Performance Monitoring tools for Ruby on Rails applications

An

Growth Session X (#10) - December 21-22 2017

Intro

- To explore, compare the performance monitoring tools for Rails app
- 2 parts
    - Explore, compare the tools (Open source and Third-party)
    - How to improve the Rails app, and fix the memory issues.

Explore the Performance monitoring tools

Open Source

	rack-mini-profiler	derailed_benchmarks	peek
Easy to setup	Yes	Yes	Require to import the js, css and add the `<%= render 'peek/bar' %>` to the layout manually => break the layout
Easy to use	Yes	No Does NOT have UI Run with CLI and run only 1 path as the 1 time	*Don't try it yet*
Run on production	Yes	No	*Don't try it yet*
Support to run with multiple instance	Yes	No	*Don't try it yet*
Pros	Nice UI, support to run on Production, only admin can see the Profiler. Support many kind of benchmark	With Statis benchmark mode, we can see which gem is take too much memory at in the Gemfile	

Explore the Performance monitoring tools

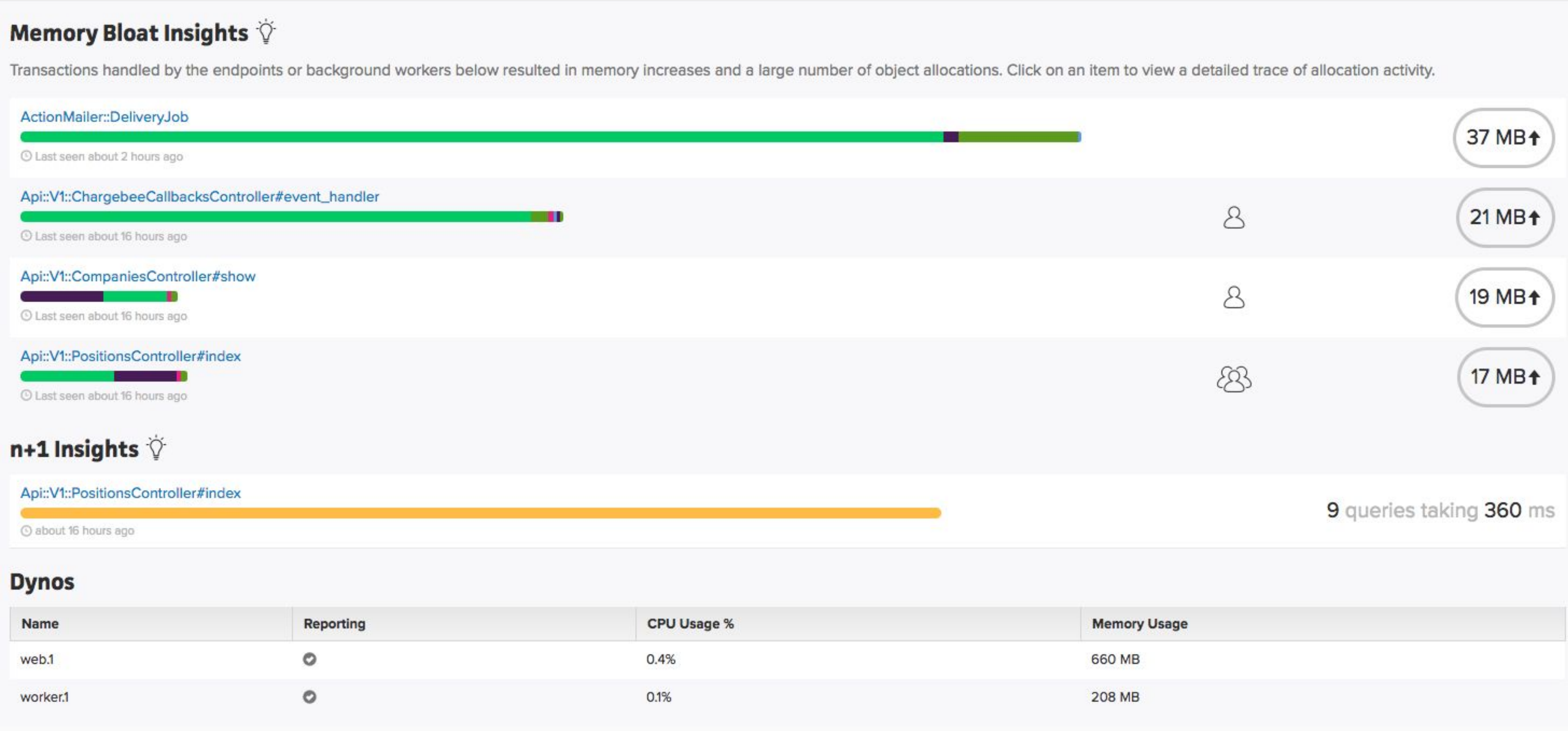
Paid Third Party

	newrelic	scoutapp	skylight	rorvswild
Easy to setup	Yes	Yes	Yes	Yes
Easy to use	No, to slow	Yes	Yes	Yes
Request monitoring	Yes	Yes	Yes	Yes
Background job monitoring	Yes	Yes	No	Yes
Memory allocate	Yes	Yes	No	No
More info (Memory bloat, n+1 query, explain SQL query)	Yes	Yes	No	No (Has explain SQL)
Addon on Heroku	Yes	Yes	No	No
Alert (free package)	No	Yes	No alert feature	Only Slack
Request limited (free package)	Unlimited	300.000/month	100.000/month	100.000/month

Open source vs Third party

	Open Source	Paid Third Party
Pros	Free	Easy to use, nice UI, nice report.
Cons	Implement manually Parse the data, do the report manually In case the app run on multiple server we have to store the report at the same place (Redis, Elasticsearch)	Not Free if the app has many request or background job processing New Relic, ScoutApp are not free without Heroku

Scout App - Dashboard



Memory Bloat Insights

Transactions handled by the endpoints or background workers below resulted in memory increases and a large number of object allocations. Click on an item to view a detailed trace of allocation activity.

ActionMailer::DeliveryJob
Last seen about 2 hours ago
37 MB↑

Api::V1::ChargebeeCallbacksController#event_handler
Last seen about 16 hours ago
21 MB↑

