Bugflow Auto Gnome Documentation

Release 0.0.0

Team Bugflow

Sep 27, 2018

Contents:

CHAPTER 1

Introduction

Bugflow is a perspective on software development that focusses on the speed and efficiency with which bugs are created (as well as features, enhancements, and other more subjective descriptions of software changes)

The Bugflow auto-gnome is a mechanism for reducing the bugflow resistance by automating tedious tasks, freeing up developers to focus on creating new bugs.

It is inspired by the concept of a WikiGnome. The source code is available, in all it's buggy glory at https://github.com/bugflow/auto-gnome/

CHAPTER 2

Using a Gnome

You do not need to install or run this code to use it.

All you need to do is:

- create a .gnome.yml file in the root of your GitHub repository
- configure the "policies" you want the gnome to follow
- register the github web hook to the bugflow callback endpoint
- give the gnome appropriate permissions to allow it to implement your policies

2.1 The .gnome.yml file

The base directory of your hithub repo needs to have a file called .gnome.yml

TODO: insert screen shor of the bugflow repo, highlighting it's .gnome.yml

This needs to be a valid yaml file (TODO: link to the yaml web site)

The yaml file needs to contain a list of *policies*. The policies that are available in this repository are the ones we find useful, you can develop your own if you want to.

To ensure the GitHub Ticket Gnomes do the things you want them to do, the first thing you have to do is tell them what those things are.

Here is an example of a very simple *.gnome.yml* file:

```
policies:
 - VerboseCallbackLogging
 - NonExistantPolicy
 - SortingHat
```

This tells the gnome to do three things:

- Apply the VerboseCallbackLogging policy. This is a fairly silly thing for most users to do, because they can't see the logs that are produced. It may sometimes be usefull for DevOps crews working on the gnome itself. Basically, you are asking the gnome to mutter furiously to itself.
- Apply the NonExistantPolicy. There is no such policy, so this will cause another kind of silent muttering (of no use to most users, and possible use to DevOps crews working on the gnome itself).

- Apply the SortingHat policy. This is an example of something you might actually want to do. SortingHat policy is an example of a simple GitHub workflow automation.

Over time we might add more policies, including policies that you want to use. Or you can develop your own. Either way, the *.gnome.yml* file is how you tell your gnome what you want it to do in response to events in this repository.

2.2 The Web Hook

TODO:

- write some words about wiring-up the GitHub web hook.
- include a screenshot
- resolve if we want to make an app integration or not?
- resolve what to do about secrets / HMAC callback verification (etc)

At the moment, the gnome needs the user to manually configure a web hook in GitHub.

There may be bettwe ways to do this.

2.3 Available Policies

TODO: cog out the module docstring for each policy, with auto-generated headers

CHAPTER 3

Operating Gnomes

These are not garden variety gnomes, they live in the computers.

