 이후에 뒤에서 post 명령으로 해당 ACL 정보나 meta 정보를 설정함으로써 인증 없이 접근이 가능하도록 하는 사례를 살펴볼 예정이다.

그밖에, Sync To, Sync Key 정보 및 관련 설정은 현재 사용하지 않는다.

다) 파일정보 조회

파일정보 조회는 stat 명령어 뒤에 컨테이너 이름과 파일 이름을 지정해서 수행한다.

```
$ swift stat testcont1 test.file3 <enter>
Account: AUTH_0ee03824681f4661a456edade051db3c
     Container: testcont1
        Object: test.files3
  Content Type: application/octet-stream
Content Length: 22
 Last Modified: Mon, 12 Dec 2016 08:30:04 GMT
          ETag: c263c13099553714aab957f5ea8fbd97
    Meta Mtime: 1481531311.652665
 Accept-Ranges: bytes
   X-Timestamp: 1481531403.90537
    X-Trans-Id: txe5e48f4e3f03457ca0176-00584e66ea
```

파일 정보를 조회하면 파일에 대한 Content Type, 마지막 수정된 - 마지막으로 업로드 된 시간, 파일에 대한 md5 해시값 등이 출력된다. Meta Mtime은 swift 에서 파일을 업로드 할 때 로컬에 저장된 파일의 시간을 메타 정보로 저장한 값이다. 나중에 upload 명령에서 해당 정보를 이용해 새로운 파일인지 비교해서 업로드 할 때 해당 정보를 참조한다.

6 파일 업로드 - upload 명령

파일 업로드를 할 때는 뒤에 인자로 파일이 업로드 될 컨테이너 이름이 지정되고, 그 뒤에 업로드 할 대상 파일(들)을 지정한다. 이때 컨테이너가 없다면 swift 에서 자동으로 만들어 준다.
아래 명령들을 수행하면서 list 및 stat 명령으로 파일이 어떻게 업로드 되었는지 확인해 보기를 바란다.

가) 단일 파일 업로드

하나의 파일을 업로드 할 경우 업로드 할 컨테이너 이름을 지정하고 뒤에 해당 파일 이름만 적어주면 된다. 현재 디렉토리에 testfile1, testfile2 등이 있다고 가정한다.

```
$ swift upload testcont1 testfile1 <enter>
```

나) 여러 파일 업로드

여러 파일을 업로드 할 경우에는 파일 이름을 공백으로 구분해서 여러 개 적어준다.

```
$ swift upload testcont1 testfile1 testfile2 <enter>
```

다) 디렉터리 업로드

디렉터리를 업로드 할 경우 해당 디렉터리 이름을 적어준다. 이 경우에 해당 디렉터리 하위에 있는 모든 파일들을 업로드 한다.

```
$ swift upload testcont1 testdir/ <enter>
```

업로드할 여러개의 파일 또는 디렉터리 이름을 혼용해서 지정할 수 있는데, 이때 주의할 점은 지정한 디렉터리 이름 문자열이 모두 포함된다는 것이다.
가령 'swift upload testcont1 ../workdir/' 이라고 적으면 업로드 되는 파일은 이름이 '../workdir/' 로 시작하게 된다. 업로드 한 뒤에 list 명령으로 파일의 이름이 어떻게 저장되는지 확인하기 바란다.