Api::V1::CompaniesController#show
Last seen about 16 hours ago
19 MB↑

Api::V1::PositionsController#index
Last seen about 16 hours ago
17 MB↑

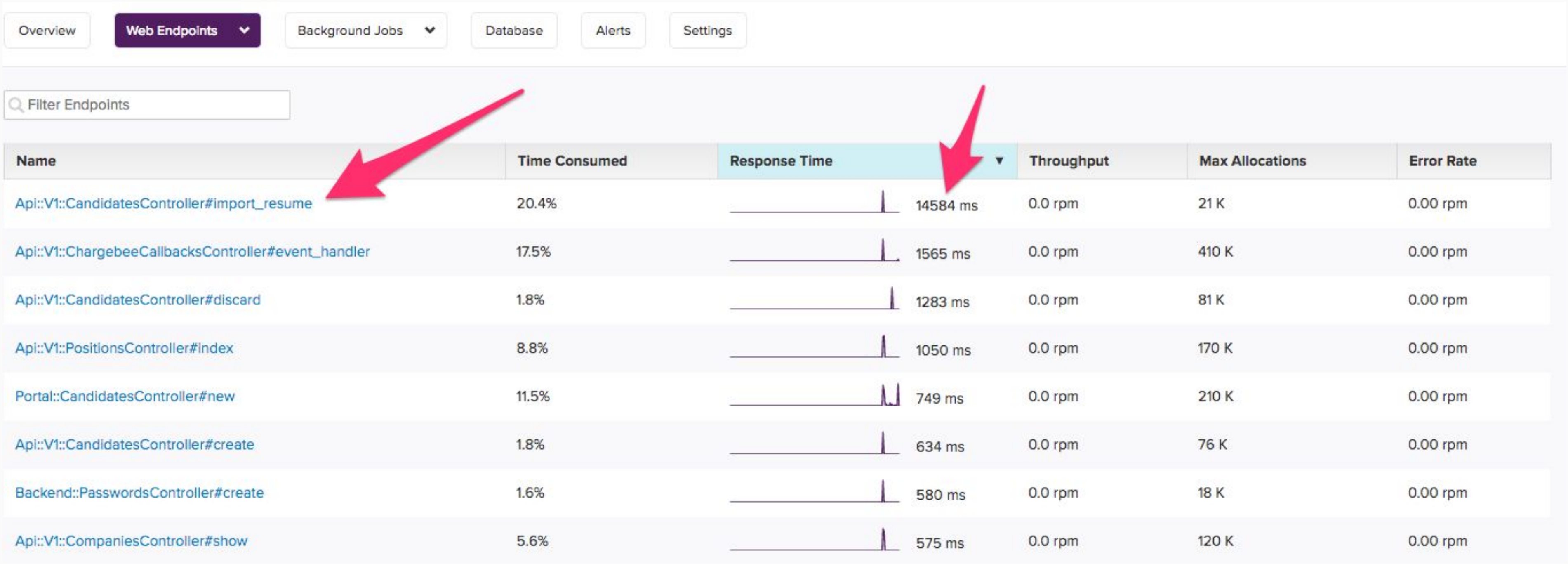
n+1 Insights

