baldrick Documentation

Release 0.2

Stuart Mumford and Thomas Robitaille

Nov 22, 2018

Contents

Baldrick is a Python package that provides a framework to set up a GitHub bot with minimal code and effort. If you run into any issues, have requests for improvement, or would like to contribute, our GitHub repository is here

CHAPTER 1

Getting started with building your bot

We provide a simple template for the files needed to set up your bot at https://github.com/OpenAstronomy/baldrick/tree/master/template. We take a look here at the minimal set of files required:

1.1 run.py

This is the main file that defines how you want your bot to behave. First, set up the bot using:

```
from baldrick import create_app
app = create_app('<your-bot-name>')
```

Then, optionally import any plugins you want to have available, including custom plugins if you have developed any additional ones. The available plugins are:

```
import baldrick.plugins.circleci_artifacts
import baldrick.plugins.github_milestones
import baldrick.plugins.github_pull_requests
import baldrick.plugins.github_towncrier_changelog
```

And finally use the following to start up the bot:

```
import os
port = int(os.environ.get('PORT', 5000))
app.run(host='0.0.0.0', port=port, debug=False)
```

1.2 pyproject.toml

This file can be used to enable/disable any of the plugins that are available by default. See *Available plugins and configuration* for more details.

1.3 Procfile

This should simply contain:

```
web: python -m run
```

and shouldn't need to be modified further.

1.4 runtime.txt

This file specifies the Python runtime to use for your bot, for example:

```
python-3.6.5
```

Note that this should be Python 3.6 or later.

1.5 requirements.txt

This provides a list of packages required for your bot, and should include at the very least:

```
baldrick
```

1.6 Other files

Of course, don't forget to include a README file and a LICENSE!

CHAPTER 2

Available plugins and configuration

This page lists the available plugins. Note that to enable a plugin, your bot app should include an `enabled = true` entry in the `pyproject.toml` file under the section for the specific plugin.

2.1 CircleCI Artifacts

The CircleCI service provides the option of storing build artifacts. The baldrick plugin will automatically post the link to the artifacts as a status check in a GitHub pull request to avoid having to click through multiple pages to find the link to the artifacts. To enable this plugin, include the following in your `pyproject.toml` file:

```
[ tool.<your-bot-name>.circleci_artifacts ]
enabled = true
```

You can then include additional sub-sections in the configuration for each set of artifacts, for example:

```
[ tool.<your-bot-name>.circleci_artifacts.sphinx ]
url = "html/index.html"
message = "This is the documentation"
```

The `url` item should be set to the file path of the artifacts, and the message is what will be shown in the status check.

2.2 Pull request handlers

We provide a plugin that will perform checks on a pull request and report the results back to the pull request, either as a comment and a single status check, or individual status checks. Which checks are done are themselves plugins and will be described in subsequent sections.

To enable pull request handlers, include the following in your `pyproject.toml` file:

```
[ tool.<your-bot-name>.pull_requests ]
enabled = true
```

In addition, you can use the following configuration items if you wish to change the default behavior:

- `post_pr_comment = false/true`: if `true`, the results of the checks will be summarized in a comment, and a single overall status check will be reported. If `false`, each check will be reported as a separate status check. The default is `false`.
- `skip_labels = []`: this can be set to a list of GitHub labels which, if present, will cause the checks to be skipped. Note that labels are case-sensitive. The default is an empty list.
- `skip_fails = false/true`: if `true`, if the checks are skipped due to `skip_labels`, then a failed status check will be posted to the pull request. If `false`, the checks will be silently skipped. The default is `true`.

By default, the comment/statuses posted by the bot should be informative, but if you wish to change the wording of these messages, you can override them with the following parameters - note that all these only apply when `post_pr_comment` is `true`:

- `skip_message = "..."`: the message to display in a comment if the checks are skipped due to `skip_labels`
- `fail_prologue = "..."` and `fail_epilogue = "..."`: the text to include before and after the results of the checks in the comment.
- `fail_status = "..."` and `pass_status = "..."`: the message to show in the overall status check.
- `all_passed_message = "..."`: the message to show in a comment if all checks passed.
- `pull_request_substring`: a string that can be used to identify previous comments posted by the bot. This should be a string common to `all_passed_message`, and `fail_prologue` or `fail_epilogue`.

