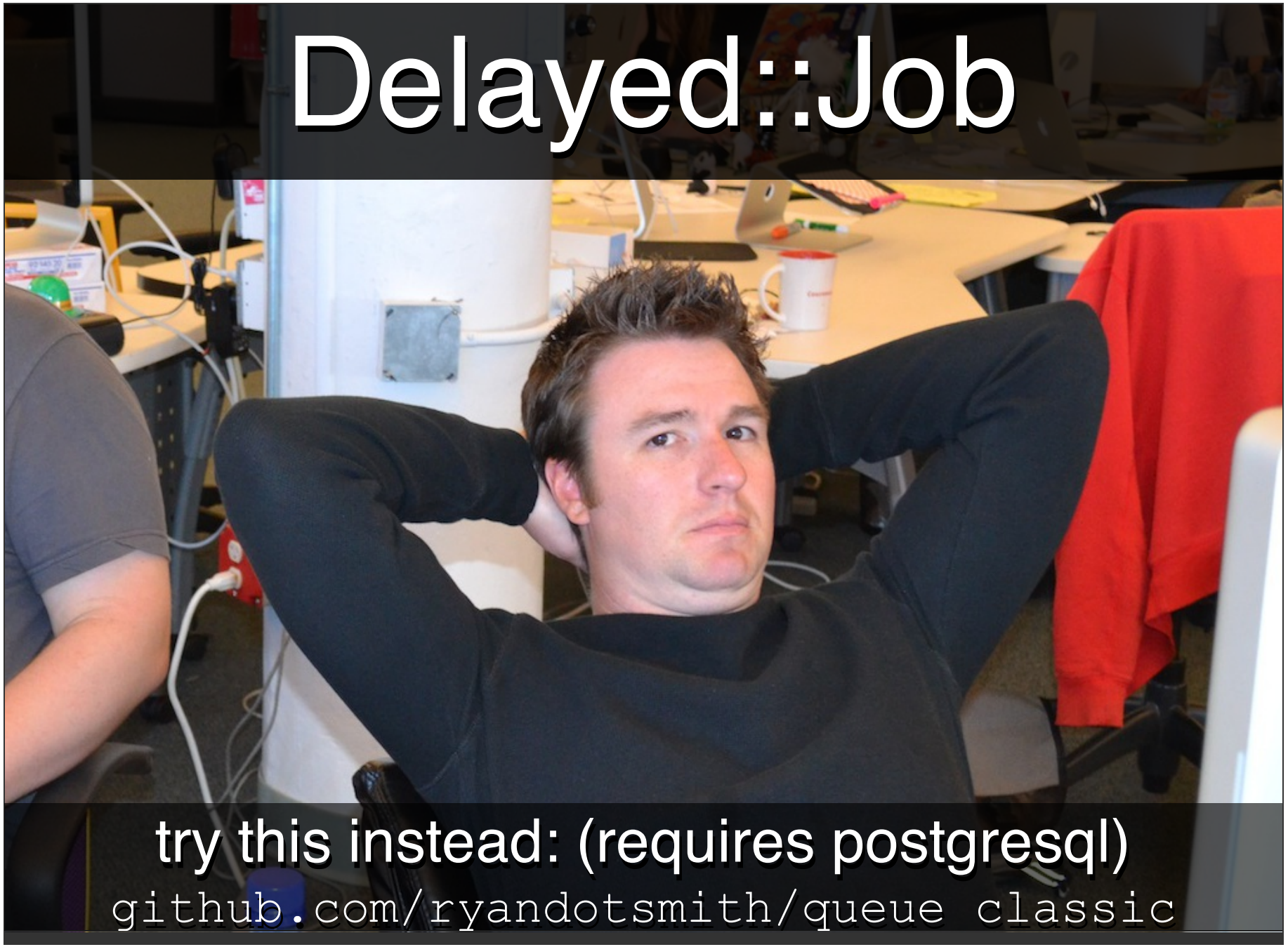
Jacob Burkhart

How to Fail at Background Jobs

jacobo.github.com/background_jobs

Delayed::Job



try this instead: (requires postgresql)