Api::V1::PositionsController#index
about 16 hours ago
9 queries taking 360 ms

Dynos

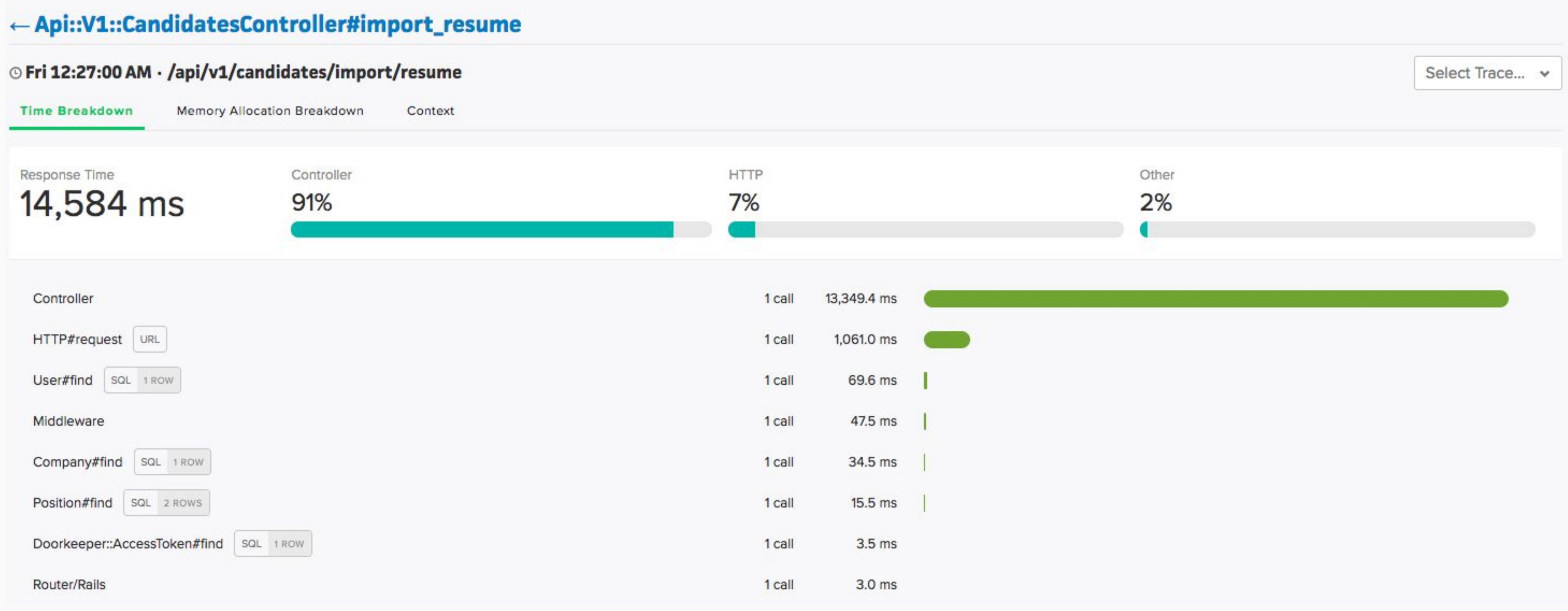
Name	Reporting	CPU Usage %	Memory Usage
web.1	✓	0.4%	660 MB
worker.1	✓	0.1%	208 MB

