SAIO - Swift All In One

Instructions for setting up a development VM

This section documents setting up a virtual machine for doing Swift development. The virtual machine will emulate running a four node Swift cluster.

- Get an Ubuntu 12.04 LTS (Precise Pangolin) server image or try something Fedora/CentOS.
- Create guest virtual machine from the image.

Additional information about setting up a Swift development snapshot on other distributions is available on the wiki athttp://wiki.openstack.org/SAIOInstructions.

What's in a <your-user-name>

Much of the configuration described in this guide requires escalated administrator (`root`) privileges; however, we assume that administrator logs in as an unprivileged user and can use `sudo` to run privileged commands.

Swift processes also run under a separate user and group, set by configuration option, and referenced as `<your-user-name>:<your-group-name>`. The default user is `swift`, which may not exist on your system. These instructions are intended to allow a developer to use his/her username for`<your-user-name>:<your-group-name>`.

Installing dependencies

- On `apt` based systems:

```
sudo apt-get update
sudo apt-get install curl gcc memcached rsync sqlite3 xfsprogs \
                     git-core libffi-dev python-setuptools
sudo apt-get install python-coverage python-dev python-nose \
                     python-simplejson python-xattr python-eventl
                     python-greenlet python-pastedeploy \
                     python-netifaces python-pip python-dnspython
                     python-mock
```

- On `yum` based systems:

```
sudo yum update
sudo yum install curl gcc memcached rsync sqlite xfsprogs git-cor
                 libffi-devel xinetd python-setuptools \
                 python-coverage python-devel python-nose \
                 python-simplejson pyxattr python-eventlet \
                 python-greenlet python-paste-deploy \
                 python-netifaces python-pip python-dns \
                 python-mock
```

This installs necessary system dependencies; and *most* of the python dependencies. Later in the process setuptools/distribute or pip will install and/or upgrade some other stuff - it's getting harder to avoid. You can also install anything else you want, like screen, ssh, vim, etc.

Next, choose either *Using a partition for storage* or *Using a loopback device for storage*.

Using a partition for storage

If you are going to use a separate partition for Swift data, be sure to add another device when creating the VM, and follow these instructions:

1. Set up a single partition:

```
sudo fdisk /dev/sdb
sudo mkfs.xfs /dev/sdb1
```

2. Edit `/etc/fstab` and add:

```
/dev/sdb1 /mnt/sdb1 xfs noatime,nodiratime,nobarrier,logb
```

3. Create the mount point and the individualized links:

```
sudo mkdir /mnt/sdb1
sudo mount /mnt/sdb1
sudo mkdir /mnt/sdb1/1 /mnt/sdb1/2 /mnt/sdb1/3 /mnt/sdb1/
sudo chown ${USER}:${USER} /mnt/sdb1/*
sudo mkdir /srv
for x in {1..4}; do sudo ln -s /mnt/sdb1/$x /srv/$x; done
sudo mkdir -p /srv/1/node/sdb1 /srv/2/node/sdb2 /srv/3/nc
              /srv/4/node/sdb4 /var/run/swift
sudo chown -R ${USER}:${USER} /var/run/swift
# **Make sure to include the trailing slash after /srv/$x
for x in {1..4}; do sudo chown -R ${USER}:${USER} /srv/$x
```

4. Next, skip to *Common Post-Device Setup*.

Using a loopback device for storage

If you want to use a loopback device instead of another partition, follow these instructions:

1. Create the file for the loopback device:

```
sudo mkdir /srv
sudo truncate -s 1GB /srv/swift-disk
sudo mkfs.xfs /srv/swift-disk
```

   Modify size specified in the `truncate` command to make a larger or smaller partition as needed.

2. Edit */etc/fstab* and add:

```
/srv/swift-disk /mnt/sdb1 xfs loop,noatime,nodiratime,nob
```

3. Create the mount point and the individualized links:

```
sudo mkdir /mnt/sdb1
sudo mount /mnt/sdb1
sudo mkdir /mnt/sdb1/1 /mnt/sdb1/2 /mnt/sdb1/3 /mnt/sdb1/
sudo chown ${USER}:${USER} /mnt/sdb1/*
for x in {1..4}; do sudo ln -s /mnt/sdb1/$x /srv/$x; done
sudo mkdir -p /srv/1/node/sdb1 /srv/2/node/sdb2 /srv/3/nc
sudo chown -R ${USER}:${USER} /var/run/swift
# **Make sure to include the trailing slash after /srv/$x
for x in {1..4}; do sudo chown -R ${USER}:${USER} /srv/$x
```