`github.com/ryandotsmith/queue_classic`

The End

distill.engineyard.com

CFP closes Tuesday, April 9

Seeking Better Abstractions for Background Jobs

jacobo.github.com/background_jobs

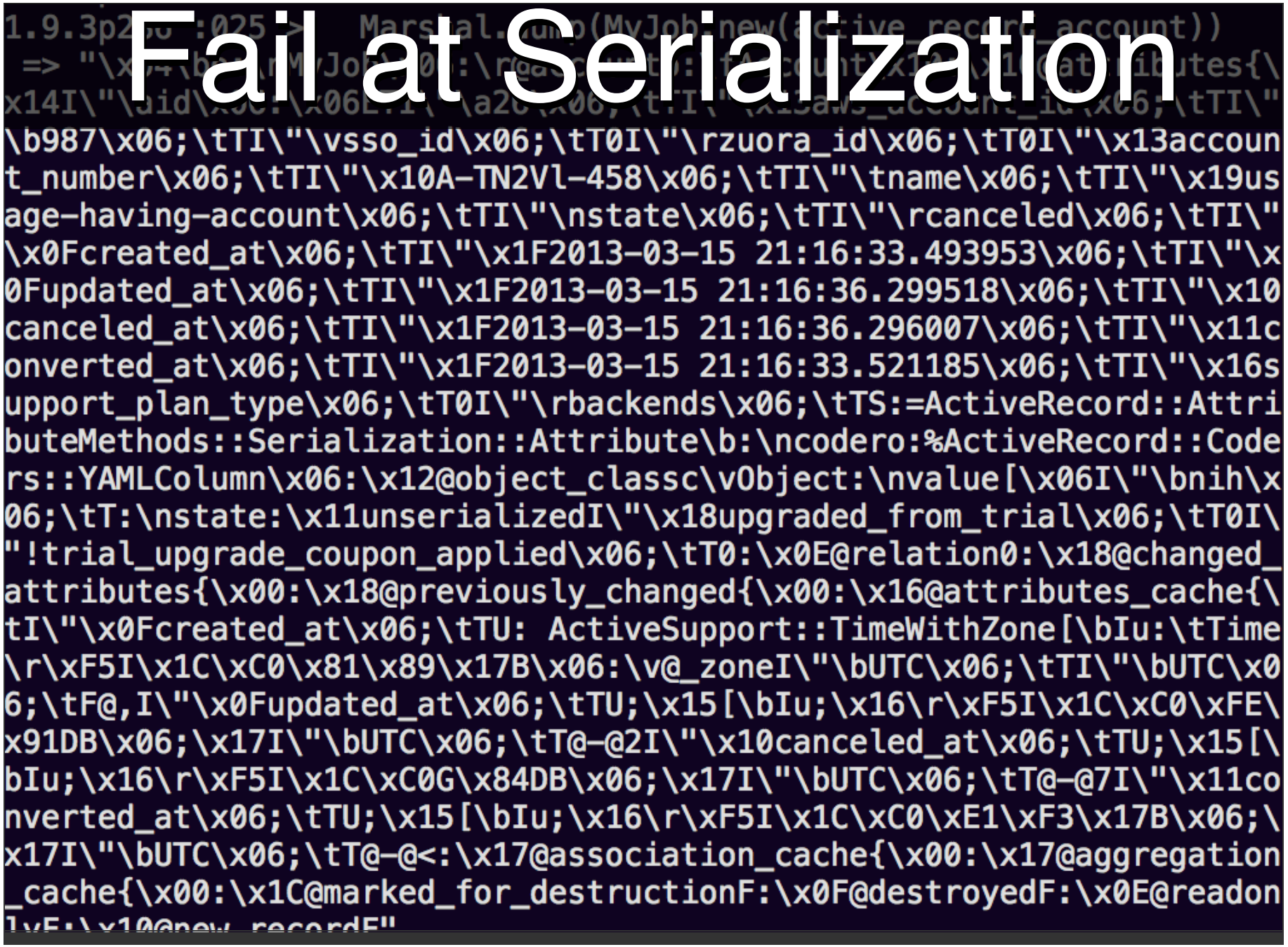
Rails 4 Queuing

github.com/rails/rails/commit/f9da785d0b1b22317cfca25c15fb555e9016accb

The API

```
class MyJob

  def initialize(account)
    @account = account
  end

  def run
    puts "working on #{@account.name}..."
  end

end

Rails.queue[:jobs].push(MyJob.new(account))
```

Fail at Serialization



Solve the wrong problem

```
class BodyProxy
  def each
    @body.each{|x| yield x}
    while(email = CleverMailer.emails_to_send.pop)
      email.deliver
    end
  end
end

def call(env)
  status, headers, body = @app.call(env)
  headers["Content-Length"] = body.map(&:bytesize).sum
  [status, headers, BodyProxy.new(body)]
end
```

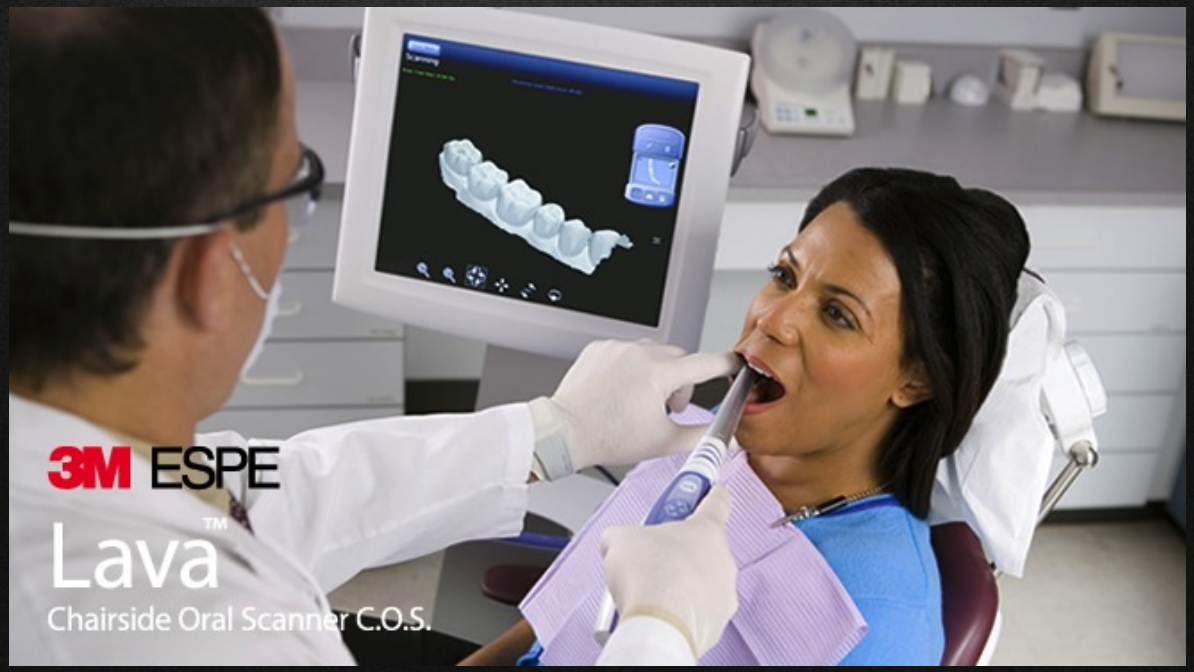
Will be ~~revisited~~ re-written in 4.1

github.com/rails/rails/pull/9910

github.com/rails/rails/pull/9924

Moving on...

