Lemur Documentation

Release 0.6.0dev

Kevin Glisson

May 26, 2017

Contents

Lemur is a TLS management service. It attempts to help track and create certificates. By removing common issues with CSR creation it gives normal developers 'sane' TLS defaults and helps security teams push TLS usage throughout an organization.

CHAPTER 1

Installation

Quickstart

This guide will step you through setting up a Python-based virtualenv, installing the required packages, and configuring the basic web service. This guide assumes a clean Ubuntu 14.04 instance, commands may differ based on the OS and configuration being used.

Pressed for time? See the Lemur docker file on Github.

Dependencies

Some basic prerequisites which you'll need in order to run Lemur:

- A UNIX-based operating system (we test on Ubuntu, develop on OS X)
- Python 3.5 or greater
- PostgreSQL 9.4 or greater
- Nginx

Note: Lemur was built with in AWS in mind. This means that things such as databases (RDS), mail (SES), and TLS (ELB), are largely handled for us. Lemur does **not** require AWS to function. Our guides and documentation try to be as generic as possible and are not intended to document every step of launching Lemur into a given environment.

Installing Build Dependencies

If installing Lemur on a bare Ubuntu OS you will need to grab the following packages so that Lemur can correctly build it's dependencies:

```
$ sudo apt-get update
$ sudo apt-get install nodejs-legacy python-pip python-dev python3-dev libpq-dev␣
↪build-essential libssl-dev libffi-dev nginx git supervisor npm postgresql
```

Note: PostgreSQL is only required if your database is going to be on the same host as the webserver. npm is needed if you're installing Lemur from the source (e.g., from git).

Now, install Python `virtualenv` package:

```
$ sudo pip install -U virtualenv
```

Setting up an Environment

In this guide, Lemur will be installed in `/www` , so you need to create that structure first:

```
$ sudo mkdir /www
$ cd /www
```

Clone Lemur inside the just created directory and give yourself write permission (we assume `lemur` is the user):

```
$ sudo useradd lemur
$ sudo passwd lemur
$ sudo mkdir /home/lemur
$ sudo chown lemur:lemur /home/lemur
$ sudo git clone https://github.com/Netflix/lemur
$ sudo chown -R lemur lemur/
```

Create the virtual environment, activate it and enter the Lemur's directory:

```
$ su lemur
$ virtualenv -p python3 lemur
$ source /www/lemur/bin/activate
$ cd lemur
```

Note: Activating the environment adjusts your PATH, so that things like pip now install into the virtualenv by default.

Installing from Source

Once your system is prepared, ensure that you are in the virtualenv:

```
$ which python
```

And then run:

```
$ make release
```

Note: This command will install npm dependencies as well as compile static assets.

You may also run with the urlContextPath variable set. If this is set it will add the desired context path for subsequent calls back to lemur. This will only edit the front end code for calls back to the server, you will have to make sure the server knows about these routes.

```
Example:
  urlContextPath=lemur
  /api/1/auth/providers -> /lemur/api/1/auth/providers
```

```
$ make release urlContextPath={desired context path}
```

Creating a configuration

Before we run Lemur, we must create a valid configuration file for it. The Lemur command line interface comes with a simple command to get you up and running quickly.

Simply run:

```
$ lemur create_config
```

Note: This command will create a default configuration under `~/.lemur/lemur.conf.py` you can specify this location by passing the `config_path` parameter to the `create_config` command.

You can specify `-c` or `--config` to any Lemur command to specify the current environment you are working in. Lemur will also look under the environmental variable `LEMUR_CONF` should that be easier to setup in your environment.

Update your configuration

Once created, you will need to update the configuration file with information about your environment, such as which database to talk to, where keys are stored etc.

```
$ vi ~/.lemur/lemur.conf.py
```

Note: If you are unfamiliar with the SQLALCHEMY_DATABASE_URI string it can be broken up like so: `postgresql://userame:password@<database-fqdn>:<database-port>/<database-name>`

Before Lemur will run you need to fill in a few required variables in the configuration file:

```
LEMUR_SECURITY_TEAM_EMAIL
#/the e-mail address needs to be enclosed in quotes
LEMUR_DEFAULT_COUNTRY
LEMUR_DEFAULT_STATE
LEMUR_DEFAULT_LOCATION
LEMUR_DEFAULT_ORGANIZATION
LEMUR_DEFAULT_ORGANIZATIONAL_UNIT
```

Setup Postgres

For production, a dedicated database is recommended, for this guide we will assume postgres has been installed and is on the same machine that Lemur is installed on.

First, set a password for the postgres user. For this guide, we will use `lemur` as an example but you should use the database password generated by Lemur:

```
$ sudo -u postgres -i
$ psql
postgres=# CREATE USER lemur WITH PASSWORD 'lemur';
```

Once successful, type CTRL-D to exit the Postgres shell.

Next, we will create our new database:

```
$ sudo -u postgres createdb lemur
```

Note: For this guide we assume you will use the *postgres* user to connect to your database, when deploying to a VM or container this is often all you will need. If you have a shared database it is recommend you give Lemur its own user.

Note: Postgres 9.4 or greater is required as Lemur relies advanced data columns (e.g. JSON Column type)

Initializing Lemur

Lemur provides a helpful command that will initialize your database for you. It creates a default user (`lemur`) that is used by Lemur to help associate certificates that do not currently have an owner. This is most commonly the case when Lemur has discovered certificates from a third party source. This is also a default user that can be used to administer Lemur.

In addition to creating a new user, Lemur also creates a few default email notifications. These notifications are based on a few configuration options such as `LEMUR_SECURITY_TEAM_EMAIL` . They basically guarantee that every certificate within Lemur will send one expiration notification to the security team.

Additional notifications can be created through the UI or API. See Creating Notifications and *Command Line Interface* for details.

Make note of the password used as this will be used during first login to the Lemur UI.

```
$ cd /www/lemur/lemur
$ lemur init
```

Note: It is recommended that once the `lemur` user is created that you create individual users for every day access. There is currently no way for a user to self enroll for Lemur access, they must have an administrator create an account for them or be enrolled automatically through SSO. This can be done through the CLI or UI. See Creating Users and *Command Line Interface* for details.

Setup a Reverse Proxy

By default, Lemur runs on port 8000. Even if you change this, under normal conditions you won't be able to bind to port 80. To get around this (and to avoid running Lemur as a privileged user, which you shouldn't), we need setup a simple web proxy. There are many different web servers you can use for this, we like and recommend Nginx.

Proxying with Nginx

You'll use the builtin `HttpProxyModule` within Nginx to handle proxying. Edit the `/etc/nginx/sites-available/default` file according to the lines below

```
location /api {
    proxy_pass  http://127.0.0.1:8000;
    proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_
↪504;
    proxy_redirect off;
    proxy_buffering off;
```

```
    proxy_set_header        Host            $host;
    proxy_set_header        X-Real-IP       $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
}

location / {
    root /www/lemur/lemur/static/dist;
    include mime.types;
    index index.html;
}
```

Note: See *Production* for more details on using Nginx.

After making these changes, restart Nginx service to apply them:

```
$ sudo service nginx restart
```

Starting the Web Service

Lemur provides a built-in web server (powered by gunicorn and eventlet) to get you off the ground quickly.

To start the web server, you simply use `lemur start`. If you opted to use an alternative configuration path you can pass that via the `--config` option.

Note: You can login with the default user created during *Initializing Lemur* or any other user you may have created.

```
# Lemur's server runs on port 8000 by default. Make sure your client reflects
# the correct host and port!
lemur --config=/etc/lemur.conf.py start -b 127.0.0.1:8000
```

You should now be able to test the web service by visiting `http://localhost:8000/`.

Running Lemur as a Service

We recommend using whatever software you are most familiar with for managing Lemur processes. One option is Supervisor.

Configure `supervisord`

Configuring Supervisor couldn't be more simple. Just point it to the `lemur` executable in your virtualenv's `bin/` folder and you're good to go.

```
[program:lemur-web]
directory=/www/lemur/
command=/www/lemur/bin/lemur start
autostart=true
autorestart=true
redirect_stderr=true
stdout_logfile=syslog
stderr_logfile=syslog
```

See *Using Supervisor* for more details on using Supervisor.

Syncing

Lemur uses periodic sync tasks to make sure it is up-to-date with its environment. Things change outside of Lemur we do our best to reconcile those changes. The recommended method is to use CRON:

```
$ crontab -e
*/15 * * * * lemur sync -s all
0 22 * * * lemur check_revoked
0 22 * * * lemur notify
```

Additional Utilities

If you're familiar with Python you'll quickly find yourself at home, and even more so if you've used Flask. The `lemur` command is just a simple wrapper around Flask's `manage.py` , which means you get all of the power and flexibility that goes with it.

Some of the features which you'll likely find useful are listed below.

lock

Encrypts sensitive key material - this is most useful for storing encrypted secrets in source code.

unlock

Decrypts sensitive key material - used to decrypt the secrets stored in source during deployment.

What's Next?

Get familiar with how Lemur works by reviewing the *User Guide*. When you're ready see *Production* for more details on how to configure Lemur for production.

The above just gets you going, but for production there are several different security considerations to take into account. Remember, Lemur is handling sensitive data and security is imperative.

Production

There are several steps needed to make Lemur production ready. Here we focus on making Lemur more reliable and secure.

Basics

Because of the sensitivity of the information stored and maintained by Lemur it is important that you follow standard host hardening practices:

- Run Lemur with a limited user
- Disabled any unneeded services

- Enable remote logging
- Restrict access to host

Credential Management

Lemur often contains credentials such as mutual TLS keys or API tokens that are used to communicate with third party resources and for encrypting stored secrets. Lemur comes with the ability to automatically encrypt these keys such that your keys not be in clear text.

The keys are located within lemur/keys and broken down by environment.

To utilize this ability use the following commands:

```
lemur lock
```

and

```
lemur unlock
```

If you choose to use this feature ensure that the keys are decrypted before Lemur starts as it will have trouble communicating with the database otherwise.

Entropy

Lemur generates private keys for the certificates it creates. This means that it is vitally important that Lemur has enough entropy to draw from. To generate private keys Lemur uses the python library Cryptography. In turn Cryptography uses OpenSSL bindings to generate keys just like you might from the OpenSSL command line. OpenSSL draws its initial entropy from system during startup and uses PRNGs to generate a stream of random bytes (as output by /dev/urandom) whenever it needs to do a cryptographic operation.

What does all this mean? Well in order for the keys that Lemur generates to be strong, the system needs to interact with the outside world. This is typically accomplished through the systems hardware (thermal, sound, video user-input, etc.) since the physical world is much more "random" than the computer world.

If you are running Lemur on its own server with its own hardware "bare metal" then the entropy of the system is typically "good enough" for generating keys. If however you are using a VM on shared hardware there is a potential that your initial seed data (data that was initially fed to the PRNG) is not very good. What's more, VMs have been known to be unable to inject more entropy into the system once it has been started. This is because there is typically very little interaction with the server once it has been started.

The amount of effort you wish to expend ensuring that Lemur has good entropy to draw from is up to your specific risk tolerance and how Lemur is configured.

If you wish to generate more entropy for your system we would suggest you take a look at the following resources:

- WES-entropy-client
- haveged

For additional information about OpenSSL entropy issues:

- Managing and Understanding Entropy Usage

TLS/SSL

Nginx

Nginx is a very popular choice to serve a Python project:

- It's fast.
- It's lightweight.
- Configuration files are simple.

Nginx doesn't run any Python process, it only serves requests from outside to the Python server.

Therefore, there are two steps:

- Run the Python process.
- Run Nginx.

You will benefit from having:

- the possibility to have several projects listening to the port 80;
- your web site processes won't run with admin rights, even if –user doesn't work on your OS;
- the ability to manage a Python process without touching Nginx or the other processes. It's very handy for updates.

You must create a Nginx configuration file for Lemur. On GNU/Linux, they usually go into /etc/nginx/conf.d/. Name it lemur.conf.

proxy_pass just passes the external request to the Python process. The port must match the one used by the Lemur process of course.

You can make some adjustments to get a better user experience:

```
server_tokens off;
add_header X-Frame-Options DENY;
add_header X-Content-Type-Options nosniff;
add_header X-XSS-Protection "1; mode=block";

server {
  listen       80;
  return       301 https://$host$request_uri;
}

server {
   listen      443;
   access_log  /var/log/nginx/log/lemur.access.log;
   error_log   /var/log/nginx/log/lemur.error.log;

   location /api {
        proxy_pass  http://127.0.0.1:8000;
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503␣
↪http_504;
        proxy_redirect off;
        proxy_buffering off;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location / {
        root /path/to/lemur/static/dist;
        include mime.types;
        index index.html;
    }
```

```
}
```

This makes Nginx serve the favicon and static files which it is much better at than python.

It is highly recommended that you deploy TLS when deploying Lemur. This may be obvious given Lemur's purpose but the sensitive nature of Lemur and what it controls makes this essential. This is a sample config for Lemur that also terminates TLS:

```
server_tokens off;
add_header X-Frame-Options DENY;
add_header X-Content-Type-Options nosniff;
add_header X-XSS-Protection "1; mode=block";

server {
  listen       80;
  return       301 https://$host$request_uri;
}

server {
   listen      443;
   access_log  /var/log/nginx/log/lemur.access.log;
   error_log   /var/log/nginx/log/lemur.error.log;

   # certs sent to the client in SERVER HELLO are concatenated in ssl_certificate
   ssl_certificate /path/to/signed_cert_plus_intermediates;
   ssl_certificate_key /path/to/private_key;
   ssl_session_timeout 1d;
   ssl_session_cache shared:SSL:50m;

   # Diffie-Hellman parameter for DHE ciphersuites, recommended 2048 bits
   ssl_dhparam /path/to/dhparam.pem;

   # modern configuration. tweak to your needs.
   ssl_protocols TLSv1.1 TLSv1.2;
   ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
↪AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-
↪AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:
↪ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-
↪AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-
↪RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-
↪RSA-AES256-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!3DES:!MD5:!PSK';
   ssl_prefer_server_ciphers on;

   # HSTS (ngx_http_headers_module is required) (15768000 seconds = 6 months)
   add_header Strict-Transport-Security max-age=15768000;

   # OCSP Stapling ---
   # fetch OCSP records from URL in ssl_certificate and cache them
   ssl_stapling on;
   ssl_stapling_verify on;

   ## verify chain of trust of OCSP response using Root CA and Intermediate certs
   ssl_trusted_certificate /path/to/root_CA_cert_plus_intermediates;

   resolver <IP DNS resolver>;

   location /api {
```

```
        proxy_pass  http://127.0.0.1:8000;
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503␣
↪http_504;
        proxy_redirect off;
        proxy_buffering off;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location / {
        root /path/to/lemur/static/dist;
        include mime.types;
        index index.html;
    }


}
```

Note: Some paths will have to be adjusted based on where you have choose to install Lemur.

Apache

An example apache config:

```
<VirtualHost *:443>
    ...
    SSLEngine on
    SSLCertificateFile      /path/to/signed_certificate
    SSLCertificateChainFile /path/to/intermediate_certificate
    SSLCertificateKeyFile   /path/to/private/key
    SSLCACertificateFile    /path/to/all_ca_certs

    # intermediate configuration, tweak to your needs
    SSLProtocol             all -SSLv2 -SSLv3
    SSLCipherSuite          ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:
↪ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:
↪DHE-DSS-AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
↪SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-
↪ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-
↪SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-
↪AES256-SHA:DHE-RSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:
↪AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:!aNULL:!eNULL:!EXPORT:
↪!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-
↪CBC3-SHA
    SSLHonorCipherOrder     on

    # HSTS (mod_headers is required) (15768000 seconds = 6 months)
    Header always set Strict-Transport-Security "max-age=15768000"
    ...

 # Set the lemur DocumentRoot to static/dist
 DocumentRoot /www/lemur/lemur/static/dist

 # Uncomment to force http 1.0 connections to proxy
```

```
 # SetEnv force-proxy-request-1.0 1

 #Don't keep proxy connections alive
 SetEnv proxy-nokeepalive 1

 # Only need to do reverse proxy
 ProxyRequests Off

 # Proxy requests to the api to the lemur service (and sanitize redirects from it)
 ProxyPass "/api" "http://127.0.0.1:8000/api"
 ProxyPassReverse "/api" "http://127.0.0.1:8000/api"

</VirtualHost>
```

Also included in the configurations above are several best practices when it comes to deploying TLS. Things like enabling HSTS, disabling vulnerable ciphers are all good ideas when it comes to deploying Lemur into a production environment.

Note: This is a rather incomplete apache config for running Lemur (needs mod_wsgi etc.), if you have a working apache config please let us know!

See also:

Mozilla SSL Configuration Generator

Supervisor

Supervisor is a very nice way to manage you Python processes. We won't cover the setup (which is just apt-get install supervisor or pip install supervisor most of the time), but here is a quick overview on how to use it.

Create a configuration file named supervisor.ini:

```
[unix_http_server]
file=/tmp/supervisor.sock

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock

[rpcinterface:supervisor]
supervisor.rpcinterface_factory=supervisor.rpcinterface:make_main_rpcinterface

[supervisord]
logfile=/tmp/lemur.log
logfile_maxbytes=50MB
logfile_backups=2
loglevel=trace
pidfile=/tmp/supervisord.pid
nodaemon=false
minfds=1024
minprocs=200

[program:lemur]
command=python /path/to/lemur/manage.py manage.py start

directory=/path/to/lemur/
environment=PYTHONPATH='/path/to/lemur/',LEMUR_CONF='/home/lemur/.lemur/lemur.conf.py'
```

```
user=lemur
autostart=true
autorestart=true
```

The 4 first entries are just boiler plate to get you started, you can copy them verbatim.

The last one defines one (you can have many) process supervisor should manage.

It means it will run the command:

```
python manage.py start
```

In the directory, with the environment and the user you defined.

This command will be ran as a daemon, in the background.

autostart and *autorestart* just make it fire and forget: the site will always be running, even it crashes temporarily or if you restart the machine.

The first time you run supervisor, pass it the configuration file:

```
supervisord -c /path/to/supervisor.ini
```

Then you can manage the process by running:

```
supervisorctl -c /path/to/supervisor.ini
```

It will start a shell from which you can start/stop/restart the service.

You can read all errors that might occur from /tmp/lemur.log.

Periodic Tasks

Lemur contains a few tasks that are run and scheduled basis, currently the recommend way to run these tasks is to create a cron job that runs the commands.

There are currently three commands that could/should be run on a periodic basis:

- *notify*
- *check_revoked*
- *sync*

How often you run these commands is largely up to the user. *notify* and *check_revoked* are typically run at least once a day. *sync* is typically run every 15 minutes.

Example cron entries:

```
0 22 * * * lemuruser export LEMUR_CONF=/Users/me/.lemur/lemur.conf.py; /www/lemur/bin/
↪lemur notify expirations
*/15 * * * * lemuruser export LEMUR_CONF=/Users/me/.lemur/lemur.conf.py; /www/lemur/
↪bin/lemur source sync -s all
0 22 * * * lemuruser export LEMUR_CONF=/Users/me/.lemur/lemur.conf.py; /www/lemur/bin/
↪lemur certificate check_revoked
```

CHAPTER 2

User Guide

User Guide

These guides are quick tutorials on how to perform basic tasks in Lemur.

Create a New Authority

Before Lemur can issue certificates you must configure the authority you wish use. Lemur itself does not issue certificates, it relies on external CAs and the plugins associated with those CAs to create the certificate that Lemur can then manage.

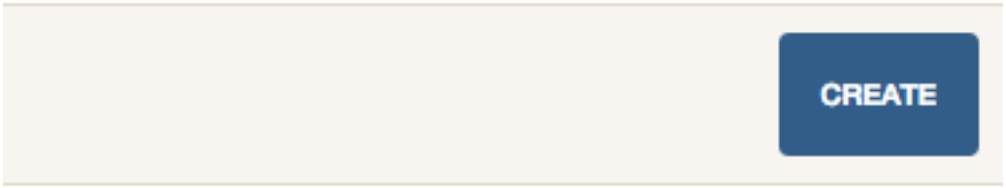

Fig. 2.1: In the authority table select “Create”

Create a New Certificate

Import an Existing Certificate

Create a New User

Create a New Role

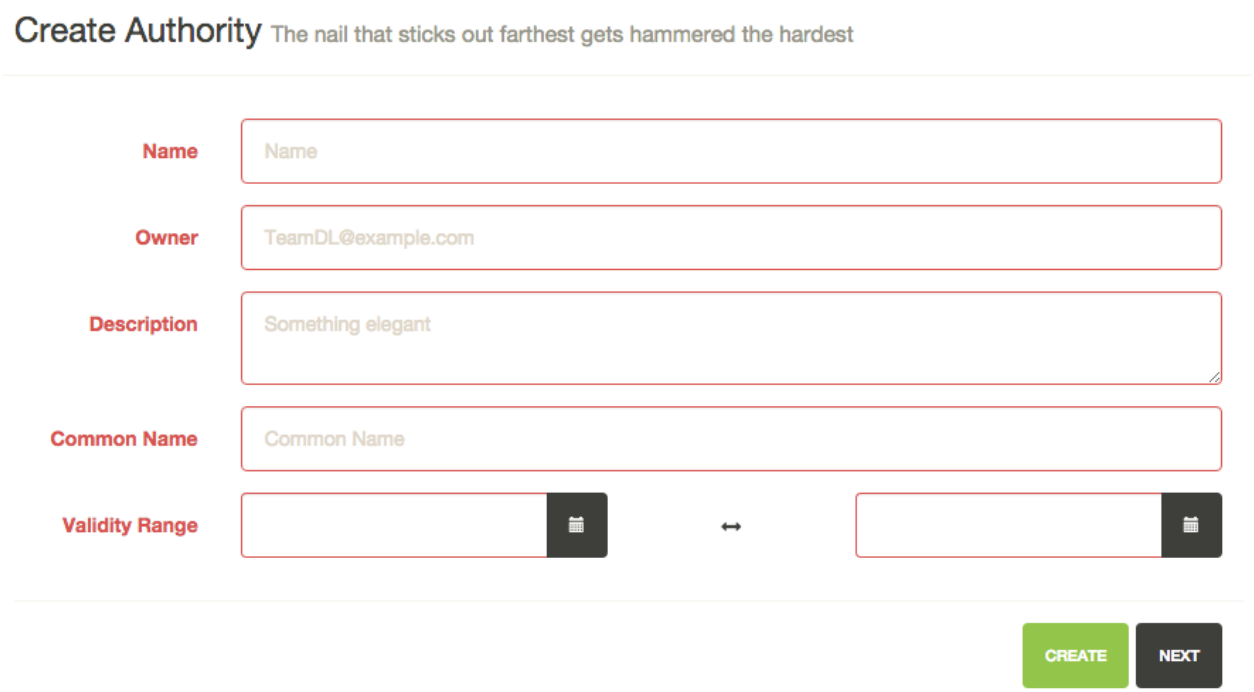

Fig. 2.2: Enter an authority name and short description about the authority. Enter an owner, and certificate common name. Depending on the authority and the authority/issuer plugin these values may or may not be used.

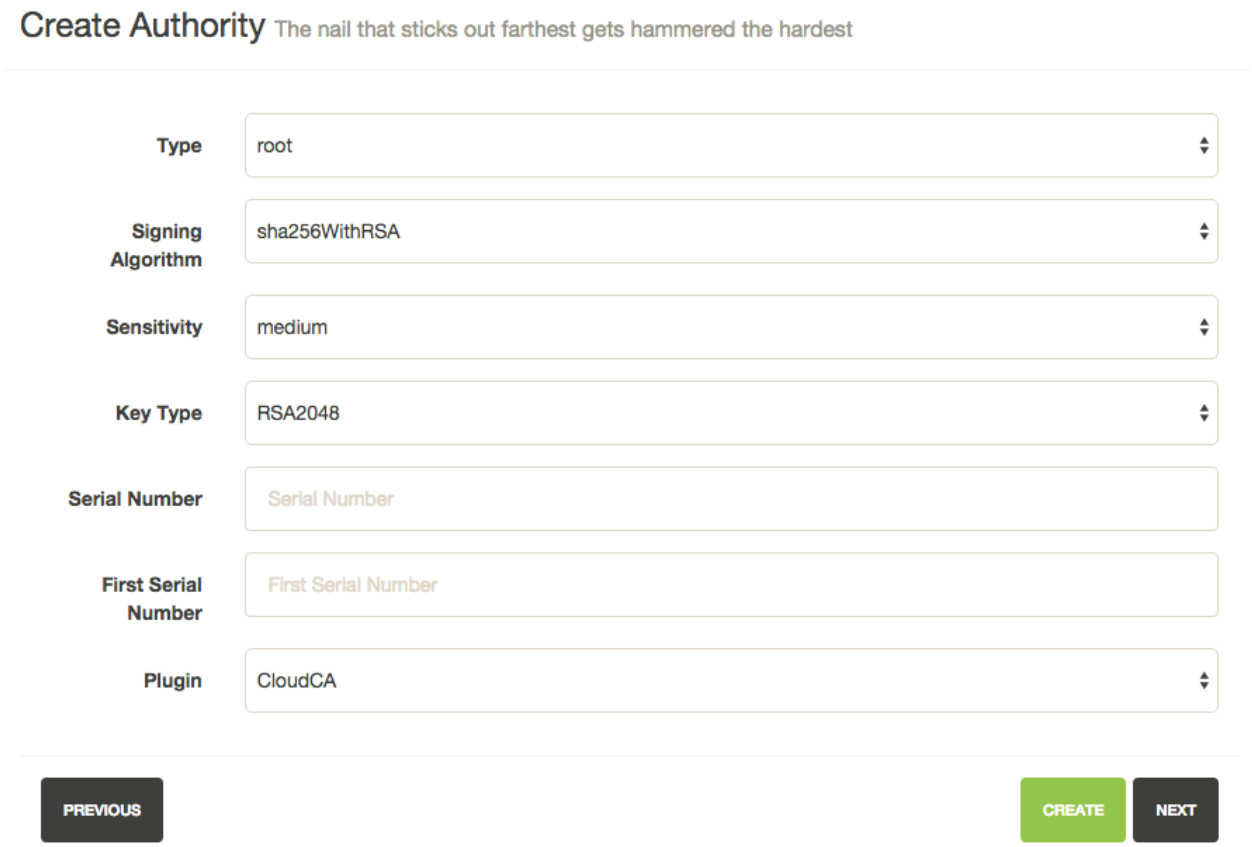

Fig. 2.3: Again how many of these values get used largely depends on the underlying plugin. It is important to make sure you select the right plugin that you wish to use.

