SLUGS Documentation

Release 1.0.0

Peter Hamilton

Jan 25, 2019

Contents

The Simple, Lightweight User Group Services (SLUGS) library provides a simple web service that serves user/group membership data over a basic REST interface.

```
>>> import requests
>>> requests.get('http://127.0.0.1:8080/slugs').json()
{u'users': [u'Jane', u'John'], u'groups': [u'Male', u'Female', u'Human']}
>>> requests.get('http://127.0.0.1:8080/slugs/users').json()
{u'users': [u'Jane', u'John']}
>>> requests.get('http://127.0.0.1:8080/slugs/users/John').status_code
200
>>> requests.get('http://127.0.0.1:8080/slugs/users/John/groups').json()
{u'groups': [u'Male', u'Human']}
>>> requests.get('http://127.0.0.1:8080/slugs/users/John/groups/Male').status_code
200
```

For more information on the SLUGS REST API, see *API*.

SLUGS is built using CherryPy, a well established object-oriented web framework for Python. To run SLUGS, simply install the library and then:

```
$ slugs -c /path/to/config/file
```

For more information on configuring SLUGS, see *Configuration*.

CHAPTER 1

Installation

You can install SLUGS via `pip`:

```
$ pip install slugs
```

See *Installation* for more information.

CHAPTER 2

Layout

2.1 API

The SLUGS REST API allows clients to query for membership information about a many-to-many relationship between a set of users and a set of groups. Queries can be made from a user-centric or group-centric point-of-view.

All REST queries are rooted under `/slugs`. If the base URL is `http://127.0.0.1:8080`, then the full service URL is `http://127.0.0.1:8080/slugs`. For example, to retrieve the list of recognized users, `GET /users`, the full API call would be: `http://127.0.0.1:8080/slugs/users`.

2.1.1 GET

/

List all users and groups recognized by the service.

Response

Response	Response Code	Details
Normal	200	User and group lists returned.

Name	Type	Description
users	array	A list of strings, representing users recognized by the service.
groups	array	A list of strings, representing groups recognized by the service.

A response example would look like:

```
{
    "users": [
        "John",
        "Jane"
    ],
    "groups": [
        "Human",
        "Male",
        "Female"
    ]
}
```

/users

List all users recognized by the service.

Response

Response	Response Code	Details
Normal	200	User list returned.

Name	Type	Description
users	array	A list of strings, representing users recognized by the service.

A response example would look like:

```
{
    "users": [
        "John",
        "Jane"
    ]
}
```

/groups

List all groups recognized by the service.

Response

Response	Response Code	Details
Normal	200	Group list returned.

Name	Type	Description
groups	array	A list of strings, representing groups recognized by the service.

A response example would look like:

```
{
    "groups": [
        "Human",
        "Male",
        "Female"
    ]
}
```

/users/{user}

Query if `{user}` is a user recognized by the service.

Response

Response	Response Code	Details
Normal	200	`{user}` is a user recognized by the service.
Error	404	`{user}` is not a user recognized by the service.

/groups/{group}

Query if `{group}` is a group recognized by the service.

Response

Response	Response Code	Details
Normal	200	`{group}` is a group recognized by the service.
Error	404	`{group}` is not a group recognized by the service.

/users/{user}/groups

List all groups associated with user `{user}`.

Response

Response	Response Code	Details
Normal	200	Group list returned.
Error	404	`{user}` is not a user recognized by the service.

Name	Type	Description
groups	array	A list of strings, representing groups associated with user `{user}`.

A response example would look like:

```
{
    "groups": [
        "Human",
        "Male"
    ]
}
```

/groups/{group}/users

List all users associated with group `{group}`.

Response

Response	Response Code	Details
Normal	200	User list returned.
Error	404	`{group}` is not a group recognized by the service.

Name	Type	Description
users	array	A list of strings, representing users associated with group `{group}`.

A response example would look like:

```
{
    "users": [
        "Jane",
        "John"
    ]
}
```

/users/{user}/groups/{group}

Query if `{group}` is a group associated with user `{user}`.

Response

Re-sponse	Response Code	Details
Normal	200	`{group}` is a group associated with user `{user}`.
Error	404	`{user}` is not a user recognized by the service, or `{group}` is not a group associated with user `{user}`.

/groups/{group}/users/{user}

Query if `{user}` is a user associated with group `{group}`.

Response

Response	Response Code	Details
Normal	200	`{user}` is a user associated with group `{group}`.
Error	404	`{group}` is not a group recognized by the service, or `{user}` is not a user associated with group `{group}`.

2.2 Changelog

2.2.1 1.1.0 - September 25, 2019

- Add TLS support to the SLUGS API endpoints
- Add Python 3.7 support
- Add Ubuntu 16.04 LTS testing support
- Update dependencies to minimize upstream build errors

2.2.2 1.0.0 - March 15, 2018

- Initial release

2.3 Configuration

SLUGS uses the CherryPy configuration system to manage both global and application level configuration settings.