라) 파일 분할 업로드

swift는 현재 최대 업로드 가능한 파일 크기를 5GB까지 지원한다. 이보다 더 큰 파일을 업로드 하기 위해서는 이용자가 해당 파일을 5GB 이하로 분할한 후 업로드 해야 한다. 그러나 swift에서도 분할(segmentation) 기능을 지원하고 있는데 5GB 이상의 큰 파일을 업로드 할 때 자동으로 서버에서 파일을 분할 저정하는 것은 아니고, 클라이언트에서 파일을 분할해서 업로드 하되 각 파일의 조각들이 이어져 있다는 것을 표시하는 메타정보를 함께 기록한다. 이렇게 업로드 된 파일은 다운로드 할 때 swift 서버에서 해당 메타정보를 기반으로 파일을 이어붙여서 전송해 주므로 이용자는 하나의 파일로 다운로드 할 수 있다.
이러한 분할 업로드 동작은 swift 에서 기능을 제공하므로 이용자는 파일 분할하거나 분할된 파일에 대한 메타정보 관리를 하지 않고도 5GB 이상의 큰 파일을 업로드 할 수 있다.
swift 를 이용해 파일을 분할 업로드 하려면 -S (또는 --segment-size) 로 분할할 크기를 바이트 단위로 지정한다.
테스트를 위해서 10MB 파일을 생성하고 1MB 단위로 업로드 해 본다. 약 10개의 세그먼트(segement) 들로 분할된 후에 각각 업로드 되는 것을 확인할 수 있다.

먼저 업로드 할 10MB 용량의 파일을 임시로 생성한다.

```
$ dd if=/dev/zero of=./10MB.data bs=$((1024*1024)) count=10 <enter>
10+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0.0124315 s, 843 MB/s
```

* 윈도우 환경에서는 dd 명령을 제공하지 않으므로, 크기가 큰 (결과를 비교하기 위해서 10MB 정도의 용량을 가진) 큰 파일을 하나 복사해서 10MB.data 라는 이름으로 준비한다.

```
$ swift upload -S 1048576 testcont1 10MB.data <enter>
10MB.data segment 6
```

```
10MB.data segment 2
10MB.data segment 8
10MB.data segment 3
10MB.data segment 0
10MB.data segment 5
10MB.data segment 1
10MB.data segment 9
10MB.data segment 4
10MB.data segment 7
10MB.data
```

분할 업로드 시에 swift 는 내부적으로 10개의 스레드를 생성해서 세그먼트를 동시에 최대 10개씩 업로드 한다. 이에 따라 위와 같이 결과가 세그먼트 단위로 업로드가 완료된 순서대로 출력된다.

이제 분할된 파일이 어떻게 swift에 저장되었는지 확인해 본다.

```
$ swift list <enter>
testcont1
testcont1_segments
testcont2
```

이와 같이 파일을 testcont1 컨테이너에 업로드 했으나 testcont1_segments 라는 컨테이너가 별도로 생성되어 있다.
일단 업로드 한 파일을 확인해 본다.

```
$ swift stat testcont1 10MB.data <enter>
Account: AUTH_0ee03824681f4661a456edade051db3c
     Container: testcont1
        Object: 10MB.data
  Content Type: application/octet-stream
Content Length: 10485760
 Last Modified: Mon, 12 Dec 2016 09:43:00 GMT
          ETag: "8d431e7531abb83a6cf67e56d91c6f74"
      Manifest: testcont1_segments/10MB.data/1481535747.244921/10485760/1048576/
    Meta Mtime: 1481535747.244921
 Accept-Ranges: bytes
```

```
  X-Timestamp: 1481535779.06657
   X-Trans-Id: tx7dc32ebe227f49899f8a2-00584e7136
```

출력된 정보에서 다른 값들은 이전과 같은데, 분할 업로드 된 파일의 경우에는 Manifest 정보가 추가되어 있다. 해당 값은 이 파일의 실제 내용이 분할된 위치를 나타낸다.
해당 정보를 다시 출력해 보면 다음과 같다.

```
$ swift list testcont1_segments <enter>
10MB.data/1481535747.244921/10485760/1048576/00000000
10MB.data/1481535747.244921/10485760/1048576/00000001
10MB.data/1481535747.244921/10485760/1048576/00000002
10MB.data/1481535747.244921/10485760/1048576/00000003
10MB.data/1481535747.244921/10485760/1048576/00000004
10MB.data/1481535747.244921/10485760/1048576/00000005
10MB.data/1481535747.244921/10485760/1048576/00000006
10MB.data/1481535747.244921/10485760/1048576/00000007
10MB.data/1481535747.244921/10485760/1048576/00000008
10MB.data/1481535747.244921/10485760/1048576/00000009
```