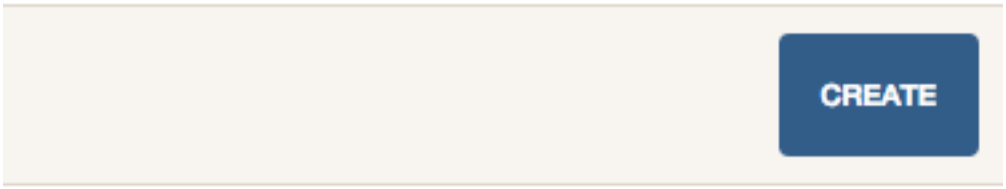

Fig. 2.4: In the certificate table select “Create”

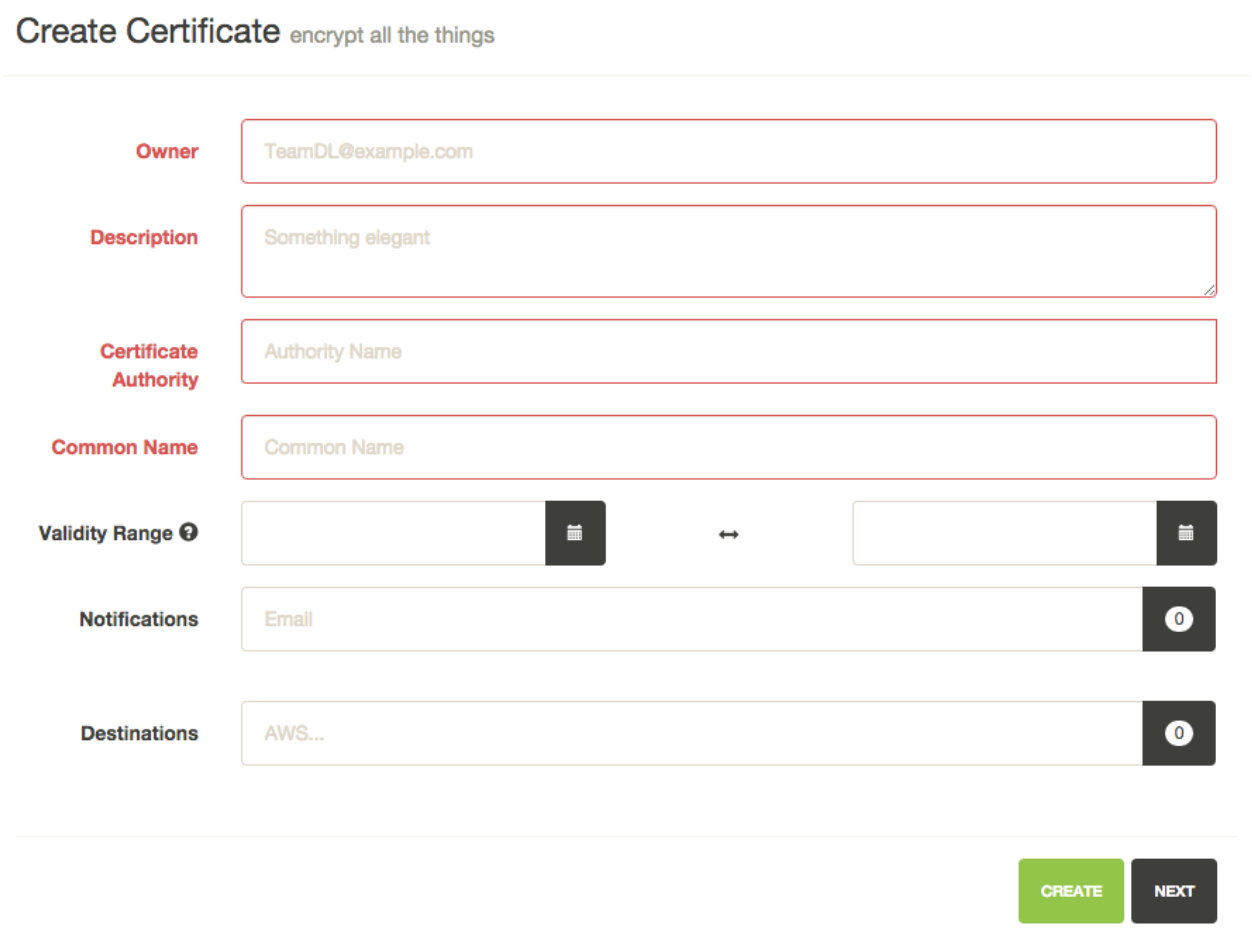

Fig. 2.5: Enter an owner, short description and the authority you wish to issue this certificate. Enter a common name into the certificate, if no validity range is selected two years is the default.

You can add notification options and upload the created certificate to a destination, both of these are editable features and can be changed after the certificate has been created.

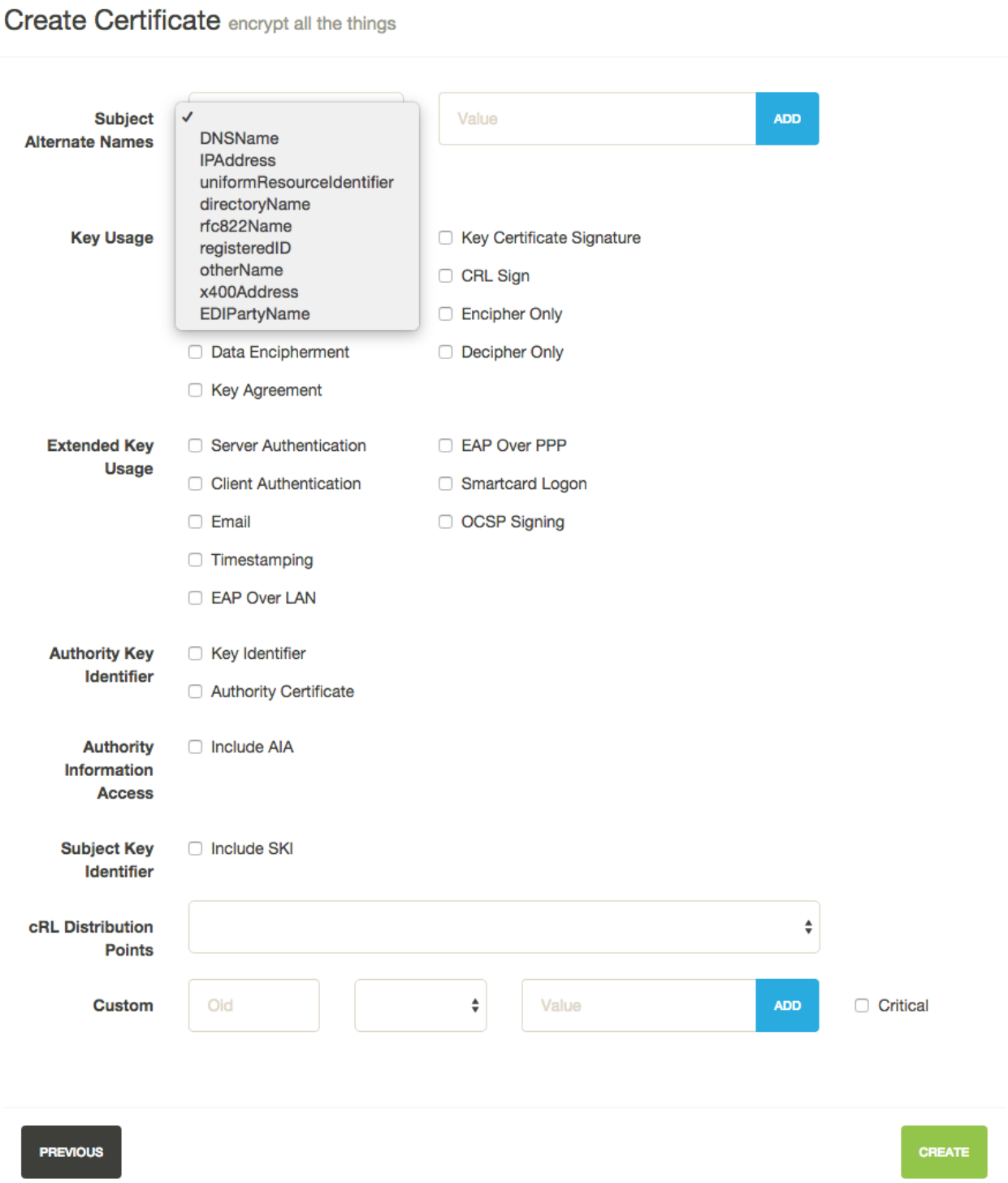

Fig. 2.6: These options are typically for advanced users, the one exception is the *Subject Alternate Names* or SAN. For certificates that need to include more than one domains, the first domain is the Common Name and all other domains are added here as DNSName entries.

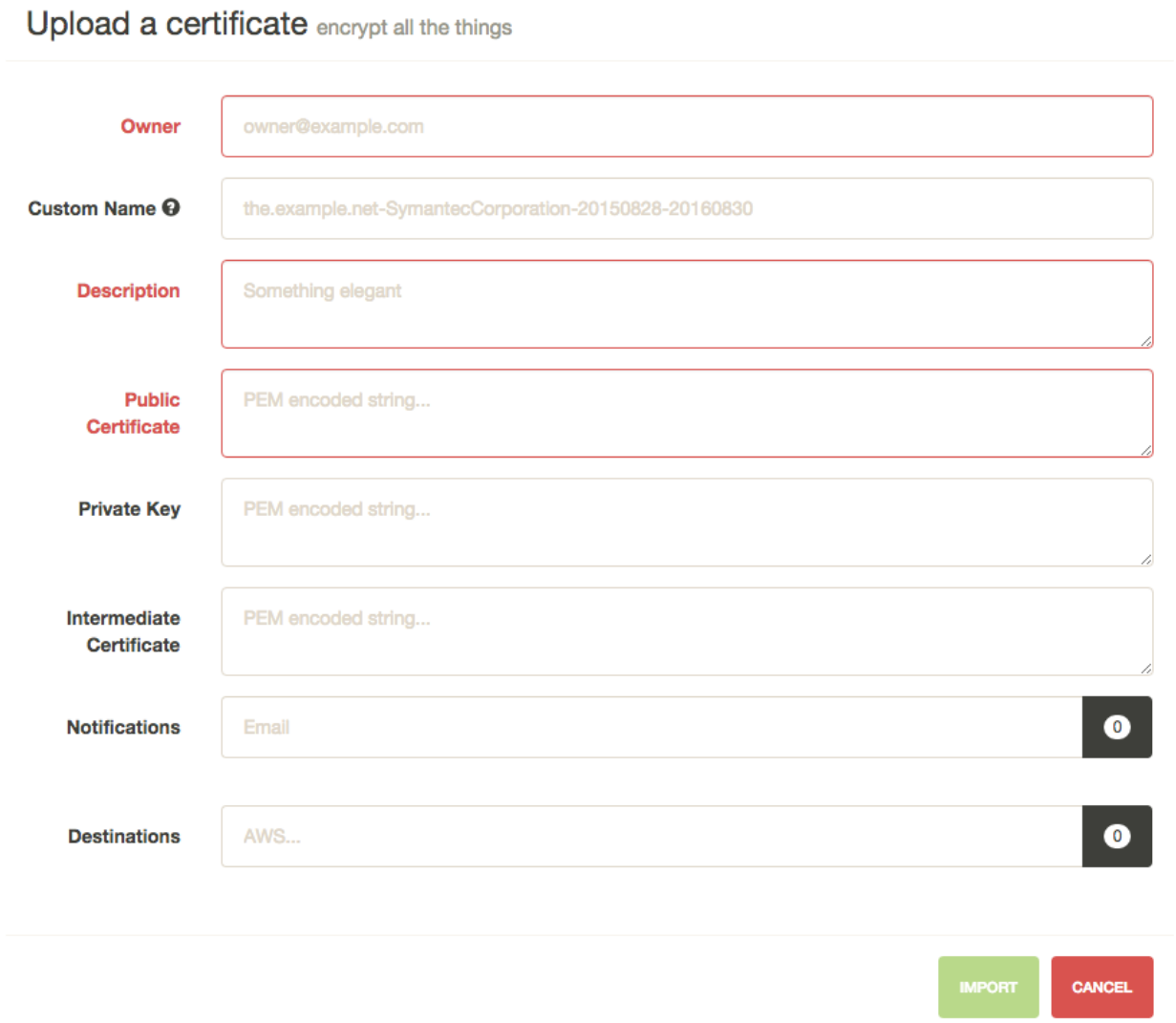

Fig. 2.7: Enter an owner, short description and public certificate. If there are intermediates and private keys Lemur will track them just as it does if the certificate were created through Lemur. Lemur generates a certificate name but you can override that by passing a value to the *Custom Name* field.

You can add notification options and upload the created certificate to a destination, both of these are editable features and can be changed after the certificate has been created.

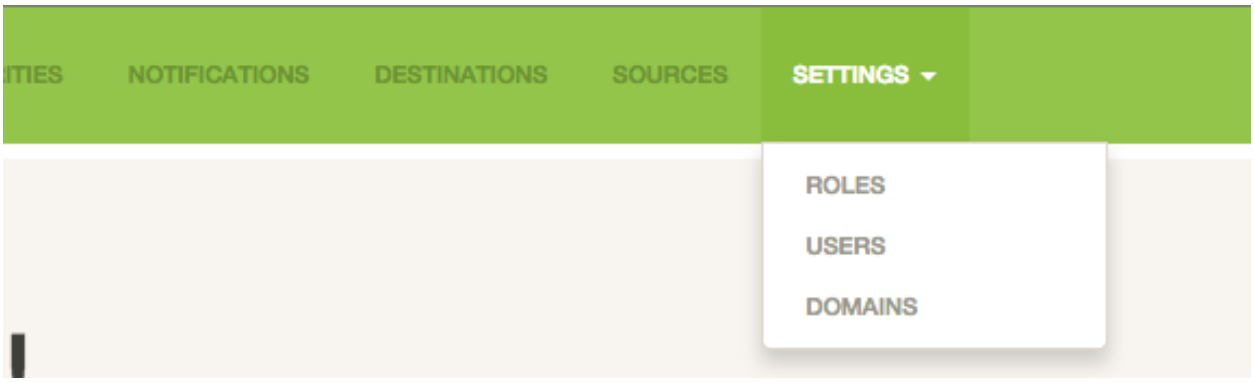

Fig. 2.8: From the settings dropdown select “Users”

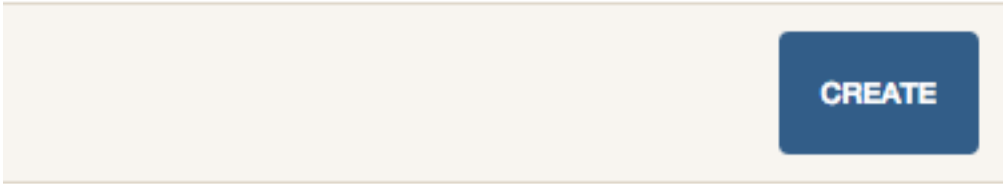

Fig. 2.9: In the user table select “Create”

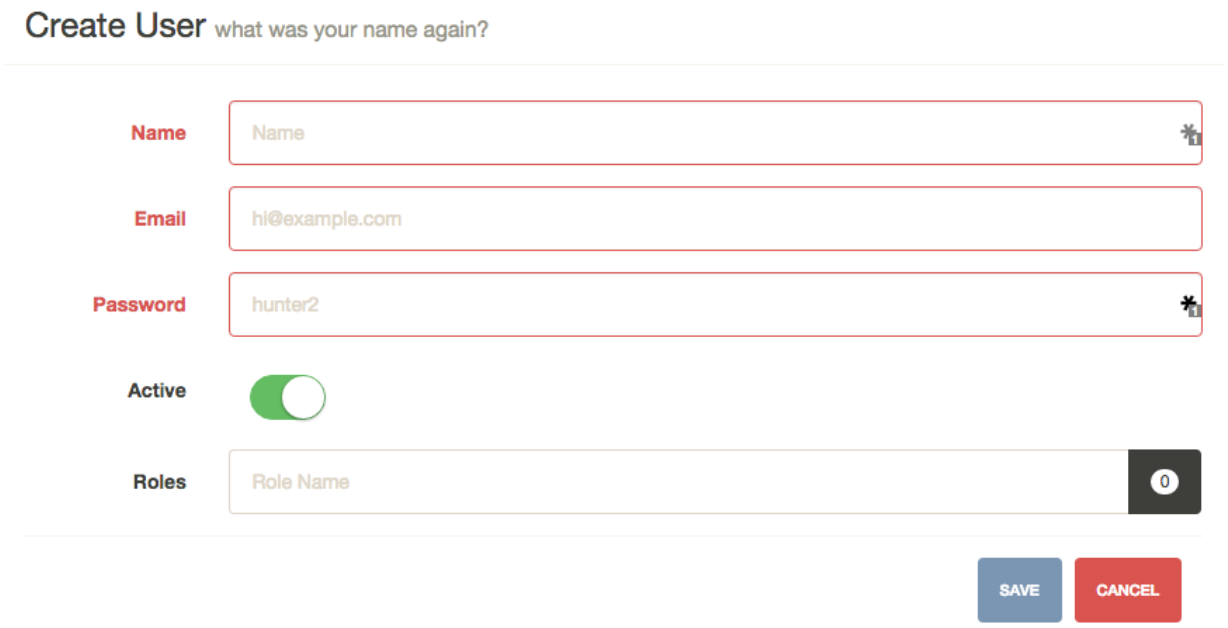

Fig. 2.10: Enter the username, email and password for the user. You can also assign any roles that the user will need when they login. While there is no deletion (we want to track creators forever) you can mark a user as 'Inactive' that will not allow them to login to Lemur.

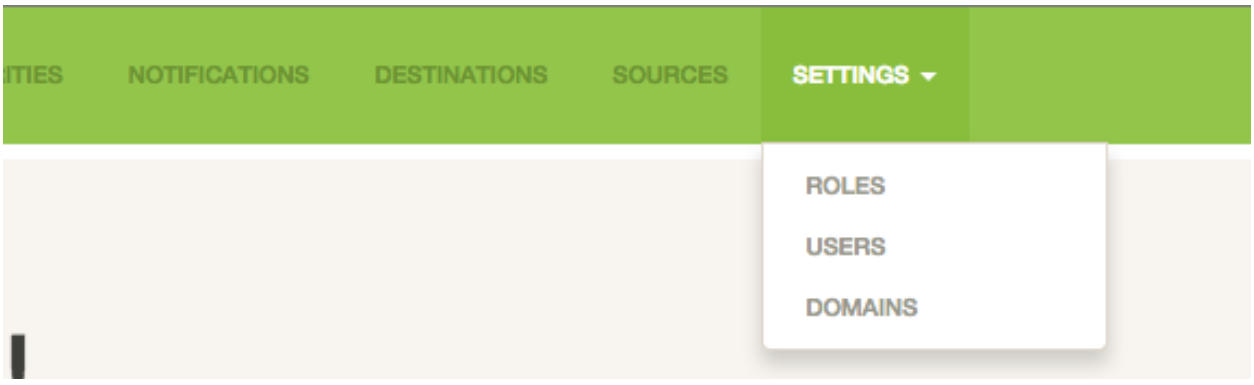

Fig. 2.11: From the settings dropdown select "Roles"

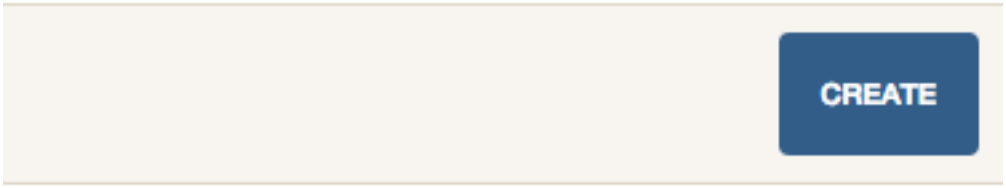

Fig. 2.12: In the role table select “Create”

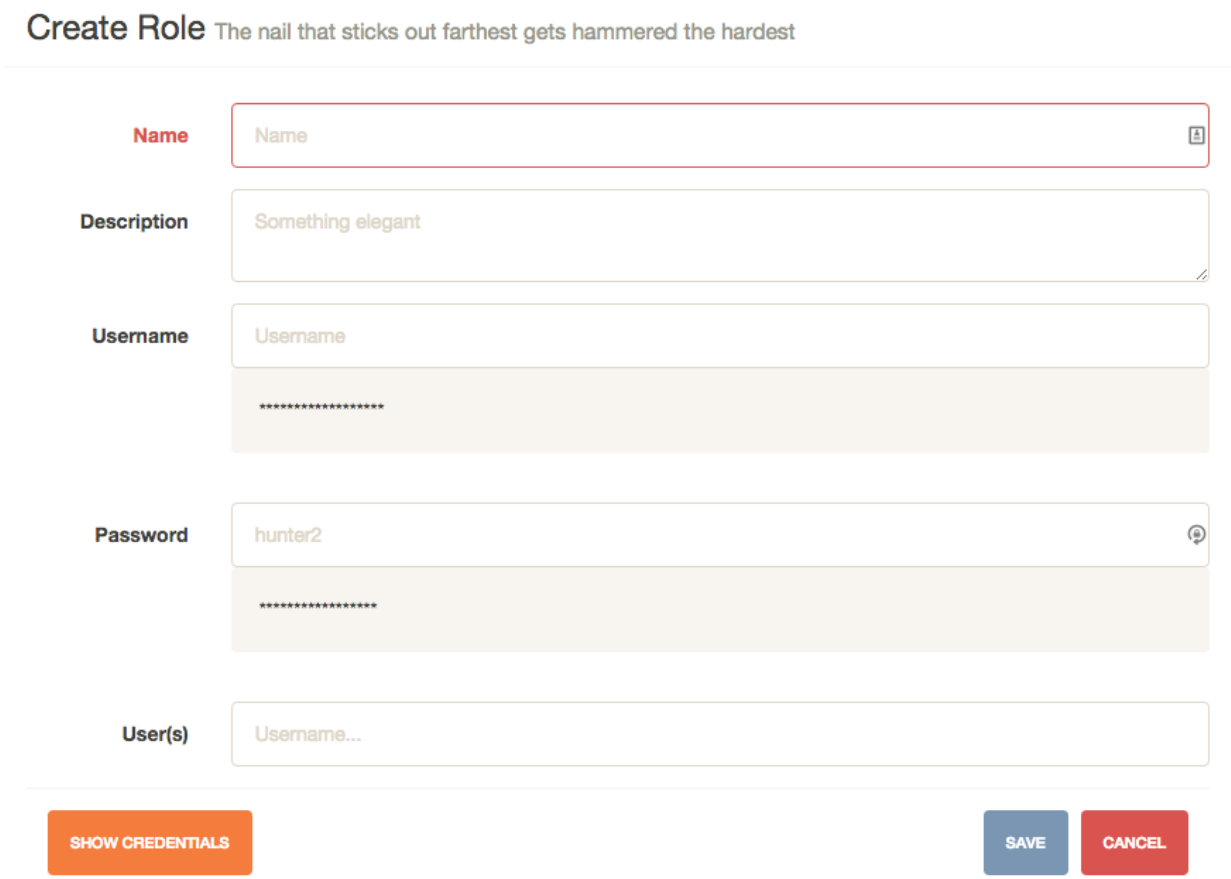

Fig. 2.13: Enter a role name and short description about the role. You can optionally store a user/password on the role. This is useful if your authority require specific roles. You can then accurately map those roles onto Lemur users. Also optional you can assign users to your new role.

