Evergreen ILS on Ubuntu 18.04

Evergreen is highly-scalable software for libraries that helps library patrons find library materials, and helps libraries manage, catalog, and circulate those materials, no matter how large or complex the libraries.

Evergreen is open source software, licensed under the GNU GPL, version 2 or later.

Learn more here. This guide will be on an Ubuntu 18.04 server.

Some reference for the installation

- The **user** Linux account is the account that you use to log onto the Linux system as a regular user.
- The **root** Linux account is an account that has system administrator privileges. Look for # on the console. Sometimes `sudo` will be used to run single commands as the root user.
- The **opensrf** Linux account is an account that you will create as part of installing OpenSRF.
- The minimum supported version of OpenSRF is 3.0.0.
- **PostgreSQL** is required The minimum supported version is 9.4.
- The **postgres** Linux account is created automatically when you install the PostgreSQL database server.
- The **evergreen** PostgreSQL account is a superuser that you will create to connect to the database server.
- The **egadmin** Evergreen account is an administrator account for Evergreen.
-
    -

OpenSRF

First download and unpack the OpenSRF source as the Linux **user**.

```
wget https://evergreen-ils.org/downloads/opensrf-3.1.0.tar.gz
tar -xvf opensrf-3.1.0.tar.gz
cd opensrf-3.1.0/
```

Issue the following commands from the opensrf folder as the **root** Linux account to install prerequisites.

```
sudo apt-get install make
sudo make -f src/extras/Makefile.install ubuntu-bionic
```

As the Linux **user,** use the `configure` command to configure OpenSRF, and the `make` command to build OpenSRF. The default installation prefix (PREFIX) for OpenSRF is `/opensrf/`.

```
./configure --prefix=/openils --sysconfdir=/openils/conf
make
```

If there were no build errors, then as **root** run the make install.

```
sudo make install
```

Create the **opensrf** user as the **root** user.

```
sudo useradd -m -s /bin/bash opensrf
sudo su -c 'echo "export PATH=\$PATH:/openils/bin" >> /home/opensrf/.bashrc'
sudo passwd opensrf
sudo chown -R opensrf:opensrf /openils
```

Define your public and private OpenSRF domains.
This is a single server setup so the /etc/hosts file will be used to add the public and private addresses.
Use the **root** user to add the following to /etc/hosts file.

```
127.0.1.2       public.localhost        public
127.0.1.3       private.localhost       private
```

Adjusting the system dynamic library path.

```
sudo su -c 'echo /openils/lib > /etc/ld.so.conf.d/opensrf.conf'
sudo ldconfig
```

OpenSRF requires an XMPP (Jabber) server. The ejabberd packages will be used.
First stop the existing service.

```
sudo systemctl stop ejabberd.service
```

Open `/etc/ejabberd/ejabberd.yml` and make the following changes as the **root** user:

> If you don't want to edit this file, download a preconfigured one fromhere

1. Define your public and private domains in the `hosts` directive. For example:
   ```
   hosts:
     - "localhost"
     - "private.localhost"
     - "public.localhost"
   ```
2. Change `starttls_required` to false
3. Change `auth_password_format` to plain
4. Change `shaper:` `normal` and `fast` values to 500000
5. Increase the `max_user_sessions:` `all:` value to 10000
6. Comment out the `mod_offline` directive
   ```
   ##mod_offline:
   ##access_max_user_messages: max_user_offline_messages
   ```
7. Uncomment the `mod_legacy_auth` directive

Or use the downloaded file.

```
sudo mv /etc/ejabberd/{ejabberd.yml,ejabberd.yml.org}
sudo wget -O /etc/ejabberd/ejabberd.yml https://git.clusterapps.com/snippets/6/raw
```

Start the ejabberd server:

```
sudo systemctl start ejabberd.service
```

On each domain, you need two Jabber users to manage the OpenSRF communications:

- a `router` user, to whom all requests to connect to an OpenSRF service will be routed; this Jabber user must be named `router`
- an `opensrf` user, which clients use to connect to OpenSRF services; this user can be named anything.

```
sudo ejabberdctl register router private.localhost UseARealPasswordNotthisone
sudo ejabberdctl register opensrf private.localhost UseARealPasswordNotthisone
sudo ejabberdctl register router public.localhost UseARealPasswordNotthisone
sudo ejabberdctl register opensrf public.localhost UseARealPasswordNotthisone
```

As the **opensrf** Linux account, copy the example configuration files to create your locally OpenSRF configuration.

```
sudo su - opensrf
cd /openils/conf
cp opensrf_core.xml.example opensrf_core.xml
cp opensrf.xml.example opensrf.xml
```

Edit `opensrf_core.xml` file to update the username / password pairs to match the Jabber user accounts you created.