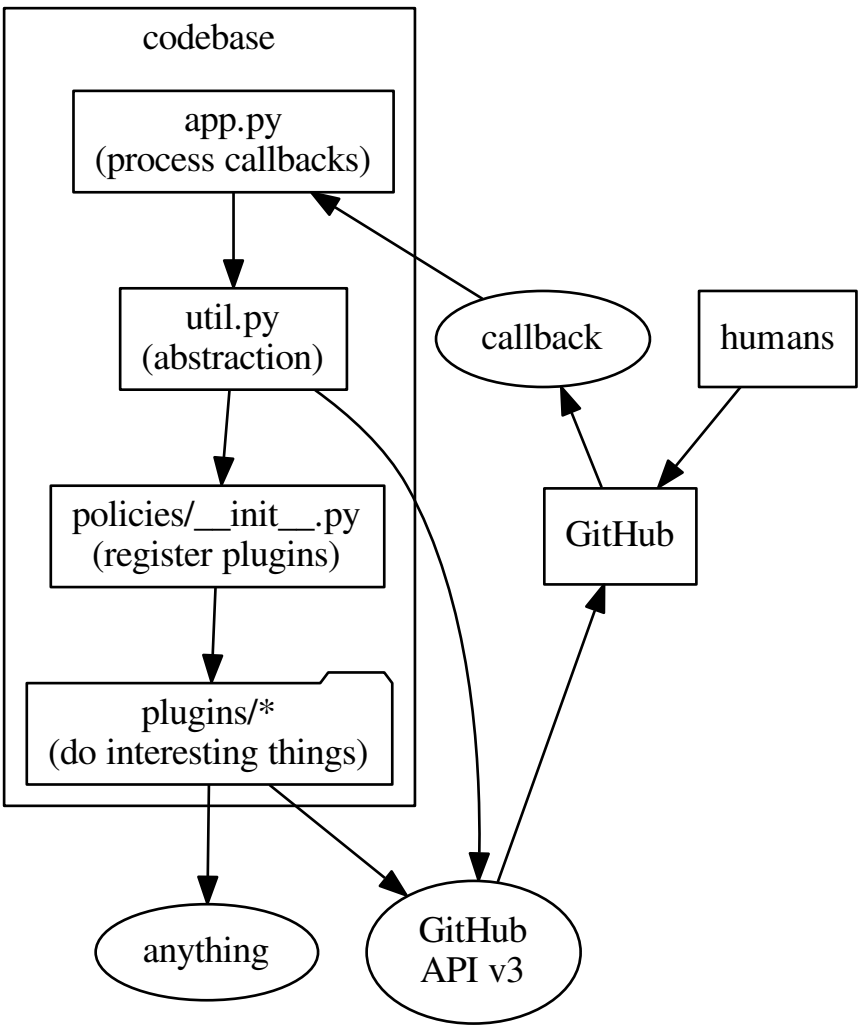
For now, got to Miserlou's Zappa web site and see all about zappa. That's how we are doing it now, but there's not much zappa-specific stuff (it's a simple web api, that mashes up other apis, written in python and making use of flask)

copy zappa_settings.json.example -> zappa_settings.json

Edit it to your liking. Make sure you have an AWS account set up (you can use the cli, right?)

CHAPTER 4

Hacking the gnome itself



4.1 Codebase

The callback service is provided by a Flask app, and the code for this is in *gnome/app.py*. It's only job is to receive callbacks from GitHub and process them.

app.py delegates the interesting stuff to code in *gnome/utils.py*. This does two things, interacts with GitHub to obtain configuration, then delegates configured tasks to the plugins (via the plugin register, *gnome/policies/__init__.py*)

Ultimately, interesting stuff is delegated to plugins. All plugins must provide a *dispatch_gnome()* method. If configured, this is called with data from the originating callback event (and config).

4.1.1 app.py

Flask API that processes callback messages from GitHub (or localhost). Messages are validated, then dispatched to all configured policies.

gnome.app.**index**()

4.1.2 gnome/utils.py

class util.**CallbackEvent**(*request*)

CallbackEvent is an abstraction over the raw flask request object. It provides convenience methods for validation and payload access.

headers()

is_valid()

payload()

class util.**Config**(*callback*)

Config is generated from the .gnome.yml file that is found in the root of a repository that is a source of GitHub callback events.

The .gnome.yml file is retrieved, parsed and validated. Then, the get_activities() method can be used to instantiate policy objects for everything that was configured in the repo.

get_activities()

This is the magic method. It processes the config (from .gnome.yml) and instantiates the policies, which are presumably dispatched.

get_yaml()

yaml_is_valid()

exception util.**InvalidPayloadJSONError**

4.1.3 gnome/gh.py

class gh.**EventSourceValidator**

GitHub publishes the address ranges that they make callbacks from.