CHAPTER 3

Administration

Configuration

Warning: There are many secrets that Lemur uses that must be protected. All of these options are set via the Lemur configuration file. It is highly advised that you do not store your secrets in this file! Lemur provides functions that allow you to encrypt files at rest and decrypt them when it's time for deployment. See *Credential Management* for more information.

Note: All configuration values are python strings unless otherwise noted.

Basic Configuration

LOG_LEVEL

```
LOG_LEVEL = "DEBUG"
```

LOG_FILE

```
LOG_FILE = "/logs/lemur/lemur-test.log"
```

debug
Sets the flask debug flag to true (if supported by the webserver)

```
debug = False
```

Warning: This should never be used in a production environment as it exposes Lemur to remote code execution through the debug console.

CORS
Allows for cross domain requests, this is most commonly used for development but could be use in production if you decided to host the webUI on a different domain than the server.

Use this cautiously, if you're not sure. Set it to *False*

```
CORS = False
```

SQLALCHEMY_DATABASE_URI

If you have ever used sqlalchemy before this is the standard connection string used. Lemur uses a postgres database and the connection string would look something like:

```
SQLALCHEMY_DATABASE_URI = 'postgresql://<user>:<password>@<hostname>:5432/lemur'
```

LEMUR_ALLOW_WEEKEND_EXPIRATION

Specifies whether to allow certificates created by Lemur to expire on weekends. Default is True.

LEMUR_RESTRICTED_DOMAINS

This allows the administrator to mark a subset of domains or domains matching a particular regex as *restricted*. This means that only an administrator is allows to issue the domains in question.

LEMUR_TOKEN_SECRET

The TOKEN_SECRET is the secret used to create JWT tokens that are given out to users. This should be securely generated and kept private.

```
LEMUR_TOKEN_SECRET = 'supersecret'
```

An example of how you might generate a random string:

```
>>> import random
>>> secret_key = ''.join(random.choice(string.ascii_uppercase) for x in range(6))
>>> secret_key = secret_key + ''.join(random.choice("~!@#$%^&*()_+") for x in␣
↪range(6))
>>> secret_key = secret_key + ''.join(random.choice(string.ascii_lowercase) for x␣
↪in range(6))
>>> secret_key = secret_key + ''.join(random.choice(string.digits) for x in␣
↪range(6))
```

LEMUR_ENCRYPTION_KEYS

The LEMUR_ENCRYPTION_KEYS is used to encrypt data at rest within Lemur's database. Without a key Lemur will refuse to start. Multiple keys can be provided to facilitate key rotation. The first key in the list is used for encryption and all keys are tried for decryption until one works. Each key must be 32 URL safe base-64 encoded bytes.

Running lemur create_config will securely generate a key for your configuration file. If you would like to generate your own, we recommend the following method:

```
>>> import os
>>> import base64
>>> base64.urlsafe_b64encode(os.urandom(32))
```

```
LEMUR_ENCRYPTION_KEYS = ['1YeftooSbxCiX2zo8m1lXtpvQjy27smZcUUaGmffhMY=',
↪'LAfQt6yrkLqOK5lwpvQcT4jf2zdeTQJV1uYeh9coT5s=']
```

Certificate Default Options

Lemur allows you to fine tune your certificates to your organization. The following defaults are presented in the UI and are used when Lemur creates the CSR for your certificates.

LEMUR_DEFAULT_COUNTRY

```
LEMUR_DEFAULT_COUNTRY = "US"
```

LEMUR_DEFAULT_STATE

```
LEMUR_DEFAULT_STATE = "California"
```

LEMUR_DEFAULT_LOCATION

```
LEMUR_DEFAULT_LOCATION = "Los Gatos"
```

LEMUR_DEFAULT_ORGANIZATION

```
LEMUR_DEFAULT_ORGANIZATION = "Netflix"
```

LEMUR_DEFAULT_ORGANIZATIONAL_UNIT

```
LEMUR_DEFAULT_ORGANIZATIONAL_UNIT = "Operations"
```

LEMUR_DEFAULT_ISSUER_PLUGIN

```
LEMUR_DEFAULT_ISSUER_PLUGIN = "verisign-issuer"
```

LEMUR_DEFAULT_AUTHORITY

```
LEMUR_DEFAULT_AUTHORITY = "verisign"
```

Notification Options

Lemur currently has very basic support for notifications. Currently only expiration notifications are supported. Actual notification is handled by the notification plugins that you have configured. Lemur ships with the 'Email' notification that allows expiration emails to be sent to subscribers.

Templates for expiration emails are located under *lemur/plugins/lemur_email/templates* and can be modified for your needs. Notifications are sent to the certificate creator, owner and security team as specified by the *LEMUR_SECURITY_TEAM_EMAIL* configuration parameter.

Certificates marked as inactive will **not** be notified of upcoming expiration. This enables a user to essentially silence the expiration. If a certificate is active and is expiring the above will be notified according to the *LEMUR_DEFAULT_EXPIRATION_NOTIFICATION_INTERVALS* or 30, 15, 2 days before expiration if no intervals are set.

Lemur supports sending certification expiration notifications through SES and SMTP.

LEMUR_EMAIL_SENDER
Specifies which service will be delivering notification emails. Valid values are *SMTP* or *SES*

Note: If using SMTP as your provider you will need to define additional configuration options as specified by Flask-Mail. See: Flask-Mail

If you are using SES the email specified by the *LEMUR_MAIL* configuration will need to be verified by AWS before you can send any mail. See: Verifying Email Address in Amazon SES

LEMUR_EMAIL

Lemur sender's email

```
LEMUR_EMAIL = 'lemur.example.com'
```

LEMUR_SECURITY_TEAM_EMAIL

This is an email or list of emails that should be notified when a certificate is expiring. It is also the contact email address for any discovered certificate.

```
LEMUR_SECURITY_TEAM_EMAIL = ['security@example.com']
```

LEMUR_DEFAULT_EXPIRATION_NOTIFICATION_INTERVALS

Lemur notification intervals

```
LEMUR_DEFAULT_EXPIRATION_NOTIFICATION_INTERVALS = [30, 15, 2]
```

Authentication Options

Lemur currently supports Basic Authentication, Ping OAuth2, and Google out of the box. Additional flows can be added relatively easily. If you are not using an authentication provider you do not need to configure any of these options.

For more information about how to use social logins, see: Satellizer

ACTIVE_PROVIDERS

```
ACTIVE_PROVIDERS = ["ping", "google", "oauth2"]
```

PING_SECRET

```
PING_SECRET = 'somethingsecret'
```

PING_ACCESS_TOKEN_URL

```
PING_ACCESS_TOKEN_URL = "https://<yourpingserver>/as/token.oauth2"
```

PING_USER_API_URL

```
PING_USER_API_URL = "https://<yourpingserver>/idp/userinfo.openid"
```

PING_JWKS_URL

```
PING_JWKS_URL = "https://<yourpingserver>/pf/JWKS"
```

PING_NAME

```
PING_NAME = "Example Oauth2 Provider"
```

PING_CLIENT_ID

```
PING_CLIENT_ID = "client-id"
```

PING_REDIRECT_URI

```
PING_REDIRECT_URI = "https://<yourlemurserver>/api/1/auth/ping"
```

PING_AUTH_ENDPOINT

```
PING_AUTH_ENDPOINT = "https://<yourpingserver>/oauth2/authorize"
```

OAUTH2_SECRET

```
OAUTH2_SECRET = 'somethingsecret'
```

OAUTH2_ACCESS_TOKEN_URL

```
OAUTH2_ACCESS_TOKEN_URL = "https://<youroauthserver> /oauth2/v1/authorize"
```

OAUTH2_USER_API_URL

```
OAUTH2_USER_API_URL = "https://<youroauthserver>/oauth2/v1/userinfo"
```

OAUTH2_JWKS_URL

```
OAUTH2_JWKS_URL = "https://<youroauthserver>/oauth2/v1/keys"
```

OAUTH2_NAME

```
OAUTH2_NAME = "Example Oauth2 Provider"
```

OAUTH2_CLIENT_ID

```
OAUTH2_CLIENT_ID = "client-id"
```

OAUTH2_REDIRECT_URI

```
OAUTH2_REDIRECT_URI = "https://<yourlemurserver>/api/1/auth/oauth2"
```

OAUTH2_AUTH_ENDPOINT

```
OAUTH2_AUTH_ENDPOINT = "https://<youroauthserver>/oauth2/v1/authorize"
```

GOOGLE_CLIENT_ID

```
GOOGLE_CLIENT_ID = "client-id"
```

GOOGLE_SECRET

```
GOOGLE_SECRET = "somethingsecret"
```

Plugin Specific Options

Verisign Issuer Plugin

Authorities will each have their own configuration options. There is currently just one plugin bundled with Lemur, Verisign/Symantec. Additional plugins may define additional options. Refer to the plugin's own documentation for those plugins.

VERISIGN_URL

This is the url for the Verisign API

VERISIGN_PEM_PATH

This is the path to the mutual TLS certificate used for communicating with Verisign

VERISIGN_FIRST_NAME

This is the first name to be used when requesting the certificate

VERISIGN_LAST_NAME

This is the last name to be used when requesting the certificate

VERISIGN_EMAIL

This is the email to be used when requesting the certificate

VERISIGN_INTERMEDIATE

This is the intermediate to be used for your CA chain

VERISIGN_ROOT

This is the root to be used for your CA chain

Digicert Issuer Plugin

The following configuration properties are required to use the Digicert issuer plugin.

DIGICERT_URL

This is the url for the Digicert API (e.g. https://www.digicert.com)

DIGICERT_API_KEY

This is the Digicert API key

DIGICERT_ORG_ID

This is the Digicert organization ID tied to your API key

DIGICERT_ROOT

This is the root to be used for your CA chain

DIGICERT_DEFAULT_VALIDITY

This is the default validity (in years), if no end date is specified. (Default: 1)

CFSSL Issuer Plugin

The following configuration properties are required to use the CFSSL issuer plugin.

CFSSL_URL

This is the URL for the CFSSL API

CFSSL_ROOT

This is the root to be used for your CA chain

CFSSL_INTERMEDIATE

This is the intermediate to be used for your CA chain

AWS Source/Destination Plugin

In order for Lemur to manage its own account and other accounts we must ensure it has the correct AWS permissions.

Note: AWS usage is completely optional. Lemur can upload, find and manage TLS certificates in AWS. But is not required to do so.

Lemur's AWS plugin uses boto heavily to talk to all the AWS resources it manages. By default it uses the on-instance credentials to make the necessary calls.

In order to limit the permissions, we will create two new IAM roles for Lemur. You can name them whatever you would like but for example sake we will be calling them LemurInstanceProfile and Lemur.

Lemur uses to STS to talk to different accounts. For managing one account this isn't necessary but we will still use it so that we can easily add new accounts.

LemurInstanceProfile is the IAM role you will launch your instance with. It actually has almost no rights. In fact it should really only be able to use STS to assume role to the Lemur role.

Here are example policies for the LemurInstanceProfile:

SES-SendEmail

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ses:SendEmail"
      ],
      "Resource": "*"
    }
  ]
}
```

STS-AssumeRole

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action":
        "sts:AssumeRole",
      "Resource": "*"
    }
  ]
}
```

Next we will create the Lemur IAM role.

Note: The default IAM role that Lemur assumes into is called *Lemur*, if you need to change this ensure you set *LEMUR_INSTANCE_PROFILE* to your role name in the configuration.

Here is an example policy for Lemur:

IAM-ServerCertificate

```
{
    "Statement": [
                {
                      "Action": [
                           "iam:ListServerCertificates",
                           "iam:UpdateServerCertificate",
                           "iam:GetServerCertificate",
                           "iam:UploadServerCertificate"
                      ],
                      "Resource": [
                           "*"
                      ],
                      "Effect": "Allow",
                      "Sid": "Stmt1404836868000"
                }
            ]
}
```

```
{
    "Statement": [
                {
                      "Action": [
                           "elasticloadbalancing:DescribeInstanceHealth",
                           "elasticloadbalancing:DescribeLoadBalancerAttributes",
                           "elasticloadbalancing:DescribeLoadBalancerPolicyTypes",
                           "elasticloadbalancing:DescribeLoadBalancerPolicies",
                           "elasticloadbalancing:DescribeLoadBalancers",
                           "elasticloadbalancing:DeleteLoadBalancerListeners",
                           "elasticloadbalancing:CreateLoadBalancerListeners"
                      ],
                      "Resource": [
                           "*"
                      ],
                      "Effect": "Allow",
                      "Sid": "Stmt1404841912000"
                }
            ]
}
```

Once we have setup our accounts we need to ensure that we create a trust relationship so that LemurInstanceProfile

can assume the Lemur role.

In the AWS console select the Lemur IAM role and select the Trust Relationships tab and click Edit Trust Relationship

Below is an example policy:

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<awsaccountnumber>:role/LemurInstanceProfile",
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To add another account we go to the new account and create a new Lemur IAM role with the same policy as above.

Then we would go to the account that Lemur is running is and edit the trust relationship policy.

An example policy:

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<awsaccountnumber>:role/LemurInstanceProfile",
          "arn:aws:iam::<awsaccountnumber1>:role/LemurInstanceProfile",
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Lemur has built in support for sending it's certificate notifications via Amazon's simple email service (SES). To force Lemur to use SES ensure you are the running as the IAM role defined above and that you have followed the steps outlined in Amazon's documentation Setting up Amazon SES

The configuration:

```
LEMUR_MAIL = 'lemur.example.com'
```

Will be the sender of all notifications, so ensure that it is verified with AWS.

SES if the default notification gateway and will be used unless SMTP settings are configured in the application configuration settings.

Command Line Interface

Lemur installs a command line script under the name `lemur` . This will allow you to perform most required operations that are unachievable within the web UI.

If you're using a non-standard configuration location, you'll need to prefix every command with –config (excluding create_config, which is a special case). For example:

```
lemur --config=/etc/lemur.conf.py help
```

For a list of commands, you can also use `lemur help` , or `lemur [command] --help` for help on a specific command.

Note: The script is powered by a library called Flask-Script

Builtin Commands

All commands default to *~/.lemur/lemur.conf.py* if a configuration is not specified.

`create_config`
Creates a default configuration file for Lemur.

Path defaults to `~/.lemur/lemur.config.py`

```
lemur create_config .
```

Note: This command is a special case and does not depend on the configuration file being set.

`init`
Initializes the configuration file for Lemur.

```
lemur -c /etc/lemur.conf.py init
```

`start`
Starts a Lemur service. You can also pass any flag that Gunicorn uses to specify the webserver configuration.

```
lemur start -w 6 -b 127.0.0.1:8080
```

`db upgrade`
Performs any needed database migrations.

```
lemur db upgrade
```

`check_revoked`
Traverses every certificate that Lemur is aware of and attempts to understand its validity. It utilizes both OCSP and CRL. If Lemur is unable to come to a conclusion about a certificates validity its status is marked 'unknown'.

`sync`
Sync attempts to discover certificates in the environment that were not created by Lemur. If you wish to only sync a few sources you can pass a comma delimited list of sources to sync.

```
lemur sync -s source1,source2
```

Additionally you can also list the available sources that Lemur can sync.

```
lemur sync
```

notify
Will traverse all current notifications and see if any of them need to be triggered.

```
lemur notify
```

Sub-commands

Lemur includes several sub-commands for interacting with Lemur such as creating new users, creating new roles and even issuing certificates.

The best way to discover these commands is by using the built in help pages

```
lemur --help
```

and to get help on sub-commands

```
lemur certificates --help
```

Upgrading Lemur

To upgrade Lemur to the newest release you will need to ensure you have the latest code and have run any needed database migrations.

To get the latest code from github run

```
cd <lemur-source-directory>
git pull -t <version>
python setup.py develop
```

Note: It's important to grab the latest release by specifying the release tag. This tags denote stable versions of Lemur. If you want to try the bleeding edge version of Lemur you can by using the master branch.

After you have the latest version of the Lemur code base you must run any needed database migrations. To run migrations

```
cd <lemur-source-directory>/lemur
lemur db upgrade
```

This will ensure that any needed tables or columns are created or destroyed.

Note: Internally, this uses Alembic to manage database migrations.

Note: By default Alembic looks for the *migrations* folder in the current working directory.The migrations folder is located under *<LEMUR_HOME>/lemur/migrations* if you are running the lemur command from any location besides *<LEMUR_HOME>/lemur* you will need to pass the *-d* flag to specify the absolute file path to the *migrations* folder.

Plugins

There are several interfaces currently available to extend Lemur. These are a work in progress and the API is not frozen.

Lemur includes several plugins by default. Including extensive support for AWS, VeriSign/Symantec.

Verisign/Symantec

Authors Kevin Glisson <kglisson@netflix.com>

Type Issuer

Description Basic support for the VICE 2.0 API

Cryptography

Authors Kevin Glisson <kglisson@netflix.com>, Mikhail Khodorovskiy <mikhail.khodorovskiy@jivesoftware.com>

Type Issuer

Description Toy certificate authority that creates self-signed certificate authorities. Allows for the creation of arbitrary authorities and end-entity certificates. This is *not* recommended for production use.

Acme

Authors Kevin Glisson <kglisson@netflix.com>, Mikhail Khodorovskiy <mikhail.khodorovskiy@jivesoftware.com>

Type Issuer

Description Adds support for the ACME protocol (including LetsEncrypt) with domain validation being handled Route53.

Atlas

Authors Kevin Glisson <kglisson@netflix.com>

Type Metric

Description Adds basic support for the Atlas telemetry system.

Email

Authors Kevin Glisson <kglisson@netflix.com>

Type Notification

Description Adds support for basic email notifications via SES.

Slack

Authors Harm Weites <harm@weites.com>

Type Notification

Description Adds support for slack notifications.

AWS

Authors Kevin Glisson <kglisson@netflix.com>

Type Source

Description Uses AWS IAM as a source of certificates to manage. Supports a multi-account deployment.

AWS

Authors Kevin Glisson <kglisson@netflix.com>

Type Destination

Description Uses AWS IAM as a destination for Lemur generated certificates. Support a multi-account deployment.

Kubernetes

Authors Mikhail Khodorovskiy <mikhail.khodorovskiy@jivesoftware.com>

Type Destination

Description Allows Lemur to upload generated certificates to the Kubernetes certificate store.

Java

Authors Kevin Glisson <kglisson@netflix.com>

Type Export

Description Generates java compatible .jks keystores and truststores from Lemur managed certificates.

Openssl

Authors Kevin Glisson <kglisson@netflix.com>

Type Export

Description Leverages Openssl to support additional export formats (pkcs12)

CFSSL

Authors Charles Hendrie <chad.hendrie@thomsonreuters.com>

Type Issuer

Description Basic support for generating certificates from the private certificate authority CFSSL

3rd Party Plugins

The following plugins are available and maintained by members of the Lemur community:

Digicert

Authors Chris Dorros

Type Issuer

Description Adds support for basic Digicert

Links https://github.com/opendns/lemur-digicert

Have an extension that should be listed here? Submit a pull request and we'll get it added.

Want to create your own extension? See *Structure* to get started.

Identity and Access Management

Lemur uses a Role Based Access Control (RBAC) mechanism to control which users have access to which resources. When a user is first created in Lemur they can be assigned one or more roles. These roles are typically dynamically created depending on an external identity provider (Google, LDAP, etc.), or are hardcoded within Lemur and associated with special meaning.

Within Lemur there are three main permissions: AdminPermission, CreatorPermission, OwnerPermission. Sub-permissions such as ViewPrivateKeyPermission are compositions of these three main Permissions.

Lets take a look at how these permissions are used:

Each *Authority* has a set of roles associated with it. If a user is also associated with the same roles that the *Authority* is associated with, Lemur allows that user to user/view/update that *Authority*.

This RBAC is also used when determining which users can access which certificate private key. Lemur's current permission structure is setup such that if the user is a *Creator* or *Owner* of a given certificate they are allow to view that private key. Owners can also be a role name, such that any user with the same role as owner will be allowed to view the private key information.

These permissions are applied to the user upon login and refreshed on every request.

See also:

Flask-Principal

CHAPTER 4

Developers

Contributing

Want to contribute back to Lemur? This page describes the general development flow, our philosophy, the test suite, and issue tracking.

Documentation

If you're looking to help document Lemur, you can get set up with Sphinx, our documentation tool, but first you will want to make sure you have a few things on your local system:

- python-dev (if you're on OS X, you already have this)
- pip
- virtualenvwrapper

Once you've got all that, the rest is simple:

```
# If you have a fork, you'll want to clone it instead
git clone git://github.com/netflix/lemur.git

# Create a python virtualenv
mkvirtualenv lemur

# Make the magic happen
make dev-docs
```

Running `make dev-docs` will install the basic requirements to get Sphinx running.

Building Documentation

Inside the `docs` directory, you can run `make` to build the documentation. See `make help` for available options and the Sphinx Documentation for more information.

Developing Against HEAD

We try to make it easy to get up and running in a development environment using a git checkout of Lemur. You'll want to make sure you have a few things on your local system first:

- python-dev (if you're on OS X, you already have this)
- pip
- virtualenv (ideally virtualenvwrapper)
- node.js (for npm and building css/javascript)
- (Optional) PostgreSQL

Once you've got all that, the rest is simple:

```
# If you have a fork, you'll want to clone it instead
git clone git://github.com/lemur/lemur.git

# Create a python virtualenv
mkvirtualenv lemur

# Make the magic happen
make
```

Running `make` will do several things, including:

- Setting up any submodules (including Bootstrap)
- Installing Python requirements
- Installing NPM requirements

Note: You will want to store your virtualenv out of the `lemur` directory you cloned above, otherwise `make` will fail.

Create a default Lemur configuration just as if this were a production instance:

```
lemur init
```

You'll likely want to make some changes to the default configuration (we recommend developing against Postgres, for example). Once done, migrate your database using the following command:

```
lemur upgrade
```

Note: The `upgrade` shortcut is simply a shortcut to Alembic's upgrade command.

Running tests with Docker and docker-compose

Alternatively you can use Docker and docker-compose for running the tests with `docker-compose run test`.

Coding Standards

Lemur follows the guidelines laid out in pep8 with a little bit of flexibility on things like line length. We always give way for the Zen of Python. We also use strict mode for JavaScript, enforced by jshint.

You can run all linters with `make lint`, or respectively `lint-python` or `lint-js`.

Spacing

Python: 4 Spaces

JavaScript: 2 Spaces

CSS: 2 Spaces

HTML: 2 Spaces

Git hooks

To help developers maintain the above standards, Lemur includes a configuration file for Yelp's pre-commit. This is an optional dependency and is not required in order to contribute to Lemur.

Running the Test Suite

The test suite consists of multiple parts, testing both the Python and JavaScript components in Lemur. If you've setup your environment correctly, you can run the entire suite with the following command:

```
make test
```

If you only need to run the Python tests, you can do so with `make test-python` , as well as `test-js` for the JavaScript tests.

You'll notice that the test suite is structured based on where the code lives, and strongly encourages using the mock library to drive more accurate individual tests.

Note: We use py.test for the Python test suite, and a combination of phantomjs and jasmine for the JavaScript tests.

Static Media

Lemur uses a library that compiles it's static media assets (LESS and JS files) automatically. If you're developing using runserver you'll see changes happen not only in the original files, but also the minified or processed versions of the file.

If you've made changes and need to compile them by hand for any reason, you can do so by running:

```
lemur compilestatic
```

The minified and processed files should be committed alongside the unprocessed changes.

It's also important to note that Lemur's frontend and API are not tied together. The API does not serve any of the static assets, we rely on nginx or some other file server to server all of the static assets. During development that means we need an additional server to serve those static files for the GUI.

This is accomplished with a Gulp task:

```
./node_modules/.bin/gulp serve
```

The gulp task compiles all the JS/CSS/HTML files and opens the Lemur welcome page in your default browsers. Additionally any changes to made to the JS/CSS/HTML with be reloaded in your browsers.

Developing with Flask

Because Lemur is just Flask, you can use all of the standard Flask functionality. The only difference is you'll be accessing commands that would normally go through manage.py using the `lemur` CLI helper instead.

For example, you probably don't want to use `lemur start` for development, as it doesn't support anything like automatic reloading on code changes. For that you'd want to use the standard builtin `runserver` command:

```
lemur runserver
```

DDL (Schema Changes)

Schema changes should always introduce the new schema in a commit, and then introduce code relying on that schema in a followup commit. This also means that new columns must be NULLable.

Removing columns and tables requires a slightly more painful flow, and should resemble the follow multi-commit flow:

- Remove all references to the column or table (but don't remove the Model itself)
- Remove the model code
- Remove the table or column

Contributing Back Code

All patches should be sent as a pull request on GitHub, include tests, and documentation where needed. If you're fixing a bug or making a large change the patch **must** include test coverage.

Uncertain about how to write tests? Take a look at some existing tests that are similar to the code you're changing, and go from there.

