Greenstalk Documentation

Release 2.0.0

Justin Mayhew

Jun 07, 2021

CONTENTS

Greenstalk is a Python client library for communicating with the beanstalkd work queue. It makes it easy to write:

- **Producers**, processes that insert jobs into a queue:

```
import greenstalk

with greenstalk.Client(('127.0.0.1', 11300)) as client:
    client.put('hello')
```

- **Consumers**, processes that take jobs from a queue and execute some work:

```
import greenstalk

with greenstalk.Client(('127.0.0.1', 11300)) as client:
    while True:
        job = client.reserve()
        print(job.body)
        client.delete(job)
```

CHAPTER

ONE

CONTENTS

1.1 Installation

Greenstalk supports Python 3.5 and later. It's available on PyPI and can be installed by running:

```
pip install greenstalk
```

If you don't have `beanstalkd` installed, it's available in most package repositories.

Debian and Ubuntu:

```
sudo apt install beanstalkd
```

macOS with Homebrew:

```
brew install beanstalkd
```

1.2 Quickstart

Before getting started, ensure that Greenstalk is *installed* and `beanstalkd` is running.

1.2.1 Setup

Begin by importing the library:

```
>>> import greenstalk
```

Create a *Client*, which immediately connects to the server on the host and port specified:

```
>>> client = greenstalk.Client(('127.0.0.1', 11300))
```

Alternatively, if your server is listening on a Unix domain socket, pass the socket path instead:

```
>>> client = greenstalk.Client('/var/run/beanstalkd/socket')
```

1.2.2 Inserting Jobs

Jobs are inserted using *put*. The job body is the only required argument:

```
>>> client.put('hello')
1
```

Jobs are inserted into the currently used tube, which defaults to `default`. The currently used tube can be changed via *use*. It can also be set with the `use` argument when creating a *Client*.

1.2.3 Consuming Jobs

Jobs are consumed using *reserve*. It blocks until a job is reserved (unless the `timeout` argument is used):

```
>>> job = client.reserve()
>>> job.id
1
>>> job.body
'hello'
```

Jobs will only be reserved from tubes on the watch list, which initially contains a single tube, `default`. You can add tubes to the watch list with *watch* and remove them with *ignore*. For convenience, it can be set with the `watch` argument when creating a *Client*.

`beanstalkd` guarantees that jobs are only reserved by a single consumer simultaneously. Let's go ahead and tell the server that we've successfully completed the job using *delete*:

```
>>> client.delete(job)
```

Here's what you can do with a reserved job to change its state:

Command	Normal use case	Effect
`delete`	Success	Job is permanently deleted
`release`	Expected failure	Job is released back into the queue to be retried
`bury`	Unknown failure	Job is put in a special FIFO list for later inspection

1.2.4 Body Serialization

From `beanstalkd`'s point of view, the body of a job is just an opaque sequence of bytes. It's up to the clients to agree on a serialization format to represent the data required to complete the job.

In the context of a web application where a user just signed up and we need to send an email with a registration code, the producer may look something like this:

```
body = json.dumps({
    'email': user.email,
    'name': user.name,
    'code': code,
})
client.put(body)
```

The consumer would then do the inverse:

```
job = client.reserve()
data = json.loads(job.body)
send_registration_email(data['email'], data['name'], data['code'])
```

1.2.5 Body Encoding

When creating a *Client*, you can use the `encoding` argument to control how job bodies are encoded and decoded. It defaults to UTF-8.

You can set the `encoding` to `None` if you're working with binary data. In that case, you're expected to pass in `bytes` (rather than `str`) bodies, and `bytes` bodies will be returned.

1.2.6 Job Priorities

Every job has a priority which is an integer between 0 and 4,294,967,295. 0 is the most urgent priority. The *put*, *release* and *bury* methods all take a `priority` argument that defaults to `2**16`.

