Aletheia Documentation

Release 0.1.0

Christopher Petrilli

Apr 24, 2017

Contents

Contents:

CHAPTER 1

Aletheia

Manage secrets in Google Cloud Platform

- Free software: BSD license
- Documentation: https://aletheia.readthedocs.io.

Features

- TODO

Coming Soon

This is what is planned:

- Refactoring of Chests and Secrets
- Command line to manage Chests and Secrets, and introspect audit data
- Clearer instructions and best-practices around how to organize projects, chests, and secrets.
- Tests? Need to figure out how to test GCP stuff better. Maybe use `get_kms_client` mocking.

Credits

Inspired by a post by mediocrity about building an app on Google AppEngine. This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

Stable release

To install Aletheia, run this command in your terminal:

```
$ pip install aletheia
```

This is the preferred method to install Aletheia, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for Aletheia can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/petrilli/aletheia
```

Or download the tarball:

```
$ curl  -OL https://github.com/petrilli/aletheia/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Quickstart

- We create a different key for every single secret. While this might be a bit more than we really need, it does allow for clear auditing of every single access.

Steps

1. Create project (you may want to put the storage buckets in a different proejct).
2. Create users
   - **User 1: Manager**
     - `roles/cloudkms.admin`
   - User 2: Consumer - `roles/cloudkms.cryptoKeyEncrypterDecrypter`, or if you want them to have even more limited access: `roles/cloudkms.cryptKeyDecrypter`.
3. Create CloudStorage bucket - Create a bucket: `gsutil mb -p [PROJECT_NAME] -c [STORAGE_CLASS] -l [BUCKET_LOCATION] gs://[BUCKET_NAME]/` By default you probably want `multi_regional` for your bucket. In that case you will want to set the bucket location to `us`, `asia`, or `eu` so it's closest to most of your infrastructure. - Grant User 1 `roles/storage.objectAdmin` - Grant User 2 `roles/storage.objectViewer`
4. Create a KeyRing for managing all the secret keys:

```
gcloud kms keyrings create aletheia --location global
```

5. Finally, we need to create a key for the project. It should share the name of your project for ease of discovery:

```
gcloud kms keys create project-1234 --location global --keyring aletheia --
↪purpose encryption
```

CHAPTER 4

Usage

To use Aletheia in a project:

```
import aletheia
```

CHAPTER 5

Cost Management

The cost of using the Aletheia library is composed a two different components:

1. Cloud Storage costs
2. Cloud KMS keys and operations

All calculations are based on the current pricing.

Cloud Storage

The cost of Google Cloud Storage is dependent on how redundant the data is. If we assume we want the highest level of availability and durability for all secrets, then a secret, average 1kiB would cost $0.000000026 per month, and $0.000004 per retrieval, or effectively zero.

Cloud KMS

Cloud KMS has two costs for it's use. The first is a per-key cost per month, and the other is the actual operations that are performed. The project costs can be calculated as follows:

- $0.06/mo per chest
- $.000003/operation

Total Cost

Assuming the project has 20 secrets, and has to unwrap them ten times a day, the cost of using Aletheia would be:

    0.06 + (20 * (0.000000026 + (30 * (10 * (0.000003 + $0.000004))))) = $0.10200052

Or, just more than 10 cents per month at list prices.

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/petrilli/aletheia/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

Aletheia could always use more documentation, whether as part of the official Aletheia docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/petrilli/aletheia/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *aletheia* for local development.

1. Fork the *aletheia* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/aletheia.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv aletheia
$ cd aletheia/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 aletheia tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.5, and 3.6. Check https://travis-ci.org/petrilli/aletheia/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_aletheia
```

CHAPTER 7

Credits

Development Lead

- Christopher Petrilli <petrilli@amber.org>

Contributors

- Drew Rothstein <drothstein@twitter.com> The original inspiration for the project.

CHAPTER 8

History

0.1.0 (2017-04-09)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search