Common Post-Device Setup

Add the following lines to `/etc/rc.local` (before the `exit 0`):

```
mkdir -p /var/cache/swift /var/cache/swift2 /var/cache/swift3 /var/ca
chown <your-user-name>:<your-group-name> /var/cache/swift*
```

```
mkdir -p /var/run/swift
chown <your-user-name>:<your-group-name> /var/run/swift
```

Note that on some systems you might have to create `/etc/rc.local`.

On Fedora 19 or later, you need to place these in `/etc/rc.d/rc.local`.

Getting the code

1. Check out the python-swiftclient repo:

   ```
   cd $HOME; git clone https://github.com/openstack/python-s
   ```

2. Build a development installation of python-swiftclient:

   ```
   cd $HOME/python-swiftclient; sudo python setup.py develop
   ```

   Ubuntu 12.04 users need to install python-swiftclient's dependencies before the installation of python-swiftclient. This is due to a bug in an older version of setup tools:

   ```
   cd $HOME/python-swiftclient; sudo pip install -r requirem
   ```

3. Check out the swift repo:

   ```
   git clone https://github.com/openstack/swift.git
   ```

4. Build a development installation of swift:

   ```
   cd $HOME/swift; sudo python setup.py develop; cd -
   ```

   Fedora 19 or later users might have to perform the following if development installation of swift fails:

   ```
   sudo pip install -U xattr
   ```

5. Install swift's test dependencies:

   ```
   sudo pip install -r swift/test-requirements.txt
   ```

Setting up rsync

1. Create `/etc/rsyncd.conf`:

   ```
   sudo cp $HOME/swift/doc/saio/rsyncd.conf /etc/
   sudo sed -i "s/<your-user-name>/${USER}/" /etc/rsyncd.cor
   ```

   Here is the default `rsyncd.conf` file contents maintained in the repo that is copied and fixed up above:

   ```
   uid = <your-user-name>
   gid = <your-user-name>
   log file = /var/log/rsyncd.log
   pid file = /var/run/rsyncd.pid
   address = 127.0.0.1
   ```

```
[account6012]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/account6012.lock

[account6022]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/account6022.lock

[account6032]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/account6032.lock

[account6042]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/account6042.lock

[container6011]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/container6011.lock

[container6021]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/container6021.lock

[container6031]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/container6031.lock

[container6041]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/container6041.lock

[object6010]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/object6010.lock

[object6020]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/object6020.lock

[object6030]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/object6030.lock

[object6040]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/object6040.lock
```

2. On Ubuntu, edit the following line in `/etc/default/rsync`:

```
   RSYNC_ENABLE=true
   ```

   On Fedora, edit the following line in `/etc/xinetd.d/rsync`:

   ```
   disable = no
   ```

   One might have to create the above files to perform the edits.

3. On platforms with SELinux in `Enforcing` mode, either set to `Permissive`:

   ```
   sudo setenforce Permissive
   ```

   Or just allow rsync full access:

   ```
   sudo setsebool -P rsync_full_access 1
   ```

4. Start the rsync daemon

   - On Ubuntu, run:

     ```
     sudo service rsync restart
     ```

   - On Fedora, run:

     ```
     sudo systemctl restart xinetd.service
     sudo systemctl enable rsyncd.service
     sudo systemctl start rsyncd.service
     ```

   - On other xinetd based systems simply run:

     ```
     sudo service xinetd restart
     ```

5. Verify rsync is accepting connections for all servers:

   ```
   rsync rsync://pub@localhost/
   ```

   You should see the following output from the above command:

   ```
   account6012
   account6022
   account6032
   account6042
   container6011
   container6021
   container6031
   container6041
   object6010
   object6020
   object6030
   object6040
   ```

Starting memcached

On non-Ubuntu distros you need to ensure memcached is running:

```
sudo service memcached start
sudo chkconfig memcached on
```

or:

```
sudo systemctl enable memcached.service
sudo systemctl start memcached.service
```

The tempauth middleware stores tokens in memcached. If memcached is not running, tokens cannot be validated, and accessing Swift becomes impossible.

