poule Documentation

Release 0.4.0

Arnaud Porterie (icecrime)

Jan 03, 2019

Contents

Poule helps you automate operations on GitHub issues and pull requests.

It allows implementing snippets of behavior (called operations) *once* and provides a way to invoke them in three different contexts:

1. As a one-time invocation, on the entire stock of GitHub items.
2. As part of a batch job alongside multiple other operations.
3. As part of a long-running daemon triggered by GitHub webhooks or scheduled.

The project was created to manage automation on the Moby project.

CHAPTER 1

Installation

Poule has not graduated to 1.0, so we don't do binary releases yet. In the meantime:

- Use the pre-built image from Docker Hub: the `latest` tag maps to the current state of the `master` branch, while individual tags exist for pre-releases (e.g., `0.4.0`).

  ```
  docker pull icecrime/poule:latest
  ```

- Build from source using with no other dependency but Docker.

  ```
  docker build -t poule https://github.com/icecrime/poule.git
  ```

CHAPTER 2

User guide

2.1 Introduction

2.1.1 Synopsis

```
NAME:
  poule - Mass interact with GitHub issues & pull requests

USAGE:
   poule [global options] command [command options] [arguments...]

VERSION:
   0.4.0

COMMANDS:
     batch     Run groups of commands described in files
     serve     Operate as a daemon listening on GitHub webhooks
     validate  Validate a Poule repository configuration file
     help, h   Shows a list of commands or help for one command

   Operations:
     ci-label-clean     Clean CI failure labels
     dco-check          Check DCO on pull requests
     label              Apply label(s) to items which title or body matches a pattern
     poule-updater      Update the poule configuration for the specified repository
     prune              Prune outdated issues
     random-assign      Assign items to a random username from the `users` list.
     rebuild            Rebuild configurations of a given state
     version-label      Apply version labels to issues
     version-milestone  Attach merged pull requests to the upcoming version's␣
↪milestone

GLOBAL OPTIONS:
```

(continues on next page)

(continued from previous page)

```
   --debug, -D          enable debug logging
   --dry-run            simulate operations
   --repository value   GitHub repository
   --token value        GitHub API token [$POULE_GITHUB_TOKEN]
   --token-file value   GitHub API token file [$POULE_GITHUB_TOKEN_FILE]
   --help, -h           show help
   --version, -v        print the version
```

2.1.2 Global options

Specifying a GitHub API token

A GitHub API token must be provided for poule to execute any modifying action (such as labeling an issue, or closing a pull request). The token can be specified:

- Directly by providing its value through the `--token` flag or the `$POULE_GITHUB_TOKEN` environment variable.
- Indirectly by providing the path to a file containing a token through the `--token-file` flag or the `$POULE_GITHUB_TOKEN_FILE` environment variable.

Simulating execution

When `--dry-run` is specified, poule retrieves GitHub issues and pull requests and calls operations as it normally would but doesn't actually *apply* the operations. Each operation will log as it is called, and what it would have done if applied.

Keep in mind that poule in dry run still issues the API calls necessary to retrieve GitHub data, and as a result contributes to consuming the GitHub's user API limit.

2.1.3 Running operations

Poule is all about running *Operations* on GitHub issues and pull requests. An operation is a snippet of GitHub automation, such as adding a label to items which body matches a given string. Once implemented, an operation can be reused in different contexts:

1. As a one-time invocation, on the entire stock of GitHub items.
2. As part of a batch job alongside multiple other operations.
3. As part of a long-running daemon triggered by GitHub webhooks or scheduled.

One-time invocation

Each operation gets surfaced in the command-line as its own subcommand, making the invocation of a one-off operation straightforward. All operations subcommand support the `--filter` flag which allows to restrict the items on which the operation will be applied. Additionally, each operation defines its own set of flags and its own input format: refer to the `--help` output for operation-specific information.

Batch execution

In batch execution, a collection of operations is described in YAML format. Similarly to the command-line invocation, each operation can be associated with a set of filters, as well as operation-specific settings.

Server mode

This is of course the most interesting mode, and deserves as such an entire documentation page: *Server mode*.

2.1.4 Configuring execution

Filtering

The following filter types are supported to restrict the set of items on which a given operation should be applied:

Type	Passes if	Values
age	Creation date > value	E.g.,: `2d`, `3w`, `4m`, `1Y`
assigned	Issue is assigned == value	`true` or `false`
comments	# comments matches predicate	E.g.,: `"=0"`, `">10"`, `"<20"`
labels	All specified labels are set	E.g.,: `"label1,label2"`
~labels	None of the specified labels are set	E.g.,: `"label1,label2"`
is	Type of item == value	`pr` or `issues`

All operations subcommands support the `--filter` with the following format:

```
--filter <filter_type_1>:<filter_value_1> [--filter <filter_type_n>:<filter_value_n> .
↪..]
```

When describing operation in YAML format (either for batch or server mode), filtering is defined as a `filters` mapping filter types to their respective values:

```
filters:
      <filter_type_1>: <filter_value_1>
      <filter_type_n>: <filter_value_n>
```

Note that sequences are used instead of comma separated values for the `labels` and `~labels` filters, for example:

```
--filter is:issue --filter label:bug --filter age:2d
```

Is expressed in YAML as the following:

```
filters:
  age:   2d
  is:    issue
  label: [ bug ]
```

2.2 Operations

2.2.1 Definition

An operation is a snippet of GitHub automation, for example: adding a label, closing a pull request, or commenting on an issue.