1. `<config><opensrf>` = use the private Jabber `opensrf` user
2. `<config><gateway>` = use the public Jabber `opensrf` user
3. `<config><routers><router>` = use the public Jabber `router` user
4. `<config><routers><router>` = use the private Jabber `router` user

Enable the opensrf Linux account to use `srfsh` as the **opensrf** user.

```
cp /openils/conf/srfsh.xml.example /home/opensrf/.srfsh.xml
```

Update the password to match the password you set for the Jabber `opensrf` user at the `private.localhost` domain.

```
vim /home/opensrf/.srfsh.xml
```

Start all OpenSRF services as the **opensrf** Linux account.

```
osrf_control --localhost --start-all
```

1. Start the `srfsh` interactive OpenSRF shell by issuing the following command as the **opensrf** Linux account:
   Starting the `srfsh` interactive OpenSRF shell
   ```
   srfsh
   ```
2. Issue the following request to test the `opensrf.math` service:
   ```
   srfsh# request opensrf.math add 2,2
   ```
   You should receive the value `4`.

Install WebSockets as the **root** user.

```
apt-get install git-core
cd /tmp
git clone https://github.com/disconnect/apache-websocket
cd apache-websocket
apxs2 -i -a -c mod_websocket.c
```

Create the websocket Apache instance

```
sh /usr/share/doc/apache2/examples/setup-instance websockets
```

Remove from the main apache instance

```
a2dismod websocket
```

Change to the directory into which you unpacked OpenSRF, then copy config files.

```
cp examples/apache_24/websockets/apache2.conf /etc/apache2-websockets/
```

Add configuration variables to the end of /etc/apache2-websockets/envvars.

```
export OSRF_WEBSOCKET_IDLE_TIMEOUT=120
export OSRF_WEBSOCKET_IDLE_CHECK_INTERVAL=5
export OSRF_WEBSOCKET_CONFIG_FILE=/openils/conf/opensrf_core.xml
export OSRF_WEBSOCKET_CONFIG_CTXT=gateway
export OSRF_WEBSOCKET_MAX_REQUEST_WAIT_TIME=600
```

Before you can start websockets, you must install a valid SSL certificate in `/etc/apache2/ssl/`.

Evergreen application

Download and extract the archive as the Linux **user.**

```
wget https://evergreen-ils.org/downloads/Evergreen-ILS-3.3.0.tar.gz
tar -xvf Evergreen-ILS-3.3.0.tar.gz
cd Evergreen-ILS-3.3.0/
```

Issue the following commands as the **root** Linux account to install prerequisites using the `Makefile.install` prerequisite installer

```
make -f Open-ILS/src/extras/Makefile.install ubuntu-bionic
```

From the Evergreen source directory, issue the following commands as the **user** Linux account to configure and build Evergreen:

```
PATH=/openils/bin:$PATH ./configure --prefix=/openils --sysconfdir=/openils/conf
make
```

Issue the following command as the **root** Linux account to install Evergreen

```
sudo make install
```

Change ownership of the Evergreen files

```
sudo chown -R opensrf:opensrf /openils
```

Run ldconfig as the **root** Linux account.

```
sudo ldconfig
```

Issue the following commands as the **root** Linux account to copy the Apache configuration files from the Evergreen source archive.

```
sudo cp Open-ILS/examples/apache_24/eg_24.conf /etc/apache2/sites-available/eg.conf
sudo cp Open-ILS/examples/apache_24/eg_vhost_24.conf /etc/apache2/eg_vhost.conf
sudo cp Open-ILS/examples/apache_24/eg_startup /etc/apache2/
```

As the **root** Linux account, edit the `eg.conf` file that you copied.

To enable access to the offline upload / execute interface from any workstation on any network, make the following change (and note that you **must** secure this for a production instance):

- Replace `Require host 10.0.0.0/8` with `Require all granted`

As the **root** Linux account, edit `/etc/apache2/envvars`. Change `export APACHE_RUN_USER=www-data` to `export APACHE_RUN_USER=opensrf`.

