Keeping it Ruby:

Why Your Product Needs a Ruby SDK

Sampo Kuokkanen, Andrey Novikov

Evil Martians

RubyWorld Conference 2024

05 December 2024

Sampo Kuokkanen

- Head of Evil Martians Japan
- Ruby enthusiast
- A fan of imgproxy

Andrey Novikov

- Ruby developer at Evil Martians
- Open source enthusiast
- imgproxy early adopter

EVIL MARTIANS

evilmartians.com

イービル・マーシャンズ

邪悪な火星人？

evilmartians.jp

Martian Open Source

Ruby Next makes modern Ruby code run in older versions and alternative implementations

Yabeda: Ruby application instrumentation framework

Lefthook: git hooks manager

AnyCable: Polyglot replacement for ActionCable server

PostCSS: A tool for transforming CSS with JavaScript

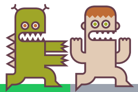
Today's topic

Imgproxy: Fast and secure standalone server for resizing and converting remote images

Overmind: Process manager for Procfile-based applications and tmux

Even more at evilmartians.com/oss

Ruby in 2024: Still Going Strong



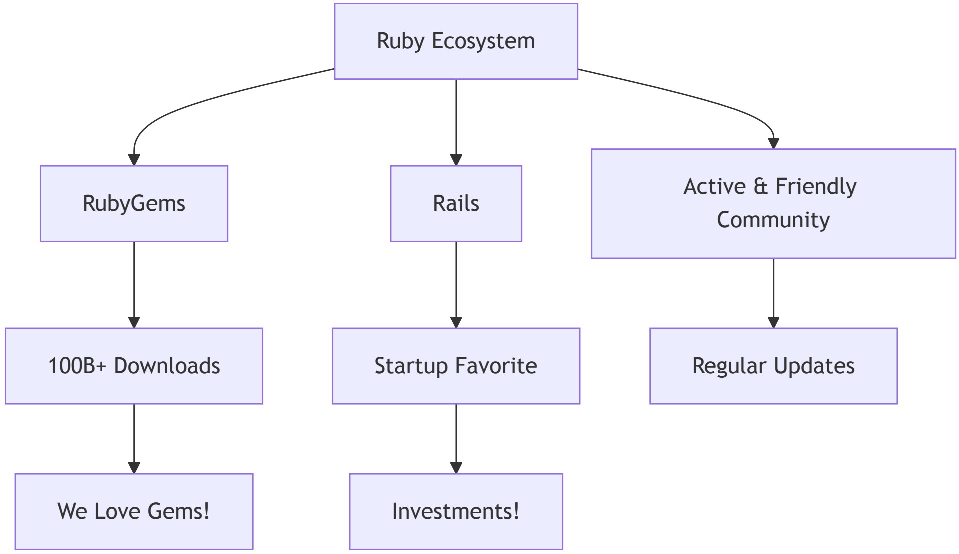
Ruby's Continuing Popularity

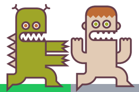
RubyGems Downloads

- Over 100 billion total downloads
- Growing year over year
- Active ecosystem

GitHub Statistics

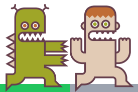
- Top 10 most popular language!
- Strong in web development
- Active community





We 💓 Ruby

But sometimes it is just not right tool for the job

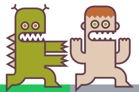


The common problem for any web app

Handling images uploaded by users: profile pictures, product photos, reviews, …

We need to store them and show in various places, of course! And for this we need to:

- Generate thumbnails to save bandwidth
- Crop to fit design
- Add watermarks to prevent theft
- …