이와 같이 분할된 파일은 업로드 될 때 컨테이너이름_segments 라는 컨테이너가 다시 새로 생성되고, 그 내부에 파일들이 분할 저장되는 것을 확인할 수 있다. 이때 생성되는 파일들의 이름 규칙은 ‘원래_파일이름/mtime/파일size*10/파일size/00000000’ 부터 시작해서 번호가 증가되면서 저장된다.
파일을 다운로드 해서 원래 파일과 비교해 보자. 파일 다운로드는 바로 뒤에서 설명하겠지만 지금은 아래와 같이 간단하게 먼저 이용해 본다.

```
$ swift download testcont1 10MB.data -o 10MB.data.download <enter>
10MB.data [auth 0.714s, headers 1.012s, total 1.184s, 22.295 MB/s]
$ diff 10MB.data 10MB.data.download  <enter>
```

* 윈도우 환경에서는 diff 명령을 제공하지 않으므로 파일 크기가 같은지 확인하는 정도로 넘어간다.

파일을 다운로드 할 때는 원래 파일을 업로드 한 컨테이너와 파일 이름을 적어준다. 그러면 서버 측에서 해당 파일에 대한 메타데이터를 확인해서 조각들을 이어 붙여서 클라이언트 쪽으로 전송해 줌으로써 단일 파일처럼 다운로드가 가능하다.
주의할 점은 이 분할하여 업로드 된 파일의 권한을 공개로 설정하면 다운로드 시에 401 에러가 발생한다. 이때에는 공개 설정한 컨테이너 뿐만 아니라 새로 생성된 컨테이너(컨테이너이름_segments)의 권한도 공개로 설정해야 하며, 추가적으로 이 컨테이너 권한에는 “.rlistings” 까지도 넣어 주어야 한다.

앞에서도 언급했지만 swift 는 최대 파일 크기를 5GB 까지만 지원하므로 그 이상의 파일은 클라이언트에서 조각으로 분할해서 업로드 해야 한다. 이때 얼마의 크기로 자를지, 자른 파일을 어떻게 파일 이름을 붙여서 어느 컨테이너에 저장할지, 파일을 어떻게 다운로드 할 지 등의 규칙을 이용자가 직접 정해서 이용할 수 있다. 그러나 swift CLI를 이용하면 여러 가지 번거로운 과정을 알아서 처리하므로 편하게 이용할 수 있다.
주의할 점은 컨테이너_segments 로 이름 지어진 컨테이너들은 분할된 조각들을 관리하고 있으므로 직접 삭제/ 변경하지 말고 swift CLI를 이용해서 원래 파일에 대한 처리만 해야 한다.
주의할 점은 segmentation을 지원하지 않는 다른 환경 - 포탈이나 cyberduck 툴 등에서는 파일 이름을 보면 용량이 0으로 출력된다. 이런 경우에 해당 툴에서 파일이 segment 형태로 분할 저장되었는지 별도로 확인해야 한다.

7 파일 다운로드 - download 명령

그냥 파일을 다운로드 하면 원본과 겹쳐서 확인하기 어려우므로 downloads 라는 디렉터리를 생성해 두고 그 내부에서 작업한다.

```
$ mkdir downloads <enter>
$ cd downloads <enter>
```

가) 한 개의 파일 다운로드

아래와 같이 다운로드를 할 수 있다.

```
$ swift download testcont1 testdir/test.file1 <enter>
testdir/test.file1 [auth 0.700s, headers 0.780s, total 0.780s, 0.000 MB/s]
$ ls –Ral <enter>
# ls -Ral
.:
total 12
drwxr-xr-x 3 root root 4096 Dec 12 18:46 .
drwxr-xr-x 4 root root 4096 Dec 12 18:46 ..
drwxr-xr-x 2 root root 4096 Dec 12 18:46 testdir

./testdir:
total 12
drwxr-xr-x 2 root root 4096 Dec 12 18:46 .
drwxr-xr-x 3 root root 4096 Dec 12 18:46 ..
-rw-r--r-- 1 root root   22 Dec 12 17:30 test.file1
```

* 윈도우 환경에서는 'ls -Ral' 대신 'dir /s' 명령으로 확인한다.
이와 같이 다운로드 하려는 파일 이름에 '/' 가 붙어 있는 경우에 swift 를 이용해 해당 파일을 다운로드 하면 자동으로 디렉터리를 생성하고 그 아래에 파일을 생성함으로써 원래 디렉터리 구조를 유지하게 해 준다.

