Sidekiq 101

Dario Castañé

Sidekiq

```ruby
class BackgroundWorker
  include Sidekiq::Worker

  def perform(user_id)
    # TODO: do something!
  end
end
```

```ruby
BackgroundWorker.perform_async(user_id)
```

```ruby
class BackgroundWorker
  include Sidekiq::Worker

  sidekiq_options retry: false
                       # or a number of retries

  def perform
    # TODO: do something!
  end
end
```

```ruby
class BackgroundWorker
  include Sidekiq::Worker

  sidekiq_options retry: false,
                  queue: :another_queue

  def perform
    # TODO: do something!
  end
end
```

```yaml
---
:verbose: false
:concurrency: 1 # threads
:timeout: 25 # seconds

# Sidekiq will run this sidekiq.yml through ERB when reading it so you can
# even put in dynamic logic, like a host-specific queue.
:queues:
  - default
  # [name, weight]
  - [critical, 1]
  - [<%= `hostname`.strip %>, 1]
  - [another_queue, 1]

# you can override concurrency based on environment
production:
  :concurrency: 5
staging:
  :concurrency: 3
```

```
worker: bundle exec sidekiq -e $RAILS_ENV -C ./config/sidekiq.yml
worker_critical: bundle exec sidekiq -e $RAILS_ENV -q critical -c 3
worker_another_queue: bundle exec sidekiq -e $RAILS_ENV -q another_queue
```

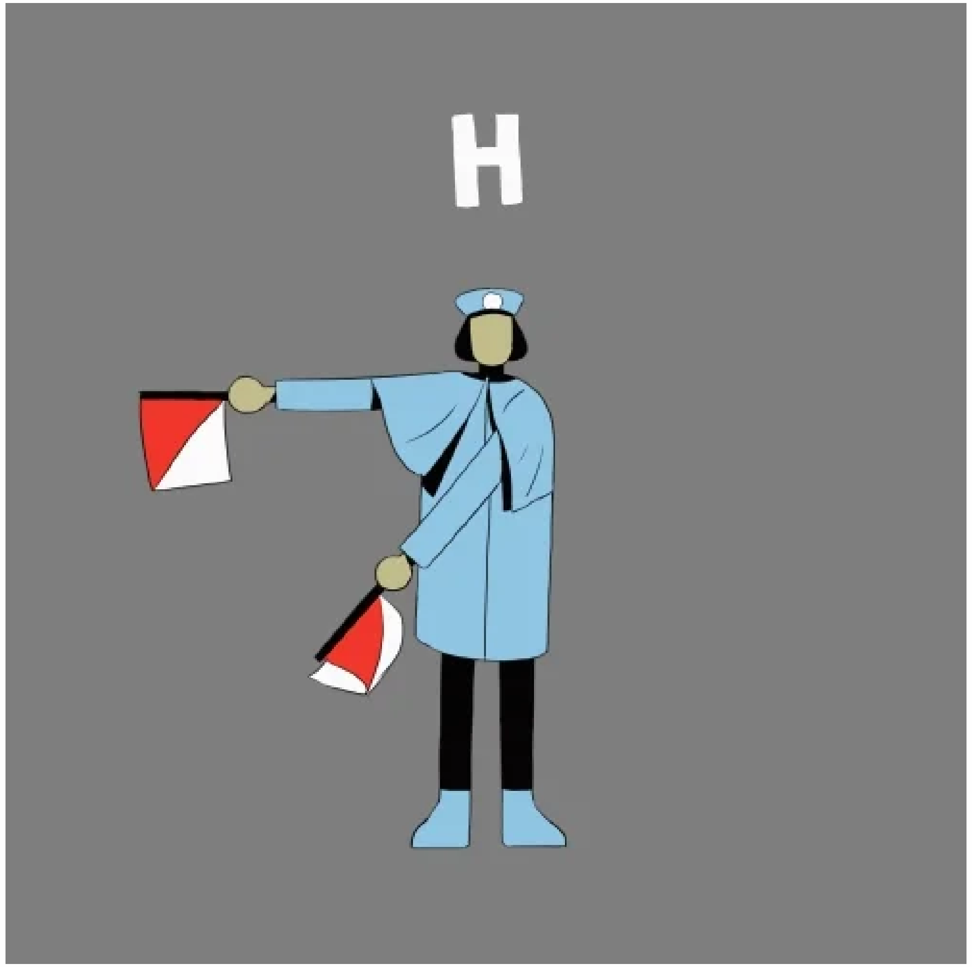
```ruby
class BackgroundWorker
  include Sidekiq::Worker

  sidekiq_options retry: false,
                  queue: :another_queue
                  # Options from mhenrixon/sidekiq-unique-jobs
                  unique: :until_and_while_executing,
                        # Other Job locks can be:
                        #   :until_executing
                        #   :until_executed
                        #   :until_expired
                        #   :while_executing
                  # This sets how long our job is considered unique while executing
                  run_lock_expiration: 2 * 60

  def perform
    # TODO: do something!
  end
end
```

```ruby
class BackgroundWorker < BaseWorker # includes more things!!1
  sidekiq_options retry: false,
                  queue: :another_queue
                  unique: :until_and_while_executing,
                  run_lock_expiration: 2 * 60

  def perform
    # TODO: do something!
  end
end
```


