Swift 安装指南

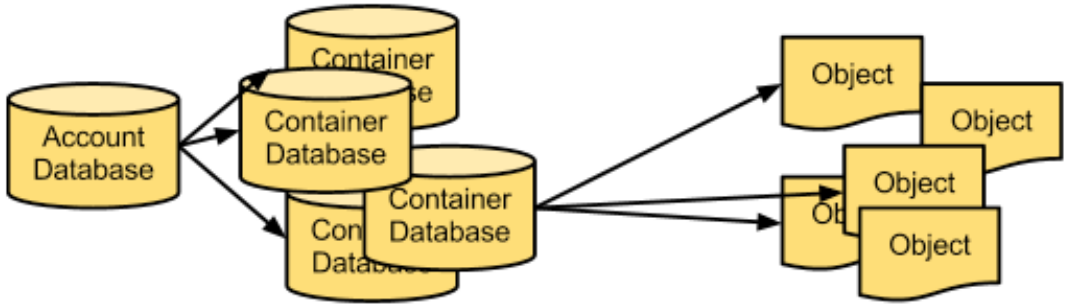
日期	联系邮箱	说明
2013/8/22	sysops@mail2000.us	由于 openstack 项目更新非常快，本安装指南只在当前版本 swift 1.9.1 测试通过，不保证其他版本也可以通过

目 录

1 swift 概述

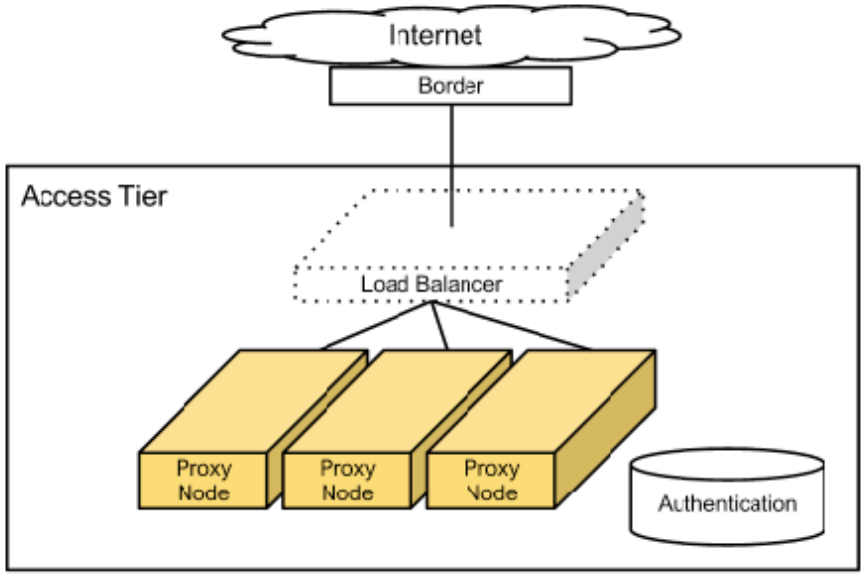
swift 是 openstack 项目的重要组成部分，即分布式文件系统，它采用对象存储（object storage）的方式，每个文件都是一个对象，在 swift 里存储多份（默认 3 份）。swift 包括 2 个组成部分，一个是代理服务（proxy），一个是存储服务（storage）。代理服务理解 swift 内部存储的拓扑逻辑，即一个具体文件位于哪个存储节点的哪个区上。它同时是一个 web 服务器，通过 http 或 https 对外提供 REST API 服务。客户端只与 proxy 服务打交道。

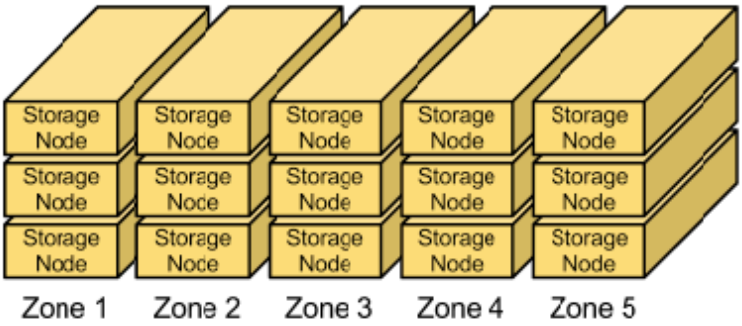
存储服务是负责文件存储的服务，由 3 个组件组成：account server、container server、object server。其中 object server 负责具体的文件存储，container server 包含到每个 object 的索引，account server 包含到每个 container 的索引。所谓 container 可以理解为通常意义上的目录。这 3 者的关系图如下：