By default, SLUGS will look for a `slugs.conf` configuration file in `/etc/slugs/` when first starting up. This file path can be changed using the `-c` option:

```
$ python slugs/app.py -c /path/to/config/file
```

The same flag can be used with the `slugs` entry point:

```
$ slugs -c /path/to/config/file
```

The following is an example `slugs.conf` file, which can be found under the `examples/` directory in the SLUGS repository.

```
[global]
environment = 'production'
server.socket_host = '127.0.0.1'
server.socket_port = 8080
log.access_file = '/var/log/cherrypy/slugs/access.log'
log.error_file = '/var/log/cherrypy/slugs/error.log'

[data]
user_group_mapping = '/etc/slugs/user_group_mapping.csv'

[/slugs]
tools.trailing_slash.on = True
```

2.3.1 Global Settings

The `[global]` configuration block contains site-wide configuration settings that will apply to every application mounted via CherryPy. The SLUGS setup assumes that SLUGS will be the only CherryPy application running on the host machine.

The different configuration options are defined below. For more information on these CherryPy settings, see CherryPy Configuration.

- **`environment`** A string indicating the type of environment hosting the application. Tells CherryPy to load in additional preset configuration settings appropriate for the environment. Should be set to `'production'` when running SLUGS in production or commented out when in development. For more information, see CherryPy Environments.
- **`server.socket_host`** The IP address of the host machine running the application.
- **`server.socket_port`** The port number on which to host the application.

  Note: SLUGS must have permission to bind to the specified port, specifically if the port is a privileged port.

- **`log.access_file`** The path to the access log file. This log contains entries for all external accesses to the application (e.g., all GET requests).

  Note: The log directory must exist before SLUGS is run; the service will not create the log directory for you. SLUGS must also have permission to access the log directory.

- **`log.error_file`** The path to the error log file. This log contains entries pertaining to the startup, maintenance, and shutdown of the application, including any errors that may occur during the lifetime of the application.

  Note: The log directory must exist before SLUGS is run; the service will not create the log directory for you. SLUGS must also have permission to access the log directory.

2.3.2 Application Settings

The SLUGS application is configured with two different configuration blocks: `[data]` and `[/slugs]`. The `[data]` block contains custom configuration settings that define the data sources SLUGS should use to serve user/group information.

- **`user_group_mapping`** The path to the CSV file containing user/group data in `user,group` format. See *Data Management* for more information.

  Note: The CSV file must exist before SLUGS is run; the service will not create the CSV file for you. SLUGS must also have permission to access the directory containing the CSV file.

The `[/slugs]` block is an application-level block that contains additional CherryPy settings for the SLUGS application.

- **`tools.trailing_slash.on`** A boolean flag that allows CherryPy to redirect incoming requests to a URL without a trailing / to the same URL with a trailing /. A `301` redirect message will be logged in `log.access_file` when this redirect occurs.

2.3.3 Data Management

The user/group information served by SLUGS is stored in a backing CSV file that is configured on application startup (see `user_group_mapping` above). The following is an example CSV file, which can be found under the `examples/` directory in the SLUGS repository.

```
John,Human
Jane,Human
John,Male
Jane,Female
```

In this example, there are two users `John` and `Jane`. Each belongs to two different groups, both belonging to the `Human` group, but each belonging to the `Male` and `Female` groups respectively.

User and group names can contain additional characters, like whitespaces and symbols. The following example is still a valid CSV file.

```
John Doe,Blood Type: AB-
Jane Doe,Blood Type: O+
```

The only user/group naming restriction is that neither can contain the delimiting character `,`. Blank lines can be included throughout the file; they are simply ignored. Lines starting with a `#` are considered comments and are also ignored. Extra whitespace at the beginning or ending of a user or group name is treated similarly:

```
John,    Male
   Jane,Female
```

The users in the above example are still `John` and `Jane`, not `John` and `___Jane`. The groups are still `Male` and `Female`, not `____Male` and `Female`.

Finally, the backing CSV file can be edited and updated while SLUGS is running. The application will automatically detect the change and reload the data file. A log message acknowledging this data update will be logged in `log.error_file` when the reload occurs.

```
[timestamp] ENGINE Monitored file (<path/here>) updated. Reloading data.
```

If an error occurs during data reload, SLUGS will stop processing the new data and will retain the prior data set it was serving. This allows data updates to be made to SLUGS without potentially breaking the application. A log message acknowledging this data update error will be logged in `log.error_file` when the error is detected.

```
[timestamp] ENGINE Error parsing monitored file (<path/here>). Halting
data reload.
```

2.4 Development

Development for SLUGS is open to all contributors. Use the information provided here to inform your contributions and help the project maintainers review and accept your work.

2.4.1 Getting Started

File a new issue on the project issue tracker on GitHub describing the work you intend on doing. Provide as much information on your feature request as possible.

The issue number for your new issue should be included at the end of the commit message of each patch related to that issue.

