Greenstalk Documentation

Release 2.0.2

Justin Mayhew

May 28, 2023

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)
```

This library is a thin wrapper over the wire protocol. The documentation doesn't attempt to fully explain the semantics of the beanstalk protocol. It's assumed that users of this library will be referring to the official beanstalkd documentation.

CHAPTER 1

Contents

1.1 Installation

Greenstalk is available on PyPI:

```
pip install greenstalk
```

The server is available in most package repositories. For Debian and Ubuntu:

```
sudo apt install beanstalkd
```

For macOS with Homebrew:

```
brew install beanstalkd
```

1.2 Overview

Before getting started, ensure that Greenstalk is *installed* and the server 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*.

The server 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

The server does not inspect the contents of job bodies, it's only concerned with routing them between clients. This gives clients full control over how they're sent and received on the underlying connection.

JSON serialized payloads encoded in UTF-8 are a great default representation.

Here's an example showing how a producer and consumer (likely running in separate processes) could communicate a user registration email job.

Producer:

```
payload = {'user_id': user_id}
body = json.dumps(payload)
client.put(body)
```

The consumer would then do the inverse:

```
job = client.reserve()
payload = json.loads(job.body)
send_registration_email(payload['user_id'])
```

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.

The server starts a timer when a job is reserved. If the consumer 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                 |
                      |                      |
                      |       bury           |
```

(continues on next page)

(continued from previous page)

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

1.3 API Reference

1.3.1 Job

class greenstalk.**Job**(*id: int, body: Union[bytes, str]*)

A job returned from the server.

id = None

A server-generated unique identifier assigned to the job on creation.

body = None

The content of the job. Also referred to as the message or payload. Producers and consumers need to agree on how these bytes are interpreted.

1.3.2 Client

class greenstalk.**Client**(*address: Union[Tuple[str, int], str], encoding: Optional[str] = 'utf-8', use: str = 'default', watch: Union[str, Iterable[str]] = 'default'*)

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

Parameters

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

close() → None

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

put(*body: Union[bytes, str], priority: int = 65536, delay: int = 0, ttr: int = 60*) → int

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

Parameters

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

use(*tube: str*) → None

Changes the currently used tube.

Parameters **tube** – The tube to use.

reserve(*timeout: Optional[int] = None*) → greenstalk.Job
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** – The maximum number of seconds to wait.

reserve_job(*id: int*) → greenstalk.Job
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** – The ID of the job to reserve.

delete(*job: Union[greenstalk.Job, int]*) → None
Deletes a job.

Parameters **job** – The job or job ID to delete.

release(*job: greenstalk.Job*, *priority: int = 65536*, *delay: int = 0*) → None
Releases a reserved job.

Parameters

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

bury(*job: greenstalk.Job*, *priority: int = 65536*) → None
Buries a reserved job.

Parameters

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

touch(*job: greenstalk.Job*) → None
Refreshes the TTR of a reserved job.

Parameters **job** – The job to touch.

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

Parameters **tube** – The tube to watch.

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

Parameters **tube** – The tube to ignore.

peek(*id: int*) → greenstalk.Job
Returns a job by ID.

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

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

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

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

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

kick_job(*job: Union[greenstalk.Job, int]*) → None
Moves a delayed or buried job into the ready queue.

Parameters **job** – The job or job ID to kick.

stats_job(*job: Union[greenstalk.Job, int]*) → Dict[str, Union[str, int]]
Returns job statistics.

Parameters **job** – The job or job ID to return statistics for.

stats_tube(*tube: str*) → Dict[str, Union[str, int]]
Returns tube statistics.

Parameters **tube** – The tube to return statistics for.

stats() → Dict[str, Union[str, int]]
Returns system statistics.

tubes() → List[str]
Returns a list of all existing tubes.

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

watching() → List[str]
Returns a list of tubes currently being watched by the client.

pause_tube(*tube: str*, *delay: int*) → None
Prevents jobs from being reserved from a tube for a period of time.

Parameters

- **tube** – The tube to pause.
- **delay** – The number of seconds to pause the tube for.

1.3.3 Exceptions

For completeness all errors that beanstalkd can return are listed here. *BadFormatError* and *ExpectedCrlfError* *should* be unreachable unless there's a bug in this library.

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

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

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

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

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

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

This can be raised in response to a put or release command.

id = None
A server-generated unique identifier that was assigned to the buried job.

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

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

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

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

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

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

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

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

status = None
The status code of the response. Contains `b'SOME_ERROR'` for the response `b'SOME_ERROR 1 2 3\r\n'`.

values = None
The remaining split values after the status code. Contains `[b'1', b'2', b'3']` for the response `b'SOME_ERROR 1 2 3\r\n'`.

Index

W