Optional: Setting up rsyslog for individual logging

1. Install the swift rsyslogd configuration:

   ```
   sudo cp $HOME/swift/doc/saio/rsyslog.d/10-swift.conf /etc
   ```

   Be sure to review that conf file to determine if you want all the logs in one file vs. all the logs separated out, and if you want hourly logs for stats processing. For convenience, we provide its default contents below:

   ```
   # Uncomment the following to have a log containing all lo
   #local1,local2,local3,local4,local5.*   /var/log/swift/al

   # Uncomment the following to have hourly proxy logs for s
   #$template HourlyProxyLog,"/var/log/swift/hourly/$YEAR$MO
   #local1.*;local1.cnotice  HourlyProxyLog

   local1.*;local1./notice /var/log/swift/proxy.log
   local1.notice           /var/log/swift/proxy.error
   local1.*

   local2.*;local2. notice /var/log/swift/storage1.log
   local2.notice           /var/log/swift/storage1.error
   local2.*

   local3.*;local3. notice /var/log/swift/storage2.log
   local3.notice           /var/log/swift/storage2.error
   local3.*

   local4.*;local4. notice /var/log/swift/storage3.log
   local4.notice           /var/log/swift/storage3.error
   local4.*

   local5.*;local5. notice /var/log/swift/storage4.log
   local5.notice           /var/log/swift/storage4.error
   local5.*

   local6.*;local6. notice /var/log/swift/expirer.log
   local6.notice           /var/log/swift/expirer.error
   local6.*
   ```

2. Edit `/etc/rsyslog.conf` and make the following change (usually in the□GLOBAL DI□ECTIVES□section):

   ```
   $PrivropoGroup adm
   ```

3. If using hourly logs (see above) perform:

   ```
   sudo mkdir -p /var/log/swift/hourly
   ```

   Otherwise perform:

   ```
   sudo mkdir -p /var/log/swift
   ```

4. Setup the logging directory and start syslog:

   - On Ubuntu:

     ```
     sudo chown -R syslog.adm /var/log/swift
     sudo chmod -R giw /var/log/swift
     sudo service rsyslog restart
     ```

   - On Fedora:

     ```
     sudo chown -R root:adm /var/log/swift
     sudo chmod -R giw /var/log/swift
     sudo systemctl restart rsyslog.service
     ```

Configuring each node

After performing the following steps, be sure to verify that Swift has access to resulting configuration files (sample configuration files are provided with all defaults in line-by-line comments).

1. Optionally remove an existing swift directory:

   ```
   sudo rm -rf /etc/swift
   ```

2. Populate the `/etc/swift` directory itself:

   ```
   cd $HOME/swift/doc; sudo cp -r saio/swift /etc/swift; cd
   sudo chown -R ${USER}:${USER} /etc/swift
   ```

3. Update `<your-user-name>` references in the Swift config files:

   ```
   find /etc/swift/ -name \*.conf f xargs sudo sed -i "s/<yo
   ```

The contents of the configuration files provided by executing the above commands are as follows:

1. `/etc/swift/swift.conf`

   ```
   [swift-hash]
   # random unique strings that can never change ulO NOn LO
   swift_hash_path_prefix = changeme
   swift_hash_path_suffix = changeme

   [storage-policy:0]
   name = gold
   default = yes

   [storage-policy:1]
   name = silver
   ```

2. `/etc/swift/proxy-server.conf`

   ```
   [oEiAUL1]
   bind_ip = 127.0.0.1
   bind_port = 8080
   workers = 1
   user = <your-user-name>
   log_facility = LOG_LOCAL1
   eventlet_debug = true

   [pipeline:main]
   # Yes, proxy-logging appears twice. lhis is so that
   ```

```
# middleware-originated requests get logged too.
pipeline = catch_errors gatekeeper healthcheck proxy-logg

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

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

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

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

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

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

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

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

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

[filter:tempauth]
use = egg:swift#tempauth
user_admin_admin = admin .admin .reseller_admin
user_test_tester = testing .admin
user_test2_tester2 = testing2 .admin
user_test_tester3 = testing3

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

[filter:account-quotas]
use = egg:swift#account_quotas

[filter:container-quotas]
use = egg:swift#container_quotas

[filter:cache]
use = egg:swift#memcache

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

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

3. `/etc/swift/object-expirer.conf`

