Lemur Documentation

Release 1.3.dev0

Netflix Security

Apr 04, 2023

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

ONE

INSTALLATION

1.1 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 18.04/20.04 instance, commands may differ based on the OS and configuration being used.

For a quicker alternative, see the Lemur docker file on Github.

1.1.1 Dependencies

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

- A UNIX-based operating system (we test on Ubuntu, develop on macOS)
- Python 3.7 or greater
- PostgreSQL 9.4 or greater
- Nginx
- Node v10.x (LTS)

Note: Ubuntu 18.04 supports by default Python 3.6.x and Node v8.x

Note: Lemur was built with 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.

1.1.2 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 its dependencies:

```
sudo apt-get update
sudo apt-get install nodejs npm python-pip python-dev python3-dev libpq-dev build-
↪essential libssl-dev libffi-dev libsasl2-dev libldap2-dev nginx git supervisor␣
↪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).

Note: Installing node from a package manager may create the nodejs bin at /usr/bin/nodejs instead of /usr/bin/node. If that is the case run the following sudo ln -s /usr/bin/nodejs /usr/bin/node

Now, install Python `virtualenv` package:

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

1.1.3 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}
```

1.1.4 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 set up in your environment.

1.1.5 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://username: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
```

1.1.6 Set Up 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)

1.1.7 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.

Your database installation requires the pg_trgm extension. If you do not have this installed already, you can allow the script to install this for you by adding the SUPERUSER permission to the lemur database user.

```
sudo -u postgres -i
psql
postgres=# ALTER USER lemur WITH SUPERUSER
```

Additional notifications can be created through the UI or API. See *Notification Options* 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: If you added the SUPERUSER permission to the lemur database user above, it is recommended you revoke that permission now.

```
sudo -u postgres -i
psql
postgres=# ALTER USER lemur WITH NOSUPERUSER
```

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 a New User* and *Command Line Interface* for details.

1.1.8 Set Up 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 to set up 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
```

1.1.9 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/`.

1.1.10 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.

1.1.11 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
```

1.1.12 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.

Automated celery tasks

Please refer to *Periodic Tasks* to learn more about task scheduling in Lemur.

1.1.13 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.

1.2 Production

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

1.2.1 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

The original *WES-entropy-client* repository by WhitewoodCrypto was removed, the link now points to a fork of it.

For additional information about OpenSSL entropy issues:

- Managing and Understanding Entropy Usage

1.2.2 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;
```

(continues on next page)

(continued from previous page)

```
    }

    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;
```

(continues on next page)

(continued from previous page)

```
    # 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
```

(continues on next page)

(continued from previous page)

```
    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

1.2.3 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.

1.2.4 Periodic Tasks

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

The following commands that could/should be run on a periodic basis:

- *notify expirations*, *notify authority_expirations*, *notify security_expiration_summary*, and *notify expiring_deployed_certificates* (see *Notification Options* for configuration info)
- *certificate identify_expiring_deployed_certificates*
- *check_revoked*
- *sync*

How often you run these commands is largely up to the user. *notify* should be run once a day (more often will result in duplicate notifications). *check_revoked* is typically run at least once a day. *sync* is typically run every 15 minutes. *fetch_all_pending_acme_certs* should be ran frequently (Every minute is fine). *remove_old_acme_certs* can be ran more rarely, such as once every week.

Example cron entries:

```
0 22 * * * lemuruser export LEMUR_CONF=/Users/me/.lemur/lemur.conf.py; /www/lemur/bin/
↪lemur notify expirations
0 22 * * * lemuruser export LEMUR_CONF=/Users/me/.lemur/lemur.conf.py; /www/lemur/bin/
↪lemur notify authority_expirations
0 22 * * * lemuruser export LEMUR_CONF=/Users/me/.lemur/lemur.conf.py; /www/lemur/bin/
↪lemur notify security_expiration_summary
*/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
```

If you are using LetsEncrypt, you must also run the following:

- *fetch_all_pending_acme_certs*
- *remove_old_acme_certs*

Rarely, lemur may see duplicate certificates issue with LetsEncrypt. This is because of the retry logic during resolution of pending certificates. To deduplicate these certificates, please consider running the celery task *disable_rotation_of_duplicate_certificates*. This task will identify duplicate certificates and disable auto rotate if it's confident that the certificate is not being used. If certificate is in use, no change is done (operation status = skipped). If unused, auto-rotation will be disabled (operation status = success). If it's not able to confidently determine that certificates are duplicates, operation status will result in *failed* for that specific set of certificates. You may want to manually check these certs to determine if you want to keep them all. The task will always keep auto-rotate on for at least one certificate.

For better metrics around job completion, we recommend using celery to schedule recurring jobs in Lemur.

Example Celery configuration (To be placed in your configuration file):

```
CELERYBEAT_SCHEDULE = {
    'fetch_all_pending_acme_certs': {
        'task': 'lemur.common.celery.fetch_all_pending_acme_certs',
        'options': {
            'expires': 180
        },
        'schedule': crontab(minute="*"),
    },
    'remove_old_acme_certs': {
        'task': 'lemur.common.celery.remove_old_acme_certs',
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour=7, minute=30, day_of_week=1),
    },
    'clean_all_sources': {
        'task': 'lemur.common.celery.clean_all_sources',
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour=1, minute=0, day_of_week=1),
    },
    'sync_all_sources': {
        'task': 'lemur.common.celery.sync_all_sources',
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour="*/3", minute=5),
    },
    'notify_expirations': {
        'task': 'lemur.common.celery.notify_expirations',
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour=22, minute=0),
    },
    'notify_authority_expirations': {
        'task': 'lemur.common.celery.notify_authority_expirations',
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour=22, minute=0),
    },
    'send_security_expiration_summary': {
        'task': 'lemur.common.celery.send_security_expiration_summary',
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour=22, minute=0),
    },
    'disable_rotation_of_duplicate_certificates': {
        'task': 'lemur.common.celery.disable_rotation_of_duplicate_certificates',
```

(continues on next page)

(continued from previous page)

```
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour=22, minute=0, day_of_week=2),
    },
    'notify_expiring_deployed_certificates': {
        'task': 'lemur.common.celery.notify_expiring_deployed_certificates',
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour=22, minute=0),
    },
    'identify_expiring_deployed_certificates': {
        'task': 'lemur.common.celery.identify_expiring_deployed_certificates',
        'options': {
            'expires': 180
        },
        'schedule': crontab(hour=20, minute=0),
    }
}
```

To enable celery support, you must also have configuration values that tell Celery which broker and backend to use. Here are the Celery configuration variables that should be set:

```
CELERY_RESULT_BACKEND = 'redis://your_redis_url:6379'
CELERY_BROKER_URL = 'redis://your_redis_url:6379/0'
CELERY_IMPORTS = ('lemur.common.celery')
CELERY_TIMEZONE = 'UTC'

REDIS_HOST="your_redis_url"
REDIS_PORT=6379
REDIS_DB=0
```

Out of the box, every Redis instance supports 16 databases. The default database (*REDIS_DB*) is set to 0, however, you can use any of the databases from 0-15. Via *redis.conf* more databases can be supported. In the *redis://* url, the database number can be added with a slash after the port. (defaults to 0, if omitted)

Do not forget to import crontab module in your configuration file:

```
from celery.task.schedules import crontab
```

You must start a single Celery scheduler instance and one or more worker instances in order to handle incoming tasks. The scheduler can be started with:

```
LEMUR_CONF='/location/to/conf.py' /location/to/lemur/bin/celery -A lemur.common.celery␣
↪beat
```

And the worker can be started with desired options such as the following:

```
LEMUR_CONF='/location/to/conf.py' /location/to/lemur/bin/celery -A lemur.common.celery␣
↪worker --concurrency 10 -E -n lemurworker1@%%h
```

supervisor or systemd configurations should be created for these in production environments as appropriate.

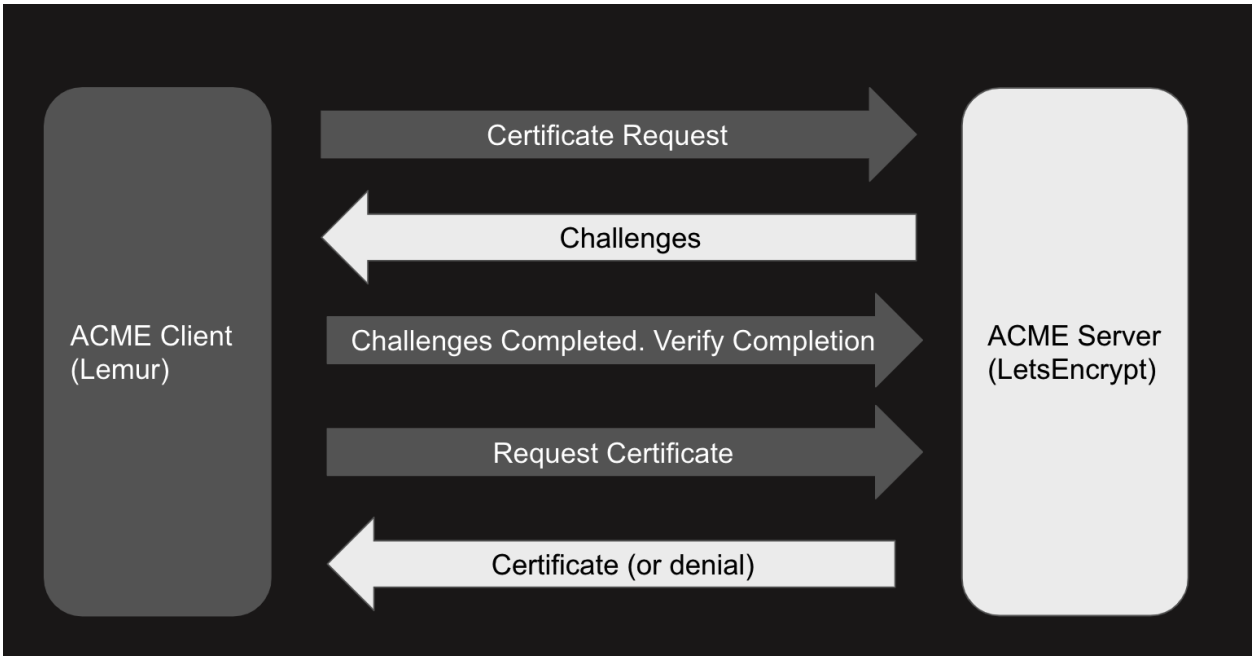
1.2.5 Add support for LetsEncrypt/ACME

LetsEncrypt is a free, limited-feature certificate authority that offers publicly trusted certificates that are valid for 90 days. LetsEncrypt does not use organizational validation (OV), and instead relies on domain validation (DV). LetsEncrypt requires that we prove ownership of a domain before we're able to issue a certificate for that domain, each time we want a certificate.

The most common methods to prove ownership are HTTP validation and DNS validation. Lemur supports DNS validation through the creation of DNS TXT records as well as HTTP validation, reusing the destination concept.

ACME DNS Challenge

In a nutshell, when we send a certificate request to LetsEncrypt, they generate a random token and ask us to put that token in a DNS text record to prove ownership of a domain. If a certificate request has multiple domains, we must prove ownership of all of these domains through this method. The token is typically written to a TXT record at -acme_challenge.domain.com. Once we create the appropriate TXT record(s), Lemur will try to validate propagation before requesting that LetsEncrypt finalize the certificate request and send us the certificate.



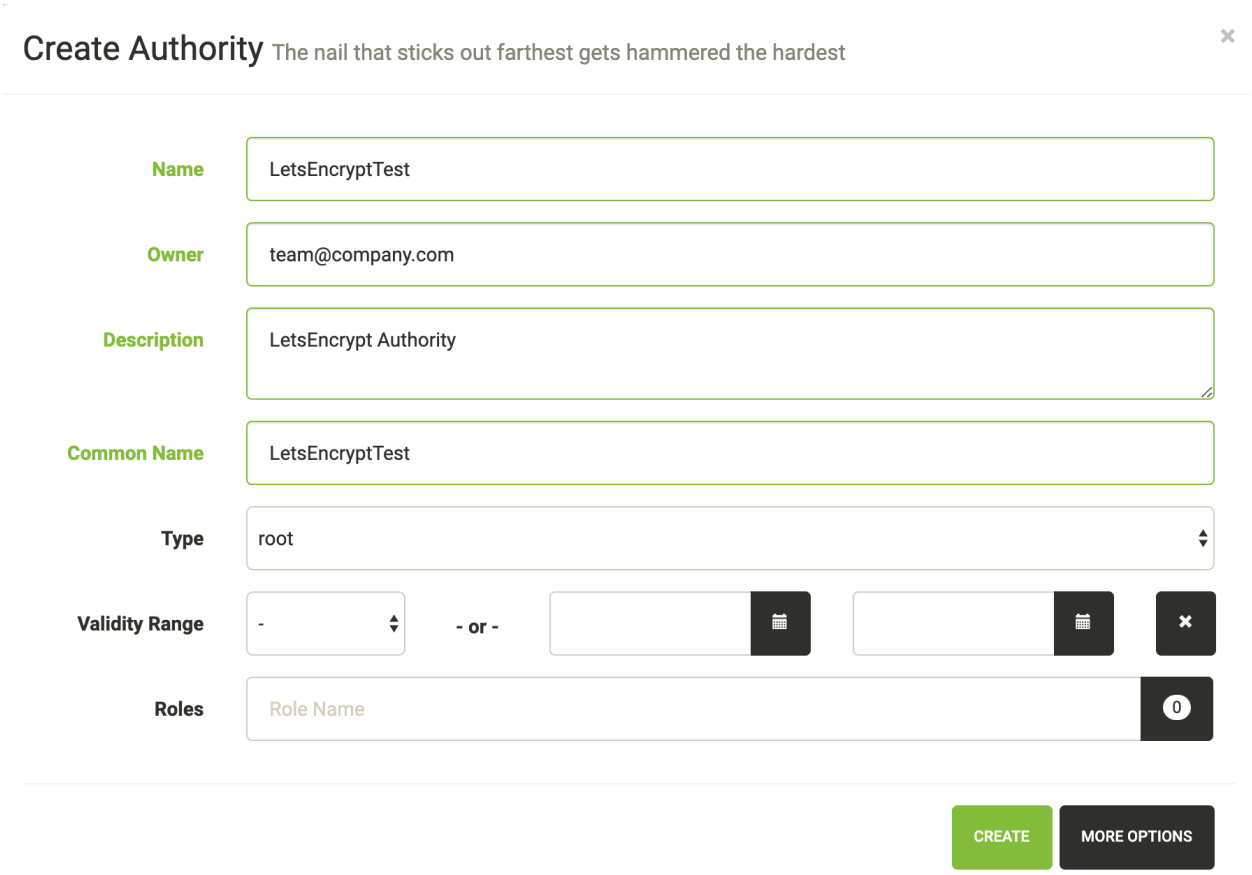
To start issuing certificates through LetsEncrypt, you must enable Celery support within Lemur first[*]_. After doing so, you need to create a LetsEncrypt authority. To do this, visit Authorities -> Create. Set the applicable attributes and click "More Options".

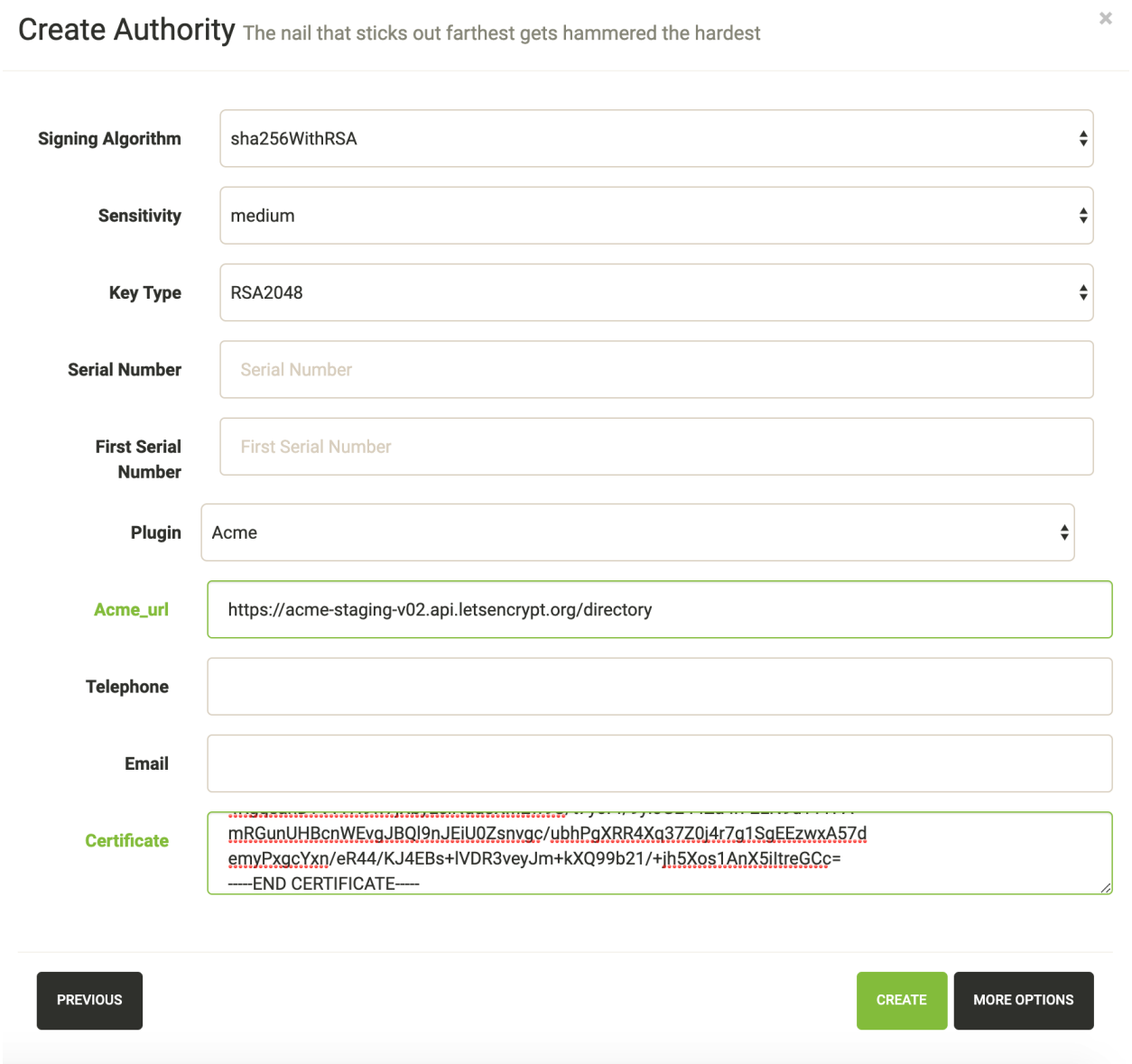
You will need to set "Certificate" to LetsEncrypt's active chain of trust for the authority you want to use. To find the active chain of trust at the time of writing, please visit LetsEncrypt.

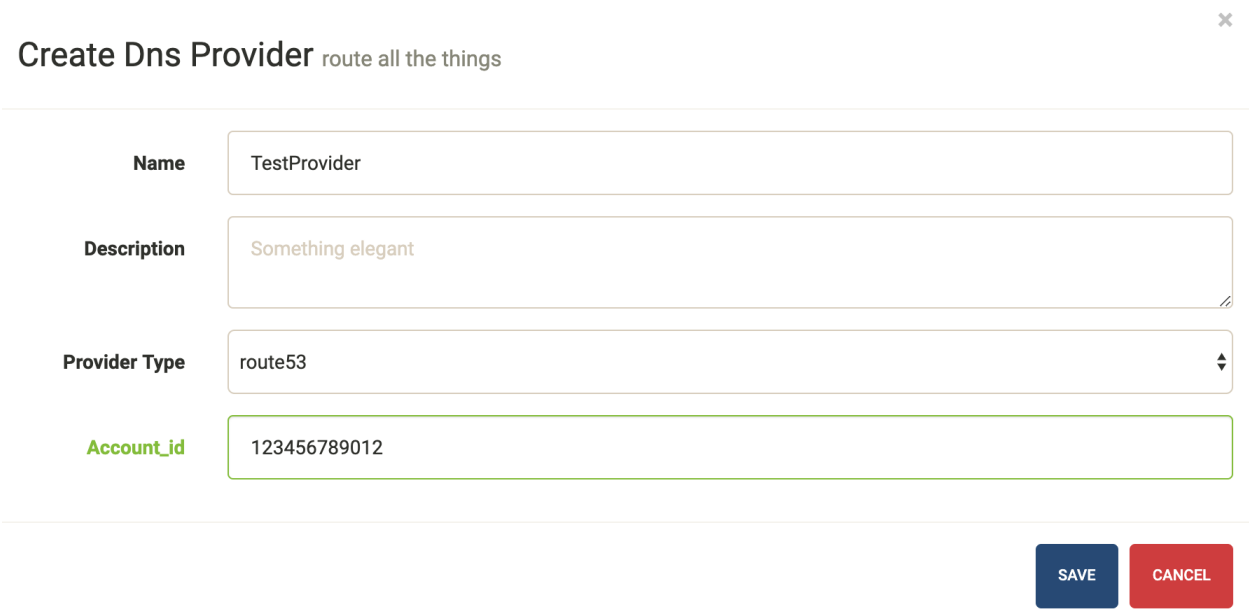
Under Acme_url, enter in the appropriate endpoint URL. Lemur supports LetsEncrypt's V2 API, and we recommend you to use this. At the time of writing, the staging and production URLs for LetsEncrypt V2 are https://acme-staging-v02.api.letsencrypt.org/directory and https://acme-v02.api.letsencrypt.org/directory.

After creating the authorities, we will need to create a DNS provider. Visit *Admin -> DNS Providers* and click *Create*. Lemur comes with a few provider plugins built in, with different options. Create a DNS provider with the appropriate choices.

By default, users will need to select the DNS provider that is authoritative over their domain in order for the LetsEncrypt flow to function. However, Lemur will attempt to automatically determine the appropriate provider if possible. To


Create Authority
The nail that sticks out farthest gets hammered the hardest
Name
LetsEncryptTest
Owner
team@company.com
Description
LetsEncrypt Authority
Common Name
LetsEncryptTest
Type
root
Validity Range
-
- or -
Roles
Role Name
0
CREATE
MORE OPTIONS


Create Authority
The nail that sticks out farthest gets hammered the hardest
Signing Algorithm
sha256WithRSA
Sensitivity
medium
Key Type
RSA2048
Serial Number
Serial Number
First Serial Number
First Serial Number
Plugin
Acme
Acme_url
https://acme-staging-v02.api.letsencrypt.org/directory
Telephone
Email
Certificate
mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzwxA57d
emyPxgcYxn/eR44/KJ4EBs+lVDR3veyJm+kXQ99b21/+jh5Xos1AnX5iltreGCc=
-----END CERTIFICATE-----
PREVIOUS
CREATE
MORE OPTIONS



enable this functionality, periodically (or through Cron/Celery) run *lemur dns_providers get_all_zones*. This command will traverse all DNS providers, determine which zones they control, and upload this list of zones to Lemur's database (in the dns_providers table). Alternatively, you can manually input this data.

ACME HTTP Challenge

The flow for requesting a certificate using the HTTP challenge is not that different from the one described for the DNS challenge. The only difference is, that instead of creating a DNS TXT record, a file is uploaded to a Webserver which serves the file at *http://<domain>/.well-known/acme-challenge/<token>*

Currently the HTTP challenge also works without Celery, since it's done while creating the certificate, and doesn't rely on celery to create the DNS record. This will change when we implement mix & match of acme challenge types.

To create a HTTP compatible Authority, you first need to create a new destination that will be used to deploy the challenge token. Visit *Admin -> Destination* and click *Create*. The path you provide for the destination needs to be the exact path that is called when the ACME providers calls *http://<domain>/.well-known/acme-challenge/*. The token part will be added dynamically by the acme_upload. Currently only the SFTP and S3 Bucket destination support the ACME HTTP challenge.

Afterwards you can create a new certificate authority as described in the DNS challenge, but need to choose *Acme HTTP-01* as the plugin type, and then the destination you created beforehand.

LetsEncrypt: pinning to cross-signed ICA

Let's Encrypt has been using a cross-signed intermediate CA by DST Root CA X3, which is included in many older devices' TrustStore.

Let's Encrypt is transitioning to use the intermediate CA issued by their own root (ISRG X1) starting from September 29th 2020. This is in preparation of concluding the initial bootstrapping of their CA, by having it cross-signed by an older CA.

Lemur can temporarily pin to the cross-signed intermediate CA (same public/private key pair as the ICA signed by ISRG X1). This will prolong support for incompatible devices.

The following must be added to the config file to activate the pinning (the pinning will be removed by September 2021):

```
# remove or update after Mar 17 16:40:46 2021 GMT
IDENTRUST_CROSS_SIGNED_LE_ICA_EXPIRATION_DATE = "17/03/21"
IDENTRUST_CROSS_SIGNED_LE_ICA = """
-----BEGIN CERTIFICATE-----
MIIEkjCCA3qgAwIBAgIQCgFBQgAAAVOFc2oLheynCDANBgkqhkiG9w0BAQsFADA/
MSQwIgYDVQQKExtEaWdpdGFsIFNpZ25hdHVyZSBUcnVzdCBDby4xFzAVBgNVBAMT
DkRTVCBSb290IENBIFgzMB4XDTE2MDMxNzE2NDA0NloXDTIxMDMxNzE2NDA0Nlow
SjELMAkGA1UEBhMCVVMxFjAUBgNVBAoTDUxldCdzIEVuY3J5cHQxIzAhBgNVBAMT
GkxldCdzIEVuY3J5cHQgQXV0aG9yaXR5IFgzMIIBIjANBgkqhkiG9w0BAQEFAAOC
AQ8AMIIBCgKCAQEAnNMM8FrlLke3cl03g7NoYzDq1zUmGSXhvb418XCSL7e4S0EF
q6meNQhY7LEqxGiHC6PjdeTm86dicbp5gWAf15Gan/PQeGdxyGkOlZHP/uaZ6WA8
SMx+yk13EiSdRxta67nsHjcAHJyse6cF6s5K671B5TaYucv9bTyWaN8jKkKQDIZ0
Z8h/pZq4UmEUEz9l6YKHy9v6Dlb2honzhT+Xhq+w3Brvaw2VFn3EK6BlspkENnWA
a6xK8xuQSXgvopZPKiAlKQTGdMDQMc2PMTiVFrqoM7hD8bEfwzB/onkxEz0tNvjj
/PIzark5McWvxI0NHWQWM6r6hCm21AvA2H3DkwIDAQABo4IBfTCCAXkwEgYDVR0T
AQH/BAgwBgEB/wIBADAOBgNVHQ8BAf8EBAMCAYYwfwYIKwYBBQUHAQEEczBxMDIG
CCsGAQUFBzABhiZodHRwOi8vaXNyZy50cnVzdGlkLm9jc3AuaWRlbnRydXN0LmNv
bTA7BggrBgEFBQcwAoYvaHR0cDovL2FwcHMuaWRlbnRydXN0LmNvbS9yb290cy9k
c3Ryb290Y2F4My5wN2MwHwYDVR0jBBgwFoAUxKexpHsscfrb4UuQdf/EFWCFiRAw
VAYDVR0gBE0wSzAIBgZngQwBAgEwPwYLKwYBBAGC3xMBAQEwMDAuBggrBgEFBQcC
ARYiaHR0cDovL2Nwcy5yb290LXgxLmxldHNlbmNyeXB0Lm9yZzA8BgNVHR8ENTAz
MDGgL6AthitodHRwOi8vY3JsLmlkZW50cnVzdC5jb20vRFNUUk9PVENBWDNDUkwu
Y3JsMB0GA1UdDgQWBBSoSmpjBH3duubRObemRWXv86jsoTANBgkqhkiG9w0BAQsF
AAOCAQEA3TPXEfNjWDjdGBX7CVW+dla5cEilaUcne8IkCJLxWh9KEik3JHRRHGJo
uM2VcGfl96S8TihRzZvoroed6ti6WqEBmtzw3Wodatg+VyOeph4EYpr/1wXKtx8/
wApIvJSwtmVi4MFU5aMqrSDE6ea73Mj2tcMyo5jMd6jmeWUHK8so/joWUoHOUgwu
X4Po1QYz+3dszkDqMp4fklxBwXRsW10KXzPMTZ+sOPAveyxindmjkW8lGy+QsRlG
PfZ+G6Z6h7mjem0Y+iWlkYcV4PIWL1iwBi8saCbGS5jN2p8M+X+Q7UNKEkROb3N6
KOqkqm57TH2H3eDJAkSnh6/DNFu0Qg==
-----END CERTIFICATE-----
"""
```

LetsEncrypt: Using a pre-existing ACME account

Let's Encrypt allows reusing an existing ACME account, to create and especially revoke certificates. The current implementation in the acme plugin, allows for a primary account for all ACME authorities. To use an existing account as the primary ACME account, you need to configure the *ACME_PRIVATE_KEY* and *ACME_REGR* variables in the lemur configuration.

Alternatively, you can set *acme_regr* and *acme_private_key* as options during setup of a new issuer in Lemur. Lemur will use the *LEMUR_ENCRYPTION_KEYS* to encrypt the *acme_private_key* before storing it in the database.

ACME_PRIVATE_KEY needs to be in the JWK format:

```
{
    "kty": "RSA",
    "n": "yr1qBwHizA7ME_iV32bY10ILp.....",
    "e": "AQAB",
    "d": "llBlYhil3I.....",
    "p": "-5LW2Lewogo.........",
    "q": "zk6dHqHfHksd.........",
```

(continues on next page)

(continued from previous page)

```
    "dp": "qfe9fFIu3mu.......",
    "dq": "cXFO-loeOyU.......",
    "qi": "AfK1sh0_8sLTb..........."
}
```

Using *python-jwt* converting an existing private key in PEM format is quite easy:

```
import python_jwt as jwt, jwcrypto.jwk as jwk

priv_key = jwk.JWK.from_pem(b"""-----BEGIN RSA PRIVATE KEY-----
...
-----END RSA PRIVATE KEY-----""")

print(priv_key.export())
```

ACME_REGR needs to be a valid JSON with a *body* and a *uri* attribute, similar to this:

