Installation of Openstack swift(SAIO - Swift All In One) on Ubuntu 16.04:

Step 1.0: Install dependencies:

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

Step 2.0: Setup a partition for storage

Here, we will be using a separate partition for the storage. So please make sure that you add another device when creating VM.

Step 2.1: Setup a single a partition

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

Where 'sdb' is the name of the disk attached for storage. Here, we formatted the disk into 'xfs' file system.

Step 2.2: Mounting the sdb drive on every boot of the system.

There are 2 possible ways to do this.

You can either edit the /etc/fstab file and write your mount instructions in that or you can run mount script manually after every restart of system.

If you want to edit the /etc/fstab file, add the following line in the file:

```
/dev/sdb1 /mnt/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0
```

(Note: It is recommended not to use the fstab file for this purpose. /etc/fstab is used to automatically mount filesystems at boot time. Make sure that the only items listed there are critical for the booting of the machine, because the machine might hang upon boot if it can't mount a device. This is likely when running a big box full of disks.)

If you want to run mount script manually after every restart of the system use the following command:

```
mount -t xfs -o rw,attr2,inode64 /home/ekstep/xfs_file /tmp
```

Step 2.3: 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/sdb1/4
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/1/node/sdb5 \
              /srv/2/node/sdb2 /srv/2/node/sdb6 \
              /srv/3/node/sdb3 /srv/3/node/sdb7 \
              /srv/4/node/sdb4 /srv/4/node/sdb8 \
              /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/; done
```

Here, "${USER}" denotes the username of the system that you are working on.

In above listed commands, we are creating 4 directories in our sdb1 drive to represent 4 different devices of the storage cluster. Then we have linked those 4 partitions with the 4 different server directories. Then we have given our user the owenership of /var/run/swift and all the server directories.

Step 3: 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/cache/swift4
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
```

Step 4: Creating an XFS tmp dir

The tests given at the end of the installation process require having an XFS directory available in /tmp or in the TMPDIR environment variable. To set up /tmp with an XFS filesystem, do the following:

```
cd ~
truncate -s 1GB xfs_file  # create 1GB fil for XFS in your home
directory
mkfs.xfs xfs_file
sudo mount -o loop,noatime,nodiratime xfs_file /tmp
sudo chmod -R 1777 /tmp
```

To persist this, edit and add the following to **/etc/fstab**:

```
/home/<your-user-name>/xfs_file /tmp xfs
rw,noatime,nodiratime,attr2,inode64,noquota 0 0
```

Step 5: Get the code

Step 5.1: Clone the python-swiftclient repo

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

Step 5.2: Build a development installation of python-swiftclient

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

Step 5.3: Clone the swift repo

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

Step 5.4: Build a development installation of swift

```
cd $HOME/swift;
sudo pip install --no-binary cryptography -r requirements.txt;
sudo python setup.py develop; cd -
```

Step 5.5: Install swift's test dependencies

```
cd $HOME/swift; sudo pip install -r test-requirements.txt
```

Step 6: Setting up rsync

Step 6.1:

Create **/etc/rsyncd.conf** and edit as per the sed command mentioned below.

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

Step 6.2: edit the following line in **/etc/default/rsync:**

```
RSYNC_ENABLE=true
```

Step 6.3: Start the rsync daemon

```
sudo systemctl enable rsync
sudo systemctl start rsync
```

Step 6.4: Verify rsync is accepting connections for all servers

Command:

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

Expected output:

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

Step 7: Start mamcached

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

Step 8: Setting up rsyslog for individual logging

Step 8.1: Install the swift rsyslogd configuration

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

To decide if you want all logs in one file or all the logs separated out or if you want hourly logs for stat processing, review the conf file. (Default contents of the conf file are given below)

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

# Uncomment the following to have hourly proxy logs for stats processing
#$template HourlyProxyLog,"/var/log/swift/hourly/%$YEAR%%$MONTH%%$DAY%%$HOUR%"
#local1.*;local1.!notice ?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.*                ~
```

Step 8.2: Edit **/etc/rsyslog.conf** and make the following change (usually in the "GLOBAL DIRECTIVES" section)

```
$PrivDropToGroup adm
```

Step 8.3: If using hourly logs (see above) perform:

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

Otherwise perform:

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

Step 8.4: Setup the logging directory and start syslog

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

Step 9: Configuring each node

Step 9.1: Optionally remove an existing directory

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

Step 9.2: 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
```