H

```ruby
class BackgroundWorker < BaseWorker
  # sidekiq_options ...

  semaphore_slug :background
                 # :background, :specific_alias
                 # This becomes background@specific_alias

  def perform_semaphore(company_slug, arg_1, arg_2, _, _)
    # TODO: do something!
  ensure
    release
  end
end
```

```ruby
BackgroundWorker.perform_with_semaphore(arg_1, arg_2)
```

```ruby
class BackgroundWorker < BaseWorker
  # sidekiq_options ...

  semaphore_slug :background
  semaphore_unique # Your process can be unique by slug
                   # :params (or by its slug, slug_alias, and params)

  def perform_semaphore(company_slug, arg_1, arg_2, ..., label, process_key)
    # TODO: do something!
  ensure
    release
  end
end
```

```ruby
class BackgroundWorker < BaseWorker
  # sidekiq_options ...

  semaphore_slug :background
  semaphore_red_lights :critical, :period
  semaphore_options on_background: true
  semaphore_unique

  def perform_semaphore(company_slug, arg_1, arg_2, ..., label, process_key)
    Company.switch! company_slug

    # TODO: do something!
  ensure
    release
  end
end
```

```ruby
class BackgroundWorker < BaseWorker
  # sidekiq_options ...
  # semaphore_* ...

  # Let's autoscale!!1

  # 1. Add the worker (if not already) in the Procfile
  # 2. Create a scalator
  # 3. Add the scalator to AutoScaleWorker#scalators array

  def perform_semaphore(company_slug, arg_1, arg_2, ..., label, process_key)
    # TODO: do something!
  ensure
    release
  end
end
```

```ruby
module Server
  class BackgroundWorkerScalator
    def name { 'worker_another_queue' }

    def command { 'bundle exec sidekiq -e $RAILS_ENV -q another_queue' }

    def size { 'standard-2x' }

    def required_number_servers
      servers = ((running_workers * JOBS_PER_COMPANY).to_f / JOBS_PER_WORKER).ceil
      servers -= 1 if servers.positive? # There is a server running
      servers = MAX_SERVERS if servers > MAX_SERVERS

      servers
    end

    # ...
  end
end
```

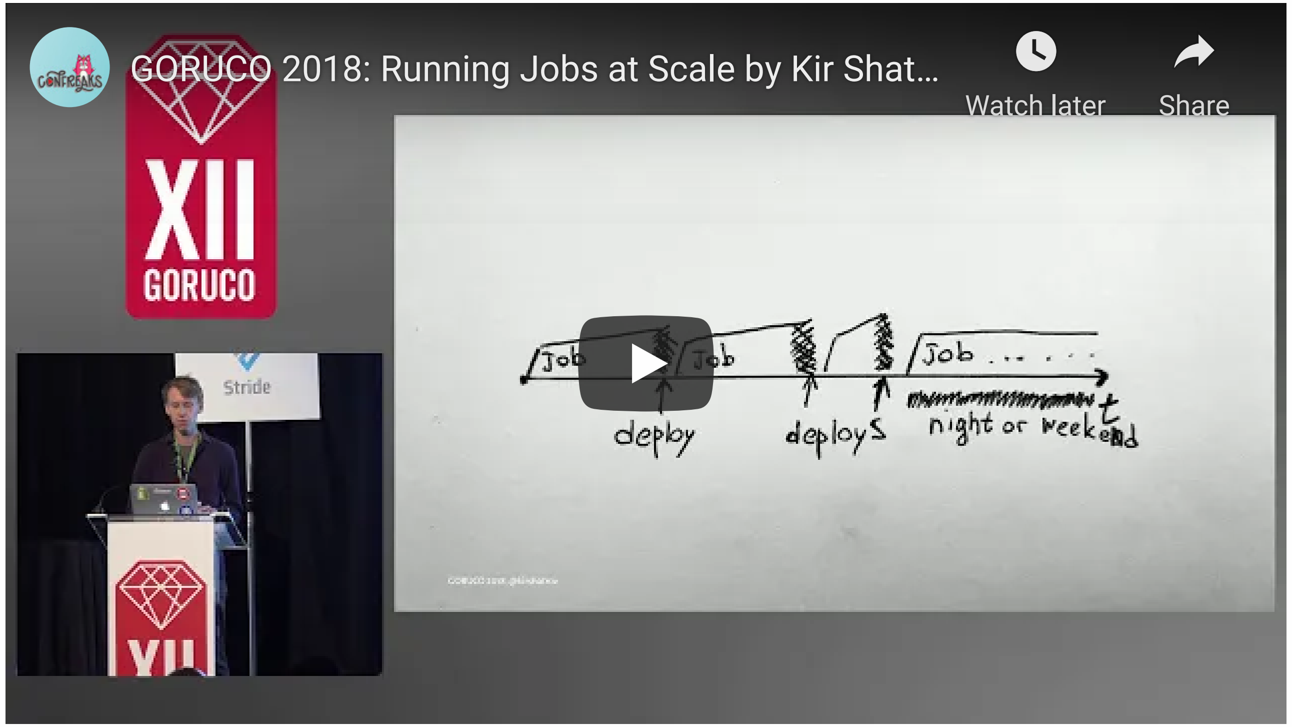
```ruby
class AutoScaleWorker < BaseWorker
  # ...

  private

  def scalators
    [
      Server::CampaignWorkerScalator.new,
      Server::PlanWorkerScalator.new,
      Server::BackgroundWorkerScalator.new
    ]
  end
end
```

More challenges

Long running jobs: Shopify/job-iteration



Thank you! Q&A time!