swift 的存储集群由多个 zone 构成，所谓 zone 是指物理上独立的资源，比如一台服务器、一个机架、一个机房，都可以成为一个 zone。我们这里采用每台服务器作为一个 zone。在 swift 集群里至少要有 5 个 zone。swift 具体的存储单元叫做分区（partition），分区实际上是文件系统里的子目录，每个分区都包含独立的 account、container、object 服务。数据复制是基于分区的。swift 有 3 个至关重要的 rings 文件，里面保存了系统的 zone、node、device、partitaion 信息，在系统上线之初，要手工创建这 3 个文件。

如下是 swift 架构图：





2 安装环境

本测试安装环境的服务器共 6 台，其中 5 台作为存储节点，1 台作为代理节点。由于是测试环境，这 6 台服务器全部是虚拟机。操作系统是 ubuntu server 12.04。注意当前的 swift 只在 12.04 上测试通过，我测试了 10.04 的安装就有问题。这 6 台服务器的情况如下：

IP 地址	存储设备	运行服务	配置文件(/etc/swift)	服务端口
172.17.6.32	无	proxy	proxy-server.conf	8080
172.17.6.21	/dev/sdb1	account container object	account-server.conf container-server.conf object-server.conf	6002 6001 6000
172.17.6.22	同上	同上	同上	同上
172.17.6.23	同上	同上	同上	同上
172.17.6.24	同上	同上	同上	同上

172.17.6.25	同上	同上	同上	同上

对于存储服务器，要有独立于系统盘的其他磁盘设备，并且使用 XFS 文件系统。

3 安装步骤

3.1 所有服务器上执行

1. 更新系统：apt-get update

2. 安装系统组件：apt-get install curl gcc memcached rsync sqlite3 xfsprogs git-core python-setuptools

3. 安装 python 组件：apt-get install python-coverage python-dev python-nose python-simplejson python-xattr python-eventlet python-greenlet python-pastedeploy python-netifaces python-pip python-dnspython python-mock

4. 安装 swiftclient:
git clone https://github.com/openstack/python-swiftclient.git
cd ~/python-swiftclient; python setup.py install; cd -

5. 安装 swift:
git clone https://github.com/openstack/swift.git
cd ~/swift; python setup.py install; cd -

6. 安装测试依赖：
pip install -r swift/test-requirements.txt

上述 4、5、6 步，可能由于网络原因或其他原因出现失败，会有报错，要留意输出，出错后重新执行一次即可。

7. 增加系统 swift 帐号：
groupadd swift; useradd swift -g swift

8. 创建配置文件目录：
mkdir -p /etc/swift; chown -R swift:swift /etc/swift/

9. 创建 swift 运行目录：
mkdir -p /var/run/swift; chown swift:swift /var/run/swift

因为/var/run/swift 在系统重启后会消失，因此上述 2 句，也需要加入到

/etc/rc.local 里。

3.2 只在 proxy 上执行

1. 生成 swift.conf:

```
cat >/etc/swift/swift.conf <<EOF
[swift-hash]
# random unique strings that can never change (DO NOT LOSE)
swift_hash_path_prefix = `od -t x8 -N 8 -A n </dev/random`
swift_hash_path_suffix = `od -t x8 -N 8 -A n </dev/random`
EOF
```

然后，将生成的/etc/swift/swift.conf 拷贝到每一个 storage 节点的/etc/swift 目录。请注意这个文件至关重要，因为 swift 的一次性哈希算法，就使用这里的随机字符串作为种子。对于已经在运行的 swift 集群，该文件不能再次变更。