```
{"body": {}, "uri": "https://acme-staging-v02.api.letsencrypt.org/acme/acct/<ACCOUNT_
↪NUMBER>"}
```

The URI can be retrieved from the ACME create account endpoint when creating a new account, using the existing key.

LetsEncrypt: Setting up a new ACME account

In case, you are not using the *ACME_PRIVATE_KEY* and *ACME_REGR* variables in the Lemur configuration to set up a pre-existing primary, Lemur will create a new account on the fly for you. Additionally, you can select the *store_account* while setting a new ACME-based issuer in Lemur, to avoid hitting rate limits for creating new accounts for each request.

External Account Binding (EAB):

The ACME protocol enables setting up a new ACME account linked to an existing external account. For this, your CA needs to issue you an hmac_key and kid, which you need while setting up a new ACME issuer in Lemur. hmac_key and kid are usually short-lived and are used to create a new account. When *store_account* is set in the options of a new issuer, Lemur will use the EAB credentials to set up a new account.

CHAPTER

TWO

USER GUIDE

2.1 User Guide

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

2.1.1 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.

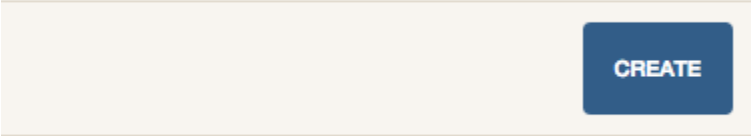


Fig. 1: In the authority table select "Create"

2.1.2 Create a New Certificate

2.1.3 Import an Existing Certificate

2.1.4 Create a New User

2.1.5 Create a New Role

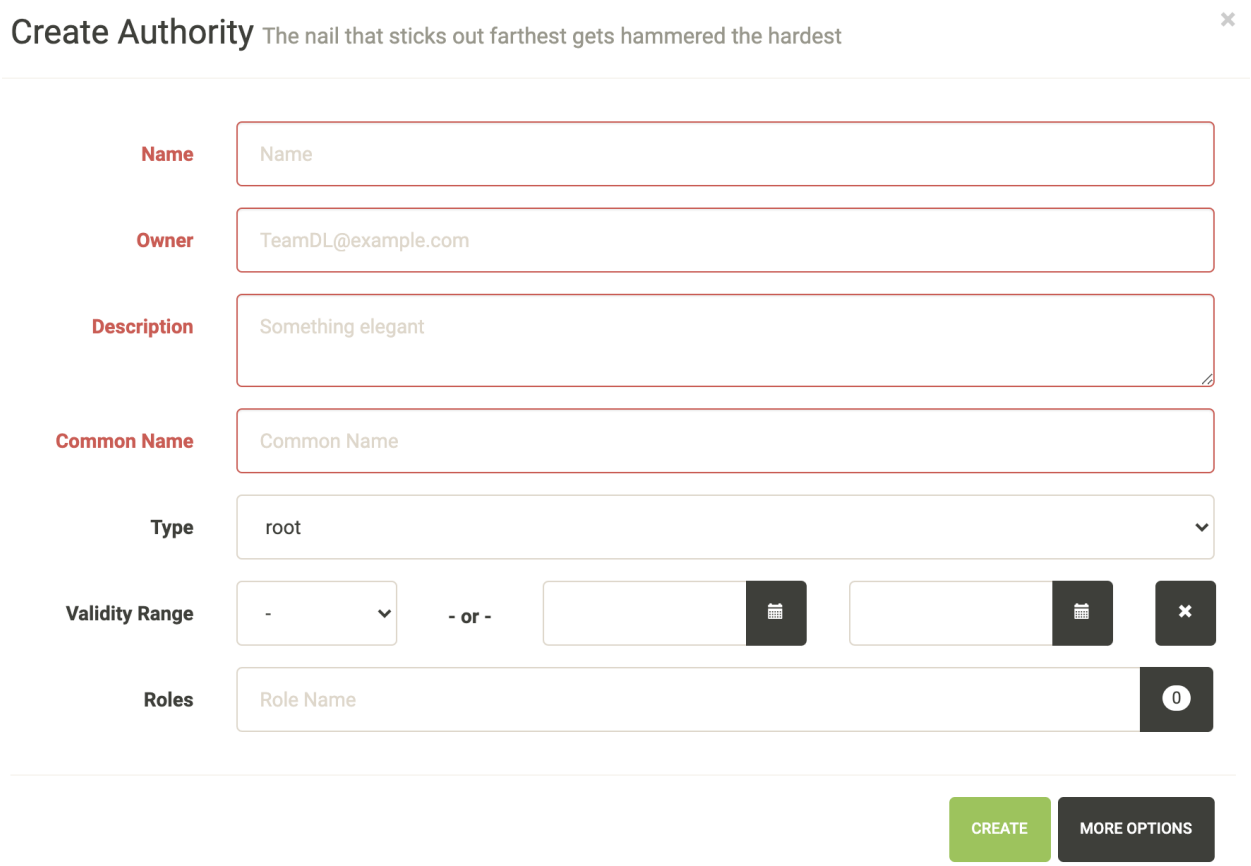


Fig. 2: Enter an authority name and short description about the authority. Enter an owner, and certificate common name. Depending on the authority and the authority/issuer plugin these values may or may not be used.

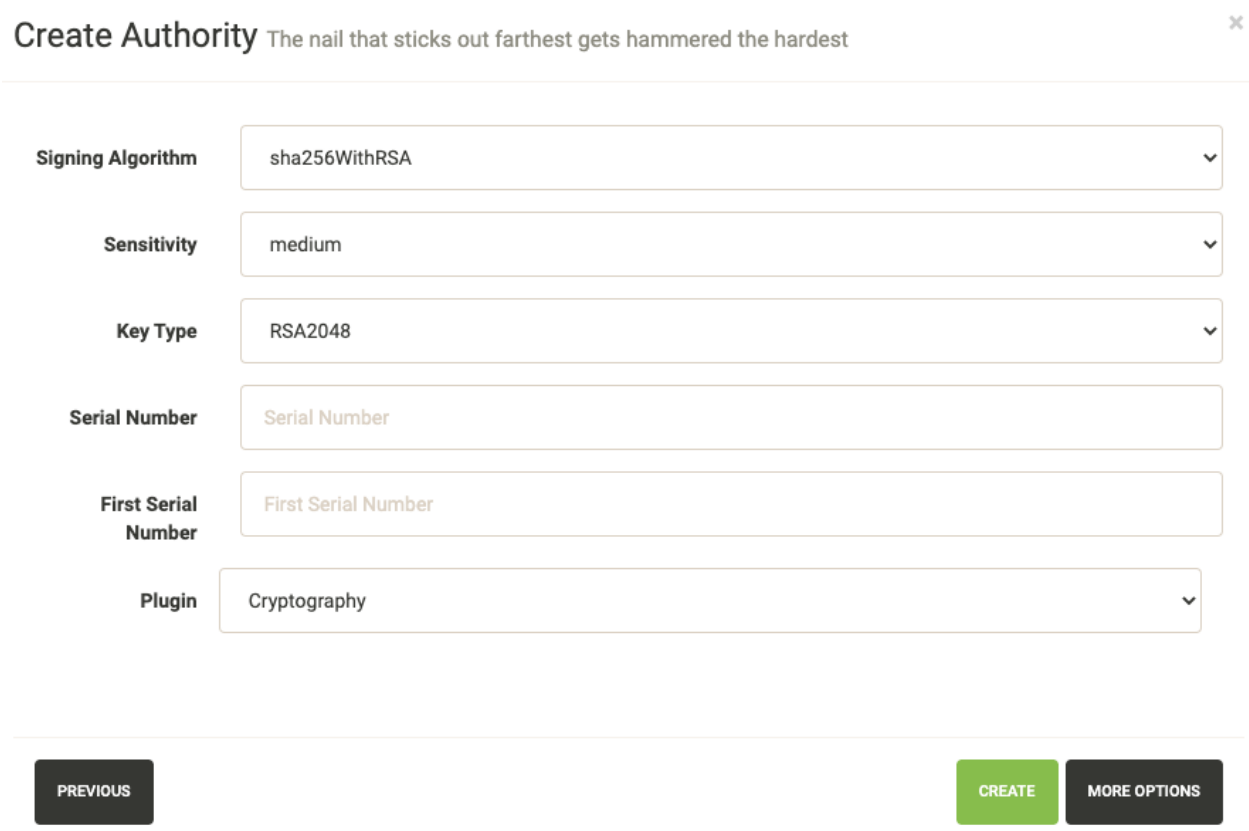


Fig. 3: Again how many of these values get used largely depends on the underlying plugin. It is important to make sure you select the right plugin that you wish to use.

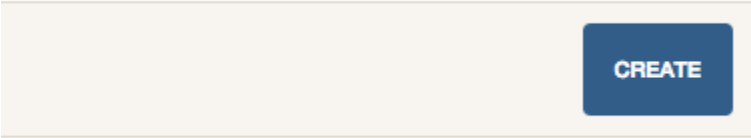


Fig. 4: In the certificate table select "Create"

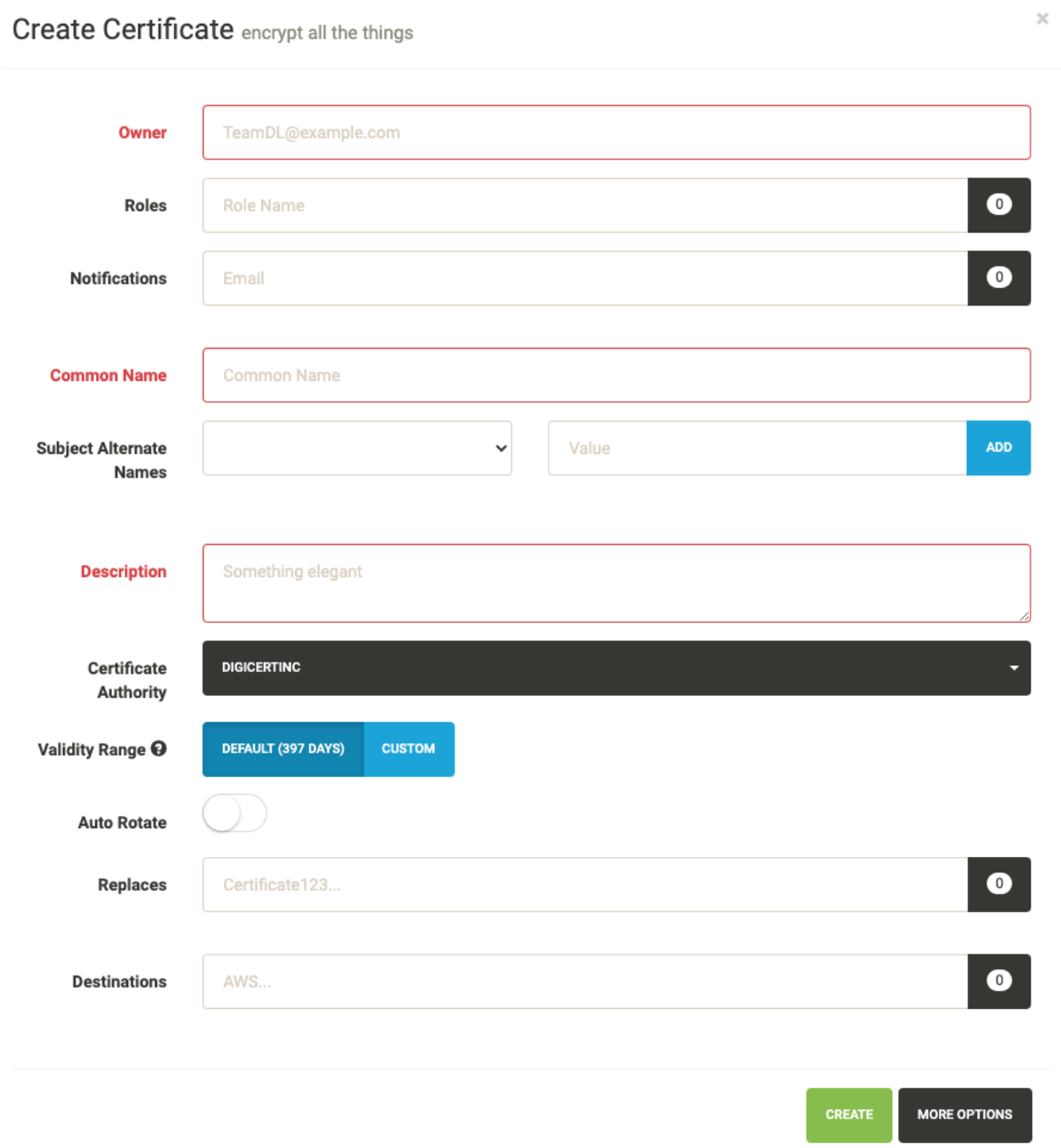


Fig. 5: Enter an owner, common name, short description and certificate authority you wish to issue this certificate. Depending upon the selected CA, the UI displays default validity of the certificate. You can select different validity by entering a custom date, if supported by the CA.

You can also add *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.

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.

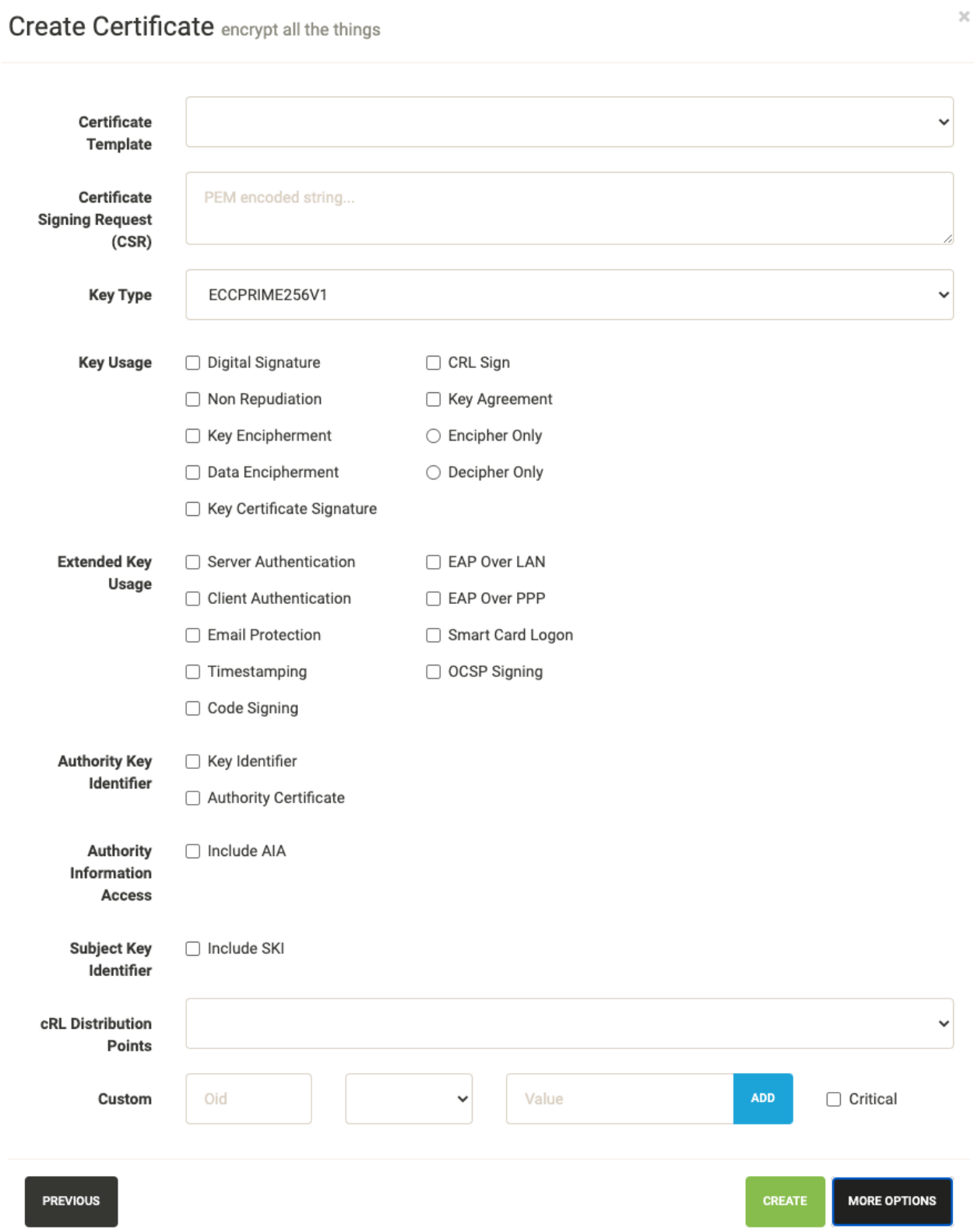


Fig. 6: These options are typically for advanced users. Lemur creates ECC based certificate (ECCPRIME256V1 in particular) by default. One can change the key type using the dropdown option listed here.

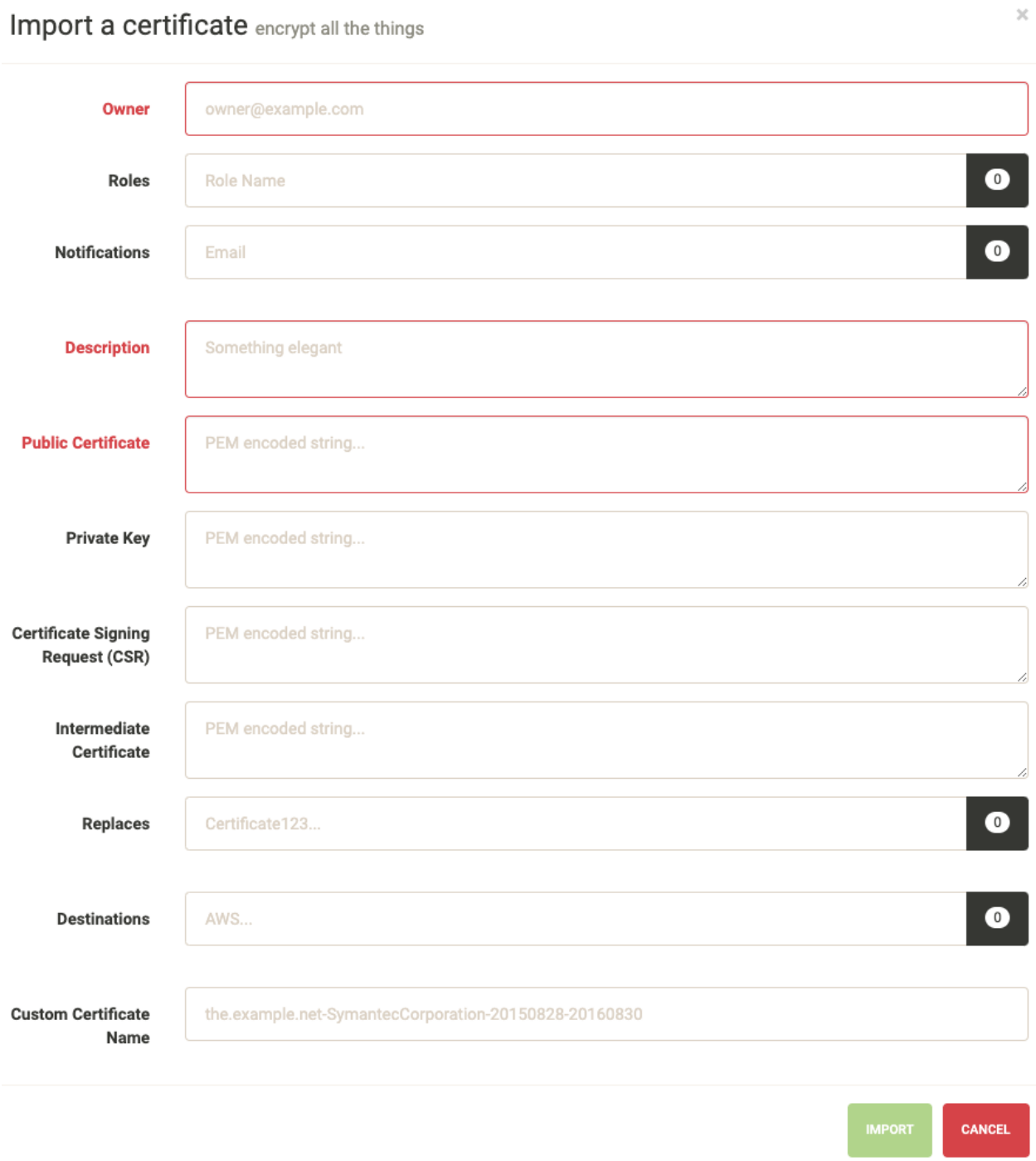


Fig. 7: Enter an owner, short description and public certificate. If there are intermediates and private keys Lemur will track them just as it does if the certificate were created through Lemur. Lemur generates a certificate name but you can override that by passing a value to the *Custom Certificate 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.

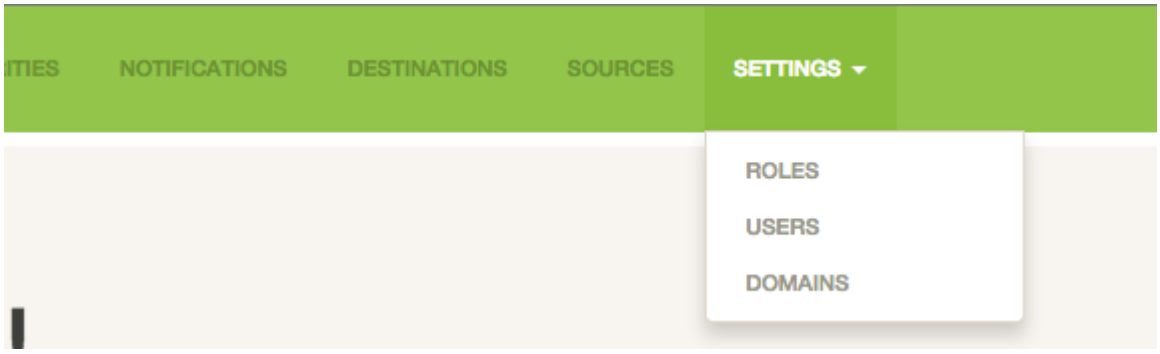


Fig. 8: From the settings dropdown select "Users"

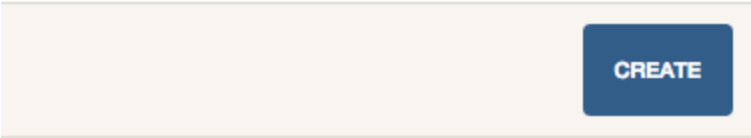


Fig. 9: In the user table select "Create"

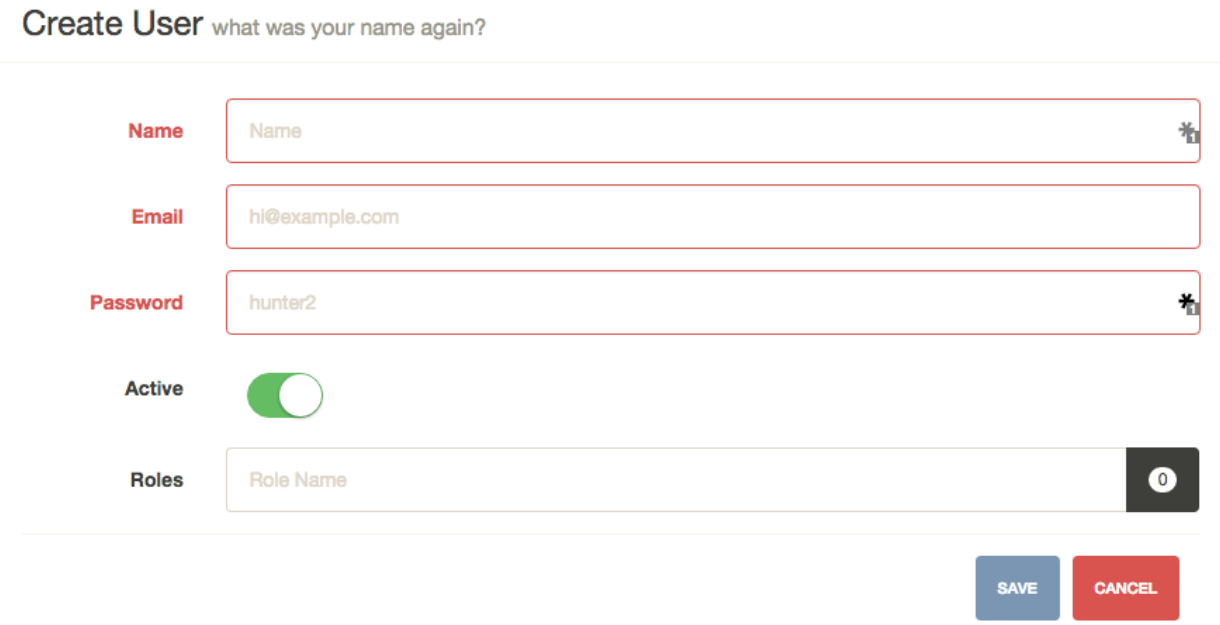


Fig. 10: Enter the username, email and password for the user. You can also assign any roles that the user will need when they login. While there is no deletion (we want to track creators forever) you can mark a user as 'Inactive' that will not allow them to login to Lemur.

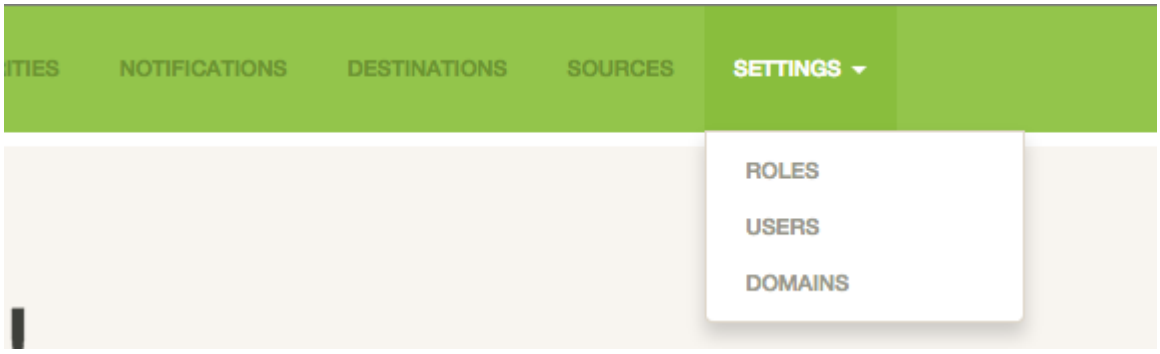


Fig. 11: From the settings dropdown select "Roles"

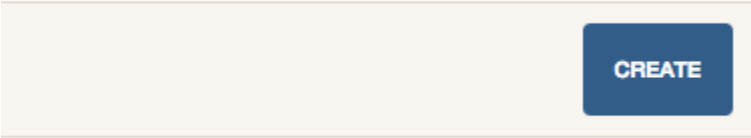


Fig. 12: In the role table select "Create"

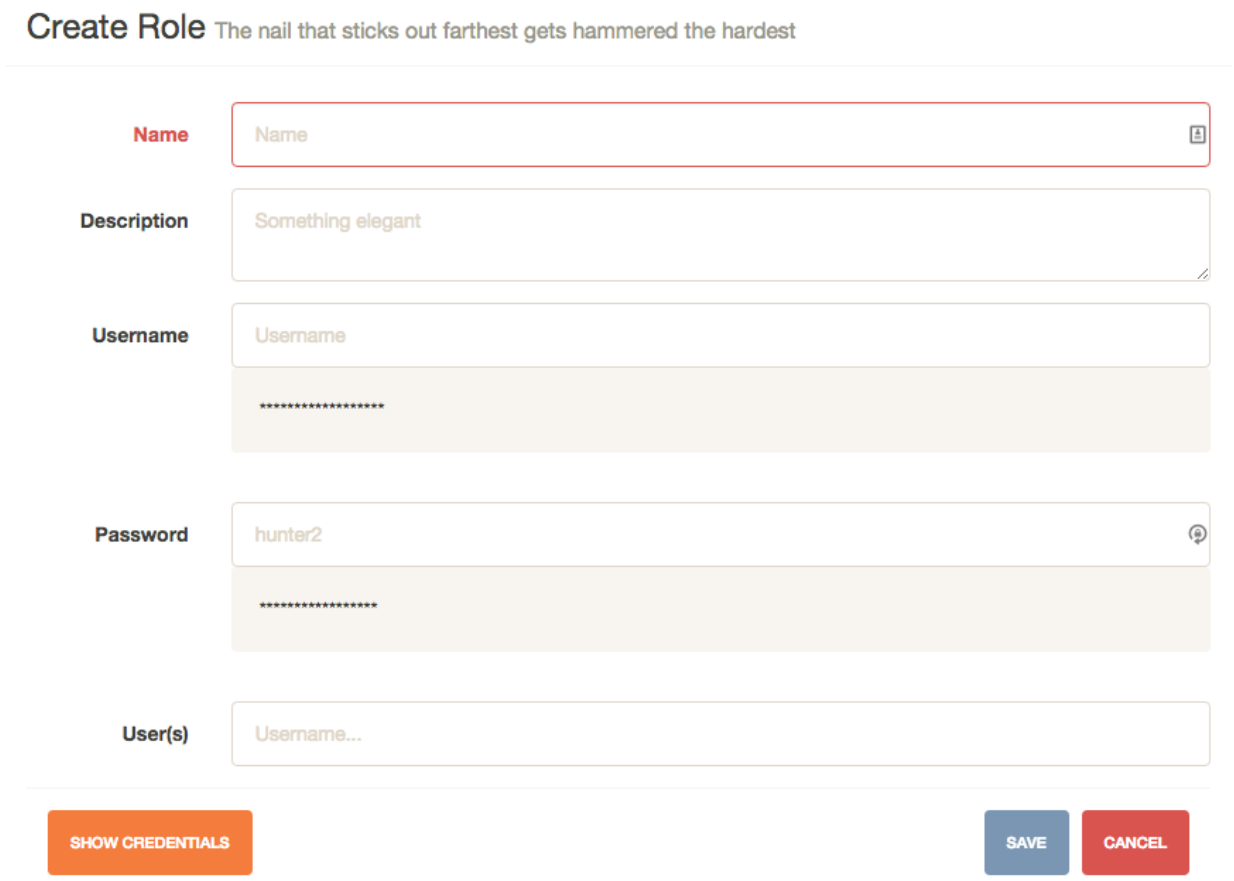


Fig. 13: Enter a role name and short description about the role. You can optionally store a user/password on the role. This is useful if your authority require specific roles. You can then accurately map those roles onto Lemur users. Also optional you can assign users to your new role.

CHAPTER

THREE

ADMINISTRATION

3.1 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.

3.1.1 Basic Configuration

LOG_LEVEL

```
LOG_LEVEL = "DEBUG"
```

LOG_FILE

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

LOG_UPGRADE_FILE

```
LOG_UPGRADE_FILE = "/logs/lemur/db_upgrade.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'
```

SQLALCHEMY_ENGINE_OPTIONS

This is an optional config that handles all engine_options to SQLAlchemy. Please refer to the flask-sqlalchemy website for more details about the individual configs.

The default connection pool size is 5 for sqlalchemy managed connections. Depending on the number of Lemur instances, please specify the per instance connection *pool_size*. Below is an example to set connection *pool_size* to 10.

max_overflow allows to create connections in addition to specified number of connections in pool size. By default, sqlalchemy allows 10 connections to create in addition to the pool size. If *pool_size* and *max_overflow* are not specified then each Lemur instance may create maximum of 15 connections.

pool_recycle defines number of seconds after which a connection is automatically recycled.

```
SQLALCHEMY_ENGINE_OPTIONS = {
    'pool_size': 10,
    'pool_recycle': 600,
    'pool_timeout': 20,
    'max_overflow': 10,
}
```

Warning: Specifying *pool_size* is an optional setting but important to review and set for optimal database connection usage and for overall database performance. Note that *SQLALCHEMY_POOL_SIZE*, *SQLALCHEMY_MAX_OVERFLOW*, *SQLALCHEMY_POOL_TIMEOUT* are deprecated since sqlalchemy v2.4.

Note: Specifying *max_overflow* to 0 will enforce limit to not create connections above specified pool size.

LEMUR_ALLOW_WEEKEND_EXPIRATION

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

LEMUR_ALLOWED_DOMAINS

List of regular expressions for domain restrictions; if the list is not empty, normal users can only issue certificates for domain names matching at least one pattern on this list. Administrators are exempt from this restriction.

Cerificate common name is matched against these rules *if* it does not contain a space. SubjectAltName DNS names are always matched against these rules.

Take care to write patterns in such way to not allow the * wildcard character inadvertently. To match a . character, it must be escaped (as .).

LEMUR_OWNER_EMAIL_IN_SUBJECT

By default, Lemur will add the certificate owner's email address to certificate subject (for CAs that allow it). Set this to *False* to disable this.

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

Only fields of type `Vault` will be encrypted. At present, only the following fields are encrypted:

- `certificates.private_key`
- `pending_certificates.private_key`
- `dns_providers.credentials`
- `roles.password`

For implementation details, see `Vault` in `utils.py`.

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=']
```

PUBLIC_CA_MAX_VALIDITY_DAYS

Use this config to override the limit of 397 days of validity for certificates issued by CA/Browser compliant authorities. The authorities with cab_compliant option set to true will use this config. The example below overrides the default validity of 397 days and sets it to 365 days.

```
PUBLIC_CA_MAX_VALIDITY_DAYS = 365
```

DEFAULT_VALIDITY_DAYS

Use this config to override the default validity of 365 days for certificates offered through Lemur UI. Any CA which is not CA/Browser Forum compliant will be using this value as default validity to be displayed on UI. Please note that this config is used for cert issuance only through Lemur UI. The example below overrides the default validity of 365 days and sets it to 1095 days (3 years).

```
DEFAULT_VALIDITY_DAYS = 1095
```

DEBUG_DUMP

Dump all imported or generated CSR and certificate details to stdout using OpenSSL. (default: *False*)

ALLOW_CERT_DELETION

When set to True, certificates can be marked as deleted via the API and deleted certificates will not be displayed in the UI. When set to False (the default), the certificate delete API will always return "405 method not allowed" and deleted certificates will always be visible in the UI. (default: *False*)

LEMUR_AWS_REGION

This is an optional config applicable for settings where Lemur is deployed in AWS. When specified, this will override the default regional AWS endpoints that are used for accessing STS and services such as IAM for example. You must set this if running in an alternative AWS partition such as GovCloud, for example.