You can see a list of open pull requests (pending changes) by visiting https://github.com/netflix/lemur/pulls

Pull requests should be against **master** and pass all TravisCI checks

Writing a Plugin

Several interfaces exist for extending Lemur:

- Issuer (lemur.plugins.base.issuer)
- Destination (lemur.plugins.base.destination)
- Source (lemur.plugins.base.source)
- Notification (lemur.plugins.base.notification)

Each interface has its own functions that will need to be defined in order for your plugin to work correctly. See *Plugin Interfaces* for details.

Structure

A plugins layout generally looks like the following:

```
setup.py
lemur_pluginname/
lemur_pluginname/__init__.py
lemur_pluginname/plugin.py
```

The `__init__.py` file should contain no plugin logic, and at most, a VERSION = 'x.x.x' line. For example, if you want to pull the version using pkg_resources (which is what we recommend), your file might contain:

```
try:
    VERSION = __import__('pkg_resources') \
        .get_distribution(__name__).version
except Exception as e:
    VERSION = 'unknown'
```

Inside of `plugin.py` , you'll declare your Plugin class:

```
import lemur_pluginname
from lemur.plugins.base.issuer import IssuerPlugin

class PluginName(IssuerPlugin):
    title = 'Plugin Name'
    slug = 'pluginname'
    description = 'My awesome plugin!'
    version = lemur_pluginname.VERSION

    author = 'Your Name'
    author_url = 'https://github.com/yourname/lemur_pluginname'

    def widget(self, request, group, **kwargs):
        return "<p>Absolutely useless widget</p>"
```

And you'll register it via `entry_points` in your `setup.py` :

```
setup(
    # ...
    entry_points={
       'lemur.plugins': [
            'pluginname = lemur_pluginname.issuers:PluginName'
        ],
    },
)
```

You can potentially package multiple plugin types in one package, say you want to create a source and destination plugins for the same third-party. To accomplish this simply alias the plugin in entry points to point at multiple plugins within your package:

```
setup(
    # ...
    entry_points={
        'lemur.plugins': [
            'pluginnamesource = lemur_pluginname.plugin:PluginNameSource',
            'pluginnamedestination = lemur_pluginname.plugin:PluginNameDestination'
        ],
    },
)
```

Once your plugin files are in place and the `/www/lemur/setup.py` file has been modified, you can load your plugin into your instance by reinstalling lemur:

```
(lemur)$cd /www/lemur
(lemur)$pip install -e .
```

That's it! Users will be able to install your plugin via `pip install <package name>`.

See also:

For more information about python packages see Python Packaging

See also:

For an example of a plugin operation outside of Lemur's core, see lemur-digicert

Plugin Interfaces

In order to use the interfaces all plugins are required to inherit and override unimplemented functions of the parent object.

Issuer

Issuer plugins are used when you have an external service that creates certificates or authorities. In the simple case the third party only issues certificates (Verisign, DigiCert, etc.).

If you have a third party or internal service that creates authorities (EJBCA, etc.), Lemur has you covered, it can treat any issuer plugin as both a source of creating new certificates as well as new authorities.

The *IssuerPlugin* exposes two functions:

```
def create_certificate(self, csr, issuer_options):
    # requests.get('a third party')
```

Lemur will pass a dictionary of all possible options for certificate creation. Including a valid CSR, and the raw options associated with the request.

If you wish to be able to create new authorities implement the following function and ensure that the ROOT_CERTIFICATE and the INTERMEDIATE_CERTIFICATE (if any) for the new authority is returned:

```
def create_authority(self, options):
    root_cert, intermediate_cert, username, password = request.get('a third party')

    # if your provider creates specific credentials for each authority you can␣
↪associated them with the role associated with the authority
    # these credentials will be provided along with any other options when a␣
↪certificate is created
    role = dict(username=username, password=password, name='generatedAuthority')
    return root_cert, intermediate_cert, [role]
```

Note: Lemur uses PEM formatted certificates as it's internal standard, if you receive certificates in other formats convert them to PEM before returning.

If instead you do not need need to generate authorities but instead use a static authority (Verisign, DigiCert), you can use publicly available constants:

```
def create_authority(self, options):
    # optionally associate a role with authority to control who can use it
    role = dict(username='', password='', name='exampleAuthority')
    # username and password don't really matter here because we do no need to␣
↪authenticate our authority against a third party
    return EXAMPLE_ROOT_CERTIFICATE, EXAMPLE_INTERMEDIATE_CERTIFICATE, [role]
```

Note: You do not need to associate roles to the authority at creation time as they can always be associated after the fact.

The *IssuerPlugin* doesn't have any options like Destination, Source, and Notification plugins. Essentially Lemur **should** already have any fields you might need to submit a request to a third party. If there are additional options you need in your plugin feel free to open an issue, or look into adding additional options to issuers yourself.

Destination

Destination plugins allow you to propagate certificates managed by Lemur to additional third parties. This provides flexibility when different orchestration systems have their own way of manage certificates or there is an existing system you wish to integrate with Lemur.

By default destination plugins have a private key requirement. If your plugin does not require a certificates private key mark *requires_key = False* in the plugins base class like so:

```
class MyDestinationPlugin(DestinationPlugin):
    requires_key = False
```

The DestinationPlugin requires only one function to be implemented:

```
def upload(self, name, body, private_key, cert_chain, options, **kwargs):
    # request.post('a third party')
```

Additionally the DestinationPlugin allows the plugin author to add additional options that can be used to help define sub-destinations.

For example, if we look at the aws-destination plugin we can see that it defines an *accountNumber* option:

```
options = [
    {
        'name': 'accountNumber',
        'type': 'int',
        'required': True,
        'validation': '/^[0-9]{12,12}$/',
        'helpMessage': 'Must be a valid AWS account number!',
    }
]
```

By defining an *accountNumber* we can make this plugin handle many N number of AWS accounts instead of just one.

The schema for defining plugin options are pretty straightforward:

- **Name**: name of the variable you wish to present the user, snake case (snakeCase) is preferred as Lemur will parse these and create pretty variable titles
- **Type there are currently four supported variable types**
  - **Int** creates an html integer box for the user to enter integers into

  - **Str** creates a html text input box
  - **Boolean** creates a checkbox for the user to signify truthiness
  - **Select creates a select box that gives the user a list of options**
    * When used a *available* key must be provided with a list of selectable options
- **Required** determines if this option is required, this **must be a boolean value**
- **Validation** simple JavaScript regular expression used to give the user an indication if the input value is valid
- **HelpMessage** simple string that provides more detail about the option

Note: DestinationPlugin, NotificationPlugin and SourcePlugin all support the option schema outlined above.

Notification

Lemur includes the ability to create Email notifications by **default**. These notifications currently come in the form of expiration notices. Lemur periodically checks certifications expiration dates and determines if a given certificate is eligible for notification. There are currently only two parameters used to determine if a certificate is eligible; validity expiration (date the certificate is no longer valid) and the number of days the current date (UTC) is from that expiration date.

There are currently two objects that available for notification plugins the first is *NotficationPlugin*. This is the base object for any notification within Lemur. Currently the only support notification type is an certificate expiration notification. If you are trying to create a new notification type (audit, failed logins, etc.) this would be the object to base your plugin on. You would also then need to build additional code to trigger the new notification type.

The second is *ExpirationNotificationPlugin*, this object inherits from *NotificationPlugin* object. You will most likely want to base your plugin on, if you want to add new channels for expiration notices (Slack, HipChat, Jira, etc.). It adds default options that are required by all expiration notifications (interval, unit). This interface expects for the child to define the following function:

```
def send(self, notification_type, message, targets, options, **kwargs):
    #  request.post("some alerting infrastructure")
```

Source

When building Lemur we realized that although it would be nice if every certificate went through Lemur to get issued, but this is not always be the case. Oftentimes there are third parties that will issue certificates on your behalf and these can get deployed to infrastructure without any interaction with Lemur. In an attempt to combat this and try to track every certificate, Lemur has a notion of certificate **Sources**. Lemur will contact the source at periodic intervals and attempt to **sync** against the source. This means downloading or discovering any certificate Lemur does not know about and adding the certificate to its inventory to be tracked and alerted on.

The *SourcePlugin* object has one default option of *pollRate*. This controls the number of seconds which to get new certificates.

Warning: Lemur currently has a very basic polling system of running a cron job every 15min to see which source plugins need to be run. A lock file is generated to guarantee that only one sync is running at a time. It also means that the minimum resolution of a source plugin poll rate is effectively 15min. You can always specify a faster cron job if you need a higher resolution sync job.

The *SourcePlugin* object requires implementation of one function:

```
def get_certificates(self, options, **kwargs):
    #  request.get("some source of certificates")
```

Note: Oftentimes to facilitate code re-use it makes sense put source and destination plugins into one package.

Export

Formats, formats and more formats. That's the current PKI landscape. See the always relevant xkcd. Thankfully Lemur supports the ability to output your certificates into whatever format you want. This integration comes by the way of Export plugins. Support is still new and evolving, the goal of these plugins is to return raw data in a new format that can then be used by any number of applications. Included in Lemur is the *JavaExportPlugin* which currently supports generating a Java Key Store (JKS) file for use in Java based applications.

The *ExportPlugin* object requires the implementation of one function:

```
def export(self, body, chain, key, options, **kwargs):
    # sys.call('openssl hokuspocus')
    # return "extension", passphrase, raw
```

Note: Support of various formats sometimes relies on external tools system calls. Always be mindful of sanitizing any input to these calls.

Testing

Lemur provides a basic py.test-based testing framework for extensions.

In a simple project, you'll need to do a few things to get it working:

setup.py

Augment your setup.py to ensure at least the following:

```
setup(
    # ...
    install_requires=[
        'lemur',
    ]
)
```

conftest.py

The `conftest.py` file is our main entry-point for py.test. We need to configure it to load the Lemur pytest configuration:

```
from lemur.tests.conftest import *  # noqa
```

Test Cases

You can now inherit from Lemur's core test classes. These are Django-based and ensure the database and other basic utilities are in a clean state:

```
import pytest
from lemur.tests.vectors import INTERNAL_CERTIFICATE_A_STR, INTERNAL_PRIVATE_KEY_A_STR

def test_export_keystore(app):
    from lemur.plugins.base import plugins
    p = plugins.get('java-keystore-jks')
    options = [{'name': 'passphrase', 'value': 'test1234'}]
    with pytest.raises(Exception):
        p.export(INTERNAL_CERTIFICATE_A_STR, "", "", options)

    raw = p.export(INTERNAL_CERTIFICATE_A_STR, "", INTERNAL_PRIVATE_KEY_A_STR,␣
↪options)
    assert raw != b""
```

Running Tests

Running tests follows the py.test standard. As long as your test files and methods are named appropriately (`test_filename.py` and `test_function()`) you can simply call out to py.test:

```
$ py.test -v
============================= test session starts ==============================
platform darwin -- Python 2.7.10, pytest-2.8.5, py-1.4.30, pluggy-0.3.1
cachedir: .cache
plugins: flask-0.10.0
collected 346 items

lemur/plugins/lemur_acme/tests/test_acme.py::test_get_certificates PASSED

=========================== 1 passed in 0.35 seconds ===========================
```

See also:

Lemur bundles several plugins that use the same interfaces mentioned above.

REST API

Lemur's front end is entirely API driven. Any action that you can accomplish via the UI can also be accomplished by the API. The following is documents and provides examples on how to make requests to the Lemur API.

Authentication

class `lemur.auth.views.` **Google**

Bases: `flask_restful.Resource`

endpoint = 'google'

mediatypes (*resource_cls*)

methods = ['POST']

post ( )

class lemur.auth.views. **Login**
Bases: flask_restful.Resource

Provides an endpoint for Lemur's basic authentication. It takes a username and password combination and returns a JWT token.

This token token is required for each API request and must be provided in the Authorization Header for the request.

```
Authorization:Bearer <token>
```

Tokens have a set expiration date. You can inspect the token expiration by base64 decoding the token and inspecting it's contents.

Note: It is recommended that the token expiration is fairly short lived (hours not days). This will largely depend on your uses cases but. It is important to not that there is currently no build in method to revoke a users token and force re-authentication.

---

endpoint = 'login'

mediatypes (*resource_cls*)

methods = ['POST']

post ( )

POST /auth/login
Login with username:password

Example request:

```
POST /auth/login HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "username": "test",
  "password": "test"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "token": "12343243243"
}
```

Parameters
- **username** – username
- **password** – password

Status Codes
- 401 Unauthorized – invalid credentials
- 200 OK – no error

class lemur.auth.views. **OAuth2**
Bases: flask_restful.Resource

endpoint = 'oauth2'

mediatypes (*resource_cls*)

methods = ['POST']

post ( )

class lemur.auth.views. **Ping**
Bases: flask_restful.Resource

This class serves as an example of how one might implement an SSO provider for use with Lemur. In this example we use an OpenIDConnect authentication flow, that is essentially OAuth2 underneath. If you have an OAuth2 provider you want to use Lemur there would be two steps:

1. Define your own class that inherits from flask.ext.restful.Resource and create the HTTP methods the provider uses for it's callbacks.
2. Add or change the Lemur AngularJS Configuration to point to your new provider

endpoint = 'ping'

mediatypes (*resource_cls*)

methods = ['POST']

post ( )

class lemur.auth.views. **Providers**
Bases: flask_restful.Resource

endpoint = 'providers'

get ( )

mediatypes (*resource_cls*)

methods = ['GET']

Destinations

class lemur.destinations.views. **CertificateDestinations**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificate/<int:certificate_id/destinations" endpoint

endpoint = 'certificateDestinations'

get (**args*, ***kwargs*)

GET /certificates/1/destinations
The current account list for a given certificates

Example request:

```
GET /certificates/1/destinations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "description": "test",
      "options": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "id": 4,
      "plugin": {
          "pluginOptions": [{
              "name": "accountNumber",
              "required": true,
              "value": "111111111111111",
              "helpMessage": "Must be a valid AWS account number!",
              "validation": "/^[0-9]{12,12}$/",
              "type": "str"
          }],
          "description": "Allow the uploading of certificates to AWS IAM",
          "slug": "aws-destination",
          "title": "AWS"
      },
      "label": "test546"
  }
  "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.destinations.views. **Destinations**

Bases: lemur.auth.service.AuthenticatedResource

delete (**args*, ***kw*)

endpoint = 'destination'

get (**args*, ***kwargs*)

GET /destinations/1

Get a specific account

Example request:

```
GET /destinations/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "test",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "111111111111111",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['DELETE', 'GET', 'PUT']

put (**args*, ***kw*)

PUT /destinations/1

Updates an account

Example request:

```
POST /destinations/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript


{
  "description": "test33",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "34324324",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "34324324",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "test",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "111111111111111",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
```

```
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Parameters
- **accountNumber** – aws account number
- **label** – human readable account label
- **description** – some description about the account

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

class lemur.destinations.views. **DestinationsList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'destinations' endpoint

endpoint = 'destinations'

get (**args*, ***kwargs*)

GET /destinations

The current account list

Example request:

```
GET /destinations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "description": "test",
      "options": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "id": 4,
      "plugin": {
          "pluginOptions": [{
              "name": "accountNumber",
              "required": true,
              "value": "111111111111111",
              "helpMessage": "Must be a valid AWS account number!",
```

```
                "validation": "/^[0-9]{12,12}$/",
                "type": "str"
            }],
            "description": "Allow the uploading of certificates to AWS IAM",
            "slug": "aws-destination",
            "title": "AWS"
        },
        "label": "test546"
    }
    "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int. default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kw*)

POST /destinations

Creates a new account

Example request:

```
POST /destinations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "description": "test33",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "34324324",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "34324324",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
```

```
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "test33",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "34324324",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Parameters
- **label** – human readable account label
- **description** – some description about the account

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

class lemur.destinations.views. **DestinationsStats**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' stats endpoint

endpoint = 'destinationStats'

get ( )

mediatypes (*resource_cls*)

methods = ['GET']

Notifications

class lemur.notifications.views. **CertificateNotifications**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificate/<int:certificate_id/notifications" endpoint

endpoint = 'certificateNotifications'

get (**args*, ***kwargs*)

GET /certificates/1/notifications
The current account list for a given certificates

Example request:

```
GET /certificates/1/notifications HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
          "description": "An example",
          "options": [
              {
                  "name": "interval",
                  "required": true,
                  "value": 555,
                  "helpMessage": "Number of days to be alert before␣
↪expiration.",
                  "validation": "^\d+$",
                  "type": "int"
              },
              {
                  "available": [
                      "days",
                      "weeks",
                      "months"
                  ],
                  "name": "unit",
                  "required": true,
                  "value": "weeks",
                  "helpMessage": "Interval unit",
                  "validation": "",
                  "type": "select"
              },
              {
                  "name": "recipients",
                  "required": true,
                  "value": "kglisson@netflix.com,example@netflix.com",
                  "helpMessage": "Comma delimited list of email addresses
↪",
```

```
                    "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$",
                    "type": "str"
                }
            ],
            "label": "example",
            "pluginName": "email-notification",
            "active": true,
            "id": 2
        }
    ],
    "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.notifications.views. **Notifications**

Bases: lemur.auth.service.AuthenticatedResource

delete (*notification_id*)

endpoint = 'notification'

get (**args*, ***kwargs*)

GET /notifications/1

Get a specific account

Example request:

```
GET /notifications/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "a test",
  "options": [
      {
          "name": "interval",
          "required": true,
          "value": 5,
```

```
            "helpMessage": "Number of days to be alert before expiration.",
            "validation": "^\d+$",
            "type": "int"
        },
        {
            "available": [
                "days",
                "weeks",
                "months"
            ],
            "name": "unit",
            "required": true,
            "value": "weeks",
            "helpMessage": "Interval unit",
            "validation": "",
            "type": "select"
        },
        {
            "name": "recipients",
            "required": true,
            "value": "kglisson@netflix.com,example@netflix.com",
            "helpMessage": "Comma delimited list of email addresses",
            "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$",
            "type": "str"
        }
    ],
    "label": "test",
    "pluginName": "email-notification",
    "active": true,
    "id": 2
}
```


Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['DELETE', 'GET', 'PUT']

put (**args*, ***kwargs*)

PUT /notifications/1

Updates an account

Example request:

```
POST /notifications/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript
```

```
{
  "id": 1,
  "accountNumber": 11111111111,
  "label": "labelChanged",
  "comments": "this is a thing"
}
```

Parameters
- **accountNumber** – aws account number
- **label** – human readable account label
- **comments** – some description about the account

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

class lemur.notifications.views. **NotificationsList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'notifications' endpoint

endpoint = 'notifications'

get (**args*, ***kwargs*)

GET /notifications

The current account list

Example request:

```
GET /notifications HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
          "description": "An example",
          "options": [
              {
                  "name": "interval",
                  "required": true,
                  "value": 5,
                  "helpMessage": "Number of days to be alert before␣
↪expiration.",
                  "validation": "^\d+$",
                  "type": "int"
              },
              {
                  "available": [
                      "days",
                      "weeks",
```

```
                        "months"
                    ],
                    "name": "unit",
                    "required": true,
                    "value": "weeks",
                    "helpMessage": "Interval unit",
                    "validation": "",
                    "type": "select"
                },
                {
                    "name": "recipients",
                    "required": true,
                    "value": "kglisson@netflix.com,example@netflix.com",
                    "helpMessage": "Comma delimited list of email addresses
↪",
                    "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$",
                    "type": "str"
                }
            ],
            "label": "example",
            "pluginName": "email-notification",
            "active": true,
            "id": 2
        }
    ],
    "total": 1
}
```


Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

POST /notifications

Creates a new account

Example request:

```
POST /notifications HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "description": "a test",
  "options": [
      {
```

```
            "name": "interval",
            "required": true,
            "value": 5,
            "helpMessage": "Number of days to be alert before expiration.",
            "validation": "^\d+$",
            "type": "int"
        },
        {
            "available": [
                "days",
                "weeks",
                "months"
            ],
            "name": "unit",
            "required": true,
            "value": "weeks",
            "helpMessage": "Interval unit",
            "validation": "",
            "type": "select"
        },
        {
            "name": "recipients",
            "required": true,
            "value": "kglisson@netflix.com,example@netflix.com",
            "helpMessage": "Comma delimited list of email addresses",
            "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$",
            "type": "str"
        }
    ],
    "label": "test",
    "pluginName": "email-notification",
    "active": true,
    "id": 2
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "a test",
  "options": [
        {
            "name": "interval",
            "required": true,
            "value": 5,
            "helpMessage": "Number of days to be alert before expiration.",
            "validation": "^\d+$",
            "type": "int"
        },
        {
            "available": [
                "days",
                "weeks",
                "months"
            ],
```

```
            "name": "unit",
            "required": true,
            "value": "weeks",
            "helpMessage": "Interval unit",
            "validation": "",
            "type": "select"
        },
        {
            "name": "recipients",
            "required": true,
            "value": "kglisson@netflix.com,example@netflix.com",
            "helpMessage": "Comma delimited list of email addresses",
            "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$",
            "type": "str"
        }
    ],
    "label": "test",
    "pluginName": "email-notification",
    "active": true,
    "id": 2
}
```

Parameters
- **accountNumber** – aws account number
- **label** – human readable account label
- **comments** – some description about the account

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

Users

class lemur.users.views. **CertificateUsers**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'certificateCreator'

get (**args*, ***kwargs*)

GET /certificates/1/creator

Get a certificate's creator

Example request:

```
GET /certificates/1/creator HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
```

```
    "id": 1,
    "active": false,
    "email": "user1@example.com",
    "username": "user1",
    "profileImage": null
}
```

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.users.views. **Me**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'me'

get (**args*, ***kwargs*)

GET /auth/me
Get the currently authenticated user

Example request:

```
GET /auth/me HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "active": false,
    "email": "user1@example.com",
    "username": "user1",
    "profileImage": null
}
```

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.users.views. **RoleUsers**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'roleUsers'

get (**args*, ***kwargs*)

GET /roles/1/users

Get all users associated with a role

Example request:

```
GET /roles/1/users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 2,
        "active": True,
        "email": "user2@example.com",
        "username": "user2",
        "profileImage": null
      },
      {
        "id": 1,
        "active": False,
        "email": "user1@example.com",
        "username": "user1",
        "profileImage": null
      }
    ]
  "total": 2
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.users.views. **Users**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'user'

get (**args*, ***kwargs*)

GET /users/1

Get a specific user

Example request:

```
GET /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "active": false,
    "email": "user1@example.com",
    "username": "user1",
    "profileImage": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET', 'PUT']

put (**args*, ***kw*)

PUT /users/1

Update a user

Example request:

```
PUT /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "username": "user1",
   "email": "user1@example.com",
   "active": false,
   "roles": [
       {'id': 1} - or - {'name': 'myRole'}
   ]
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "id": 1,
   "username": "user1",
   "email": "user1@example.com",
   "active": false,
```

```
    "profileImage": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

class lemur.users.views. **UsersList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'users' endpoint

endpoint = 'users'

get (**args*, ***kwargs*)

GET /users

The current user list

Example request:

```
GET /users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "items": [
        {
            "id": 2,
            "active": True,
            "email": "user2@example.com",
            "username": "user2",
            "profileImage": null
        },
        {
            "id": 1,
            "active": False,
            "email": "user1@example.com",
            "username": "user1",
            "profileImage": null
        }
    ]
    "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kw*)

POST /users

Creates a new user

Example request:

```
POST /users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "username": "user3",
   "email": "user3@example.com",
   "active": true,
   "roles": [
      {'id': 1} - or - {'name': 'myRole'}
   ]
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 3,
    "active": True,
    "email": "user3@example.com,
    "username": "user3",
    "profileImage": null
}
```

Parameters
- **username** – username for new user
- **email** – email address for new user
- **password** – password for new user
- **active** – boolean, if the user is currently active
- **roles** – list, roles that the user should be apart of

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

Roles

class lemur.roles.views. **AuthorityRolesList**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'authorityRoles'

get (**args*, ***kwargs*)

GET /authorities/1/roles
List of roles for a given authority

Example request:

```
GET /authorities/1/roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "role1",
        "description": "this is role1"
      },
      {
        "id": 2,
        "name": "role2",
        "description": "this is role2"
      }
    ]
  "total": 2
}
```

Query Parameters
- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.roles.views. **RoleViewCredentials**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'roleCredentials''

get (*role_id*)

GET /roles/1/credentials

View a roles credentials

Example request:

```
GET /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "username: "ausername",
    "password": "apassword"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.roles.views. **Roles**

Bases: lemur.auth.service.AuthenticatedResource

delete (**args*, ***kw*)

DELETE /roles/1

Delete a role

Example request:

```
DELETE /roles/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "message": "ok"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

endpoint = 'role'

get (**args*, ***kwargs*)

GET /roles/1

Get a particular role

Example request:

```
GET /roles/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "name": "role1",
    "description": "this is role1"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['DELETE', 'GET', 'PUT']

put (**args*, ***kwargs*)

PUT /roles/1

Update a role

Example request:

```
PUT /roles/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "name": "role1",
   "description": "This is a new description"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "id": 1,
   "name": "role1",
   "description": "this is a new description"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.roles.views. **RolesList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'roles'

get (**args*, ***kwargs*)

GET /roles

The current role list

Example request:

```
GET /roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "role1",
        "description": "this is role1"
      },
      {
        "id": 2,
        "name": "role2",
        "description": "this is role2"
      }
    ]
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on

- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kw*)

POST /roles

Creates a new role

Example request:

```
POST /roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "name": "role3",
   "description": "this is role3",
   "username": null,
   "password": null,
   "users": [
      {'id': 1}
   ]
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 3,
    "description": "this is role3",
    "name": "role3"
}
```

Parameters

- **name** – name for new role
- **description** – description for new role
- **password** – password for new role
- **username** – username for new role
- **users** – list, of users to associate with role

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.roles.views. **UserRolesList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'userRoles'

get (**args*, ***kwargs*)

GET /users/1/roles

List of roles for a given user

Example request:

```
GET /users/1/roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "role1",
        "description": "this is role1"
      },
      {
        "id": 2,
        "name": "role2",
        "description": "this is role2"
      }
    ]
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

Certificates

class lemur.certificates.views. **CertificateExport**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'exportCertificate'

mediatypes (*resource_cls*)

methods = ['POST']

post (**args*, ***kwargs*)

POST /certificates/1/export
Export a certificate

Example request:

```
PUT /certificates/1/export HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "export": {
      "plugin": {
          "pluginOptions": [{
              "available": ["Java Key Store (JKS)"],
              "required": true,
              "type": "select",
              "name": "type",
              "helpMessage": "Choose the format you wish to export",
              "value": "Java Key Store (JKS)"
          }, {
              "required": false,
              "type": "str",
              "name": "passphrase",
              "validation": "^(?=.*[A-Za-z])(?=.*\d)(?=.*[$@$!%*#?&])[A-
↪Za-z\d$@$!%*#?&]{8,}$",
              "helpMessage": "If no passphrase is given one will be␣
↪generated for you, we highly recommend this. Minimum length is 8."
          }, {
              "required": false,
              "type": "str",
              "name": "alias",
              "helpMessage": "Enter the alias you wish to use for the␣
↪keystore."
          }],
          "version": "unknown",
          "description": "Attempts to generate a JKS keystore or␣
↪truststore",
          "title": "Java",
          "author": "Kevin Glisson",
          "type": "export",
          "slug": "java-export"
      }
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "data": "base64encodedstring",
  "passphrase": "UAWOHW#&@_%!tnwmxh832025",
  "extension": "jks"
}
```

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.certificates.views. **CertificatePrivateKey**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'privateKeyCertificates'

get (*certificate_id*)

GET /certificates/1/key
Retrieves the private key for a given certificate

Example request:

```
GET /certificates/1/key HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "key": "-----BEGIN ...",
}
```

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.certificates.views. **Certificates**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'certificate'

get (**args*, ***kwargs*)

GET /certificates/1

One certificate

Example request:

```
GET /certificates/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "status": null,
  "cn": "*.test.example.net",
  "chain": "",
  "authority": {
      "active": true,
      "owner": "secure@example.com",
      "id": 1,
      "description": "verisign test authority",
      "name": "verisign"
  },
  "owner": "joe@example.com",
  "serial": "823110587320259241427891793688893091 56",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
  "description": null,
  "deleted": null,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "replaces": [],
  "replaced": [],
  "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-20180112
↪",
  "roles": [{
      "id": 464,
```

```
      "description": "This is a google group based role created by Lemur",
      "name": "joe@example.com"
  }],
  "san": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'PUT']

put (**args*, ***kwargs*)

PUT /certificates/1

Update a certificate

Example request:

```
PUT /certificates/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "owner": "jimbob@example.com",
   "active": false
   "notifications": [],
   "destinations": [],
   "replacements": []
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "status": null,
  "cn": "*.test.example.net",
  "chain": "",
  "authority": {
      "active": true,
      "owner": "secure@example.com",
      "id": 1,
      "description": "verisign test authority",
      "name": "verisign"
  },
  "owner": "joe@example.com",
  "serial": "823110587320259241427891793688893091 56",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
```

```
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
  "description": null,
  "deleted": null,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "replaces": [],
  "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-20180112
↪",
  "roles": [{
      "id": 464,
      "description": "This is a google group based role created by Lemur",
      "name": "joe@example.com"
  }],
  "san": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.certificates.views. **CertificatesList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' endpoint

endpoint = 'certificates'

get (**args*, ***kwargs*)

GET /certificates

The current list of certificates

Example request:

```
GET /certificates HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "status": null,
      "cn": "*.test.example.net",
      "chain": "",
      "authority": {
          "active": true,
          "owner": "secure@example.com",
          "id": 1,
          "description": "verisign test authority",
          "name": "verisign"
      },
      "owner": "joe@example.com",
      "serial": "823110587320259241427891793688889309156",
      "id": 2288,
      "issuer": "SymantecCorporation",
      "notBefore": "2016-06-03T00:00:00+00:00",
      "notAfter": "2018-01-12T23:59:59+00:00",
      "destinations": [],
      "bits": 2048,
      "body": "-----BEGIN CERTIFICATE-----...",
      "description": null,
      "deleted": null,
      "notifications": [{
          "id": 1
      }]
      "signingAlgorithm": "sha256",
      "user": {
          "username": "jane",
          "active": true,
          "email": "jane@example.com",
          "id": 2
      },
      "active": true,
      "domains": [{
          "sensitive": false,
          "id": 1090,
          "name": "*.test.example.net"
      }],
      "replaces": [],
      "replaced": [],
      "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
      "roles": [{
          "id": 464,
          "description": "This is a google group based role created by␣
↪Lemur",
          "name": "joe@example.com"
      }],
      "san": null
  }],
  "total": 1
}
```

Query Parameters
- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int. default is 1
- **filter** – key value pair format is k;v
- **count** – count number. default is 10

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

POST /certificates

Creates a new certificate

Example request:

```
POST /certificates HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
    "owner": "secure@example.net",
    "commonName": "test.example.net",
    "country": "US",
    "extensions": {
      "subAltNames": {
        "names": [
          {
            "nameType": "DNSName",
            "value": "*.test.example.net"
          },
          {
            "nameType": "DNSName",
            "value": "www.test.example.net"
          }
        ]
      }
    },
    "replacements": [{
      "id": 1
    },
    "notify": true,
    "validityEnd": "2026-01-01T08:00:00.000Z",
    "authority": {
      "name": "verisign"
    },
    "organization": "Netflix, Inc.",
    "location": "Los Gatos",
    "state": "California",
    "validityStart": "2016-11-11T04:19:48.000Z",
    "organizationalUnit": "Operations"
```

```
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "status": null,
  "cn": "*.test.example.net",
  "chain": "",
  "authority": {
      "active": true,
      "owner": "secure@example.com",
      "id": 1,
      "description": "verisign test authority",
      "name": "verisign"
  },
  "owner": "joe@example.com",
  "serial": "823110587320259241427891793688893091 56",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
  "description": null,
  "deleted": null,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "replaces": [{
      "id": 1
  }],
  "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-20180112
↪",
  "roles": [{
      "id": 464,
      "description": "This is a google group based role created by Lemur",
      "name": "joe@example.com"
  }],
  "san": null
}
```

Parameters
- **extensions** – extensions to be used in the certificate
- **description** – description for new certificate
- **owner** – owner email
- **validityStart** – when the certificate should start being valid
- **validityEnd** – when the certificate should expire
- **authority** – authority that should issue the certificate
- **country** – country for the CSR
- **state** – state for the CSR
- **location** – location for the CSR
- **organization** – organization for CSR
- **commonName** – certificate common name

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.certificates.views. **CertificatesReplacementsList**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'replacements'

get (**args*, ***kwargs*)

GET /certificates/1/replacements

One certificate

Example request:

```
GET /certificates/1/replacements HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "status": null,
      "cn": "*.test.example.net",
      "chain": "",
      "authority": {
          "active": true,
          "owner": "secure@example.com",
          "id": 1,
          "description": "verisign test authority",
          "name": "verisign"
      },
      "owner": "joe@example.com",
      "serial": "823110587320259241427891793688893091 56",
      "id": 2288,
      "issuer": "SymantecCorporation",
      "notBefore": "2016-06-03T00:00:00+00:00",
      "notAfter": "2018-01-12T23:59:59+00:00",
```

```
      "destinations": [],
      "bits": 2048,
      "body": "-----BEGIN CERTIFICATE-----...",
      "description": null,
      "deleted": null,
      "notifications": [{
          "id": 1
      }]
      "signingAlgorithm": "sha256",
      "user": {
          "username": "jane",
          "active": true,
          "email": "jane@example.com",
          "id": 2
      },
      "active": true,
      "domains": [{
          "sensitive": false,
          "id": 1090,
          "name": "*.test.example.net"
      }],
      "replaces": [],
      "replaced": [],
      "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
      "roles": [{
          "id": 464,
          "description": "This is a google group based role created by␣
↪Lemur",
          "name": "joe@example.com"
      }],
      "san": null
  }],
  "total": 1
}
```


Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.certificates.views. **CertificatesStats**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' stats endpoint

endpoint = 'certificateStats'

get ( )

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.certificates.views. **CertificatesUpload**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' upload endpoint

endpoint = 'certificateUpload'

mediatypes (*resource_cls*)

methods = ['POST']

post (**args*, ***kwargs*)

POST /certificates/upload

Upload a certificate

Example request:

```
POST /certificates/upload HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "owner": "joe@example.com",
   "publicCert": "-----BEGIN CERTIFICATE-----...",
   "intermediateCert": "-----BEGIN CERTIFICATE-----...",
   "privateKey": "-----BEGIN RSA PRIVATE KEY-----..."
   "destinations": [],
   "notifications": [],
   "replacements": [],
   "name": "cert1"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "status": null,
  "cn": "*.test.example.net",
  "chain": "",
  "authority": {
      "active": true,
      "owner": "secure@example.com",
      "id": 1,
      "description": "verisign test authority",
      "name": "verisign"
  },
  "owner": "joe@example.com",
  "serial": "823110587320259241427891793688893091 56",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
  "description": null,
  "deleted": null,
  "notifications": [{
      "id": 1
```

```
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "replaces": [],
  "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-20180112
↪",
  "roles": [{
      "id": 464,
      "description": "This is a google group based role created by Lemur",
      "name": "joe@example.com"
  }],
  "san": null
}
```

Parameters

- **owner** – owner email for certificate
- **publicCert** – valid PEM public key for certificate

:arg intermediateCert valid PEM intermediate key for certificate :arg privateKey: valid PEM private key for certificate :arg destinations: list of aws destinations to upload the certificate to :reqheader Authorization: OAuth token to authenticate :statuscode 403: unauthenticated :statuscode 200: no error

class lemur.certificates.views. **NotificationCertificatesList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' endpoint

endpoint = 'notificationCertificates'

get (**args*, ***kwargs*)

GET /notifications/1/certificates

The current list of certificates for a given notification

Example request:

```
GET /notifications/1/certificates HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
```

```
  "items": [{
      "status": null,
      "cn": "*.test.example.net",
      "chain": "",
      "authority": {
          "active": true,
          "owner": "secure@example.com",
          "id": 1,
          "description": "verisign test authority",
          "name": "verisign"
      },
      "owner": "joe@example.com",
      "serial": "823110587320259241427891793688893091 56",
      "id": 2288,
      "issuer": "SymantecCorporation",
      "notBefore": "2016-06-03T00:00:00+00:00",
      "notAfter": "2018-01-12T23:59:59+00:00",
      "destinations": [],
      "bits": 2048,
      "body": "-----BEGIN CERTIFICATE-----...",
      "description": null,
      "deleted": null,
      "notifications": [{
          "id": 1
      }]
      "signingAlgorithm": "sha256",
      "user": {
          "username": "jane",
          "active": true,
          "email": "jane@example.com",
          "id": 2
      },
      "active": true,
      "domains": [{
          "sensitive": false,
          "id": 1090,
          "name": "*.test.example.net"
      }],
      "replaces": [],
      "replaced": [],
      "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
      "roles": [{
          "id": 464,
          "description": "This is a google group based role created by␣
↪Lemur",
          "name": "joe@example.com"
      }],
      "san": null
  }],
  "total": 1
}
```


Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v

- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

Authorities

class lemur.authorities.views. **Authorities**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'authority'

get (**args*, ***kwargs*)

GET /authorities/1

One authority

Example request:

```
GET /authorities/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "roles": [{
      "id": 123,
      "name": "secure@example.com"
  }, {
      "id": 564,
      "name": "TestAuthority_admin"
  }, {
      "id": 565,
      "name": "TestAuthority_operator"
  }],
  "active": true,
  "owner": "secure@example.com",
  "id": 43,
  "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
}
```

Parameters

- **description** – a sensible description about what the CA with be used for
- **owner** – the team or person who 'owns' this authority
- **active** – set whether this authoritity is currently in use

Request Headers

- Authorization – OAuth token to authenticate
- Authorization – OAuth token to authenticate

Status Codes

- 403 Forbidden – unauthenticated
- 200 OK – no error
- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'PUT']

put (**args*, ***kwargs*)

PUT /authorities/1

Update an authority

Example request:

```
PUT /authorities/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "name": "TestAuthority5",
  "roles": [{
      "id": 566,
      "name": "TestAuthority5_admin"
  }, {
      "id": 567,
      "name": "TestAuthority5_operator"
  }, {
      "id": 123,
      "name": "secure@example.com"
  }],
  "active": true,
  "authorityCertificate": {
      "body": "-----BEGIN CERTIFICATE-----",
      "status": null,
      "cn": "AcommonName",
      "description": "This is the ROOT certificate for the TestAuthority5␣
↪certificate authority.",
      "chain": "",
      "notBefore": "2016-06-03T00:00:51+00:00",
      "notAfter": "2036-06-03T23:59:51+00:00",
      "owner": "secure@example.com",
      "user": {
          "username": "joe@example.com",
          "active": true,
          "email": "joe@example.com",
          "id": 3
      },
      "active": true,
      "bits": 2048,
      "id": 2280,
      "name": "TestAuthority5"
  },
  "owner": "secure@example.com",
```

```
  "id": 44,
  "description": "This is the ROOT certificate for the TestAuthority5␣
↪certificate authority."
 }
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "name": "TestAuthority",
  "roles": [{
      "id": 123,
      "name": "secure@example.com"
  }, {
      "id": 564,
      "name": "TestAuthority_admin"
  }, {
      "id": 565,
      "name": "TestAuthority_operator"
  }],
  "options": null,
  "active": true,
  "authorityCertificate": {
      "body": "-----BEGIN CERTIFICATE-----IyMzU5MTVaMHk...",
      "status": true,
      "cn": "AcommonName",
      "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority.",
      "chain": "",
      "notBefore": "2016-06-02T00:00:15+00:00",
      "notAfter": "2023-06-02T23:59:15+00:00",
      "owner": "secure@example.com",
      "user": {
          "username": "joe@example.com",
          "active": true,
          "email": "joe@example.com",
          "id": 3
      },
      "active": true,
      "bits": 2048,
      "id": 2235,
      "name": "TestAuthority"
  },
  "owner": "secure@example.com",
  "id": 43,
  "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.authorities.views. **AuthoritiesList**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'authorities' endpoint

endpoint = 'authorities'

get (**args*, ***kwargs*)

GET /authorities
The current list of authorities

Example request:

```
GET /authorities HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "name": "TestAuthority",
      "roles": [{
          "id": 123,
          "name": "secure@example.com"
      }, {
          "id": 564,
          "name": "TestAuthority_admin"
      }, {
          "id": 565,
          "name": "TestAuthority_operator"
      }],
      "options": null,
      "active": true,
      "authorityCertificate": {
          "body": "-----BEGIN CERTIFICATE-----IyMzU5MTVaMHk...",
          "status": true,
          "cn": "AcommonName",
          "description": "This is the ROOT certificate for the␣
↪TestAuthority certificate authority.",
          "chain": "",
          "notBefore": "2016-06-02T00:00:15+00:00",
          "notAfter": "2023-06-02T23:59:15+00:00",
          "owner": "secure@example.com",
          "user": {
              "username": "joe@example.com",
              "active": true,
              "email": "joe@example.com",
              "id": 3
          },
          "active": true,
          "bits": 2048,
          "id": 2235,
          "name": "TestAuthority"
```

```
      },
      "owner": "secure@example.com",
      "id": 43,
      "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
  }
  "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair. format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

Note this will only show certificates that the current user is authorized to use

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

POST /authorities

Create an authority

Example request:

```
POST /authorities HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "country": "US",
   "state": "California",
   "location": "Los Gatos",
   "organization": "Netflix",
   "organizationalUnit": "Operations",
   "type": "root",
   "signingAlgorithm": "sha256WithRSA",
   "sensitivity": "medium",
   "keyType": "RSA2048",
   "plugin": {
       "slug": "cloudca-issuer",
   },
   "name": "TimeTestAuthority5",
   "owner": "secure@example.com",
   "description": "test",
   "commonName": "AcommonName",
   "validityYears": "20",
   "extensions": {
       "subAltNames": {
           "names": []
```

```
        },
        "custom": []
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "name": "TestAuthority",
  "roles": [{
      "id": 123,
      "name": "secure@example.com"
  }, {
      "id": 564,
      "name": "TestAuthority_admin"
  }, {
      "id": 565,
      "name": "TestAuthority_operator"
  }],
  "options": null,
  "active": true,
  "authorityCertificate": {
      "body": "-----BEGIN CERTIFICATE-----IyMzU5MTVaMHk...",
      "status": true,
      "cn": "AcommonName",
      "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority.",
      "chain": "",
      "notBefore": "2016-06-02T00:00:15+00:00",
      "notAfter": "2023-06-02T23:59:15+00:00",
      "owner": "secure@example.com",
      "user": {
          "username": "joe@example.com",
          "active": true,
          "email": "joe@example.com",
          "id": 3
      },
      "active": true,
      "bits": 2048,
      "id": 2235,
      "name": "TestAuthority"
  },
  "owner": "secure@example.com",
  "id": 43,
  "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
}
```

Parameters

- **name** – authority's name
- **description** – a sensible description about what the CA with be used for
- **owner** – the team or person who 'owns' this authority
- **validityStart** – when this authority should start issuing certificates
- **validityEnd** – when this authority should stop issuing certificates

- **validityYears** – starting from *now* how many years into the future the authority should be valid
- **extensions** – certificate extensions
- **plugin** – name of the plugin to create the authority
- **type** – the type of authority (root/subca)
- **parent** – the parent authority if this is to be a subca
- **signingAlgorithm** – algorithm used to sign the authority
- **keyType** – key type
- **sensitivity** – the sensitivity of the root key, for CloudCA this determines if the root keys are stored

in an HSM :arg keyName: name of the key to store in the HSM (CloudCA) :arg serialNumber: serial number of the authority :arg firstSerial: specifies the starting serial number for certificates issued off of this authority :reqheader Authorization: OAuth token to authenticate :statuscode 403: unauthenticated :statuscode 200: no error

class lemur.authorities.views. **AuthorityVisualizations**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'authority_visualizations'

get (*authority_id*)
{"name": "flare", "children": [

{ "name": "analytics", "children": [

{ "name": "cluster", "children": [

{"name": "AgglomerativeCluster", "size": 3938}, {"name": "CommunityStructure", "size": 3812}, {"name": "HierarchicalCluster", "size": 6714}, {"name": "MergeEdge", "size": 743}

]

}

}

]}

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.authorities.views. **CertificateAuthority**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'certificateAuthority'

get (**args*, ***kwargs*)

GET /certificates/1/authority
One authority for given certificate

Example request:

```
GET /certificates/1/authority HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "name": "TestAuthority",
  "roles": [{
      "id": 123,
      "name": "secure@example.com"
  }, {
      "id": 564,
      "name": "TestAuthority_admin"
  }, {
      "id": 565,
      "name": "TestAuthority_operator"
  }],
  "options": null,
  "active": true,
  "authorityCertificate": {
      "body": "-----BEGIN CERTIFICATE-----IyMzU5MTVaMHk...",
      "status": true,
      "cn": "AcommonName",
      "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority.",
      "chain": "",
      "notBefore": "2016-06-02T00:00:15+00:00",
      "notAfter": "2023-06-02T23:59:15+00:00",
      "owner": "secure@example.com",
      "user": {
          "username": "joe@example.com",
          "active": true,
          "email": "joe@example.com",
          "id": 3
      },
      "active": true,
      "bits": 2048,
      "id": 2235,
      "name": "TestAuthority"
  },
  "owner": "secure@example.com",
  "id": 43,
  "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

Domains

class lemur.domains.views. **CertificateDomains**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'domains' endpoint

endpoint = 'certificateDomains'

get (**args*, ***kwargs*)

GET /certificates/1/domains
The current domain list

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "www.example.com",
        "sensitive": false
      },
      {
        "id": 2,
        "name": "www.example2.com",
        "sensitive": false
      }
    ]
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.domains.views. **Domains**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'domain'

get (**args*, ***kwargs*)

GET /domains/1
Fetch one domain

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "name": "www.example.com",
    "sensitive": false
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'PUT']

put (**args*, ***kwargs*)

GET /domains/1
update one domain

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
```

```
    "name": "www.example.com",
    "sensitive": false
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "name": "www.example.com",
    "sensitive": false
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.domains.views. **DomainsList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'domains' endpoint

endpoint = 'domains'

get (**args*, ***kwargs*)

GET /domains

The current domain list

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "www.example.com",
        "sensitive": false
      },
      {
        "id": 2,
```

```
        "name": "www.example2.com",
        "sensitive": false
      }
    ]
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number. default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

POST /domains

The current domain list

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "name": "www.example.com",
  "sensitive": false
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "id": 1,
  "name": "www.example.com",
  "sensitive": false
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

Internals

lemur Package

lemur Package

constants Module

database Module

lemur.database. **add** (*model*)
Helper to add a *model* to the current session.

Parameters **model** –

Returns

lemur.database. **clone** (*model*)
Clones the given model and removes it's primary key :param model: :return:

lemur.database. **commit** ( )
Helper to commit the current session.

lemur.database. **create** (*model*)
Helper that attempts to create a new instance of an object.

Parameters **model** –

Returns

raise IntegrityError

lemur.database. **create_query** (*model*, *kwargs*)
Returns a SQLAlchemy query object for specified *model*. Model filtered by the kwargs passed.

Parameters

- **model** –
- **kwargs** –

Returns

lemur.database. **delete** (*model*)
Helper that attempts to delete a model.

Parameters **model** –

lemur.database. **filter** (*query*, *model*, *terms*)
Helper that searched for 'like' strings in column values.

Parameters

- **query** –
- **model** –
- **terms** –

Returns

lemur.database. **filter_none** (*kwargs*)
Remove all *None* values froma given dict. SQLAlchemy does not like to have values that are None passed to it.

Parameters **kwargs** – Dict to filter

Returns Dict without any 'None' values

lemur.database. **find_all** (*query*, *model*, *kwargs*)
Returns a query object that ensures that all kwargs are present.

Parameters

- **query** –
- **model** –
- **kwargs** –

Returns

lemur.database. **find_any** (*query*, *model*, *kwargs*)
Returns a query object that allows any kwarg to be present.

Parameters

- **query** –
- **model** –
- **kwargs** –

Returns

lemur.database. **get** (*model*, *value*, *field='id'*)
Returns one object filtered by the field and value.

Parameters

- **model** –
- **value** –
- **field** –

Returns

lemur.database. **get_all** (*model*, *value*, *field='id'*)
Returns query object with the fields and value filtered.

Parameters

- **model** –
- **value** –
- **field** –

Returns

lemur.database. **paginate** (*query*, *page*, *count*)

Returns the items given the count and page specified

Parameters

- **query** –
- **page** –
- **count** –

lemur.database. **session_query** (*model*)

Returns a SQLAlchemy query object for the specified *model*.

If *model* has a `query` attribute already, that object will be returned. Otherwise a query will be created and returned based on *session*.

Parameters **model** – sqlalchemy model

Returns query object for model

lemur.database. **sort** (*query*, *model*, *field*, *direction*)

Returns objects of the specified *model* in the field and direction given

Parameters

- **query** –
- **model** –
- **field** –
- **direction** –

lemur.database. **sort_and_page** (*query*, *model*, *args*)

Helper that allows us to combine sorting and paging

Parameters

- **query** –
- **model** –
- **args** –

Returns

lemur.database. **update** (*model*)

Helper that attempts to update a model.

Parameters **model** –

Returns

lemur.database. **update_list** (*model*, *model_attr*, *item_model*, *items*)

Helper that correctly updates a models items depending on what has changed

Parameters

- **model_attr** –
- **item_model** –
- **items** –
- **model** –

Returns

decorators Module

lemur.decorators. **crossdomain** (*origin=None*, *methods=None*, *headers=None*, *max_age=21600*, *attach_to_all=True*, *automatic_options=True*)

exceptions Module

exception lemur.exceptions. **AttrNotFound** (*field*)
Bases: lemur.exceptions.LemurException

exception lemur.exceptions. **DuplicateError** (*key*)
Bases: lemur.exceptions.LemurException

exception lemur.exceptions. **InvalidConfiguration**
Bases: exceptions.Exception

exception lemur.exceptions. **InvalidListener** (**args*, ***kwargs*)
Bases: lemur.exceptions.LemurException

exception lemur.exceptions. **LemurException** (**args*, ***kwargs*)
Bases: exceptions.Exception

extensions Module

factory Module

lemur.factory. **configure_app** (*app*, *config=None*)
Different ways of configuration

Parameters

- **app** –
- **config** –

Returns

lemur.factory. **configure_blueprints** (*app*, *blueprints*)
We prefix our APIs with their given version so that we can support multiple concurrent API versions.

Parameters

- **app** –
- **blueprints** –

lemur.factory. **configure_extensions** (*app*)
Attaches and configures any needed flask extensions to our app.

Parameters **app** –

lemur.factory. **configure_logging** (*app*)
Sets up application wide logging.

Parameters **app** –

lemur.factory. **create_app** (*app_name=None*, *blueprints=None*, *config=None*)
Lemur application factory

Parameters

- **config** –
- **app_name** –
- **blueprints** –

Returns

lemur.factory. **from_file** (*file_path*, *silent=False*)
Updates the values in the config from a Python file. This function behaves as if the file was imported as module with the

Parameters

- **file_path** –
- **silent** –

lemur.factory. **install_plugins** (*app*)
Installs new issuers that are not currently bundled with Lemur.

Parameters **app** –

Returns

manage Module

models Module

Subpackages

auth Package

permissions Module

lemur.auth.permissions. **AuthorityCreator**
alias of authority

lemur.auth.permissions. **AuthorityOwner**
alias of authority

class lemur.auth.permissions. **AuthorityPermission** (*authority_id*, *roles*)
Bases: flask_principal.Permission

lemur.auth.permissions. **CertificateOwner**
alias of certificate

class lemur.auth.permissions. **CertificatePermission** (*owner*, *roles*)
Bases: flask_principal.Permission

lemur.auth.permissions. **RoleMember**
alias of role

class lemur.auth.permissions. **RoleMemberPermission** (*role_id*)
Bases: flask_principal.Permission

class lemur.auth.permissions. **SensitiveDomainPermission**
Bases: flask_principal.Permission

service Module

class lemur.auth.service. **AuthenticatedResource**
Bases: flask_restful.Resource

Inherited by all resources that need to be protected by authentication.

method_decorators = [<function login_required>]

lemur.auth.service. **create_token** (*user*)
Create a valid JWT for a given user, this token is then used to authenticate sessions until the token expires.

Parameters **user** –

Returns

lemur.auth.service. **fetch_token_header** (*token*)
Fetch the header out of the JWT token.

Parameters **token** –

Returns

raise jwt.DecodeError

lemur.auth.service. **get_rsa_public_key** (*n*, *e*)
Retrieve an RSA public key based on a module and exponent as provided by the JWKS format.

Parameters

- **n** –
- **e** –

Returns a RSA Public Key in PEM format

lemur.auth.service. **login_required** (*f*)
Validates the JWT and ensures that is has not expired and the user is still active.

Parameters **f** –

Returns

lemur.auth.service. **on_identity_loaded** (*sender*, *identity*)
Sets the identity of a given option, assigns additional permissions based on the role that the user is a part of.

Parameters

- **sender** –
- **identity** –

views Module

class lemur.auth.views. **Google**
Bases: flask_restful.Resource

endpoint = 'google'

mediatypes (*resource_cls*)

methods = ['POST']

post ( )

class lemur.auth.views. **Login**
Bases: flask_restful.Resource

Provides an endpoint for Lemur's basic authentication. It takes a username and password combination and returns a JWT token.

This token token is required for each API request and must be provided in the Authorization Header for the request.