나) 다른 이름으로 다운로드

파일을 다운로드 받은 후에 다른 이름으로 변경할 수도 있지만, 다운로드 할 때 -o 옵션으로 저장될 파일 이름을 지정할 수 있다. 이때는 지정된 파일 이름으로 파일이 생성되고, 만일 특정 디렉터리에 저장하고 싶으면 해당 디렉터리까지 포함해서 파일 이름을 지정한다.

```
$ swift download testcont1 testdir/test.file2 -o test.file2 <enter>
testdir/test.file2
$ ls -Ral
total 8
drwxr-xr-x  4 jkyoung  staff  136 12  9 11:38 .
drwxr-xr-x  8 jkyoung  staff  272 12  9 11:33 ..
-rw-r--r--  1 jkyoung  staff   22 12  9 11:38 test.file2
drwxr-xr-x  3 jkyoung  staff  102 12  9 11:35 testdir

./testdir:
total 8
drwxr-xr-x  3 jkyoung  staff  102 12  9 11:35 .
drwxr-xr-x  4 jkyoung  staff  136 12  9 11:38 ..
-rw-r--r--  1 jkyoung  staff   22 12  9 11:04 test.file1
```

다) 다운로드 한 파일을 표준 출력 (stdout) 으로 redirection

파일을 다운로드 해서 다른 filter나 pipe 처리를 하고 싶은 경우에는 -o 옵션 뒤에 '-' 을 적는다.

```
$ swift download testcont2 test.file4 -o -  <enter>
this is a test file 4
$ swift download testcont2 test.file4 -o - | wc  <enter>
       1       6      22
```

* wc 명령은 word count 명령으로 출력 결과가 각각 줄 수, 단어 수, 문자 수 (공백 포함) 에 해당한다. 자세한 정보는 man wc로 확인한다.
* 윈도우 환경에서는 wc 명령이 없으므로 위와 같이 파이프 형태로 wc 명령을 사용할 수 없다. 다른 명령

이 있다면 그것을 대신 사용해 본다.

라) 여러 개의 파일 동시에 다운로드

여러 개의 파일을 동시에 다운로드 하려면 해당 파일 이름을 공백으로 구분해 적어준다.

```
$ swift download testcont1 testdir/test.file2 test.file3 <enter>
test.file3
testdir/test.file2
```

이 경우에 swift CLI 내부에서 여러 개의 스레드로 각각의 파일을 다운로드 받으므로 전체 다운로드 시간이 단축되는 장점이 있다.

마) 특정 컨테이너에 포함된 파일 모두 다운로드

download 명령 뒤에 컨테이너 이름만 적어주고 이후에 아무것도 지정하지 않으면 해당 컨테이너에 포함된 모든 파일들을 디렉터리 구조에 맞게 다운로드 한다. 결과를 확인하기 위해 먼저 현재 디렉터리를 깨끗하게 지운다. (현재 디렉터리를 지우기 전에 미리 지워도 되는지 확인한다. 설명의 절차를 따르면 현재 downloads 라는 디렉터리에 위치해 있다.)

```
$ swift list testcont1 <enter>
10MB.data
test.file3
testdir/test.file1
testdir/test.file2
$ rm -rf * <enter>
$ swift download testcont1
testdir/test.file2
test.file3
testdir/test.file1
10MB.data
$ ls -Ral
total 20488
drwxr-xr-x  5 jkyoung  staff      170 12  9 11:48 .
drwxr-xr-x  8 jkyoung  staff      272 12  9 11:33 ..
-rw-r--r--  1 jkyoung  staff  10485760 12  9 10:33 10MB.data
-rw-r--r--  1 jkyoung  staff       22 12  9 11:04 test.file3
drwxr-xr-x  4 jkyoung  staff      136 12  9 11:48 testdir
./testdir:
```

```
total 16
drwxr-xr-x  4 jkyoung  staff  136 12  9 11:48 .
drwxr-xr-x  5 jkyoung  staff  170 12  9 11:48 ..
-rw-r--r--  1 jkyoung  staff   22 12  9 11:04 test.file1
-rw-r--r--  1 jkyoung  staff   22 12  9 11:04 test.file2
```