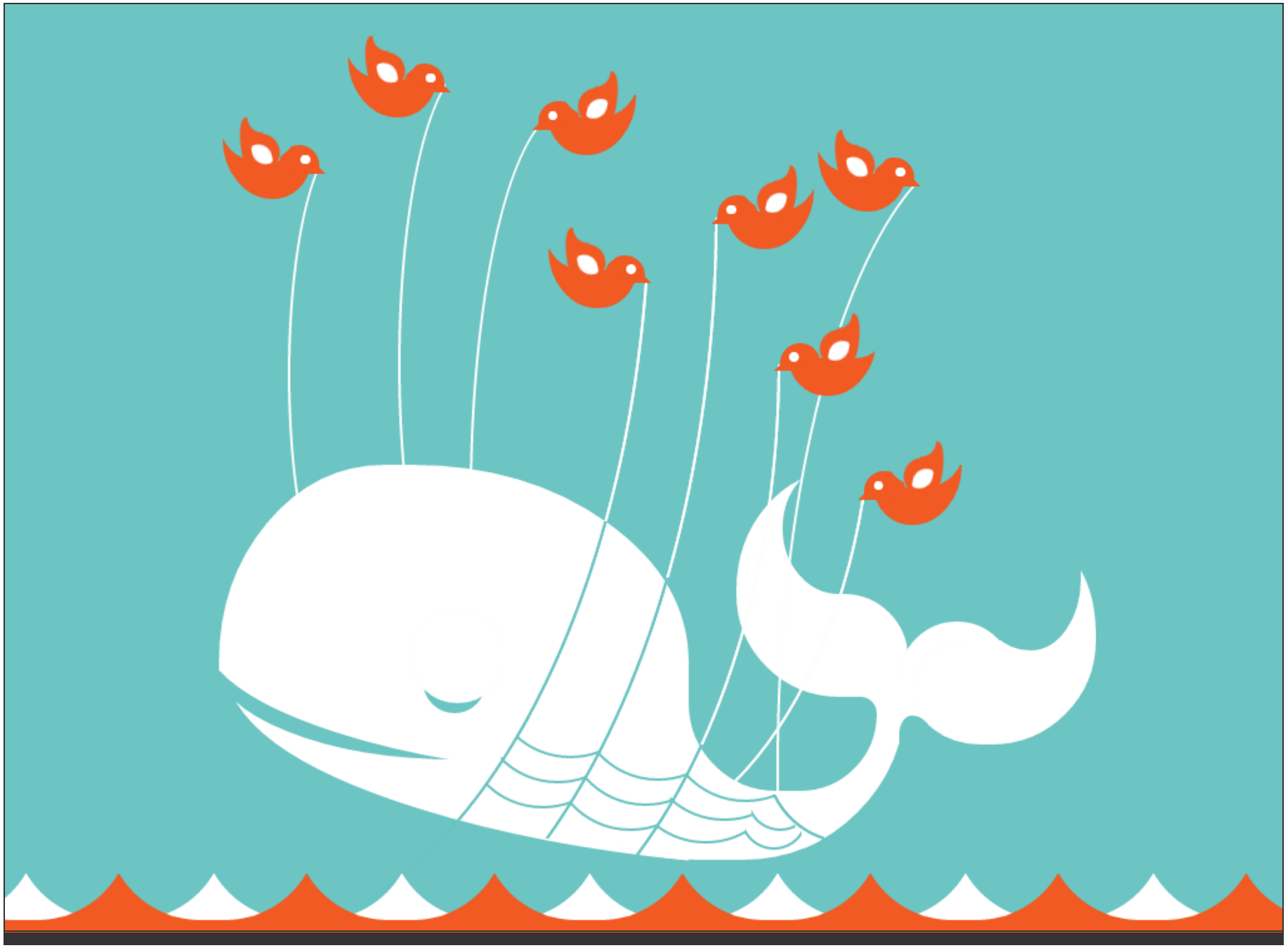
2009

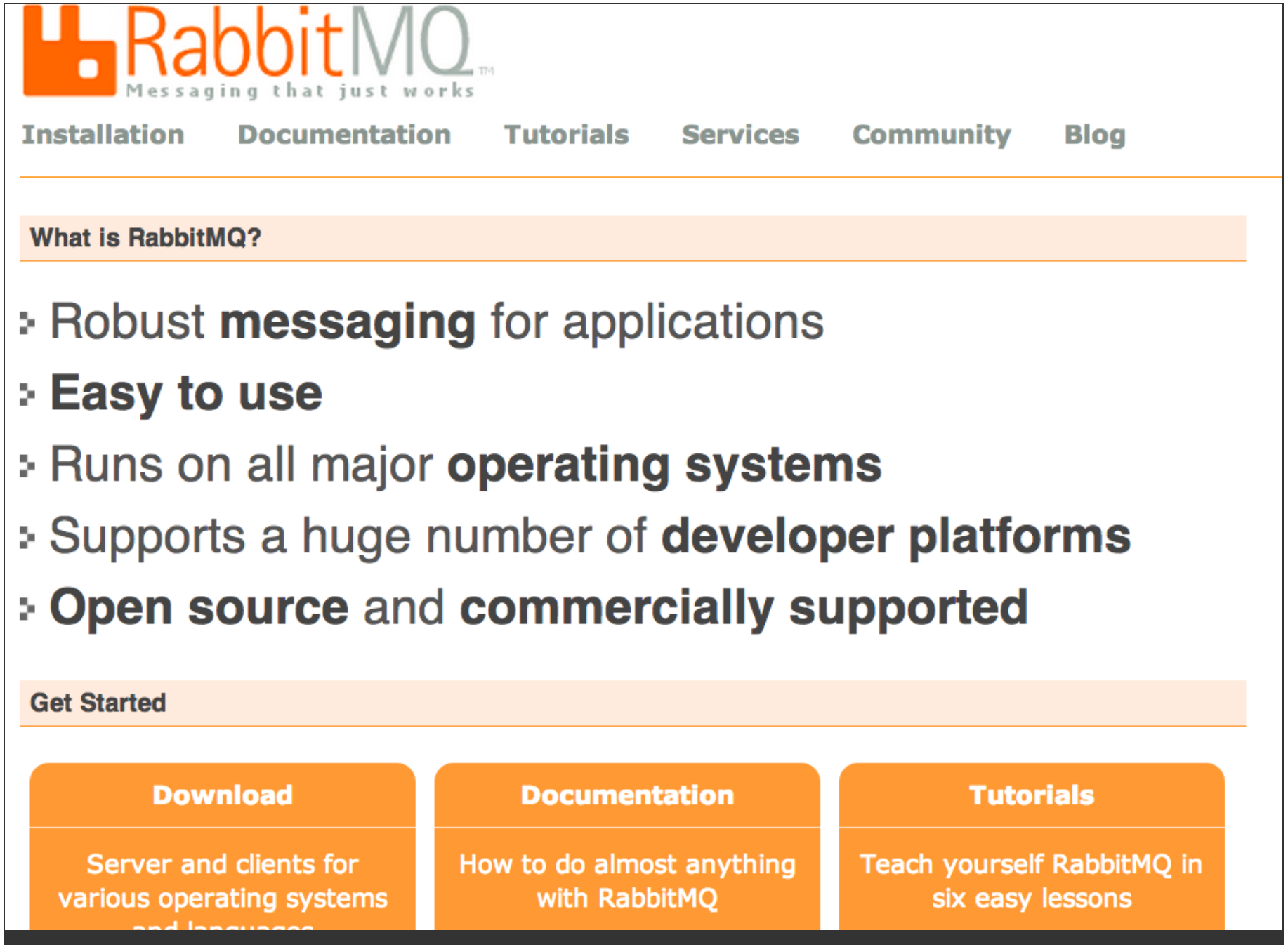


Starling, Workling

```
class FileCopier < Workling::Base
  def copy_case_files(options)
    ...
  end
end
FileCopier.async_copy_case_files(...)

Workling::Remote.dispatcher =
  Workling::Remote::Runners::StarlingRunner.new
```

```
starling -f config/starling.yml
script/workling_client run
```