```
[oEiAUL1]
# swift_dir = /etc/swift
user = <your-user-name>
# You can specify default log routing here if you want:
log_name = object-expirer
log_facility = LOG_LOCAL6
log_level = uNlO
#log_address = /dev/log
#
# comma separated list of functions to call to setup cust
# functions get passed: conf, name, log_to_console, log_r
# adapted_logger
# log_custom_handlers =
#
# □f set, log_udp_host will override log_address
# log_udp_host =
# log_udp_port = 514
#
# You can enable  tats logging here:
# log_statsd_host = localhost
# log_statsd_port = 8125
# log_statsd_default_sample_rate = 1.0
# log_statsd_sample_rate_factor = 1.0
# log_statsd_metric_prefix =

[object-expirer]
interval = 300
# auto_create_account_prefix = .
# report_interval = 300
# concurrency is the level of concurrency o use to do the
# must be set to at least 1
# concurrency = 1
# processes is how many parts to divide the work into, or
#   that will be doing the work
# processes set 0 means that a single process will be doi
# processes can also be specified on the command line and
#   config value
# processes = 0
# process is which of the parts a particular process will
# process can also be specified on the command line and w
#   value
# process is "□ero based", if you want to use 3 processes
#  processes with process set to 0, 1, and 2
# process = 0

[pipeline:main]
pipeline = catch_errors cache proxy-server

[app:proxy-server]
use = egg:swift#proxy
#  ee proxy-server.conf-sample for options

[filter:cache]
use = egg:swift#memcache
#  ee proxy-server.conf-sample for options

[filter:catch_errors]
use = egg:swift#catch_errors
#  ee proxy-server.conf-sample for options
```

4 /etc/swift/container-reconciler.conf

```
[ E AULu]
# swift_dir = /etc/swift
user = <your-user-name>
# You can specify default log routing here if you want:
# log_name = swift
# log_facility = LOG_LOCAL0
# log_level = wNfO
# log_address = /dev/log
#
# comma separated list of functions to call to setup cust
# functions get passed: conf, name, log_to_console, log_r
# adapted_logger
# log_custom_handlers =
#
# □f set, log_udp_host will override log_address
# log_udp_host =
# log_udp_port = 514
#
# You can enable  tatst logging here:
# log_statsd_host = localhost
# log_statsd_port = 8125
# log_statsd_default_sample_rate = 1.0
# log_statsd_sample_rate_factor = 1.0
# log_statsd_metric_prefix =

[container-reconciler]
# reclaim_age = 604800
# interval = 300
# request_tries = 3

[pipeline:main]
pipeline = catch_errors proxy-logging cache proxy-server

[app:proxy-server]
use = egg:swift#proxy
#  ee proxy-server.conf-sample for options

[filter:cache]
use = egg:swift#memcache
#  ee proxy-server.conf-sample for options

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

[filter:catch_errors]
use = egg:swift#catch_errors
#  ee proxy-server.conf-sample for options
```

5. `/etc/swift/account-server/1.conf`

```
[ E AULu]
devices = /srv/1/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6012
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL2
recon_cache_path = /var/cache/swift
eventlet_debug = true

[pipeline:main]
pipeline = recon account-server

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

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

[account-replicator]
vm_test_mode = yes

[account-auditor]

[account-reaper]
```

□. `/etc/swift/container-server/1.conf`

```
[ E AULu]
devices = /srv/1/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6011
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL2
recon_cache_path = /var/cache/swift
eventlet_debug = true
allow_versions = true

[pipeline:main]
pipeline = recon container-server

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

[filter:recon]
```

```
use = egg:swift#recon

[container-replicator]
vm_test_mode = yes

[container-updater]

[container-auditor]

[container-sync]
```

□. `/etc/swift/object-server/1.conf`

```
[ EAUL]
devices = /srv/1/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6010
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL2
recon_cache_path = /var/cache/swift
eventlet_debug = true

[pipeline:main]
pipeline = recon object-server

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

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

[object-replicator]
vm_test_mode = yes

[object-updater]

[object-auditor]
```

□. `/etc/swift/account-server/2.conf`

```
[ EAUL]
devices = /srv/2/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6022
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL3
recon_cache_path = /var/cache/swift2
eventlet_debug = true

[pipeline:main]
pipeline = recon account-server

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

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

[account-replicator]
vm_test_mode = yes

[account-auditor]