2. 在 proxy 上启动 memcached 服务，默认配置文件里的 listen 地址要改一下，可被其他服务器访问到。

3. 创建 proxy-server.conf 配置文件：

```
cat >/etc/swift/proxy-server.conf <<EOF
[DEFAULT]
bind_port = 8080
workers = 8
user = swift

[pipeline:main]
pipeline = healthcheck proxy-logging cache tempauth proxy-logging proxy-server

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

[filter:proxy-logging]
use = egg:swift#proxy_logging

[filter:tempauth]
use = egg:swift#tempauth
user_system_root = testpass .admin http://172.17.6.32:8080/v1/AUTH_system
```

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

[filter:cache]
use = egg:swift#memcache
memcache_servers = 172.17.6.32:11211
EOF

这里也可以指定 proxy 使用 https，但是在产品环境里，不要配置 proxy 使用 SSL，可以在 proxy 前端的负载均衡器（例如 nginx）那里进行 SSL 终结。
workers 参数表示工作进程数，推荐配置是跟 CPU 核心数保持一致。

5. 创建 rings:
cd /etc/swift
swift-ring-builder account.builder create 18 3 1
swift-ring-builder container.builder create 18 3 1
swift-ring-builder object.builder create 18 3 1

这里的参数比较重要，18 代表 2 的 18 次方，创建 262144 个分区（前面讲过，swift 的存储单元是基于分区的）。分区的数量，应该是 swift 集群里预计的最大磁盘数量乘以 100.

3 代表每个文件存储 3 份，请注意只有 3 这个值被严格测试过。最后一个 1 的单位是小时，指分区在 1 个小时内移动不超过 1 次。

6. 执行如下脚本：
#!/bin/sh

cd /etc/swift

swift-ring-builder account.builder add z1-172.17.6.21:6002/sdb1 100
swift-ring-builder container.builder add z1-172.17.6.21:6001/sdb1 100
swift-ring-builder object.builder add z1-172.17.6.21:6000/sdb1 100

swift-ring-builder account.builder add z2-172.17.6.22:6002/sdb1 100
swift-ring-builder container.builder add z2-172.17.6.22:6001/sdb1 100
swift-ring-builder object.builder add z2-172.17.6.22:6000/sdb1 100

swift-ring-builder account.builder add z3-172.17.6.23:6002/sdb1 100
swift-ring-builder container.builder add z3-172.17.6.23:6001/sdb1 100
swift-ring-builder object.builder add z3-172.17.6.23:6000/sdb1 100

swift-ring-builder account.builder add z4-172.17.6.24:6002/sdb1 100
swift-ring-builder container.builder add z4-172.17.6.24:6001/sdb1 100

swift-ring-builder object.builder add z4-172.17.6.24:6000/sdb1 100

swift-ring-builder account.builder add z5-172.17.6.25:6002/sdb1 100
swift-ring-builder container.builder add z5-172.17.6.25:6001/sdb1 100
swift-ring-builder object.builder add z5-172.17.6.25:6000/sdb1 100

这个过程就是往 rings 里增加磁盘，最后一个 100 表示权重，越大越快的磁盘，应该使用越高的权重。

7. 运行如下命令验证配置：
swift-ring-builder account.builder
swift-ring-builder container.builder
swift-ring-builder object.builder

8. 运行 rebalance:
swift-ring-builder account.builder rebalance
swift-ring-builder container.builder rebalance
swift-ring-builder object.builder rebalance

这个过程比较久点。

9. 将上述生成的 account.ring.gz, container.ring.gz, object.ring.gz 拷贝到每一个存储节点的/etc/swift 目录。

10. 更改目录权限：chown -R swift:swift /etc/swift

11. 启动 proxy 服务：swift-init proxy start

如果看到 8080 端口打开，没其他告错，proxy 服务就启动正常。

3.3 只在 storage 上执行

1. 首先对磁盘进行分区，挂载 XFS 文件系统：
   - fdisk /dev/sdb (设置一个单一分区)
   - mkfs.xfs /dev/sdb1
   - 编辑/etc/fstab 增加一行：
   /dev/sdb1 /mnt/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0
   - mkdir /mnt/sdb1
   - mount /mnt/sdb1

2. 创建如下目录：
   mkdir -p /var/cache/swift /srv/node/
   chown swift:swift /var/cache/swift

3. 设置符号链接：
   cd /srv/node/; ln -s /mnt/sdb1/ .
   chown swift:swift /mnt/sdb1

请注意，swift 默认的存储设备位于/srv/node，但是我们把磁盘挂载在/mnt/sdb1，所以要做一个软链接过去。不推荐这种软链接方式，因为 storage 服务会检查磁盘是否挂载，这种软链接会导致系统告错：ERROR Insufficient Storage