LEMUR_AWS_PARTITION

Specifies the AWS partition that Lemur should use. Valid values are 'aws', 'aws-us-gov', and 'aws-cn'. Defaults to 'aws'. If Lemur is deployed in and managing endpoints AWS GovCloud, for example, you must set this to *aws-us-gov*.

SENTRY_DSN

To initialize the Sentry integration to capture errors and exceptions, the *SENTRY_DSN* is required to be set to the respective URL. *LEMUR_ENV* is also a related variable to define the environment for sentry events, e.g., 'test' or 'prod'.

Note that previously Lemur relied on Raven[flask] before migrating to *sentry_sdk*. In this case, you might be using the legacy *SENTRY_CONFIG*, which Lemur attempts to respect, in case *SENTRY_DSN* is missing, with environment set to empty.

Example for using Senty to capture exceptions:

```
>>> from sentry_sdk import capture_exception
>>> ..
>>> capture_exception()
>>> # supplying extra information
>>> capture_exception(extra={"certificate_name": str(certificate.name)})
```

3.1.2 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 = ""
```

LEMUR_DEFAULT_ISSUER_PLUGIN

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

LEMUR_DEFAULT_AUTHORITY

```
LEMUR_DEFAULT_AUTHORITY = "verisign"
```

3.1.3 Notification Options

Lemur supports a small variety of notification types through a set of notification plugins. By default, Lemur configures a standard set of email notifications for all certificates.

Plugin-capable notifications

These notifications can be configured to use all available notification plugins.

Supported types:

- Certificate expiration (Celery: *notify_expirations*, cron: *notify expirations*)

Email-only notifications

These notifications can only be sent via email and cannot use other notification plugins.

Supported types:

- CA certificate expiration (Celery: *notify_authority_expirations*, cron: *notify authority_expirations*)
- Pending ACME certificate failure
- Certificate rotation

- Certificate reissued with no endpoints
- Certificate reissue failed
- Certificate revocation
- Security certificate expiration summary (Celery: *send_security_expiration_summary*, cron: *notify security_expiration_summary*)
- Certificate expiration where certificates are still detected as deployed at any associated domain (Celery: *notify_expiring_deployed_certificates*, cron: *notify expiring_deployed_certificates*)

Default notifications

When a certificate is created, the following email notifications are created for it if they do not exist. If these notifications already exist, they will be associated with the new certificate.

- `DEFAULT_<OWNER>_X_DAY`, where X is the set of values specified in `LEMUR_DEFAULT_EXPIRATION_NOTIFICATION_INTERVALS` and defaults to 30, 15, and 2 if not specified. The owner's username will replace `<OWNER>`.
- `DEFAULT_SECURITY_X_DAY`, where X is the set of values specified in `LEMUR_SECURITY_TEAM_EMAIL_INTERVALS` and defaults to `LEMUR_DEFAULT_EXPIRATION_NOTIFICATION_INTERVALS` if not specified (which also defaults to 30, 15, and 2 if not specified).

These notifications can be disabled if desired. They can also be unassociated with a specific certificate.

Disabling notifications

Notifications can be disabled either for an individual certificate (which disables all notifications for that certificate) or for an individual notification object (which disables that notification for all associated certificates). At present, disabling a notification object will only disable certificate expiration notifications, and not other types, since other notification types don't use notification objects.

Certificate expiration

Certificate expiration notifications are sent when the scheduled task to send certificate expiration notifications runs (see *Periodic Tasks*). Specific patterns of certificate names may be excluded using `--exclude` (when using cron; you may specify this multiple times for multiple patterns) or via the config option `EXCLUDE_CN_FROM_NOTIFICATION` (when using celery; this is a list configuration option, meaning you specify multiple values, such as `['exclude', 'also exclude']`). The specified exclude pattern will match if found anywhere in the certificate name.

When the periodic task runs, Lemur checks for certificates meeting the following conditions:

- Certificate has notifications enabled
- Certificate is not expired
- Certificate is not revoked
- Certificate name does not match the *exclude* parameter
- Certificate has at least one associated notification object
- That notification is active
- That notification's configured interval and unit match the certificate's remaining lifespan

All eligible certificates are then grouped by owner and applicable notification. For each notification and certificate group, Lemur will send the expiration notification using whichever plugin was configured for that notification object. In addition, Lemur will send an email to the certificate owner and security team (as specified by the `LEMUR_SECURITY_TEAM_EMAIL` configuration parameter). The security team will be omitted if `LEMUR_DISABLE_SECURITY_TEAM_EXPIRATION_EMAILS` is enabled.

CA certificate expiration

Certificate authority certificate expiration notifications are sent when the scheduled task to send authority certificate expiration notifications runs (see *Periodic Tasks*). Notifications are sent via the intervals configured in the configuration parameter `LEMUR_AUTHORITY_CERT_EXPIRATION_EMAIL_INTERVALS`, with a default of 365 and 180 days.

When the periodic task runs, Lemur checks for certificates meeting the following conditions:

- Certificate has notifications enabled
- Certificate is not expired
- Certificate is not revoked
- Certificate is associated with a CA
- Certificate's remaining lifespan matches one of the configured intervals

All eligible certificates are then grouped by owner and expiration interval. For each interval and certificate group, Lemur will send the CA certificate expiration notification via email to the certificate owner and security team (as specified by the `LEMUR_SECURITY_TEAM_EMAIL` configuration parameter).

Pending ACME certificate failure

Whenever a pending ACME certificate fails to be issued, Lemur will send a notification via email to the certificate owner and security team (as specified by the `LEMUR_SECURITY_TEAM_EMAIL` configuration parameter). This email is not sent if the pending certificate had notifications disabled.

Lemur will attempt 3x times to resolve a pending certificate. This can at times result into 3 duplicate certificates, if all certificate attempts get resolved. There is a way to deduplicate these certificates periodically using a celery task `disable_rotation_of_duplicate_certificates`.

This needs 2 configurations

`AUTHORITY_TO_DISABLE_ROTATE_OF_DUPLICATE_CERTIFICATES`

List names of the authorities for which *disable_rotation_of_duplicate_certificates* should run. The task will consider certificates issued by authorities configured here.

```
AUTHORITY_TO_DISABLE_ROTATE_OF_DUPLICATE_CERTIFICATES = ["LetsEncrypt"]
```

`DAYS_SINCE_ISSUANCE_DISABLE_ROTATE_OF_DUPLICATE_CERTIFICATES`

Use this config (optional) to configure the number of days. The task *disable_rotation_of_duplicate_certificates* will then consider valid certificates issued only in last those many number of days for deduplication. If not configured, the task considers all the valid certificates. Ideally set this config to a value which is same as the number of days between the two runs of *disable_rotation_of_duplicate_certificates*

```
DAYS_SINCE_ISSUANCE_DISABLE_ROTATE_OF_DUPLICATE_CERTIFICATES = 7
```

Certificate re-issuance

When a cert is reissued (i.e. a new certificate is minted to replace it), *and* the re-issuance either fails or succeeds but the certificate has no associated endpoints (meaning the subsequent rotation step will not occur), Lemur will send a notification via email to the certificate owner. This notification is disabled by default; to enable it, you must set the option `--notify` (when using cron) or the configuration parameter `ENABLE_REISSUE_NOTIFICATION` (when using celery).

Certificate rotation

Whenever a cert is rotated, Lemur will send a notification via email to the certificate owner. This notification is disabled by default; to enable it, you must set the option `--notify` (when using cron) or the configuration parameter `ENABLE_ROTATION_NOTIFICATION` (when using celery).

Certificate revocation

Whenever a cert is revoked, Lemur will send a notification via email to the certificate owner. This notification will only be sent if the certificate's "notify" option is enabled.

Security certificate expiration summary

If you enable the Celery or cron task to send this notification type, Lemur will send a summary of all certificates with upcoming expiration date that occurs within the number of days specified by the `LEMUR_EXPIRATION_SUMMARY_EMAIL_THRESHOLD_DAYS` configuration parameter (with a fallback of 14 days). Note that certificates will be included in this summary even if they do not have any associated notifications.

This notification type also supports the same `--exclude` and `EXCLUDE_CN_FROM_NOTIFICATION` options as expiration emails.

NOTE: At present, this summary email essentially duplicates the certificate expiration notifications, since all certificate expiration notifications are also sent to the security team. This issue will be fixed in the future.

Notification configuration

The following configuration options are supported:

`EXCLUDE_CN_FROM_NOTIFICATION`

Specifies CNs to exclude from notifications. This includes both single notifications as well as the notification summary. The specified exclude pattern will match if found anywhere in the certificate name.

Note:

This is only used for celery. The equivalent for cron is '-e' or '–exclude'.

```
EXCLUDE_CN_FROM_NOTIFICATION = ['exclude', 'also exclude']
```

`DISABLE_NOTIFICATION_PLUGINS`

Specifies a set of notification plugins to disable. Notifications will not be sent using these plugins. Currently only applies to expiration notifications, since they are the only type that utilize plugins. This option may be particularly useful in a test environment, where you might wish to enable the notification job without actually sending notifications of a certain type (or all types).

Note:

This is only used for celery. The equivalent for cron is '-d' or '–disabled-notification-plugins'.

```
DISABLE_NOTIFICATION_PLUGINS = ['email-notification']
```

Email notifications

Templates for emails are located under *lemur/plugins/lemur_email/templates* and can be modified for your needs.

The following configuration options are supported:

`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_EMAIL* configuration will need to be verified by AWS before you can send any mail. See: Verifying Email Address in Amazon SES

LEMUR_SES_SOURCE_ARN

Specifies an ARN to use as the SourceArn when sending emails via SES.

Note: This parameter is only required if you're using a sending authorization with SES. See: Using sending authorization with Amazon SES

LEMUR_SES_REGION

Specifies a region for sending emails via SES.

Note: This parameter defaults to us-east-1 and is only required if you wish to use a different region.

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_DISABLE_SECURITY_TEAM_EXPIRATION_EMAILS

This specifies whether or not LEMUR_SECURITY_TEAM_EMAIL will be included on all expiration emails. IMPORTANT: You will also need to disable the DEFAULT_SECURITY_X_DAY notifications to truly disable sending expiration emails to the security team. This double configuration is required for backwards compatibility.

```
LEMUR_DISABLE_SECURITY_TEAM_EXPIRATION_EMAILS = True
```

LEMUR_DEFAULT_EXPIRATION_NOTIFICATION_INTERVALS

Lemur notification intervals. If unspecified, the value [30, 15, 2] is used.

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

LEMUR_SECURITY_TEAM_EMAIL_INTERVALS

Alternate notification interval set for security team notifications. Use this if you would like the default security team notification interval for new certificates to differ from the global default as specified in LEMUR_DEFAULT_EXPIRATION_NOTIFICATION_INTERVALS. If unspecified, the value of LEMUR_DEFAULT_EXPIRATION_NOTIFICATION_INTERVALS is used. Security team default notifications for new certificates can effectively be disabled by setting this value to an empty array.

```
LEMUR_SECURITY_TEAM_EMAIL_INTERVALS = [15, 2]
```

LEMUR_AUTHORITY_CERT_EXPIRATION_EMAIL_INTERVALS

Notification interval set for CA certificate expiration notifications. If unspecified, the value [365, 180] is used (roughly one year and 6 months).

```
LEMUR_AUTHORITY_CERT_EXPIRATION_EMAIL_INTERVALS = [365, 180]
```

LEMUR_PORTS_FOR_DEPLOYED_CERTIFICATE_CHECK

Specifies the set of ports to use when checking if a certificate is still deployed at a given domain. This is utilized for the alert that is sent when an expiring certificate is detected to still be deployed.

```
LEMUR_PORTS_FOR_DEPLOYED_CERTIFICATE_CHECK = [443]
```

LEMUR_DEPLOYED_CERTIFICATE_CHECK_COMMIT_MODE

Specifies whether or not to commit changes when running the deployed certificate check. If False, the DB will not be updated; network calls will still be made and logs/metrics will be emitted.

```
LEMUR_DEPLOYED_CERTIFICATE_CHECK_COMMIT_MODE = True
```

LEMUR_DEPLOYED_CERTIFICATE_CHECK_EXCLUDED_DOMAINS

Specifies a set of domains to exclude from the deployed certificate checks. Anything specified here is treated as a substring; in other words, if you set this to ['excluded.com'], then 'abc.excluded.com' and 'unexcluded.com' will both be excluded; 'ex-cluded.com' will not be excluded.

```
LEMUR_DEPLOYED_CERTIFICATE_CHECK_EXCLUDED_DOMAINS = ['excluded.com']
```

LEMUR_DEPLOYED_CERTIFICATE_CHECK_EXCLUDED_OWNERS

Specifies a set of owners to exclude from the deployed certificate checks. Anything specified here is treated as an exact match, NOT as a substring.

```
LEMUR_DEPLOYED_CERTIFICATE_CHECK_EXCLUDED_OWNERS = ['excludedowner@example.
↪com']
```

LEMUR_REISSUE_NOTIFICATION_EXCLUDED_DESTINATIONS

Specifies a set of destination labels to exclude from the reissued with endpoint notification checks. If a certificate is reissued without endpoints, but any of its destination labels are specified in this list, no "reissued without endpoints" notification will be sent.

```
LEMUR_REISSUE_NOTIFICATION_EXCLUDED_DESTINATIONS = ['excluded-destination']
```

3.1.4 Celery Options

To make use of automated tasks within lemur (e.g. syncing source/destinations, or reissuing ACME certificates), you need to configure celery. See *Periodic Tasks* for more in depth documentation.

CELERY_RESULT_BACKEND

The url to your redis backend (needs to be in the format *redis://<host>:<port>/<database>*)

CELERY_BROKER_URL

The url to your redis broker (needs to be in the format *redis://<host>:<port>/<database>*)

CELERY_IMPORTS

The module that celery needs to import, in our case thats *lemur.common.celery*

CELERY_TIMEZONE

The timezone for celery to work with

CELERYBEAT_SCHEDULE

This defines the schedule, with which the celery beat makes the worker run the specified tasks.

CELERY_ENDPOINTS_EXPIRE_TIME_IN_HOURS

This is an optional parameter that defines the expiration time for endpoints when the endpoint expiration celery task is running. Default value is set to 2h.

Since the celery module, relies on the RedisHandler, the following options also need to be set.

REDIS_HOST

Hostname of your redis instance

REDIS_PORT

Port on which redis is running (default: 6379)

REDIS_DB

Which redis database to be used, by default redis offers databases 0-15 (default: 0)

3.1.5 Authentication Options

Lemur currently supports Basic Authentication, LDAP Authentication, Ping OAuth2, and Google out of the box. Additional flows can be added relatively easily.

LDAP Options

Lemur supports the use of an LDAP server in conjunction with Basic Authentication. Lemur local users can still be defined and take precedence over LDAP users. If a local user does not exist, LDAP will be queried for authentication. Only simple ldap binding with or without TLS is supported.

LDAP support requires the pyldap python library, which also depends on the following openldap packages.

```
$ sudo apt-get update
$ sudo apt-get install libldap2-dev libsasl2-dev libldap2-dev libssl-dev
```

To configure the use of an LDAP server, a number of settings need to be configured in *lemur.conf.py*.

Here is an example LDAP configuration stanza you can add to your config. Adjust to suit your environment of course.

```
LDAP_AUTH = True
LDAP_BIND_URI='ldaps://secure.evilcorp.net'
LDAP_BASE_DN='DC=users,DC=evilcorp,DC=net'
LDAP_EMAIL_DOMAIN='evilcorp.net'
```

(continues on next page)

(continued from previous page)

```
LDAP_USE_TLS = True
LDAP_CACERT_FILE = '/opt/lemur/trusted.pem'
LDAP_REQUIRED_GROUP = 'certificate-management-access'
LDAP_GROUPS_TO_ROLES = {'certificate-management-admin': 'admin', 'certificate-management-
↪read-only': 'read-only'}
LDAP_IS_ACTIVE_DIRECTORY = True
```

The lemur ldap module uses the *user principal name* (upn) of the authenticating user to bind. This is done once for each user at login time. The UPN is effectively the email address in AD/LDAP of the user. If the user doesn't provide the email address, it constructs one based on the username supplied (which should normally match the samAccountName) and the value provided by the config LDAP_EMAIL_DOMAIN. The config LDAP_BASE_DN tells lemur where to search within the AD/LDAP tree for the given UPN (user). If the bind with those credentials is successful - there is a valid user in AD with correct password.

Each of the LDAP options are described below.

LDAP_AUTH

This enables the use of LDAP

```
LDAP_AUTH = True
```

LDAP_BIND_URI

Specifies the LDAP server connection string

```
LDAP_BIND_URI = 'ldaps://hostname'
```

LDAP_BASE_DN

Specifies the LDAP distinguished name location to search for users

```
LDAP_BASE_DN = 'DC=Users,DC=Evilcorp,DC=com'
```

LDAP_EMAIL_DOMAIN

The email domain used by users in your directory. This is used to build the userPrincipalName to search with.

```
LDAP_EMAIL_DOMAIN = 'evilcorp.com'
```

The following LDAP options are not required, however TLS is always recommended.

LDAP_USE_TLS

Enables the use of TLS when connecting to the LDAP server. Ensure the LDAP_BIND_URI is using ldaps scheme.

```
LDAP_USE_TLS = True
```

LDAP_CACERT_FILE

Specify a Certificate Authority file containing PEM encoded trusted issuer certificates. This can be used if your LDAP server is using certificates issued by a private CA.

```
LDAP_CACERT_FILE = '/path/to/cacert/file'
```

LDAP_REQUIRED_GROUP

Lemur has pretty open permissions. You can define an LDAP group to specify who can access Lemur. Only members of this group will be able to login.

```
LDAP_REQUIRED_GROUP = 'Lemur LDAP Group Name'
```

LDAP_GROUPS_TO_ROLES

You can also define a dictionary of ldap groups mapped to lemur roles. This allows you to use ldap groups to manage access to owner/creator roles in Lemur

```
LDAP_GROUPS_TO_ROLES = {'lemur_admins': 'admin', 'Lemur Team DL Group':
↪'team@example.com'}
```

LDAP_IS_ACTIVE_DIRECTORY

When set to True, nested group memberships are supported, by searching for groups with the member:1.2.840.113556.1.4.1941 attribute set to the user DN. When set to False, the list of groups will be determined by the 'memberof' attribute of the LDAP user logging in.

```
LDAP_IS_ACTIVE_DIRECTORY = False
```

Authentication Providers

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_URL

```
PING_URL = "https://<yourlemurserver>"
```

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_URL

```
OAUTH2_URL = "https://<yourlemurserver>"
```

OAUTH2_REDIRECT_URI

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

OAUTH2_AUTH_ENDPOINT

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

OAUTH2_VERIFY_CERT

```
OAUTH2_VERIFY_CERT = True
```

OAUTH_STATE_TOKEN_SECRET

The OAUTH_STATE_TOKEN_SECRET is used to sign state tokens to guard against CSRF attacks. Without a secret configured, Lemur will create a fallback secret on a per-server basis that would last for the length of the server's lifetime (e.g., between re-deploys). The secret must be *bytes-like <https://cryptography.io/en/latest/glossary/#term-bytes-like>*; it will be used to instantiate the key parameter of *HMAC <https://cryptography.io/en/latest/hazmat/primitives/mac/hmac/#cryptography.hazmat.primitives.hmac.HMAC>*.

For implementation details, see `generate_state_token()` and `verify_state_token()` in `lemur/auth/views.py`.

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
>>> KEY_LENGTH = 32  # tweak as needed
>>> base64.b64encode(os.urandom(KEY_LENGTH))
```

```
OAUTH_STATE_TOKEN_SECRET = lemur.common.utils.get_state_token_secret()
```

OAUTH_STATE_TOKEN_STALE_TOLERANCE_SECONDS

Defaults to 15 seconds if configuration is not discovered.

```
OAUTH_STATE_TOKEN_STALE_TOLERANCE_SECONDS = 15
```

GOOGLE_CLIENT_ID

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

GOOGLE_SECRET

```
GOOGLE_SECRET = "somethingsecret"
```

TOKEN_AUTH_HEADER_CASE_SENSITIVE

This is an optional parameter to change the case sensitivity of the access token request authorization header. This is required if the oauth provider has implemented the access token request authorization header in a case-sensitive way

```
TOKEN_AUTH_HEADER_CASE_SENSITIVE = True
```

USER_MEMBERSHIP_PROVIDER

An optional plugin to provide membership details. Provide plugin slug here. Plugin is used post user validation to update membership details in Lemur. Also, it is configured to provide APIs to validate user email, team email/DL.

```
USER_MEMBERSHIP_PROVIDER = "<yourmembershippluginslug>"
```

Authorization Providers

If you are not using a custom authorization provider you do not need to configure any of these options

USER_DOMAIN_AUTHORIZATION_PROVIDER

An optional plugin to perform domain level authorization during certificate issuance. Provide plugin slug here. Plugin is used to check if caller is authorized to issue a certificate for a given Common Name and Subject Alternative Name (SAN) of type DNSName. Plugin shall be an implementation of DomainAuthorizationPlugin.

```
USER_DOMAIN_AUTHORIZATION_PROVIDER = "<yourauthorizationpluginslug>"
```

LEMUR_PRIVATE_AUTHORITY_PLUGIN_NAMES

Lemur can be used to issue certificates with private CA. One can write own issuer plugin to do so. Domain level authorization is skipped for private CA i.e., the one implementing custom issuer plugin. Currently this config is not used elsewhere.

```
LEMUR_PRIVATE_AUTHORITY_PLUGIN_NAMES = ["issuerpluginslug1",
↪"issuerpluginslug2"]
```

Metric Providers

If you are not using a metric provider you do not need to configure any of these options.

ACTIVE_PROVIDERS

A list of metric plugins slugs to be ativated.

```
METRIC_PROVIDERS = ['atlas-metric']
```

3.1.6 Plugin Specific Options

ACME Plugin

ACME_DNS_PROVIDER_TYPES

Dictionary of ACME DNS Providers and their requirements.

ACME_ENABLE_DELEGATED_CNAME

Enables delegated DNS domain validation using CNAMES. When enabled, Lemur will attempt to follow CNAME records to authoritative DNS servers when creating DNS-01 challenges.

The following configration properties are optional for the ACME plugin to use. They allow reusing an existing ACME account. See *Using a pre-existing ACME account* for more details.

ACME_PRIVATE_KEY

This is the private key, the account was registered with (in JWK format)

ACME_REGR

This is the registration for the ACME account, the most important part is the uri attribute (in JSON)

ACME_PREFERRED_ISSUER

This is an optional parameter to indicate the preferred chain to retrieve from ACME when finalizing the order. This is applicable to Let's Encrypts recent migration to their own root, where they provide two distinct certificate chains (fullchain_pem vs. alternative_fullchains_pem); the main chain will be the long chain that is rooted in the expiring DTS root, whereas the alternative chain is rooted in X1 root CA. Select "X1" to get the shorter chain (currently alternative), leave blank or "DST Root CA X3" for the longer chain.

Active Directory Certificate Services Plugin

ADCS_SERVER

FQDN of your ADCS Server

ADCS_AUTH_METHOD

The chosen authentication method. Either 'basic' (the default), 'ntlm' or 'cert' (SSL client certificate). The next 2 variables are interpreted differently for different methods.

ADCS_USER

The username (basic) or the path to the public cert (cert) of the user accessing PKI

ADCS_PWD

The passwd (basic) or the path to the private key (cert) of the user accessing PKI

ADCS_TEMPLATE

Template to be used for certificate issuing. Usually display name w/o spaces

ADCS_TEMPLATE_<upper(authority.name)>

If there is a config variable ADCS_TEMPLATE_<upper(authority.name)> take the value as Cert template else default to ADCS_TEMPLATE to be compatible with former versions. Template to be used for certificate issuing. Usually display name w/o spaces

ADCS_START

Used in ADCS-Sourceplugin. Minimum id of the first certificate to be returned. ID is increased by one until ADCS_STOP. Missing cert-IDs are ignored

ADCS_STOP

Used for ADCS-Sourceplugin. Maximum id of the certificates returned.

ADCS_ISSUING

Contains the issuing cert of the CA

ADCS_ROOT

Contains the root cert of the CA

Entrust Plugin

Enables the creation of Entrust certificates. You need to set the API access up with Entrust support. Check the information in the Entrust Portal as well. Certificates are created as "SERVER_AND_CLIENT_AUTH". Caution: Sometimes the entrust API does not respond in a timely manner. This error is handled and reported by the plugin. Should this happen you just have to hit the create button again after to create a valid certificate. The following parameters have to be set in the configuration files.

ENTRUST_URL

This is the url for the Entrust API. Refer to the API documentation.

ENTRUST_API_CERT

Path to the certificate file in PEM format. This certificate is created in the onboarding process. Refer to the API documentation.

ENTRUST_API_KEY

Path to the key file in RSA format. This certificate is created in the onboarding process. Refer to the API documentation. Caution: the request library cannot handle encrypted keys. The keyfile therefore has to contain the unencrypted key. Please put this in a secure location on the server.

ENTRUST_API_USER

String with the API user. This user is created in the onboarding process. Refer to the API documentation.

ENTRUST_API_PASS

String with the password for the API user. This password is created in the onboarding process. Refer to the API documentation.

ENTRUST_NAME

String with the name that should appear as certificate owner in the Entrust portal. Refer to the API documentation.

ENTRUST_EMAIL

String with the email address that should appear as certificate contact email in the Entrust portal. Refer to the API documentation.

ENTRUST_PHONE

String with the phone number that should appear as certificate contact in the Entrust portal. Refer to the API documentation.

ENTRUST_ISSUING

Contains the issuing cert of the CA

ENTRUST_ROOT

Contains the root cert of the CA

ENTRUST_PRODUCT_<upper(authority.name)>

If there is a config variable ENTRUST_PRODUCT_<upper(authority.name)> take the value as cert product name else default to "STANDARD_SSL". Refer to the API documentation for valid products names.

ENTRUST_CROSS_SIGNED_RSA_L1K

This is optional. Entrust provides support for cross-signed subCAS. One can set ENTRUST_CROSS_SIGNED_RSA_L1K to the respective cross-signed RSA-based subCA PEM and Lemur will replace the retrieved subCA with ENTRUST_CROSS_SIGNED_RSA_L1K.

ENTRUST_CROSS_SIGNED_ECC_L1F

This is optional. Entrust provides support for cross-signed subCAS. One can set ENTRUST_CROSS_SIGNED_ECC_L1F to the respective cross-signed EC-based subCA PEM and Lemur will replace the retrieved subCA with ENTRUST_CROSS_SIGNED_ECC_L1F.

ENTRUST_USE_DEFAULT_CLIENT_ID

If set to True, Entrust will use the primary client ID of 1, which applies to most use-case. Otherwise, Entrust will first lookup the clientId before ordering the certificate.

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_ORDER_TYPE

This is the type of certificate to order. (e.g. ssl_plus, ssl_ev_plus see: https://www.digicert.com/services/v2/documentation/order/overview-submit)

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_DAYS

This is the default validity (in days), if no end date is specified. (Default: 397)

DIGICERT_MAX_VALIDITY_DAYS

This is the maximum validity (in days). (Default: value of DIGICERT_DEFAULT_VALIDITY_DAYS)

DIGICERT_PRIVATE

This is whether or not to issue a private certificate. (Default: False)

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

CFSSL_KEY

This is the hmac key to authenticate to the CFSSL service. (Optional)

Hashicorp Vault Source/Destination Plugin

Lemur can import and export certificate data to and from a Hashicorp Vault secrets store. Lemur can connect to a different Vault service per source/destination.

Note: This plugin does not supersede or overlap the 3rd party Vault Issuer plugin.

Note: Vault does not have any configuration properties however it does read from a file on disk for a vault access token. The Lemur service account needs read access to this file.

Vault Source

The Vault Source Plugin will read from one Vault object location per source defined. There is expected to be one or more certificates defined in each object in Vault.

Vault Destination

A Vault destination can be one object in Vault or a directory where all certificates will be stored as their own object by CN.

Vault Destination supports a regex filter to prevent certificates with SAN that do not match the regex filter from being deployed. This is an optional feature per destination defined.

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.