2.2.1 GitHub milestone checker

This pull request handler plugin checks whether the milestone has been set. To enable this plugin, include the following in your `pyproject.toml` file:

```
[ tool.<your-bot-name>.milestones ]
enabled = true
```

If you wish to customize the message shown in the results of the check, you can use the `missing_message = "..."` and `present_message = "..."` configuration items.

2.2.2 Towncrier changelog checker

Another built-in pull request handler plugin can be used to check that towncrier changelog changes in a pull request are consistent with other details about the pull request (e.g. the pull request number). To enable this plugin, include the following in your `pyproject.toml` file:

```
[ tool.<your-bot-name>.towncrier_changelog ]
enabled = true
```

This plugin has the following additional configuration items:

- `verify_pr_number = true`: whether to check that the name of the towncrier file added is consistent with the pull request number.
- `changelog_skip_label = "..."`: the name of a GitHub label which, if present, causes the towncrier changelog checks to be skipped.

- `help_url = "..."`: this can be set to the URL to use for the status check 'Details' link - you can set this to a URL explaining how to use towncrier for example.

By default, the comment/statuses posted by the bot should be informative, but if you wish to change the wording of these messages, you can override them with the following parameters:

- `changelog_exists = "..."` and `changelog_missing = "..."`: the messages to use when a changelog entry exists or is missing.
- `number_correct = "..."` and `number_incorrect = "..."`: the messages to use when a changelog entry has the correct or incorrect pull request number.
- `type_correct = "..."` and `type_incorrect = "..."`: the messages to use when a changelog entry is not of the right type.

2.2.3 Custom plugin

If you want to write your own pull request checker, import `pull_request_handler` from baldrick as follows:

```
from baldrick.plugins.github_pull_requests import pull_request_handler
```

then use it to decorate a function of the form:

```
@pull_request_handler
def check_changelog_consistency(pr_handler, repo_handler):
    ...
```

This function will be called with `pr_handler`, an instance of *`PullRequestHandler`*, and `repo_handler`, an instance of *`RepoHandler`* (click on the class names to find out the available properties/methods).

Your function should then return either *None* (no check results), or a dictionary where each key is the code name for one of the checks (this will be used to match checks with previous checks, so make sure this is consistent across calls), and the value should be a dictionary with two entries: `state`, which can be set to `'failure'` or `'success'`, and `description`, which gives a description of the check results.

CHAPTER 3

Setting up an app on Heroku

Once you have an app ready to go using baldrick, you can deploy it to any server you want. Here we provide instructions on setting it up on Heroku.

To start off, create a free account on Heroku if you don't already have one. When you see the option to create a new app, select it (ignore the "add to pipeline" option). Give a name to your app; You need to select a name that is not already taken and it does not have to be the same as the bot's name here.

You should now be on the "Deploy" section. Again, ignore the pipeline option. Select Github as "Deployment Method". Enter the relevant GitHub organization or account that the bot resides in (this should be automatically populated if you have given Heroku access to your GitHub account) and type in the bot's repository name (either this bot or a forked version of it).

If you want to enable automatic deployment from a selected branch of the repository, click the "Enable Automatic Deploys" button. This will pick up changes to the given branch and re-deploy the bot as needed. For most cases, you don't need the "wait for CI to pass before deploy" option as the bot is already tested here.

For the first time, you also need to manually deploy the bot by clicking "Deploy Branch".

Once it is successfully deployed, and once you have followed the instructions to add the app to GitHub (see *Registering and installing a GitHub app*) go to "Settings" tab of the app on Heroku and you can customize its behavior using "Config Vars". This is the only custom configuration on Heroku and can be set through the Heroku admin interface, as mentioned. The main required environment variables (also see "Authentication" section below) are:

- `GITHUB_APP_INTEGRATION_ID`, which should be set to the integration ID provided by GitHub app (see "GitHub settings" section below) under "General Settings", specifically "About… ID". This is a numerical integer value.
- `GITHUB_APP_PRIVATE_KEY`, which is generated by the GitHub app (see "GitHub settings" section below). This private key should look like:

  ` -----BEGIN RSA PRIVATE KEY----- <some random characters> -----END RSA PRIVATE KEY----- `

  The whole key, including the `BEGIN` and `END` header and footer should be pasted into the field.

