flask-s3 Documentation

Release 0.3.1

Edward Robinson

July 21, 2016

Contents

Flask-S3 allows you to easily serve all your Flask application's static assets from Amazon S3, without having to modify your templates.

CHAPTER 1

How it works

Flask-S3 has two main functions:

1. Walk through your application's static folders, gather all your static assets together, and upload them to a bucket of your choice on S3;
2. Replace the URLs that Flask's `flask.url_for()` function would insert into your templates, with URLs that point to the static assets in your S3 bucket.

The process of gathering and uploading your static assets to S3 need only be done once, and your application does not need to be running for it to work. The location of the S3 bucket can be inferred from Flask-S3 *settings* specified in your Flask application, therefore when your application is running there need not be any communication between the Flask application and Amazon S3.

Internally, every time `url_for` is called in one of your application's templates, *`flask_s3.url_for`* is instead invoked. If the endpoint provided is deemed to refer to static assets, then the S3 URL for the asset specified in the `filename` argument is instead returned. Otherwise, *`flask_s3.url_for`* passes the call on to `flask.url_for`.

CHAPTER 2

Installation

If you use pip then installation is simply:

```
$ pip install flask-s3
```

or, if you want the latest github version:

```
$ pip install git+git://github.com/e-dard/flask-s3.git
```

You can also install Flask-S3 via Easy Install:

```
$ easy_install flask-s3
```

2.1 Dependencies

Aside from the obvious dependency of Flask itself, Flask-S3 makes use of the boto library for uploading assets to Amazon S3. **Note**: Flask-S3 currently only supports applications that use the jinja2 templating system.

CHAPTER 3

Using Flask-S3

Flask-S3 is incredibly simple to use. In order to start serving your Flask application's assets from Amazon S3, the first thing to do is let Flask-S3 know about your `flask.Flask` application object.

```
from flask import Flask
from flask_s3 import FlaskS3

app = Flask(__name__)
app.config['FLASKS3_BUCKET_NAME'] = 'mybucketname'
s3 = FlaskS3(app)
```

In many cases, however, one cannot expect a Flask instance to be ready at import time, and a common pattern is to return a Flask instance from within a function only after other configuration details have been taken care of. In these cases, Flask-S3 provides a simple function, `init_app`, which takes your application as an argument.

```
from flask import Flask
from flask_s3 import FlaskS3

s3 = FlaskS3()

def start_app():
    app = Flask(__name__)
    s3.init_app(app)
    return app
```

In terms of getting your application to use external Amazon S3 URLs when referring to your application's static assets, passing your `Flask` object to the `FlaskS3` object is all that needs to be done. Once your app is running, any templates that contained relative static asset locations, will instead contain hosted counterparts on Amazon S3.

3.1 Uploading your Static Assets

You only need to upload your static assets to Amazon S3 once. Of course, if you add or modify your existing assets then you will need to repeat the uploading process.

Uploading your static assets from a Python console is as simple as follows.

```
>>> import flask_s3
>>> from my_application import app
>>> flask_s3.create_all(app)
>>>
```

Flask-S3 will proceed to walk through your application's static assets, including those belonging to *registered* blueprints, and upload them to your Amazon S3 bucket.

3.1.1 Static Asset URLs

Within your bucket on S3, Flask-S3 replicates the static file hierarchy defined in your application object and any registered blueprints. URLs generated by Flask-S3 will look like the following:

`/static/foo/style.css` becomes `https://mybucketname.s3.amazonaws.com/static/foo/style.css`, assuming that `mybucketname` is the name of your S3 bucket, and you have chosen to have assets served over HTTPS.

3.1.2 Setting Custom HTTP Headers

To set custom HTTP headers on the files served from S3 specify what headers you want to use with the `FLASKS3_HEADERS` option.

```
FLASKS3_HEADERS = {
    'Expires': 'Thu, 15 Apr 2010 20:00:00 GMT',
    'Cache-Control': 'max-age=86400',
}
```

See Yahoo! more information on how to set good values for your headers.

3.2 Flask-S3 Options

Within your Flask application's settings you can provide the following settings to control the behaviour of Flask-S3. None of the settings are required, but if not present, some will need to be provided when uploading assets to S3.

CHAPTER 4

API Documentation

Flask-S3 is a very simple extension. The few exposed objects, methods and functions are as follows.

4.1 The FlaskS3 Object

class flask_s3.**FlaskS3**(*app=None*)

The FlaskS3 object allows your application to use Flask-S3.

When initialising a FlaskS3 object you may optionally provide your flask.Flask application object if it is ready. Otherwise, you may provide it later by using the *init_app()* method.

Parameters **app** (flask.Flask or None) – optional flask.Flask application object

init_app(*app*)

An alternative way to pass your flask.Flask application object to Flask-S3. *init_app()* also takes care of some default *settings*.

Parameters **app** – the flask.Flask application object.

4.2 S3 Interaction

flask_s3.**create_all**(*app*, *user=None*, *password=None*, *bucket_name=None*, *location=None*, *include_hidden=False*, *filepath_filter_regex=None*, *put_bucket_acl=True*)

Uploads of the static assets associated with a Flask application to Amazon S3.

All static assets are identified on the local filesystem, including any static assets associated with *registered* blueprints. In turn, each asset is uploaded to the bucket described by bucket_name. If the bucket does not exist then it is created.

Flask-S3 creates the same relative static asset folder structure on S3 as can be found within your Flask application.

Many of the optional arguments to *create_all* can be specified instead in your application's configuration using the Flask-S3 *configuration* variables.

Parameters

- **app** – a flask.Flask application object.
- **user** (basestring or None) – an AWS Access Key ID. You can find this key in the Security Credentials section of your AWS account.

- **password** (basestring or None) – an AWS Secret Access Key. You can find this key in the Security Credentials section of your AWS account.
- **bucket_name** (basestring or None) – the name of the bucket you wish to server your static assets from. **Note**: while a valid character, it is recommended that you do not include periods in bucket_name if you wish to serve over HTTPS. See Amazon's bucket restrictions for more details.
- **location** (basestring or None) – the AWS region to host the bucket in; an empty string indicates the default region should be used, which is the US Standard region. Possible location values include: `'DEFAULT'`, `'EU'`, `'USWest'`, `'APSoutheast'`
- **include_hidden** (bool) – by default Flask-S3 will not upload hidden files. Set this to true to force the upload of hidden files.
- **filepath_filter_regex** (basestring or None) – if specified, then the upload of static assets is limited to only those files whose relative path matches this regular expression string. For example, to only upload files within the 'css' directory of your app's static store, set to r'^css'.
- **put_bucket_acl** (bool) – by default Flask-S3 will set the bucket ACL to public. Set this to false to leave the policy unchanged.

flask_s3.**url_for**(*endpoint*, ***values*)

Generates a URL to the given endpoint.

If the endpoint is for a static resource then an Amazon S3 URL is generated, otherwise the call is passed on to `flask.url_for`.

Because this function is set as a jinja environment variable when *`FlaskS3.init_app`* is invoked, this function replaces `flask.url_for` in templates automatically. It is unlikely that this function will need to be directly called from within your application code, unless you need to refer to static assets outside of your templates.

f

C

F

I

U