“Classic” way

- **Upload image to the server**

  Probably among other form fields

- **Store it somewhere**

  Often on S3 or other cloud storage

- **Generate all required thumbnails**

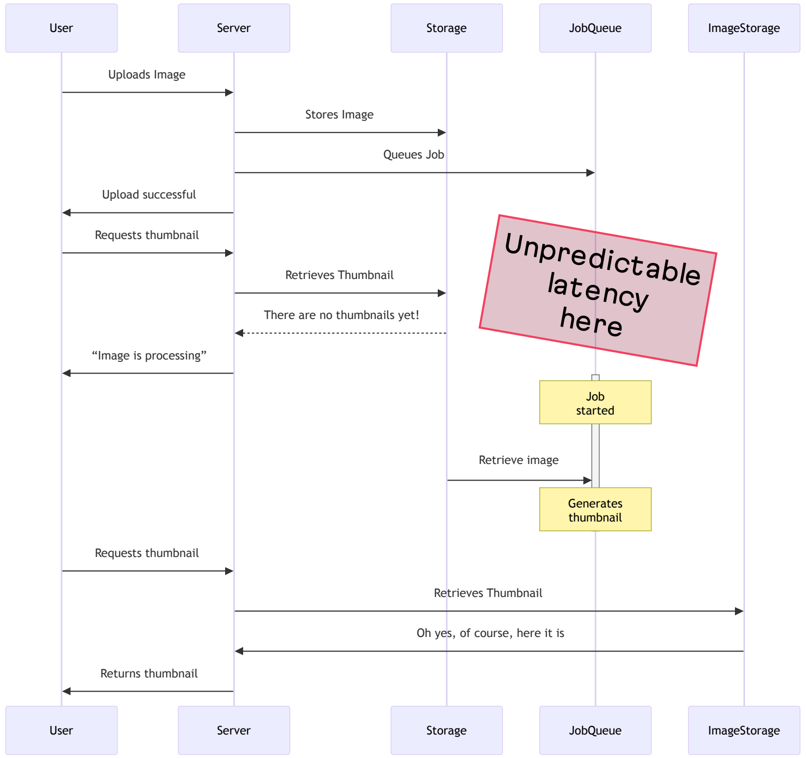
  As many as your design requires

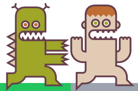
- **Store them somewhere**

  Again S3 or other cloud storage

- **Serve them to the user**

  CDN will help here





Problems of “classic” approach

- Hard to predict latency: **background jobs can queue**

  It can take a while to get your image processed, and “image is processing” fallbacks are ugly

- Hard to add new variants: **need to reprocess all images**

  Possibly millions of jobs to run before enabling it on the front-end

- And hard to clean up old ones

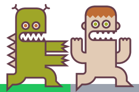
  Space is cheap, but not free

- Deployment: **gets complicated**

  You need to install ImageMagick or libvips on all servers/containers

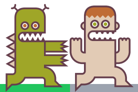
- Security: **it is your headache**

  Processing images on your servers is a security and stability risk, e.g. PNG decompression bomb.



Do we have to do things this way?

What if we could *just* generate thumbnails on the fly?

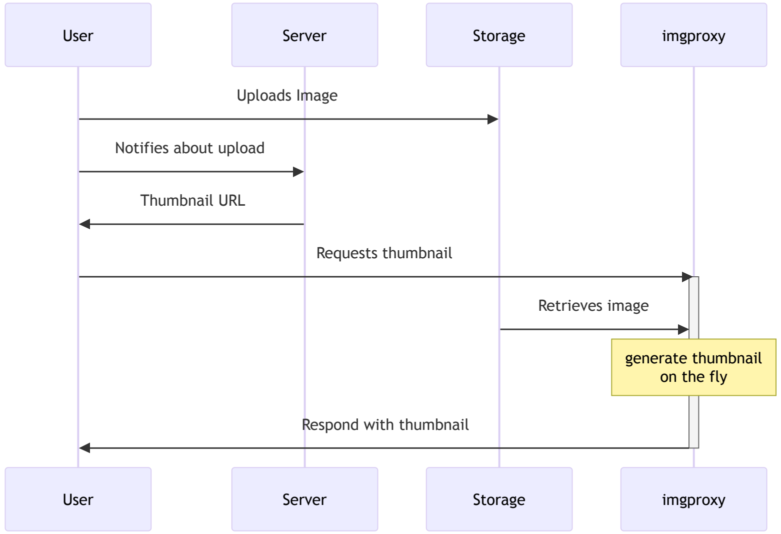


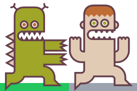
Meet image processing servers

They do just one thing, but do it well

There are many of them:

- imaginary
- thumbor
- cloudinary
- imgix
- imagor
- **imgproxy** (our favorite ✨)





Solving it with on-the-fly processing

- Complexity: **replace your code with a microservice**

  Throw away all these background jobs, and replace them with a simple URL construction.

- Latency: **dedicated service that do only images processing**

  Very performant per se, and you can scale it independently from your main application, also add CDN in front of it

- Adding new variants: **just construct new URL**

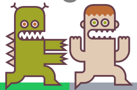
  Construct new URL, request it, done!

- Cleaning up old ones: **let CDN caches to expire**

  Do you really need to store thumbnails at all? Care only for originals.

- Security and stability: **it is separate from your main application**

  It handles image bombs, and other nasty stuff, but even if some malicious code will be executed, it will find itself in empty Docker container without anything in it.



Which one to choose?

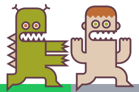
- Should it be one *written* in Ruby?

  But if it is a dedicated service, does it matter?