CHAPTER 4

Registering and installing a GitHub app

4.1 Registering the app

Once you have set up the bot on a server (e.g. *Setting up an app on Heroku*), you will need to tell GitHub about the app. To add the bot to your own organization or account, go to your GitHub organization or account URL (not the repository) and then its settings. Then, click on "Developer settings" at the very bottom of the left navigation bar and the "New GitHub App" button on top right.

Give your bot a "GitHub App name" as you want it to appear on GitHub activities. Under "Homepage URL", enter the GitHub repository URL where the bot code resides (either here or your fork, as appropriate).

For the **User authorization callback URL**, it should be in the format of `http://<heroku-bot-name>.herokuapp.com/installation_authorized`.

For the **Webhook URL**, it should be in the format of `http://<heroku-bot-name>.herokuapp.com/github`.

You can ignore "Setup URL" and "Webhook secret". It would be useful to provide a description of what your bot intends to do but not required.

The permissions of the app should be read/write access to **Commit statuses**, **Issues**, and **Pull requests**. Once you have checked these options, you will see extra "Subscribe to events" entries that you can check as well. For the events, it should be sufficient to only check **Status**, **Issue comment**, **Issues**, **Pull request**, **Pull request review**, and **Pull request review comment**.

It is up to you to choose whether you want to allow your GitHub app here to be installed only on your account or by any user or organization.

Once you have clicked "Create GitHub App" button, you can go back to the app's "General" settings and upload a logo, which is basically a profile picture of your bot.

4.2 Install the bot

Go to `https://github.com/apps/<github-app-name>`. Then, click on the big green "Install" button. You can choose to install the bot on all or select repositories under your account or organization. It is recommended to only install it for select repositories by start typing a repository name and let auto-completion do the hard work for you (repeat this once per repository). Once you are done, click "Install".

After a successfull installation, you will be taken to a `https://github.com/settings/installations/<installation-number>` page. This page is also accessible from your account or organization settings in "Applications", specifically under "Installed GitHub Apps". You can change the installation settings by clicking the "Configure" button next to the listed app, if desired.

CHAPTER 5

Trying out components of the bot locally

5.1 GitHub API

The different components of the bot interact with GitHub via a set of helper classes that live in `baldrick.github`. These classes are *`RepoHandler`*, `IssueHandler`, and *`PullRequestHandler`*. It is possible to try these out locally, at least for the parts of the GitHub API that do not require authentication. For example, the following should work:

```
>>> from baldrick.github.github_api import RepoHandler, IssueHandler,␣
↪PullRequestHandler
>>> repo = RepoHandler('astropy/astropy')
>>> repo.get_issues('open', 'Close?')
[6025, 5193, 4842, 4549, 4058, 3951, 3845, 2603, 2232, 1920, 1024, 435, 383, 282]
>>> issue = IssueHandler('astropy/astropy', 6597)
>>> issue.labels
['Bug', 'coordinates']
>>> pr = PullRequestHandler('astropy/astropy', 6606)
>>> pr.labels
['Enhancement', 'Refactoring', 'testing', 'Work in progress']
>>> pr.last_commit_date
1506374526.0
```

However since these are being run un-authenticated, you may quickly run into the GitHub public API limits. If you are interested in authenticating locally, see the *Authenticating locally* section below.

5.2 Authenticating locally

In some cases, you may want to test the bot locally as if it was running on Heroku. In order to do this you will need to make sure you have all the environment variables described above set correctly.