Setting up IAM roles

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": [
                          "cloudfront:GetDistribution",
                          "cloudfront:GetDistributionConfig",
                          "cloudfront:ListDistributions",
                          "cloudfront:UpdateDistribution",
                          "elasticloadbalancing:DescribeInstanceHealth",
                          "elasticloadbalancing:DescribeLoadBalancerAttributes",
                          "elasticloadbalancing:DescribeLoadBalancerPolicyTypes",
                          "elasticloadbalancing:DescribeLoadBalancerPolicies",
                          "elasticloadbalancing:DescribeLoadBalancers",
                          "elasticloadbalancing:DeleteLoadBalancerListeners",
                          "elasticloadbalancing:CreateLoadBalancerListeners"
                     ],
                     "Resource": [
                          "*"
                     ],
                     "Effect": "Allow",
                     "Sid": "Stmt1404841912000"
                }
           ]
}
```

Setting up STS access

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"
    }
  ]
}
```

Adding N+1 accounts

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"
    }
  ]
}
```

Setting up SES

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_EMAIL = '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.

NS1 ACME Plugin

The NS1 ACME plugin allows DNS1 validation using NS1 domain services.

NS1_KEY

The NS1 read/write API key for managing TXT records for domain validation

PowerDNS ACME Plugin

The following configuration properties are required to use the PowerDNS ACME Plugin for domain validation.

ACME_POWERDNS_DOMAIN

This is the FQDN for the PowerDNS API (without path)

ACME_POWERDNS_SERVERID

This is the ServerID attribute of the PowerDNS API Server (i.e. "localhost")

ACME_POWERDNS_APIKEYNAME

This is the Key name to use for authentication (i.e. "X-API-Key")

ACME_POWERDNS_APIKEY

This is the API Key to use for authentication (i.e. "Password")

ACME_POWERDNS_RETRIES

This is the number of times DNS Verification should be attempted (i.e. 20)

ACME_POWERDNS_VERIFY

This configures how TLS certificates on the PowerDNS API target are validated. The PowerDNS Plugin depends on the PyPi requests library, which supports the following options for the verify parameter:

True: Verifies the TLS certificate was issued by a known publicly-trusted CA. (Default)

False: Disables certificate validation (Not Recommended)

File/Dir path to CA Bundle: Verifies the TLS certificate was issued by a Certificate Authority in the provided CA bundle.

3.2 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

3.2.1 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
```

acme

Handles all ACME related tasks, like ACME plugin testing.

```
lemur acme
```

3.2.2 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
```

3.3 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.

3.4 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.

3.4.1 Verisign/Symantec

Authors
Kevin Glisson <kglisson@netflix.com>, Curtis Castrapel <ccastrapel@netflix.com>, Hossein Shafagh <hshafagh@netflix.com>

Type
Issuer

Description
Basic support for the VICE 2.0 API

3.4.2 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.

3.4.3 Acme

Authors
Kevin Glisson <kglisson@netflix.com>, Curtis Castrapel <ccastrapel@netflix.com>, Hossein Shafagh <hshafagh@netflix.com>, Mikhail Khodorovskiy <mikhail.khodorovskiy@jivesoftware.com>, Chad Sine <csine@netflix.com>

Type
Issuer

Description
Adds support for the ACME protocol (including LetsEncrypt) with domain validation using several providers.

3.4.4 Atlas

Authors
Kevin Glisson <kglisson@netflix.com>, Curtis Castrapel <ccastrapel@netflix.com>, Hossein Shafagh <hshafagh@netflix.com>

Type
Metric

Description
Adds basic support for the Atlas telemetry system.

3.4.5 Email

Authors
Kevin Glisson <kglisson@netflix.com>, Curtis Castrapel <ccastrapel@netflix.com>, Hossein Shafagh <hshafagh@netflix.com>

Type
Notification

Description
Adds support for basic email notifications via SES.

3.4.6 Slack

Authors
Harm Weites <harm@weites.com>

Type
Notification

Description
Adds support for slack notifications.

3.4.7 AWS (Source)

Authors
Kevin Glisson <kglisson@netflix.com>, Curtis Castrapel <ccastrapel@netflix.com>, Hossein Shafagh <hshafagh@netflix.com>

Type
Source

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

3.4.8 AWS (Destination)

Authors
Kevin Glisson <kglisson@netflix.com>, Curtis Castrapel <ccastrapel@netflix.com>, Hossein Shafagh <hshafagh@netflix.com>

Type
Destination

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

3.4.9 AWS (SNS Notification)

Authors
Jasmine Schladen <jschladen@netflix.com>

Type
Notification

Description
Adds support for SNS notifications. SNS notifications (like other notification plugins) are currently only supported for certificate expiration. Configuration requires a region, account number, and SNS topic name; these elements are then combined to build the topic ARN. Lemur must have access to publish messages to the specified SNS topic.

3.4.10 Kubernetes

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

Type
Destination

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

3.4.11 Java

Authors
Kevin Glisson <kglisson@netflix.com>

Type
Export

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

3.4.12 Openssl

Authors
Kevin Glisson <kglisson@netflix.com>

Type
Export

Description
Leverages Openssl to support additional export formats (pkcs12)

3.4.13 CFSSL

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

Type
Issuer

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

3.4.14 Vault

Authors
Christopher Jolley <chris@alwaysjolley.com>

Type
Source

Description
Source plugin imports certificates from Hashicorp Vault secret store.

3.4.15 Vault

Authors
Christopher Jolley <chris@alwaysjolley.com>

Type
Destination

Description
Destination plugin to deploy certificates to Hashicorp Vault secret store.

3.5 3rd Party Plugins

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

3.5.1 Digicert

Authors
Chris Dorros

Type
Issuer

Description
Adds support for basic Digicert

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

3.5.2 InfluxDB

Authors
Titouan Christophe

Type
Metric

Description
Sends key metrics to InfluxDB

Links
https://github.com/titouanc/lemur-influxdb

3.5.3 Hashicorp Vault

Authors
Ron Cohen

Type
Issuer

Description
Adds support for basic Vault PKI secret backend.

Links
https://github.com/RcRonco/lemur_vault

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.

3.6 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

To allow integration with external access/membership management tools that may exist in your organization, lemur offers below plugins in addition to it's own RBAC implementation.

3.6.1 Membership Plugin

Authors
Sayali Charhate <scharhate@netflix.com>

Type
User Membership

Description
Adds support to learn and validate user membership details from an external service. User memberships are used to create user roles dynamically as described in *Identity and Access Management*. Configure this plugin slug as *USER_MEMBERSHIP_PROVIDER*

3.6.2 Authorization Plugins

Authors
Sayali Charhate <scharhate@netflix.com>

Type
External Authorization

Description
Adds support to implement custom authorization logic that is best suited for your enterprise. Lemur offers *AuthorizationPlugin* and its extended version *DomainAuthorizationPlugin*. One can implement *DomainAuthorizationPlugin* and configure its slug as *USER_DOMAIN_AUTHORIZATION_PROVIDER* to check if caller is authorized to issue a certificate for a given Common Name and Subject Alternative Name (SAN) of type DNSName

CHAPTER

FOUR

DEVELOPERS

4.1 Contributing

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

4.1.1 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 and activate python virtualenv from within the lemur repo
python3 -m venv env
. env/bin/activate

# Install doc requirements

make dev-docs

# Make the docs
cd docs
make html
```

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.

Adding New Modules to Documentation

When a new module is added, it will need to be added to the documentation. Ideally, we might rely on sphinx-apidoc to autogenerate our documentation. Unfortunately, this causes some build problems. Instead, you'll need to add new modules by hand.

4.1.2 Developing Against HEAD

We try to make it easy to get up and running in a development environment using a git checkout of Lemur. There are two ways to run Lemur locally: directly on your development machine, or in a Docker container.

Running in a Docker container

Look at the lemur-docker project. Usage instructions are self-contained in the README for that project.

Running directly on your development machine

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)
- 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
python3 -m venv env

# 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 create_config
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

If you just want to run tests in a Docker container, you can use Docker and docker-compose for running the tests with `docker-compose run test` directly in the `lemur` project.

(For running the Lemur service in Docker, see lemur-docker.)

4.1.3 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.

4.1.4 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 `make 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.

4.1.5 Static Media

Lemur uses a library that compiles its 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.

4.1.6 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
```

4.1.7 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

4.1.8 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

4.2 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.

4.2.1 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
```

(continues on next page)

(continued from previous page)

```
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.

4.2.2 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 four functions functions:

```
def create_certificate(self, csr, issuer_options):
    # requests.get('a third party')
def revoke_certificate(self, certificate, reason):
    # requests.put('a third party')
def get_ordered_certificate(self, order_id):
    # requests.get('already existing certificate')
def canceled_ordered_certificate(self, pending_cert, **kwargs):
    # requests.put('cancel an order that has yet to be issued')
```

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.

Asynchronous Certificates An issuer may take some time to actually issue a certificate for an order. In this case, a *PendingCertificate* is returned, which holds information to recreate a *Certificate* object at a later time. Then, *get_ordered_certificate()* should be run periodically via *python manage.py pending_certs fetch -i all* to attempt to retrieve an ordered certificate:

```
def get_ordered_ceriticate(self, order_id):
    # order_id is the external id of the order, not the external_id of the certificate
    # retrieve an order, and check if there is an issued certificate attached to it
```

cancel_ordered_certificate() should be implemented to allow an ordered certificate to be canceled before it is issued:

```
def cancel_ordered_certificate(self, pending_cert, **kwargs):
    # pending_cert should contain the necessary information to match an order
    # kwargs can be given to provide information to the issuer for canceling
```

4.2.3 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:

```
from lemur.common.utils import check_validation

options = [
    {
        'name': 'accountNumber',
        'type': 'int',
        'required': True,
        'validation': check_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 Python (re) and JavaScript regular expression used to give the user an indication if the input value is valid. Use *check_validation()* from *lemur.common.utils* to ensure your expression will compile successfully prior to use.
- **HelpMessage** simple string that provides more detail about the option

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

4.2.4 Notification

Lemur includes the ability to create Email notifications by **default**. These notifications currently come in the form of expiration and rotation notices for all certificates, expiration notices for CA certificates, and ACME certificate creation failure notices. Lemur periodically checks certificate 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.

Certificate expiration notifications can also be configured for Slack or AWS SNS. Other notifications are not configurable. Notifications sent to a certificate owner and security team (*LEMUR_SECURITY_TEAM_EMAIL*) can currently only be sent via email.

There are currently two objects that are available for notification plugins. The first is *NotificationPlugin*, which is the base object for any notification within Lemur. Currently the only supported notification type is a 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*, which inherits from the *NotificationPlugin* object. You will most likely want to base your plugin on this object if you want to add new channels for expiration notices (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")
```

4.2.5 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.

4.2.6 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.

4.2.7 Membership

Membership plugin allows Lemur to learn and validate membership details from an external service. Currently the plugin is configured to support 3 APIs:

```
def does_principal_exist(self, principal_email):
    raise NotImplementedError

def does_group_exist(self, group_email):
    # check if a group (Team DL) exists

def retrieve_user_memberships(self, user_id):
    # get a list of groups a user belongs to
```

4.2.8 Custom TLS Provider

Managing TLS at the enterprise scale could be hard and often organizations offer custom wrapper implementations. It could be ideal to use those while making calls to internal services. The *TLSPlugin* would help to achieve this. It requires the implementation of one function which creates a TLS session:

```
def session(self, server_application):
   # return active session
```

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:

4.2.9 setup.py

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

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

4.2.10 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
```

4.2.11 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""
```

4.2.12 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.

4.3 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.

4.3.1 Authentication

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

Bases: Resource

endpoint = 'google'

mediatypes()

methods = {'POST'}

A list of methods this view can handle.

post()

class lemur.auth.views.**Login**

Bases: 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()

methods = {'POST'}

A list of methods this view can handle.

post()

POST /auth/login

Login with username:password

Example request:

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

(continues on next page)

(continued from previous page)

```
Content-Type: application/json;charset=UTF-8

{
  "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: Resource

endpoint = 'oauth2'

get()

mediatypes()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post()

class lemur.auth.views.**Ping**

Bases: 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_restful.Resource` and create the HTTP methods the provider uses for its callbacks.
2. Add or change the Lemur AngularJS Configuration to point to your new provider

endpoint = 'ping'

get()

mediatypes()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post()

class lemur.auth.views.**Providers**

Bases: Resource

endpoint = 'providers'

get()

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

lemur.auth.views.**build_hmac()**

lemur.auth.views.**create_user_roles**(*profile*)

Creates new roles based on profile information.

Parameters

profile –

Returns

lemur.auth.views.**exchange_for_access_token**(*code*, *redirect_uri*, *client_id*, *secret*, *access_token_url=None*, *verify_cert=True*)

Exchanges authorization code for access token.

Parameters

- **code** –
- **redirect_uri** –
- **client_id** –
- **secret** –
- **access_token_url** –
- **verify_cert** –

Returns

Returns

lemur.auth.views.**generate_state_token()**

lemur.auth.views.**retrieve_user**(*user_api_url*, *access_token*)

Fetch user information from provided user api_url.

Parameters

- **user_api_url** –
- **access_token** –

Returns

lemur.auth.views.**retrieve_user_memberships**(*user_api_url*, *user_membership_provider*, *access_token*)

`lemur.auth.views.update_user(user, profile, roles)`

Updates user with current profile information and associated roles.

Parameters

- **user** –
- **profile** –
- **roles** –

`lemur.auth.views.validate_id_token(id_token, client_id, jwks_url)`

Ensures that the token we receive is valid.

Parameters

- **id_token** –
- **client_id** –
- **jwks_url** –

Returns

`lemur.auth.views.verify_state_token(token)`

4.3.2 Destinations

class `lemur.destinations.views.CertificateDestinations`

Bases: `AuthenticatedResource`

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

endpoint = 'certificateDestinations'

get(*certificate_id*)

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",
```

(continues on next page)

(continued from previous page)

```
            "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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

delete(*destination_id*)

endpoint = 'destination'

get(*destination_id*)

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()

methods = {'DELETE', 'GET', 'PUT'}

A list of methods this view can handle.

put(*destination_id*, *data=None*)

PUT /destinations/1

Updates an account

Example request:

```
POST /destinations/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8


{
  "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",
```

(continues on next page)

(continued from previous page)

```
            "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: AuthenticatedResource

Defines the 'destinations' endpoint

endpoint = 'destinations'

get()

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!",
```

(continues on next page)

(continued from previous page)

```
            "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /destinations

Creates a new account

Example request:

```
POST /destinations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "description": "test33",
  "options": [{
      "name": "accountNumber",
```

(continues on next page)

(continued from previous page)

```
        "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"
```

(continues on next page)

(continued from previous page)

```
  },
  "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: AuthenticatedResource

Defines the 'destinations' stats endpoint

endpoint = 'destinationStats'

get()

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

4.3.3 Notifications

class lemur.notifications.views.**CertificateNotifications**

Bases: AuthenticatedResource

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

endpoint = 'certificateNotifications'

get(*certificate_id*)

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": [
```

(continues on next page)

(continued from previous page)

```
    {
        "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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

delete(*notification_id*)

endpoint = 'notification'

get(*notification_id*)

GET /notifications/1

Get a specific notification

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",
```

(continues on next page)

(continued from previous page)

```
            "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()

methods = {'DELETE', 'GET', 'PUT'}

A list of methods this view can handle.

put(*notification_id*, *data=None*)

PUT /notifications/1

Updates a notification

Example request:

```
PUT /notifications/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "label": "labelChanged",
  "plugin": {
      "slug": "email-notification",
      "plugin_options": "???"
    },
  "description": "Sample notification",
  "active": "true",
  "added_certificates": "???",
  "removed_certificates": "???"
}
```

Example response:

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

{
```

(continues on next page)

(continued from previous page)

```
  "id": 1,
  "label": "labelChanged",
  "plugin": {
      "slug": "email-notification",
      "plugin_options": "???"
    },
  "description": "Sample notification",
  "active": "true",
  "added_certificates": "???",
  "removed_certificates": "???"
}
```

Label label
notification name

Label slug
notification plugin slug

Label plugin_options
notification plugin options

Label description
notification description

Label active
whether or not the notification is active/enabled

Label added_certificates
certificates to add

Label removed_certificates
certificates to remove

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

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

Bases: `AuthenticatedResource`

Defines the 'notifications' endpoint

endpoint = 'notifications'

get()

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
```

(continues on next page)

(continued from previous page)

```
{
  "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /notifications

Creates a new notification

Example request:

```
POST /notifications HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "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",
```

(continues on next page)

(continued from previous page)

```
  "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
}
```

Label label
notification name

Label slug

notification plugin slug

Label plugin_options

notification plugin options

Label description

notification description

Label active

whether or not the notification is active/enabled

Label certificates

certificates to attach to notification

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

4.3.4 Users

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

Bases: `AuthenticatedResource`

endpoint = 'certificateCreator'

get(*certificate_id*)

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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.users.views.**Me**

Bases: AuthenticatedResource

endpoint = 'me'

get()

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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.users.views.**RoleUsers**

Bases: AuthenticatedResource

endpoint = 'roleUsers'

get(*role_id*)

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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

endpoint = 'user'

get(*user_id*)

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
```

(continues on next page)

(continued from previous page)

```
{
    "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()

methods = {'GET', 'PUT'}

A list of methods this view can handle.

put(*user_id*, *data=None*)

PUT /users/1

Update a user

Example request with ID:

```
PUT /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

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

Example request with name:

```
PUT /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "username": "user1",
   "email": "user1@example.com",
   "active": false,
   "roles": [
       {"name": "myRole"}
```

(continues on next page)

(continued from previous page)

```
   ]
}
```

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: AuthenticatedResource

Defines the 'users' endpoint

endpoint = 'users'

get()

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
```

(continues on next page)

(continued from previous page)

```
      },
      {
         "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /users

Creates a new user

Example request with ID:

```
POST /users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

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

Example request with name:

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

(continues on next page)

(continued from previous page)

```
Content-Type: application/json;charset=UTF-8

{
   "username": "user3",
   "email": "user3@example.com",
   "active": true,
   "roles": [
      {"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

4.3.5 Roles

class lemur.roles.views.**AuthorityRolesList**

Bases: AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'authorityRoles'

get(*authority_id*)

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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.roles.views.**RoleViewCredentials**

Bases: 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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

delete(*role_id*)

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(*role_id*)

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()

methods = {'DELETE', 'GET', 'PUT'}

A list of methods this view can handle.

put(*role_id*, *data=None*)

PUT /roles/1

Update a role

Example request:

```
PUT /roles/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "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: AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'roles'

get()

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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /roles

Creates a new role

Example request:

```
POST /roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "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: AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'userRoles'

get(*user_id*)

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()

methods = {'GET'}

A list of methods this view can handle.

4.3.6 Certificates

class lemur.certificates.views.**CertificateDeactivate**

Bases: AuthenticatedResource

endpoint = 'deactivateCertificate'

mediatypes()

methods = {'PUT'}

A list of methods this view can handle.

put(*certificate_id*)

PUT /certificates/1/deactivate

deactivate a certificate (integration test only) **Example request**:

```
PUT /certificates/1/deactivate HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8
```

Example response:

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

{
  "id": 1
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated or cert attached to LB
- 400 Bad Request – encountered error, more details in error message

class lemur.certificates.views.**CertificateExport**

Bases: AuthenticatedResource

endpoint = 'exportCertificate'

mediatypes()

methods = {'POST'}

A list of methods this view can handle.

post(*certificate_id*, *data=None*)

POST /certificates/1/export

Export a certificate

Example request:

```
PUT /certificates/1/export HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "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
```

(continues on next page)

(continued from previous page)

```
{
  "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: 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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.certificates.views.**CertificateRevoke**

Bases: AuthenticatedResource

endpoint = 'revokeCertificate'

mediatypes()

methods = {'PUT'}

A list of methods this view can handle.

put(*certificate_id*, *data=None*)

PUT /certificates/1/revoke

Revoke a certificate. One can mention the reason of revocation using crlReason (optional) as per RFC 5280 section 5.3.1 The allowed values for crlReason can also be found in Lemur in constants.py/CRLReason Additional information can be captured using comments (optional).

Example request:

```
PUT /certificates/1/revoke HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "crlReason": "affiliationChanged",
  "comments": "Additional details if any"
}
```

Example response:

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

{
  "id": 1
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated or cert attached to LB
- 400 Bad Request – encountered error, more details in error message

class lemur.certificates.views.**CertificateUpdateOwner**

Bases: AuthenticatedResource

endpoint = 'certificateUpdateOwner'

mediatypes()

methods = {'POST'}

A list of methods this view can handle.

post(*certificate_id*, *data=None*)

POST /certificates/1/update/owner

Update certificate owner

Example request:

```
POST /certificates/1/update/owner HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "owner": "joan@example.com"
}
```

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",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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,
  "notify": false,
  "rotation": false,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
  "user": {
      "username": "jane",
      "active": true,
      "email": "jane@example.com",
      "id": 2
  },
  "active": true,
  "domains": [{
      "sensitive": false,
```

(continues on next page)

(continued from previous page)

```
        "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"
    }],
    "rotation": true,
    "rotationPolicy": {"name": "default"},
    "san": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.certificates.views.**Certificates**

Bases: AuthenticatedResource

delete(*certificate_id*, *data=None*)

DELETE /certificates/1

Delete a certificate

Example request:

```
DELETE /certificates/1 HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 204 OK
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 204 No Content – no error
- 403 Forbidden – unauthenticated
- 404 Not Found – certificate not found
- 405 Method Not Allowed – certificate deletion is disabled

endpoint = 'certificateUpdateSwitches'

get(*certificate_id*)

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": "",
  "csr": "-----BEGIN CERTIFICATE REQUEST-----"
  "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",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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"
  }],
  "rotation": true,
  "rotationPolicy": {"name": "default"},
  "replaces": [],
```

(continues on next page)

(continued from previous page)

```
  "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()

methods = {'DELETE', 'GET', 'POST', 'PUT'}

A list of methods this view can handle.

post(*certificate_id*, *data=None*)

POST /certificates/1/update/switches

Update certificate boolean switches for notification or rotation

Example request:

```
POST /certificates/1/update/switches HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "notify": false,
   "rotation": false
}
```

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"
```

(continues on next page)

(continued from previous page)

```
  },
  "owner": "joe@example.com",
  "serial": "82311058732025924142789179368889309156",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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,
  "notify": false,
  "rotation": false,
  "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"
  }],
  "rotation": true,
  "rotationPolicy": {"name": "default"},
  "san": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

put(*certificate_id*, *data=None*)

PUT /certificates/1

Update a certificate

Example request:

```
PUT /certificates/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "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",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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
```

(continues on next page)

(continued from previous page)

```
  },
  "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"
  }],
  "rotation": true,
  "rotationPolicy": {"name": "default"},
  "san": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

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

Bases: AuthenticatedResource

Defines the 'certificates' endpoint

endpoint = 'certificates'

get()

GET /certificates

The current list of certificates. This API supports additional params like

Pagination, sorting:

/certificates?count=10&page=1&short=true&sortBy=id&sortDir=desc

Filters, mentioned as url param filter=field;value

/certificates?filter=cn;lemur.test.com /certificates?filter=notify;true /certificates?filter=rotation;true /certificates?filter=name;lemur.test.cert /certificates?filter=issuer;Digicert

Request expired certs

/certificates?showExpired=1

Search by Serial Number

Decimal: /certificates?serial=218243997808053074560741989466015229225 Hex: /certificates?serial=0xA43043DAB7F6F8AE115E94854EEB6529 /certificates?serial=a4:30:43:da:b7:f6:f8:ae:11:5e:94:85:4e:eb:65:29

Example request:

```
GET /certificates?serial=82311058732025924142789179368889309156 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": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----"
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
      "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,
```

(continues on next page)

(continued from previous page)

```
            "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /certificates

Creates a new certificate

Example request:

```
POST /certificates HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
    "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"
          }
        ]
```

(continues on next page)

(continued from previous page)

```
      }
    },
    "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": "82311058732025924142789179368889309156",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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",
```

(continues on next page)

(continued from previous page)

```
        "active": true,
        "email": "jane@example.com",
        "id": 2
    },
    "active": true,
    "domains": [{
        "sensitive": false,
        "id": 1090,
        "name": "*.test.example.net"
    }],
    "replaces": [{
        "id": 1
    }],
    "rotation": true,
    "rotationPolicy": {"name": "default"},
    "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.**CertificatesListValid**

Bases: AuthenticatedResource

Defines the 'certificates/valid' endpoint

endpoint = 'certificatesListValid'

get()

GET /certificates/valid/<query>

The current list of not-expired certificates for a given common name, and owner. The API offers optional pagination. One can send page number(>=1) and desired count per page. The returned data contains total number of certificates which can help in determining the last page. Pagination will not be offered if page or count info is not sent or if it is zero.

Example request:

```
GET /certificates/valid?filter=cn;*.test.example.net&owner=joe@example.com&
↪page=1&count=20 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response (with single cert to be concise):

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

{
  "items": [{
      "status": null,
      "cn": "*.test.example.net",
      "chain": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----"
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
      "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
```

(continues on next page)

(continued from previous page)

```
↪",
            "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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.certificates.views.**CertificatesNameQuery**

Bases: AuthenticatedResource

Defines the 'certificates/name' endpoint

endpoint = 'certificatesNameQuery'

get(*certificate_name*)

GET /certificates/name/<query>

The current list of certificates

Example request:

```
GET /certificates/name/WILDCARD.test.example.net-SymantecCorporation-
↪20160603-20180112 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": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----"
      "authority": {
          "active": true,
          "owner": "secure@example.com",
          "id": 1,
          "description": "verisign test authority",
```

(continues on next page)

(continued from previous page)


```
            "name": "verisign"
        },
        "owner": "joe@example.com",
        "serial": "82311058732025924142789179368889309156",
        "id": 2288,
        "issuer": "SymantecCorporation",
        "dateCreated": "2016-06-03T06:09:42.133769+00:00",
        "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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

endpoint = 'replacements'

get(*certificate_id*)

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": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----",
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
      "notBefore": "2016-06-03T00:00:00+00:00",
      "notAfter": "2018-01-12T23:59:59+00:00",
      "destinations": [],
      "bits": 2048,
      "body": "-----BEGIN CERTIFICATE-----...",
      "description": null,
```

(continues on next page)

(continued from previous page)

```
        "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": [],
        "rotation": true,
        "rotationPolicy": {"name": "default"},
        "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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

Defines the 'certificates' stats endpoint

endpoint = 'certificateStats'

get()

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

Defines the 'certificates' upload endpoint

endpoint = 'certificateUpload'

mediatypes()

methods = {'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /certificates/upload

Upload a certificate

Example request:

```
POST /certificates/upload HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "owner": "joe@example.com",
   "body": "-----BEGIN CERTIFICATE-----...",
   "chain": "-----BEGIN CERTIFICATE-----...",
   "privateKey": "-----BEGIN RSA PRIVATE KEY-----..."
   "csr": "-----BEGIN CERTIFICATE REQUEST-----..."
   "destinations": [],
   "notifications": [],
   "replacements": [],
   "roles": [],
   "notify": true,
   "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",
```

(continues on next page)

(continued from previous page)


```
        "id": 1,
        "description": "verisign test authority",
        "name": "verisign"
    },
    "owner": "joe@example.com",
    "serial": "82311058732025924142789179368889309156",
    "id": 2288,
    "issuer": "SymantecCorporation",
    "dateCreated": "2016-06-03T06:09:42.133769+00:00",
    "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": [],
    "rotation": true,
    "rotationPolicy": {"name": "default"},
    "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

- 403 Forbidden – unauthenticated
- 200 OK – no error

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

Bases: AuthenticatedResource

Defines the 'certificates' endpoint

endpoint = 'notificationCertificates'

get(*notification_id*)

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": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----"
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
      "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
```

(continues on next page)

(continued from previous page)

```
      },
      "active": true,
      "domains": [{
          "sensitive": false,
          "id": 1090,
          "name": "*.test.example.net"
      }],
      "replaces": [],
      "replaced": [],
      "rotation": true,
      "rotationPolicy": {"name": "default"},
      "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()

methods = {'GET'}

A list of methods this view can handle.

4.3.7 Authorities

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

Bases: AuthenticatedResource

endpoint = 'authority'

get(*authority_id*)

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()

methods = {'GET', 'PUT'}

A list of methods this view can handle.

put(*authority_id*, *data=None*)

PUT /authorities/1

Update an authority

Example request:

```
PUT /authorities/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "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",
```

(continues on next page)

(continued from previous page)

```
  "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: AuthenticatedResource

Defines the 'authorities' endpoint

endpoint = 'authorities'

get()

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␣
```

(continues on next page)

(continued from previous page)

```
↪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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /authorities

Create an authority

Example request:

```
POST /authorities HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "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",
```

(continues on next page)

(continued from previous page)

```
   "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␣
```

(continues on next page)

(continued from previous page)

```
↪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
- **keyName** – name of the key to store in the HSM (CloudCA)
- **serialNumber** – serial number of the authority
- **firstSerial** – specifies the starting serial number for certificates issued off of this authority

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 403 Forbidden – unauthenticated
- 200 OK – no error

class lemur.authorities.views.**AuthorityVisualizations**

Bases: AuthenticatedResource

endpoint = 'authority_visualizations'

get(*authority_id*)

GET /authorities/1/visualize

Authority visualization

Example request:

```
GET /certificates/1/visualize 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": "flare",
      "children": [
          {
```

(continues on next page)

(continued from previous page)

```
                "name": "analytics",
                "children": [
                    {
                        "name": "cluster",
                        "children": [
                            {"name": "AgglomerativeCluster", "size": 3938},
                            {"name": "CommunityStructure", "size": 3812},
                            {"name": "HierarchicalCluster", "size": 6714},
                            {"name": "MergeEdge", "size": 743}
                        ]
                    }
                ]
            }
        ]
    }
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

class lemur.authorities.views.**CertificateAuthority**

Bases: AuthenticatedResource

endpoint = 'certificateAuthority'

get(*certificate_id*)

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"
```

(continues on next page)

(continued from previous page)

```
  }, {
      "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()

methods = {'GET'}

A list of methods this view can handle.

4.3.8 Domains

class lemur.domains.views.**CertificateDomains**

Bases: AuthenticatedResource

Defines the 'domains' endpoint

endpoint = 'certificateDomains'

get(*certificate_id*)

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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.domains.views.**Domains**

Bases: AuthenticatedResource

endpoint = 'domain'

get(*domain_id*)

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()

methods = {'GET', 'PUT'}

A list of methods this view can handle.

put(*domain_id*, *data=None*)

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: AuthenticatedResource

Defines the 'domains' endpoint

endpoint = 'domains'

get()

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
      }
    ]
```

(continues on next page)

(continued from previous page)

```
  "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /domains

The current domain list

Example request:

```
POST /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

4.3.9 Endpoints

class lemur.endpoints.views.**Endpoints**

Bases: AuthenticatedResource

endpoint = 'endpoint'

get(*endpoint_id*)

GET /endpoints/1

One endpoint

Example request:

```
GET /endpoints/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
```

Request Headers
- Authorization – OAuth token to authenticate

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

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

class lemur.endpoints.views.**EndpointsList**

Bases: AuthenticatedResource

Defines the 'endpoints' endpoint

endpoint = 'endpoints'

get()

GET /endpoints

The current list of endpoints

Example request:

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

Example response:

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

Query Parameters

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

methods = {'GET'}

A list of methods this view can handle.

4.3.10 Logs

class lemur.logs.views.**LogsList**

Bases: AuthenticatedResource

Defines the 'logs' endpoint

endpoint = 'logs'

get()

GET /logs

The current log list

Example request:

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

Example response:

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

(continues on next page)

(continued from previous page)

```
{
  "items": [
    ]
  "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()

methods = {'GET'}

A list of methods this view can handle.

4.3.11 Sources

class lemur.sources.views.**CertificateSources**

Bases: AuthenticatedResource

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

endpoint = 'certificateSources'

get(*certificate_id*)

GET /certificates/1/sources

The current account list for a given certificates

Example request:

```
GET /certificates/1/sources 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": [
      {
          "options": [
```

(continues on next page)

(continued from previous page)

```
                {
                    "name": "accountNumber",
                    "required": true,
                    "value": 111111111112,
                    "helpMessage": "Must be a valid AWS account number!",
                    "validation": "^[0-9]{12,12}$",
                    "type": "int"
                }
            ],
            "pluginName": "aws-source",
            "id": 3,
            "lastRun": "2015-08-01T15:40:58",
            "description": "test",
            "label": "test"
        }
    ],
    "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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.sources.views.**Sources**

Bases: AuthenticatedResource

delete(*source_id*)

endpoint = 'account'

get(*source_id*)

GET /sources/1

Get a specific account

Example request:

```
GET /sources/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

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
  ],
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
  "label": "test"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes()

methods = {'DELETE', 'GET', 'PUT'}

A list of methods this view can handle.

put(*source_id*, *data=None*)

PUT /sources/1

Updates an account

Example request:

```
POST /sources/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
```

(continues on next page)

(continued from previous page)

```
  ],
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
  "label": "test"
}
```

Example response:

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

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
  ],
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
  "label": "test"
}
```

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.sources.views.**SourcesList**

Bases: AuthenticatedResource

Defines the 'sources' endpoint

endpoint = 'sources'

get()

GET /sources

The current account list

Example request:

```
GET /sources 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": [
      {
          "options": [
              {
                  "name": "accountNumber",
                  "required": true,
                  "value": 111111111112,
                  "helpMessage": "Must be a valid AWS account number!",
                  "validation": "^[0-9]{12,12}$",
                  "type": "int"
              }
          ],
          "pluginName": "aws-source",
          "lastRun": "2015-08-01T15:40:58",
          "id": 3,
          "description": "test",
          "label": "test"
      }
  ],
  "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /sources

Creates a new account

Example request:

```
POST /sources HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
  ],
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
  "label": "test"
}
```

Example response:

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

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
  ],
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
  "label": "test"
}
```

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

4.4 Internals

4.4.1 lemur Package

lemur Package

constants Module

class lemur.constants.**CRLReason**(*value*)

Bases: IntEnum

An enumeration.

aACompromise = 10

affiliationChanged = 3

cACompromise = 2

certificateHold = 6

cessationOfOperation = 5

keyCompromise = 1

privilegeWithdrawn = 9

removeFromCRL = 8

superseded = 4

unspecified = 0

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 from a 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.get_count(q)`

Count the number of rows in a table. More efficient than count(*) :param q: :return:

`lemur.database.get_model_column(model, field)`

`lemur.database.rollback()`

Helper to rollback the current session.