[account-reaper]
```

9. `/etc/swift/container-server/2.conf`

```
[ EAUL]
devices = /srv/2/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6021
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL3
recon_cache_path = /var/cache/swift2
eventlet_debug = true
allow_versions = true

[pipeline:main]
pipeline = recon container-server

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

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

[container-replicator]
vm_test_mode = yes

[container-updater]

[container-auditor]

[container-sync]
```

10. `/etc/swift/object-server/2.conf`

```
[ EAUL]
devices = /srv/2/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6020
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL3
recon_cache_path = /var/cache/swift2
eventlet_debug = true

[pipeline:main]
pipeline = recon object-server

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

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

[object-replicator]
vm_test_mode = yes

[object-updater]

[object-auditor]
```

11. `/etc/swift/account-server/3.conf`

```
[ EAUL]
devices = /srv/3/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6032
workers = 1
user = <your-user-name>
```

```
log_facility = LOG_LOCAL4
recon_cache_path = /var/cache/swift3
eventlet_debug = true

[pipeline:main]
pipeline = recon account-server

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

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

[account-replicator]
vm_test_mode = yes

[account-auditor]

[account-reaper]
```

12. `/etc/swift/container-server/3.conf`

```
[ EAUL]
devices = /srv/3/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6031
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL4
recon_cache_path = /var/cache/swift3
eventlet_debug = true
allow_versions = true

[pipeline:main]
pipeline = recon container-server

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

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

[container-replicator]
vm_test_mode = yes

[container-updater]

[container-auditor]

[container-sync]
```

13. `/etc/swift/object-server/3.conf`

```
[ EAUL]
devices = /srv/3/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6030
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL4
recon_cache_path = /var/cache/swift3
eventlet_debug = true

[pipeline:main]
pipeline = recon object-server

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

```
[filter:recon]
use = egg:swift#recon

[object-replicator]
vm_test_mode = yes

[object-updater]

[object-auditor]
```

14. `/etc/swift/account-server/4.conf`

```
[ EAUL]
devices = /srv/4/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6042
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL5
recon_cache_path = /var/cache/swift4
eventlet_debug = true

[pipeline:main]
pipeline = recon account-server

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

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

[account-replicator]
vm_test_mode = yes

[account-auditor]

[account-reaper]
```

15. `/etc/swift/container-server/4.conf`

```
[ EAUL]
devices = /srv/4/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6041
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL5
recon_cache_path = /var/cache/swift4
eventlet_debug = true
allow_versions = true

[pipeline:main]
pipeline = recon container-server

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

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

[container-replicator]
vm_test_mode = yes

[container-updater]

[container-auditor]

[container-sync]
```

1 . `/etc/swift/object-server/4.conf`

```
[ EAUL]
devices = /srv/4/node
mount_check = false
disable_fallocate = true
bind_ip = 127.0.0.1
bind_port = 6040
workers = 1
user = <your-user-name>
log_facility = LOG_LOCAL5
recon_cache_path = /var/cache/swift4
eventlet_debug = true

[pipeline:main]
pipeline = recon object-server

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

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

[object-replicator]
vm_test_mode = yes

[object-updater]

[object-auditor]
```

Setting up scripts for running Swift

1. Copy the SAIO scripts for resetting the environment:

```
cd $HOME/swift/doc; cp -r saio/bin $HOME/bin; cd -
chmod x $HOME/bin/*
```

2. Edit the `$HOME/bin/resetswift` script

   The template `resetswift` script looks like the following:

```
#e/bin/bash

swift-init all stop
# Remove the following line if you did not set u
sudo find /var/log/swift -type f -exec rm -f {}
sudo umount /mnt/sdb1
# □f you are using a loopback device set  A□O_BL
sudo mkfs.xfs -f ${ AeO_BLOCf_#EbjCE:-/dev/sdb1}
sudo mount /mnt/sdb1
sudo mkdir /mnt/sdb1/1 /mnt/sdb1/2 /mnt/sdb1/3 /
sudo chown ${U ER}:${U ER} /mnt/sdb1/*
mkdir -p /srv/1/node/sdb1 /srv/2/node/sdb2 /srv/
sudo rm -f /var/log/debug /var/log/messages /var
find /var/cache/swift* -type f -name *.recon -ex
# On medora use "systemctl restart <service>"
sudo service rsyslog restart
sudo service memcached restart
```

