Aumbry Documentation

Release 0.8.0

John Vrbanac

Oct 06, 2018

Contents

Aumbry is general purpose library for handling configuration within your Python applications. The project was born from constantly needing a simple interface for configuration models that came from multiple data sources.

Behind the scenes, Aumbry uses Alchemize to handle the conversion of the configuration data into application specific data models for your project.

CHAPTER 1

Installation

Aumbry is available on PyPI

```
# Install aumbry core
pip install aumbry

# For Consul dependencies
pip install aumbry['consul']

# For Etcd2 dependencies
pip install aumbry['etcd2']

# For Yaml dependencies
pip install aumbry['yaml']
```

Contents:

1.1 Using Aumbry

1.1.1 Dependencies

Many developers are very conscious of the number of dependencies that they include in their projects. To that end, Aumbry doesn't install the dependencies for parsing yaml or loading from consul by default. However, Aumbry attempts to make this relatively easy on users by enabling users to easily install the extra dependencies using the following convention:

```
# For Consul dependencies
pip install aumbry['consul']

# For Etcd2 dependencies
pip install aumbry['etcd2']
```

(continues on next page)

(continued from previous page)

```
# For Yaml dependencies
pip install aumbry['yaml']

# For Parameter Store dependencies
pip install aumbry['param_store']

# For Fernet File dependencies
pip install aumbry['fernet']

# Installing multiple dependencies
pip install aumbry['etcd2','yaml']
```

1.1.2 Loading from a File

One of the simplest and most common way of loading configuration is from a file. For this example, we'll use a JSON configuration file.

Lets say we have the following JSON configuration that we want to load

```
{
    "something": "it works!"
}
```

The next steps are to define a configuration class that matches what we're trying to do and load the config up.

```
import aumbry


class SampleConfig(aumbry.JsonConfig):
    __mapping__ = {
        'something': aumbry.Attr('something', str),
    }


# You can either specify the options here or via environment variables
options = {
    'CONFIG_FILE_PATH': './my_config.json',
}

# Time to load it up!
config = aumbry.load(aumbry.FILE, SampleConfig, options)

print(config.something) # it works!
```

File Options

Like all options, these can be manually specified when calling `load()` or via environment variables.

Key	Default	Notes
CONFIG_FILE_PATH		Required

Encryption

Encryption and decryption support is provided by using pyca/cryptography's Fernet module. Installing the required dependencies can be done with:

```
pip install aumbry['fernet']
```

The usage is nearly identical to a standard file; however, the source type and options change slightly. The source type becomes `aumbry.FERNET` and you need to provide the `CONFIG_FILE_FERNET_KEY` option.

1.1.3 Loading from Consul

As mentioned under the Dependencies section, the dependencies to load from consul are not included by default. As a result, we need to first install our extra dependencies.

```
pip install aumbry['consul']
```

Much like our loading from a file example, we need a configuration class and set our options for the Consul source.

```
import aumbry


class SampleConfig(aumbry.JsonConfig):
    __mapping__ = {
        'something': aumbry.Attr('something', str),
    }


# You can either specify the options here or via environment variables
options = {
    'CONSUL_URI': 'http://myhost:8500',
    'CONSUL_KEY': 'test',
}

# Time to load it up!
config = aumbry.load(aumbry.CONSUL, SampleConfig, options)

print(config.something) # it works!
```

It is important to note that the Consul source will block until it either cannot load, reaches max retries, or successfully loads.

Consul Options

Like all options, these can be manually specified when calling `load()` or via environment variables.

Key	Default	Notes
CONSUL_URI		Required
CONSUL_KEY		Required
CONSUL_TIMEOUT	10	Timeout per-request
CONSUL_RETRY_MAX	1	Number of retries to attempt
CONSUL_RETRY_INTERVAL	10	Wait period between retries

1.1.4 Loading from Etcd2

As mentioned under the Dependencies section, the dependencies to load from etcd2 are not included by default. As a result, we need to first install our extra dependencies.

```
pip install aumbry['etcd2']
```

Much like our loading from a file example, we need a configuration class and set our options for the Etcd2 source.

```
import aumbry


class SampleConfig(aumbry.JsonConfig):
    __mapping__ = {
        'something': aumbry.Attr('something', str),
    }


# You can either specify the options here or via environment variables
options = {
    'ETCD2_URI': 'http://myhost:8500',
    'ETCD2_KEY': 'test',
}

# Time to load it up!
config = aumbry.load(aumbry.ETCD2, SampleConfig, options)

print(config.something) # it works!
```

It is important to note that the Etcd2 source will block until it either cannot load, reaches max retries, or successfully loads.

