octomachinery Documentation

Release 0.3.8.dev59+gb36c3d3

Sviatoslav Sydorenko (@webknjaz)

Nov 04, 2023

CONTENTS:

How-to create a GitHub Bot tutorial is ready for preview @ tutorial.octomachinery.dev

CHAPTER

ONE

ELEVATOR PITCH

Here's how you a just-created comment:

```
from octomachinery.app.server.runner import run as run_app
from octomachinery.routing import process_event_actions
from octomachinery.routing.decorators import process_webhook_payload
from octomachinery.runtime.context import RUNTIME_CONTEXT


@process_event_actions('issue_comment', {'created'})
@process_webhook_payload
async def on_comment(
        *,
        action, issue, comment,
        repository=None, sender=None,
        installation=None,
        assignee=None, changes=None,
):
    github_api = RUNTIME_CONTEXT.app_installation_client
    comment_reactions_api_url = f'{comment["url"]}/reactions'
    await github_api.post(
        comment_reactions_api_url,
        preview_api_version='squirrel-girl',
        data={'content': '+1'},
    )


run_app(
    name='Thumbs-Up-Bot',
    version='1.0.0',
    url='https://github.com/apps/thuuuuuuuuuuuuuumbs-uuuuuuuuuuuup',
)
```

CHAPTER

TWO

PREREQUISITES

Python 3.7+

CHAPTER

THREE

CONTRIBUTE OCTOMACHINERY

Want to add something to upstream? Feel free to submit a PR or file an issue if unsure. Note that PR is more likely to be accepted if it includes tests and detailed description helping maintainers to understand it better

Oh, and be pythonic, please

Don't know how? Check out How to Contribute to Open Source article by GitHub

CHAPTER

FOUR

LICENSE

The source code and the documentation in this project are released under the GPL v3 license.

CHAPTER

FIVE

FOR ENTERPRISE

octomachinery is available as part of the Tidelift Subscription.

The octomachinery maintainers and the maintainers of thousands of other packages are working with Tidelift to deliver one enterprise subscription that covers all of the open source you use.

Learn more.

5.1 Getting started

The documentation isn't ready yet so I suggest you going through the How-to create a GitHub Bot tutorial which should give you basic understanding of GitHub Apps and how to write them with octomachinery.

5.1.1 Runtime pre-requisites

- Python 3.7+ as octomachinery relies on `contextvars` which doesn't have a backport.
- GitHub App credentials and GitHub Action events are supplied via environment variables. They are also loaded from a `.env` file if it exists in a development environment.

  Warning: Be aware that some tools in your environment may conflict with autoloading vars from a `.env` file. It is recommended to disable those. One example of such tool is Pipenv.

  For the production deployments, please use a way of supplying env vars via tools provided by the application orchestration software of your choice.

5.2 How-to guides

5.2.1 Running a one-off task against GitHub API

Sometimes you need to run a series of queries against GitHub API. To do this, initialize a token (it's taken from the `GITHUB_TOKEN` env var in the example below), construct a GitHub API wrapper and you are good to go.

`RawGitHubAPI` is a wrapper around interface provided by `GitHubAPI`, you can find the usage interface on its documentation page. Don't forget to specify a `user_agent` string — it's mandatory!

API calls return native Python `dict` or iterable objects.

```
import asyncio
import os

from octomachinery.github.api.tokens import GitHubOAuthToken
from octomachinery.github.api.raw_client import RawGitHubAPI


async def main():
    access_token = GitHubOAuthToken(os.environ["GITHUB_TOKEN"])
    github_api = RawGitHubAPI(access_token, user_agent='webknjaz')
    await github_api.post(
        '/repos/mariatta/strange-relationship/issues',
        data={
            'title': 'We got a problem',
            'body': 'Use more emoji!',
        },
    )


asyncio.run(main())
```

5.2.2 Authenticating as a bot (GitHub App)

To act as a bot, you should use a special kind of integration with GitHub — Apps. They are reusable entities in the GitHub Platform available to be installed into multiple accounts and organizations.

Classic GitHub App requires a web-sever part to be deployed somewhere on the Internet in order to receive events GitHub Platform would send there.

Yet, sometimes, you just want to act as a bot without any of that deployment hustle. You may want to have a `[bot]` label next to comments you'll post via API. You may want to manage rate limits better. This will allow you to run one-off tasks like batch changes/migrations.

You'll still need to register a GitHub App and install it into the target user account or organization. You'll also have to specify APIs you'd like to access using this App.

Then, you'll need to get your App's ID and a private key.

Now, first, specify the App ID, the key path and the target account. After that, create a `GitHubAppIntegrationConfig` instance also specifying app name, version and some URL (these will be used to generate a `User-Agent` HTTP header for API queries). Then, create a `GitHubApp` instance from that config. Retrieve a list of places where the App is installed, filter out the target Installation and get an API client for it. Finally, use `RawGitHubAPI` as usual.

```
import asyncio
import pathlib

from aiohttp.client import ClientSession

from octomachinery.github.api.app_client import GitHubApp
from octomachinery.github.config.app import GitHubAppIntegrationConfig


target_github_account_or_org = 'webknjaz'  # where the app is installed to
```

(continues on next page)

(continued from previous page)

```
github_app_id = 12345
github_app_private_key_path = pathlib.Path(
    '~/Downloads/star-wars.2011-05-04.private-key.pem',
).expanduser().resolve()

github_app_config = GitHubAppIntegrationConfig(
    app_id=github_app_id,
    private_key=github_app_private_key_path.read_text(),

    app_name='MyGitHubClient',
    app_version='1.0',
    app_url='https://awesome-app.dev',
)


async def get_github_client(github_app, account):
    github_app_installations = await github_app.get_installations()
    target_github_app_installation = next(  # find the one
        (
            i for n, i in github_app_installations.items()
            if i._metadata.account['login'] == account
        ),
        None,
    )
    return target_github_app_installation.api_client


async def main():
    async with ClientSession() as http_session:
        github_app = GitHubApp(github_app_config, http_session)
        github_api = await get_github_client(
            github_app, target_github_account_or_org,
        )
        user = await github_api.getitem(
            '/users/{account_name}',
            url_vars={'account_name': target_github_account_or_org},
        )
        print(f'User found: {user["login"]}')
        print(f'Rate limit stats: {github_api.rate_limit!s}')


asyncio.run(main())
```

5.2.3 Making API queries against preview endpoints

Endpoints with stable interfaces in GitHub API are easy to hit. But some are marked as preview API. For those, GitHub requires special Accept headers to be passed along with a normal HTTP request. The exact strings are documented at https://developers.github.com under specific endpoint sections in their description.

Given that you've already got an instance of `RawGitHubAPI` initialized, what's left is to pass `preview_api_version` argument with the appropriate preview API code name when making query to the API endpoint requiring that.

```
github_api: RawGitHubAPI

repo_slug = 'sanitizers/octomachinery'
issue_number = 15

await github_api.post(
    f'/repos/{repo_slug}/issues/{issue_number}/reactions',
    preview_api_version='squirrel-girl',
    data={'content': 'heart'},
)
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search