The main ones to get right as far as authentication is concerned are as follows (see *Setting up an app on Heroku* for further details):

- GITHUB_APP_INTEGRATION_ID
- GITHUB_APP_PRIVATE_KEY

The last thing you will need is an **Installation ID** - a GitHub app can be linked to different GitHub accounts, and for each account or organization, it has a unique ID. You can find out this ID by going to **Your installations** and then clicking on the settings box next to the account where you have a test repository you want to interact with. The URL of the page you go to will contain the Installation ID and look like:

> https://github.com/settings/installations/36238

In this case, 36238 is the installation ID. Provided you set the environment variables correctly, you should then be able to do e.g.:

```
>>> from baldrick.github.github_api import IssueHandler
>>> issue = IssueHandler('astrofrog/test-bot', 5, installation=36238)
>>> issue.submit_comment('I am alive!')
```

Note: Authentication will not work properly if you have a `.netrc` file in your home directory, so you will need to rename this file temporarily.

CHAPTER 6

API documentation

6.1 baldrick.github.github_api Module

Module to handle GitHub API.

6.1.1 Classes

GitHubHandler(repo[, installation])	A base class for things that represent things the github app can operate on.
RepoHandler(repo[, branch, installation])	
PullRequestHandler(repo, number[, installation])	

GitHubHandler

class baldrick.github.github_api.**GitHubHandler**(*repo*, *installation=None*)

Bases: object

A base class for things that represent things the github app can operate on.

Methods Summary

get_config_value(cfg_key[, cfg_default, branch])	Convenience method to extract user configuration values.
get_file_contents(path_to_file[, branch])	
get_repo_config([branch, path_to_file, ...])	Load configuration from the repository.
invalidate_cache()	
list_statuses(commit_hash)	List status messages on a commit on GitHub.

Continued on next page

Table 2 – continued from previous page

set_status(state, description, context, ...)	Set status message on a commit on GitHub.

Methods Documentation

get_config_value(*cfg_key*, *cfg_default=None*, *branch=None*)
Convenience method to extract user configuration values.

Values are extracted from the repository configuration, and if not defined, they are extracted from the global app configuration. If this does not exist either, the value is set to the `cfg_default` argument.

get_file_contents(*path_to_file*, *branch=None*)

get_repo_config(*branch=None*, *path_to_file='pyproject.toml'*, *warn_on_failure=True*)
Load configuration from the repository.

Parameters

- **branch** (*str*) – The branch to read the config file from. (Will default to 'master')
- **path_to_file** (*str*) – Path to the `pyproject.toml` file in the repository. Will default to the root of the repository.
- **warn_on_failure** (*bool*) – Emit warning on failure to load the pyproject file.

Returns **cfg** – Configuration parameters.

Return type *baldrick.config.Config*

invalidate_cache()

list_statuses(*commit_hash*)
List status messages on a commit on GitHub.

Parameters **commit_hash** (*str*) – The commit has to get the statuses for

set_status(*state*, *description*, *context*, *commit_hash*, *target_url=None*)
Set status message on a commit on GitHub.

Parameters

- **state** (*{ 'pending' | 'success' | 'error' | 'failure' }*) – The state to set for the pull request.
- **description** (*str*) – The message that appears in the status line.
- **context** (*str*) – A string used to identify the status line.
- **commit_hash** (*str*) – The commit hash to set the status on.
- **target_url** (str or *None*) – Link to bot comment that is relevant to this status, if given.

RepoHandler

class baldrick.github.github_api.**RepoHandler**(*repo*, *branch='master'*, *installation=None*)
Bases: *baldrick.github.github_api.GitHubHandler*

Methods Summary

get_all_labels()	Get all label options for this repo
get_file_contents(path_to_file[, branch])	
get_issues(state, labels[, exclude_pr])	Get a list of issues.
open_pull_requests()	

Methods Documentation

get_all_labels()
Get all label options for this repo

get_file_contents(*path_to_file*, *branch=None*)

get_issues(*state*, *labels*, *exclude_pr=True*)
Get a list of issues.

Parameters

- **state** (*{'open', ..}*) – Status of the issues.
- **labels** (*str*) – List of comma-separated labels; e.g., `Closed?`.
- **exclude_pr** (*bool*) – Exclude pull requests from result.

Returns **issue_list** – A list of matching issue numbers.