  Maybe it is better to choose most performant one?

- Should it be one that is **easy to use from Ruby**?

  What are you looking first for when choosing a new dependency?



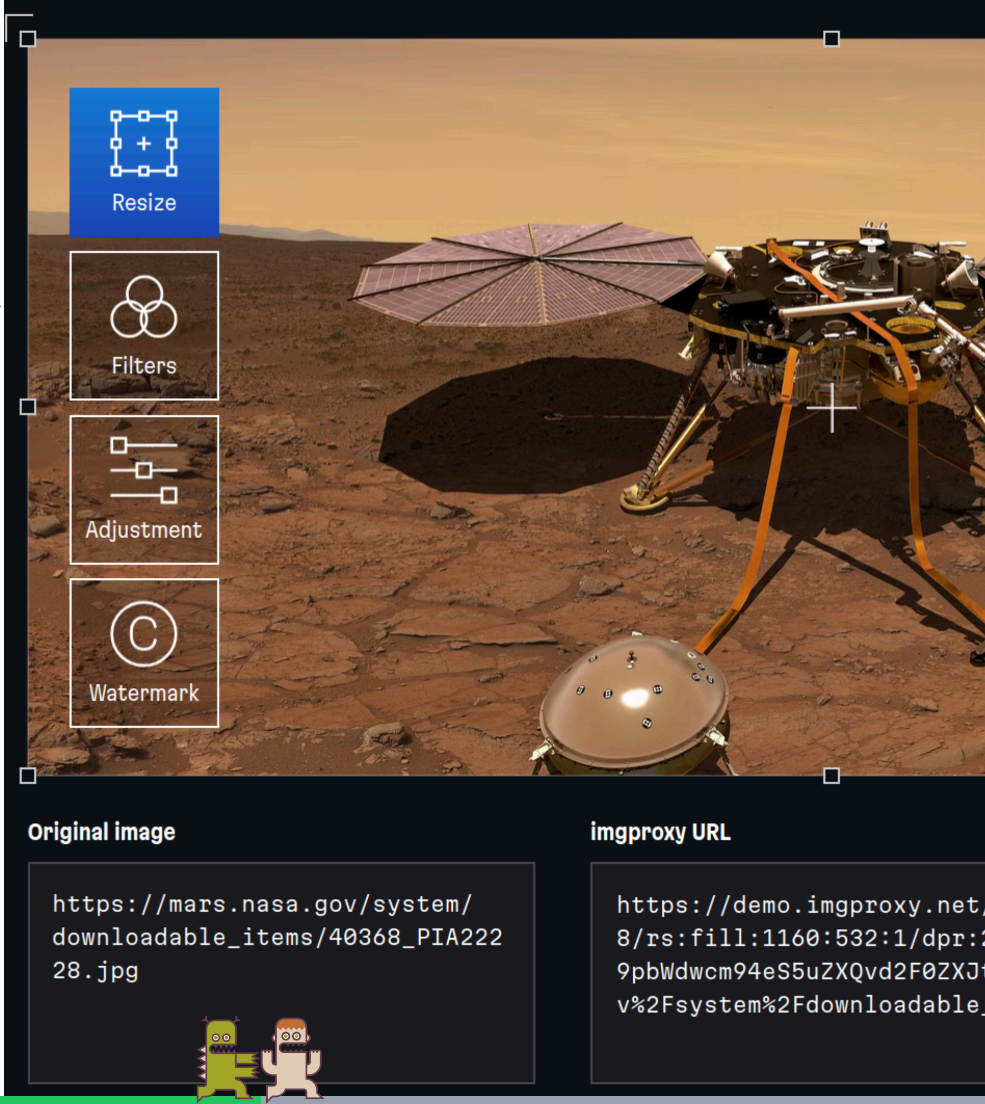
Is there a gem?

of course there is!

Introducing imgproxy

- Open source image processing server
- Written in Go and C for performance
- Uses libvips for optimal image processing
- Dockerized and easy to deploy
- Most Ruby-friendly solution[1]
- Started at Evil Martians
- Used by companies big and small: Bluesky, dev.to, Photobucket, eBay, …

1. There is a gem! Two of them! ↩



Original image

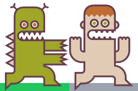
```
https://mars.nasa.gov/system/
downloadable_items/40368_PIA222
28.jpg
```

imgproxy URL

```
https://demo.imgproxy.net
8/rs:fill:1160:532:1/dpr:
9pbWdwcm94eS5uZXQvd2F0ZXJ
v%2Fsystem%2Fdownloadable_
```

But why gem?

What value it brings to both product owners and users?



Technical example: URL signing

The only thing a client need to care about is constructing URLs to images processed through imgproxy.