```
Authorization:Bearer <token>
```

Tokens have a set expiration date. You can inspect the token expiration by base64 decoding the token and inspecting it's contents.

Note: It is recommended that the token expiration is fairly short lived (hours not days). This will largely depend on your uses cases but. It is important to not that there is currently no build in method to revoke a users token and force re-authentication.

---

endpoint = 'login'

mediatypes (*resource_cls*)

methods = ['POST']

post ( )

POST /auth/login
Login with username:password

Example request:

```
POST /auth/login HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "username": "test",
  "password": "test"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "token": "12343243243"
}
```

Parameters

- **username** – username
- **password** – password

Status Codes

- 401 Unauthorized – invalid credentials
- 200 OK – no error

class lemur.auth.views. **OAuth2**
Bases: flask_restful.Resource

endpoint = 'oauth2'

mediatypes (*resource_cls*)

methods = ['POST']

post ( )

class lemur.auth.views. **Ping**
Bases: flask_restful.Resource

This class serves as an example of how one might implement an SSO provider for use with Lemur. In this example we use an OpenIDConnect authentication flow, that is essentially OAuth2 underneath. If you have an OAuth2 provider you want to use Lemur there would be two steps:

1. Define your own class that inherits from `flask.ext.restful.Resource` and create the HTTP methods the provider uses for it's callbacks.
2. Add or change the Lemur AngularJS Configuration to point to your new provider

endpoint = 'ping'

mediatypes (*resource_cls*)

methods = ['POST']

post ( )

class lemur.auth.views. **Providers**
Bases: flask_restful.Resource

endpoint = 'providers'

get ( )

mediatypes (*resource_cls*)

methods = ['GET']

authorities Package

models Module

class lemur.authorities.models. **Authority** (***kwargs*)
Bases: flask_sqlalchemy.Model

active

authority_certificate

body

certificates

chain

date_created

description

id

name

options

owner

plugin

plugin_name

roles

user_id

service Module

lemur.authorities.service. **create** (***kwargs*)
Creates a new authority.

lemur.authorities.service. **create_authority_roles** (*roles*, *owner*, *plugin_title*, *creator*)
Creates all of the necessary authority roles. :param creator: :param roles: :return:

lemur.authorities.service. **get** (*authority_id*)
Retrieves an authority given it's ID

Parameters **authority_id** –

Returns

lemur.authorities.service. **get_all** ( )
Get all authorities that are currently in Lemur.

:rtype : List :return:

lemur.authorities.service. **get_authority_role** (*ca_name*, *creator=None*)
Attempts to get the authority role for a given ca uses current_user as a basis for accomplishing that.

Parameters **ca_name** –

lemur.authorities.service. **get_by_name** (*authority_name*)
Retrieves an authority given it's name.

Parameters **authority_name** –

Returns

lemur.authorities.service. **mint** (***kwargs*)
Creates the authority based on the plugin provided.

lemur.authorities.service. **render** (*args*)
Helper that helps us render the REST Api responses. :param args: :return:

lemur.authorities.service. **update** (*authority_id*, *description=None*, *owner=None*, *active=None*, *roles=None*)
Update an authority with new values.

Parameters

- **authority_id** –
- **roles** – roles that are allowed to use this authority

Returns

views Module

class lemur.authorities.views. **Authorities**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'authority'

get (**args*, ***kwargs*)

GET /authorities/1

One authority

Example request:

```
GET /authorities/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "roles": [{
      "id": 123,
      "name": "secure@example.com"
  }, {
      "id": 564,
      "name": "TestAuthority_admin"
  }, {
      "id": 565,
      "name": "TestAuthority_operator"
  }],
  "active": true,
  "owner": "secure@example.com",
  "id": 43,
  "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
}
```

Parameters

- **description** – a sensible description about what the CA with be used for
- **owner** – the team or person who 'owns' this authority
- **active** – set whether this authoritity is currently in use

Request Headers

- Authorization – OAuth token to authenticate

- Authorization – OAuth token to authenticate

Status Codes

- 403 Forbidden – unauthenticated
- 200 OK – no error
- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'PUT']

put (**args*, ***kwargs*)

PUT /authorities/1

Update an authority

Example request:

```
PUT /authorities/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "name": "TestAuthority5",
  "roles": [{
      "id": 566,
      "name": "TestAuthority5_admin"
  }, {
      "id": 567,
      "name": "TestAuthority5_operator"
  }, {
      "id": 123,
      "name": "secure@example.com"
  }],
  "active": true,
  "authorityCertificate": {
      "body": "-----BEGIN CERTIFICATE-----",
      "status": null,
      "cn": "AcommonName",
      "description": "This is the ROOT certificate for the␣
↪TestAuthority5 certificate authority.",
      "chain": "",
      "notBefore": "2016-06-03T00:00:51+00:00",
      "notAfter": "2036-06-03T23:59:51+00:00",
      "owner": "secure@example.com",
      "user": {
          "username": "joe@example.com",
          "active": true,
          "email": "joe@example.com",
          "id": 3
      },
      "active": true,
      "bits": 2048,
      "id": 2280,
      "name": "TestAuthority5"
  },
```

```
  "owner": "secure@example.com",
  "id": 44,
  "description": "This is the ROOT certificate for the TestAuthority5␣
↪certificate authority."
 }
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "name": "TestAuthority",
  "roles": [{
      "id": 123,
      "name": "secure@example.com"
  }, {
      "id": 564,
      "name": "TestAuthority_admin"
  }, {
      "id": 565,
      "name": "TestAuthority_operator"
  }],
  "options": null,
  "active": true,
  "authorityCertificate": {
      "body": "-----BEGIN CERTIFICATE-----IyMzU5MTVaMHk...",
      "status": true,
      "cn": "AcommonName",
      "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority.",
      "chain": "",
      "notBefore": "2016-06-02T00:00:15+00:00",
      "notAfter": "2023-06-02T23:59:15+00:00",
      "owner": "secure@example.com",
      "user": {
          "username": "joe@example.com",
          "active": true,
          "email": "joe@example.com",
          "id": 3
      },
      "active": true,
      "bits": 2048,
      "id": 2235,
      "name": "TestAuthority"
  },
  "owner": "secure@example.com",
  "id": 43,
  "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.authorities.views. **AuthoritiesList**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'authorities' endpoint

endpoint = 'authorities'

get (**args*, ***kwargs*)

GET /authorities
The current list of authorities

Example request:

```
GET /authorities HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "name": "TestAuthority",
      "roles": [{
          "id": 123,
          "name": "secure@example.com"
      }, {
          "id": 564,
          "name": "TestAuthority_admin"
      }, {
          "id": 565,
          "name": "TestAuthority_operator"
      }],
      "options": null,
      "active": true,
      "authorityCertificate": {
          "body": "-----BEGIN CERTIFICATE-----IyMzU5MTVaMHk...",
          "status": true,
          "cn": "AcommonName",
          "description": "This is the ROOT certificate for the␣
↪TestAuthority certificate authority.",
          "chain": "",
          "notBefore": "2016-06-02T00:00:15+00:00",
          "notAfter": "2023-06-02T23:59:15+00:00",
          "owner": "secure@example.com",
          "user": {
              "username": "joe@example.com",
              "active": true,
              "email": "joe@example.com",
              "id": 3
          },
          "active": true,
```

```
            "bits": 2048,
            "id": 2235,
            "name": "TestAuthority"
        },
        "owner": "secure@example.com",
        "id": 43,
        "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
    }
    "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair. format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

Note this will only show certificates that the current user is authorized to use

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

POST /authorities

Create an authority

Example request:

```
POST /authorities HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "country": "US",
   "state": "California",
   "location": "Los Gatos",
   "organization": "Netflix",
   "organizationalUnit": "Operations",
   "type": "root",
   "signingAlgorithm": "sha256WithRSA",
   "sensitivity": "medium",
   "keyType": "RSA2048",
   "plugin": {
```

```
        "slug": "cloudca-issuer",
    },
    "name": "TimeTestAuthority5",
    "owner": "secure@example.com",
    "description": "test",
    "commonName": "AcommonName",
    "validityYears": "20",
    "extensions": {
        "subAltNames": {
            "names": []
        },
        "custom": []
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "name": "TestAuthority",
  "roles": [{
      "id": 123,
      "name": "secure@example.com"
  }, {
      "id": 564,
      "name": "TestAuthority_admin"
  }, {
      "id": 565,
      "name": "TestAuthority_operator"
  }],
  "options": null,
  "active": true,
  "authorityCertificate": {
      "body": "-----BEGIN CERTIFICATE-----IyMzU5MTVaMHk...",
      "status": true,
      "cn": "AcommonName",
      "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority.",
      "chain": "",
      "notBefore": "2016-06-02T00:00:15+00:00",
      "notAfter": "2023-06-02T23:59:15+00:00",
      "owner": "secure@example.com",
      "user": {
          "username": "joe@example.com",
          "active": true,
          "email": "joe@example.com",
          "id": 3
      },
      "active": true,
      "bits": 2048,
      "id": 2235,
      "name": "TestAuthority"
  },
  "owner": "secure@example.com",
  "id": 43,
  "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
```

```
}
```

Parameters

- **name** – authority's name
- **description** – a sensible description about what the CA with be used for
- **owner** – the team or person who 'owns' this authority
- **validityStart** – when this authority should start issuing certificates
- **validityEnd** – when this authority should stop issuing certificates
- **validityYears** – starting from *now* how many years into the future the authority should be valid
- **extensions** – certificate extensions
- **plugin** – name of the plugin to create the authority
- **type** – the type of authority (root/subca)
- **parent** – the parent authority if this is to be a subca
- **signingAlgorithm** – algorithm used to sign the authority
- **keyType** – key type
- **sensitivity** – the sensitivity of the root key, for CloudCA this determines if the root keys are stored

in an HSM :arg keyName: name of the key to store in the HSM (CloudCA) :arg serialNumber: serial number of the authority :arg firstSerial: specifies the starting serial number for certificates issued off of this authority :reqheader Authorization: OAuth token to authenticate :statuscode 403: unauthenticated :statuscode 200: no error

class lemur.authorities.views. **AuthorityVisualizations**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'authority_visualizations'

get (*authority_id*)

{"name": "flare", "children": [

{ "name": "analytics", "children": [

{ "name": "cluster", "children": [

{"name": "AgglomerativeCluster", "size": 3938}, {"name": "CommunityStructure", "size": 3812}, {"name": "HierarchicalCluster", "size": 6714}, {"name": "MergeEdge", "size": 743}

]

}

}

]}

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.authorities.views. **CertificateAuthority**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'certificateAuthority'

get (**args*, ***kwargs*)

GET /certificates/1/authority
One authority for given certificate

Example request:

```
GET /certificates/1/authority HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "name": "TestAuthority",
  "roles": [{
      "id": 123,
      "name": "secure@example.com"
  }, {
      "id": 564,
      "name": "TestAuthority_admin"
  }, {
      "id": 565,
      "name": "TestAuthority_operator"
  }],
  "options": null,
  "active": true,
  "authorityCertificate": {
      "body": "-----BEGIN CERTIFICATE-----IyMzU5MTVaMHk...",
      "status": true,
      "cn": "AcommonName",
      "description": "This is the ROOT certificate for the TestAuthority
↪certificate authority.",
      "chain": "",
      "notBefore": "2016-06-02T00:00:15+00:00",
      "notAfter": "2023-06-02T23:59:15+00:00",
      "owner": "secure@example.com",
      "user": {
          "username": "joe@example.com",
          "active": true,
          "email": "joe@example.com",
          "id": 3
      },
      "active": true,
      "bits": 2048,
      "id": 2235,
      "name": "TestAuthority"
  },
  "owner": "secure@example.com",
  "id": 43,
```

```
  "description": "This is the ROOT certificate for the TestAuthority␣
↪certificate authority."
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

certificates Package

exceptions Module

models Module

class lemur.certificates.models. **Certificate** (***kwargs*)

Bases: flask_sqlalchemy.Model

active

authority_id

bits

body

chain

cn

country

date_created

deleted

description

destinations

domains

endpoints

expired = <sqlalchemy.sql.elements.Case object>

extensions

get_arn (*account_number*)

Generate a valid AWS IAM arn

:rtype : str :param account_number: :return:

id

issuer

key_type

location

logs

name

not_after

not_before

notifications

notify

organization

organizational_unit

owner

private_key

public_key

replaces

revoked = <sqlalchemy.sql.elements.Case object>

roles

root_authority_id

rotation

san

serial

signing_algorithm

sources

state

status

subject

user_id

validity_range

validity_remaining

lemur.certificates.models. **get_or_increase_name** (*name*)

lemur.certificates.models. **get_sequence** (*name*)

lemur.certificates.models. **update_destinations** (*target*, *value*, *initiator*)

Attempt to upload certificate to the new destination

Parameters

- **target** –
- **value** –

- **initiator** –

Returns

lemur.certificates.models. **update_replacement** (*target*, *value*, *initiator*)
When a certificate is marked as 'replaced' we should not notify.

Parameters

- **target** –
- **value** –
- **initiator** –

Returns

service Module

lemur.certificates.service. **calculate_reissue_range** (*start*, *end*)
Determine what the new validity_start and validity_end dates should be. :param start: :param end: :return:

lemur.certificates.service. **create** (***kwargs*)
Creates a new certificate.

lemur.certificates.service. **create_certificate_roles** (***kwargs*)

lemur.certificates.service. **create_csr** (***csr_config*)
Given a list of domains create the appropriate csr for those domains

Parameters **csr_config** –

lemur.certificates.service. **delete** (*cert_id*)
Delete's a certificate.

Parameters **cert_id** –

lemur.certificates.service. **export** (*cert*, *export_plugin*)
Exports a certificate to the requested format. This format may be a binary format.

Parameters

- **export_plugin** –
- **cert** –

Returns

lemur.certificates.service. **find_duplicates** (*cert*)
Finds certificates that already exist within Lemur. We do this by looking for certificate bodies that are the same. This is the most reliable way to determine if a certificate is already being tracked by Lemur.

Parameters **cert** –

Returns

lemur.certificates.service. **get** (*cert_id*)
Retrieves certificate by its ID.

Parameters **cert_id** –

Returns

lemur.certificates.service. **get_account_number** (*arn*)
Extract the account number from an arn.

Parameters **arn** – IAM SSL arn

Returns account number associated with ARN

lemur.certificates.service. **get_all_certs** ( )
Retrieves all certificates within Lemur.

Returns

lemur.certificates.service. **get_all_pending_cleaning** (*source*)
Retrieves all certificates that are available for cleaning.

Parameters **source** –

Returns

lemur.certificates.service. **get_all_pending_reissue** ( )
Retrieves all certificates that need to be rotated.

Must be X days from expiration, uses *LEMUR_DEFAULT_ROTATION_INTERVAL* to determine how many days from expiration the certificate must be for rotation to be pending.

Returns

lemur.certificates.service. **get_by_name** (*name*)
Retrieves certificate by its Name.

Parameters **name** –

Returns

lemur.certificates.service. **get_certificate_primitives** (*certificate*)
Retrieve key primitive from a certificate such that the certificate could be recreated with new expiration or be used to build upon. :param certificate: :return: dict of certificate primitives, should be enough to effectively re-issue certificate via *create*.

lemur.certificates.service. **get_name_from_arn** (*arn*)
Extract the certificate name from an arn.

Parameters **arn** – IAM SSL arn

Returns name of the certificate as uploaded to AWS

lemur.certificates.service. **import_certificate** (***kwargs*)
Uploads already minted certificates and pulls the required information into Lemur.

This is to be used for certificates that are created outside of Lemur but should still be tracked.

Internally this is used to bootstrap Lemur with external certificates, and used when certificates are 'discovered' through various discovery techniques. was still in aws.

Parameters **kwargs** –

lemur.certificates.service. **mint** (***kwargs*)
Minting is slightly different for each authority. Support for multiple authorities is handled by individual plugins.

lemur.certificates.service. **reissue_certificate** (*certificate*, *replace=None*, *user=None*)
Reissue certificate with the same properties of the given certificate. :param certificate: :param replace: :param user: :return:

lemur.certificates.service. **render** (*args*)
Helper function that allows use to render our REST Api.

Parameters **args** –

Returns

lemur.certificates.service. **stats** (***kwargs*)
Helper that defines some useful statistics about certifications.

Parameters **kwargs** –

Returns

lemur.certificates.service. **update** (*cert_id*, ***kwargs*)
Updates a certificate :param cert_id: :return:

lemur.certificates.service. **upload** (***kwargs*)
Allows for pre-made certificates to be imported into Lemur.

verify Module

lemur.certificates.verify. **crl_verify** (*cert_path*)
Attempts to verify a certificate using CRL.

Parameters **cert_path** –

Returns True if certificate is valid, False otherwise

Raises **Exception** – If certificate does not have CRL

lemur.certificates.verify. **ocsp_verify** (*cert_path*, *issuer_chain_path*)
Attempts to verify a certificate via OCSP. OCSP is a more modern version of CRL in that it will query the OCSP URI in order to determine if the certificate as been revoked

Parameters

- **cert_path** –
- **issuer_chain_path** –

Return bool True if certificate is valid, False otherwise

lemur.certificates.verify. **verify** (*cert_path*, *issuer_chain_path*)
Verify a certificate using OCSP and CRL

Parameters

- **cert_path** –
- **issuer_chain_path** –

Returns True if valid, False otherwise

lemur.certificates.verify. **verify_string** (*cert_string*, *issuer_string*)
Verify a certificate given only it's string value

Parameters

- **cert_string** –
- **issuer_string** –

Returns True if valid, False otherwise

views Module

class lemur.certificates.views. **CertificateExport**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'exportCertificate'

mediatypes (*resource_cls*)

methods = ['POST']

post (**args*, ***kwargs*)

POST /certificates/1/export

Export a certificate

Example request:

```
PUT /certificates/1/export HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "export": {
      "plugin": {
          "pluginOptions": [{
              "available": ["Java Key Store (JKS)"],
              "required": true,
              "type": "select",
              "name": "type",
              "helpMessage": "Choose the format you wish to export",
              "value": "Java Key Store (JKS)"
          }, {
              "required": false,
              "type": "str",
              "name": "passphrase",
              "validation": "^(?=.*[A-Za-z])(?=.*\d)(?=.*[$@$!%*#?&])[A-
↪Za-z\d$@$!%*#?&]{8,}$",
              "helpMessage": "If no passphrase is given one will be␣
↪generated for you, we highly recommend this. Minimum length is 8."
          }, {
              "required": false,
              "type": "str",
              "name": "alias",
              "helpMessage": "Enter the alias you wish to use for the␣
↪keystore."
          }],
          "version": "unknown",
          "description": "Attempts to generate a JKS keystore or␣
↪truststore",
          "title": "Java",
          "author": "Kevin Glisson",
          "type": "export",
          "slug": "java-export"
      }
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript
```

```
{
  "data": "base64encodedstring",
  "passphrase": "UAWOHW#&@_%!tnwmxh832025",
  "extension": "jks"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.certificates.views. **CertificatePrivateKey**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'privateKeyCertificates'

get (*certificate_id*)

GET /certificates/1/key

Retrieves the private key for a given certificate

Example request:

```
GET /certificates/1/key HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "key": "-----BEGIN ...",
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.certificates.views. **Certificates**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'certificate'

get (**args*, ***kwargs*)

GET /certificates/1

One certificate

Example request:

```
GET /certificates/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "status": null,
  "cn": "*.test.example.net",
  "chain": "",
  "authority": {
      "active": true,
      "owner": "secure@example.com",
      "id": 1,
      "description": "verisign test authority",
      "name": "verisign"
  },
  "owner": "joe@example.com",
  "serial": "823110587320259241427891793688893091 56",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
  "description": null,
  "deleted": null,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "replaces": [],
  "replaced": [],
  "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
```

```
  "roles": [{
      "id": 464,
      "description": "This is a google group based role created by Lemur
↪",
      "name": "joe@example.com"
  }],
  "san": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'PUT']

put (**args*, ***kwargs*)

PUT /certificates/1

Update a certificate

Example request:

```
PUT /certificates/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "owner": "jimbob@example.com",
   "active": false
   "notifications": [],
   "destinations": [],
   "replacements": []
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "status": null,
  "cn": "*.test.example.net",
  "chain": "",
  "authority": {
      "active": true,
      "owner": "secure@example.com",
      "id": 1,
      "description": "verisign test authority",
      "name": "verisign"
  },
```

```
  "owner": "joe@example.com",
  "serial": "82311058732025924142789179368889309156",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
  "description": null,
  "deleted": null,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "replaces": [],
  "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
  "roles": [{
      "id": 464,
      "description": "This is a google group based role created by Lemur
↪",
      "name": "joe@example.com"
  }],
  "san": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class `lemur.certificates.views.` **`CertificatesList`**

Bases: `lemur.auth.service.AuthenticatedResource`

Defines the 'certificates' endpoint

`endpoint = 'certificates'`

`get` (**args*, ***kwargs*)