为了避免这种错误，需要在所有 storage 服务的配置文件里（包括 account-server.conf、container-server.conf、object-server.conf）加入：
mount_check = false

然后运行：swift-init all restart 重启存储服务。

4. 创建 rsync 配置文件:

```
cat >/etc/rsyncd.conf <<EOF
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 0.0.0.0       # should be changed to private IP in production env

[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
EOF
```

5. 修改/etc/default/rsync，将 RSYNC_ENABLE=false 改成 RSYNC_ENABLE=true，然后执行 service rsync start 启动 rsync.

6. 创建 account server 的配置文件：

```
cat >/etc/swift/account-server.conf <<EOF
[DEFAULT]
bind_ip = 172.17.6.2x
workers = 2

[pipeline:main]
pipeline = account-server

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

[account-replicator]

[account-auditor]

[account-reaper]
EOF
```

将上述及后述的 172.17.6.2x 替换成本机的真实 IP（内网 IP）。

7. 创建 container server 的配置文件：

```
cat >/etc/swift/container-server.conf <<EOF
[DEFAULT]
bind_ip = 172.17.6.2x
workers = 2

[pipeline:main]
pipeline = container-server

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

[container-replicator]

[container-updater]

[container-auditor]

[container-sync]
EOF
```

8. 创建 object server 的配置文件

```
cat >/etc/swift/object-server.conf <<EOF
[DEFAULT]
bind_ip = 172.17.6.2x
workers = 2

[pipeline:main]
pipeline = object-server

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

[object-replicator]

[object-updater]

[object-auditor]
EOF
```

9. 更改目录权限：chown -R swift:swift /etc/swift

10. 启动 storage 服务：swift-init all start
如果启动过程没有报错，并且 6000、6001、6002 端口都打开的话，说明正常。

4 测试

1. 获取 Storage URL 和 Auth Token:
$ curl -k -v -H 'X-Storage-User: system:root' -H 'X-Storage-Pass: testpass' http://172.17.6.32:8080/auth/v1.0

返回的 header:
```
< HTTP/1.1 200 OK
< X-Storage-Url: http://172.17.6.32:8080/v1/AUTH_system
< X-Auth-Token: AUTH_tk02863583400d4141a522fd185432a5f3
< Content-Type: text/html; charset=UTF-8
< X-Storage-Token: AUTH_tk02863583400d4141a522fd185432a5f3
< Content-Length: 0
< Date: Thu, 22 Aug 2013 08:04:10 GMT
```

里面包含了 X-Storage-Url 和 X-Auth-Token 两个参数。

2. 根据上述返回的参数执行 HEAD 命令：
$ curl -k -v -H 'X-Auth-Token: AUTH_tk02863583400d4141a522fd185432a5f3' http://172.17.6.32:8080/v1/AUTH_system

3. 执行 stat 命令：

```
$ swift -A http://172.17.6.32:8080/auth/v1.0 -U system:root -K testpass stat
     Account: AUTH_system
Containers: 1
     Objects: 0
        Bytes: 0
Accept-Ranges: bytes
X-Timestamp: 1377152716.51055
Content-Type: text/plain; charset=utf-8
```

4. 上传 2 个文件到 myfiles 这个 container:

```
$ swift -A http://172.17.6.32:8080/auth/v1.0 -U system:root -K testpass upload myfiles swift.tgz

$ swift -A http://172.17.6.32:8080/auth/v1.0 -U system:root -K testpass upload myfiles python-swiftclient.tgz
```

5. 下载这个 container 里的所有文件：

```
$ swift -A http://172.17.6.32:8080/auth/v1.0 -U system:root -K testpass download myfiles
```

5 参考

swift 官方文档：
http://swift.openstack.org

部署指南（必看）：
http://docs.openstack.org/developer/swift/deployment_guide.html

单机环境部署：
http://docs.openstack.org/developer/swift/development_saio.html

多机环境部署：
http://docs.openstack.org/developer/swift/howto_installmultinode.html

swift 架构：
http://swiftstack.com/openstack-swift/architecture/