`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. Note that paging is not safe unless combined with sorting.

Parameters

- **query** – search query
- **model** – model to use for resulting items
- **args** – arguments to query with, including sorting and paging parameters

Returns
the items given the count and page specified

`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

exceptions Module

exception `lemur.exceptions.AttrNotFound(field)`

Bases: `LemurException`

exception `lemur.exceptions.DuplicateError(key)`

Bases: `LemurException`

exception `lemur.exceptions.InvalidAuthority`

Bases: `Exception`

exception `lemur.exceptions.InvalidConfiguration`

Bases: `Exception`

exception `lemur.exceptions.InvalidDistribution(field)`

Bases: `LemurException`

exception `lemur.exceptions.InvalidListener(*args, **kwargs)`

Bases: `LemurException`

exception `lemur.exceptions.LemurException(*args, **kwargs)`

Bases: `Exception`

exception `lemur.exceptions.UnknownProvider`

Bases: `Exception`

extensions Module

class lemur.extensions.**SQLAlchemy**(*app=None*, *use_native_unicode=True*, *session_options=None*, *metadata=None*, *query_class=<class 'flask_sqlalchemy.BaseQuery'>*, *model_class=<class 'flask_sqlalchemy.model.Model'>*, *engine_options=None*)

Bases: SQLAlchemy

apply_pool_defaults(*app*, *options*)

Set default engine options. We enable *pool_pre_ping* to be the default value.

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_database**(*app*)

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

`class lemur.manage.CreateRole(func=None)`

Bases: `Command`

This command allows for the creation of a new role within Lemur

option_list = (<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.commands.Option object>)

`run(name, users, description)`

Runs a command. This must be implemented by the subclass. Should take arguments as configured by the Command options.

`class lemur.manage.CreateUser(func=None)`

Bases: `Command`

This command allows for the creation of a new user within Lemur.

option_list = (<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.commands.Option object>)

`run(username, email, active, roles, password)`

Runs a command. This must be implemented by the subclass. Should take arguments as configured by the Command options.

`class lemur.manage.InitializeApp(func=None)`

Bases: `Command`

This command will bootstrap our database with any destinations as specified by our config.

Additionally a Lemur user will be created as a default user and be used when certificates are discovered by Lemur.

option_list = (<flask_script.commands.Option object>,)

`run(password)`

Runs a command. This must be implemented by the subclass. Should take arguments as configured by the Command options.

class lemur.manage.**LemurServer**(*func=None*)

Bases: Command

This is the main Lemur server, it runs the flask app with gunicorn and uses any configuration options passed to it.

You can pass all standard gunicorn flags to this command as if you were running gunicorn itself.

For example:

lemur start -w 4 -b 127.0.0.0:8002

Will start gunicorn with 4 workers bound to 127.0.0.0:8002

description = 'Run the app within Gunicorn'

get_options()

By default, returns self.option_list. Override if you need to do instance-specific configuration.

run(**args*, ***kwargs*)

Runs a command. This must be implemented by the subclass. Should take arguments as configured by the Command options.

class lemur.manage.**ResetPassword**(*func=None*)

Bases: Command

This command allows you to reset a user's password.

option_list = (<flask_script.commands.Option object>,)

run(*username*)

Runs a command. This must be implemented by the subclass. Should take arguments as configured by the Command options.

lemur.manage.**create**()

lemur.manage.**create_config**(*config_path=None*)

Creates a new configuration file if one does not already exist

lemur.manage.**drop_all**()

lemur.manage.**generate_settings**()

This command is run when `default_path` doesn't exist, or `init` is run and returns a string representing the default data to put into their settings file.

lemur.manage.**lock**(*path=None*)

Encrypts a given path. This directory can be used to store secrets needed for normal Lemur operation. This is especially useful for storing secrets needed for communication with third parties (e.g. external certificate authorities).

Lemur does not assume anything about the contents of the directory and will attempt to encrypt all files contained within. Currently this has only been tested against plain text files.

Path defaults ~/.lemur/keys

Param
path

lemur.manage.**main**()

`lemur.manage.make_shell_context()`

Creates a python REPL with several default imports in the context of the current_app

Returns

`lemur.manage.publish_verisign_units()`

Simple function that queries verisign for API units and posts the mertics to Atlas API for other teams to consume. :return:

`lemur.manage.unlock(path=None)`

Decrypts all of the files in a given directory with provided password. This is most commonly used during the startup sequence of Lemur allowing it to go from source code to something that can communicate with external services.

Path defaults ~/.lemur/keys

Param
path

`models` Module

Subpackages

auth Package

`permissions` Module

class `lemur.auth.permissions.ApiKeyCreatorPermission`

Bases: `Permission`

`lemur.auth.permissions.AuthorityCreator`

alias of `authority`

`lemur.auth.permissions.AuthorityOwner`

alias of `authority`

class `lemur.auth.permissions.AuthorityPermission(authority_id, roles)`

Bases: `Permission`

`lemur.auth.permissions.CertificateOwner`

alias of `certificate`

class `lemur.auth.permissions.CertificatePermission(owner, roles)`

Bases: `Permission`

`lemur.auth.permissions.RoleMember`

alias of `role`

class `lemur.auth.permissions.RoleMemberPermission(role_id)`

Bases: `Permission`

class `lemur.auth.permissions.SensitiveDomainPermission`

Bases: `Permission`

service Module

class lemur.auth.service.**AuthenticatedResource**

Bases: Resource

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

method_decorators = [<function login_required>]

lemur.auth.service.**create_token**(*user*, *aid=None*, *ttl=None*)

Create a valid JWT for a given user/api key, 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: Resource

endpoint = 'google'

mediatypes()

methods = {'POST'}

A list of methods this view can handle.

post()

class lemur.auth.views.**Login**

Bases: 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()

methods = {'POST'}

A list of methods this view can handle.

post()

POST /auth/login

Login with username:password

Example request:

```
POST /auth/login HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "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: Resource

endpoint = 'oauth2'

get()

mediatypes()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post()

class lemur.auth.views.**Ping**

Bases: 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_restful.Resource` and create the HTTP methods the provider uses for its callbacks.
2. Add or change the Lemur AngularJS Configuration to point to your new provider

endpoint = 'ping'

get()

mediatypes()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post()

class lemur.auth.views.**Providers**

Bases: Resource

endpoint = 'providers'

get()

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

lemur.auth.views.**build_hmac**()

lemur.auth.views.**create_user_roles**(*profile*)

Creates new roles based on profile information.

Parameters

profile –

Returns

lemur.auth.views.**exchange_for_access_token**(*code*, *redirect_uri*, *client_id*, *secret*, *access_token_url=None*, *verify_cert=True*)

Exchanges authorization code for access token.

Parameters

- **code** –
- **redirect_uri** –
- **client_id** –
- **secret** –
- **access_token_url** –
- **verify_cert** –

Returns

Returns

lemur.auth.views.**generate_state_token**()

lemur.auth.views.**retrieve_user**(*user_api_url*, *access_token*)

Fetch user information from provided user api_url.

Parameters

- **user_api_url** –
- **access_token** –

Returns

lemur.auth.views.**retrieve_user_memberships**(*user_api_url*, *user_membership_provider*, *access_token*)

lemur.auth.views.**update_user**(*user*, *profile*, *roles*)

Updates user with current profile information and associated roles.

Parameters

- **user** –
- **profile** –
- **roles** –

`lemur.auth.views.validate_id_token(id_token, client_id, jwks_url)`

Ensures that the token we receive is valid.

Parameters

- **id_token** –
- **client_id** –
- **jwks_url** –

Returns

`lemur.auth.views.verify_state_token(token)`

authorities Package

models Module

class `lemur.authorities.models.Authority(**kwargs)`

Bases: `Model`

active

authority_certificate

authority_pending_certificate

body

certificates

chain

date_created

property default_validity_days

description

id

property is_cab_compliant

Parse the options to find whether authority is CAB Forum Compliant, i.e., adhering to the CA/Browser Forum Baseline Requirements. Returns None if option is not available

property is_cn_optional

Parse the options to find whether common name is treated as an optional field. Returns False if option is not available

property is_private_authority

Tells if authority is private/internal. In other words, it is not publicly trusted. If plugin is configured in list LEMUR_PRIVATE_AUTHORITY_PLUGIN_NAMES, the authority is treated as private :return: True if private, False otherwise

property max_issuance_days

name

options

owner

pending_certificates

property 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_authorities_by_name**(*authority_names*)

Retrieves an authority given it's name.

Parameters
authority_names – list with authority names to match

Returns

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, owner, active, roles)`

Update an authority with new values.

Parameters

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

Returns

`lemur.authorities.service.update_options(authority_id, options)`

Update an authority with new options.

Parameters

- **authority_id** –
- **options** – the new options to be saved into the authority

Returns

views Module

class `lemur.authorities.views.Authorities`

Bases: `AuthenticatedResource`

endpoint = 'authority'

get(*authority_id*)

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,
```

(continues on next page)

(continued from previous page)

```
      "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()

methods = {'GET', 'PUT'}

A list of methods this view can handle.

put(*authority_id*, *data=None*)

PUT /authorities/1

Update an authority

Example request:

```
PUT /authorities/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "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-----",
```

(continues on next page)

(continued from previous page)

```
      "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": "",
```

(continues on next page)

(continued from previous page)

```
      "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: AuthenticatedResource

Defines the 'authorities' endpoint

endpoint = 'authorities'

get()

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": [{
```

(continues on next page)

(continued from previous page)


```
            "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /authorities

Create an authority

Example request:

```
POST /authorities HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "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"
  }, {
```

(continues on next page)

(continued from previous page)

```
        "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
- **keyName** – name of the key to store in the HSM (CloudCA)
- **serialNumber** – serial number of the authority

- **firstSerial** – specifies the starting serial number for certificates issued off of this authority

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 403 Forbidden – unauthenticated
- 200 OK – no error

class lemur.authorities.views.**AuthorityVisualizations**

Bases: AuthenticatedResource

endpoint = 'authority_visualizations'

get(*authority_id*)

GET /authorities/1/visualize

Authority visualization

Example request:

```
GET /certificates/1/visualize 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": "flare",
      "children": [
          {
              "name": "analytics",
              "children": [
                  {
                      "name": "cluster",
                      "children": [
                          {"name": "AgglomerativeCluster", "size": 3938},
                          {"name": "CommunityStructure", "size": 3812},
                          {"name": "HierarchicalCluster", "size": 6714},
                          {"name": "MergeEdge", "size": 743}
                      ]
                  }
              ]
          }
      ]
  }
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

class lemur.authorities.views.**CertificateAuthority**

Bases: AuthenticatedResource

endpoint = 'certificateAuthority'

get(*certificate_id*)

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
      },
```

(continues on next page)

(continued from previous page)

```
        "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()

methods = {'GET'}

A list of methods this view can handle.

certificates Package

models Module

class lemur.certificates.models.**Certificate**(***kwargs*)

Bases: Model

property active

authority

authority_id

bits

body

certificate_associations

chain

check_integrity()

Integrity checks: Does the cert have a valid chain and matching private key?

cn

property country

csr

date_created

deleted

description

destinations

property distinguished_name

dns_provider

dns_provider_id

domains = ObjectAssociationProxyInstance(AssociationProxy('certificate_associations', 'domain'))

endpoints

expired

property extensions

external_id

has_private_key

id

in_rotation_window

Determines if a certificate is available for rotation based on the rotation policy associated. :return:

issuer

ix = Index('ix_certificates_id_desc', <sqlalchemy.sql.elements.UnaryExpression object>, unique=True)

key_type

property location

logs

name

not_after

not_after_ix = Index('ix_certificates_not_after', <sqlalchemy.sql.elements.UnaryExpression object>)

not_before

notification

notifications

notify

property organization

property organizational_unit

owner

property parsed_cert

pending_cert

private_key

property public_key

replaced

replaced_by_pending

replaces

revoked

role

roles

root_authority

root_authority_id

rotation

rotation_policy

rotation_policy_id

san

sensitive_fields = ('private_key',)

serial

signing_algorithm

sources

property state

status

property subject

user

user_id

property validity_range

property validity_remaining

class lemur.certificates.models.**CertificateAssociation**(*domain=None*, *certificate=None*, *ports=None*)

Bases: Model

certificate

certificate_id

domain

domain_id

ports

lemur.certificates.models.**get_or_increase_name**(*name*, *serial*)

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.**allowed_issuance_for_domain**(*common_name*, *extensions*)

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.**cleanup_after_revoke**(*certificate*)

Perform the needed cleanup for a revoked certificate. This includes - 1. Notify (if enabled) 2. Disabling notification 3. Disabling auto-rotation 4. Update certificate status to 'revoked' 5. Remove from AWS :param certificate: Certificate object to modify and update in DB :return: None

lemur.certificates.service.**cleanup_owner_roles_notification**(*owner_name*, *kwargs*)

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.deactivate(certificate)`

`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_and_persist_domains_where_cert_is_deployed(certificate, excluded_domains, commit, timeout_seconds_per_network_call)`

Checks if the specified cert is still deployed. Returns a list of domains to which it's deployed.

We use the serial number to identify that a certificate is identical. If there were multiple certificates issued for the same domain with identical serial numbers, this could return a false positive.

Note that this checks *all* configured ports (specified in config LEMUR_PORTS_FOR_DEPLOYED_CERTIFICATE_CHECK) for all the domains in the cert. If the domain is valid but the port is not, we have to wait for the connection to time out, which means this can be quite slow.

Returns
A dictionary of the form {'domain1': [ports], 'domain2': [ports]}

`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_certs_attached_to_destination_without_autorotate(plugin_name=None)`

Retrieves all certificates that are attached to a destination, but that do not have autorotate enabled.

Parameters
plugin_name – Optional destination plugin name to query. Queries certificates attached to any destination if not provided.

Returns
list of certificates attached to a destination without autorotate

`lemur.certificates.service.get_all_certs_attached_to_endpoint_without_autorotate()`

Retrieves all certificates that are attached to an endpoint, but that do not have autorotate enabled.

Returns
list of certificates attached to an endpoint without autorotate

`lemur.certificates.service.get_all_pending_cleaning_expired(source)`

Retrieves all certificates that are available for cleaning. These are certificates which are expired and are not attached to any endpoints.

Parameters
source – the source to search for certificates

Returns
list of pending certificates

`lemur.certificates.service.get_all_pending_cleaning_expiring_in_days(source, days_to_expire)`

Retrieves all certificates that are available for cleaning, not attached to endpoint, and within X days from expiration.

Parameters

- **days_to_expire** – defines how many days till the certificate is expired
- **source** – the source to search for certificates

Returns
list of pending certificates

`lemur.certificates.service.get_all_pending_cleaning_issued_since_days(source, days_since_issuance)`

Retrieves all certificates that are available for cleaning: not attached to endpoint, and X days since issuance.

Parameters

- **days_since_issuance** – defines how many days since the certificate is issued
- **source** – the source to search for certificates

Returns
list of pending certificates

`lemur.certificates.service.get_all_pending_reissue()`

Retrieves all certificates that need to be rotated.

Must be X days from expiration, uses the certificates rotation policy to determine how many days from expiration the certificate must be for rotation to be pending.

Returns

`lemur.certificates.service.get_all_valid_certificates_with_destination(destination_id)`

Return list of certificates :param destination_id: :return:

`lemur.certificates.service.get_all_valid_certificates_with_source(source_id)`

Return list of certificates :param source_id: :return:

`lemur.certificates.service.get_all_valid_certs(authority_plugin_name, paginate=False, page=1, count=1000, created_on_or_before=None)`

Retrieves all valid (not expired & not revoked) certificates within Lemur, for the given authority plugin names ignored if no authority_plugin_name provided.

Note that depending on the DB size retrieving all certificates might an expensive operation :param paginate: option to use pagination, for large number of certificates. default to false :param page: the page to turn. default to 1 :param count: number of return certificates per page. default 1000 :param created_on_or_before: optional Arrow date to select only certificates issued on or before the date

Returns
list of certificates to check for revocation

`lemur.certificates.service.get_by_attributes(conditions)`

Retrieves certificate(s) by conditions given in a hash of given key=>value pairs. :param serial: :return:

`lemur.certificates.service.get_by_name(name)`

Retrieves certificate by its Name.

Parameters
name –

Returns

`lemur.certificates.service.get_by_serial(serial)`

Retrieves certificate(s) by serial number. :param serial: :return:

`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_certificates_for_expiration_metrics(expiry_window)`

Parameters
expiry_window – defines the window for cert filter, ex: 90 will only return certs expiring in the next 90 days.

Returns
list of certificates

`lemur.certificates.service.get_certificates_with_same_prefix_with_rotate_on(prefix)`

Find certificates with given prefix that are still valid, not replaced and marked for auto-rotate

Parameters
prefix – prefix to match

Returns

`lemur.certificates.service.get_certs_for_expiring_deployed_cert_check(exclude_domains, exclude_owners)`

`lemur.certificates.service.get_expiring_deployed_certificates(exclude=None)`

Finds all certificates that are eligible for deployed expiring cert notifications. Returns the set of domain/port pairs at which each certificate was identified as in use (deployed).

Sample response:

defaultdict(<class 'list'>,

{'testowner2@example.com': [(Certificate(name=certificate100),
defaultdict(<class 'list'>, {'localhost': [65521, 65522, 65523]}))],

'testowner3@example.com': [(Certificate(name=certificate101),
defaultdict(<class 'list'>, {'localhost': [65521, 65522, 65523]}))]})

Returns
A dictionary with owner as key, and a list of certificates associated with domains/ports.

`lemur.certificates.service.get_issued_cert_count_for_authority(authority)`

Returns the count of certs issued by the specified authority.

Returns

`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.identify_and_persist_expiring_deployed_certificates(exclude_domains, exclude_owners, commit, timeout_seconds_per_network_call=`

Finds all certificates expiring soon but are still being used for TLS at any domain with which they are associated. Identified ports will then be persisted on the certificate_associations row for the given cert/domain combo.

Note that this makes actual TLS network calls in order to establish the "deployed" part of this check.

`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.is_attached_to_endpoint(certificate_name, endpoint_name)`

Find if given certificate is attached to the endpoint. Both, certificate and endpoint, are identified by name. This method talks to elb and finds the real time information. :param certificate_name: :param endpoint_name: :return: True if certificate is attached to the given endpoint, False otherwise

`lemur.certificates.service.like_domain_query(term)`

`lemur.certificates.service.list_duplicate_certs_by_authority(authority_ids, days_since_issuance)`

Find duplicate certificates issued by given authorities that are still valid, not replaced, have auto-rotation ON, with names that are forced to be unique using serial number like 'some.name.prefix-YYYYMMDD-YYYYMMDD-serialnumber', thus the pattern "%-[0-9]{8}-[0-9]{8}-%" :param authority_ids: :param days_since_issuance: If not none, include certificates issued in only last days_since_issuance days :return: List of certificates matching criteria

`lemur.certificates.service.mint(**kwargs)`

Minting is slightly different for each authority. Support for multiple authorities is handled by individual plugins.

`lemur.certificates.service.query_common_name(common_name, args)`

Helper function that queries for not expired certificates by common name (and owner)

Parameters

- **common_name** –
- **args** –

Returns

`lemur.certificates.service.query_name(certificate_name, args)`

Helper function that queries for a certificate by name

Parameters

args –

Returns

`lemur.certificates.service.reissue_certificate(certificate, notify=None, replace=None, user=None)`

Reissue certificate with the same properties of the given certificate. :param certificate: :param notify: :param replace: :param user: :return:

`lemur.certificates.service.remove_destination_association(certificate, destination, clean=True)`

`lemur.certificates.service.remove_from_destination(certificate, destination)`

Remove the certificate from given destination if clean() is implemented :param certificate: :param destination: :return:

`lemur.certificates.service.remove_source_association(certificate, source)`

`lemur.certificates.service.render(args)`

Helper function that allows use to render our REST Api.

Parameters

args –

Returns

`lemur.certificates.service.revoke(certificate, reason)`

`lemur.certificates.service.send_certificate_expiration_metrics(expiry_window=None)`

Iterate over each certificate and emit a metric for how many days until expiration.

Parameters

expiry_window – defines the window for cert filter, ex: 90 will only return certs expiring in the next 90 days.

`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.update_owner(cert, new_cert_data)`

Modify owner for certificate. Removes roles and notifications associated with prior owner. :param cert: Certificate object to be updated :param new_cert_data: Dictionary including cert fields to be updated (owner, notifications, roles). These values are set in CertificateEditInputSchema and are generated for the new owner. :return:

`lemur.certificates.service.update_switches(cert, notify_flag=None, rotation_flag=None)`

Toggle notification and/or rotation values which are boolean :param notify_flag: new notify value :param rotation_flag: new rotation value :param cert: Certificate object to be updated :return:

`lemur.certificates.service.upload(**kwargs)`

Allows for pre-made certificates to be imported into Lemur.

`verify` Module

`lemur.certificates.verify.crl_verify(cert, cert_path)`

Attempts to verify a certificate using CRL.

Parameters

- **cert** –
- **cert_path** –

Returns
True if certificate is valid, False otherwise

Raises
Exception – If certificate does not have CRL

`lemur.certificates.verify.ocsp_verify(cert, 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 has been revoked

Parameters

- **cert** –
- **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.CertificateDeactivate`

Bases: `AuthenticatedResource`

endpoint = 'deactivateCertificate'

mediatypes()

methods = {'PUT'}

A list of methods this view can handle.

put(*certificate_id*)

PUT /certificates/1/deactivate

deactivate a certificate (integration test only) **Example request**:

```
PUT /certificates/1/deactivate HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8
```

Example response:

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

{
  "id": 1
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated or cert attached to LB
- 400 Bad Request – encountered error, more details in error message

class lemur.certificates.views.**CertificateExport**

Bases: AuthenticatedResource

endpoint = 'exportCertificate'

mediatypes()

methods = {'POST'}

A list of methods this view can handle.

post(*certificate_id*, *data=None*)

POST /certificates/1/export

Export a certificate

Example request:

```
PUT /certificates/1/export HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "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"
      }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

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: 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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.certificates.views.**CertificateRevoke**

Bases: AuthenticatedResource

endpoint = 'revokeCertificate'

mediatypes()

methods = {'PUT'}

A list of methods this view can handle.

put(*certificate_id*, *data=None*)

PUT /certificates/1/revoke

Revoke a certificate. One can mention the reason of revocation using crlReason (optional) as per RFC 5280 section 5.3.1 The allowed values for crlReason can also be found in Lemur in constants.py/CRLReason Additional information can be captured using comments (optional).

Example request:

```
PUT /certificates/1/revoke HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "crlReason": "affiliationChanged",
  "comments": "Additional details if any"
}
```

Example response:

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

{
  "id": 1
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated or cert attached to LB
- 400 Bad Request – encountered error, more details in error message

class lemur.certificates.views.**CertificateUpdateOwner**

Bases: AuthenticatedResource

endpoint = 'certificateUpdateOwner'

mediatypes()

methods = {'POST'}

A list of methods this view can handle.

post(*certificate_id*, *data=None*)

POST /certificates/1/update/owner

Update certificate owner

Example request:

```
POST /certificates/1/update/owner HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "owner": "joan@example.com"
}
```

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",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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,
  "notify": false,
  "rotation": false,
  "notifications": [{
      "id": 1
  }]
  "signingAlgorithm": "sha256",
```

(continues on next page)

(continued from previous page)

```
  "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"
  }],
  "rotation": true,
  "rotationPolicy": {"name": "default"},
  "san": null
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

class lemur.certificates.views.**Certificates**

Bases: AuthenticatedResource

delete(*certificate_id*, *data=None*)

DELETE /certificates/1

Delete a certificate

Example request:

```
DELETE /certificates/1 HTTP/1.1
Host: example.com
```

Example response:

```
HTTP/1.1 204 OK
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 204 No Content – no error
- 403 Forbidden – unauthenticated
- 404 Not Found – certificate not found
- 405 Method Not Allowed – certificate deletion is disabled

endpoint = 'certificateUpdateSwitches'

get(*certificate_id*)

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": "",
  "csr": "-----BEGIN CERTIFICATE REQUEST-----"
  "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",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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,
```

(continues on next page)

(continued from previous page)

```
  "domains": [{
      "sensitive": false,
      "id": 1090,
      "name": "*.test.example.net"
  }],
  "rotation": true,
  "rotationPolicy": {"name": "default"},
  "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()

methods = {'DELETE', 'GET', 'POST', 'PUT'}

A list of methods this view can handle.

post(*certificate_id*, *data=None*)

POST /certificates/1/update/switches

Update certificate boolean switches for notification or rotation

Example request:

```
POST /certificates/1/update/switches HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "notify": false,
   "rotation": false
}
```

Example response:

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

{
  "status": null,
```

(continues on next page)

(continued from previous page)


```
  "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",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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,
  "notify": false,
  "rotation": false,
  "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"
  }],
  "rotation": true,
  "rotationPolicy": {"name": "default"},
  "san": null
}
```


Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

put(*certificate_id*, *data=None*)

PUT /certificates/1

Update a certificate

Example request:

```
PUT /certificates/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "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",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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": [{
```

(continues on next page)

(continued from previous page)


```
        "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"
    }],
    "rotation": true,
    "rotationPolicy": {"name": "default"},
    "san": null
}
```


Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

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

Bases: AuthenticatedResource

Defines the 'certificates' endpoint

endpoint = 'certificates'

get()

GET /certificates

The current list of certificates. This API supports additional params like

Pagination, sorting:

/certificates?count=10&page=1&short=true&sortBy=id&sortDir=desc

Filters, mentioned as url param filter=field;value

/certificates?filter=cn;lemur.test.com /certificates?filter=notify;true /certificates?filter=rotation;true /certificates?filter=name;lemur.test.cert /certificates?filter=issuer;Digicert

Request expired certs

/certificates?showExpired=1

Search by Serial Number

Decimal: /certificates?serial=218243997808053074560741989466015229225

Hex: /certificates?serial=0xA43043DAB7F6F8AE115E94854EEB6529 /certificates?serial=a4:30:43:da:b7:f6:f8:ae:11:5e:94:85:4e:eb:65:29

Example request:

```
GET /certificates?serial=82311058732025924142789179368889309156 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": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----"
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
      "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,
```

(continues on next page)

(continued from previous page)

```
            "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /certificates

Creates a new certificate

Example request:

```
POST /certificates HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
    "owner": "secure@example.net",
    "commonName": "test.example.net",
    "country": "US",
    "extensions": {
      "subAltNames": {
        "names": [
          {
```

(continues on next page)

(continued from previous page)

```
            "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": "82311058732025924142789179368889309156",
  "id": 2288,
  "issuer": "SymantecCorporation",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "notBefore": "2016-06-03T00:00:00+00:00",
  "notAfter": "2018-01-12T23:59:59+00:00",
  "destinations": [],
  "bits": 2048,
  "body": "-----BEGIN CERTIFICATE-----...",
```

(continues on next page)

(continued from previous page)


```
  "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
  }],
  "rotation": true,
  "rotationPolicy": {"name": "default"},
  "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.**CertificatesListValid**

Bases: AuthenticatedResource

Defines the 'certificates/valid' endpoint

endpoint = 'certificatesListValid'

get()

GET /certificates/valid/<query>

The current list of not-expired certificates for a given common name, and owner. The API offers optional pagination. One can send page number(>=1) and desired count per page. The returned data contains total number of certificates which can help in determining the last page. Pagination will not be offered if page or count info is not sent or if it is zero.

Example request:

```
GET /certificates/valid?filter=cn;*.test.example.net&owner=joe@example.com&
↪page=1&count=20 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response (with single cert to be concise):

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

{
  "items": [{
      "status": null,
      "cn": "*.test.example.net",
      "chain": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----"
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
      "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": [],
```

(continues on next page)

(continued from previous page)

```
      "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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.certificates.views.**CertificatesNameQuery**

Bases: AuthenticatedResource

Defines the 'certificates/name' endpoint

endpoint = 'certificatesNameQuery'

get(*certificate_name*)

GET /certificates/name/<query>

The current list of certificates

Example request:

```
GET /certificates/name/WILDCARD.test.example.net-SymantecCorporation-
↪20160603-20180112 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": "",
```

(continues on next page)

(continued from previous page)


```
      "csr": "-----BEGIN CERTIFICATE REQUEST-----"
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
      "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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

endpoint = 'replacements'

get(*certificate_id*)

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": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----",
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
```

(continues on next page)

(continued from previous page)


```
      "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": [],
      "rotation": true,
      "rotationPolicy": {"name": "default"},
      "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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

Defines the 'certificates' stats endpoint

endpoint = 'certificateStats'

get()

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

Defines the 'certificates' upload endpoint

endpoint = 'certificateUpload'

mediatypes()

methods = {'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /certificates/upload

Upload a certificate

Example request:

```
POST /certificates/upload HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "owner": "joe@example.com",
   "body": "-----BEGIN CERTIFICATE-----...",
   "chain": "-----BEGIN CERTIFICATE-----...",
   "privateKey": "-----BEGIN RSA PRIVATE KEY-----..."
   "csr": "-----BEGIN CERTIFICATE REQUEST-----..."
   "destinations": [],
   "notifications": [],
   "replacements": [],
   "roles": [],
   "notify": true,
   "name": "cert1"
}
```

Example response:

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

{
  "status": null,
```

(continues on next page)

(continued from previous page)


```
  "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",
  "dateCreated": "2016-06-03T06:09:42.133769+00:00",
  "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": [],
  "rotation": true,
  "rotationPolicy": {"name": "default"},
  "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

- 403 Forbidden – unauthenticated

- 200 OK – no error

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

Bases: AuthenticatedResource

Defines the 'certificates' endpoint

endpoint = 'notificationCertificates'

get(*notification_id*)

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": "",
      "csr": "-----BEGIN CERTIFICATE REQUEST-----"
      "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",
      "dateCreated": "2016-06-03T06:09:42.133769+00:00",
      "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",
```

(continues on next page)