GET /certificates

The current list of certificates

Example request:

```
GET /certificates HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "status": null,
      "cn": "*.test.example.net",
      "chain": "",
      "authority": {
          "active": true,
          "owner": "secure@example.com",
          "id": 1,
          "description": "verisign test authority",
          "name": "verisign"
      },
      "owner": "joe@example.com",
      "serial": "823110587320259241427891793688893091 56",
      "id": 2288,
      "issuer": "SymantecCorporation",
      "notBefore": "2016-06-03T00:00:00+00:00",
      "notAfter": "2018-01-12T23:59:59+00:00",
      "destinations": [],
      "bits": 2048,
      "body": "-----BEGIN CERTIFICATE-----...",
      "description": null,
      "deleted": null,
      "notifications": [{
          "id": 1
      }]
      "signingAlgorithm": "sha256",
      "user": {
          "username": "jane",
          "active": true,
          "email": "jane@example.com",
          "id": 2
      },
      "active": true,
      "domains": [{
          "sensitive": false,
          "id": 1090,
          "name": "*.test.example.net"
      }],
      "replaces": [],
      "replaced": [],
      "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
      "roles": [{
```

```
            "id": 464,
            "description": "This is a google group based role created by␣
↪Lemur",
            "name": "joe@example.com"
        }],
        "san": null
    }],
    "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int. default is 1
- **filter** – key value pair format is k;v
- **count** – count number. default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

POST /certificates

Creates a new certificate

Example request:

```
POST /certificates HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
    "owner": "secure@example.net",
    "commonName": "test.example.net",
    "country": "US",
    "extensions": {
      "subAltNames": {
        "names": [
          {
            "nameType": "DNSName",
            "value": "*.test.example.net"
          },
          {
            "nameType": "DNSName",
            "value": "www.test.example.net"
          }
```

```
        ]
      }
    },
    "replacements": [{
      "id": 1
    },
    "notify": true,
    "validityEnd": "2026-01-01T08:00:00.000Z",
    "authority": {
      "name": "verisign"
    },
    "organization": "Netflix, Inc.",
    "location": "Los Gatos",
    "state": "California",
    "validityStart": "2016-11-11T04:19:48.000Z",
    "organizationalUnit": "Operations"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "status": null,
  "cn": "*.test.example.net",
  "chain": "",
  "authority": {
      "active": true,
      "owner": "secure@example.com",
      "id": 1,
      "description": "verisign test authority",
      "name": "verisign"
  },
  "owner": "joe@example.com",
  "serial": "823110587320259241427891793688893091 56",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
  "description": null,
  "deleted": null,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
```

```
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "replaces": [{
      "id": 1
  }],
  "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
  "roles": [{
      "id": 464,
      "description": "This is a google group based role created by Lemur
↪",
      "name": "joe@example.com"
  }],
  "san": null
}
```

Parameters

- **extensions** – extensions to be used in the certificate
- **description** – description for new certificate
- **owner** – owner email
- **validityStart** – when the certificate should start being valid
- **validityEnd** – when the certificate should expire
- **authority** – authority that should issue the certificate
- **country** – country for the CSR
- **state** – state for the CSR
- **location** – location for the CSR
- **organization** – organization for CSR
- **commonName** – certificate common name

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.certificates.views. **CertificatesReplacementsList**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'replacements'

get (**args*, ***kwargs*)

GET /certificates/1/replacements

One certificate

Example request:

```
GET /certificates/1/replacements HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "status": null,
      "cn": "*.test.example.net",
      "chain": "",
      "authority": {
          "active": true,
          "owner": "secure@example.com",
          "id": 1,
          "description": "verisign test authority",
          "name": "verisign"
      },
      "owner": "joe@example.com",
      "serial": "823110587320259241427891793688893091 56",
      "id": 2288,
      "issuer": "SymantecCorporation",
      "notBefore": "2016-06-03T00:00:00+00:00",
      "notAfter": "2018-01-12T23:59:59+00:00",
      "destinations": [],
      "bits": 2048,
      "body": "-----BEGIN CERTIFICATE-----...",
      "description": null,
      "deleted": null,
      "notifications": [{
          "id": 1
      }]
      "signingAlgorithm": "sha256",
      "user": {
          "username": "jane",
          "active": true,
          "email": "jane@example.com",
          "id": 2
      },
      "active": true,
      "domains": [{
          "sensitive": false,
          "id": 1090,
          "name": "*.test.example.net"
      }],
      "replaces": [],
      "replaced": [],
      "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
      "roles": [{
          "id": 464,
          "description": "This is a google group based role created by␣
↪Lemur",
          "name": "joe@example.com"
```

```
        }],
        "san": null
    }],
    "total": 1
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.certificates.views. **CertificatesStats**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' stats endpoint

endpoint = 'certificateStats'

get ( )

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.certificates.views. **CertificatesUpload**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' upload endpoint

endpoint = 'certificateUpload'

mediatypes (*resource_cls*)

methods = ['POST']

post (**args*, ***kwargs*)

POST /certificates/upload
Upload a certificate

Example request:

```
POST /certificates/upload HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "owner": "joe@example.com",
   "publicCert": "-----BEGIN CERTIFICATE-----...",
   "intermediateCert": "-----BEGIN CERTIFICATE-----...",
   "privateKey": "-----BEGIN RSA PRIVATE KEY-----..."
   "destinations": [],
   "notifications": [],
   "replacements": [],
```

```
   "name": "cert1"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "status": null,
  "cn": "*.test.example.net",
  "chain": "",
  "authority": {
      "active": true,
      "owner": "secure@example.com",
      "id": 1,
      "description": "verisign test authority",
      "name": "verisign"
  },
  "owner": "joe@example.com",
  "serial": "823110587320259241427891793688893091 56",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
  "description": null,
  "deleted": null,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "replaces": [],
  "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
  "roles": [{
      "id": 464,
      "description": "This is a google group based role created by Lemur
↪",
      "name": "joe@example.com"
  }],
  "san": null
}
```

Parameters

- **owner** – owner email for certificate
- **publicCert** – valid PEM public key for certificate

:arg intermediateCert valid PEM intermediate key for certificate :arg privateKey: valid PEM private key for certificate :arg destinations: list of aws destinations to upload the certificate to :reqheader Authorization: OAuth token to authenticate :statuscode 403: unauthenticated :statuscode 200: no error

class lemur.certificates.views. **NotificationCertificatesList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' endpoint

endpoint = 'notificationCertificates'

get (**args*, ***kwargs*)

GET /notifications/1/certificates

The current list of certificates for a given notification

Example request:

```
GET /notifications/1/certificates HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "status": null,
      "cn": "*.test.example.net",
      "chain": "",
      "authority": {
          "active": true,
          "owner": "secure@example.com",
          "id": 1,
          "description": "verisign test authority",
          "name": "verisign"
      },
      "owner": "joe@example.com",
      "serial": "82311058732025924142789179368889309156",
      "id": 2288,
      "issuer": "SymantecCorporation",
      "notBefore": "2016-06-03T00:00:00+00:00",
      "notAfter": "2018-01-12T23:59:59+00:00",
      "destinations": [],
      "bits": 2048,
      "body": "-----BEGIN CERTIFICATE-----...",
      "description": null,
      "deleted": null,
      "notifications": [{
          "id": 1
```

```
      }]
      "signingAlgorithm": "sha256",
      "user": {
          "username": "jane",
          "active": true,
          "email": "jane@example.com",
          "id": 2
      },
      "active": true,
      "domains": [{
          "sensitive": false,
          "id": 1090,
          "name": "*.test.example.net"
      }],
      "replaces": [],
      "replaced": [],
      "name": "WILDCARD.test.example.net-SymantecCorporation-20160603-
↪20180112",
      "roles": [{
          "id": 464,
          "description": "This is a google group based role created by␣
↪Lemur",
          "name": "joe@example.com"
      }],
      "san": null
  }],
  "total": 1
}
```


Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

common Package

health Module

lemur.common.health. **health** ( )

lemur.common.health. **healthcheck** (*db*)

managers Module

class lemur.common.managers. **InstanceManager** (*class_list=None*, *instances=True*)
Bases: object

add (*class_path*)

all ( )
Returns a list of cached instances.

get_class_list ( )

remove (*class_path*)

update (*class_list*)
Updates the class list and wipes the cache.

utils Module

lemur.common.utils. **column_windows** (*session*, *column*, *windowsize*)
Return a series of WHERE clauses against a given column that break it into windows.

Result is an iterable of tuples, consisting of ((start, end), whereclause), where (start, end) are the ids.

Requires a database that supports window functions, i.e. Postgresql, SQL Server, Oracle.

Enhance this yourself ! Add a "where" argument so that windows of just a subset of rows can be computed.

lemur.common.utils. **generate_private_key** (*key_type*)
Generates a new private key based on key_type.

Valid key types: RSA2048, RSA4096

Parameters **key_type** –

Returns

lemur.common.utils. **get_psuedo_random_string** ( )
Create a random and strongish challenge.

lemur.common.utils. **is_weekend** (*date*)
Determines if a given date is on a weekend.

Parameters **date** –

Returns

lemur.common.utils. **parse_certificate** (*body*)
Helper function that parses a PEM certificate.

Parameters **body** –

Returns

lemur.common.utils. **validate_conf** (*app*, *required_vars*)
Ensures that the given fields are set in the applications conf.

Parameters

- **app** –

- **required_vars** – list

lemur.common.utils. **windowed_query** (*q*, *column*, *windowsize*)
"Break a Query into windows on a given column.

destinations Package

models Module

class lemur.destinations.models. **Destination** (***kwargs*)
Bases: flask_sqlalchemy.Model

description

id

label

options

plugin

plugin_name

service Module

lemur.destinations.service. **create** (*label*, *plugin_name*, *options*, *description=None*)
Creates a new destination, that can then be used as a destination for certificates.

Parameters

- **label** – Destination common name
- **description** –

:rtype : Destination :return: New destination

lemur.destinations.service. **delete** (*destination_id*)
Deletes an destination.

Parameters **destination_id** – Lemur assigned ID

lemur.destinations.service. **get** (*destination_id*)
Retrieves an destination by its lemur assigned ID.

Parameters **destination_id** – Lemur assigned ID

:rtype : Destination :return:

lemur.destinations.service. **get_all** ( )
Retrieves all destination currently known by Lemur.

Returns

lemur.destinations.service. **get_by_label** (*label*)
Retrieves a destination by its label

Parameters **label** –

Returns

lemur.destinations.service. **render** (*args*)

lemur.destinations.service. **stats** (***kwargs*)
Helper that defines some useful statistics about destinations.

Parameters **kwargs** –

Returns

lemur.destinations.service. **update** (*destination_id*, *label*, *options*, *description*)
Updates an existing destination.

Parameters

- **destination_id** – Lemur assigned ID
- **label** – Destination common name
- **description** –

:rtype : Destination :return:

views Module

class lemur.destinations.views. **CertificateDestinations**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificate/<int:certificate_id/destinations" endpoint

endpoint = 'certificateDestinations'

get (**args*, ***kwargs*)

GET /certificates/1/destinations
The current account list for a given certificates

Example request:

```
GET /certificates/1/destinations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "description": "test",
      "options": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "id": 4,
      "plugin": {
          "pluginOptions": [{
```

```
                "name": "accountNumber",
                "required": true,
                "value": "111111111111111",
                "helpMessage": "Must be a valid AWS account number!",
                "validation": "/^[0-9]{12,12}$/",
                "type": "str"
            }],
            "description": "Allow the uploading of certificates to AWS IAM
↪",
            "slug": "aws-destination",
            "title": "AWS"
        },
        "label": "test546"
    }
    "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.destinations.views. **Destinations**

Bases: lemur.auth.service.AuthenticatedResource

delete (**args*, ***kw*)

endpoint = 'destination'

get (**args*, ***kwargs*)

GET /destinations/1

Get a specific account

Example request:

```
GET /destinations/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "test",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "111111111111111",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['DELETE', 'GET', 'PUT']

put (**args*, ***kw*)

PUT /destinations/1

Updates an account

Example request:

```
POST /destinations/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript


{
  "description": "test33",
  "options": [{
      "name": "accountNumber",
```

```
      "required": true,
      "value": "34324324",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "34324324",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "test",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "111111111111111",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Parameters

- **accountNumber** – aws account number
- **label** – human readable account label
- **description** – some description about the account

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

class lemur.destinations.views. **DestinationsList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'destinations' endpoint

endpoint = 'destinations'

get (**args*, ***kwargs*)

GET /destinations

The current account list

Example request:

```
GET /destinations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [{
      "description": "test",
      "options": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "id": 4,
      "plugin": {
          "pluginOptions": [{
              "name": "accountNumber",
              "required": true,
              "value": "111111111111111",
              "helpMessage": "Must be a valid AWS account number!",
              "validation": "/^[0-9]{12,12}$/",
              "type": "str"
          }],
          "description": "Allow the uploading of certificates to AWS IAM
↪",
          "slug": "aws-destination",
```

```
            "title": "AWS"
        },
        "label": "test546"
    }
    "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int. default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kw*)

POST /destinations

Creates a new account

Example request:

```
POST /destinations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "description": "test33",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "34324324",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "34324324",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
```

```
      }],
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "test33",
  "options": [{
      "name": "accountNumber",
      "required": true,
      "value": "34324324",
      "helpMessage": "Must be a valid AWS account number!",
      "validation": "/^[0-9]{12,12}$/",
      "type": "str"
  }],
  "id": 4,
  "plugin": {
      "pluginOptions": [{
          "name": "accountNumber",
          "required": true,
          "value": "111111111111111",
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "/^[0-9]{12,12}$/",
          "type": "str"
      }],
      "description": "Allow the uploading of certificates to AWS IAM",
      "slug": "aws-destination",
      "title": "AWS"
  },
  "label": "test546"
}
```

Parameters

- **label** – human readable account label
- **description** – some description about the account

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

class lemur.destinations.views. **DestinationsStats**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificates' stats endpoint

endpoint = 'destinationStats'

get ( )

mediatypes (*resource_cls*)

methods = ['GET']

domains Package

models Module

class lemur.domains.models. **Domain** (***kwargs*)
Bases: flask_sqlalchemy.Model

id

name

sensitive

service Module

lemur.domains.service. **create** (*name*, *sensitive*)
Create a new domain

Parameters

- **name** –
- **sensitive** –

Returns

lemur.domains.service. **get** (*domain_id*)
Fetches one domain

Parameters **domain_id** –

Returns

lemur.domains.service. **get_all** ( )
Fetches all domains

Returns

lemur.domains.service. **get_by_name** (*name*)
Fetches domain by its name

Parameters **name** –

Returns

lemur.domains.service. **render** (*args*)
Helper to parse REST Api requests

Parameters **args** –

Returns

lemur.domains.service. **update** (*domain_id*, *name*, *sensitive*)
Update an existing domain

Parameters

- **domain_id** –
- **name** –
- **sensitive** –

Returns

views Module

class lemur.domains.views. **CertificateDomains**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'domains' endpoint

endpoint = 'certificateDomains'

get (**args*, ***kwargs*)

GET /certificates/1/domains
The current domain list

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "www.example.com",
        "sensitive": false
      },
      {
        "id": 2,
        "name": "www.example2.com",
        "sensitive": false
      }
    ]
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v

- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.domains.views. **Domains**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'domain'

get (**args*, ***kwargs*)

GET /domains/1

Fetch one domain

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "name": "www.example.com",
    "sensitive": false
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'PUT']

put (**args*, ***kwargs*)

GET /domains/1

update one domain

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
    "name": "www.example.com",
    "sensitive": false
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "name": "www.example.com",
    "sensitive": false
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.domains.views. **DomainsList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'domains' endpoint

endpoint = 'domains'

get (**args*, ***kwargs*)

GET /domains

The current domain list

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
```

```
        "id": 1,
        "name": "www.example.com",
        "sensitive": false
      },
      {
        "id": 2,
        "name": "www.example2.com",
        "sensitive": false
      }
    ]
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number. default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

POST /domains

The current domain list

Example request:

```
GET /domains HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "name": "www.example.com",
  "sensitive": false
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
```

```
  "id": 1,
  "name": "www.example.com",
  "sensitive": false
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

notifications Package

models Module

class lemur.notifications.models. **Notification** (***kwargs*)

Bases: flask_sqlalchemy.Model

active

certificates

description

id

label

options

plugin

plugin_name

service Module

lemur.notifications.service. **create** (*label*, *plugin_name*, *options*, *description*, *certificates*)

Creates a new notification.

Parameters

- **label** – Notification label
- **plugin_name** –
- **options** –

- **description** –
- **certificates** –

:rtype : Notification :return:

lemur.notifications.service. **create_default_expiration_notifications** (*name*, *recipients*)

Will create standard 30, 10 and 2 day notifications for a given owner. If standard notifications already exist these will be returned instead of new notifications.

Parameters

- **name** –
- **recipients** –

Returns

lemur.notifications.service. **delete** (*notification_id*)

Deletes an notification.

Parameters **notification_id** – Lemur assigned ID

lemur.notifications.service. **get** (*notification_id*)

Retrieves an notification by its lemur assigned ID.

Parameters **notification_id** – Lemur assigned ID

:rtype : Notification :return:

lemur.notifications.service. **get_all** ( )

Retrieves all notification currently known by Lemur.

Returns

lemur.notifications.service. **get_by_label** (*label*)

Retrieves a notification by its label

Parameters **label** –

Returns

lemur.notifications.service. **render** (*args*)

lemur.notifications.service. **update** (*notification_id*, *label*, *options*, *description*, *active*, *certificates*)

Updates an existing notification.

Parameters

- **notification_id** –
- **label** – Notification label
- **options** –
- **description** –
- **active** –
- **certificates** –

:rtype : Notification :return:

views Module

class lemur.notifications.views. **CertificateNotifications**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'certificate/<int:certificate_id/notifications" endpoint

endpoint = 'certificateNotifications'

get (**args*, ***kwargs*)

GET /certificates/1/notifications

The current account list for a given certificates

Example request:

```
GET /certificates/1/notifications HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
          "description": "An example",
          "options": [
              {
                  "name": "interval",
                  "required": true,
                  "value": 555,
                  "helpMessage": "Number of days to be alert before␣
↪expiration.",
                  "validation": "^\d+$",
                  "type": "int"
              },
              {
                  "available": [
                      "days",
                      "weeks",
                      "months"
                  ],
                  "name": "unit",
                  "required": true,
                  "value": "weeks",
                  "helpMessage": "Interval unit",
                  "validation": "",
                  "type": "select"
              },
              {
                  "name": "recipients",
                  "required": true,
                  "value": "kglisson@netflix.com,example@netflix.com",
                  "helpMessage": "Comma delimited list of email addresses
↪",
```

```
                    "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$
↪",
                    "type": "str"
                }
            ],
            "label": "example",
            "pluginName": "email-notification",
            "active": true,
            "id": 2
        }
    ],
    "total": 1
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.notifications.views. **Notifications**

Bases: lemur.auth.service.AuthenticatedResource

delete (*notification_id*)

endpoint = 'notification'

get (**args*, ***kwargs*)

GET /notifications/1

Get a specific account

Example request:

```
GET /notifications/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript
```

```
{
  "description": "a test",
  "options": [
      {
          "name": "interval",
          "required": true,
          "value": 5,
          "helpMessage": "Number of days to be alert before expiration.",
          "validation": "^\d+$",
          "type": "int"
      },
      {
          "available": [
              "days",
              "weeks",
              "months"
          ],
          "name": "unit",
          "required": true,
          "value": "weeks",
          "helpMessage": "Interval unit",
          "validation": "",
          "type": "select"
      },
      {
          "name": "recipients",
          "required": true,
          "value": "kglisson@netflix.com,example@netflix.com",
          "helpMessage": "Comma delimited list of email addresses",
          "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$",
          "type": "str"
      }
  ],
  "label": "test",
  "pluginName": "email-notification",
  "active": true,
  "id": 2
}
```


Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['DELETE', 'GET', 'PUT']

put (**args*, ***kwargs*)

PUT /notifications/1

Updates an account

Example request:

```
POST /notifications/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "id": 1,
  "accountNumber": 11111111111,
  "label": "labelChanged",
  "comments": "this is a thing"
}
```

Parameters

- **accountNumber** – aws account number
- **label** – human readable account label
- **comments** – some description about the account

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

class lemur.notifications.views. **NotificationsList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'notifications' endpoint

endpoint = 'notifications'

get (**args*, ***kwargs*)

GET /notifications

The current account list

Example request:

```
GET /notifications HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
          "description": "An example",
```

```
          "options": [
              {
                  "name": "interval",
                  "required": true,
                  "value": 5,
                  "helpMessage": "Number of days to be alert before␣
↪expiration.",
                  "validation": "^\d+$",
                  "type": "int"
              },
              {
                  "available": [
                      "days",
                      "weeks",
                      "months"
                  ],
                  "name": "unit",
                  "required": true,
                  "value": "weeks",
                  "helpMessage": "Interval unit",
                  "validation": "",
                  "type": "select"
              },
              {
                  "name": "recipients",
                  "required": true,
                  "value": "kglisson@netflix.com,example@netflix.com",
                  "helpMessage": "Comma delimited list of email addresses
↪",
                  "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$
↪",
                  "type": "str"
              }
          ],
          "label": "example",
          "pluginName": "email-notification",
          "active": true,
          "id": 2
      }
  ],
  "total": 1
}
```


Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

POST /notifications

Creates a new account

Example request:

```
POST /notifications HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
  "description": "a test",
  "options": [
      {
          "name": "interval",
          "required": true,
          "value": 5,
          "helpMessage": "Number of days to be alert before expiration.",
          "validation": "^\d+$",
          "type": "int"
      },
      {
          "available": [
              "days",
              "weeks",
              "months"
          ],
          "name": "unit",
          "required": true,
          "value": "weeks",
          "helpMessage": "Interval unit",
          "validation": "",
          "type": "select"
      },
      {
          "name": "recipients",
          "required": true,
          "value": "kglisson@netflix.com,example@netflix.com",
          "helpMessage": "Comma delimited list of email addresses",
          "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$",
          "type": "str"
      }
  ],
  "label": "test",
  "pluginName": "email-notification",
  "active": true,
  "id": 2
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "description": "a test",
  "options": [
      {
          "name": "interval",
          "required": true,
          "value": 5,
          "helpMessage": "Number of days to be alert before expiration.",
          "validation": "^\d+$",
          "type": "int"
      },
      {
          "available": [
              "days",
              "weeks",
              "months"
          ],
          "name": "unit",
          "required": true,
          "value": "weeks",
          "helpMessage": "Interval unit",
          "validation": "",
          "type": "select"
      },
      {
          "name": "recipients",
          "required": true,
          "value": "kglisson@netflix.com,example@netflix.com",
          "helpMessage": "Comma delimited list of email addresses",
          "validation": "^([\w+-.%]+@[\w-.]+\.[A-Za-z]{2,4},?)+$",
          "type": "str"
      }
  ],
  "label": "test",
  "pluginName": "email-notification",
  "active": true,
  "id": 2
}
```

Parameters

- **accountNumber** – aws account number
- **label** – human readable account label
- **comments** – some description about the account

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

plugins Package

plugins Package

views Module

class lemur.plugins.views. **Plugins**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'plugins' endpoint

endpoint = 'pluginName'

get (**args*, ***kwargs*)

GET /plugins/<name>
The current plugin list

Example request:

```
GET /plugins HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "accountNumber": 222222222,
    "label": "account2",
    "description": "this is a thing"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.plugins.views. **PluginsList**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'plugins' endpoint

endpoint = 'plugins'

get (**args*, ***kwargs*)

GET /plugins
The current plugin list

Example request:

```
GET /plugins HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 2,
        "accountNumber": 222222222,
        "label": "account2",
        "description": "this is a thing"
      },
      {
        "id": 1,
        "accountNumber": 11111111111,
        "label": "account1",
        "description": "this is a thing"
      },
    ]
  "total": 2
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

Subpackages

base Package

base Package

manager Module

class lemur.plugins.base.manager. **PluginManager** (*class_list=None*, *instances=True*)

Bases: lemur.common.managers.InstanceManager

all (*version=1*, *plugin_type=None*)

first (*func_name*, **args*, ***kwargs*)

get (*slug*)

register (*cls*)

unregister (*cls*)

v1 Module

class lemur.plugins.base.v1. **IPlugin**
Bases: thread._local

Plugin interface. Should not be inherited from directly. A plugin should be treated as if it were a singleton. The owner does not control when or how the plugin gets instantiated, nor is it guaranteed that it will happen, or happen more than once. >>> from lemur.plugins import Plugin >>> >>> class MyPlugin(Plugin): >>> def get_title(self): >>> return 'My Plugin' As a general rule all inherited methods should allow `**kwargs` to ensure ease of future compatibility.

author = None

author_url = None

can_disable = True

conf_key = None

conf_title = None

description = None

enabled = True

get_conf_key ( )
Returns a string representing the configuration keyspace prefix for this plugin.

get_conf_title ( )
Returns a string representing the title to be shown on the configuration page.

get_description ( )
Returns the description for this plugin. This is shown on the plugin configuration page. >>> plugin.get_description()

static get_option (*name*, *options*)

get_resource_links ( )
Returns a list of tuples pointing to various resources for this plugin. >>> def get_resource_links(self): >>> return [>>> ('Documentation', 'http://lemury.readthedocs.org'), >>> ('Bug Tracker', 'https://github.com/Netflix/lemur/issues'), >>> ('Source', 'https://github.com/Netflix/lemur'), >>>]

get_title ( )
Returns the general title for this plugin. >>> plugin.get_title()

is_enabled ( )
Returns a boolean representing if this plugin is enabled. If `project` is passed, it will limit the scope to that project. >>> plugin.is_enabled()

options = {}

resource_links = ()

slug = None

title = None

version = None

class lemur.plugins.base.v1. **Plugin**
Bases: lemur.plugins.base.v1.IPlugin

A plugin should be treated as if it were a singleton. The owner does not control when or how the plugin gets instantiated, nor is it guaranteed that it will happen, or happen more than once.

class lemur.plugins.base.v1. **PluginMount**
Bases: type

bases Package

bases Package

destination Module

class lemur.plugins.bases.destination. **DestinationPlugin**
Bases: lemur.plugins.base.v1.Plugin

requires_key = True

slug = 'destinationplugin'

title = 'DestinationPlugin'

type = 'destination'

upload (*name*, *body*, *private_key*, *cert_chain*, *options*, ***kwargs*)

issuer Module

class lemur.plugins.bases.issuer. **IssuerPlugin**
Bases: lemur.plugins.base.v1.Plugin

This is the base class from which all of the supported issuers will inherit from.

create_authority (*options*)

create_certificate (*csr*, *issuer_options*)

slug = 'issuerplugin'

title = 'IssuerPlugin'

type = 'issuer'

notification Module

class lemur.plugins.bases.notification. **ExpirationNotificationPlugin**
Bases: lemur.plugins.bases.notification.NotificationPlugin

This is the base class for all expiration notification plugins. It contains some default options that are needed for all expiration notification plugins.

default_options = [{'helpMessage': 'Number of days to be alert before expiration.', 'required': True, 'type': 'int', 'r

options

send (*notification_type*, *message*, *targets*, *options*, ***kwargs*)

class lemur.plugins.bases.notification. **NotificationPlugin**
Bases: lemur.plugins.base.v1.Plugin

This is the base class from which all of the supported issuers will inherit from.

send (*notification_type*, *message*, *targets*, *options*, ***kwargs*)

slug = 'notificationplugin'

title = 'NotificationPlugin'

type = 'notification'

source Module

class lemur.plugins.bases.source. **SourcePlugin**
Bases: lemur.plugins.base.v1.Plugin

clean (*certificate*, *options*, ***kwargs*)

default_options = [{'default': '60', 'required': False, 'type': 'int', 'name': 'pollRate', 'helpMessage': 'Rate in secon

get_certificates (*options*, ***kwargs*)

get_endpoints (*options*, ***kwargs*)

options

slug = 'sourceplugin'

title = 'SourcePlugin'

type = 'source'

lemur_aws Package

lemur_aws Package

elb Module

iam Module

plugin Module

sts Module

lemur_cfssl Package

lemur_cfssl Package

plugin Module

class lemur.plugins.lemur_cfssl.plugin. **CfsslIssuerPlugin** (**args*, ***kwargs*)
Bases: lemur.plugins.bases.issuer.IssuerPlugin

author = 'Charles Hendrie'

author_url = 'https://github.com/netflix/lemur.git'

static create_authority (*options*)
Creates an authority, this authority is then used by Lemur to allow a user to specify which Certificate Authority they want to sign their certificate.