Scout App - Web endpoint (request)



Name	Time Consumed	Response Time	Throughput	Max Allocations	Error Rate
Api::V1::CandidatesController#import_resume	20.4%	14584 ms	0.0 rpm	21 K	0.00 rpm
Api::V1::ChargebeeCallbacksController#event_handler	17.5%	1565 ms	0.0 rpm	410 K	0.00 rpm
Api::V1::CandidatesController#discard	1.8%	1283 ms	0.0 rpm	81 K	0.00 rpm
Api::V1::PositionsController#index	8.8%	1050 ms	0.0 rpm	170 K	0.00 rpm
Portal::CandidatesController#new	11.5%	749 ms	0.0 rpm	210 K	0.00 rpm
Api::V1::CandidatesController#create	1.8%	634 ms	0.0 rpm	76 K	0.00 rpm
Backend::PasswordsController#create	1.6%	580 ms	0.0 rpm	18 K	0.00 rpm
Api::V1::CompaniesController#show	5.6%	575 ms	0.0 rpm	120 K	0.00 rpm

Scout App - Detail request



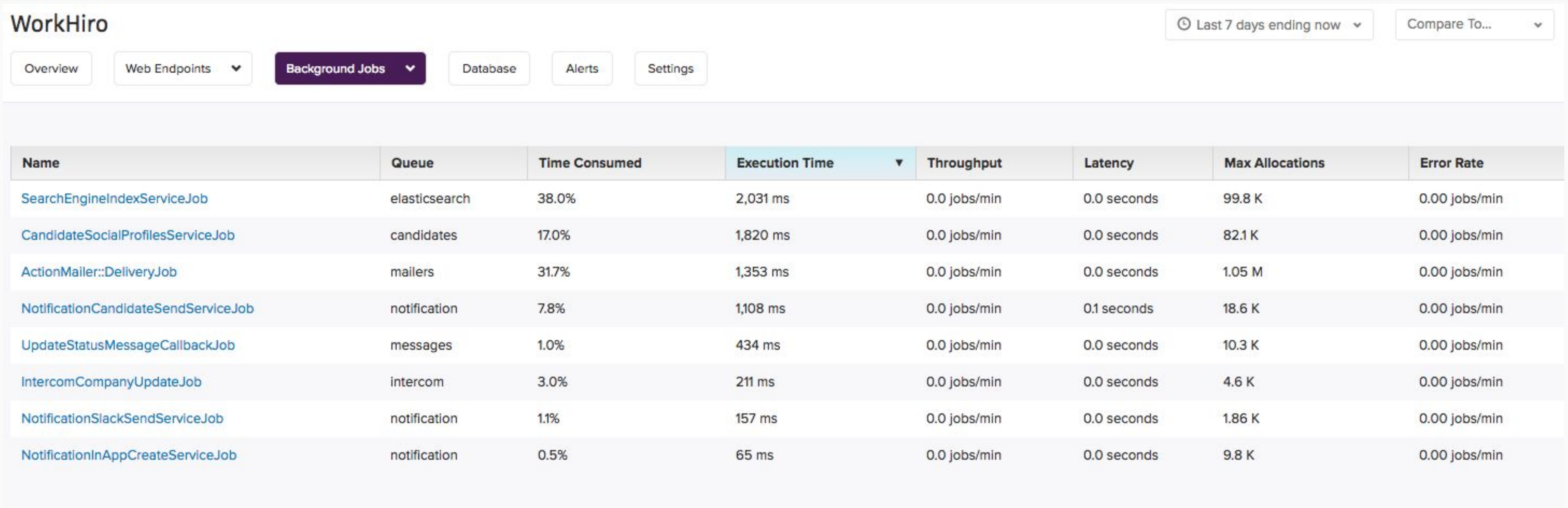
← Api::V1::CandidatesController#import_resume

Fri 12:27:00 AM · /api/v1/candidates/import/resume

Response Time	Controller	HTTP	Other
14,584 ms	91%	7%	2%

Controller		1 call	13,349.4 ms
HTTP#request	URL	1 call	1,061.0 ms
User#find	SQL 1 ROW	1 call	69.6 ms
Middleware		1 call	47.5 ms
Company#find	SQL 1 ROW	1 call	34.5 ms
Position#find	SQL 2 ROWS	1 call	15.5 ms
Doorkeeper::AccessToken#find	SQL 1 ROW	1 call	3.5 ms
Router/Rails		1 call	3.0 ms