As **root** edit `/etc/apache2/apache2.conf` and make the following changes.

```
sudo vim /etc/apache2/apache2.conf
```

1. Change `KeepAliveTimeout` to `1`.
2. Change `MaxKeepAliveRequests` to `100`.

As the **root** Linux account, configure the prefork module to start and keep enough Apache servers available to provide quick responses to clients.

```
sudo vim /etc/apache2/mods-available/mpm_prefork.conf
```

```
<IfModule mpm_prefork_module>
  StartServers          15
  MinSpareServers        5
  MaxSpareServers       15
```

```
    MaxRequestWorkers       75
    MaxConnectionsPerChild 500
</IfModule>
```

As the **root** user, enable the mpm_prefork module:

```
sudo a2dismod mpm_event
sudo a2enmod mpm_prefork
```

As the **root** Linux account, enable the Evergreen site:

```
sudo a2dissite 000-default
sudo a2ensite eg.conf
```

As the **root** Linux account, enable Apache to write to the lock directory

```
sudo chown opensrf /var/lock/apache2
```

To configure OpenSRF for Evergreen, issue the following commands as the **opensrf** Linux account. (Yes you did edit these files earlier but now there are more settings)

```
cp -b /openils/conf/opensrf_core.xml.example /openils/conf/opensrf_core.xml
cp -b /openils/conf/opensrf.xml.example /openils/conf/opensrf.xml
```

Edited the `opensrf_core.xml` as the **opensrf** user and enter the same passwords from the OpenSRF install steps.

To enable the default set of hooks, issue the following command as the **opensrf** Linux account:

```
cp -b /openils/conf/action_trigger_filters.json.example /openils/conf/action_trigger_filters.json
```

Evergreen database

In production environments, this would be on a dedicated server or cluster. For this deployment the database will be on the same server. If you are serious about your setup you would never install the application and the database on the same server. This is for testing only. For the best performance, run PostgreSQL on RHEL or Solaris.

Installing PostgreSQL server packages as **root**. Use the Makefile provided in the Evergreen archive.

```
sudo make -f Open-ILS/src/extras/Makefile.install postgres-server-ubuntu-bionic
```

Issue the following command as the **postgres** Linux account to create a new PostgreSQL superuser named `evergreen`.

```
sudo su - postgres
createuser -s -P evergreen
```

Issue the following command as the **root** Linux account from inside the Evergreen source directory, replacing <user>, <password>, <hostname>, <port>, and <dbname> with the appropriate values for your PostgreSQL database (where <user> and <password> are for the **evergreen** PostgreSQL account you just created), and replace <admin-user> and <admin-pass> with the values you want for the **egadmin** Evergreen administrator account:

```
perl Open-ILS/src/support-scripts/eg_db_config --update-config \
 --service all --create-database --create-schema --create-offline \
 --user <user> --password <password> --hostname <hostname> --port <port> \
 --database <dbname> --admin-user <admin-user> --admin-pass <admin-pass>
```

If you add the `--load-all-sample` parameter to the `eg_db_config` command, a set of authority and bibliographic records, call numbers, copies, staff and regular users, and transactions will be loaded into your target database. This sample dataset is commonly referred to as the *concerto* sample data, and can be useful for testing out Evergreen functionality and for creating problem reports that developers can easily recreate with their own copy of the *concerto* sample data.

Starting Evergreen

Start the `memcached` and `ejabberd` services

```
sudo systemctl enable --now ejabberd
sudo systemctl enable --now memcached
```

As the **opensrf** Linux account, start Evergreen. The `-l` flag in the following command is only necessary if you want to force Evergreen to treat the hostname as `localhost`;

```
osrf_control -l --start-all
```

if you configured `opensrf.xml` using the real hostname of your machine as returned by `perl -ENet::Domain 'print Net::Domain::hostfqdn() . "\n";'`, you should not use the `-l` flag. In a multi server setup, do not use localhost.

As the **opensrf** Linux account, generate the Web files needed by the web staff client and catalogue and update the organization unit proximity. Do this the first time you start Evergreen, and after that each time you change the library org unit configuration.

```
autogen.sh
```

As the **root** Linux account, restart the Apache Web server:

```
systemctl restart apache2
```

Testing connections

Once you have installed and started Evergreen, test your connection to Evergreen via `srfsh`.
As the **opensrf** Linux account. <admin-user> <admin-pass> are the egadmin username and password created earlier.

```
/openils/bin/srfsh
srfsh% login <admin-user> <admin-pass>
```

The output should look like this:

```
##############################################
Received Data: "$2a$10$londKQogYvvF71H92Wwpme"

------------------------------------
Request Completed Successfully
Request Time in seconds: 0.052501
------------------------------------

Received Data: {
"ilsevent":0,
"textcode":"SUCCESS",
"desc":"Success",
"pid":32474,
"stacktrace":"oils_auth.c:636",
"payload":{
"authtoken":"bd4f67a646ee4c39e923a272dc6c79a3",
"authtime":420
}
}

------------------------------------
Request Completed Successfully
Request Time in seconds: 0.171812
------------------------------------
Login Session: bd4f67a646ee4c39e923a272dc6c79a3. Session timeout: 420.000000

##############################################
```

Revision #1
Created 29 April 2019 22:59:50 by Michael Cleary
Updated 16 February 2022 23:58:51 by Michael Cleary