Parameters **options** –

Returns

create_certificate (*csr*, *issuer_options*)
Creates a CFSSL certificate.

Parameters

- **csr** –
- **issuer_options** –

Returns

description = 'Enables the creation of certificates by CFSSL private CA'

slug = 'cfssl-issuer'

title = 'CFSSL'

version = 'unknown'

lemur_email Package

lemur_email Package

plugin Module

Subpackages

templates Package

config Module

lemur_verisign Package

lemur_verisign Package

constants Module

plugin Module

class lemur.plugins.lemur_verisign.plugin. **VerisignIssuerPlugin** (**args*, ***kwargs*)
Bases: lemur.plugins.bases.issuer.IssuerPlugin

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur.git'

static **create_authority** (*options*)
Creates an authority, this authority is then used by Lemur to allow a user to specify which Certificate Authority they want to sign their certificate.

Parameters **options** –

Returns

create_certificate (*csr*, *issuer_options*)
Creates a Verisign certificate.

Parameters

- **csr** –
- **issuer_options** –

Returns

raise Exception

description = 'Enables the creation of certificates by the VICE2.0 verisign API.'

get_available_units ( )
Uses the Verisign to fetch the number of available units left. This can be used to get tabs on the number of certificates that can be issued.

Returns

get_pending_certificates ( )
Uses Verisign to fetch the number of certificate awaiting approval.

Returns

slug = 'verisign-issuer'

title = 'Verisign'

version = 'unknown'

class lemur.plugins.lemur_verisign.plugin. **VerisignSourcePlugin** (**args*, ***kwargs*)
Bases: lemur.plugins.bases.source.SourcePlugin

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur.git'

description = 'Allows for the polling of issued certificates from the VICE2.0 verisign API.'

get_certificates ( )

slug = 'verisign-source'

title = 'Verisign'

version = 'unknown'

lemur.plugins.lemur_verisign.plugin. **get_additional_names** (*options*)
Return a list of strings to be added to a SAN certificates.

Parameters **options** –

Returns

lemur.plugins.lemur_verisign.plugin. **get_default_issuance** (*options*)
Gets the default time range for certificates

Parameters **options** –

Returns

lemur.plugins.lemur_verisign.plugin. **handle_response** (*content*)
Helper function for parsing responses from the Verisign API. :param content: :return: :raise Exception:

lemur.plugins.lemur_verisign.plugin. **log_status_code** (*r*, **args*, ***kwargs*)
Is a request hook that logs all status codes to the verisign api.

Parameters

- **r** –
- **args** –
- **kwargs** –

Returns

lemur.plugins.lemur_verisign.plugin. **process_options** (*options*)
Processes and maps the incoming issuer options to fields/options that verisign understands

Parameters **options** –

Returns dict or valid verisign options

roles Package

models Module

class lemur.roles.models. **Role** (***kwargs*)
Bases: flask_sqlalchemy.Model

authorities

authority_id

certificates

description

id

name

password

user_id

username

users

service Module

lemur.roles.service. **create** (*name*, *password=None*, *description=None*, *username=None*, *users=None*)
Create a new role

Parameters

- **name** –
- **users** –

- **description** –
- **username** –
- **password** –

Returns

lemur.roles.service. **delete** (*role_id*)
Remove a role

Parameters **role_id** –

Returns

lemur.roles.service. **get** (*role_id*)
Retrieve a role by ID

Parameters **role_id** –

Returns

lemur.roles.service. **get_by_name** (*role_name*)
Retrieve a role by its name

Parameters **role_name** –

Returns

lemur.roles.service. **render** (*args*)
Helper that filters subsets of roles depending on the parameters passed to the REST Api

Parameters **args** –

Returns

lemur.roles.service. **update** (*role_id*, *name*, *description*, *users*)
Update a role

Parameters

- **role_id** –
- **name** –
- **description** –
- **users** –

Returns

views Module

class lemur.roles.views. **AuthorityRolesList**
Bases: lemur.auth.service.AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'authorityRoles'

get (**args*, ***kwargs*)

GET /authorities/1/roles

List of roles for a given authority

Example request:

```
GET /authorities/1/roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "role1",
        "description": "this is role1"
      },
      {
        "id": 2,
        "name": "role2",
        "description": "this is role2"
      }
    ]
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.roles.views. **RoleViewCredentials**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'roleCredentials''

get (*role_id*)

GET /roles/1/credentials

View a roles credentials

Example request:

```
GET /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "username: "ausername",
    "password": "apassword"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.roles.views. **Roles**

Bases: lemur.auth.service.AuthenticatedResource

delete (**args*, ***kw*)

DELETE /roles/1

Delete a role

Example request:

```
DELETE /roles/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "message": "ok"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

endpoint = 'role'

get (**args*, ***kwargs*)

GET /roles/1

Get a particular role

Example request:

```
GET /roles/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "name": "role1",
    "description": "this is role1"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['DELETE', 'GET', 'PUT']

put (**args*, ***kwargs*)

PUT /roles/1

Update a role

Example request:

```
PUT /roles/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "name": "role1",
   "description": "This is a new description"
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "id": 1,
   "name": "role1",
   "description": "this is a new description"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.roles.views. **RolesList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'roles'

get (**args*, ***kwargs*)

GET /roles

The current role list

Example request:

```
GET /roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "role1",
        "description": "this is role1"
      },
      {
        "id": 2,
        "name": "role2",
        "description": "this is role2"
      }
    ]
```

```
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kw*)

POST /roles

Creates a new role

Example request:

```
POST /roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "name": "role3",
   "description": "this is role3",
   "username": null,
   "password": null,
   "users": [
      {'id': 1}
   ]
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 3,
    "description": "this is role3",
    "name": "role3"
}
```

Parameters

- **name** – name for new role
- **description** – description for new role
- **password** – password for new role
- **username** – username for new role
- **users** – list, of users to associate with role

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.roles.views. **UserRolesList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'userRoles'

get (**args*, ***kwargs*)

GET /users/1/roles

List of roles for a given user

Example request:

```
GET /users/1/roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 1,
        "name": "role1",
        "description": "this is role1"
      },
      {
        "id": 2,
        "name": "role2",
        "description": "this is role2"
      }
    ]
  "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v
- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

users Package

models Module

class lemur.users.models. **User** (***kwargs*)

Bases: flask_sqlalchemy.Model

active

authorities

certificates

check_password (*password*)

Hash a given password and check it against the stored value to determine it's validity.

Parameters **password** –

Returns

confirmed_at

email

hash_password ( )

Generate the secure hash for the password.

Returns

id

is_admin

Determine if the current user has the 'admin' role associated with it.

Returns

logs

password

profile_picture

roles

username

lemur.users.models. **hash_password** (*mapper*, *connect*, *target*)
Helper function that is a listener and hashes passwords before insertion into the database.

Parameters

- **mapper** –
- **connect** –
- **target** –

service Module

lemur.users.service. **create** (*username*, *password*, *email*, *active*, *profile_picture*, *roles*)
Create a new user

Parameters

- **username** –
- **password** –
- **email** –
- **active** –
- **profile_picture** –
- **roles** –

Returns

lemur.users.service. **get** (*user_id*)
Retrieve a user from the database

Parameters **user_id** –

Returns

lemur.users.service. **get_all** ( )
Retrieve all users from the database.

Returns

lemur.users.service. **get_by_email** (*email*)
Retrieve a user from the database by their email address

Parameters **email** –

Returns

lemur.users.service. **get_by_username** (*username*)
Retrieve a user from the database by their username

Parameters **username** –

Returns

lemur.users.service. **render** (*args*)
Helper that paginates and filters data when requested through the REST Api

Parameters **args** –

Returns

lemur.users.service. **update** (*user_id*, *username*, *email*, *active*, *profile_picture*, *roles*)
Updates an existing user

Parameters

- **user_id** –
- **username** –
- **email** –
- **active** –
- **profile_picture** –
- **roles** –

Returns

lemur.users.service. **update_roles** (*user*, *roles*)
Replaces the roles with new ones. This will detect when are roles added as well as when there are roles removed.

Parameters

- **user** –
- **roles** –

views Module

class lemur.users.views. **CertificateUsers**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'certificateCreator'

get (**args*, ***kwargs*)

GET /certificates/1/creator
Get a certificate's creator

Example request:

```
GET /certificates/1/creator HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "active": false,
    "email": "user1@example.com",
    "username": "user1",
    "profileImage": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.users.views. **Me**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'me'

get (**args*, ***kwargs*)

GET /auth/me
Get the currently authenticated user

Example request:

```
GET /auth/me HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "active": false,
    "email": "user1@example.com",
    "username": "user1",
    "profileImage": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.users.views. **RoleUsers**
Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'roleUsers'

get (**args*, ***kwargs*)

GET /roles/1/users
Get all users associated with a role

Example request:

```
GET /roles/1/users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
  "items": [
      {
        "id": 2,
        "active": True,
        "email": "user2@example.com",
        "username": "user2",
        "profileImage": null
      },
      {
        "id": 1,
        "active": False,
        "email": "user1@example.com",
        "username": "user1",
        "profileImage": null
      }
    ]
  "total": 2
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET']

class lemur.users.views. **Users**

Bases: lemur.auth.service.AuthenticatedResource

endpoint = 'user'

get (**args*, ***kwargs*)

GET /users/1

Get a specific user

Example request:

```
GET /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 1,
    "active": false,
    "email": "user1@example.com",
    "username": "user1",
    "profileImage": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET', 'PUT']

put (**args*, ***kw*)

PUT /users/1

Update a user

Example request:

```
PUT /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "username": "user1",
   "email": "user1@example.com",
   "active": false,
   "roles": [
       {'id': 1} - or - {'name': 'myRole'}
   ]
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "id": 1,
   "username": "user1",
   "email": "user1@example.com",
   "active": false,
   "profileImage": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

class lemur.users.views. **UsersList**

Bases: lemur.auth.service.AuthenticatedResource

Defines the 'users' endpoint

endpoint = 'users'

get (**args*, ***kwargs*)

GET /users

The current user list

Example request:

```
GET /users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
   "items": [
      {
         "id": 2,
         "active": True,
         "email": "user2@example.com",
         "username": "user2",
         "profileImage": null
      },
      {
         "id": 1,
         "active": False,
         "email": "user1@example.com",
         "username": "user1",
         "profileImage": null
      }
   ]
   "total": 2
}
```

Query Parameters

- **sortBy** – field to sort on
- **sortDir** – asc or desc
- **page** – int default is 1
- **filter** – key value pair format is k;v

- **count** – count number default is 10

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes (*resource_cls*)

methods = ['GET', 'POST']

post (**args*, ***kw*)

POST /users

Creates a new user

Example request:

```
POST /users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript

{
   "username": "user3",
   "email": "user3@example.com",
   "active": true,
   "roles": [
      {'id': 1} - or - {'name': 'myRole'}
   ]
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript

{
    "id": 3,
    "active": True,
    "email": "user3@example.com,
    "username": "user3",
    "profileImage": null
}
```

Parameters

- **username** – username for new user
- **email** – email address for new user
- **password** – password for new user
- **active** – boolean, if the user is currently active
- **roles** – list, roles that the user should be apart of

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

Security

Security

We take the security of `lemur` seriously. The following are a set of policies we have adopted to ensure that security issues are addressed in a timely fashion.

Reporting a security issue

We ask that you do not report security issues to our normal GitHub issue tracker.

If you believe you've identified a security issue with `lemur` , please report it to `cloudsecurity@netflix.com` .

Once you've submitted an issue via email, you should receive an acknowledgment within 48 hours, and depending on the action to be taken, you may receive further follow-up emails.

Supported Versions

At any given time, we will provide security support for the master branch as well as the 2 most recent releases.

Disclosure Process

Our process for taking a security issue from private discussion to public disclosure involves multiple steps.

Approximately one week before full public disclosure, we will send advance notification of the issue to a list of people and organizations, primarily composed of operating-system vendors and other distributors of `lemur` . This notification will consist of an email message containing:

- A full description of the issue and the affected versions of `lemur` .
- The steps we will be taking to remedy the issue.
- The patches, if any, that will be applied to `lemur` .
- The date on which the `lemur` team will apply these patches, issue new releases, and publicly disclose the issue.

Simultaneously, the reporter of the issue will receive notification of the date on which we plan to make the issue public.

On the day of disclosure, we will take the following steps:

- Apply the relevant patches to the `lemur` repository. The commit messages for these patches will indicate that they are for security issues, but will not describe the issue in any detail; instead, they will warn of upcoming disclosure.
- Issue the relevant releases.

If a reported issue is believed to be particularly time-sensitive – due to a known exploit in the wild, for example – the time between advance notification and public disclosure may be shortened considerably.

The list of people and organizations who receives advanced notification of security issues is not, and will not, be made public. This list generally consists of high-profile downstream distributors and is entirely at the discretion of the `lemur` team.

Doing a Release

Doing a release

Doing a release of `lemur` requires a few steps.

Bumping the version number

The next step in doing a release is bumping the version number in the software.

- Update the version number in `lemur/__about__.py`.
- Set the release date in the *Changelog*.
- Do a commit indicating this.
- Send a pull request with this.
- Wait for it to be merged.

Performing the release

The commit that merged the version number bump is now the official release commit for this release. You will need to have `gpg` installed and a `gpg` key in order to do a release. Once this has happened:

- Run `invoke release {version}`.

The release should now be available on PyPI and a tag should be available in the repository.

Verifying the release

You should verify that `pip install lemur` works correctly:

```
>>> import lemur
>>> lemur.__version__
'...'
```

Verify that this is the version you just released.

Post-release tasks

- Update the version number to the next major (e.g. `0.5.dev1`) in `lemur/__about__.py` and
- Add new *Changelog* entry with next version and note that it is under active development
- Send a pull request with these items
- Check for any outstanding code undergoing a deprecation cycle by looking in `lemur.utils` for `DeprecatedIn**` definitions. If any exist open a ticket to increment them for the next release.

CHAPTER 7

FAQ

Frequently Asked Questions

Common Problems

In my startup logs I see *'Aborting... Lemur cannot locate db encryption key, is LEMUR_ENCRYPTION_KEYS set?'* You likely have not correctly configured **LEMUR_ENCRYPTION_KEYS**. See administration/index for more information.

I am seeing Lemur's javascript load in my browser but not the CSS. Ensure that you are placing *include mime.types;* to your Nginx static file location. See *Production* for example configurations.

After installing Lemur I am unable to login Ensure that you are trying to login with the credentials you entered during *lemur init*. These are separate from the postgres database credentials.

Running 'lemur db upgrade' seems stuck. Most likely, the upgrade is stuck because an existing query on the database is holding onto a lock that the migration needs.

To resolve, login to your lemur database and run:

> SELECT * FROM pg_locks l INNER JOIN pg_stat_activity s ON (l.pid = s.pid) WHERE waiting AND NOT granted;

This will give you a list of queries that are currently waiting to be executed. From there attempt to idenity the PID of the query blocking the migration. Once found execute:

> select pg_terminate_backend(<blocking-pid>);

See http://stackoverflow.com/questions/22896496/alembic-migration-stuck-with-postgresql for more.

How do I

... script the Lemur installation to bootstrap things like roles and users? Lemur is a simple Flask (Python) application that runs using a utility runner. A script that creates a project and default user might look something like this:

```
# Bootstrap the Flask environment
from flask import current_app

from lemur.users.service import create as create_user
from lemur.roles.service import create as create_role
from lemur.accounts.service import create as create_account
```

```
role = create_role('aRole', 'this is a new role')
create_user('admin', 'password', 'lemur@nobody', True, [role]
```

CHAPTER 8

Reference

Changelog

0.6 - *master*

Note: This version is not yet released and is under active development

0.5 - *2016-04-08*

This release is most notable for dropping support for python2.7. All Lemur versions >0.4 will now support python3.5 only.

Big thanks to neilschelly for quite a lot of improvements to the *lemur-cryptography* plugin.

Other Highlights:

- Closed #501 - Endpoint resource as now kept in sync via an

expiration mechanism. Such that non-existant endpoints gracefully fall out of Lemur. Certificates are never removed from Lemur.

- Closed #551 - Added the ability to create a 4096 bit key during certificate

creation. Closed #528 to ensure that issuer plugins supported the new 4096 bit keys.

- Closed #566 - Fixed an issue changing the notification status for certificates

without private keys.

- Closed #594 - Added *replaced* field indicating if a certificate has been superseded.
- Closed #602 - AWS plugin added support for ALBs for endpoint tracking.

Special thanks to all who helped with with this release, notably:

- RcRonco
- harmw
- jeremyguarini

See the full list of issues closed in 0.5.

Upgrading

Note: This release will need a slight migration change. Please follow the documentation to upgrade Lemur.

0.4 - *2016-11-17*

There have been quite a few issues closed in this release. Some notables:

- Closed #284 - Created new models for *Endpoints* created associated

AWS ELB endpoint tracking code. This was the major stated goal of this milestone and should serve as the basis for future enhancements of Lemur's certificate 'deployment' capabilities.

- Closed #334 - Lemur not has the ability

to restrict certificate expiration dates to weekdays.

Several fixes/tweaks to Lemurs python3 support (thanks chadhendrie!)

This will most likely be the last release to support python2.7 moving Lemur to target python3 exclusively. Please comment on issue #340 if this negatively affects your usage of Lemur.

See the full list of issues closed in 0.4.

Upgrading

Note: This release will need a slight migration change. Please follow the documentation to upgrade Lemur.

0.3.0 - *2016-06-06*

This is quite a large upgrade, it is highly advised you backup your database before attempting to upgrade as this release requires the migration of database structure as well as data.

Upgrading

Please follow the documentation to upgrade Lemur.

Source Plugin Owners

The dictionary returned from a source plugin has changed keys from *public_certificate* to *body* and *intermediate_certificate* to chain.

Issuer Plugin Owners

This release may break your plugins, the keys in *issuer_options* have been changed from *camelCase* to *under_score*. This change was made to break a undue reliance on downstream options maintains a more pythonic naming convention. Renaming these keys should be fairly trivial, additionally pull requests have been submitted to affected plugins to help ease the transition.

Note: This change only affects issuer plugins and does not affect any other types of plugins.

- **Closed #63 - Validates all endpoints with Marshmallow schemas, this allows for** stricter input validation and better error messages when validation fails.
- Closed #146 - Moved authority type to first pane of authority creation wizard.
- **Closed #147 - Added and refactored the relationship between authorities and their** root certificates. Displays the certificates (and chains) next the the authority in question.
- **Closed #199 - Ensures that the dates submitted to Lemur during authority and** certificate creation are actually dates.
- **Closed #230 - Migrated authority dropdown to a ui-select based dropdown, this** should be easier to determine what authorities are available and when an authority has actually been selected.
- **Closed #254 - Forces certificate names to be generally unique. If a certificate name** (generated or otherwise) is found to be a duplicate we increment by appending a counter.
- **Closed #254 - Switched to using Fernet generated passphrases for exported items.** These are more sounds that pseudo random passphrases generated before and have the nice property of being in base64.
- **Closed #278 - Added ability to specify a custom name to certificate creation, previously** this was only available in the certificate import wizard.
- **Closed #281 - Fixed an issue where notifications could not be removed from a certificate** via the UI.
- Closed #289 - Fixed and issue where intermediates were not being properly exported.
- **Closed #315 - Made how roles are associated with certificates and authorities much more** explict, including adding the ability to add roles directly to certificates and authorities on creation.

0.2.2 - 2016-02-05

- **Closed #234 - Allows export plugins to define whether they need** private key material (default is True)
- **Closed #231 - Authorities were not respecting 'owning' roles and their** users
- Closed #228 - Fixed documentation with correct filter values
- **Closed #226 - Fixes issue were *import_certificate* was requiring** replacement certificates to be specified
- Closed #224 - Fixed an issue where NPM might not be globally available (thanks AlexClineBB!)
- **Closed #221 - Fixes several reported issues where older migration scripts were** missing tables, this change removes pre 0.2 migration scripts
- Closed #218 - Fixed an issue where export passphrases would not validate

0.2.1 - 2015-12-14

- Fixed bug with search not refreshing values
- Cleaned up documentation, including working supervisor example (thanks rpicard!)
- Closed #165 - Fixed an issue with email templates
- Closed #188 - Added ability to submit third party CSR
- Closed #176 - Java-export should allow user to specify truststore/keystore

- Closed #176 - Extended support for exporting certificate in P12 format

0.2.0 - 2015-12-02

- Closed #120 - Error messages not displaying long enough
- Closed #121 - Certificate create form should not be valid until a Certificate Authority object is available
- **Closed #122 - Certificate API should allow for the specification of preceding certificates** You can now target a certificate(s) for replacement. When specified the replaced certificate will be marked as 'inactive'. This means that there will be no notifications for that certificate.
- Closed #139 - SubCA autogenerated descriptions for their certs are incorrect
- Closed #140 - Permalink does not change with filtering
- Closed #144 - Should be able to search certificates by domains covered, included wildcards
- Closed #165 - Cleaned up expiration notification template
- Closed #160 - Cleaned up quickstart documentation (thanks forkd!)
- Closed #144 - Now able to search by all domains in a given certificate, not just by common name

0.1.5 - 2015-10-26

- **SECURITY ISSUE**: Switched from use a AES static key to Fernet encryption. Affects all versions prior to 0.1.5. If upgrading this will require a data migration. see: Upgrading Lemur

License

Lemur is licensed under a three clause APACHE License.

The full license text can be found below (*Lemur License*).

Authors

Lemur was originally written and is maintained by Kevin Glisson.

A list of additional contributors can be seen on GitHub.

Lemur License

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity,

or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

/auth

/authorities

/certificates

/destinations

/domains

/notifications

/plugins

/roles

/users

l

A

C

D

E

G

I

L

M

N

O

P

R

S

U