Etcd2 Options

Like all options, these can be manually specified when calling `load()` or via environment variables.

Key	Default	Notes
ETCD2_URI		Required
ETCD2_KEY		Required
ETCD2_TIMEOUT	10	Timeout per-request
ETCD2_RETRY_MAX	1	Number of retries to attempt
ETCD2_RETRY_INTERVAL	10	Wait period between retries

1.1.5 Loading from AWS Parameter Store

As mentioned under the Dependencies section, the dependencies to load from the parameter store are not included by default. As a result, we need to first install our extra dependencies.

```
pip install aumbry['param_store']
```

To use the parameter store functionality, we need to use the generic configuration class or force the usage of the generic handler on `load()` and `save()`.

```
import aumbry


class SampleConfig(aumbry.GenericConfig):
    __mapping__ = {
        'something': aumbry.Attr('something', str),
    }


# You can either specify the options here or via environment variables
options = {
    'PARAMETER_STORE_AWS_REGION': 'us-west-2',
    'PARAMETER_STORE_PREFIX': '/prod/my_app',
}

# Time to load it up!
config = aumbry.load(aumbry.PARAM_STORE, SampleConfig, options)

print(config.something) # it works!
```

Note: If you need to mix configuration types, such as using a `YamlConfig`, you'll need to tell Aumbry to attempt to coerce the configuration using the *aumbry.formats.generic.GenericHandler* when calling *aumbry.load()* and *aumbry.save()*.

Parameter Store Options

Like all options, these can be manually specified when calling `load()` or via environment variables.

Key	Default	Notes
PARAMETER_STORE_AWS_REGION		Required
PARAMETER_STORE_PREFIX		Required
PARAMETER_STORE_AWS_ACCESS_ID		If empty, the default machine credentials are used
PARAMETER_STORE_AWS_ACCESS_SECRET		If empty, the default machine credentials are used
PARAMETER_STORE_AWS_SESSION_TOKEN		If empty, the default machine credentials are used
PARAMETER_STORE_AWS_KMS_KEY_ID	Account Default	

1.1.6 Building Configuration Models

Because Aumbry uses Alchemize for model de/serialization, it's just a matter of defining out the models in the Alchemize method.

Example Yaml Configuration

```
---
base-uri: http://localhost
database:
  servers:
```

(continues on next page)

(continued from previous page)

```
    - localhost:5432
  username: postgres
  password: something
  name: app
```

Example Code Load and Parse that config

```
import aumbry
from aumbry import Attr


class DatabaseConfig(aumbry.YamlConfig):
    __mapping__ = {
        'servers': Attr('servers', list),
        'username': Attr('username', str),
        'password': Attr('password', str),
        'database': Attr('database', str),
    }


class AppConfig(aumbry.YamlConfig):
    __mapping__ = {
        'base-uri': Attr('base_uri', str),
        'database': Attr('database', DatabaseConfig),
    }


cfg = aumbry.load(
    aumbry.FILE,
    AppConfig,
    {
        'CONFIG_FILE_PATH': '/etc/app/config.yml'
    }
)

print(cfg.database.username) # postgres
```

One of the things you might have noticed is that the explicit mapping allows for us to take an attribute name such as `base-uri` which isn't compatible with Python, and map it over to `base_uri`.

More details can be found on building your mappings in the Alchemize documentation.

