nacelle Documentation

Release 0.4.1

Patrick Carey

August 16, 2014

Contents

nacelle is a lightweight Python web framework, built on top of webapp2, designed for use on Google Appengine.

nacelle aims to provide a small but solid set of tools that enable developers to quickly get a new app up and running, whilst not sacrificing any of the flexibility and power of webapp2 in the process. Nacelle is suitable for building everything from tiny prototypes to large complex applications, it should never get in your way.

Note: If you need portability from appengine then nacelle probably isn't for you, use Django, Flask or one of the many other awesome Python web frameworks that exist, you'll thank me later.

nacelle is Free Software, released under the MIT license.

CHAPTER 1

Standing on the shoulders of giants

nacelle wouldn't be possible without the hard work of others and the fantastic libraries upon which it builds. **If you have a problem with nacelle, and you can't find the answer in these docs, you should try the documentation of nacelle's components** by following the links below.

- **webapp2:** A lightweight Python web framework compatible with Google Appengine's webapp. webapp2 is the main foundation for most of nacelle's WSGI/web functionality.
- **Jinja2:** A modern and designer friendly templating language for Python, modelled after Django's templates. nacelle includes a preconfigured Jinja2 environment ready for you to use.

This documentation won't try to cover absolutely everything, instead it aims to cover only those bits of nacelle that differ significantly from webapp2, pointing back to the original webapp2 docs where appropriate.

CHAPTER 2

Contents

2.1 Getting Started

nacelle depends on a few external external libraries for operation, however, we have a policy that nacelle core should never have a hard dependency on any library which is not available with the Google Appengine Python SDK, therefore getting started is easy.

You'll first need to install the App Engine Python SDK. See the README file for directions. You'll need python 2.7 and pip 1.4 or later installed too.

Appengine requires that all external libraries being used should be included (vendored) into the application directory itself. Different developers have their own methods for dealing with this issue, and nacelle should work perfectly fine with whatever your preferred application structure is (if it doesn't, let us know). This article covers one such method, aiming to provide a quick overview of what it takes to get started with nacelle.

2.1.1 New Project Skeleton

The nacelle project provides a barebones skeleton you can use to get your application off the ground quickly. You should clone the skeleton's git repository to a convenient location:

```
$ git clone https://github.com/nacelle/nacelle-skeleton.git your-project-name
```

Once cloned, you should use pip to install the latest version of nacelle into the application's `vendor` directory:

```
$ cd your-project-name
$ pip install -r requirements.txt -t vendor
```

Note: App Engine can only import libraries from inside your project directory.

You should now be able to run your project locally from the command line using the regular Appengine SDK tools:

```
$ dev_appserver.py .
```

Visit the application http://localhost:8080

See the development server documentation for options when running dev_appserver.py.

2.1.2 Deploying your application

To deploy your application you first need to ensure you've created a project using the appengine Admin Console. Once created, you can deploy your application with the following command:

```
$ appcfg.py -A <your-project-id> --oauth2 update .
```

Congratulations! Your application is now live at your-project-id.appspot.com

2.1.3 Installing additional libraries

See the Third party libraries page for libraries that are already included in the SDK. To include SDK libraries, add them in your app.yaml file. Other than libraries included in the SDK, only pure python libraries may be added to an App Engine project.

To install additional libraries from pypi you can use pip, e.g. to install the raven library for reporting errors to a sentry server, simply run:

```
$ pip install raven -t vendor
```

2.2 User Guide

You can find (well, you'll be able to soon) all you need to know about nacelle in the sections below.

2.2.1 URL Routing

The URL router is the central part of a nacelle application, mapping incoming URLs to the appropriate handlers.

Routes Configuration

By default, nacelle looks for `ROUTES` (should be an iterable containing `webapp2.Route` objects as in webapp2 itself) in the `app` module. This location can be configured using nacelle's `settings.py` as such:

```
# Python dotted path to the routes for the app
ROUTES_MODULE = 'app.ROUTES'  # default routes module
```

For example, if your list of configured routes was defined as `routes` in the `apps.core.routes` module, you could update your `settings.py` to include the line:

```
# Python dotted path to the routes for the app
ROUTES_MODULE = 'apps.core.routes.routes'  # custom routes module
```

Simple Routing

For the most part, **nacelle uses webapp2's routing infrastructure**, and you should consult those docs when you have any questions or issues.

When a request comes in, the application will match the request path to find the corresponding handler. If no route matches, an `HTTPException` is raised with status code 404, and the WSGI application can handle it accordingly.

