# Sidekiq Enterprise

examples and practical use cases

Bruno Sutic

# Intro

- Q: who uses sidekiq?
- standard for background job processing in Ruby
- 3 versions: free, pro and enterprise
  (focus on pro and enterprise)
- Q: who uses sidekiq pro/enterprise?

# Free version

- job retries (simple and fine grained)
- delaying jobs: “run this job in 15 minutes”
- web UI
- Ruby API
- works as a Rails’ ActiveJob runner

# Periodic/cron jobs

- Problem, unix cron jobs are finicky:
  - a devops problem, not a coding one
  - extra knowledge about the cron environment
  - always need to be (manually) tested
  - separate logs to check success/failure

# Periodic/cron jobs

```ruby
# config/initializers/sidekiq.rb

Sidekiq.configure_server do |config|
  config.periodic do |mgr|
    mgr.register '* * * * *', MinuteWorker
    mgr.register '0 * * * *', HourlyWorker
    mgr.register '0 0 * * *', DailyWorker
    mgr.register '0 7 * * 1', WeeklyWorker
  end
end

class HourlyWorker
  include Sidekiq::Worker

  def perform
    SomeClass.perform_later
    Another.perform_later
  end
end

class DailyWorker
  # omitted (similar to HourlyWorker)
end
```

# Periodic/cron jobs

- 100% ruby code - no devops
- if sidekiq works, periodic jobs work
- web UI for failed jobs (no server logs)

# Job batches

- Problems:
  - lot of jobs, how to "group" them?
  - execute a hook after a group of jobs is done?
  - a group of 1M jobs is working, what's the progress?

# Job batches

```ruby
class Subscription < ActiveRecord::Base
  def self.update_subscriptions
    batch = Sidekiq::Batch.new
    batch.description = 'Update billing subscriptions'

    batch.jobs do
      find_each do |subscription|
        CreateNextSubscription.perform_async(subscription.to_gid)
      end
    end
  end

  # more code
end
```

Sidekiq Enterprise active

Dashboard | Busy | Queues | Retries | Scheduled | Dead | Plugins

Live Poll

| Processed | Failed | Busy | Enqueued | Retries | Scheduled | Dead |
|---|---|---|---|---|---|---|

# Batches

| Started | Description | Job Count | Pending | Failed | Status |
|---|---|---|---|---|---|
| 24 days ago | Update billing subscriptions | 3743 | 1 | 1 | |
| 24 days ago | DailyWorker | 3 | 0 | 0 | |
| 25 days ago | Update billing subscriptions | 3741 | 1 | 1 | |
| 25 days ago | DailyWorker | 3 | 0 | 0 | |

Sidekiq Enterprise idle

Dashboard | Busy | Queues | Retries | Scheduled | Dead | Batches | Cron | Limits | Live Poll

# Batch TmVkQG-_ElvrMQ

| | |
|---|---|
| **Status** | |
| **Description** | Update billing subscriptions |
| **Parent** | g-fM5JECLPWMqg |
| **Created** | 25 days ago |
| **Expires** | 5 days from now |
| **Size** | 3741 |
| **Pending** | 1 |
| **Failures** | 1 |

