PyRMQ

Alexandre Gerona & Jasper Sibayan

Jun 16, 2020

CONTENTS

Python with RabbitMQ—simplified so you won't have to.

CHAPTER

ONE

FEATURES

Stop worrying about boilerplating and implementing retry logic on your queues. PyRMQ already does it for you.

- Use out-of-the-box and thread-safe `Consumer` and `Publisher` classes created from pika for your projects and tests.
- Built-in retry-backoff logic for connecting, consuming, and publishing.
- Works with Python 3.
- Production ready

CHAPTER

TWO

QUICKSTART

PyRMQ is available at PyPI.

```
$ pip install pyrmq
```

Just instantiate the feature you want with their respective settings. PyRMQ already works out of the box with RabbitMQ's default initialization settings.

```
from pyrmq import Publisher
publisher = Publisher(
    exchange_name="exchange_name",
    queue_name="queue_name",
    routing_key="routing_key",
)
publisher.publish({"pyrmq": "My first message"})
```

CHAPTER

THREE

USER GUIDE

3.1 PyRMQ Installation

There are multiple ways to install PyRMQ as long as multiple versions to choose from.

3.1.1 Stable Version

PyRMQ is available at PyPI.

```
$ pip install pyrmq
```

3.1.2 Development Version

Since PyRMQ is continuously used in a growing number of internal microservices all working with RabbitMQ, you can see or participate in its active development in its GitHub repository.

There are two ways to work or collaborate with its development version.

Git Checkout

Clone the code from GitHub and run it in a *virtualenv*.

```
$ git clone git@github.com:altusgerona/pyrmq.git
$ virtualenv venv --distribute
$ . venv/bin/activate
$ python setup.py install
```

This will setup PyRMQ and its dependencies on your local machine. Just fetch/pull code from the master branch to keep your copy up to date.

PyPI

```
$ mkdir pyrmq
$ cd pyrmq
$ virtualenv venv --distribute
$ . venv/bin/activate
$ pip install git+git://github.com/altusgerona/pyrmq.git
```

3.2 How to use PyRMQ

3.2.1 Publishing

Instantiate the `Publisher` class and plug in your application specific settings. PyRMQ already works out of the box with RabbitMQ's default initialization settings.

```
from pyrmq import Publisher
publisher = Publisher(
    exchange_name="exchange_name",
    queue_name="queue_name",
    routing_key="routing_key",
)
publisher.publish({"pyrmq": "My first message"})
```

This publishes a message that uses a BlockingConnection on its own thread with default settings and an exponential backoff logic for its retries.

Retries

PyRMQ's `Publisher` retries happen on two levels: connecting and publishing.

Connecting

PyRMQ instantiates a BlockingConnection when connecting. If this fails, it will retry for 2 more times by default with a delay of 5 seconds, a backoff base of 2 seconds, and a backoff constant of 5 seconds. All these settings are configurable via the `Publisher` class.

Publishing

PyRMQ calls pika's basic_publish when publishing. If this fails, it will retry for 2 more times by default with a delay of 5 seconds, a backoff base of 2 seconds, and a backoff constant of 5 seconds. All these settings are configurable via the `Publisher` class.

Max retries reached

When PyRMQ has tried one too many times, it will call your specified callback.

3.2.2 Consuming

Instantiate the `Consumer` class and plug in your application specific settings. PyRMQ already works out of the box with RabbitMQ's default initialization settings.

```
from pyrmq import Consumer


def callback(data):
    print(f"Received {data}!")

consumer = Consumer(
    exchange_name="exchange_name",
    queue_name="queue_name",
    routing_key="routing_key",
)

consumer.start()
```

Once the `Consumer` class is instantiated, just run `start()` to start its own thread that targets pika's start_consuming method on its own thread with default settings and an exponential backoff logic for its retries. Consumption calls basic_ack with `delivery_tag` set to what the message's `method`'s was.

Retries

PyRMQ's `Consumer` retries happen on two levels: connecting and consuming.

Connecting

PyRMQ instantiates a BlockingConnection when connecting. If this fails, it will retry for 2 more times by default with a delay of 5 seconds, a backoff base of 2 seconds, and a backoff constant of 5 seconds. All these settings are configurable via the `Consumer` class.

Consuming

PyRMQ calls pika's start_consuming when `Consumer` is instantiated. If this fails, it will retry for 2 more times by default with a delay of 5 seconds, a backoff base of 2 seconds, and a backoff constant of 5 seconds. All these settings are configurable via the `Consumer` class.

Max retries reached

When PyRMQ has tried one too many times, it will call your specified callback.

3.3 API Documentation

3.3.1 Publisher Class

3.3.2 Consumer Class

3.4 Testing PyRMQ

We're not gonna lie. Testing RabbitMQ, mocks or not, is infuriating. Much harder than a traditional integration testing with a database. That said, we hope that you could help us expand on what we have started should you feel our current tests aren't enough.

3.4.1 RabbitMQ

Since PyRMQ strives to be as complete with testing as it can be, it has several integration tests that need a running RabbitMQ to pass. Currently, PyRMQ is tested against `rabbitmq:3.8`.

Run Docker image (recommended)

```
$ docker run -d --hostname my-rabbit --name rabbitmq -p 5672:5672 rabbitmq:alpine
```

This allows you to connect to RabbitMQ via localhost through port 5672. Default credentials are `guest`/`guest`.

Install and run RabbitMQ locally

```
$ # Depending on your OS
$ # Ubuntu
$ sudo apt install rabbitmq
$ # Arch Linux
$ sudo pacman -S rabbitmq
```

3.4.2 Using tox

Install `pip install tox` and run:

```
$ tox
$ tox -e py38  # If this is what you have installed or don't want to bother testing␣
↪for other versions
```