(continued from previous page)

```
        "user": {
            "username": "jane",
            "active": true,
            "email": "jane@example.com",
            "id": 2
        },
        "active": true,
        "domains": [{
            "sensitive": false,
            "id": 1090,
            "name": "*.test.example.net"
        }],
        "replaces": [],
        "replaced": [],
        "rotation": true,
        "rotationPolicy": {"name": "default"},
        "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()

methods = {'GET'}

A list of methods this view can handle.

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.**base64decode**(*base64_input*)

lemur.common.utils.**base64encode**(*string*)

lemur.common.utils.**check_cert_signature**(*cert*, *issuer_public_key*)

Check a certificate's signature against an issuer public key. Before EC validation, make sure we support the algorithm, otherwise raise UnsupportedAlgorithm On success, returns None; on failure, raises UnsupportedAlgorithm or InvalidSignature.

lemur.common.utils.**check_validation**(*validation*)

Checks that the given validation string compiles successfully.

Parameters
validation –

Return str
The validation pattern, if compilation succeeds

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.`**`convert_pkcs7_bytes_to_pem`**(*certs_pkcs7*)

Given a list of certificates in pkcs7 encoding (bytes), covert them into a list of PEM encoded files :raises ValueError or ValidationError :param certs_pkcs7: :return: list of certs in PEM format

`lemur.common.utils.`**`data_decrypt`**(*ciphertext*)

takes a ciphertext and returns the respective string reusing the Vault DB encryption module :param ciphertext: base64 ciphertext :return: plaintext string

`lemur.common.utils.`**`data_encrypt`**(*data*)

takes an input and returns a base64 encoded encryption reusing the Vault DB encryption module :param data: string :return: base64 ciphertext

`lemur.common.utils.`**`drop_last_cert_from_chain`**(*full_chain: str*) → str

drops the last certificate from a certificate chai, if more than one CA/subCA in the chain :param full_chain: string of a certificate chain :return: string of a new certificate chain, omitting the last certificate

`lemur.common.utils.`**`find_matching_certificates_by_hash`**(*cert*, *matching_certs*)

Given a Cryptography-formatted certificate cert, and Lemur-formatted certificates (matching_certs), determine if any of the certificate hashes match and return the matches.

`lemur.common.utils.`**`generate_private_key`**(*key_type*)

Generates a new private key based on key_type.

Valid key types: RSA2048, RSA4096', 'ECCPRIME192V1', 'ECCPRIME256V1', 'ECCSECP192R1', 'ECCSECP224R1', 'ECCSECP256R1', 'ECCSECP384R1', 'ECCSECP521R1', 'ECCSECP256K1', 'ECCSECT163K1', 'ECCSECT233K1', 'ECCSECT283K1', 'ECCSECT409K1', 'ECCSECT571K1', 'ECCSECT163R2', 'ECCSECT233R1', 'ECCSECT283R1', 'ECCSECT409R1', 'ECCSECT571R2'

Parameters
`key_type` –

Returns

`lemur.common.utils.`**`get_authority_key`**(*body*)

Returns the authority key for a given certificate in hex format

`lemur.common.utils.`**`get_certificate_via_tls`**(*host*, *port*, *timeout=10*)

Makes a TLS network connection to retrieve the current certificate for the specified host and port.

Note that if the host is valid but the port is not, we'll wait for the timeout for the connection to fail, so this should remain low when doing bulk operations.

Parameters

- **`host`** – Host to get certificate for
- **`port`** – Port to get certificate for
- **`timeout`** – Timeout in seconds

`lemur.common.utils.`**`get_key_type_from_certificate`**(*body*)

Helper function to determine key type by pasrding given PEM certificate

Parameters
`body` – PEM string

Returns
Key type string

`lemur.common.utils.get_key_type_from_ec_curve(curve_name)`

Give an EC curve name, return the matching key_type.

Param
curve_name

Returns
key_type

`lemur.common.utils.get_psuedo_random_string()`

Create a random and strongish challenge.

`lemur.common.utils.get_random_secret(length)`

Similar to get_pseudo_random_string, but accepts a length parameter.

`lemur.common.utils.get_state_token_secret()`

`lemur.common.utils.is_json(json_input)`

Test if input is json :param json_input: :return: True or False

`lemur.common.utils.is_selfsigned(cert)`

Returns True if the certificate is self-signed. Returns False for failed verification or unsupported signing algorithm.

`lemur.common.utils.is_weekend(date)`

Determines if a given date is on a weekend.

Parameters
date –

Returns

`lemur.common.utils.parse_cert_chain(pem_chain)`

Helper function to split and parse a series of PEM certificates.

Parameters
pem_chain – string

Returns
List of parsed certificates

`lemur.common.utils.parse_certificate(body)`

Helper function that parses a PEM certificate.

Parameters
body –

Returns

`lemur.common.utils.parse_csr(csr)`

Helper function that parses a CSR.

Parameters
csr –

Returns

`lemur.common.utils.parse_private_key(private_key)`

Parses a PEM-format private key (RSA, DSA, ECDSA or any other supported algorithm).

Raises ValueError for an invalid string. Raises AssertionError when passed value is not str-type.

Parameters
private_key – String containing PEM private key

`lemur.common.utils.parse_serial(pem_certificate)`

Parses a serial number from a PEM-encoded certificate.

`lemur.common.utils.split_pem(data)`

Split a string of several PEM payloads to a list of strings.

Parameters
data – String

Returns
List of strings

`lemur.common.utils.truthiness(s)`

If input string resembles something truthy then return True, else False.

`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: `Model`

certificate

description

id

label

options

pending_cert

property plugin

plugin_name

validate_label(*key*, *label*)

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** –

Return type
Destination

Returns
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

Return type
Destination

Returns

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*, *plugin_name*, *options*, *description*)

Updates an existing destination.

Parameters

- **destination_id** – Lemur assigned ID

- **label** – Destination common name
- **plugin_name** –
- **options** –
- **description** –

Return type
Destination

Returns

views Module

class lemur.destinations.views.**CertificateDestinations**

Bases: AuthenticatedResource

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

endpoint = 'certificateDestinations'

get(*certificate_id*)

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",
```

(continues on next page)

(continued from previous page)

```
                "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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

delete(*destination_id*)

endpoint = 'destination'

get(*destination_id*)

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

{
```

(continues on next page)

(continued from previous page)

```
  "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()

methods = {'DELETE', 'GET', 'PUT'}

A list of methods this view can handle.

put(*destination_id*, *data=None*)

PUT /destinations/1

Updates an account

Example request:

```
POST /destinations/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8


{
  "description": "test33",
  "options": [{
      "name": "accountNumber",
      "required": true,
```

(continues on next page)

(continued from previous page)

```
        "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"
    },
```

(continues on next page)

(continued from previous page)

```
  "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: AuthenticatedResource

Defines the 'destinations' endpoint

endpoint = 'destinations'

get()

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!",
```

(continues on next page)

(continued from previous page)

```
                "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /destinations

Creates a new account

Example request:

```
POST /destinations HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "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",
```

(continues on next page)

(continued from previous page)

```
            "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: AuthenticatedResource

Defines the 'destinations' stats endpoint

endpoint = 'destinationStats'

get()

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

domains Package

models Module

class lemur.domains.models.**Domain**(***kwargs*)

Bases: 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.is_authorized_for_domain(name)`

If authorization plugin is available, perform the check to see if current user can issue certificate for a given domain. Raises UnauthorizedError if unauthorized. If authorization plugin is not available, it returns without performing any check

Parameters
name – domain (string) for which authorization check is being done

`lemur.domains.service.is_domain_sensitive(name)`

Return True if domain is marked sensitive

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: `AuthenticatedResource`

Defines the 'domains' endpoint

endpoint = 'certificateDomains'

get(*certificate_id*)

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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.domains.views.**Domains**

Bases: AuthenticatedResource

endpoint = 'domain'

get(*domain_id*)

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()

methods = {'GET', 'PUT'}

A list of methods this view can handle.

put(*domain_id*, *data=None*)

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: AuthenticatedResource

Defines the 'domains' endpoint

endpoint = 'domains'

get()

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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /domains

The current domain list

Example request:

```
POST /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: Model

active

certificate

certificates

description

id

label

options

pending_cert

pending_certificates

property 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** –

Return type
Notification

Returns

lemur.notifications.service.**create_default_expiration_notifications**(*name*, *recipients*, *intervals=None*)

Will create standard 30, 10 and 2 day notifications for a given owner unless an alternate set of intervals is supplied. 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

Return type
Notification

Returns

`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, plugin_name, options, description, active, added_certificates, removed_certificates)`

Updates an existing notification.

Parameters

- `notification_id` –
- `label` – Notification label
- `plugin_name` –
- `options` –
- `description` –
- `active` –
- `added_certificates` –
- `removed_certificates` –

Return type
Notification

Returns

`views` Module

`class lemur.notifications.views.CertificateNotifications`

Bases: `AuthenticatedResource`

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

`endpoint = 'certificateNotifications'`

`get(certificate_id)`

`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
```

(continues on next page)

(continued from previous page)

```
}
```

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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

delete(*notification_id*)

endpoint = 'notification'

get(*notification_id*)

GET /notifications/1

Get a specific notification

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"
      },
      {
```

(continues on next page)

(continued from previous page)

```
            "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()

methods = {'DELETE', 'GET', 'PUT'}

A list of methods this view can handle.

put(*notification_id*, *data=None*)

PUT /notifications/1

Updates a notification

Example request:

```
PUT /notifications/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "label": "labelChanged",
  "plugin": {
      "slug": "email-notification",
      "plugin_options": "???"
```

(continues on next page)

(continued from previous page)

```
    },
  "description": "Sample notification",
  "active": "true",
  "added_certificates": "???",
  "removed_certificates": "???"
}
```

Example response:

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

{
  "id": 1,
  "label": "labelChanged",
  "plugin": {
      "slug": "email-notification",
      "plugin_options": "???"
    },
  "description": "Sample notification",
  "active": "true",
  "added_certificates": "???",
  "removed_certificates": "???"
}
```

Label label
notification name

Label slug
notification plugin slug

Label plugin_options
notification plugin options

Label description
notification description

Label active
whether or not the notification is active/enabled

Label added_certificates
certificates to add

Label removed_certificates
certificates to remove

Request Headers
- Authorization – OAuth token to authenticate

Status Codes
- 200 OK – no error

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

Bases: `AuthenticatedResource`

Defines the 'notifications' endpoint

endpoint = 'notifications'

get()

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,
```

(continues on next page)

(continued from previous page)

```
            "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /notifications

Creates a new notification

Example request:

```
POST /notifications HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "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",
```

(continues on next page)

(continued from previous page)

```
            "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,
```

(continues on next page)

(continued from previous page)

```
            "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
}
```

Label label
notification name

Label slug
notification plugin slug

Label plugin_options
notification plugin options

Label description
notification description

Label active
whether or not the notification is active/enabled

Label certificates
certificates to attach to notification

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: AuthenticatedResource

Defines the 'plugins' endpoint

endpoint = 'pluginName'

get(*name*)

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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.plugins.views.**PluginsList**

Bases: AuthenticatedResource

Defines the 'plugins' endpoint

endpoint = 'plugins'

get()

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,
```

(continues on next page)

(continued from previous page)

```
        "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()

methods = {'GET'}

A list of methods this view can handle.

Subpackages

base Package

base Package

manager Module

class lemur.plugins.base.manager.**PluginManager**(*class_list=None*, *instances=True*)

Bases: InstanceManager

all(*version=1*, *plugin_type=None*)

Returns a list of cached instances.

first(*func_name*, **args*, ***kwargs*)

get(*slug*)

register(*cls*)

unregister(*cls*)

v1 Module

class lemur.plugins.base.v1.**IPlugin**

Bases: _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()

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', 'https://lemur.readthedocs.io'), >>> ('Bug Tracker', 'https://github.com/Netflix/lemur/issues'), >>> ('Source', 'https://github.com/Netflix/lemur'), >>>]

get_server_options(*name*)

get_title()

Returns the general title for this plugin. >>> plugin.get_title()

get_user_option(*name*, *options*)

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

validate_option_value(*option_name*, *value*)

version = None

class lemur.plugins.base.v1.**Plugin**

Bases: `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**(*name*, *bases*, *attrs*)

Bases: `type`

bases Package

bases Package

destination Module

class lemur.plugins.bases.destination.**DestinationPlugin**

Bases: Plugin

requires_key = True

sync_as_source = False

sync_as_source_name = ''

type = 'destination'

upload(*name*, *body*, *private_key*, *cert_chain*, *options*, ***kwargs*)

class lemur.plugins.bases.destination.**ExportDestinationPlugin**

Bases: DestinationPlugin

default_options = [{'name': 'exportPlugin', 'type': 'export-plugin', 'required': True, 'helpMessage': 'Export plugin to use before sending data to destination.'}]

export(*body*, *private_key*, *cert_chain*, *options*)

property options

Gets/sets options for the plugin.

Returns

upload(*name*, *body*, *private_key*, *cert_chain*, *options*, ***kwargs*)

issuer Module

class lemur.plugins.bases.issuer.**IssuerPlugin**

Bases: Plugin

This is the base class from which all of the supported issuers will inherit from.

cancel_ordered_certificate(*pending_cert*, ***kwargs*)

create_authority(*options*)

create_certificate(*csr*, *issuer_options*)

deactivate_certificate(*certificate*)

get_ordered_certificate(*certificate*)

revoke_certificate(*certificate*, *reason*)

type = 'issuer'

notification Module

class lemur.plugins.bases.notification.**ExpirationNotificationPlugin**

Bases: 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 = [{'name': 'interval', 'type': 'int', 'required': True, 'validation': '^\\d+$', 'helpMessage': 'Number of days to be alert before expiration.'}, {'name': 'unit', 'type': 'select', 'required': True, 'validation': '', 'available': ['days', 'weeks', 'months'], 'helpMessage': 'Interval unit'}]

property options

Gets/sets options for the plugin.

Returns

send(*notification_type*, *message*, *excluded_targets*, *options*, ***kwargs*)

class lemur.plugins.bases.notification.**NotificationPlugin**

Bases: Plugin

This is the base class from which all of the supported issuers will inherit from.

get_recipients(*options*, *additional_recipients*)

Given a set of options (which should include configured recipient info), returns the parsed list of recipients from those options plus the additional recipients specified. The returned value has no duplicates.

For any notification types where recipients can't be dynamically modified, this returns only the additional recipients.

send(*notification_type*, *message*, *targets*, *options*, ***kwargs*)

type = 'notification'

source Module

class lemur.plugins.bases.source.**SourcePlugin**

Bases: Plugin

clean(*certificate*, *options*, ***kwargs*)

default_options = [{'name': 'pollRate', 'type': 'int', 'required': False, 'helpMessage': 'Rate in seconds to poll source for new information.', 'default': '60'}]

get_certificates(*options*, ***kwargs*)

get_endpoints(*options*, ***kwargs*)

property options

Gets/sets options for the plugin.

Returns

type = 'source'

lemur_aws Package

lemur_aws Package

elb Module

lemur.plugins.lemur_aws.elb.**attach_certificate**(*name*, *port*, *certificate_id*, ***kwargs*)

Attaches a certificate to a listener, throws exception if certificate specified does not exist in a particular account.

Parameters

- **name** –
- **port** –
- **certificate_id** –

lemur.plugins.lemur_aws.elb.**attach_certificate_v2**(*listener_arn*, *port*, *certificates*, ***kwargs*)

Attaches a certificate to a listener, throws exception if certificate specified does not exist in a particular account.

Parameters

- **listener_arn** –
- **port** –
- **certificates** –

lemur.plugins.lemur_aws.elb.**describe_listeners_v2**(***kwargs*)

Fetches one page of listener objects for a given elb arn.

Parameters
kwargs –

Returns

lemur.plugins.lemur_aws.elb.**describe_load_balancer_policies**(*load_balancer_name*, *policy_names*, ***kwargs*)

Fetching all policies currently associated with an ELB.

Parameters
load_balancer_name –

Returns

lemur.plugins.lemur_aws.elb.**describe_load_balancer_types**(*policies*, ***kwargs*)

Describe the policies with policy details.

Parameters
policies –

Returns

lemur.plugins.lemur_aws.elb.**describe_ssl_policies_v2**(*policy_names*, ***kwargs*)

Fetching all policies currently associated with an ELB.

Parameters
policy_names –

Returns

`lemur.plugins.lemur_aws.elb.get_all_elbs(**kwargs)`

Fetches all elbs for a given account/region

Parameters
kwargs –

Returns

`lemur.plugins.lemur_aws.elb.get_all_elbs_v2(**kwargs)`

Fetches all elbs for a given account/region

Parameters
kwargs –

Returns

`lemur.plugins.lemur_aws.elb.get_elbs(**kwargs)`

Fetches one page elb objects for a given account and region.

`lemur.plugins.lemur_aws.elb.get_elbs_v2(**kwargs)`

Fetches one page of elb objects for a given account and region.

Parameters
kwargs –

Returns

`lemur.plugins.lemur_aws.elb.get_listener_arn_from_endpoint(endpoint_name, endpoint_port, **kwargs)`

Get a listener ARN from an endpoint. :param endpoint_name: :param endpoint_port: :return:

`lemur.plugins.lemur_aws.elb.get_load_balancer_arn_from_endpoint(endpoint_name, **kwargs)`

Get a load balancer ARN from an endpoint. :param endpoint_name: :return:

`lemur.plugins.lemur_aws.elb.is_valid(listener_tuple)`

There are a few rules that aws has when creating listeners, this function ensures those rules are met before we try and create or update a listener.

While these could be caught with boto exception handling, I would rather be nice and catch these early before we sent them out to aws. It also gives us an opportunity to create nice user warnings.

This validity check should also be checked in the frontend but must also be enforced by server.

Parameters
listener_tuple –

`lemur.plugins.lemur_aws.elb.retry_throttled(exception)`

Determines if this exception is due to throttling :param exception: :return:

`iam` Module

`lemur.plugins.lemur_aws.iam.create_arn_from_cert(account_number, partition, certificate_name, path='')`

Create an ARN from a certificate. :param path: :param account_number: :param partition: :param certificate_name: :return:

`lemur.plugins.lemur_aws.iam.delete_cert(cert_name, **kwargs)`

Delete a certificate from AWS

Parameters
cert_name –

Returns

`lemur.plugins.lemur_aws.iam.get_all_certificates(restrict_path=None, **kwargs)`

Use STS to fetch all of the SSL certificates from a given account :param restrict_path: If provided, only return certificates with a matching Path value.

`lemur.plugins.lemur_aws.iam.get_certificate(name, **kwargs)`

Retrieves an SSL certificate.

Returns

`lemur.plugins.lemur_aws.iam.get_certificate_id_to_name(**kwargs)`

Use STS to fetch a map of IAM certificate IDs to names

`lemur.plugins.lemur_aws.iam.get_certificates(**kwargs)`

Fetches one page of certificate objects for a given account. :param kwargs: :return:

`lemur.plugins.lemur_aws.iam.get_name_from_arn(arn)`

Extract the certificate name from an arn.

examples: 'arn:aws:iam::123456789012:server-certificate/example.com' –> 'example.com' 'arn:aws:iam::123456789012:server-certificate/cloudfront/example.com-cloudfront' –> 'example.com-cloudfront' 'arn:aws:acm:us-west-2:123456789012:certificate/example.com' –> 'example.com'

Parameters
arn – IAM TLS certificate arn

Returns
name of the certificate as uploaded to AWS

`lemur.plugins.lemur_aws.iam.get_path_from_arn(arn)`

Get the certificate path from the certificate arn.

examples: 'arn:aws:iam::123456789012:server-certificate/example.com' –> '' 'arn:aws:iam::123456789012:server-certificate/cloudfront/example.com-cloudfront' –> 'cloudfront' 'arn:aws:iam::123456789012:server-certificate/cloudfront/2/example.com-cloudfront' –> 'cloudfront/2' 'arn:aws:acm:us-west-2:123456789012:certificate/example.com' –> ''

Parameters
arn – IAM TLS certificate arn

Returns
empty or the certificate path without the certificate name

`lemur.plugins.lemur_aws.iam.get_registry_type_from_arn(arn)`

Get the registery type based on the arn.

examples: 'arn:aws:iam::123456789000:server-certificate/example.com' –> 'iam' 'arn:aws:iam::123456789000:server-certificate/cloudfront/example.com-cloudfront' –> 'iam' 'arn:aws:acm:us-west-2:123456789000:certificate/example.com' –> 'acm'

Parameters
arn – IAM TLS certificate arn

Returns
iam or acm or unkown

lemur.plugins.lemur_aws.iam.**retry_throttled**(*exception*)

Determines if this exception is due to throttling :param exception: :return:

lemur.plugins.lemur_aws.iam.**upload_cert**(*name*, *body*, *private_key*, *path*, *cert_chain=None*, ***kwargs*)

Upload a certificate to AWS

Parameters

- **name** –
- **body** –
- **private_key** –
- **cert_chain** –
- **path** –

Returns

plugin Module

class lemur.plugins.lemur_aws.plugin.**AWSDestinationPlugin**

Bases: DestinationPlugin

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur'

clean(*certificate*, *options*, ***kwargs*)

deploy(*elb_name*, *account*, *region*, *certificate*)

description = 'Allow the uploading of certificates to AWS IAM'

options = [{'name': 'accountNumber', 'type': 'str', 'required': True, 'validation': '[0-9]{12}', 'helpMessage': 'Must be a valid AWS account number!'}, {'name': 'path', 'type': 'str', 'validation': '^(?:|/|/\\S+/)$', 'default': '/', 'helpMessage': 'Path prefix for uploaded certificates.'}]

slug = 'aws-destination'

sync_as_source = True

sync_as_source_name = 'aws-source'

title = 'AWS'

upload(*name*, *body*, *private_key*, *cert_chain*, *options*, ***kwargs*)

version = 'unknown'

class lemur.plugins.lemur_aws.plugin.**AWSSourcePlugin**

Bases: SourcePlugin

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur'

clean(*certificate*, *options*, ***kwargs*)

description = 'Discovers all SSL certificates and ELB or Cloudfront endpoints in an AWS account'

get_certificate_by_name(*certificate_name*, *options*)

get_certificates(*options*, ***kwargs*)

get_distributions(*options*, ***kwargs*)

get_endpoint_certificate_names(*endpoint*)

get_endpoints(*options*, ***kwargs*)

get_load_balancers(*options*, ***kwargs*)

options = [{'name': 'accountNumber', 'type': 'str', 'required': True, 'validation': '^[0-9]{12,12}$', 'helpMessage': 'Must be a valid AWS account number!'}, {'name': 'regions', 'type': 'str', 'helpMessage': 'Comma separated list of regions to search in, if no region is specified we look in all regions.'}, {'name': 'path', 'type': 'str', 'validation': '^(?:|/|/\\S+/)$', 'default': '/', 'helpMessage': "Only discover certificates with this path prefix. Must begin and end with slash. For CloudFront sources, use '/cloudfront/'."}, {'name': 'endpointType', 'type': 'select', 'available': ['elb', 'cloudfront', 'none'], 'default': 'elb', 'helpMessage': 'Type of AWS endpoint to discover. Defaults to elb if not set.'}]

slug = 'aws-source'

title = 'AWS'

update_endpoint(*endpoint*, *certificate*)

version = 'unknown'

class lemur.plugins.lemur_aws.plugin.**S3DestinationPlugin**(**args*, ***kwargs*)

Bases: ExportDestinationPlugin

additional_options = [{'name': 'bucket', 'type': 'str', 'required': True, 'validation': '[0-9a-z.-]{3,63}', 'helpMessage': 'Must be a valid S3 bucket name!'}, {'name': 'accountNumber', 'type': 'str', 'required': True, 'validation': '[0-9]{12}', 'helpMessage': 'A valid AWS account number with permission to access S3'}, {'name': 'region', 'type': 'str', 'default': 'us-east-1', 'required': False, 'helpMessage': 'Region bucket exists', 'available': ['us-east-1', 'us-west-2', 'eu-west-1']}, {'name': 'encrypt', 'type': 'bool', 'required': False, 'helpMessage': 'Enable server side encryption', 'default': True}, {'name': 'prefix', 'type': 'str', 'required': False, 'validation': '^(?:[^/].*|)$', 'helpMessage': 'Must be a valid S3 object prefix!', 'default': ''}]

author = 'Mikhail Khodorovskiy, Harm Weites <harm@weites.com>'

author_url = 'https://github.com/Netflix/lemur'

clean(*certificate*, *options*, ***kwargs*)

delete_acme_token(*token_path*, *options*, ***kwargs*)

description = 'Allow the uploading of certificates to Amazon S3'

slug = 'aws-s3'

title = 'AWS-S3'

upload(*name*, *body*, *private_key*, *chain*, *options*, ***kwargs*)

upload_acme_token(*token_path*, *token*, *options*, ***kwargs*)

This is called from the acme http challenge

Parameters

- **self** –
- **token_path** –
- **token** –
- **options** –
- **kwargs** –

Returns

class lemur.plugins.lemur_aws.plugin.**SNSNotificationPlugin**

Bases: ExpirationNotificationPlugin

additional_options = [{'name': 'accountNumber', 'type': 'str', 'required': True, 'validation': '[0-9]{12}', 'helpMessage': 'A valid AWS account number with permission to access the SNS topic'}, {'name': 'region', 'type': 'str', 'required': True, 'validation': '[0-9a-z\\-]{1,25}', 'helpMessage': 'Region in which the SNS topic is located, e.g. "us-east-1"'}, {'name': 'topicName', 'type': 'str', 'required': True, 'validation': '^[a-zA-Z0-9_\\-]{1,256}$', 'helpMessage': 'The name of the topic to use for expiration notifications'}]

author = 'Jasmine Schladen <jschladen@netflix.com>'

author_url = 'https://github.com/Netflix/lemur'

description = 'Sends notifications to AWS SNS'

send(*notification_type*, *message*, *excluded_targets*, *options*, ***kwargs*)

While we receive a *targets* parameter here, it is unused, as the SNS topic is pre-configured in the plugin configuration, and can't reasonably be changed dynamically.

slug = 'aws-sns'

title = 'AWS SNS'

version = 'unknown'

lemur.plugins.lemur_aws.plugin.**format_elb_cipher_policy**(*policy*)

Attempts to format cipher policy information into a common format. :param policy: :return:

lemur.plugins.lemur_aws.plugin.**format_elb_cipher_policy_v2**(*policy*)

Attempts to format cipher policy information for elbv2 into a common format. :param policy: :return:

lemur.plugins.lemur_aws.plugin.**get_distribution_endpoint**(*account_number*, *cert_id_to_name*, *distrib_dict*)

Constructs endpoint data from a distribution response, or None if it does not represent a distribution Lemur cares about. :param account_number: :param cert_id_to_name: map of IAM certificate IDs to names :param distrib_dict: :return: a list of endpoint dictionaries

lemur.plugins.lemur_aws.plugin.**get_elb_endpoints**(*account_number*, *region*, *elb_dict*)

Retrieves endpoint information from elb response data. :param account_number: :param region: :param elb_dict: :return:

lemur.plugins.lemur_aws.plugin.**get_elb_endpoints_v2**(*account_number*, *region*, *elb_dict*)

Retrieves endpoint information from elbv2 response data. :param account_number: :param region: :param elb_dict: :return:

lemur.plugins.lemur_aws.plugin.**get_region_from_dns**(*dns*)

sts Module

lemur.plugins.lemur_aws.sts.**sts_client**(*service*, *service_type='client'*)

lemur_cfssl Package

lemur_cfssl Package

plugin Module

class lemur.plugins.lemur_cfssl.plugin.**CfsslIssuerPlugin**(**args*, ***kwargs*)

Bases: 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'

revoke_certificate(*certificate*, *reason*)

Revoke a CFSSL certificate.

slug = 'cfssl-issuer'

title = 'CFSSL'

version = 'unknown'

lemur_email Package

lemur_email Package

plugin Module

class lemur.plugins.lemur_email.plugin.**EmailNotificationPlugin**(**args*, ***kwargs*)

Bases: ExpirationNotificationPlugin

additional_options = [{'name': 'recipients', 'type': 'str', 'required': True, 'validation': '([-!#-\'*+/-9=?A-Z^-~]+(\\.[-!#-\'*+/-9=?A-Z^-~]+)*|\\"([]!#-[^-~ \\t]|(\\\\[\\t -~]))+\\")@([-!#-\'*+/-9=?A-Z^-~]+(\\.[-!#-\'*+/-9=?A-Z^-~]+)*|\\[[\\t -Z^-~]*])', 'helpMessage': 'Comma delimited list of email addresses'}]

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur'

description = 'Sends expiration email notifications'

static get_recipients(*options*, *additional_recipients*, ***kwargs*)

Given a set of options (which should include configured recipient info), returns the parsed list of recipients from those options plus the additional recipients specified. The returned value has no duplicates.

For any notification types where recipients can't be dynamically modified, this returns only the additional recipients.

static send(*notification_type*, *message*, *targets*, *options*, ***kwargs*)

slug = 'email-notification'

title = 'Email'

version = 'unknown'

class lemur.plugins.lemur_email.plugin.**TitleParser**

Bases: HTMLParser

handle_data(*data*)

handle_starttag(*tag*, *attributes*)

lemur.plugins.lemur_email.plugin.**render_html**(*template_name*, *options*, *certificates*)

Renders the html for our email notification.

Parameters

- **template_name** –
- **options** –
- **certificates** –

Returns

lemur.plugins.lemur_email.plugin.**send_via_ses**(*subject*, *body*, *targets*, ***kwargs*)

Attempts to deliver email notification via SES service. :param subject: :param body: :param targets: :return:

`lemur.plugins.lemur_email.plugin.send_via_smtp(subject, body, targets)`

Attempts to deliver email notification via SMTP.

Parameters

- **subject** –
- **body** –
- **targets** –

Returns

Subpackages

templates Package

config Module

`lemur.plugins.lemur_email.templates.config.human_time(time)`

`lemur.plugins.lemur_email.templates.config.interval(options)`

`lemur.plugins.lemur_email.templates.config.unit(options)`

lemur_verisign Package

lemur_verisign Package

plugin Module

class `lemur.plugins.lemur_verisign.plugin.VerisignIssuerPlugin(*args, **kwargs)`

Bases: `IssuerPlugin`

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur.git'

clear_pending_certificates()

Uses Verisign to clear the pending certificates awaiting approval.

Returns

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

slug = 'verisign-issuer'

title = 'Verisign'

version = 'unknown'

class lemur.plugins.lemur_verisign.plugin.**VerisignSourcePlugin**(**args*, ***kwargs*)

Bases: 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: `Model`

authorities

authority

authority_id

certificate

certificates

description

id

name

password

pending_cert

pending_certificates

sensitive_fields = ('password',)

third_party

user

user_id

username

users

service Module

lemur.roles.service.**create**(*name*, *password=None*, *description=None*, *username=None*, *users=None*, *third_party=False*)

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.**get_or_create**(*role_name*, *description*)

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.**set_third_party**(*role_id*, *third_party_status=False*)

Sets a role to be a third party role. A user should pretty much never call this directly.

Parameters

- **role_id** –
- **third_party_status** –

Returns

lemur.roles.service.**update**(*role_id*, *name*, *description*, *users*)

Update a role

Parameters

- **role_id** –
- **name** –
- **description** –
- **users** –

Returns

lemur.roles.service.**warn_user_updates**(*role_name*, *current_users*, *new_users*)

views Module

class lemur.roles.views.**AuthorityRolesList**

Bases: AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'authorityRoles'

get(*authority_id*)

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",
```

(continues on next page)

(continued from previous page)

```
        "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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.roles.views.**RoleViewCredentials**

Bases: 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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

delete(*role_id*)

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(*role_id*)

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()

methods = {'DELETE', 'GET', 'PUT'}

A list of methods this view can handle.

put(*role_id*, *data=None*)

PUT /roles/1

Update a role

Example request:

```
PUT /roles/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "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: AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'roles'

get()

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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /roles

Creates a new role

Example request:

```
POST /roles HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "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: AuthenticatedResource

Defines the 'roles' endpoint

endpoint = 'userRoles'

get(*user_id*)

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()

methods = {'GET'}

A list of methods this view can handle.

users Package

models Module

class lemur.users.models.**User**(***kwargs*)

Bases: 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

property is_admin

Determine if the current user has the 'admin' role associated with it.

Returns

property is_admin_or_global_cert_issuer

Determine if the current user is a global cert issuer. The user has either 'admin' or 'global_cert_issuer' role associated with them.

Returns

keys

logs

password

pending_certificates

profile_picture

role

roles

sensitive_fields = ('password',)

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: `AuthenticatedResource`

endpoint = 'certificateCreator'

get(*certificate_id*)

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
```

(continues on next page)

(continued from previous page)

```
{
    "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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.users.views.**Me**

Bases: AuthenticatedResource

endpoint = 'me'

get()

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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.users.views.**RoleUsers**

Bases: AuthenticatedResource

endpoint = 'roleUsers'

get(*role_id*)

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()

methods = {'GET'}

A list of methods this view can handle.