As a simple example, assuming the default configuration, the following could be defined in *app.py* and would map 3 different URL patterns to 3 different request handlers:

```
ROUTES = [
    webapp2.Route(r'/', handler=HomeHandler, name='home'),
    webapp2.Route(r'/products', handler=ProductListHandler, name='product-list'),
    webapp2.Route(r'/products/<product_id>', handler=ProductHandler, name='product'),
]
```

The first argument in the routes above is a URL template, the *handler* argument is the request handler to be used (can also be a string in dotted notation to be lazily imported when needed), and the *name* argument third is a name used to build a URI for that route.

Multi-prefix routes

The webapp2_extras.routes provides several classes to wrap routes that share common characteristics:

- `webapp2_extras.routes.PathPrefixRoute`: receives a url path prefix and a list of routes that start with that prefix.
- `webapp2_extras.routes.HandlerPrefixRoute`: receives a handler module prefix in dotted notation and a list of routes that use that module.
- `webapp2_extras.routes.NamePrefixRoute`: receives a handler name prefix and a list of routes that start with that name.

The intention is to avoid repetition when defining routes. nacelle takes this concept one step further and provides a single route class that allows combining all three types of built-in `PrefixRoute`.

For example, imagine we have these routes:

```
from webapp2 import Route

ROUTES = [
    Route('/users/<user:\w+>/', 'users.UserOverviewHandler', 'user-overview'),
    Route('/users/<user:\w+>/profile', 'users.UserProfileHandler', 'user-profile'),
    Route('/users/<user:\w+>/projects', 'users.UserProjectsHandler', 'user-projects'),
]
```

We could refactor them to use common prefixes:

```
from nacelle.core.routes import MultiPrefixRoute
from webapp2 import Route

ROUTES = [
    MultiPrefixRoute(
        handler_pfx='users.',
        name_pfx='user-',
        path_pfx='/users/<user:\w+>',
        routes=[
            Route('/', 'UserOverviewHandler', 'overview'),
            Route('/profile', 'UserProfileHandler', 'profile'),
            Route('/projects', 'UserProjectsHandler', 'projects'),
        ],
    )
]
```

This is not only convenient, but also performs better: the nested routes will only be tested if the path prefix matches.

2.2.2 Settings

Nacelle comes with some default settings that you can override to configure your app. These can be found in `nacelle/conf/default_settings.py`

Create a file named `settings.py` in the root of your project folder and override the settings there

2.2.3 HTTP basic authentication

Sometimes you need to protect your site to only let certain people access it, whether it be for a staging site or a client preview or simply because you have personal information that you dont want just anyone to see. Nacelle makes this really easy to do

Add the following to your user settings file:

```
DISPATCHER_MODULE = 'nacelle.contrib.lockdown.dispatcher.lockdown_dispatcher'
LOCKDOWN_USERNAME = 'foo'
LOCKDOWN_PASSWORD = 'bar'
LOCKDOWN_URL_EXCEPTIONS = []
```

While http basic authentication is handy for quickly restricting access to a site you should make use of the API's provided by App Engine for production sites.

In basic authentication, the username and password are transmitted as plain-text to the server. This makes basic authentication un-suitable for applications without SSL, as you would end up exposing sensitive passwords.

2.3 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:

2.3.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/nacelle/nacelle/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” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

NacelleDocs could always use more documentation, whether as part of the official nacelle 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/nacelle/nacelle/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 :)

2.3.2 Get Started!

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

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

   ```
   $ git clone git@github.com:your_name_here/nacelle.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 nacelle
   $ cd nacelle/
   $ pip install -r requirements.txt
   ```

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 the tests:

   ```
   $ make test
   ```

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.

2.3.3 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 (the latest supported Python version on Google Appengine). Check https://travis-ci.org/nacelle/nacelle/pull_requests and make sure that the tests pass for all supported versions.

2.4 Credits

2.4.1 Authors

- Patrick Carey <patrick@rehabstudio.com>

Why not submit a pull request and add your name to this list?

2.4.2 Documentation

As nacelle is largely derived from webapp2, so it follows that a large part of its documentation has been derived from there too. A lot of nacelle's documentation has been copied verbatim from webapp2's. Some docs have examples changed to suit nacelle's style, and sections added or removed where appropriate.

webapp2 documentation by Rodrigo Moraes.

2.5 History

2.5.1 0.4.1 (2014-08-16)

- Minor bugfix release
- Added missing default setting.

2.5.2 0.4.0 (2014-08-15)

- The slow march towards 1.0 continues. Mostly a bugfix release.

2.5.3 0.3.0 (2014-05-17)

- First release on PyPI.