Given original image URL:

```
https://mars.nasa.gov/system/downloadable_items/40368_PIA22228.jpg
```

Result URL to get 300×150 thumbnail for Retina displays, smart cropped, and saturated, with watermark in right bottom corner:

```
https://demo.imgproxy.net/
doqHNTjtFpozyphRzlQTHyBloSoYS13lLuMDozTnxqA/
rs:fill:300:150:1/dpr:2/g:ce/sa:1.4/
wm:0.5:soea:0:0:0.2/wmu:aHR0cHM6Ly9pbWdwcm94eS5uZXQvd2F0ZXJtYXJrLnN2Zw/
plain/
https:%2F%2Fmars.nasa.gov%2Fsystem%2Fdownloadable_items%2F40368_PIA22228.jpg
```

Digital signature

Processing options

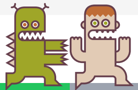
Original image URL

See https://docs.imgproxy.net/generating_the_url

Plain Ruby implementation

It is easy to implement yourself (for one specific use case)

```ruby
require 'base64'
require 'openssl'

key = ['943b421c9eb07c83...'].pack('H*')
salt = ['520f986b998545b4...'].pack('H*')

def generate_url(url, width, height)
  encoded_url = Base64.urlsafe_encode64(url).tr('=', '')
  encoded_url = encoded_url.scan(/.{1,16}/).join('/')

  path = "/resize:fill:#{width}:#{height}/#{encoded_url}"
  hmac = OpenSSL.hmac(
    OpenSSL::Digest.new('sha256'), key, "#{salt}#{path}"
  )
  signature = Base64.urlsafe_encode64(hmac).tr('=', '')

  "http://imgproxy.example.com/#{signature}#{path}"
end

url = generate_url("http://example.com/image.jpg", 300, 400)
```



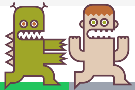
With imgproxy gem

But always better to use a battle-tested library that will hide all gory details

```ruby
require 'imgproxy'

Imgproxy.configure do |config|
  # Full URL to where your imgproxy lives.
  config.endpoint = "http://imgproxy.example.com"
  # Hex-encoded signature key and salt
  config.key = '943b421c9eb07c83...'
  config.salt = '520f986b998545b4...'
end
```

```erb
<%# show.erb.html %>
<%= image_tag Imgproxy.url_for(
  "http://images.example.com/images/image.jpg",
  width: 500,
  height: 400,
  resizing_type: :fill
) %>
```

imgproxy.rb gem



ActiveStorage + imgproxy

What is even better: to use familiar API and don’t change your codebase!

```
# Gemfile
gem 'imgproxy-rails'
```

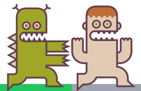
```
# development.rb: use built-in Rails proxy
config.active_storage.resolve_model_to_route = :rails_storage_proxy

# production.rb: use imgproxy
config.active_storage.resolve_model_to_route = :imgproxy_active_storage
```

```
<%# show.erb.html %>
<%= image_tag Current.user.avatar.variant(resize: "100x100") %>
```

You don’t even have to know that you are using imgproxy! ✨

And you can migrate the whole application to imgproxy in an hour!

imgproxy-rails gem



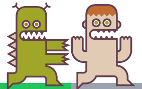
Let the community speak

> I clicked the button, deployed the OSS version and hooked up **the imgproxy.rb ruby gem** in my app in under an hour.
>
> Within a few weeks, we had switched over all of our upload, template, and graphic previews to Imgproxy…
>
> Doing so resulted in the **removal of hundreds of lines of code** while also **enabling new functionality.**

— **John Nunemaker**: Ruby programmer and founder, author of flipper and httparty gems

https://www.johnnunemaker.com/imgproxy/

Imgproxy is Amazing



Why to "keep it Ruby?"

Why to spend time and effort to provide official Ruby SDK?

Answer is in this quote from the previous slide:

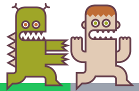
> I clicked the button, deployed the OSS version and hooked up the imgproxy.rb ruby gem in my app **in under an hour**.

It wouldn't be possible without a ready to use Ruby gem!

Keeping your product Ruby-friendly

=

more customers, happier customers



Keep it Ruby! Thank you!

@imgproxy

@imgproxy_net

@imgproxy@mastodon.social

@imgproxy.net

imgproxy.net

@evilmartians

@evilmartians

@evilmartians@mastodon.social

@evilmartians.com

EVIL MARTIANS

evilmartians.com

Our awesome blog: **evilmartians.com/chronicles**!

See these slides at envek.github.io/rubyworld-keep-it-ruby