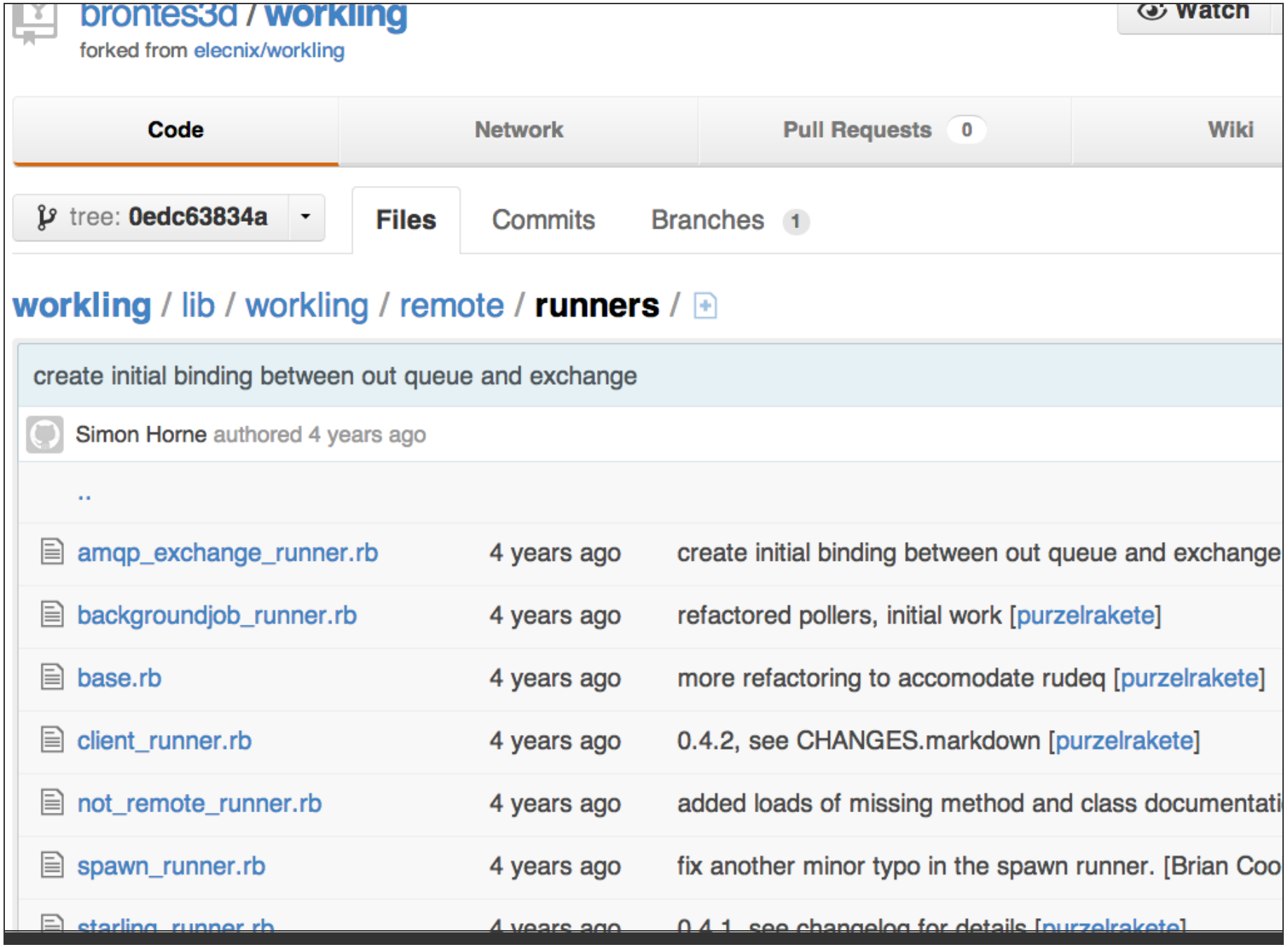

RabbitMQ
Messaging that just works
Installation
Documentation
Tutorials
Services
Community
Blog
What is RabbitMQ?
Robust messaging for applications
Easy to use
Runs on all major operating systems
Supports a huge number of developer platforms
Open source and commercially supported
Get Started
Download
Server and clients for various operating systems
Documentation
How to do almost anything with RabbitMQ
Tutorials
Teach yourself RabbitMQ in six easy lessons

AMQP

```ruby
connection = AMQP.connect(:host => '127.0.0.1')

channel  = AMQP::Channel.new(connection)
queue    = channel.queue("some.q")
exchange = channel.default_exchange

queue.subscribe do |payload|
  puts "Received a message: #{payload}"
end

exchange.publish "Hello, world!",
                 :routing_key => queue.name
```

github.com/ruby-amqp/amqp

github.com/ruby-amqp/bunny


brontes3d / workling
forked from elecnix/workling
Watch
Code
Network
Pull Requests 0
Wiki
tree: 0edc63834a
Files
Commits
Branches 1
workling / lib / workling / remote / runners /
create initial binding between out queue and exchange
Simon Horne authored 4 years ago
..
amqp_exchange_runner.rb 4 years ago create initial binding between out queue and exchange
backgroundjob_runner.rb 4 years ago refactored pollers, initial work [purzelrakete]
base.rb 4 years ago more refactoring to accomodate rudeq [purzelrakete]
client_runner.rb 4 years ago 0.4.2, see CHANGES.markdown [purzelrakete]
not_remote_runner.rb 4 years ago added loads of missing method and class documentati
spawn_runner.rb 4 years ago fix another minor typo in the spawn runner. [Brian Coo
starling_runner.rb 4 years ago 0.4.1, see changelog for details [purzelrakete]

Little Bug

ActiveRecord::RecordNotFound

```
class Device < ActiveRecord::Base
  after_create do |d|
    BackgroundJob.async_run(d.id)
  end
end

class BackgroundJob
  def run(device_id)
    #sleep 1 ?
    Device.find(device_id)
  end
end
```

Hack ActiveRecord

```
class Device < ActiveRecord::Base
  after_create do |d|
    d.commit_callback do
      BackgroundJob.async_run(d.id)
    end
  end
end

...
def commit_callback
  self.connection.instance_eval do
    class << self
      alias commit_db_transaction_original commit_db_transac
      ...
```

github.com/brontes3d/commit_callback

(rails 2.3 only)