If you simply want to request a new feature but do not intend on working on it, file your issue as normal and the project maintainers will triage it for future work.

2.4.2 Writing Code

New code should be written in its own Git branch, ideally branched from `HEAD` on `master`. If other commits are merged into `master` after your branch was created, be sure to rebase your work on the current state of `master` before submitting a pull request to GitHub.

New code should generally follow `PEP 8` style guidelines, though there are exceptions that will be allowed in special cases. Run the `flake8` tests to check your code before submitting a pull request (see *Running Tests*).

To prepare your local Python environment for SLUGS development, install the project requirements:

```
$ pip install -r requirements.txt
```

2.4.3 Writing Documentation

Like new code, new documentation should be written in its own Git branch. All SLUGS documentation is written in RST format and managed using `sphinx`. It can be found under `docs/source`.

If you are interested in contributing to the project documentation, install the project documentation requirements:

```
$ pip install -r doc-requirements.txt
```

To build the documentation, navigate into the `docs` directory and run:

```
$ make html
```

This will build the SLUGS documentation as HTML and place it under the new `docs/build/html` directory. View it using your preferred web browser.

2.4.4 Commit Messages

Commit messages should include a single line title (75 characters max) followed by a blank line and a description of the change, including feature details, testing and documentation updates, feature limitations, known issues, etc.

The issue number for the issue associated with the commit should be included at the end of the commit message, if it exists. If the commit is the final one for a specific issue, use `Closes #XXX` or `Fixes #XXX` to link the issue and close it simultaneously.

2.4.5 Bug Fixes

If you have found a bug in SLUGS, file a new issue and use the title format `Bug: <brief description here>`. In the body of the issue please provide as much information as you can, including Python version, SLUGS version, operating system version, and any stacktraces or logging information produced by SLUGS related to the bug. See What to put in your bug report for a breakdown of bug reporting best practices.

If you are working on a bug fix for a bug in `master`, follow the general guidelines above for branching and code development (see *Writing Code*).

If you are working on a bug fix for an older version of SLUGS, your branch should be based on the latest commit of the repository branch for the version of SLUGS the bug applies to (e.g., branch `release-1.0.0` for SLUGS 1.0). The pull request for your bug fix should also target the version branch in question. If applicable, it will be pulled forward to newer versions of SLUGS, up to and including `master`.

2.4.6 Running Tests

SLUGS uses `tox` to manage testing across multiple Python versions. Test infrastructure currently supports Python 2.7, 3.4, 3.5, and 3.6. Additional test environments are provided for security, style, and documentation checks.

Note: All of the `tox` commands discussed in this section should be run from the root of the SLUGS repository, in the same directory as the `tox.ini` configuration file.

The style checks leverage `flake8` and can be run like so:

```
$ tox -e pep8
```

The security checks use `bandit` and can be run like so:

```
$ tox -e bandit
```

The documentation checks leverage `sphinx` to build the HTML documentation in a temporary directory, verifying that there are no errors. These checks can be run like so:

```
$ tox -e docs
```

To run the above checks along with the entire unit test suite, simple run `tox` without any arguments.

```
$ tox
```

Unit Tests

The unit test suite tests each individual component of the SLUGS code base, verifying that each component works correctly in isolation. Ideal code coverage would include the entire code base. To facilitate improving coverage, test coverage results are included with each Python unit test environment.

To test against a specific Python version (e.g., Python 2.7), run:

```
$ tox -e py27
```

Integration Tests

The integration test suite tests the REST API provided by SLUGS, verifying that the right response data and response status codes are returned for specific queries. An instance of SLUGS must already be running and serving the `examples/user_group_mapping.csv` data file for the integration test cases to pass.

Code base coverage is not a goal of the integration test suite. Code coverage statistics are therefore not included in the output of the integration tests. For code coverage, run the unit tests above.

To run the integration test suite, the URL to the SLUGS instance must be passed to the test suite using the `--url` configuration argument. Assuming the SLUGS URL is `http://127.0.0.1:8080/slugs`, the following `tox` command will set up and execute the integration tests:

```
$ tox -r -e integration -- --url http://127.0.0.1:8080/slugs
```

For more information on the testing tools used here, see the following resources:

- bandit
- flake8
- sphinx
- tox

2.5 Installation

You can install SLUGS via `pip`:

```
$ pip install slugs
```

2.5.1 Supported platforms

SLUGS is tested on Python 2.7, 3.4, 3.5, and 3.6 on the following operating systems:

- Ubuntu 12.04, 14.04, and 16.04

2.5.2 Building SLUGS on Linux

You can install SLUGS from source via `git`:

```
$ git clone https://github.com/openkmip/slugs.git
$ cd slugs
$ python setup.py install
```

If you are on a fresh Linux build, you may also need several additional system dependencies, including headers for Python.

Ubuntu

Replace `python-dev` with `python3-dev` if you are using Python 3.0+.

```
$ sudo apt-get install python-dev
```