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

Bases: AuthenticatedResource

endpoint = 'user'

get(*user_id*)

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()

methods = {'GET', 'PUT'}

A list of methods this view can handle.

put(*user_id*, *data=None*)

PUT /users/1

Update a user

Example request with ID:

```
PUT /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

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

(continues on next page)

(continued from previous page)

```
    "roles": [
        {"id": 1}
    ]
}
```

Example request with name:

```
PUT /users/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
    "username": "user1",
    "email": "user1@example.com",
    "active": false,
    "roles": [
        {"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: `AuthenticatedResource`

Defines the 'users' endpoint

endpoint = 'users'

get()

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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /users

Creates a new user

Example request with ID:

```
POST /users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

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

Example request with name:

```
POST /users HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
   "username": "user3",
   "email": "user3@example.com",
   "active": true,
   "roles": [
      {"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

sources Package

sources Module

cli Module

lemur.sources.cli.**clean**(*source_strings*, *commit*)

lemur.sources.cli.**clean_unused_and_expiring_within_days**(*source_strings*, *days_to_expire*, *commit*)

lemur.sources.cli.**clean_unused_and_issued_since_days**(*source_strings*, *days_since_issuance*, *commit*)

lemur.sources.cli.**enable_cloudfront**(*source_label*)

Given the label of a legacy AWS source (without path or endpointType options), set up the source for CloudFront:

1. Update the source options to the newest template, inheriting the existing values.
2. Set `path` to "/" and `endpointType` to "elb" to restrict the source to discovering ELBs and related certs only.
3. Create a new source (and destination) for the same accountNumber with `path` as "/cloudfront/" and `endpointType` as "cloudfront"

Parameters
source_strings –

Returns

lemur.sources.cli.**execute_clean**(*plugin*, *certificate*, *source*)

lemur.sources.cli.**sync**(*source_strings*, *ttl*)

lemur.sources.cli.**sync_source_destination**(*labels*)

This command will sync destination and source, to make sure eligible destinations are also present as source. Destination eligibility is determined on the sync_as_source attribute of the plugin. The destination sync_as_source_name provides the name of the suitable source-plugin. We use (account number, IAM path) tuple uniqueness to avoid duplicate sources.

Lemur now does this automatically during destination create and update, so this command is primarily useful for migrating legacy destinations. Set "-d all" to sync all destinations.

lemur.sources.cli.**validate_destinations**(*destination_strings*)

lemur.sources.cli.**validate_sources**(*source_strings*)

models Module

class lemur.sources.models.**Source**(***kwargs*)

Bases: `Model`

active

certificate

description

endpoints

id

label

last_run

options

pending_cert

property plugin

plugin_name

schemas Module

class lemur.sources.schemas.**SourceInputSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: LemurInputSchema

opts = <marshmallow.schema.SchemaOpts object>

class lemur.sources.schemas.**SourceOutputSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: LemurOutputSchema

fill_object(*data*)

opts = <marshmallow.schema.SchemaOpts object>

service Module

lemur.sources.service.**add_aws_destination_to_sources**(*dst*)

Given a destination, check if it can be added as sources, and include it if not already a source We identify qualified destinations based on the sync_as_source attributed of the plugin. The destination sync_as_source_name reveals the name of the suitable source-plugin. We rely on account numbers to avoid duplicates. :return: true for success and false for not adding the destination as source

lemur.sources.service.**certificate_create**(*certificate*, *source*)

lemur.sources.service.**certificate_update**(*certificate*, *source*)

lemur.sources.service.**create**(*label*, *plugin_name*, *options*, *description=None*)

Creates a new source, that can then be used as a source for certificates.

Parameters

- **label** – Source common name
- **plugin_name** –
- **options** –

- **description** –

Return type
Source

Returns
New source

`lemur.sources.service.delete(source_id)`

Deletes an source.

Parameters
source_id – Lemur assigned ID

`lemur.sources.service.expire_endpoints(source, ttl_hours)`

`lemur.sources.service.find_cert(certificate)`

`lemur.sources.service.get(source_id)`

Retrieves an source by its lemur assigned ID.

Parameters
source_id – Lemur assigned ID

Return type
Source

Returns

`lemur.sources.service.get_all()`

Retrieves all source currently known by Lemur.

Returns

`lemur.sources.service.get_by_label(label)`

Retrieves a source by its label

Parameters
label –

Returns

`lemur.sources.service.render(args)`

`lemur.sources.service.sync(source, user, ttl_hours=2)`

`lemur.sources.service.sync_certificates(source, user)`

`lemur.sources.service.sync_endpoints(source)`

`lemur.sources.service.sync_update_destination(certificate, source)`

`lemur.sources.service.update(source_id, label, plugin_name, options, description)`

Updates an existing source.

Parameters

- **source_id** – Lemur assigned ID
- **label** – Source common name
- **options** –
- **plugin_name** –

- **description** –

Return type
Source

Returns

views Module

class lemur.sources.views.**CertificateSources**

Bases: AuthenticatedResource

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

endpoint = 'certificateSources'

get(*certificate_id*)

GET /certificates/1/sources

The current account list for a given certificates

Example request:

```
GET /certificates/1/sources 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": [
      {
          "options": [
              {
                  "name": "accountNumber",
                  "required": true,
                  "value": 111111111112,
                  "helpMessage": "Must be a valid AWS account number!",
                  "validation": "^[0-9]{12,12}$",
                  "type": "int"
              }
          ],
          "pluginName": "aws-source",
          "id": 3,
          "lastRun": "2015-08-01T15:40:58",
          "description": "test",
          "label": "test"
      }
  ],
  "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()

methods = {'GET'}

A list of methods this view can handle.

class lemur.sources.views.**Sources**

Bases: AuthenticatedResource

delete(*source_id*)

endpoint = 'account'

get(*source_id*)

GET /sources/1

Get a specific account

Example request:

```
GET /sources/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

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
  ],
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
```

(continues on next page)

(continued from previous page)

```
  "label": "test"
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error

mediatypes()

methods = {'DELETE', 'GET', 'PUT'}

A list of methods this view can handle.

put(*source_id*, *data=None*)

PUT /sources/1

Updates an account

Example request:

```
POST /sources/1 HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
  ],
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
  "label": "test"
}
```

Example response:

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

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
```

(continues on next page)

(continued from previous page)

```
            "value": 111111111112,
            "helpMessage": "Must be a valid AWS account number!",
            "validation": "^[0-9]{12,12}$",
            "type": "int"
        }
    ],
    "pluginName": "aws-source",
    "id": 3,
    "lastRun": "2015-08-01T15:40:58",
    "description": "test",
    "label": "test"
}
```

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.sources.views.**SourcesList**

Bases: AuthenticatedResource

Defines the 'sources' endpoint

endpoint = 'sources'

get()

GET /sources

The current account list

Example request:

```
GET /sources 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": [
        {
            "options": [
                {
                    "name": "accountNumber",
                    "required": true,
                    "value": 111111111112,
```

(continues on next page)

(continued from previous page)

```
                    "helpMessage": "Must be a valid AWS account number!",
                    "validation": "^[0-9]{12,12}$",
                    "type": "int"
                }
            ],
            "pluginName": "aws-source",
            "lastRun": "2015-08-01T15:40:58",
            "id": 3,
            "description": "test",
            "label": "test"
        }
    ],
    "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()

methods = {'GET', 'POST'}

A list of methods this view can handle.

post(*data=None*)

POST /sources

Creates a new account

Example request:

```
POST /sources HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
Content-Type: application/json;charset=UTF-8

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
  ],
```

(continues on next page)

(continued from previous page)

```
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
  "label": "test"
}
```

Example response:

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

{
  "options": [
      {
          "name": "accountNumber",
          "required": true,
          "value": 111111111112,
          "helpMessage": "Must be a valid AWS account number!",
          "validation": "^[0-9]{12,12}$",
          "type": "int"
      }
  ],
  "pluginName": "aws-source",
  "id": 3,
  "lastRun": "2015-08-01T15:40:58",
  "description": "test",
  "label": "test"
}
```

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

logs Package

logs Module

models Module

class lemur.logs.models.**Log**(***kwargs*)

Bases: Model

certificate

certificate_id

id

log_type

logged_at

user

user_id

schemas Module

class lemur.logs.schemas.**LogOutputSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: LemurOutputSchema

opts = <marshmallow.schema.SchemaOpts object>

service Module

lemur.logs.service.**audit_log**(*action*, *entity*, *message*)

Logs given action :param action: The action being logged e.g. assign_role, create_role etc :param entity: The entity undergoing the action e.g. name of the role :param message: Additional info e.g. Role being assigned to user X :return:

lemur.logs.service.**create**(*user*, *type*, *certificate=None*)

Creates logs a given action.

Parameters

- **user** –
- **type** –
- **certificate** –

Returns

lemur.logs.service.**get_all**()

Retrieve all logs from the database.

Returns

lemur.logs.service.**render**(*args*)

Helper that paginates and filters data when requested through the REST Api

Parameters

args –

Returns

views Module

class lemur.logs.views.**LogsList**

Bases: AuthenticatedResource

Defines the 'logs' endpoint

endpoint = 'logs'

get()

GET /logs

The current log list

Example request:

```
GET /logs 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": [
    ]
  "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()

methods = {'GET'}

A list of methods this view can handle.

reporting Package

reporting Module

cli Module

lemur.reporting.cli.**expiring**(*ttl*, *deployment*)

Returns certificates expiring in the next n days.

lemur.reporting.cli.**fqdn**(*deployment*, *validity*)

Generates a report in order to determine the number of FQDNs covered by Lemur issued certificates.

service Module

lemur.reporting.service.**expiring_certificates**(***kwargs*)

Returns an Expiring report. :return:

lemur.reporting.service.**filter_by_deployment**(*query*, *deployment=None*)

lemur.reporting.service.**filter_by_issuer**(*query*, *issuer=None*)

lemur.reporting.service.**filter_by_owner**(*query*, *owner=None*)

lemur.reporting.service.**filter_by_validity**(*query*, *validity=None*)

lemur.reporting.service.**filter_by_validity_end**(*query*, *validity_end=None*)

lemur.reporting.service.**fqdns**(***kwargs*)

Returns an FQDN report. :return:

views Module

tests Package

tests Module

deployment Package

deployment Module

service Module

lemur.deployment.service.**rotate_certificate**(*endpoint*, *new_cert*)

Rotates a certificate on a given endpoint.

Parameters

- **endpoint** –
- **new_cert** –

Returns

endpoints Package

endpoints Module

models Module

class lemur.endpoints.models.**Cipher**(***kwargs*)

Bases: Model

deprecated

id

name

policy

class lemur.endpoints.models.**Endpoint**(***kwargs*)

Bases: Model

active

aliases

certificate

certificate_id

certificate_path

date_created

property dns_aliases

dnsname

id

property issues

last_updated

name

owner

policy

policy_id

port

registry_type

replaced = ObjectAssociationProxyInstance(AssociationProxy('certificate', 'replaced'))

sensitive

source

source_id

property source_label

type

class lemur.endpoints.models.**EndpointDnsAlias**(***kwargs*)

Bases: Model

alias

endpoint

endpoint_id

id

class lemur.endpoints.models.**Policy**(***kwargs*)

Bases: Model

ciphers

endpoint

id

name

schemas Module

class lemur.endpoints.schemas.**CipherNestedOutputSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: LemurOutputSchema

opts = <marshmallow.schema.SchemaOpts object>

class lemur.endpoints.schemas.**EndpointOutputSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: LemurOutputSchema

opts = <marshmallow.schema.SchemaOpts object>

class lemur.endpoints.schemas.**PolicyNestedOutputSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: LemurOutputSchema

opts = <marshmallow.schema.SchemaOpts object>

service Module

lemur.endpoints.service.**create**(***kwargs*)

Creates a new endpoint. :param kwargs: :return:

lemur.endpoints.service.**get**(*endpoint_id*)

Retrieves an endpoint given it's ID

Parameters
endpoint_id –

Returns

lemur.endpoints.service.**get_all**()

Get all endpoints that are currently in Lemur.

:rtype : List :return:

lemur.endpoints.service.**get_all_pending_rotation**()

Retrieves all endpoints which have certificates deployed that have been replaced. :return:

lemur.endpoints.service.**get_by_dnsname**(*dnsname*)

Retrieves an endpoint given it's name.

Parameters
dnsname –

Returns

lemur.endpoints.service.**get_by_dnsname_and_port**(*dnsname*, *port*)

Retrieves and endpoint by it's dnsname and port. :param dnsname: :param port: :return:

lemur.endpoints.service.**get_by_name**(*name*)

Retrieves an endpoint given it's name.

Parameters
name –

Returns

lemur.endpoints.service.**get_by_name_and_source**(*name*, *source*)

Retrieves an endpoint by it's name and source. :param name: :param source: :return:

lemur.endpoints.service.**get_by_source**(*source_label*)

Retrieves all endpoints for a given source. :param source_label: :return:

lemur.endpoints.service.**get_or_create_cipher**(***kwargs*)

lemur.endpoints.service.**get_or_create_policy**(***kwargs*)

lemur.endpoints.service.**render**(*args*)

Helper that helps us render the REST Api responses. :param args: :return:

lemur.endpoints.service.**stats**(***kwargs*)

Helper that defines some useful statistics about endpoints.

Parameters
kwargs –

Returns

lemur.endpoints.service.**update**(*endpoint_id*, ***kwargs*)

views Module

class lemur.endpoints.views.**Endpoints**

Bases: AuthenticatedResource

endpoint = 'endpoint'

get(*endpoint_id*)

GET /endpoints/1

One endpoint

Example request:

```
GET /endpoints/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
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

class lemur.endpoints.views.**EndpointsList**

Bases: AuthenticatedResource

Defines the 'endpoints' endpoint

endpoint = 'endpoints'

get()

GET /endpoints

The current list of endpoints

Example request:

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

Example response:

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

Query Parameters

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

methods = {'GET'}

A list of methods this view can handle.

defaults Package

defaults Module

schemas Module

class lemur.defaults.schemas.**DefaultOutputSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: LemurOutputSchema

opts = <marshmallow.schema.SchemaOpts object>

views Module

class lemur.defaults.views.**LemurDefaults**

Bases: AuthenticatedResource

Defines the 'defaults' endpoint

endpoint = 'default'

get()

GET /defaults

Returns defaults needed to generate CSRs

Example request:

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

Example response:

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

{
   "country": "US",
   "state": "CA",
   "location": "Los Gatos",
   "organization": "Netflix",
   "organizationalUnit": "Operations",
   "dnsProviders": [{"name": "test", ...}, {...}],
}
```

Request Headers

- Authorization – OAuth token to authenticate

Status Codes

- 200 OK – no error
- 403 Forbidden – unauthenticated

mediatypes()

methods = {'GET'}

A list of methods this view can handle.

lemur_acme package

lemur_acme Module

acme_handlers Module

class lemur.plugins.lemur_acme.acme_handlers.**AcmeDnsHandler**

Bases: AcmeHandler

autodetect_dns_providers(*domain*)

Get DNS providers associated with a domain when it has not been provided for certificate creation. :param domain: :return: dns_providers: List of DNS providers that have the correct zone.

cleanup_dns_challenges(*acme_client*, *authorizations*)

Best effort attempt to delete DNS challenges that may not have been deleted previously. This is usually called on an exception

Parameters

- **acme_client** –
- **account_number** –
- **dns_provider** –
- **authorizations** –
- **dns_provider_options** –

Returns

complete_dns_challenge(*acme_client*, *authz_record*)

finalize_authorizations(*acme_client*, *authorizations*)

get_all_zones(*dns_provider*)

get_authorizations(*acme_client*, *order*, *order_info*)

The list can be empty if all hostname validations are still valid

get_cname(*domain*)

Parameters
domain – Domain name to look up a CNAME for.

Returns
First CNAME target or False if no CNAME record exists.

get_dns_challenges(*host*, *authorizations*)

Get dns challenges for provided domain Also indicate if the hostname is already validated

get_dns_provider(*type*)

start_dns_challenge(*acme_client*, *account_number*, *domain*, *target_domain*, *dns_provider*, *order*, *dns_provider_options*)

class lemur.plugins.lemur_acme.acme_handlers.**AcmeHandler**

Bases: object

extract_cert_and_chain(*fullchain_pem*, *alternative_fullchains_pem*, *preferred_issuer=None*)

get_domains(*options*)

Fetches all domains currently requested :param options: :return:

log_remaining_validation(*authorizations*, *acme_account*)

maybe_add_extension(*host*, *dns_provider_options*)

request_certificate(*acme_client*, *authorizations*, *order*)

reuse_account(*authority*)

revoke_certificate(*certificate*, *crl_reason=0*)

setup_acme_client(*authority*)

strip_wildcard(*host*)

Removes the leading wildcard and returns Host and whether it was removed or not (True/False)

class lemur.plugins.lemur_acme.acme_handlers.**AuthorizationRecord**(*domain*, *target_domain*, *authz*, *dns_challenge*, *change_id*, *cname_delegation*)

Bases: object

challenge_types Module

class lemur.plugins.lemur_acme.challenge_types.**AcmeChallenge**

Bases: object

This is the base class, all ACME challenges will need to extend, allowing for future extendability

cleanup(*challenge*, *acme_client*, *validation_target*)

Ideally the challenge should be cleaned up, after the validation is done :param challenge: Needed to identify the challenge to be removed :param acme_client: an already bootstrapped acme_client, to avoid passing all issuer_options and so on :param validation_target: Needed to remove the validation

create_certificate(*csr*, *issuer_options*)

Create the new certificate, using the provided CSR and issuer_options. Right now this is basically a copy of the create_certificate methods in the AcmeHandlers, but should be cleaned and tried to make use of the deploy and cleanup methods

Parameters

- **csr** –
- **issuer_options** –

Returns

deploy(*challenge*, *acme_client*, *validation_target*)

In here the challenge validation is fetched and deployed somewhere that it can be validated by the provider

Parameters

- **self** –
- **challenge** – the challenge object, must match for the challenge implementation
- **acme_client** – an already bootstrapped acme_client, to avoid passing all issuer_options and so on
- **validation_target** – an identifier for the validation target, e.g. the name of a DNS provider

exception lemur.plugins.lemur_acme.challenge_types.**AcmeChallengeMissmatchError**(**args*, ***kwargs*)

Bases: LemurException

class lemur.plugins.lemur_acme.challenge_types.**AcmeDnsChallenge**

Bases: AcmeChallenge

cleanup(*authorizations*, *acme_client*, *validation_target=None*)

Best effort attempt to delete DNS challenges that may not have been deleted previously. This is usually called on an exception

Parameters

- **authorizations** – all the authorizations to be cleaned up
- **acme_client** – an already bootstrapped acme_client, to avoid passing all issuer_options and so on
- **validation_target** – Unused right now

Returns

create_certificate(*csr*, *issuer_options*)

Creates an ACME certificate.

Parameters

- **csr** –
- **issuer_options** –

Returns

raise Exception

create_certificate_immediately(*acme_client*, *order_info*, *csr*)

deploy(*challenge*, *acme_client*, *validation_target*)

In here the challenge validation is fetched and deployed somewhere that it can be validated by the provider

Parameters

- **self** –
- **challenge** – the challenge object, must match for the challenge implementation
- **acme_client** – an already bootstrapped acme_client, to avoid passing all issuer_options and so on
- **validation_target** – an identifier for the validation target, e.g. the name of a DNS provider

class lemur.plugins.lemur_acme.challenge_types.**AcmeHttpChallenge**

Bases: AcmeChallenge

cleanup(*token_path*, *validation_target*)

Ideally the challenge should be cleaned up, after the validation is done :param challenge: Needed to identify the challenge to be removed :param acme_client: an already bootstrapped acme_client, to avoid passing all issuer_options and so on :param validation_target: Needed to remove the validation

create_certificate(*csr*, *issuer_options*)

Creates an ACME certificate using the HTTP-01 challenge.

Parameters

- **csr** –
- **issuer_options** –

Returns

raise Exception

deploy(*challenge*, *acme_client*, *validation_target*)

In here the challenge validation is fetched and deployed somewhere that it can be validated by the provider

Parameters

- **self** –
- **challenge** – the challenge object, must match for the challenge implementation
- **acme_client** – an already bootstrapped acme_client, to avoid passing all issuer_options and so on
- **validation_target** – an identifier for the validation target, e.g. the name of a DNS provider

cloudflare Module

lemur.plugins.lemur_acme.cloudflare.**cf_api_call**()

lemur.plugins.lemur_acme.cloudflare.**create_txt_record**(*host*, *value*, *account_number*)

lemur.plugins.lemur_acme.cloudflare.**delete_txt_record**(*change_ids*, *account_number*, *host*, *value*)

lemur.plugins.lemur_acme.cloudflare.**find_zone_id**(*host*)

lemur.plugins.lemur_acme.cloudflare.**wait_for_dns_change**(*change_id*, *account_number=None*)

dyn Module

lemur.plugins.lemur_acme.dyn.**create_txt_record**(*domain*, *token*, *account_number*)

lemur.plugins.lemur_acme.dyn.**delete_acme_txt_records**(*domain*)

lemur.plugins.lemur_acme.dyn.**delete_txt_record**(*change_id*, *account_number*, *domain*, *token*)

lemur.plugins.lemur_acme.dyn.**get_authoritative_nameserver**(*domain*)

lemur.plugins.lemur_acme.dyn.**get_dynect_session**()

lemur.plugins.lemur_acme.dyn.**get_zone_name**(*domain*)

lemur.plugins.lemur_acme.dyn.**get_zones**(*account_number*)

lemur.plugins.lemur_acme.dyn.**wait_for_dns_change**(*change_id*, *account_number=None*)

nsone Module

ACME DNS providor for NS1

lemur.plugins.lemur_acme.nsone.**create_txt_record**(*domain*, *token*, *account_number=None*)

Create a TXT record for the given domain and token and return a change_id tuple

Parameters

- **domain** – FQDN
- **token** – challenge value
- **account_number** –

Returns

tuple of domain/token

lemur.plugins.lemur_acme.nsone.**delete_txt_record**(*change_id*, *account_number*, *domain*, *token*)

Delete the TXT record for the given domain and token

Parameters

- **change_id** – tuple of domain/token
- **account_number** –
- **domain** – FQDN
- **token** – challenge to delete

Returns

`lemur.plugins.lemur_acme.nsone.get_zones()`

Retrieve authoritative zones from the NS1 API and return a list of zones

Raise
Exception

Returns
list of Zone Objects

`lemur.plugins.lemur_acme.nsone.wait_for_dns_change(change_id, account_number=None)`

Checks the authoritative DNS Server to see if changes have propagated.

Parameters

- **change_id** – tuple of domain/token
- **account_number** –

Returns

plugin Module

class `lemur.plugins.lemur_acme.plugin.ACMEHttpIssuerPlugin(*args, **kwargs)`

Bases: `IssuerPlugin`

author = 'Netflix'

author_url = 'https://github.com/netflix/lemur.git'

cancel_ordered_certificate(*pending_cert*, ***kwargs*)

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 an ACME certificate using the HTTP-01 challenge.

Parameters

- **csr** –
- **issuer_options** –

Returns

raise Exception

description = "Enables the creation of certificates via ACME CAs (including Let's Encrypt), using the HTTP-01 challenge"

```
options = [{'name': 'acme_url', 'type': 'str', 'required': True, 'validation':
'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',
'helpMessage': 'Must be a valid web url starting with http[s]://'}, {'name':
'telephone', 'type': 'str', 'default': '', 'helpMessage': 'Telephone to use'},
{'name': 'email', 'type': 'str', 'default': '', 'validation':
'([-!#-\'*+/-9=?A-Z^-~]+(\\.[-!#-\'*+/-9=?A-Z^-~]+)*|\\"([]!#-[^-~ \\t]|(\\\\[\\t
-~]))+\\")@([-!#-\'*+/-9=?A-Z^-~]+(\\.[-!#-\'*+/-9=?A-Z^-~]+)*|\\[[\\t -Z^-~]*])',
'helpMessage': 'Email to use'}, {'name': 'certificate', 'type': 'textarea',
'default': '', 'validation': '^-----BEGIN CERTIFICATE-----', 'helpMessage':
'ACME root Certificate'}, {'name': 'store_account', 'type': 'bool', 'required':
False, 'helpMessage': 'Disable to create a new account for each ACME request',
'default': False}, {'name': 'eab_kid', 'type': 'str', 'default': '',
'required': False, 'helpMessage': 'Key identifier for the external account.'},
{'name': 'eab_hmac_key', 'type': 'str', 'default': '', 'required': False,
'helpMessage': 'HMAC key for the external account.'}, {'name': 'acme_private_key',
'type': 'textarea', 'default': '', 'required': False, 'helpMessage': 'Account
Private Key. Will be encrypted.'}, {'name': 'acme_regr', 'type': 'textarea',
'default': '', 'required': False, 'helpMessage': 'Account Registration'},
{'name': 'tokenDestination', 'type': 'destinationSelect', 'required': True,
'helpMessage': 'The destination to use to deploy the token.'}, {'name':
'drop_last_cert_from_chain', 'type': 'bool', 'required': False, 'helpMessage':
'Drops the last certificate, i.e., the Cross Signed root, from the Chain',
'default': False}]
```

revoke_certificate(*certificate*, *reason*)

slug = 'acme-http-issuer'

title = 'Acme HTTP-01'

version = 'unknown'

class lemur.plugins.lemur_acme.plugin.**ACMEIssuerPlugin**(**args*, ***kwargs*)

Bases: IssuerPlugin

author = 'Netflix'

author_url = 'https://github.com/netflix/lemur.git'

cancel_ordered_certificate(*pending_cert*, ***kwargs*)

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 an ACME certificate using the DNS-01 challenge.

Parameters

- **csr** –
- **issuer_options** –

Returns

raise Exception

description = "Enables the creation of certificates via ACME CAs (including Let's Encrypt), using the DNS-01 challenge"

get_ordered_certificate(*pending_cert*)

get_ordered_certificates(*pending_certs*)

options = [{'name': 'acme_url', 'type': 'str', 'required': True, 'validation': 'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', 'helpMessage': 'ACME resource URI. Must be a valid web url starting with http[s://'}, {'name': 'telephone', 'type': 'str', 'default': '', 'helpMessage': 'Telephone to use'}, {'name': 'email', 'type': 'str', 'default': '', 'validation': '([-!#-\'*+/-9=?A-Z^-~]+(\\.[-!#-\'*+/-9=?A-Z^-~]+)*|\\"([]!#-[^-~ \\t]|(\\\\[\\t -~]))+\\")@([-!#-\'*+/-9=?A-Z^-~]+(\\.[-!#-\'*+/-9=?A-Z^-~]+)*|\\[[\\t -Z^-~]*])', 'helpMessage': 'Email to use'}, {'name': 'certificate', 'type': 'textarea', 'default': '', 'validation': '^-----BEGIN CERTIFICATE-----', 'helpMessage': 'ACME root certificate'}, {'name': 'store_account', 'type': 'bool', 'required': False, 'helpMessage': 'Disable to create a new account for each ACME request', 'default': False}, {'name': 'eab_kid', 'type': 'str', 'required': False, 'helpMessage': 'Key identifier for the external account.'}, {'name': 'eab_hmac_key', 'type': 'str', 'required': False, 'helpMessage': 'HMAC key for the external account.'}, {'name': 'acme_private_key', 'type': 'textarea', 'default': '', 'required': False, 'helpMessage': 'Account Private Key. Will be encrypted.'}, {'name': 'acme_regr', 'type': 'textarea', 'default': '', 'required': False, 'helpMessage': 'Account Registration'}, {'name': 'drop_last_cert_from_chain', 'type': 'bool', 'required': False, 'helpMessage': 'Drops the last certificate, i.e., the Cross Signed root, from the Chain', 'default': False}]

revoke_certificate(*certificate*, *reason*)

slug = 'acme-issuer'

title = 'Acme'

version = 'unknown'

powerdns Module

class lemur.plugins.lemur_acme.powerdns.**Record**(*_data*)

Bases: object

This class implements a PowerDNS record.

property content

property disabled

property name

property ttl

property type

class lemur.plugins.lemur_acme.powerdns.**Zone**(*_data*)

Bases: object

This class implements a PowerDNS zone in JSON.

property id

Zone id, has a trailing "." at the end, which we manually remove.

property kind

Indicates whether the zone is setup as a PRIMARY or SECONDARY

property name

Zone name, has a trailing "." at the end, which we manually remove.

lemur.plugins.lemur_acme.powerdns.**create_txt_record**(*domain*, *token*, *account_number*)

Create a TXT record for the given domain and token and return a change_id tuple

Parameters

- **domain** – FQDN
- **token** – challenge value
- **account_number** –

Returns

tuple of domain/token

lemur.plugins.lemur_acme.powerdns.**delete_txt_record**(*change_id*, *account_number*, *domain*, *token*)

Delete the TXT record for the given domain and token

Parameters

- **change_id** – tuple of domain/token
- **account_number** –
- **domain** – FQDN
- **token** – challenge to delete

Returns

lemur.plugins.lemur_acme.powerdns.**get_zones**(*account_number*)

Retrieve authoritative zones from the PowerDNS API and return a list of zones

Parameters

account_number –

Raise

Exception

Returns

list of Zone Objects

lemur.plugins.lemur_acme.powerdns.**wait_for_dns_change**(*change_id*, *account_number=None*)

Checks the authoritative DNS Server to see if changes have propagated.

Parameters

- **change_id** – tuple of domain/token
- **account_number** –

Returns

route53 Module

lemur.plugins.lemur_acme.route53.**change_txt_record**(*action*, *zone_id*, *domain*, *value*, *client=None*)

lemur.plugins.lemur_acme.route53.**create_txt_record**(*host*, *value*, *account_number*)

lemur.plugins.lemur_acme.route53.**delete_txt_record**(*change_ids*, *account_number*, *host*, *value*)

lemur.plugins.lemur_acme.route53.**find_zone_id**(*domain*, *client=None*)

lemur.plugins.lemur_acme.route53.**get_zones**(*client=None*)

lemur.plugins.lemur_acme.route53.**wait_for_dns_change**(*change_id*, *client=None*)

ultradns Module

class lemur.plugins.lemur_acme.ultradns.**Record**(*_data*)

Bases: object

This class implements an Ultra DNS record.

Accepts the response from the API call as the argument.

property name

property rdata

property rrtype

property ttl

class lemur.plugins.lemur_acme.ultradns.**Zone**(*_data*, *_client='Client'*)

Bases: object

This class implements an Ultra DNS zone.

property authoritative_type

Indicates whether the zone is setup as a PRIMARY or SECONDARY

property name

Zone name, has a trailing "." at the end, which we manually remove.

property record_count

property status

Returns the status of the zone - ACTIVE, SUSPENDED, etc

lemur.plugins.lemur_acme.ultradns.**create_txt_record**(*domain*, *token*, *account_number*)

Create a TXT record for the given domain.

The part of the domain that matches with the zone becomes the zone name. The remainder becomes the owner name (referred to as node name here) Example: Let's say we have a zone named "exmaple.com" in UltraDNS and we get a request to create a cert for lemur.example.com Domain - _acme-challenge.lemur.example.com Matching zone - example.com Owner name - _acme-challenge.lemur

lemur.plugins.lemur_acme.ultradns.**delete_acme_txt_records**(*domain*)

lemur.plugins.lemur_acme.ultradns.**delete_txt_record**(*change_id*, *account_number*, *domain*, *token*)

Delete the TXT record that was created in the create_txt_record() function.

UltraDNS handles records differently compared to Dyn. It creates an RRSet which is a set of records of the same type and owner. This means that while deleting the record, we cannot delete any individual record from the RRSet. Instead, we have to delete the entire RRSet. If multiple certs are being created for the same domain at the same time, the challenge TXT records that are created will be added under the same RRSet. If the RRSet had more than 1 record, then we create a new RRSet on UltraDNS minus the record that has to be deleted.

lemur.plugins.lemur_acme.ultradns.**get_authoritative_nameserver**(*domain*)

Get the authoritative nameserver for the given domain

lemur.plugins.lemur_acme.ultradns.**get_public_authoritative_nameserver**()

lemur.plugins.lemur_acme.ultradns.**get_ultradns_token**()

Function to call the UltraDNS Authorization API.