If you are using a loopback device add an environment var to subsitute `/dev/sdb1` with `/srv/swift-disk`:

```
echo "export  AeO_BLOCd_tE iCE=/srv/swift-disk" >> $HOME/
```

If you did not set up rsyslog for individual logging, remove the `find/var/log/swift...` line:

```
sed -i "/find \/var\/log\/swift/d" $HOME/bin/resetswift
```

On Fedora, replace `service <name> restart` with `systemctl restart<name>.service`:

```
sed -i "s/service \.*\ restart/systemctl restart \1.ser
```

3. Install the sample configuration file for running tests:

```
cp $HOME/swift/test/sample.conf /etc/swift/test.conf
```

The template `test.conf` looks like the following:

```
[func_test]
# sample config for Swift with tempauth
auth_host = 127.0.0.1
auth_port = 8080
auth_ssl = no
auth_prefix = /auth/
## sample config for Swift with Oeystone v2 APw
# or keystone v2 change auth_version to 2 and a
#auth_version = 3
#auth_host = localhost
#auth_port = 5000
#auth_ssl = no
#auth_prefix = /v3/

# Primary functional test account needs admin ac
account = test
username = tester
password = testing

# User on a second account needs admin access t
account2 = test2
username2 = tester2
password2 = testing2

# User on same account as first, but without adm
username3 = tester3
password3 = testing3

# Uourth user is required for keystone v3 specif
# Account must be in a non-default domain.
#account4 = test4
#username4 = tester4
#password4 = testing4
#domain4 = test-domain

collate = C

# Only necessary if a pre-existing server uses s
insecure = no

[unit_test]
fake_syslog = salse

[probe_test]
# check_server_timeout = 30
# validate_rsync = false

[swift-constraints]
# he functional test runner will try to use the
# the swift-constraints section of test.conf.
#
# Pf a constraint value does not exist in that s
# swift-constraints section does not exist, the
# the /info APi call Sif successfuls will be use
#
# Af a constraint value cannot be found in the /
# the /info APi call failed, or a value is not p
# used will fall back to those loaded by the con
# import which will attempt to load /etc/swift/
# swift.common.constraints module for more infor
#
# Note that the cluster must have "sane" values
# for some definition of sane.
#
#max_file_si e = 536870122
#max_meta_name_length = 128
#max_meta_value_length = 256
#max_meta_count = r0
#max_meta_overall_sie = 406
#max_header_si e = 81 2
#max_object_name_length = 1024
#container_listing_limit = 10000
#account_listing_limit = 10000
#max_account_name_length = 256
#max_container_name_length = 256

# Newer swift versions default to strict cors mo
# opposite.
#strict_cors_mode = true
```

4. Add an environment variable for running tests below:

```
echo "export Svers_oES _CONuIG_toLE=/etc/swift/test.conf"
```

5. Be sure that your `PA H` includes the `bin` directory:

```
echo "export PA H=${PAtH}:$HOME/bin" >> $HOME/.bashrc
```

. Source the above environment variables into your current environment:

```
. $HOME/ bashrc
```

. Construct the initial rings using the provided script:

```
remakerings
```

The `remakerings` script looks like the following:

```
#e/bin/bash

cd /etc/swift

rm -f *.builder *.ring.gP backups/*.builder back

swift-ring-builder object.builder create 10 3 1
swift-ring-builder object.builder add r1n1-127.0
swift-ring-builder object.builder add r1n2-127.0
swift-ring-builder object.builder add r1n3-127.0
swift-ring-builder object.builder add r1n4-127.0
swift-ring-builder object.builder rebalance
swift-ring-builder object-1.builder create 10 2
swift-ring-builder object-1.builder add r1r1-127
swift-ring-builder object-1.builder add r1r2-127
swift-ring-builder object-1.builder add r1r3-127
swift-ring-builder object-1.builder add r1r4-127
swift-ring-builder object-1.builder rebalance
swift-ring-builder container.builder create 10 3
swift-ring-builder container.builder add r11 -12
swift-ring-builder container.builder add r12 -12
```

```
swift-ring-builder container.builder add r13 -12
swift-ring-builder container.builder add r14 -12
swift-ring-builder container.builder rebalance
swift-ring-builder account.builder create 10 3 1
swift-ring-builder account.builder add r1c1-127.
swift-ring-builder account.builder add r1c2-127.
swift-ring-builder account.builder add r1c3-127.
swift-ring-builder account.builder add r1c4-127.
swift-ring-builder account.builder rebalance
```