Return type list

open_pull_requests()

PullRequestHandler

class baldrick.github.github_api.**PullRequestHandler**(*repo*, *number*, *installation=None*)
Bases: `baldrick.github.github_api.IssueHandler`

Attributes Summary

base_branch
base_sha
head_branch
head_repo_name
head_sha
json
last_commit_date
milestone
user

Methods Summary

get_file_contents(path_to_file[, branch])	Get the contents of a file.
get_modified_files()	Get all the filenames of the files modified by this PR.
get_repo_config([branch, path_to_file, ...])	Load user configuration for bot.

Continued on next page

Table 5 – continued from previous page

has_modified(filelist)	Check if PR has modified any of the given list of filename(s).
list_statuses([commit_hash])	List status messages on a commit on GitHub.
set_status(state, description, context[, ...])	Set status message on a commit on GitHub.
submit_review(decision, body)	Submit a review comment to the pull request

Attributes Documentation

base_branch

base_sha

head_branch

head_repo_name

head_sha

json

last_commit_date

milestone

user

Methods Documentation

get_file_contents(*path_to_file*, *branch=None*)
Get the contents of a file.

This will get the file from the head branch of the PR by default.

get_modified_files()
Get all the filenames of the files modified by this PR.

get_repo_config(*branch=None*, *path_to_file='pyproject.toml'*, *warn_on_failure=True*)
Load user configuration for bot.

Parameters

- **branch** (*str*) – The branch to read the config file from. (Will default to the base branch of the PR i.e. the one the PR is opened against.)
- **path_to_file** (*str*) – Path to the `pyproject.toml` file in the repository. Will default to the root of the repository.
- **warn_on_failure** (*bool*) – Emit warning on failure to load the pyproject file.

Returns **cfg** – Configuration parameters.

Return type dict

has_modified(*filelist*)
Check if PR has modified any of the given list of filename(s).

list_statuses(*commit_hash='head'*)
List status messages on a commit on GitHub.

Parameters **commit_hash** (*str, optional*) – The commit hash to set the status on. Defaults to "head" can also be "base".

set_status(*state*, *description*, *context*, *commit_hash='head'*, *target_url=None*)
Set status message on a commit on GitHub.

Parameters

- **state** (*{ 'pending' | 'success' | 'error' | 'failure' }*) – The state to set for the pull request.
- **description** (*str*) – The message that appears in the status line.
- **context** (*str*) – A string used to identify the status line.
- **commit_hash** (*str*) – The commit hash to set the status on. Defaults to "head" can also be "base".
- **target_url** (str or *None*) – Link to bot comment that is relevant to this status, if given.

submit_review(*decision*, *body*)
Submit a review comment to the pull request

Parameters

- **decision** (*{ 'approve' | 'request_changes' | 'comment' }*) – The decision as to whether to aprove or reject the changes so far.
- **body** (*str*) – The body of the review comment

6.2 baldrick.github.github_auth Module

6.2.1 Functions

get_app_name()	Return the login name of the authenticated app.
get_installation_token(installation)	Get access token for installation
get_json_web_token()	Prepares the JSON Web Token (JWT) based on the private key.
github_request_headers(installation)	
netrc_exists()	
repo_to_installation_id(repository)	Return the installation ID for a repository.
repo_to_installation_id_mapping()	Returns a dictionary mapping full repository name to installation id.

get_app_name

baldrick.github.github_auth.**get_app_name**()
Return the login name of the authenticated app.

get_installation_token

baldrick.github.github_auth.**get_installation_token**(*installation*)
Get access token for installation

get_json_web_token

baldrick.github.github_auth.**get_json_web_token**()
Prepares the JSON Web Token (JWT) based on the private key.

github_request_headers

baldrick.github.github_auth.**github_request_headers**(*installation*)

netrc_exists

baldrick.github.github_auth.**netrc_exists**()

repo_to_installation_id

baldrick.github.github_auth.**repo_to_installation_id**(*repository*)
Return the installation ID for a repository.

repo_to_installation_id_mapping

baldrick.github.github_auth.**repo_to_installation_id_mapping**()
Returns a dictionary mapping full repository name to installation id.

Python Module Index

b

Index

O

P

R

S

U