Returns the Authorization access_token which is valid for 1 hour. Each request calls this function and we generate a new token every time.

lemur.plugins.lemur_acme.ultradns.**get_zone_name**(*domain*, *account_number*)

Get the matching zone for the given domain

lemur.plugins.lemur_acme.ultradns.**get_zones**(*account_number*)

Get zones from the UltraDNS

lemur.plugins.lemur_acme.ultradns.**wait_for_dns_change**(*change_id*, *account_number=None*)

Waits and checks if the DNS changes have propagated or not.

First check the domains authoritative server. Once this succeeds, we ask a public DNS server (Google <8.8.8.8> in our case).

lemur_atlas package

lemur_atlas Module

plugin Module

class lemur.plugins.lemur_atlas.plugin.**AtlasMetricPlugin**

Bases: MetricPlugin

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur'

description = 'Adds support for sending key metrics to Atlas'

metric_data = {}

options = [{'name': 'sidecar_host', 'type': 'str', 'required': False, 'help_message': 'If no host is provided localhost is assumed', 'default': 'localhost'}, {'name': 'sidecar_port', 'type': 'int', 'required': False, 'default': 8078}]

sidecar_host = None

sidecar_port = None

slug = 'atlas-metric'

submit(*metric_name*, *metric_type*, *metric_value*, *metric_tags=None*, *options=None*)

title = 'Atlas'

version = 'unknown'

lemur.plugins.lemur_atlas.plugin.**millis_since_epoch**()

current time since epoch in milliseconds

lemur_cryptography package

lemur_cryptography Module

plugin Module

class lemur.plugins.lemur_cryptography.plugin.**CryptographyIssuerPlugin**

Bases: 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*, *options*)

Creates a certificate.

Parameters

- **csr** –
- **options** –

Returns

raise Exception

description = 'Enables the creation and signing of self-signed certificates'

slug = 'cryptography-issuer'

title = 'Cryptography'

version = 'unknown'

lemur.plugins.lemur_cryptography.plugin.**build_certificate_authority**(*options*)

lemur.plugins.lemur_cryptography.plugin.**filter_san_extensions**(*ext*)

lemur.plugins.lemur_cryptography.plugin.**issue_certificate**(*csr*, *options*, *private_key=None*)

lemur.plugins.lemur_cryptography.plugin.**normalize_extensions**(*csr*)

lemur_digicert package

lemur_digicert Module

plugin Module

class lemur.plugins.lemur_digicert.plugin.**DigiCertCISIssuerPlugin**(**args*, ***kwargs*)

Bases: IssuerPlugin

Wrap the Digicert Certificate Issuing API.

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur.git'

static create_authority(*options*)

Create an authority.

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*)

Create a DigiCert certificate.

description = 'Enables the creation of certificates by the DigiCert CIS REST API.'

revoke_certificate(*certificate*, *reason*)

Revoke a Digicert certificate.

slug = 'digicert-cis-issuer'

title = 'DigiCert CIS'

version = 'unknown'

class lemur.plugins.lemur_digicert.plugin.**DigiCertCISSourcePlugin**(**args*, ***kwargs*)

Bases: SourcePlugin

Wrap the Digicert CIS Certifcate API.

additional_options = []

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur.git'

description = 'Enables the use of Digicert as a source of existing certificates.'

get_certificates(*options*, ***kwargs*)

Fetch all Digicert certificates.

slug = 'digicert-cis-source'

title = 'DigiCert'

version = 'unknown'

class lemur.plugins.lemur_digicert.plugin.**DigiCertIssuerPlugin**(**args*, ***kwargs*)

Bases: IssuerPlugin

Wrap the Digicert Issuer API.

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur.git'

cancel_ordered_certificate(*pending_cert*, ***kwargs*)

Set the certificate order to canceled

static create_authority(*options*)

Create an authority.

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*)

Create a DigiCert certificate.

Parameters

- **csr** –
- **issuer_options** –

Returns

raise Exception

description = 'Enables the creation of certificates by the DigiCert REST API.'

get_ordered_certificate(*pending_cert*)

Retrieve a certificate via order id

revoke_certificate(*certificate*, *reason*)

Revoke a Digicert certificate.

slug = 'digicert-issuer'

title = 'DigiCert'

version = 'unknown'

class lemur.plugins.lemur_digicert.plugin.**DigiCertSourcePlugin**(**args*, ***kwargs*)

Bases: SourcePlugin

Wrap the Digicert Certifcate API.

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur.git'

description = 'Enables the use of Digicert as a source of existing certificates.'

get_certificates()

slug = 'digicert-source'

title = 'DigiCert'

version = 'unknown'

lemur.plugins.lemur_digicert.plugin.**determine_end_date**(*end_date*)

Determine appropriate end date

Parameters
end_date –

Returns
validity_end

lemur.plugins.lemur_digicert.plugin.**determine_validity_years**(*years*)

Considering maximum allowed certificate validity period of 397 days, this method should not return more than 1 year of validity. Thus changing it to always return 1. Lemur will change this method in future to handle validity in months (determine_validity_months) instead of years. This will allow flexibility to handle short-lived certificates.

Parameters
years –

Returns
1

lemur.plugins.lemur_digicert.plugin.**get_additional_names**(*options*)

Return a list of strings to be added to a SAN certificates.

Parameters
options –

Returns

lemur.plugins.lemur_digicert.plugin.**get_certificate_id**(*session*, *base_url*, *order_id*)

Retrieve certificate order id from Digicert API.

lemur.plugins.lemur_digicert.plugin.**get_cis_certificate**(*session*, *base_url*, *order_id*)

Retrieve certificate order id from Digicert API, including the chain

lemur.plugins.lemur_digicert.plugin.**handle_cis_response**(*session*, *response*)

Handle the DigiCert CIS API response and any errors it might have experienced. :param response: :return:

lemur.plugins.lemur_digicert.plugin.**handle_response**(*response*)

Handle the DigiCert API response and any errors it might have experienced. :param response: :return:

lemur.plugins.lemur_digicert.plugin.**log_status_code**(*r*, **args*, ***kwargs*)

Is a request hook that logs all status codes to the digicert api.

Parameters

- **r** –

- **args** –
- **kwargs** –

Returns

`lemur.plugins.lemur_digicert.plugin.log_validity_truncation(options, function)`

`lemur.plugins.lemur_digicert.plugin.map_cis_fields(options, csr)`

MAP issuer options to DigiCert CIS fields/options.

Parameters

- **options** –
- **csr** –

Returns
data

`lemur.plugins.lemur_digicert.plugin.map_fields(options, csr)`

Set the incoming issuer options to DigiCert fields/options.

Parameters

- **options** –
- **csr** –

Returns
dict or valid DigiCert options

`lemur.plugins.lemur_digicert.plugin.reset_cis_session(session)`

The current session might be in a bad state with wrong headers. Let's attempt to update the session back to the initial state. :param session: :return:

`lemur.plugins.lemur_digicert.plugin.signature_hash(signing_algorithm)`

Converts Lemur's signing algorithm into a format DigiCert understands.

Parameters
signing_algorithm –

Returns
str digicert specific algorithm string

lemur_jks package

lemur_jks Module

plugin Module

class `lemur.plugins.lemur_jks.plugin.JavaKeystoreExportPlugin`

Bases: `ExportPlugin`

author = 'Marti Raudsepp'

author_url = 'https://github.com/intgr'

description = 'Generates a JKS keystore'

export(*body*, *chain*, *key*, *options*, ***kwargs*)

Generates a Java Keystore

options = [{'name': 'passphrase', 'type': 'str', 'required': False, 'helpMessage': 'If no passphrase is given one will be generated for you, we highly recommend this.', 'validation': ''}, {'name': 'alias', 'type': 'str', 'required': False, 'helpMessage': 'Enter the alias you wish to use for the keystore.'}]

slug = 'java-keystore-jks'

title = 'Java Keystore (JKS)'

version = 'unknown'

class lemur.plugins.lemur_jks.plugin.JavaTruststoreExportPlugin

Bases: ExportPlugin

author = 'Marti Raudsepp'

author_url = 'https://github.com/intgr'

description = 'Generates a JKS truststore'

export(*body*, *chain*, *key*, *options*, ***kwargs*)

Generates a Java Truststore

options = [{'name': 'alias', 'type': 'str', 'required': False, 'helpMessage': 'Enter the alias you wish to use for the truststore.'}, {'name': 'passphrase', 'type': 'str', 'required': False, 'helpMessage': 'If no passphrase is given one will be generated for you, we highly recommend this.', 'validation': ''}]

requires_key = False

slug = 'java-truststore-jks'

title = 'Java Truststore (JKS)'

version = 'unknown'

lemur.plugins.lemur_jks.plugin.cert_chain_as_der(*cert*, *chain*)

Return a certificate and its chain in a list format, as expected by pyjks.

lemur.plugins.lemur_jks.plugin.create_keystore(*cert*, *chain*, *key*, *alias*, *passphrase*)

lemur.plugins.lemur_jks.plugin.create_truststore(*cert*, *chain*, *alias*, *passphrase*)

lemur_kubernetes package

lemur_kubernetes Module

plugin Module

class lemur.plugins.lemur_kubernetes.plugin.K8sSession(*bearer*, *cert_file*)

Bases: Session

request(*method*, *url*, *params=None*, *data=None*, *headers=None*, *cookies=None*, *files=None*, *auth=None*, *timeout=30*, *allow_redirects=True*, *proxies=None*, *hooks=None*, *stream=None*, *verify=None*, *cert=None*, *json=None*)

This method overrides the default timeout to be 10s.

class lemur.plugins.lemur_kubernetes.plugin.**KubernetesDestinationPlugin**(**args*, ***kwargs*)

Bases: DestinationPlugin

author = 'Mikhail Khodorovskiy'

author_url = 'https://github.com/mik373/lemur'

description = 'Allow the uploading of certificates to Kubernetes as secret'

k8s_bearer(*options*)

k8s_cert(*options*)

k8s_namespace(*options*)

options = [{'name': 'secretNameFormat', 'type': 'str', 'required': False, 'validation': '(?:[a-z0-9.-]|\\{common_name\\})+', 'helpMessage': 'Must be a valid secret name, possibly including "{common_name}"', 'default': '{common_name}'}, {'name': 'kubernetesURL', 'type': 'str', 'required': False, 'validation': 'https?://[a-zA-Z0-9.-]+(?::[0-9]+)?', 'helpMessage': 'Must be a valid Kubernetes server URL!', 'default': 'https://kubernetes.default'}, {'name': 'kubernetesAuthToken', 'type': 'str', 'required': False, 'validation': '[0-9a-zA-Z-_.]+', 'helpMessage': 'Must be a valid Kubernetes server Token!'}, {'name': 'kubernetesAuthTokenFile', 'type': 'str', 'required': False, 'validation': '(/[^/]+)+', 'helpMessage': 'Must be a valid file path!', 'default': '/var/run/secrets/kubernetes.io/serviceaccount/token'}, {'name': 'kubernetesServerCertificate', 'type': 'textarea', 'required': False, 'validation': '-----BEGIN CERTIFICATE-----[a-zA-Z0-9/+\\s\\r\\n]+-----END CERTIFICATE-----', 'helpMessage': 'Must be a valid Kubernetes server Certificate!'}, {'name': 'kubernetesServerCertificateFile', 'type': 'str', 'required': False, 'validation': '(/[^/]+)+', 'helpMessage': 'Must be a valid file path!', 'default': '/var/run/secrets/kubernetes.io/serviceaccount/ca.crt'}, {'name': 'kubernetesNamespace', 'type': 'str', 'required': False, 'validation': '[a-z0-9]([-a-z0-9]*[a-z0-9])?', 'helpMessage': 'Must be a valid Kubernetes Namespace!'}, {'name': 'kubernetesNamespaceFile', 'type': 'str', 'required': False, 'validation': '(/[^/]+)+', 'helpMessage': 'Must be a valid file path!', 'default': '/var/run/secrets/kubernetes.io/serviceaccount/namespace'}, {'name': 'secretFormat', 'type': 'select', 'required': True, 'available': ['Full', 'TLS', 'Certificate'], 'helpMessage': 'The type of Secret to create.', 'default': 'Full'}]

slug = 'kubernetes-destination'

title = 'Kubernetes'

upload(*name*, *body*, *private_key*, *cert_chain*, *options*, ***kwargs*)

lemur.plugins.lemur_kubernetes.plugin.**build_secret**(*secret_format*, *secret_name*, *body*, *private_key*, *cert_chain*)

lemur.plugins.lemur_kubernetes.plugin.**ensure_resource**(*k8s_api*, *k8s_base_uri*, *namespace*, *kind*, *name*, *data*)

lemur_openssl package

lemur_openssl Module

plugin Module

class lemur.plugins.lemur_openssl.plugin.**OpenSSLExportPlugin**

Bases: ExportPlugin

author = 'Kevin Glisson'

author_url = 'https://github.com/netflix/lemur'

description = 'Is a loose interface to openssl and support various formats'

export(*body*, *chain*, *key*, *options*, ***kwargs*)

Generates a PKCS#12 archive.

Parameters

- **key** –
- **chain** –
- **body** –
- **options** –
- **kwargs** –

options = [{'name': 'type', 'type': 'select', 'required': True, 'available': ['PKCS12 (.p12)'], 'helpMessage': 'Choose the format you wish to export'}, {'name': 'passphrase', 'type': 'str', 'required': False, 'helpMessage': 'If no passphrase is given one will be generated for you, we highly recommend this.', 'validation': ''}, {'name': 'alias', 'type': 'str', 'required': False, 'helpMessage': 'Enter the alias you wish to use for the keystore.'}]

slug = 'openssl-export'

title = 'OpenSSL'

version = 'unknown'

lemur.plugins.lemur_openssl.plugin.**create_pkcs12**(*cert*, *chain*, *p12_tmp*, *key*, *alias*, *passphrase*)

Creates a pkcs12 formated file. :param cert: :param chain: :param p12_tmp: :param key: :param alias: :param passphrase:

lemur.plugins.lemur_openssl.plugin.**run_process**(*command*)

Runs a given command with pOpen and wraps some error handling around it. :param command: :return:

lemur_slack package

lemur_slack Module

plugin Module

class lemur.plugins.lemur_slack.plugin.**SlackNotificationPlugin**

Bases: ExpirationNotificationPlugin

additional_options = [{'name': 'webhook', 'type': 'str', 'required': True, 'validation': '^https:\\/\\/hooks\\.slack\\.com\\/services\\/.+$', 'helpMessage': 'The url Slack told you to use for this integration'}, {'name': 'username', 'type': 'str', 'validation': '^.+$', 'helpMessage': 'The great storyteller', 'default': 'Lemur'}, {'name': 'recipients', 'type': 'str', 'required': True, 'validation': '^(@|#).+$', 'helpMessage': 'Where to send to, either @username or #channel'}]

author = 'Harm Weites'

author_url = 'https://github.com/netflix/lemur'

description = 'Sends notifications to Slack'

send(*notification_type*, *message*, *targets*, *options*, ***kwargs*)

A typical check can be performed using the notify command: *lemur notify*

While we receive a *targets* parameter here, it is unused, as Slack webhooks do not allow dynamic re-targeting of messages. The webhook itself specifies a channel.

slug = 'slack-notification'

title = 'Slack'

version = 'unknown'

lemur.plugins.lemur_slack.plugin.**create_certificate_url**(*name*)

lemur.plugins.lemur_slack.plugin.**create_expiration_attachments**(*certificates*)

lemur.plugins.lemur_slack.plugin.**create_rotation_attachments**(*certificate*)

CHAPTER

FIVE

SECURITY

5.1 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.

5.1.1 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 `lemur@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.

5.1.2 Supported Versions

At any given time, we will provide security support for the master branch as well as the most recent release.

5.1.3 Disclosure Process

Our process for taking a security issue from private discussion to public disclosure involves multiple steps. Our standard process utilizes a GitHub Security Advisory.

The general process is as follows:

1. Receive a private report of a security issue
2. Acknowledge receipt of the report
3. Post advance notice to the GitHub repo indicating that a security issue exists
4. Prepare a GitHub Security Advisory
5. Merge code fix
6. Make Security Advisory public

Private report

After receiving a private report of a security issue, the reporter will receive notification of the date on which we plan to make the issue public. We also ask the reporter for their GitHub username if they'd like to receive credit for their finding.

Advance Notice

Approximately one week before full public disclosure, we will provide advance notification that a security issue exists. This will take the form of an issue posted to the Lemur repository. The notification should contain the following, as appropriate (details will only be shared to the extent that they do not highlight an unpatched vulnerability):

- A description of the potential impact
- The affected versions of `lemur`
- The steps we will be taking to remedy the issue
- The date on which the `lemur` team will apply these patches, issue new releases, and publicly disclose the issue

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.

GitHub Security Advisory

During the (approximate) week between advance notice and public disclosure, we will prepare a description of the security issue using a GitHub Security Advisory. The fix for the issue should also be prepared using the private fork provided by the security advisory.

Day of Disclosure

On the day of disclosure, we will take the following steps:

1. Merge relevant patches to the `lemur` repository (from the security advisory fork)
2. Issue an updated release
3. Make the security advisory public

CHAPTER

SIX

DOING A RELEASE

6.1 Doing a release

Doing a release of `lemur` is now mostly automated and consists of the following steps:

- Raise a PR to add the release date and summary in the *Changelog*.
- Merge above PR and create a new Github release: set the tag starting with v, e.g., v0.9.0

The publish workflow uses the git tag to set the release version.

The following describes the manual release steps, which is now obsolete:

6.1.1 Manually 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, and raise a pull request with this.
- Wait for it to be merged.

6.1.2 Manually Performing the release

The commit that merged the version number bump is now the official release commit for this release. You need an API key, which requires permissions to maintain the Lemur project.

For creating the release, follow these steps (more details here)

- Make sure you have the latest versions of setuptools and wheel installed:

  ```
  python3 -m pip install --user --upgrade setuptools wheel
  ```

- Now run this command from the same directory where setup.py is located:

  ```
  python3 setup.py sdist bdist_wheel
  ```

- Once completed it should generate two files in the dist directory:

```
$ ls dist/
lemur-0.8.0-py2.py3-none-any.whl    lemur-0.8.0.tar.gz
```

- In this step, the distribution will be uploaded. You'll need to install Twine:

```
python3 -m pip install --user --upgrade twine
```

- Once installed, run Twine to upload all of the archives under dist. Once installed, run Twine to upload all of the archives under dist:

```
python3 -m twine upload --repository pypi dist/*
```

The release should now be available on PyPI Lemur and a tag should be available in the repository.

Make sure to also make a github release which will pick up the latest version.

6.1.3 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.

6.1.4 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

SEVEN

FAQ

7.1 Frequently Asked Questions

7.1.1 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 *Configuration* 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.

7.1.2 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
```

(continues on next page)

(continued from previous page)

```
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

EIGHT

REFERENCE

8.1 Changelog

8.1.1 Unreleased

8.1.2 1.4.0 - *2023-04-04*

Added support for Python 3.10, Postgres 15, and Ubuntu 22.04. Removed support for Postgres 10 and Ubuntu 18.04.

Python 3.11 is known not to work with the current version of Flask.

All combinations tested via GitHub Actions are listed below:

Table 1: Version Support Matrix

Python	Postgres	Ubuntu
3.8	12	20.04
3.8	15	20.04
3.9	12	20.04
3.9	15	20.04
3.9	15	20.04
3.10	12	22.04
3.10	15	22.04

Added additional validation and logging for destinations. Destination labels are now limited to 32 characters, and s3 prefixes can no longer begin with /. S3 destination path prefixes now default to “” instead of “None/”

Enforce case consistency in authority signing algorithms. Specifically, this renames SHA384withECDSA -> sha384WithECDSA and SHA512withECDSA -> sha512WithECDSA. Notably, the backend schema will still accept the uppercase equivalents to maintain backwards compatibility.

8.1.3 1.3.2 - *2023-02-24*

This release contains a fix for a security vulnerability.

8.1.4 1.3.1 - *2023-02-15*

This release contains no changes.

8.1.5 1.3.0 - *2023-02-13*

This release contains many dependency updates, and numerous added or improved features over the last year.

Some of the notable changes in this release are: - Removal of AWS S3 destinations and the respetive resources via the UI - No fine-grained authz for role global_cert_issuer - De-activate endpoint (Entrust Plugin) - Remove unsafe paginate method and replace with sort_and_page - Move to github workflows for tests - Detect duplicate certs - Metrics for certificate expiry - Sync source: handling idle/invalidated connection - Sync endpoint: capture error and continue - Domain-level fine-grained authz - Handle and report authz warmup exception - Ensure secondary certificates are not removed when rotating AWS endpoints - Improved metric around expired endpoints - Change pkg_resources call in plugin loading to use resolve rather than load - Log when an expiring deployed certificate is detected - NS1 DNS ACME Plugin - Add a new endpoint that allows updating a certificate owner - Support rotating endpoints with non-unique names via CLI - Restrict multiple accounts on a certificate, by plugin - Moving to dependabot's auto versioning strategy

Special thanks to all who contributed to this release, notably:

- Neil Schelly
- Mitch Cail
- Bob Shannon
- alwaysjolley

8.1.6 1.2.0 - *2022-01-31*

This release fixes a vulnerability where creating an authority automatically granted the selected owner role to the authority creator, which allowed users to grant themselves to arbitrary roles. The owner role is no longer auto-assigned when creating an authority.

Additionally, all authorities now receive a unique role upon creation. Previously, authorities using the same issuer plugin would always share a role (for example, Entrust authorities always used the role "entrust"). Now, authorities are associated with a unique role named in the format *issuerPlugin_authority_name_admin*. The creator will not be automatically added to this role.

Other notable changes: - The Endpoints UI page now displays endpoint source and allows filtering by source

8.1.7 1.1.0 - *2022-01-10*

Introducing new Plugins AuthorizationPlugin(Plugin) and DomainAuthorizationPlugin(AuthorizationPlugin). One can implement a DomainAuthorizationPlugin to check if caller is authorized to issue a certificate for a given Common Name and Subject Alternative Name (SAN) of type DNSName (PR #3889)

Related to the above change (PR #3889), a new column *application_name* is added to the *api_keys* table. Null values are allowed making sure this change is backward compatible.

Other notable changes: - A task name is fixed from *identity_expiring_deployed_certificates* -> *identify_expiring_deployed_certificates*. The old task name with typo is marked as deprecated and will be removed in future release flagging it as a breaking change. (Thanks to Bob Shannon) - ID filter on certificates UI requires a numeric value.

8.1.8 1.0.0 - *2022-01-06*

This is our first major release due to a dependency on Python 3.8. Lemur is now using flake8>=4.0 and pyflakes>=2.4, requiring Python 3.8 or higher. Our GitHub Actions Builds are currently on Python 3.8 and Python 3.9.

8.1.9 0.11.0 - *2022-01-05*

This release includes multiple improvements on many fronts. The next release will be a major release, requiring Python 3.8 or higher.

Some of the notable changes in this release are:

- CloudFront Plugin: a new endpoint with rotation support
- Improved Endpoint expiration flow; the Sync job now expires old endpoints
- AWS ELB tag supports to opt-out of auto-rotate for load balancers
- Membership plugin
- Moving Travis Build to Node 16
- OAuth2 & Ping Config improvement
- Improved Certificate status check
- **Improved ACME plugin:**
  - reuse existing domain validation resulting in faster issuance
  - IP certificate issuance support, accompanied by UI support
  - emit remaining domain validation
- Azure destination: Switch to PCKS12 upload
- **Improved logs, such as:**
  - Warning logs for admin role assignment and authority creation
  - Audit logs in JSON format for better search
  - Improved SES logging

Special thanks to all who contributed to this release, notably: - Bob Shannon - sirferl - Sam Havron - Guillaume Dumont - Joe McRobot

8.1.10 0.10.0 - *2021-06-28*

This release introduces a breaking change (PR #3646) to the following API endpoint:

- POST /certificates/1/update/notify

The endpoint is now:

- POST /certificates/1/update/switches

The new endpoint honors the existing *notify* request parameter, and additionally accepts a new *rotation* parameter. As a result of this change, the certificate table view now includes rotation switches and filtering by rotation status.

Other notable changes in this release:

- **ACME:**
  - New celery task to prevent duplicate certificates from being autorotated
  - ACME DNS-01 Challenges are supported in synchronous mode
  - DNS provider check fails gracefully if not found
- **Authentication:**
  - SSO auth now returns a newly created user during initial login
  - CSRF protection is added to OAuth2.0
- **Notifications:**
  - New reissue failed notification
  - New reissue with no endpoints notification
  - New revocation notification
- **Plugins:**
  - Plugin option values are validated server-side
  - Some plugin option validations updated to compile successfully server-side
- Database: - Source and Destination deletions remove certificate associations with new confirmation dialog
- Dependency updates and conflict resolutions
- Expanded audit logs

And several smaller bugfixes and improvements.

Special thanks to all who contributed to this release, notably:

- havron
- tho
- mizzy

8.1.11 0.9.0 - *2021-03-17*

This release fixes three critical vulnerabilities where an authenticated user could retrieve/access unauthorized information. (Issue #3463)

8.1.12 0.8.1 - *2021-03-12*

This release includes improvements on many fronts, such as:

- **Notifications:**
  - Enhanced SNS flow
  - Expiration Summary
  - CA expiration email
- EC algorithm as the default
- Improved revocation flow
- Localized AWS STS option
- Improved Lemur doc building
- **ACME:**
  - reduced failed attempts to 3x trials
  - support for selecting the chain (Let's Encrypt X1 transition)
  - revocation
  - http01 documentation
- **Entrust:**
  - Support for cross-signed intermediate CA
- Revised disclosure process
- Dependency updates and conflict resolutions

Special thanks to all who contributed to this release, notably:

- peschmae
- atugushev
- sirferl

8.1.13 0.8.0 - *2020-11-13*

This release comes after more than two years and contains many interesting new features and improvements. In addition to multiple new plugins, such as ACME-http01, ADCS, PowerDNS, UltraDNS, Entrust, SNS, many of Lemur's existing flows have improved.

In the future, we plan to do frequent releases.

Summary of notable changes:

- AWS S3 plugin: added delete, get methods, and support for uploading/deleting acme tokens
- **ACME plugin:**

  - revamp of the plugin
  - support for http01 domain validation, via S3 and SFTP as destination for the acme token
  - support for CNAME delegated domain validation
  - store-acme-account-details
- PowerDNS plugin
- UltraDNS plugin
- ADCS plugin
- SNS plugin
- Entrust plugin
- **Rotation:**
  - respecting keyType and extensions
  - region-by-region rotation option
  - default to auto-rotate when cert attached to endpoint
  - default to 1y validity during rotation for multi-year browser-trusted certs
- Certificate: search_by_name, and important performance improvements
- **UI**
  - reducing the EC curve options to the relevant ones
  - edit option for notifications, destinations and sources
  - showing 13 month validity as default
  - option to hide certs expired since 3month
  - faster Permalink (no search involved)
  - commonName Auto Added as DNS in the UI
  - improved search and cert lookup
- celery tasks instead of crone, for better logging and monitoring
- **countless bugfixes**
  - group-lookup-fix-referral
  - url_context_path
  - duplicate notification
  - digicert-time-bug-fix
  - improved-csr-support
  - fix-cryptography-intermediate-ca
  - enhanced logging
  - vault-k8s-auth
  - cfssl-key-fix
  - cert-sync-endpoint-find-by-hash
  - nlb-naming-bug

  - fix_vault_api_v2_append
  - aid_openid_roles_provider_integration
  - rewrite-java-keystore-use-pyjks
  - vault_kv2

To see the full list of changes, you can run

$ git log –merges –first-parent master –pretty=format:"%h %<(10,trunc)%aN %C(white)%<(15)%ar%Creset %C(red bold)%<(15)%D%Creset %s" | grep -v "depend"

Special thanks to all who contributed to this release, notably:

- peschmae
- sirferl
- lukasmrtvy
- intgr
- kush-bavishi
- alwaysjolley
- jplana
- explody
- titouanc
- jramosf

Upgrading

Note: This release will need a migration change. Please follow the documentation to upgrade Lemur.

8.1.14 0.7 - *2018-05-07*

This release adds LetsEncrypt support with DNS providers Dyn, Route53, and Cloudflare, and expands on the pending certificate functionality. The linux_dst plugin will also be deprecated and removed.

The pending_dns_authorizations and dns_providers tables were created. New columns were added to the certificates and pending_certificates tables, (For the DNS provider ID), and authorities (For options). Please run a database migration when upgrading.

The Let's Encrypt flow will run asynchronously. When a certificate is requested through the acme-issuer, a pending certificate will be created. A cron needs to be defined to run *lemur pending_certs fetch_all_acme*. This command will iterate through all of the pending certificates, request a DNS challenge token from Let's Encrypt, and set the appropriate _acme-challenge TXT entry. It will then iterate through and resolve the challenges before requesting a certificate for each pending certificate. If a certificate is successfully obtained, the pending_certificate will be moved to the certificates table with the appropriate properties.

Special thanks to all who helped with this release, notably:

- The folks at Cloudflare
- dmitryzykov

- jchuong
- seils
- titouanc

Upgrading

Note: This release will need a migration change. Please follow the documentation to upgrade Lemur.

8.1.15 0.6 - *2018-01-02*

Happy Holidays! This is a big release with lots of bug fixes and features. Below are the highlights and are not exhaustive.

Features:

- Per-certificate rotation policies, requires a database migration. The default rotation policy for all certificates is 30 days. Every certificate will gain a policy regardless of if auto-rotation is used.
- Adds per-user API Keys, allows users to issue multiple long-lived API tokens with the same permission as the user creating them.
- Adds the ability to revoke certificates from the Lemur UI/API, this is currently only supported for the digicert CIS and cfssl plugins.
- Allow destinations to support an export function. Useful for file system destinations e.g. S3 to specify the export plugin you wish to run before being sent to the destination.
- Adds support for uploading certificates to Cloudfront.
- Re-worked certificate metadata pane for improved readability.
- Adds support for LDAP user authentication

Bugs:

- Closed #767 - Fixed issue with login redirect loop.
- Closed #792 - Fixed an issue with a unique constraint was violated when replacing certificates.
- Closed #752 - Fixed an internal server error when validating notification units.
- Closed #684 - Fixed migration failure when null values encountered.
- Closes #661 - Fixed an issue where default values were missing during clone operations.

Special thanks to all who helped with this release, notably:

- intgr
- SecurityInsanity
- johanneslange
- RickB17
- pr8kerl
- bunjiboys

See the full list of issues closed in 0.6.

Upgrading

Note: This release will need a migration change. Please follow the documentation to upgrade Lemur.

8.1.16 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 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.

8.1.17 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.

8.1.18 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 an 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 to 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 an 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 #275 - 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** explicit, including adding the ability to add roles directly to certificates and authorities on creation.

8.1.19 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

8.1.20 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

8.1.21 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

8.1.22 0.1.5 - 2015-10-26

- **SECURITY ISSUE**: Switched from use an 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

8.2 License

Lemur is licensed under a three clause APACHE License.

The full license text can be found below (*Lemur License*).

8.2.1 Authors

Lemur was originally written and is maintained by Kevin Glisson.

A list of additional contributors can be seen on GitHub.

8.2.2 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.
```

(continues on next page)

(continued from previous page)

```
    "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)
```

(continues on next page)

(continued from previous page)

```
      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
```

(continues on next page)

(continued from previous page)

```
   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
```

(continues on next page)

(continued from previous page)

```
   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.

Copyright 2018 Netflix, Inc.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

PYTHON MODULE INDEX

l

HTTP ROUTING TABLE

/users

INDEX

G

I

L

M

N

O

P

R

S

U

V