# Ruby - Workers - Sneakers

<https://github.com/jondot/sneakers>

A high-performance RabbitMQ background processing framework for Ruby.

Sneakers is being used in production for both I/O and CPU intensive workloads, and have achieved the goals of high-performance and 0-maintenance, as designed.

Compared to Sidekiq, per my use case, I needed

* A great-performing framework limited only by broker speed - at least 1000req/s acknowledged and persisted on EC2-Large (Sneakers does more than that), and
* That would use all cores
* A highly available processing framework (here we have same guarantees as RabbitMQ offers, which is great)
* A familiar DSL/API that also supports advanced messaging semantics such as reject, requeue, acknowledge, etc, and
* That would not expose the whole guts of AMQP at me, but just-enough from it.
* It should use a ruby that doesn't care about content of gems and can run C-extensions. MRI.
* A production-ready package that holds all of these together allowing me to be as lazy as possible
* Metrics and logging baked in
* Convenient deployment, maintenance and supervision story

## General

A sneakers class is created by including the Sneakers::Worker module, defining a queue to subscribe to, and adding a work function.

For example: ./sneakers-processor.rb

require 'sneakers'

require 'json'

class Processor

include Sneakers::Worker

from\_queue :logs

def work(msg)

err = JSON.parse(msg)

if err["type"] == "error"

console.log('error', err["error"])

end

ack!

end

end

Then run ```sneakers work Processor --require sneakers-processor.rb``` to start the worker, which will create and subscribe the logs queue.

Use the Bunny gem to publish messages to a rabbitmq queue. For example: ./publish-log-message.rb

require 'bunny'

require 'json'

conn = Bunny.new

conn.start

ch = conn.create\_channel

ch.default\_exchange.publish({ type: 'error', message: 'HALP!', error: 'CODE001' }.to\_json, routing\_key: 'logs')

conn.close

Use ```ruby ./publish-log-message.rb``` to pulish an error message.

## Job Control

The following job controll signals are available:

* ack! - acknowledgement of job completion
* requeue! - sends item back to the top of the queue (can create endless loop)
* reject! - move message to different queue or dead letter box

By default sneaker will use an acknowledgement connection, so ```ack!``` must be called a the end of the work method to acknowledge completion and remove it from the rabbitmq queue.

The signal message must be returned from the work method to work. So use an explict return if there is the chance of error.

## Publishing Messages

The Sneakers::Worker mixin will include a publish method which allows for new message to be published to a rabbitmq queue. For example:

publish(doc.css('title').text, :to\_queue => 'title\_classification')

## Logging

The Sneakers::Worker mixin will include a logger method which can be used to log. For example:

logger.info 'job complete'

## Scaling

Sneakers is designed to scale using threads and processes.

To scale processes, simple create more processes of sneaker workers.

To scale threads add the number of threads and prefetch to match to the queue definition of the worker. For example:

class WebScraper

include Sneakers::Worker

from\_queue :web\_pages,

:threads => 50,

:prefetch => 50,

:timeout\_job\_after => 1

…

## Config

<https://github.com/jondot/sneakers/wiki/Configuration>