More Bugs

Fundamental Flaw

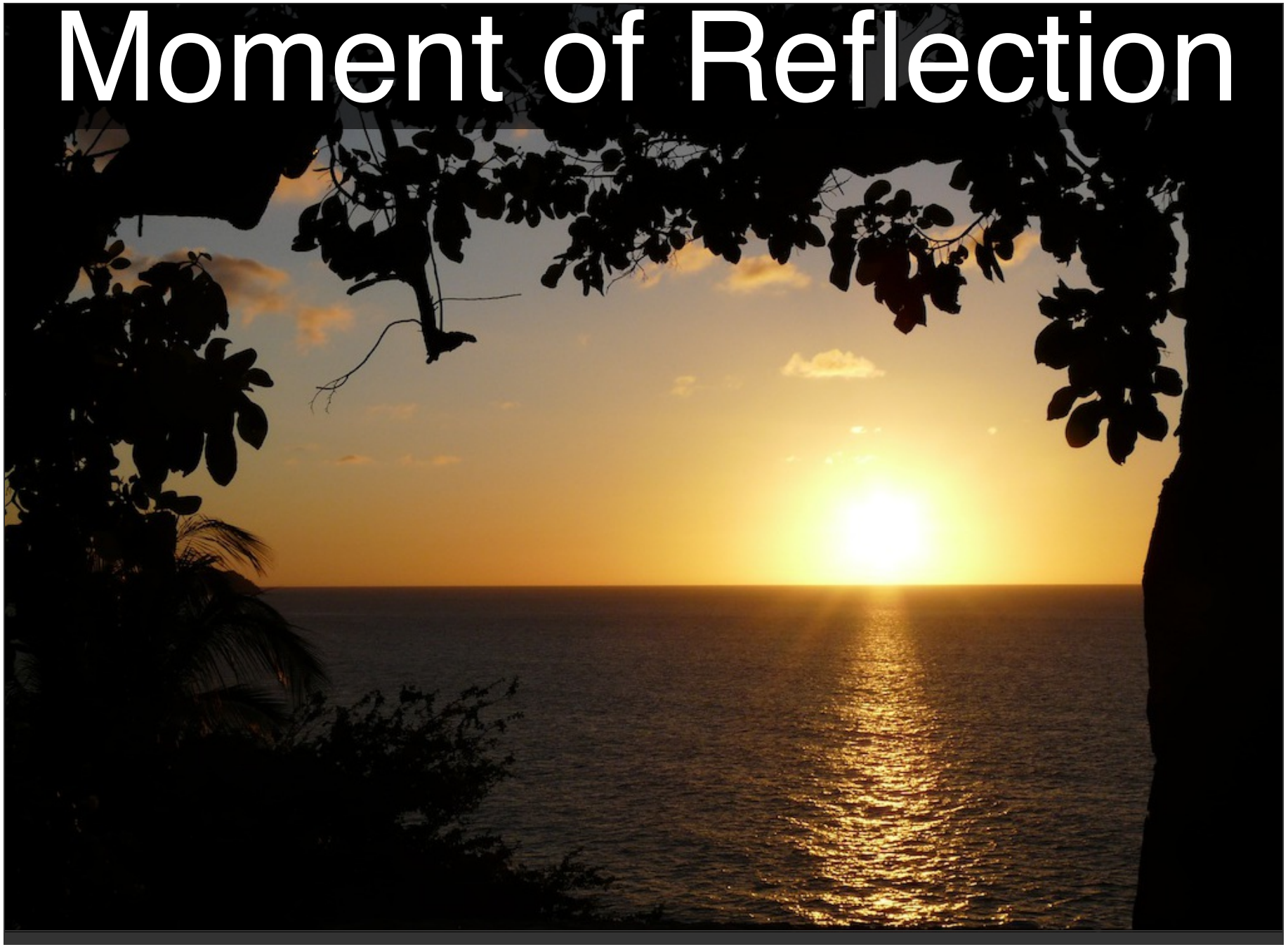
Do it Yourself

```
class FileCopier < AmqpListener::Listener
  subscribes_to :case_file_copy_requests

  def handle(options)
    ...
  end
end

FileCopier.notify(...)
```

```
script/amqp_listener run
```

github.com/brontes3d/amqp_listener

Moment of Reflection



Trains

... and Resque

Boot an EC2 Server

```
class InstanceProvision

  def self.perform(instance_id)
    instance = Instance.find(instance_id)

    fog = Fog::Compute.new(...)
    server = fog.servers.create(...)
    instance.amazon_id = server.id

    while(!server.ready?)
      sleep 1
      server.reload
    end

    instance.attach_ip!
    ...
```

Extractable?

```
class InstanceProvision

  def self.perform(aws_creds, create_params, callback_url)
    fog = Fog::Compute.new(aws_creds)
    server = fog.servers.create(create_params)
    API.post(callback_url, :instance_amazon_id => server.id)

    while(!server.ready?)
      sleep 1
      server.reload
    end

    ip = fog.ips.create!
    ip.server = server
    API.post(callback_url, :attached_ip_amazon_id => ip.id)
    ...
```

Generalizable?

```ruby
class MethodCalling
  def self.perform(class_str, method, id, *args)
    model_class = Object.const_get(class_str)
    model = model_class.find(id)
    model.send(method, *args)
  end
end

class Instance
  def provision
    Resque.enqueue(MethodCalling, Instance, :provision!, id)
  end

  def provision!
    #actually do it
  end
end
```

Async

```
require 'async'
require 'async/resque'
Async.backend = Async::ResqueBackend

class Instance < ActiveRecord::Base
  def provision(*args)
    Async.run{ provision_now(*args)}
  end
  def provision_now(*args)
    #actually do it
  end
end
```

github.com/engineyard/async

Application Instances

Application High CPU Medium (32 bit)	i-f3b19dc1 SSH / HTTP	Logs: Base / Custom Performance graphs	Terminate
Waiting - about 6 hours ago			
Waiting - about 6 hours ago			
Waiting - about 6 hours ago			
Application Master High CPU Medium (32 bit)	i-8fb19dbd SSH / HTTP	Logs: Base / Custom Performance graphs	
Waiting - about 6 hours ago			
Waiting - about 6 hours ago			
Waiting - about 6 hours ago			

Database Instances

Master Database High CPU Medium (32 bit)	i-85b19db7 SSH	Logs: Base / Custom Performance graphs
Waiting - about 6 hours ago		
Waiting - about 6 hours ago		
Waiting - about 6 hours ago		

More Options

Edit Environment Make Stack Changes	**Crontabs** Add or Edit a Cron Job

46 of 157 Workers Working

The list below contains all workers which are currently running a job.

	Where	Queue	Processing
	ip-25-100-68-98:11991	INSTANCE_PROVISIONING	InstanceProvisioning about 14 hours ago
	ip-25-5-66-63:18066	INSTANCE_PROVISIONING	InstanceProvisioning about 12 hours ago
	domU-22-31-38-5F-01-E1:6053	INSTANCE_PROVISIONING	InstanceProvisioning about 10 hours ago
	domU-22-31-38-5E-51-31:6379	INSTANCE_PROVISIONING	InstanceProvisioning about 10 hours ago
	ip-25-89-63-160:31812	INSTANCE_PROVISIONING	InstanceProvisioning about 8 hours ago
	ip-25-126-111-88:19063	INSTANCE_PROVISIONING	InstanceProvisioning about 7 hours ago
	domU-22-31-38-5E-51-31:6449	INSTANCE_PROVISIONING	InstanceProvisioning about 7 hours ago
	ip-25-80-39-43:24306	INSTANCE_PROVISIONING	InstanceProvisioning about 7 hours ago
	domU-22-31-38-5E-51-31:6760	INSTANCE_PROVISIONING	InstanceProvisioning about 4 hours ago
	domU-22-31-38-5E-51-31:6670	INSTANCE_PROVISIONING	InstanceProvisioning about 2 hours ago
	domU-22-31-38-5E-51-31:6370	INSTANCE_PROVISIONING	InstanceProvisioning about 2 hours ago
	ip-25-80-39-43:24268	INSTANCE_PROVISIONING	InstanceProvisioning about 2 hours ago
	ip-25-126-111-88:19161	INSTANCE_PROVISIONING	InstanceProvisioning about 2 hours ago
	ip-25-89-63-160:31872	INSTANCE_PROVISIONING	InstanceProvisioning about 2 hours ago

Reliability

Reliable Queue vs Reliable Job

Retry (raise, crash, hang). Time-out

Monitor / Maintain N workers

Graceful Restart

Did it run? Did it fail? How? Where?