- Operations are idempotent, which means that they can safely be applied multiple times.
- An operation can apply to GitHub issues, pull requests, or both. For example, a `label` operation may know to operate independently on issues and pull requests, while a `rebuild` operation which triggers CI may only apply on pull requests.
- A catalog of builtin operations is provided and documented.

2.2.2 Builtin operations

Operation	Docker specific	Issues	Pull Requests	Purpose
`ci-label-clean`				Remove CI failures labels where necessary.
`dco-check`	.			Check for commit signatures, label and post a comment if missing.
`label`				Auto-label issues and pull requests according on matching regexps.
`poule-updater`				Reload `poule` configuration when a pull request modifies it.
`prune`				Manage issues with no activities.
`random-assign`				Auto-assign a random user to issues and pull requests.
`rebuild`				Rebuild all or selected pull request jobs.
`version-label`				Add a `version/x` label based on Docker version string in the body.
`version-milestone`				Add merged pull requests to the upcoming milestone.

More details on each operation can be found on GitHub.

2.2.3 Creating custom operations

Creating custom operations is not yet supported and requires modifying the project. However, issue icecrime/poule#4 is about adding support for Golang 1.8 plugins in such way that custom operations can be added at runtime.

2.3 Server mode

2.3.1 Main configuration

Listening for events

Using GitHub webhooks

Poule can listen on HTTP for incoming GitHub webhooks. Under this mode, the repository's webhook settings in GitHub must point to the publicly accessible URL of a poule server instance.

The following configuration elements are required:

- The `http_listen` address.
- The `http_secret` value which must correspond to the secret value specified in the repository configuration on GitHub.

Example configuration:

```
http_listen: ":80"
http_secret: "S3CR3T"

repositories:
  icecrime/poule: ""
```

Using NSQ

NSQ is a "realtime distributed messaging platform" which, in combination with crosbymichael/hooks, can be used to distribute GitHub events. Relying on a message queue for this use case has several advantages:

- Messages are persisted: events will be queued when poule is offline and will catch-up as soon as it gets back online.
- A single webhook endpoint in the repository's settings in GitHub can fan out messages to a variety of listeners through the messaging infrastructure.

Configuring poule to listens on NSQ requires several configuration elements:

1. The `nsq_lookupd` address.
2. The `nsq_channel` to subscribe to.
3. For each repository, the queue name to monitor.

Example configuration:

```
nsq_channel: "poule"
nsq_lookupd: "127.0.0.1:4161"

repositories:
  icecrime/poule: "hooks-poule"
```

2.3.2 Repository configuration

The server-mode configuration can contain both infrastructure-level settings (such as the NSQ configuration) and operations. However, having the entire configuration in a single file is impratical when managing a large collection of repositories.

In server mode, poule will look for a special `poule.yml` file at the root of each configured repository and load it as repository-specific configuration. This allows each individual repository and group of maintainers to manage their own set of rules. Furthermore, this allows to keep the central configuration private as it typically contains secret information.

Monitoring for updates

Repository-specific configurations will be loaded at poule startup. However, poule also provides a builtin `poule-updater` operation which looks for merged pull requests which either modify or add the special `poule.yml` file at the root of the repository.

When configured to be triggered on a pull request closed event, the operation will auto-refresh the configuration settings for the repository without having to restart the server. One possibily is to add this operation in the main configuration, hence covering all repositories:

```
common_configuration:

  # Poule updater watches for merged pull requests which modify the `poule.yml` file␣
↪at the root
  # of the repository, and takes these changes into account live.
  - triggers:
        pull_request:   [ closed ]
    operations:
        - type:         poule-updater
```

CHAPTER 3

Examples

3.1 One-time operations

Use the `label` operation to add label `bug` to issues which title or body matches the strings "panic" in repository `icecrime/poule`:

```
$ poule --repository icecrime/poule label --filter is:issue bug:panic
```

Use the `random-assign` operation to randomly assigns pull requests older than 2 weeks among 3 GitHub users in repository `icecrime/poule`:

```
$ poule --repository icecrime/poule random-assign --filter is:pr --filter age:2w␣
↪user1 user2 user3
```

3.2 Batch mode

A batch on repository `icecrime/poule` which combines both of the operations described above, and can together be executed in a single command.

```
$ cat poule-batch.yml
repository: icecrime/poule

operations:

  - type: random-assign
    filters:
      age: "2w"
      is:  "pr"
    settings:
      users: [ "user1", "user2", "user3" ]
```

(continues on next page)

(continued from previous page)

```
  - type: label
    filters:
      is: "issue"
    settings:
      patterns:
        bug: [ "panic" ]

$ poule batch poule-batch.yml
```

3.3 Server mode

A server configuration which listens on port 80 for incoming GitHub webhooks. It applies the `label` operation described above *live* as issues get edited, opened, or reopened. It also randomly assigns pull requests older than 2 weeks on a daily basis.

```
$ cat poule-server.yml
http_listen: ":80"
http_secret: "S3CR3T"

repositories:
  icecrime/poule: ""

common_configuration:

  - triggers:
      issues: [ edited, opened, reopened ]
    operations:
      - type: label
        settings:
          patterns:
            bug: [ "panic" ]

  - schedule: "@daily"
    operations:
      - type: random-assign
        filters:
          age: "2w"
          is:  "pr"
        settings:
          users: [ "user1", "user2", "user3" ]

$ poule serve --config poule-server.yml
```

CHAPTER 4

Contributing

- Repository: https://github.com/icecrime/poule/
- Issue tracker: https://github.com/icecrime/poule/issues