

stripe

StaffPlus 2024

**So you've outgrown your
monolith**

Goals of the talk

How do you know it's time to
break up your monolith?

How do you advocate for the work
it's going to take?

How do you pull it off?

A bit about me

Me (ains)

- Staff Engineer @ Stripe
- Joined Developer Productivity in 2018
- Tech Lead of Developer Infrastructure since 2021

My Team (Developer Infrastructure)

~250 engineers

- Developer Productivity
 - Build, Test & Tooling
 - Language & Runtime Tools
- Corporate Technology
- Compliance
- Observability

A bit about Stripe

~50

Countries

~7,000

Employees

~1 Trillion

Dollars processed in 2023

A bit the monolith “bapi”

~500 million

API requests /day

~13 million

Lines of Ruby

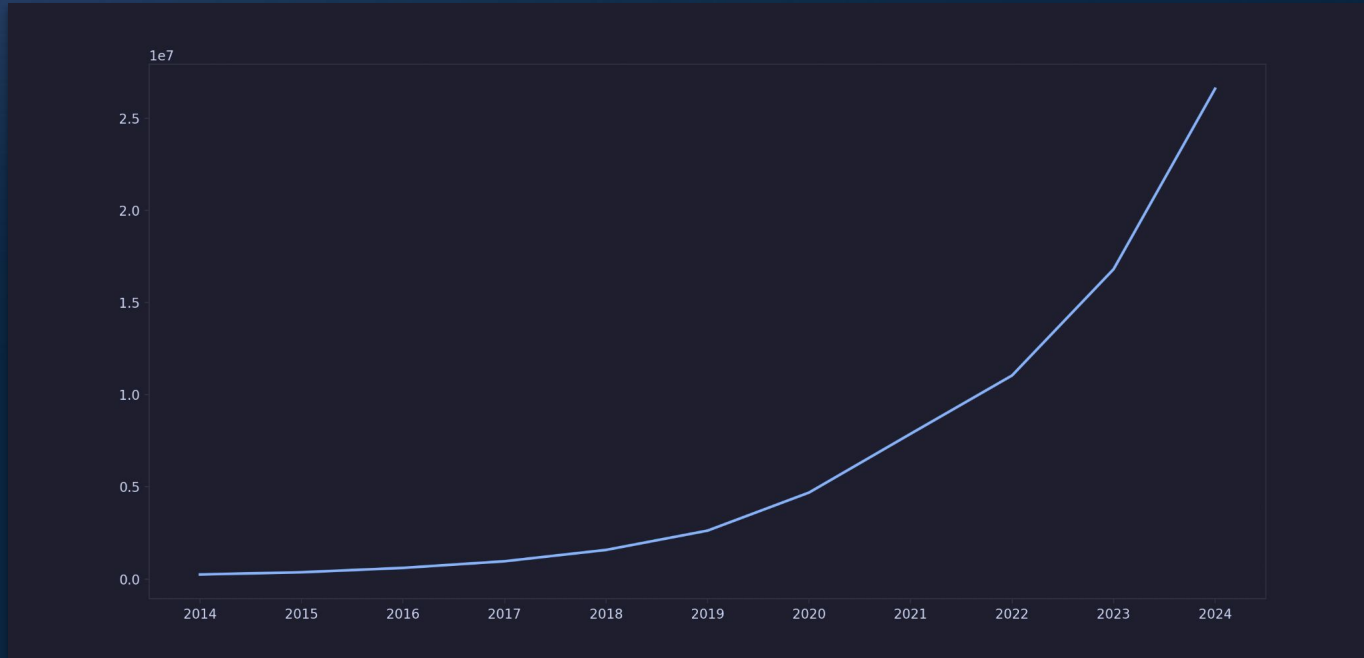
~6000

Deployments in 2022

How do you know it's time to break up your monolith?

How do you know it's time to break up your monolith?

Growth of the monolith (LoC)



How do you know it's time to break up your monolith?

Growth of the monolith (CI CPU Time)



How do you know it's time to break up your monolith?

Growth of the monolith (CI CPU Time)



How do you know it's time to break up your monolith?

Growth of the monolith (CI CPU Time)



How do you know it's time to break up your monolith?

Growth of the monolith (CI CPU Time)



How do you know it's time to break up your monolith?

Growth of the monolith (CI CPU Time)



How do you know it's time to break up your monolith?

The growth of *everything*



How do you know it's time to break up your monolith?

You can't perform heroics forever



How do you know it's time to break up your monolith?

It's not just about exponential growth

Spooky action at a distance

Cognitive overhead

Slower shipping

Ok, so it's time to break up the Monolith...