Make a Resque Plugin

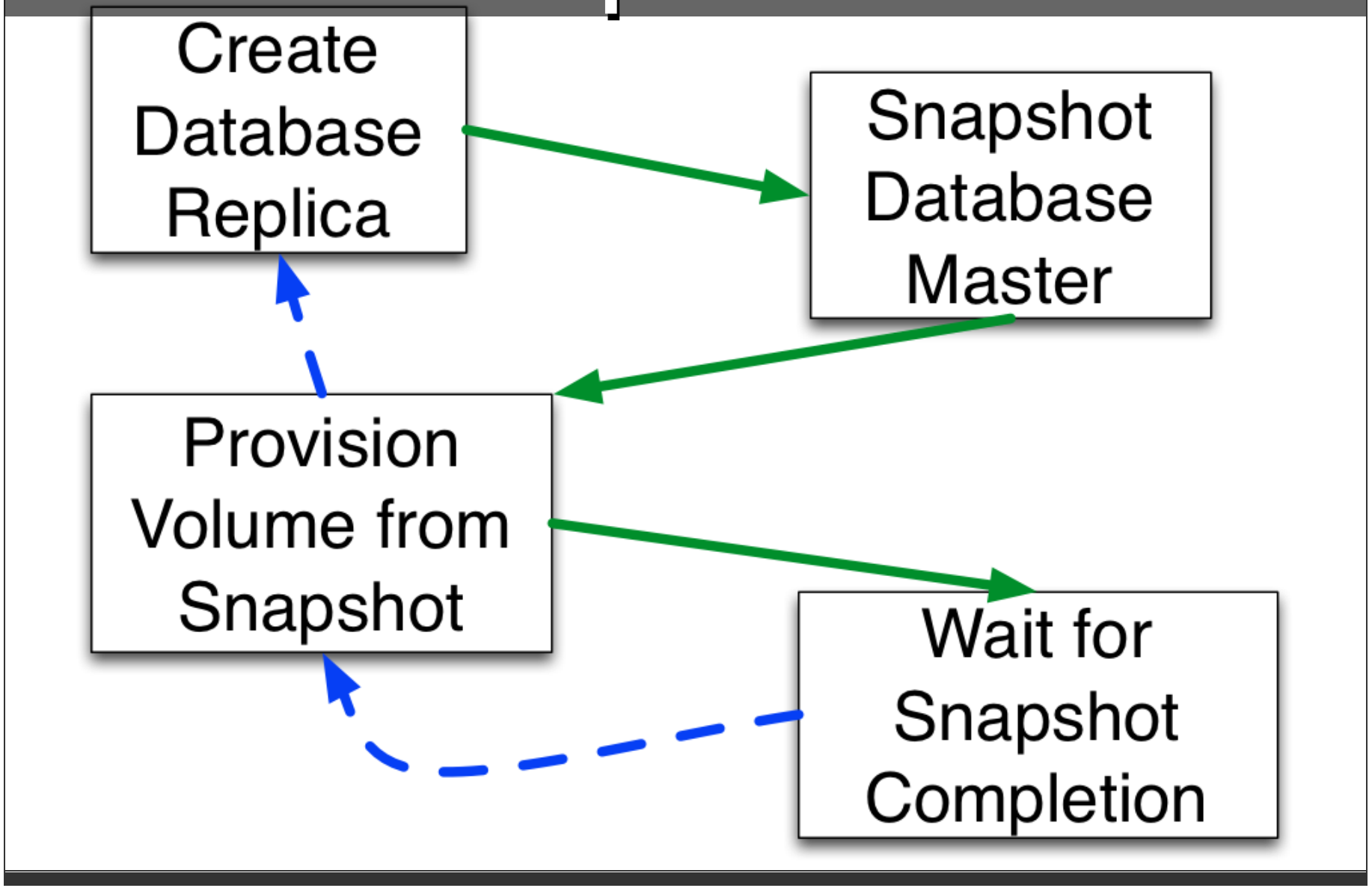
```
class InstanceProvision
  extend Resque::Plugins::JobTracking

  def self.track(instance_id, opts)
    i = Instance.find(instance_id)
    ["Account:#{i.account_id}",
     "Instance:#{instance_id}"]
  end

  def self.perform(instance_id, opts)
    #do stuff
  end
end
```

github.com/engineyard/resque-job-tracking

Helps a little?

```
InstanceProvision.pending_jobs("Instance:532")

InstanceProvision.failed_jobs("Account:121")
```

Jobs Dependencies



Make another

```
class Sandwich
  extend Resque::Plugins::Delegation

  def self.steps(tomato_color, cheese_please, cheesemaker)
    step "fetch a", :tomato do
      depend_on(Tomato, tomato_color)
    end
    step "slice the ", :tomato, " and make", :tomato_slices
      tomato.split(",")
    end
    step "fetch the", :cheese_slices do
      if cheese_please
        depend_on(Cheese, cheesemaker)
        ...
```

github.com/engineyard/resque-delegation

Desperation?

```
class ResqueJob < ActiveRecord::Base
end
```

```
class AbstractJob

  def self.store_meta(meta)
    meta_id = meta["meta_id"]
    ResqueJob.create!(meta.slice(
      "meta_id", "job_class",
      "enqueued_at", "started_at", "finished_at"))
    super(meta)
  end

...
```

Maybe we just need better logging?

http://youtu.be/NpTT30wLL-w

Model Intent

Data belongs in a database (not redis)

```
class InstanceProvision < ActiveRecord::Base
  belongs_to :instance

  def run(instance_id)
    fog = Fog::Compute.new(...)
    server = fog.servers.create(...)
    instance.amazon_id = server.id

    while(!server.ready?)
      sleep 1
      server.reload
    end

    instance.attach_ip!
```

requested_at

started_at

finished_at

state

Idempotent

```
class InstanceProvision < ActiveRecord::Base
  belongs_to :instance

  def run(instance_id)
    with_lock do
      if self.state == "running"
        raise
      else
        self.state = "running"
        self.started_at = Time.now
        save!
      end
    end
    ...
```