# Failures

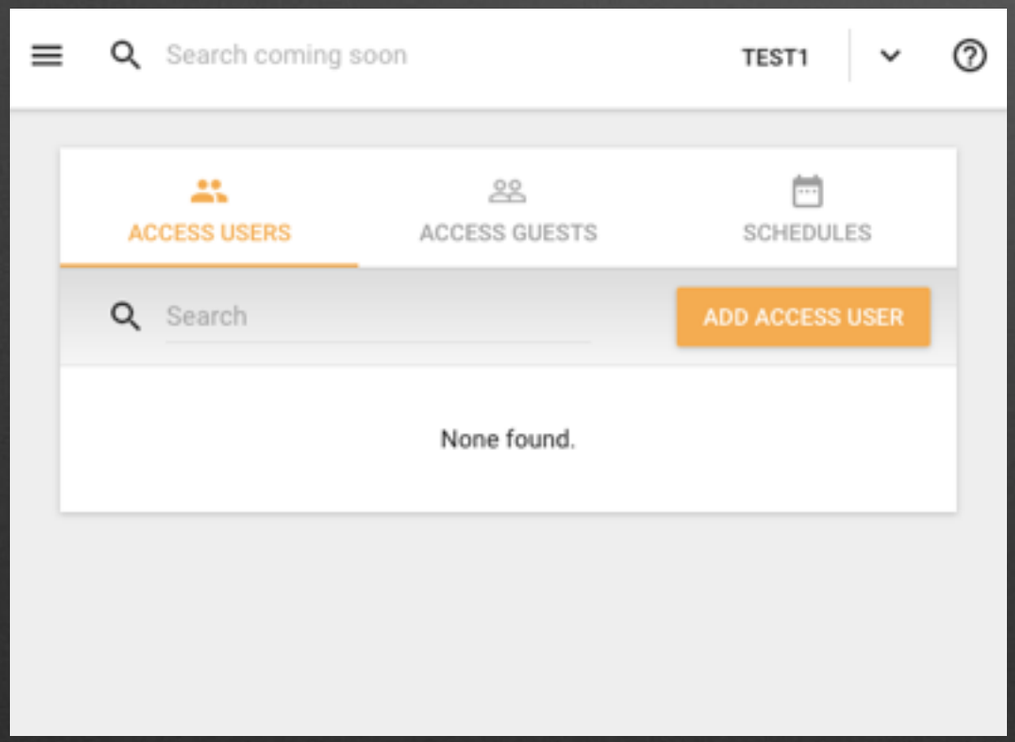
| Job | Class | Error Message |
|---|---|---|
| d5795baa5bc18ff3fdd608bd | Stripe::APIError | Error while talking to one of our backends. We have been notified of the problem. Please contact support@stripe.com with any questions you may have. |

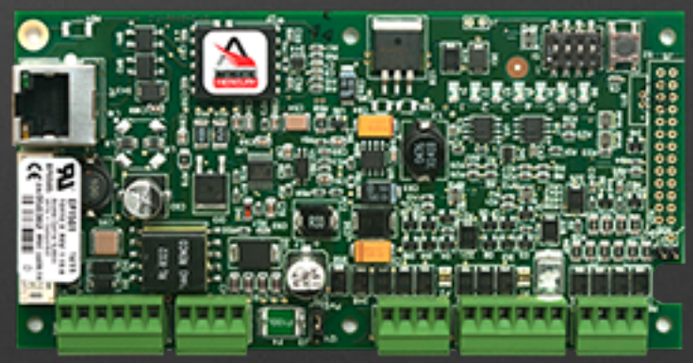
# Job batches

- batches: a group of jobs
- monitoring: better insight in the sidekiq UI
- progress indicator: e.g. 43 of 100 jobs done
- hooks (efficient)

# Unique jobs

- Problem:
  - expensive or unnecessary work triggered by UI or API
  - context: multiple servers, multiple processes, multiple threads



# Unique jobs

```ruby
class UserSync
  include Sidekiq::Worker

  sidekiq_options unique_for: 1.minute

  def perform(controller_id)
    Controller.find(controller_id).sync_users
  end
end

# Usage
def create_user(args = {})
  create(args)
  UserSync.perform_in(15.minutes, controller_id)
end
```

# Unique jobs

- job is unique based on the worker arguments
- works great with delaying feature:
  job is unique for delay time + till job is done

# Limiter

- Problem:
  - send events from one service to the other, but not too fast
  - context: multiple servers, multiple processes, multiple threads

```ruby
EVENT_THROTTLER = Sidekiq::Limiter.concurrent(
  'event-throttler',          # unique name
  2,                          # number of concurrent executions
  wait_timeout: 5,            # number of seconds other jobs wait
  lock_timeout: 30            # max allowed execution time
)

# Usage
class EventWorker
  include Sidekiq::Worker
  # implicit retry mechanism: 25 times

  def perform(event_id)
    event = GlobalID::Locator.locate(event_id)
    EVENT_THROTTLER.within_limit do
      # this block is limited
      send_event(event)
    end
  end
end
```

# Limiter

- efficient
- docs say it's "hard on redis"
- other limiter types (window, bucket)

# Limiter metrics

event-throttler

| | |
|---|---|
| **Status** | |
| **Type** | Concurrent |
| **Size** | 2 |
| **Available** | 2 |
| **In Use** | 0 |

## Metrics

| | |
|---|---|
| **Held** | 9468233 |
| **Hold Time** | 1743899.173 |
| **Immediate** | 6286383 |
| **Waited** | 3181850 |
| **Wait Time** | 1372142.439 |
| **Overages** | 1563 |
| **Reclaimed** | 1591 |

# Other features

- process / Ruby VM crash protection
- job argument encryption
- multicore processing

# Conclusion

- Works and scales really well
- Nice features that cost $$
- Free gems for almost all the features!

# Questions?