1.2.7 Delaying a Job

Sometimes you'll want to schedule work to be executed sometime in the future. Both the *put* and *release* methods have a `delay` argument.

1.2.8 Time to Run

Every job has an associated time to run (TTR) value specified by the `ttr` argument to the *put* method. It defaults to 60 seconds.

As soon as a job is reserved, `beanstalkd` starts a timer. If the client doesn't send a *delete*, *release*, or *bury* command within the TTR, the job will time out and be released back into the ready queue.

If more time is required to complete a job, the *touch* method can be used to refresh the TTR.

1.2.9 Job Lifecycle

Here's a great flowchart from the `beanstalkd` protocol documentation:

```
 put with delay               release with delay
----------------> [DELAYED] <------------.
                      |                   |
                      | (time passes)     |
                      |                   |
 put                  v     reserve       |       delete
-----------------> [READY] ---------> [RESERVED] --------> *poof*
                     ^  ^                |  |
                     |   \  release      |  |
                     |    `-------------'   |
                     |                      |
                     | kick                 |
                     |                      |
```

(continues on next page)

(continued from previous page)

```
                   |       bury           |
                [BURIED] <---------------'
                   |
                   |  delete
                    `--------> *poof*
```

1.3 API Reference

class greenstalk.**Client**(*address*, *encoding='utf-8'*, *use='default'*, *watch='default'*)

A client implementing the beanstalk protocol. Upon creation a connection with beanstalkd is established and tubes are initialized.

Parameters

- **address** (Union[Tuple[str, int], str]) – A socket address pair (host, port) or a Unix domain socket path.
- **encoding** (Optional[str]) – The encoding used to encode and decode job bodies.
- **use** (str) – The tube to use after connecting.
- **watch** (Union[str, Iterable[str]]) – The tubes to watch after connecting. The `default` tube will be ignored if it's not included.

close()

Closes the connection to beanstalkd. The client instance should not be used after calling this method.

Return type None

put(*body*, *priority=65536*, *delay=0*, *ttr=60*)

Inserts a job into the currently used tube and returns the job ID.

Parameters

- **body** (Union[bytes, str]) – The data representing the job.
- **priority** (int) – An integer between 0 and 4,294,967,295 where 0 is the most urgent.
- **delay** (int) – The number of seconds to delay the job for.
- **ttr** (int) – The maximum number of seconds the job can be reserved for before timing out.

Return type int

use(*tube*)

Changes the currently used tube.

Parameters **tube** (str) – The tube to use.

Return type None

reserve(*timeout=None*)

Reserves a job from a tube on the watch list, giving this client exclusive access to it for the TTR. Returns the reserved job.

This blocks until a job is reserved unless a `timeout` is given, which will raise a *TimedOutError* if a job cannot be reserved within that time.

Parameters **timeout** (Optional[int]) – The maximum number of seconds to wait.

Return type *Job*

reserve_job(*id*)
Reserves a job by ID, giving this client exclusive access to it for the TTR. Returns the reserved job.

A *NotFoundError* is raised if a job with the specified ID could not be reserved.

Parameters **id** (int) – The ID of the job to reserve.

Return type *Job*

delete(*job*)
Deletes a job.

Parameters **job** (Union[*Job*, int]) – The job or job ID to delete.

Return type None

release(*job*, *priority=65536*, *delay=0*)
Releases a reserved job.

Parameters

- **job** (*Job*) – The job to release.
- **priority** (int) – An integer between 0 and 4,294,967,295 where 0 is the most urgent.
- **delay** (int) – The number of seconds to delay the job for.

Return type None

bury(*job*, *priority=65536*)
Buries a reserved job.

Parameters

- **job** (*Job*) – The job to bury.
- **priority** (int) – An integer between 0 and 4,294,967,295 where 0 is the most urgent.

Return type None

touch(*job*)
Refreshes the TTR of a reserved job.

Parameters **job** (*Job*) – The job to touch.