Non-Resque specific job-middleware

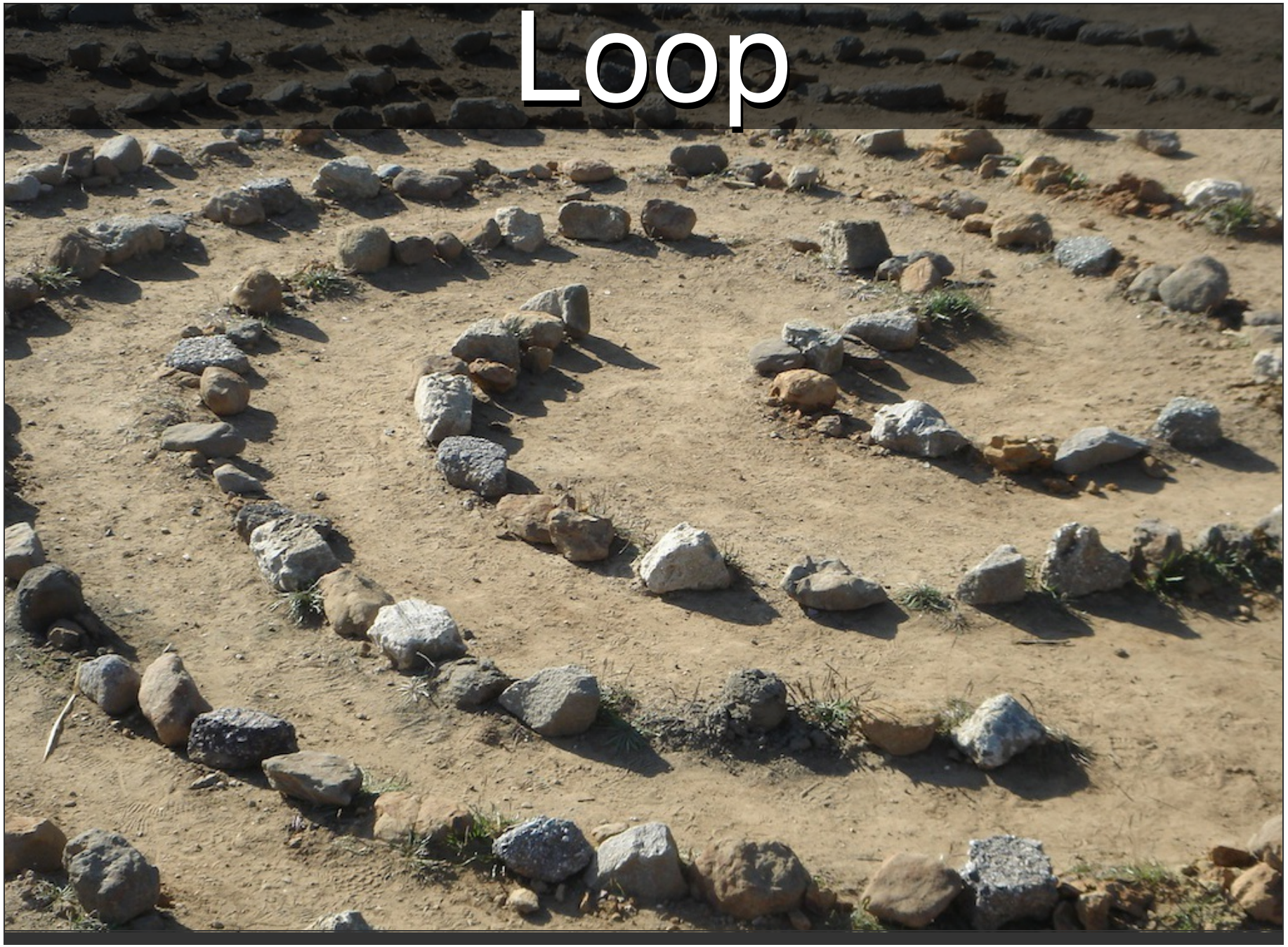
```
procedure = Viaduct::Builder.new do
  use Instrumetation
  use ProvisionNotification
  use ProvisionInstance
end
Viaduct::Runner.new.run(procedure, data)
```

github.com/slack/viaduct

Moment of Reflection


3 Ingredients
Work Loop
Monitor & Restart
Queue

Loop



Resque Event Loop

```
def work(interval = 5, &block)
  loop do
    run_hook :before_fork, job

    if @child = fork
      procline "Forked #{@child} at #{Time.now.to_i}"
      Process.wait
    else
      procline "Processing #{job.queue} since #{Time.now.to_
      perform(job, &block)
      exit! unless @cant_fork
    end
  end
```

github.com/defunkt/resque/blob/master/lib/resque/worker.rb

EventMachine

```
void EventMachine_t::Run()
  //Epoll and Kqueue stuff..
  ...

  while (true) {
    _UpdateTime();
    _RunTimers();

    _AddNewDescriptors();
    _ModifyDescriptors();

    _RunOnce();
    if (bTerminateSignalReceived)
      break;
  }
}
```

github.com/eventmachine/eventmachine/blob/master/ext/em.cpp

EM.next_tick

```
require 'eventmachine'
EM.run {
  EM.start_server(host, port, self)
}

EM.next_tick{ puts "do something" }
```

Celluloid / Sidekiq

```
class Sidekiq::Manager
  include Celluloid
  ???

class Sidekiq::Fetcher
  include Celluloid
  ???

class Sidekiq::Processor
  include Celluloid
  ???
```

sidekiq.org

github.com/celluloid/celluloid

Monitor & Restart

Resque Signals

`QUIT` - Wait for child to finish then exit

`TERM` / `INT` - Immediately kill child, exit

`USR1` - Immediately kill child, don't exit

`USR2` -Don't start to process new jobs

`CONT` - Start to process new jobs again after a USR2

Monitor with God

```
5.times do |n|
  God.watch do |w|
    w.name      = "resque-#{num}"
    w.group     = 'resque'
    w.interval = 30.seconds
    w.log       = "#{app_root}/log/worker.#{num}.log"
    w.dir       = app_root
    w.env       = {
      "GOD_WATCH"    => w.name,
      "QUEUE"        => '*'
    }
    w.start     = "bundle exec rake --trace resque:work"
  ...
```

godrb.com

Or Daemons

```ruby
require 'daemons'

options = {
  :app_name => "worker",
  :log_output => true,
  :backtrace => true,
  :dir_mode => :normal,
  :dir => File.expand_path('../../tmp/pids',  __FILE__),
  :log_dir => File.expand_path('../../log',  __FILE__),
  :multiple => true,
  :monitor => true
}

Daemons.run(File.expand_path('../worker',  __FILE__), option
```

daemons.rubyforge.org

Or Don't?

torquebox.org

Queue

Thread-safe Queue

```
module BundlerApi
  class ConsumerPool

    def initialize
      @queue = Queue.new
    end

    def enq(job)
      @queue.enq(job)
    end

    ...
      job = @queue.deq
      job.run
```

github.com/rubygems/bundler-api/blob/master/lib/bundler_api/update/consumer_pool.rb