위와 같이 testcont1에 포함된 모든 파일들을 다운로드 해서 디렉터리 구조에 맞게 저장한 것을 확인할 수 있다.

* 윈도우에서는 'ls -Ral' 대신 'dir /s' 명령을 사용해서 파일 리스트를 출력해 볼 수 있고, 'rm -rf *' 대신 'rmdir /s .' 명령을 사용해서 현재 디렉터리에 속한 파일들과 하위 디렉터리를 삭제한다. 삭제하기 전에 미리 중요한 파일이 있는지 먼저 확인하고 삭제해야 한다.

바) 특정 디렉터리 이름을 가진 파일 다운로드

참고로, 현재 swift 툴에서는 특정 컨테이너에 저장된 여러 파일 중에서 특정 prefix (가령 특정 디렉터리 이름으로 시작하는 파일)을 다운로드 하는 기능은 아직 제공되지 않으므로 이용자가 별도의 스크립트 등을 이용해서 해당 파일들에 대한 리스트를 먼저 만들고 이 파일들을 다운로드 하는 작업을 직접 수행해야 한다. 여러가지 방법이 있겠으나, 한 가지 예를 들면 다음과 같다.

```
$ dnlist=`swift list testcont1 -p testdir/` && swift download testcont1 $dnlist
testdir/out.file [auth 0.727s, headers 0.782s, total 0.783s, 0.000 MB/s]
testdir/test.file1 [auth 0.739s, headers 0.798s, total 0.798s, 0.000 MB/s]
testdir/test.file4 [auth 0.733s, headers 0.789s, total 0.789s, 0.000 MB/s]
```

* 윈도우 환경에서는 위와 같은 쉘 명령이 수행되지 않는다. 대신에 아래와 같이 임시파일을 이용해 처리해 볼 수 있다.

```
> swift.py list testcont1 -p testdir > list.temp <enter>
> swift.py download testcont1 < list.temp <enter>
```

위와 같이 먼저 list 명령으로 특정 prefix로 시작하는 (위에서는 'testdir/') 파일 리스트를 만들어서 dnlist 변수에 저장한 후에(혹은 임시 파일로 저장한 이후에) 해당 변수 또는 파일을 이용해 download 명령을 수행하면 해당 리스트에 포함된 파일들만 한번에 다운로드 받을 수 있다. 그러나 변수를 사용할 경우에 해당되는 파일의 수가 많아지면 오류가 발생할 수 있으므로 이런 경우에는 파일 리스트를 임시 파일에 저장한 후에 이를 이용하는 것이 좋다.

8 메타정보 관리 - post 명령

지금까지 swift CLI를 이용하면서, 특히 stat 명령을 사용할 때 파일에 관한 정보가 여러 가지 출력되는 것을 볼 수 있었다. 이러한 정보를 메타 정보라고 할 수 있는데, 기본 메타정보는 지금까지 본 것과 같이 파일 크기, 타입, 시간, 분할 정보, ACL 등이 있다. 이외에도 사용자가 직접 자신이 필요한 메타정보를 기록하거나 읽을 수 도 있다.
여러 메타 정보 중에 ACL (Accesss Control List) 와 사용자 정의 메타정보를 설정하고 읽는 방법을 알아본다.