Instances of this class can be used to validate ip addresses, like a kind of dymanic whitelist.

get_hook_blocks()

Fetch the whitelisted addresses blocks published by GitHub (directly, or from cache).

ip_str_is_valid(*ip_str*)

This function returns true if the IP address (string) passed to it is within the address blocks published by GitHub.

class gh.**Issue**(*repo*, *gh_issue*)
Wrapper of pygithub.Issue.Issue, with cache and convenience methods.

has_milestone()

move_to_milestone(*new_milestone*)

class gh.**Milestone**(*repo*, *milestone*)
Wrapper of pygithub.Milestone.Milestone with cache and convenience methods. Bound to a Repo instance for access to the Github connection (credentials etc).

description

due_on

number

open_tickets()

title

update(***kwargs*)

class gh.**Repo**(*repo_name*)
This class is an abstraction over the GitHub repository.

It interacts with GitHub as the configured user. Note this is a double-stacked abstraction (it's a wrapper around the PyGithub library, which wraps the GitHub API v3). That makes the code seem a little strange on first reading, however it simplifies mocking in tests at the business logic layer.

create_milestone(*milestone_name*, *state='open'*, *description=None*, *due_on=None*)
If the milestone does not exist, create it.

If (optional) date passed in, set that date on the milestone. Dito for description. Otherwise, both empty.

Returns the created (or pre-existing) Milestone instance.

ensure_milestone_exists(*milestone_name*, *description=None*, *date=None*)
If the repo does not have a milestone with the given name then create one.

If description or date parameters are provided, and the milestone is created, then they will be used.

If the milestone already exists, and the date or description differ from the ones provided, they will be ignored. This is NOT follow the "upsert" pattern.

ensure_milestone_has_due_date(*milestone_name*, *due_date*)
If the milestone does not have a due date, or if it has a due date that differs from the one provided, then update the due date to the one provided.

get_config()

get_milestone(*milestone_name*, *cache=True*)
Returns the milestone with by name (or None)

milestone_exists(*milestone_name*)
Returns True if the milestone exists.

milestones

update_milestones()

upsert_milestone(*title*, ***kwargs*)

gh.**repo_from_callback**(*callback*)

4.1.4 gnome/policies/__init__.py

The util module instantiates the *policy* module. This is a very simple thing, all it does is import the relevant classes (from modules in the plugins directory).

When you make a new plugin, it won't do anything until you register it by importing the relevant class into policies/__int__.py

Browse from there the plugins (see next section, Hacking Policies)...

4.2 Tests

CHAPTER 5

Plugin Development

Assuming everything is working, you probably want to hack on "policies". They are the things that do the stuff you want done. They are kind of like plugins. Users specify the policy they want to operate in their repo (using .gnome.yml), and you write the policy in python code that does what they want. Whenever the service get's a callback from GitHub, it "dispatches" all the configured policies. Simple.

5.1 Policy

class policies.**Policy**(*config*, *callback*)

Abstract base-class. Inherited by policies that actually do stuff.

Don't put this in your *.gnome.yml*, it's ignored.

dispatch_gnome()

The method that does the stuff you want done.

This method must be over-ridden in actual policies.

5.2 SortingHat

class policies.**SortingHat**(*config*, *callback*)

The "Sorting Hat" is a milestone with no due-date, that signifies some sort of ticket grooming/prioritisation process is necessary.

The sorting hat is like an in-tray for the person/people responsible for assessing and prioritising tickets.

```
As a software developer
I want my auto-gnome to use a sorting hat policy
So that my tickets are continuously groomed and prioritised
```

dispatch_gnome()

The method that does the stuff you want done.

This method must be over-ridden in actual policies.

5.3 VerboseCallbackLogging

class policies.**VerboseCallbackLogging**(*config*, *callback*)

This policy is very simple, it is primarially used for debugging the ticket gnome itself.

As a Gnome user, enabling this policy will achieve nothing because you don't have access to the logs it creates.

As a Gnome operator, you may find it useful for debugging but probably not.

As a Gnome developer, it serves as a canonical example of how to create a policy to be enacted by the Gnome. That is why there is so much more documentation than code.

dispatch_gnome()

You can consider the "dispatch_gnome" method like "main" method for gnomes. It is the only method every subclass of Policy requires, and is the method invoked in response to callback event from GitHub (if the repo is configured with this policy active).

Python Module Index

g

u

Index