Step 9.3: Update **<your-user-name>** references in the Swift config files

```
find /etc/swift/ -name \*.conf | xargs sudo sed -i
"s/<your-user-name>/${USER}/"
```

Step 10: Setting up scripts for running Swift

Step 10.1: Copy the swift scripts for resetting the environment

```
mkdir -p $HOME/bin
cd $HOME/swift/doc; cp saio/bin/* $HOME/bin; cd -
chmod +x $HOME/bin/*
```

Step 10.2: Install the sample configuration file for running tests

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

Step 10.3: Add an environment variable for running tests below

```
echo "export SWIFT_TEST_CONFIG_FILE=/etc/swift/test.conf" >>
$HOME/.bashrc
```

Step 10.4: Be sure that your **PATH** includes the **bin** directory

```
echo "export PATH=${PATH}:$HOME/bin" >> $HOME/.bashrc
```

Step 10.5: Source the above environment variables into your current environment

```
. $HOME/.bashrc
```

Step 10.6: Construct the initial rings using the provided script

```
remakerings
```

The **remakerings** script looks like the following:

```
#!/bin/bash

set -e

cd /etc/swift

rm -f *.builder *.ring.gz backups/*.builder backups/*.ring.gz

swift-ring-builder object.builder create 10 3 1
swift-ring-builder object.builder add r1z1-127.0.0.1:6010/sdb1 1
swift-ring-builder object.builder add r1z2-127.0.0.2:6020/sdb2 1
swift-ring-builder object.builder add r1z3-127.0.0.3:6030/sdb3 1
swift-ring-builder object.builder add r1z4-127.0.0.4:6040/sdb4 1
swift-ring-builder object.builder rebalance
swift-ring-builder object-1.builder create 10 2 1
swift-ring-builder object-1.builder add r1z1-127.0.0.1:6010/sdb1 1
swift-ring-builder object-1.builder add r1z2-127.0.0.2:6020/sdb2 1
swift-ring-builder object-1.builder add r1z3-127.0.0.3:6030/sdb3 1
swift-ring-builder object-1.builder add r1z4-127.0.0.4:6040/sdb4 1
swift-ring-builder object-1.builder rebalance
swift-ring-builder object-2.builder create 10 6 1
swift-ring-builder object-2.builder add r1z1-127.0.0.1:6010/sdb1 1
swift-ring-builder object-2.builder add r1z1-127.0.0.1:6010/sdb5 1
swift-ring-builder object-2.builder add r1z2-127.0.0.2:6020/sdb2 1
swift-ring-builder object-2.builder add r1z2-127.0.0.2:6020/sdb6 1
swift-ring-builder object-2.builder add r1z3-127.0.0.3:6030/sdb3 1
swift-ring-builder object-2.builder add r1z3-127.0.0.3:6030/sdb7 1
swift-ring-builder object-2.builder add r1z4-127.0.0.4:6040/sdb4 1
swift-ring-builder object-2.builder add r1z4-127.0.0.4:6040/sdb8 1
swift-ring-builder object-2.builder rebalance
swift-ring-builder container.builder create 10 3 1
swift-ring-builder container.builder add r1z1-127.0.0.1:6011/sdb1 1
swift-ring-builder container.builder add r1z2-127.0.0.2:6021/sdb2 1
swift-ring-builder container.builder add r1z3-127.0.0.3:6031/sdb3 1
swift-ring-builder container.builder add r1z4-127.0.0.4:6041/sdb4 1
swift-ring-builder container.builder rebalance
swift-ring-builder account.builder create 10 3 1
swift-ring-builder account.builder add r1z1-127.0.0.1:6012/sdb1 1
swift-ring-builder account.builder add r1z2-127.0.0.2:6022/sdb2 1
swift-ring-builder account.builder add r1z3-127.0.0.3:6032/sdb3 1
swift-ring-builder account.builder add r1z4-127.0.0.4:6042/sdb4 1
swift-ring-builder account.builder rebalance
```