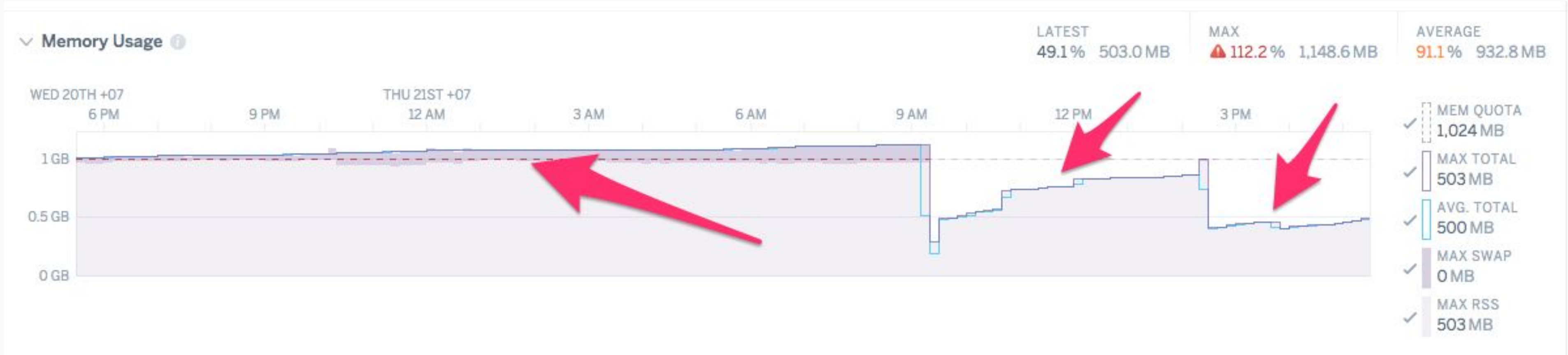
Scout App - Background jobs



Name	Queue	Time Consumed	Execution Time ▼	Throughput	Latency	Max Allocations	Error Rate
SearchEngineIndexServiceJob	elasticsearch	38.0%	2,031 ms	0.0 jobs/min	0.0 seconds	99.8 K	0.00 jobs/min
CandidateSocialProfilesServiceJob	candidates	17.0%	1,820 ms	0.0 jobs/min	0.0 seconds	82.1 K	0.00 jobs/min
ActionMailer::DeliveryJob	mailers	31.7%	1,353 ms	0.0 jobs/min	0.0 seconds	1.05 M	0.00 jobs/min
NotificationCandidateSendServiceJob	notification	7.8%	1,108 ms	0.0 jobs/min	0.1 seconds	18.6 K	0.00 jobs/min
UpdateStatusMessageCallbackJob	messages	1.0%	434 ms	0.0 jobs/min	0.0 seconds	10.3 K	0.00 jobs/min
IntercomCompanyUpdateJob	intercom	3.0%	211 ms	0.0 jobs/min	0.0 seconds	4.6 K	0.00 jobs/min
NotificationSlackSendServiceJob	notification	1.1%	157 ms	0.0 jobs/min	0.0 seconds	1.86 K	0.00 jobs/min
NotificationInAppCreateServiceJob	notification	0.5%	65 ms	0.0 jobs/min	0.0 seconds	9.8 K	0.00 jobs/min

Improve the Rails App

- Large response time
  - Refactor the code, move code (if can) to Background Job, optimize the SQL query, …
- Large memory allocate, memory leak:
  - Refactor the code, avoid to create a global variable, immutable variable.
  - Improve the Garbage Collection in Ruby, use tunemygc.com
  - Memory of the app still large and it increase the memory out of the server memory. Use the puma_worker_killer



Thanks!

Contact Nimbl3

hello@nimbl3.com

399 Sukhumvit Road, Interchange 21
Klongtoey nua, Wattana
Bangkok 10110

20th Floor, Central Tower
28 Queen's Road
Central, Hong Kong

nimbl3.com