labelbot

Release 0.0.6

Apr 18, 2019

Contents:

CHAPTER 1

Deployment instructions for AWS Lambda

1.1 Prerequisites

- You will need a Github account
- You will need an AWS account.

1.2 Creating the lambda function

1. Create a function on AWS Lambda and author from scratch. Under `choose or create an execution role`, choose `create a execution role from a policy template` with `Amazon S3 read only permissions` and name your role. Set your runtime to `Python 3.7` and choose a name for your function. press `Create function`
2. Choose `upload a .zip file` for your `Code entry type`
3. Set your handler to `labelbot.bot.lambda_handler`
4. Add an API Gateway as a trigger. Create a new API and set it to Open. Press add, and then save. This will give you a webhook url to add to your github app.

1.3 Create the github app

1. Under `Settings>Developer settings>Github Apps`, press the `New Github App` button.
2. Give your app a name and enter a homepage url, for example your fork of labelbot.
3. Under `Webhook url` enter your API gateway url.
4. Under `Webhook secret (optional)`, enter a secret token, as described in Githubs documentation.
5. Under `Permissions`, add `read-only` access to `Repository contents` and add `Read and write` access to `Issues`

6. Under `Subscribe to events`, subscribe to the `Label` event.
7. Under `Where can this GitHub App be installed?`, set `Only on this account`
8. Press the `Create Github App`
9. Generate a private key and save it to an S3 bucket that is not publicly accessible.

1.4 Set enviroment variables to hold private data in your Lambda function.

1. `APP_ID` : Shall be set to the App ID of your github app.
2. `BUCKET_NAME`: The name of your S3 bucket.
3. `BUCKET_KEY`: the unique identifier of your key file stored in S3.
4. `SECRET_KEY`: Shall be the same value as your secret token, that was set to secure the webhook.

After all enviroment variables have been added, save the changes.

1.5 Give your lambda function read access to your s3 bucket.

In `S3>:your bucket:>Permissions>Bucket Policy` give your lambda role the rights to read from your bucket.

1.6 Creating and uploading a deployment package

To create a deployment package it is required to build in a linux enviroment with python 3.7 and pip installed, as Lambda runs in a linux enviroment.

Run the `package.sh` script from the repository root, which packages the application iin a zip file, that can be uploaded to AWS to deploy the application. It will create a file called `labelbot.zip` which should be uploaded to your AWS lambda function. Save after uploading the file.

1.7 You have now created and deployed your own github app with AWS Lambda

You can now install it on your own account and use it to label issues by pressing the `Install App` button.

CHAPTER 2

Labelbot Module Reference

2.1 bot

Event handler for AWS lambda.

This is the main module of labelbot, and contains the event handler for AWS lambda. If for any reason one would like to use a different service than AWS lambda, this is the functionality that needs to be changed.

2.2 auth

Functions for handling authentication procedures.

`labelbot.auth.`**`authenticate_request`**(*shared_secret*, *body*, *signature*)

Checks if the MAC (message authentication code) sent in the request is really from GitHub.

Parameters

- **`shared_secret`** (`str`) – A secret shared between GitHub and the bot.
- **`body`** (`str`) – Body of the HTTP request.
- **`signature`** (`str`) – The header containing the MAC.

Return type `bool`

Returns True iff the signature is a MAC computed with the body of the request and the shared secret.

`labelbot.auth.`**`generate_installation_access_token`**(*jwt_token*, *installation_id*)

Generates an installation access token using a JWT token and an installation id.

An installation access token is valid for 1 hour.

Parameters

- **`jwt_token`** (`str`) – a valid JWT token

- **installation_id** – the installation id of the app.

Return type str

Returns An installation access token from the GitHub API

labelbot.auth.**generate_jwt_token**(*private_pem*, *app_id*)
Generates a JWT token valid for 10 minutes using the private key.

Parameters

- **private_pem** (bytes) – the private key that is used to generate a JWT
- **the Application id** (*app_id*) –

Return type str

Returns The JWT that was generated using the private key and the app id

labelbot.auth.**get_pem**(*bucket_name*, *bucket_key*)
Reads a private PEM file from an S3 bucket.

Parameters

- **bucket_name** (str) – Name of the S3 bucket.
- **bucket_key** (str) – Bucket key for the PEM file.

Return type bytes

Returns Contents of the PEM file.

2.3 github_api

Functions for interacting with the GitHub API.

exception labelbot.github_api.**APIError**
Raise when something goes wrong with the api.

labelbot.github_api.**get_file_contents**(*owner*, *repo*, *filepath*, *access_token*)
Fetch the contents of a file in the repo.

Parameters

- **owner** (str) – User/Organization that owns the repo.
- **repo** (str) – Name of the repo.
- **filepath** (str) – Path to the file from the repo root.
- **access_token** (str) – A API access token.

Return type str

Returns The contents of the the specified file.

labelbot.github_api.**get_labels**(*owner*, *repo*, *issue_nr*, *access_token*)
Get the labels from the given repository issue.

Parameters

- **owner** (str) – User/Organization that owns the repo.
- **repo** (str) – Name of the repo.
- **issue_nr** (int) – Number of the issue.

- **access_token** (str) – An installation access token for the repo.

Return type List[str]

Returns A list of labels.

labelbot.github_api.**set_allowed_labels**(*owner*, *repo*, *issue_nr*, *issue_body*, *current_labels*, *access_token*)

Set the current labels plus any requested labels in the issue body that are also allowed by the .allowed-labels file.

Parameters

- **owner** (str) – User/Organization that owns the repo.
- **repo** (str) – Name of the repo.
- **issue_nr** (int) – Number of the issue.
- **access_token** (str) – An installation access token for the repo.

Return type bool

labelbot.github_api.**set_labels**(*labels*, *owner*, *repo*, *issue_nr*, *access_token*)

Unconditionally set the provided labels on a repository issue.

Parameters

- **labels** (Iterable[str]) – A sequence of labels to set.
- **owner** (str) – User/Organization that owns the repo.
- **repo** (str) – Name of the repo.
- **issue_nr** (int) – Number of the issue.
- **access_token** (str) – An installation access token for the repo.

Return type bool

Returns True if the API request was succesful

2.4 parse

Parse label information from issue and config files.

labelbot.parse.**parse_allowed_labels**(*text*)

Parse the allowd labels from the contents of a .allowed-labels file. Each line constitutes a single label. Lines starting with a # sign are ignored.

Parameters **text** (str) – The contents of a .allowed-labels file.

Return type List[str]

Returns A list of defined labels

labelbot.parse.**parse_wanted_labels**(*text*)

Extract the labels defined by **:label:‘LABEL‘** tokens in a string.

Parameters **text** (str) – Typically an issue body with label markup.

Return type List[str]

Returns A list of extracted labels.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index