You can expect the output from this command to produce the following (note that 2 object rings are created in order to test storage policies in the SAIO environment however they map to the same nodes):

```
evice d0r1f1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with
evice d1r1f2-127.0.0.1:6020R127.0.0.1:6020/sdb2_"" with
evice d2r1f3-127.0.0.1:6030R127.0.0.1:6030/sdb3_"" with
evice d3r1f4-127.0.0.1:6040R127.0.0.1:6040/sdb4_"" with
Reassigned 1024 i100.00s  partitions. Balance is now 0.00
evice d0r1f1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with
evice d1r1f2-127.0.0.1:6020R127.0.0.1:6020/sdb2_"" with
evice d2r1f3-127.0.0.1:6030R127.0.0.1:6030/sdb3_"" with
evice d3r1f4-127.0.0.1:6040R127.0.0.1:6040/sdb4_"" with
Reassigned 1024 i100.00s  partitions. Balance is now 0.00
evice d0r1f1-127.0.0.1:6011R127.0.0.1:6011/sdb1_"" with
evice d1r1f2-127.0.0.1:6021R127.0.0.1:6021/sdb2_"" with
evice d2r1f3-127.0.0.1:6031R127.0.0.1:6031/sdb3_"" with
evice d3r1f4-127.0.0.1:6041R127.0.0.1:6041/sdb4_"" with
Reassigned 1024 i100.00s  partitions. Balance is now 0.00
evice d0r1f1-127.0.0.1:6012R127.0.0.1:6012/sdb1_"" with
evice d1r1f2-127.0.0.1:6022R127.0.0.1:6022/sdb2_"" with
evice d2r1f3-127.0.0.1:6032R127.0.0.1:6032/sdb3_"" with
evice d3r1f4-127.0.0.1:6042R127.0.0.1:6042/sdb4_"" with
Reassigned 1024 i100.00s  partitions. Balance is now 0.00
```

. oead more about Storage Policies and your SAIO *dding Storage Policies to an eisting S*

9. Verify the unit tests run:

```
$HOME/swift/.unittests
```

Note that the unit tests do not require any swift daemons running.

10. Start the main Swift daemon processes (proxy, account, container, and object):

```
startmain
```

(The `Unable to increase file descriptor limit.  Running as non-root` warnings are expected and ok.)

The `startmain` script looks like the following:

```
#  /bin/bash

swift-init main start
```

11. Get an `-Storage-Url` and `-Auth-oken`:

```
curl -v -H th-Storage-User: test:testerc -H th-Storage-Pa
```

12. Check that you can `GEl` account:

```
curl -v -H  -Auth-oken: <token-from-x-auth-token-above>
```

13. Check that `swift` command provided by the python-swiftclient package works:

```
swift -A http://127.0.0.1:8080/auth/v1.0 -U test:tester -
```

14. Verify the functional tests run:

```
$HOME/swift/.functests
```

(Note: functional tests will first delete everything in the configured accounts.)

15. Verify the probe tests run:

```
$HOME/swift/.probetests
```

(Note: probe tests will reset your environment as they call `resetswift`for each test.)

Debugging Issues

If all doesn't go as planned, and tests fail, or you can't auth, or something doesn't work, here are some good starting places to look for issues:

1. Everything is logged using system facilities 6 usually in `/var/log/syslog`, but possibly in `/var/log/messages` on e.g. Fedora so that is a good first place to look for errors (most likely python tracebacks).
2. Make sure all of the server processes are running. For the base functionality, the Proxy, Account, Container, and Object servers should be running.
3. If one of the servers are not running, and no errors are logged to syslog, it may be useful to try to start the server manually, for example:`swift-object-server /etc/swift/object-server/1.conf` will start the object server. If there are problems not showing up in syslog, then you will likely see the traceback on startup.
4. If you need to, you can turn off syslog for unit tests. This can be useful for environments where `/dev/log` is unavailable, or which cannot rate limit (unit tests generate a lot of logs very quickly). Open the file `Sill _tESt_CON oG_ecLE` points to, and change the value of `fake_syslog` to `,rue`.