What's next?

Time to break up the monolith

Concretize your ***technical*** goals

Tackle the “ball of mud”
(Modularity)

Split the monolith into services

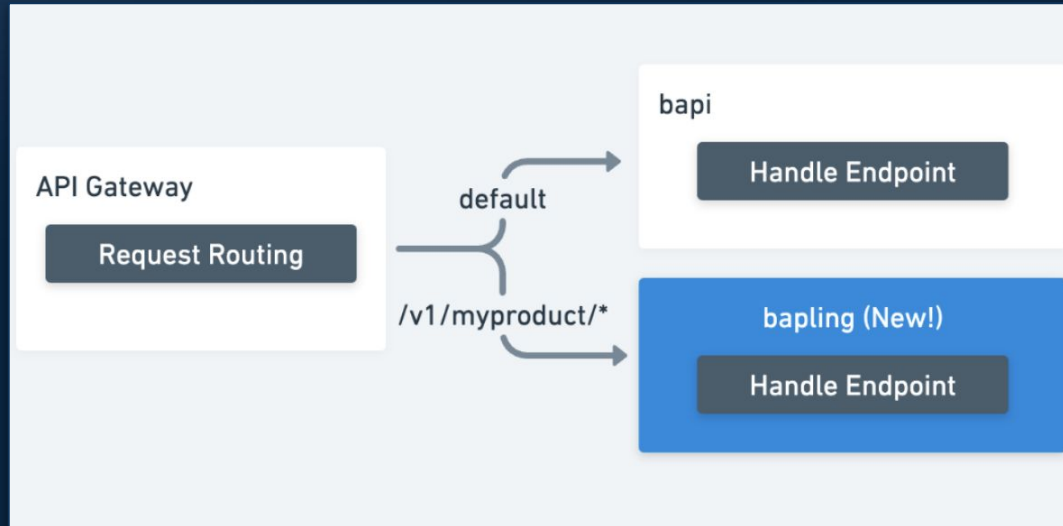
Time to break up the monolith

The “ball of mud”



Time to break up the monolith

New Services



Time to break up the monolith

Concretize your ***business*** goals

Allow engineers to iterate faster

Reduce API running costs

Improve API latency

Time to break up the monolith

Who's going to do the work?

Centralized

Work is predominantly carried out by members of the Developer Infrastructure team and closely related teams (such as Service Platform)

VS

Distributed

Tooling and runbooks are written by Developer Infrastructure. Work is distributed to the teams who own the code.