가) ACL(Access Control List) 설정 - 읽기 권한

ACL은 접근 권한 리스트라고 해석되는데, 특정 컨테이너에 대해 읽기/ 쓰기 권한을 지정하는 정보이다. 아직 ucloud storage 2.0에서는 전체공개 권한만 허용하고 있다.

```
$ swift stat testcont1 <enter>
Account: AUTH_0ee03824681f4661a456edade051db3c
      Container: testcont1
        Objects: 16
          Bytes: 278
       Read ACL:
      Write ACL:
        Sync To:
       Sync Key:
   Accept-Ranges: bytes
X-Storage-Policy: Policy-0
     X-Timestamp: 1481531397.33871
      X-Trans-Id: tx28dc6d02d3b149ddb3149-00584e731b
    Content-Type: text/plain; charset=utf-8
```

위와 같이 읽기 권한 리스트 (Read ACL) 및 쓰기 권한 리스트 (Write ACL)에 아무 값이 없다. 이제 해당 컨테이너에 대해 읽기 권한을 공개로 설정해 본다.

```
$ swift post -r'.r:*' testcont1 <enter>
```

이때 읽기 권한은 -r 옵션으로 지정하며 값을 '.r:*' (윈도우에서는 ".r:*")로 지정하면 누구나 읽을 수 있는 공개된 상태로 설정된다.
다시 컨테이너 정보를 조회해 본다.

```
$ swift stat testcont1 <enter>
Account: AUTH_0ee03824681f4661a456edade051db3c
      Container: testcont1
        Objects: 16
          Bytes: 278
       Read ACL: .r:*
      Write ACL:
        Sync To:
       Sync Key:
   Accept-Ranges: bytes
     X-Trans-Id: txe2fcbcda7c29415ebbc4b-00584e7333
X-Storage-Policy: Policy-0
    X-Timestamp: 1481531397.33871
    Content-Type: text/plain; charset=utf-8
```

이제 Read ACL 항목에 지정한 값이 저장된 것을 볼 수 있다.
공개 설정된 컨테이너는 인증 받지 않은 사용자도 읽기가 가능해진다. 이때 접근하는 URL은 https://ssproxy2.ucloudbiz.olleh.com/v1/Account/Container/File 과 같은 형태로 지정된다.

이제 curl 커맨드나 웹 브라우저를 이용해 해당 컨테이너에 포함된 파일을 다운로드 할 수 있다.

```
$ curl https://ssproxy2.ucloudbiz.olleh.com/v1/AUTH_1497b65d-d50a-4bfa-bc05-
6267d5ff6713/testcont1/test.file3 <enter>
this is a test file 3
```

* 윈도우에서는 curl 명령이 제공되지 않으므로, 별도로 설치해서 이용하거나 혹은 아래와 같이 웹 브라우저를 이용해 테스트 해 본다.

혹은 웹 브라우저 창에서 해당 url을 적으면 파일을 다운로드 하거나 일부 파일은 바로 내용이 보인다.
(위의 경우에 파일의 확장자가 file3 이므로 다운로드가 될 것이다. 다운로드 된 파일을 에디터 등을 이용해 열어보면 내용을 확인할 수 있다.)

공개 설정을 해제하려면 post -r' ' (윈도우에서는 post –r" ")과 같이 공백 문자를 적어주면 기존 내용을 덮어서 기록함으로써 공개 설정이 해제된다.

공개 설정 이외에도 읽기 권한을 swift 계정을 가진 다른 이용자에게 부여할 수도 있는데, 현재는 KT ucloud storage 2.0 에서는 해당 기능을 지원하지 않는다. 추후에 지원하게 되면 공지가 될 예정이다. 기

타 도메인 정보로 설정하는 기능도 지원하지 않는다.

자세한 문법이나 권한 설정은 swift 툴의 도움말을 확인하기를 바란다.