The expected output is:

```
Device d0r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with 1.0 weight got id 0
Device d1r1z2-127.0.0.2:6020R127.0.0.2:6020/sdb2_"" with 1.0 weight got id 1
```

```
Device d2r1z3-127.0.0.3:6030R127.0.0.3:6030/sdb3_"" with 1.0 weight got id 2
Device d3r1z4-127.0.0.4:6040R127.0.0.4:6040/sdb4_"" with 1.0 weight got id 3
Reassigned 3072 (300.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with 1.0 weight got id 0
Device d1r1z2-127.0.0.2:6020R127.0.0.2:6020/sdb2_"" with 1.0 weight got id 1
Device d2r1z3-127.0.0.3:6030R127.0.0.3:6030/sdb3_"" with 1.0 weight got id 2
Device d3r1z4-127.0.0.4:6040R127.0.0.4:6040/sdb4_"" with 1.0 weight got id 3
Reassigned 2048 (200.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with 1.0 weight got id 0
Device d1r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb5_"" with 1.0 weight got id 1
Device d2r1z2-127.0.0.2:6020R127.0.0.2:6020/sdb2_"" with 1.0 weight got id 2
Device d3r1z2-127.0.0.2:6020R127.0.0.2:6020/sdb6_"" with 1.0 weight got id 3
Device d4r1z3-127.0.0.3:6030R127.0.0.3:6030/sdb3_"" with 1.0 weight got id 4
Device d5r1z3-127.0.0.3:6030R127.0.0.3:6030/sdb7_"" with 1.0 weight got id 5
Device d6r1z4-127.0.0.4:6040R127.0.0.4:6040/sdb4_"" with 1.0 weight got id 6
Device d7r1z4-127.0.0.4:6040R127.0.0.4:6040/sdb8_"" with 1.0 weight got id 7
Reassigned 6144 (600.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6011R127.0.0.1:6011/sdb1_"" with 1.0 weight got id 0
Device d1r1z2-127.0.0.2:6021R127.0.0.2:6021/sdb2_"" with 1.0 weight got id 1
Device d2r1z3-127.0.0.3:6031R127.0.0.3:6031/sdb3_"" with 1.0 weight got id 2
Device d3r1z4-127.0.0.4:6041R127.0.0.4:6041/sdb4_"" with 1.0 weight got id 3
Reassigned 3072 (300.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6012R127.0.0.1:6012/sdb1_"" with 1.0 weight got id 0
Device d1r1z2-127.0.0.2:6022R127.0.0.2:6022/sdb2_"" with 1.0 weight got id 1
Device d2r1z3-127.0.0.3:6032R127.0.0.3:6032/sdb3_"" with 1.0 weight got id 2
Device d3r1z4-127.0.0.4:6042R127.0.0.4:6042/sdb4_"" with 1.0 weight got id 3
Reassigned 3072 (300.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
```

Note that 3 object rings are created in order to test storage policies and EC in the SAIO environment. The EC ring is the only one with all 8 devices. There are also two replication rings, one for 3x replication and another for 2x replication, but those rings only use 4 devices.

Step 10.7: Verify the unit tests run

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

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

Step 10.8: 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.)

Step 10.9: Get an **X-Storage-Url** and **X-Auth-Token**

```
curl -v -H 'X-Storage-User: test:tester' -H 'X-Storage-Pass:
testing' http://127.0.0.1:8080/auth/v1.0
```

Step 10.10: Check that you can **GET** account

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

Step 10.11: 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 -K testing
```

```
stat
```

Step 10.12: Verify the functional tests run

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

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

Step 10.13: Verify the probe tests run

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

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

Troubleshooting:

Following are some useful tips for troubleshooting while installing openstack swift:

1) You might encounter some errors related to some installed packages. The error might be because of some compatibility issues with the installed version of that package. In such cases, clone that package's latest code from github and then again install that package. **(Installing from apt repository won't help.)**
2) Whenever you encounter any permission related errors, check that in **/var/run/swift** directory, all the directories' and their **sub-directories'** owner is same as the user on which you are working.
3) To remove some errors, you might have to make some files under **/var/run/swift** executable.
4) More tips about troubleshooting an be found on the following link:

https://docs.openstack.org/swift/latest/development_saio.html#debugging-issues

References:

https://docs.openstack.org/swift/latest/development_saio.html#debugging-issues
https://www.safaribooksonline.com/library/view/openstack-swift/9781491903841/ch09.html