DRB objects

```
class MultiHeadedGreekMonster
  class ServiceManager
    def initialize(progress)
      @things = []
    end
    def give(thing)
      @things.unshift(thing)
    end
    def take
      @things && @things.pop
    end

...
DRb.start_service "druby://localhost:#{@on_port}",
                  ServiceManager.new(@progress)
```

github.com/engineyard/multi_headed_greek_monster/blob/master/lib/multi_headed_greek_monster.rb

Qu Push/Pop Redis

```
payload.id = SimpleUUID::UUID.new.to_guid
redis.set("job:#{payload.id}",
  encode('klass' => payload.klass.to_s,
         'args' => payload.args))
redis.rpush("queue:#{payload.queue}", payload.id)
redis.sadd('queues', payload.queue)

...

redis.lpop(queue)
```

github.com/bkeepers/qu/blob/master/lib/qu/backend/redis.rb

Qu Push/Pop Mongo

```
payload.id = SimpleUUID::UUID.new.to_guid
payload.id = BSON::ObjectId.new
jobs(payload.queue).insert({
  '_id' => payload.id,
  'klass' => payload.klass.to_s, 'args' => payload.args})
self[:queues].update({'name' => payload.queue},
  {'name' => payload.queue}, 'upsert' => true)

...

if doc = jobs(queue).find_and_modify(:remove => true)
  doc['id'] = doc.delete('_id')
  return Payload.new(doc)
end
```

github.com/bkeepers/qu/blob/master/lib/qu/backend/mongo.rb

All Together

script/invoice_worker

```
#!/usr/bin/env ruby
require File.expand_path('../../config/environment',
  __FILE__)

TrapLoop.start do
  InvoiceProcessingTask.process_invoices!
end
```

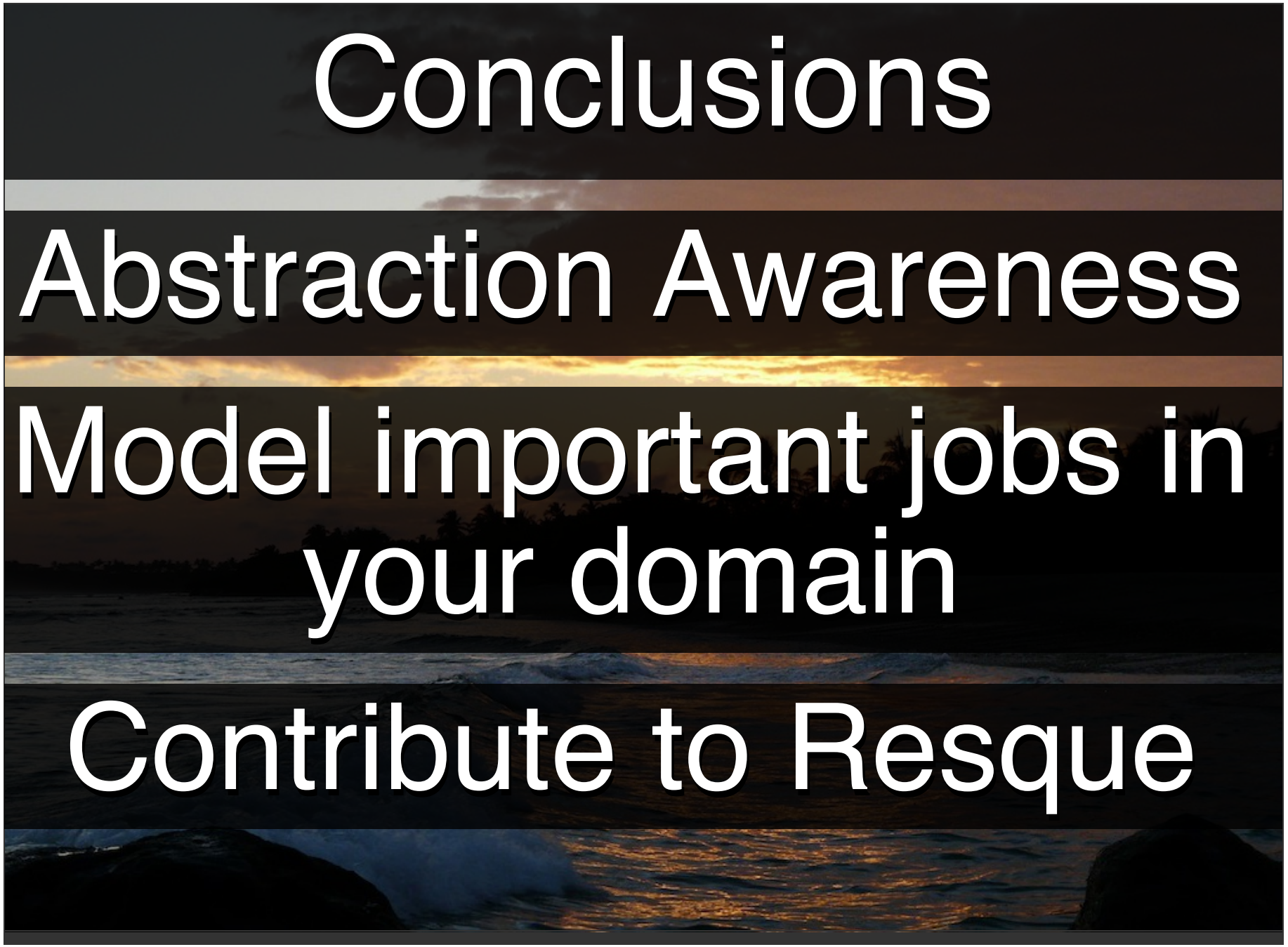
script/invoice_runner start

script/invoice_runner stop

```
require 'daemons'
Daemons.run(File.expand_path('../invoice_worker',  __FILE__)
```

```ruby
class TrapLoop
  trap('TERM') { stop! }
  trap('INT')  { stop! }
  trap('SIGTERM') { stop! }

  def self.stop!
    @loop = false
  end

  def self.safe_exit_point!
    if @started && !@loop
      raise Interrupt
    end
  end

  def self.start(&block)
    @started = true
    @loop = true
    while(@loop) do
      yield
    end
  end
```

```ruby
class InvoiceProcessingTask
  def self.enq_task(task_id, invoice_id)
    REDIS.rpush("tasks:#{invoice_id}", task_id)
    REDIS.rpush("invoices", invoice_id)
  end

  def self.process_invoices!
    while(invoice_id = REDIS.lpop("invoices"))
      process_tasks!(invoice_id)
      TrapLoop.safe_exit_point!
    end
  end

  def self.process_tasks!(invoice_id)
    while(task_id = REDIS.lpop("tasks:#{invoice_id}"))
      if task = InvoiceProcessingTask.find_by_id(task_id)
        task.process!
        TrapLoop.safe_exit_point!
      end
    end
  end
```


Conclusions
Abstraction Awareness
Model important jobs in your domain
Contribute to Resque

Questions?

jacobo.github.com/background_jobs

@beanstalksurf

jacobo.github.com/talks