9 파일 삭제 - delete 명령

swift에서는 파일을 삭제하면 복구할 수 있는 방법이 없으므로 삭제 명령을 수행할 때는 주의를 기울여야 한다. 파일을 삭제하기 전에 현재까지 swift툴로 업로드 된 파일들을 다시 확인해 본다. 지금까지 내용대로 툴을 이용했으면 아래와 유사한 결과가 나올 것이다.

```
$ swift list <enter>
testcont1
testcont1_segments
testcont2
$ swift list testcont1 <enter>
10MB.data
test.file3
testdir/test.file1
testdir/test.file2
$ swift list testcont2 <enter>
test.file4
```

가) 특정 파일 삭제

```
$ swift delete testcont1 test.file3 <enter>
test.file3
```

나) 여러 파일 삭제

여러 개의 파일을 동시에 삭제하려면 공백으로 구분해서 파일 이름들을 적어준다.

```
$ swift delete testcont1 testdir/test.file1 testdir/test.file2 <enter>
testdir/test.file2
testdir/test.file1
```

다) 컨테이너 삭제

swift에서 컨테이너를 삭제하려면 미리 컨테이너 내부의 모든 파일이 삭제되어 있어야 한다. 그러나 swift CLI를 이용해서 컨테이너를 삭제하면 친절하게 내부의 파일이 있는 경우 이를 모두 삭제하고 컨테이너까

지 삭제해 준다.

```
$ swift delete testcont1 <enter>
testcont1_segments/10MB.data/1323394414.0/10485760/00000001
testcont1_segments/10MB.data/1323394414.0/10485760/00000002
testcont1_segments/10MB.data/1323394414.0/10485760/00000007
testcont1_segments/10MB.data/1323394414.0/10485760/00000003
testcont1_segments/10MB.data/1323394414.0/10485760/00000000
testcont1_segments/10MB.data/1323394414.0/10485760/00000004
testcont1_segments/10MB.data/1323394414.0/10485760/00000008
testcont1_segments/10MB.data/1323394414.0/10485760/00000005
testcont1_segments/10MB.data/1323394414.0/10485760/00000006
testcont1_segments/10MB.data/1323394414.0/10485760/00000009
10MB.data
```

이전에 10MB.data 파일을 testcont1 컨테이너에 업로드 할 때 여러 개의 작은 파일로 분할해서 업로드 했으므로 swift에서는 자동으로 해당 조각들을 먼저 삭제하고 나서 마지막으로 파일을 삭제한다.

참고로 분할되어 업로드 된 파일을 삭제할 때 --leave-segments 옵션을 주면 세그먼트 된 파일들은 삭제하지 않고 메타 정보를 기록한 파일만 삭제하는 기능도 제공한다. 이는 파일의 정보는 삭제하면서 실제 파일의 내용을 남겨두게 되므로 일종의 휴지통처럼 쓸 수도 있으나 권하지는 않는다. 위에서 설명을 생략했으나 파일을 업로드 할 때에도 --leave-segment 옵션을 주면 기존 세그먼트 된 정보들은 그대로 둔 채 새로운 세그먼트를 업로드 해서 파일을 관리하므로 일종의 파일 버전 관리와 유사한 기능을 제공한다. 업로드나 삭제 명령에서 --leave-segment 옵션을 주면 그에 따라 파일이 삭제되지 않고 남아있으므로 저장 공간을 더 사용하게 된다.

라) 계정의 전체 데이터 삭제

delete 명령 뒤에 -a 옵션을 주면, 해당 계정에 존재하는 모든 컨테이너 및 하부의 파일들을 일괄 삭제한다. (segment 파일들도 포함해서 모두 삭제된다.) 삭제된 파일은 복구되지 않는다는 것을 고려해서 해당 명령은 신중하게 사용해야 한다.

```
$ swift delete -a <enter>
```

10 참고사항

1) 기본 swift에 대한 정보는 opentstack 홈페이지에서 얻을 수 있습니다. 또한 KT에서 운영하는

ucloudbiz.olleh.com 포탈에서도 사용자에게 필요한 정보 및 문서들을 찾을 수 있습니다.

2) 본 문서를 포함해 swift 툴의 사용법 등에 대해 변경사항이 발생할 수 있으므로, 이에 대한 내역은 KT 포탈 공지 및 게시판에서 해당 내용을 참고하시길 바랍니다.

* 문서의 내용을 포함해서 설정이나 사용, 기타 문의는 KT 포탈을 통해 문의해 주시길 요청 드립니다.
* 개발이나 패치 등의 작업이 필요하시면 github를 통해 의견 교환이나 패치 제출을 하실 수 있습니다.