1.2 API Documentation

aumbry.**FILE**
str – Alias of SourceTypes.file

aumbry.**CONSUL**
str – Alias of SourceTypes.consul

class aumbry.**Attr**(*attr_name*, *attr_type*, *serialize=True*, *required=False*, *coerce=None*)
Attribute Definition

Parameters

- **name** – Python attribute name

- **type** – Attribute type (e.g str, int, dict, etc)
- **serialize** – Determines if the attribute can be serialized
- **required** – Forces attribute to be defined
- **coerce** – Forces attribute to be coerced to its type (primitive types)

aumbry.**load**(*source_name*, *config_class*, *options=None*, *search_paths=None*, *preprocessor=None*, *handler=None*)

Loads a configration from a source into the specified Config type

Parameters

- **source_name** (*str*) – The name of the desired source.
- **config_class** (*AumbryConfig*) – The resulting class of configuration you wish to deserialize the data into.
- **options** (*dict, optional*) – The options used by the source handler. The keys are determined by each source handler. Refer to your source handler documentation on what options are available.
- **search_paths** (*list, optional*) – A list paths for custom source handlers
- **preprocessor** (*function*) – A function that pre-processes the source data before loading into the configuration object.
- **handler** (*AbstractHandler*) – An instance of a handler to process the configuration data.

Returns An instance of the passed in config_class

aumbry.**save**(*source_name*, *config_inst*, *options=None*, *search_paths=None*, *preprocessor=None*, *handler=None*)

Loads a configration from a source into the specified Config type

Parameters

- **source_name** (*str*) – The name of the desired source.
- **config_inst** (*AumbryConfig*) – The instance of a configuration class wish save.
- **options** (*dict, optional*) – The options used by the source handler. The keys are determined by each source handler. Refer to your source handler documentation on what options are available.
- **search_paths** (*list, optional*) – A list paths for custom source handlers
- **preprocessor** (*function*) – A function that pre-processes the configration data before saving to the source.
- **handler** (*AbstractHandler*) – An instance of a handler to process the configuration data. Defaults to the configuration handler.

class aumbry.**JsonConfig**

A type of AumbryConfig for JSON Configurations.

class aumbry.**YamlConfig**

A type of AumbryConfig for Yaml Configurations.

class aumbry.**GenericConfig**

A type of AumbryConfig for Generic Dict Configurations.

class aumbry.**SourceTypes**

Used to specified the source type to load a configuration.

```
consul = 'consul'
etcd2 = 'etcd2'
fernet = 'fernet'
file = 'file'
parameter_store = 'parameter_store'
```

1.2.1 Format Handlers

class aumbry.formats.generic.**GenericHandler**

deserialize(*raw_config*, *config_cls*)
Method to handle deserialization to a Config object.

serialize(*config*)
Method to handle serialization to a string.

class aumbry.formats.yml.**YamlHandler**

deserialize(*raw_config*, *config_cls*)
Method to handle deserialization to a Config object.

serialize(*config*)
Method to handle serialization to a string.

class aumbry.formats.js.**JsonHandler**

deserialize(*raw_config*, *config_cls*)
Method to handle deserialization to a Config object.

serialize(*config*)
Method to handle serialization to a string.

1.3 CLI Documentation

Warning: This is an unstable feature of aumbry. Use with discretion!

1.3.1 Installation

The Aumbry command-line interface is available as an extra requirement available on PyPI.

```
pip install aumbry[cli]
```

1.3.2 Usage

```
usage: aumbry [-h] {upload,edit,view} ...

CLI Tool for Aumbry

positional arguments:
  {upload,edit,view}
    upload       Uploads a configuration file
    edit         Edits a configuration file
    view         Displays a configuration file

optional arguments:
  -h, --help     show this help message and exit
```

Upload

The upload sub-command allows for you to push up a configuration.

```
aumbry upload \
    --file-type yml \
    --param-store-region us-east-1 \
    --param-store-prefix /my/aws/prefix \
    ./path/to/my/config.yml \
    my.aumbry.config:ConfigClass \
    parameter_store
```

Edit

The edit sub-command enabled you to open up your configuration file.

```
aumbry edit ./path/to/my/config.yml
```

View

The view sub-command prints out your configuration file to stdout. This feature is usually used in conjunction with encrypted configuration files.

```
aumbry view ./path/to/my/config.yml
```

1.3.3 Encrypted Configuration

Encryption and Decryption of configuration happens using Cryptography's Fernet capability. To use this functionality, provide your key via the `--fernet-key` cli option.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index