Return type None

watch(*tube*)
Adds a tube to the watch list. Returns the number of tubes this client is watching.

Parameters **tube** (str) – The tube to watch.

Return type int

ignore(*tube*)
Removes a tube from the watch list. Returns the number of tubes this client is watching.

Parameters **tube** (str) – The tube to ignore.

Return type int

peek(*id*)
Returns a job by ID.

Parameters **id** (int) – The ID of the job to peek.

Return type *Job*

peek_ready()
Returns the next ready job in the currently used tube.

Return type *Job*

peek_delayed()
Returns the next available delayed job in the currently used tube.

Return type *Job*

peek_buried()
Returns the oldest buried job in the currently used tube.

Return type *Job*

kick(*bound*)
Moves delayed and buried jobs into the ready queue and returns the number of jobs effected.

Only jobs from the currently used tube are moved.

A kick will only move jobs in a single state. If there are any buried jobs, only those will be moved. Otherwise delayed jobs will be moved.

Parameters **bound** (int) – The maximum number of jobs to kick.

Return type int

kick_job(*job*)
Moves a delayed or buried job into the ready queue.

Parameters **job** (Union[*Job*, int]) – The job or job ID to kick.

Return type None

stats_job(*job*)
Returns job statistics.

Parameters **job** (Union[*Job*, int]) – The job or job ID to return statistics for.

Return type Dict[str, Union[str, int]]

stats_tube(*tube*)
Returns tube statistics.

Parameters **tube** (str) – The tube to return statistics for.

Return type Dict[str, Union[str, int]]

stats()
Returns system statistics.

Return type Dict[str, Union[str, int]]

tubes()
Returns a list of all existing tubes.

Return type List[str]

using()
Returns the tube currently being used by the client.

Return type str

watching()
Returns a list of tubes currently being watched by the client.

Return type List[str]

pause_tube(*tube*, *delay*)
Prevents jobs from being reserved from a tube for a period of time.

Parameters

- **tube** (str) – The tube to pause.
- **delay** (int) – The number of seconds to pause the tube for.

Return type None

class greenstalk.**Job**(*id*, *body*)
A job returned from the server.

class greenstalk.**Error**
Base class for non-connection related exceptions. Connection related issues use the built-in ConnectionError.

class greenstalk.**UnknownResponseError**(*status*, *values*)
The server sent a response that this client does not understand.

class greenstalk.**BeanstalkdError**
Base class for error messages returned from the server.

class greenstalk.**BadFormatError**
The client sent a malformed command.

class greenstalk.**BuriedError**(*values=None*)
The server ran out of memory trying to grow the priority queue and had to bury the job.

class greenstalk.**DeadlineSoonError**
The client has a reserved job timing out within the next second.

class greenstalk.**DrainingError**
The client tried to insert a job while the server was in drain mode.

class greenstalk.**ExpectedCrlfError**
The client sent a job body without a trailing CRLF.

class greenstalk.**InternalError**
The server detected an internal error.

class greenstalk.**JobTooBigError**
The client attempted to insert a job larger than max-job-size.

class greenstalk.**NotFoundError**
For the delete, release, bury, and kick commands, it means that the job does not exist or is not reserved by the client.

For the peek commands, it means the requested job does not exist or that there are no jobs in the requested state.

class greenstalk.**NotIgnoredError**
The client attempted to ignore the only tube on its watch list.

class greenstalk.**OutOfMemoryError**
The server could not allocate enough memory for a job.

class greenstalk.**TimedOutError**
A job could not be reserved within the specified timeout.

class greenstalk.**UnknownCommandError**
The client sent a command that the server does not understand.

CHAPTER

TWO

LINKS

This project is developed on GitHub. Contributions are welcome.

- Code
- Issue tracker

CHAPTER

THREE

INSPIRATION

Greenstalk is heavily inspired by the following libraries:

- Go - beanstalk
- Python - beanstalkc

INDEX