retask Documentation

Release 0.1

Kushal Das

July 11, 2012

CONTENTS

retask is a python module to create and manage distributed task queues.

It uses Redis to create task queues. User can enqueue and dequeue tasks in the queues they manage. Each task can contain any serializable python objects. We use *JSON* internally to store the tasks in the queues.

CHAPTER

ONE

DEPENDENCIES

- python-redis
- mock
- A running Redis server

CHAPTER

TWO

USER GUIDE

2.1 Introduction

2.1.1 First requirement

For various others projects I had to start looking for a simple task queue and solve kind of classical producer-consumer problems using them.

This project started from that idea.

2.1.2 Why Redis

I am following Redis development for a time and using it in various other projects. The simplicity it provides and rich datastructures are always a plus to use it. It is also very fast.

2.1.3 Retask License

2.2 Setting up the Redis Server

You can download and install Redis on your distro. In Fedora you can just `yum install redis` for the same.

To start the server in the local folder use the following command:

```
$ redis-server
```

On Fedora you can start the service as *root*:

```
# systemctl enable redis.service
# systemctl start redis.service
```

2.3 Installation

This part of the documentation covers the installation of Retask. The first step to using any software package is getting it properly installed.

2.3.1 Distribute & Pip

Installing requests is simple with pip:

```
$ pip install retask
```

or, with easy_install:

```
$ easy_install retask
```

But, you really shouldn't do that.

2.3.2 Get the Code

Retask is actively developed on GitHub, where the code is always available.

You can either clone the public repository:

```
git clone git://github.com/retask/retask.git
```

Download the tarball:

```
$ curl -OL https://github.com/kushaldas/retask/tarball/master
```

Or, download the zipball:

```
$ curl -OL https://github.com/kushaldas/retask/tarball/master
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

2.3.3 Installing redis-py

You can install `redis-py` with `pip`:

```
$ pip install redis
```

2.4 Quickstart

For this example to work you should have your redis instance up and running.

2.4.1 producer.py

This code puts new task in the queue. We will have a dictionary as the information in this example.

```
from retask.task import Task
from retask.queue import Queue
queue = Queue('example')
info1 = {'user':'kushal', 'url':'http://kushaldas.in'}
info2 = {'user':'fedora planet', 'url':'http://planet.fedoraproject.org'}
task1 = Task(info1)
task2 = Task(info2)
queue.connect()
queue.enqueue(task1)
queue.enqueue(task2)
```

2.4.2 consumer.py

This code gets the tasks from the queue. Based on the actual requirement, the client will work on the information it received as the task. For now we will just print the data.

```
from retask.task import Task
from retask.queue import Queue
queue = Queue('example')
queue.connect()
while queue.length != 0:
    task = queue.dequeue()
    print task.data
```

CHAPTER

THREE

API DOCUMENTATION

3.1 API

This part contains the API documentation of the module.

3.1.1 Classes

retask.queue

This module contains the primary `Queue` which can be used to create and manage queues.

class `retask.queue.`**Queue**(*name*, *config={}*)

Returns the `Queue` object with the given name. If the user passes optional config dictionary with details for Redis server, it will connect to that instance. By default it connects to the localhost.

connect()

Creates the connection with the redis server. Return `True` if the connection works, else returns `False`. It does not take any arguments.

Returns `Boolean` value

Note: After creating the `Queue` object the user should call the `connect` method so create the connection.

```
>>> from retask.queue import Queue
>>> q = Queue('test')
>>> q.connect()
True
```

dequeue()

Returns a `Task` object from the queue. Returns `None` if the queue is empty.

Returns `Task` object from the queue

If the queue is not connected then it will raise `retask.ConnectionError`

```
>>> from retask.queue import Queue
>>> q = Queue('test')
>>> q.connect()
True
>>> t = q.dequeue()
```

```
>>> print t.data
{u'name': u'kushal'}
```

enqueue (*task*)
Enqueues the given `Task` object to the queue and returns a tuple. Value in index 0 is `Boolean` explaining the enqueue operation is a success or not. Value at index 1 is string with error/success message (if any).

Parameters **task** – :`Task` object

Returns Tuple with `Boolean` value and `string` message.

If the queue is not connected then it will raise `retask.ConnectionError`.

```
>>> from retask.queue import Queue
>>> q = Queue('test')
>>> q.connect()
True
>>> from retask.task import Task
>>> task = Task({'name':'kushal'})
>>> q.enqueue(task)
(True, 'Pushed')
```

length
Gives the length of the queue. Returns `None` if the queue is not connected.

If the queue is not connected then it will raise `retask.ConnectionError`.

retask.task

This module conatins generic task class, which can be used to create any kind of given task with serializable python objects as data.

class `retask.task.`**Task** (*data=None*, *raw=False*)
Returns a new Task object, the information for the task is passed through argument `data`

Parameters **data** – Python object which contains information for the task. Should be serialzable through `JSON`.

data
The python object containing information for the current task

rawdata
The string representation of the actual python objects for the task

---

Note: This should not be used directly by the users. This is for internal use only.

3.1.2 Exceptions

exception `retask.`**RetaskException**
Some ambigous exception occured

exception `retask.`**ConnectionError**
A Connection error occurred.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*

PYTHON MODULE INDEX

r