Making it happen

Making it happen

Project Skywalker



Making it happen

Progressively riskier derisking

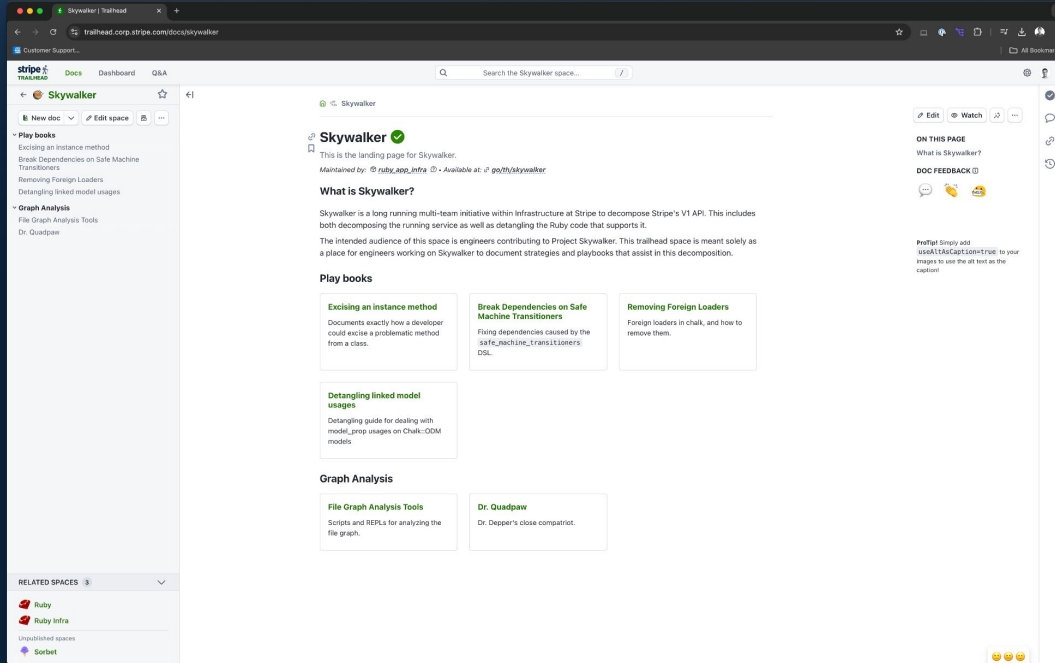
Can we detangle CIBot

Tackle a “scary” core component

Detangle full service alongside
pilot teams

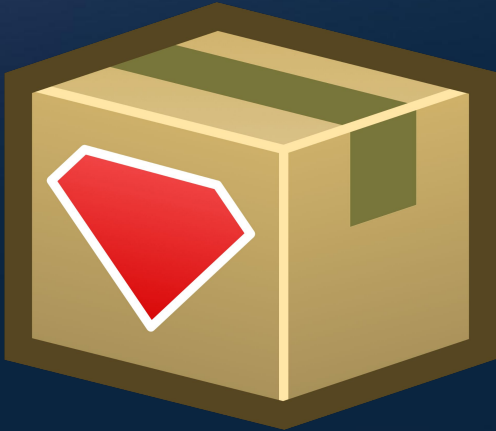
Making it happen

Saving recipes for success



Making it happen

Preventing regressions



What is a Package, anyways?

Detailed explanation and overview of a Ruby package.

Maintained by: @obj_sfta · Available at: @package_layout · <Linked by 7 other docs

Think of a package in pay-server as a small code repository. Similar to any other repo, developers have to be very conscious of the inputs and outputs of their code.

This document is meant to describe what a package is and what invariants pay-server maintains regarding packages.


Package Structure




All Ruby code must belong to a package. Placing a `__package.rb` file into the directory containing this code satisfies this requirement. Code "belongs" to the package immediately adjacent or above in the file-system.


```
1 # frozen_string_literal: true
2 # typed: strict
3
4 # Here be the documentation for my cool package
5 # All documentation is listed as a YARDoc comment on the class itself.
6
7 # Note also that the package spec defines a Ruby constant: Open::MyCoolPackage
8 # All Ruby code in this package must be defined under an Open::MyCoolPackage::Ruby
9 # namespace.
10
11 class Open::MyCoolPackage < PackageSpec
12   # This package is connected to an Open::Project. This provides metrics/ordering
13   # capabilities as well as a basic code ownership declaration.
14   # In this example: Open::Project::ruby_info
15   project :ruby_info
16 end
17
18 # Layering
19 # All packages are grouped into "layers". These layers will be described
20 # below. This dictates what types of other packages are allowed to be
21 # imported.
22 layer :business
23
24 # Strict Dependencies
25 # Strict dependencies describes to what degree restrictions are applied
26 # in the dependencies of this package. This will also be described below.
27 strict_dependencies :big
28
29 # Sortset
30 # Sortset encodes some data into the package spec. This field describes what
31 # is the primary allowed/typical type level for files belonging to this package.
32 sortset_min_type_level :strict, sortset_max_type_level :strict
33
34 # Package Imports
35 # Unlike in other languages, in Stripe's Ruby packages, imports are done at
36 # the level of a package and apply to all code within that package, instead
37 # of being done at the file or individual files in the package.
38 # I.e. Somewhere in Open::MyCoolPackage is Ruby code that accesses the
39 # Database package.
40 imports Database
41
42 import Open::Package::Tokens
43 import Open::Common
44 import Open::Config
```



Keeping up the momentum


Celebrate the small wins


 **reese** 🌸 9:51 AM
With our newly-expanded pokemon set (!), we can now start the countdown for detangling the payment model:

 2  1 




 **reese** 🌸 9:51 AM
set the channel topic: Refunds: 🐉 | Payments: 🦊 | 🍷 highly nontrivial pay-server detangling performed here

 4 

 **bhang** 📅 4:01 PM

 :pm-000:

set the channel topic: Refunds: 📅 | Payments: 🦊 | 🍷 highly nontrivial pay-server detangling performed here

 3  1 

Making it happen

Celebrate the big wins

Skywalker: Detangled V1 API services cheaper, 2x faster p99 latency, use 60% less memory 11 views

Peter Kennedy - pkenn@stripe.com
to Peter Kennedy, Ruby Application Infrastructure Team
Nov 15, 2023, 1:19:18 PM

BLUF: Skywalker optimized Capital's API p99 latency by 2x, improved service start time by 5x, reduced memory usage by 60%, and allowed for Capital engineers to scale down their API infrastructure, saving 20% of their API service costs. This was done by isolating Capital's codebase into a new V1 API service (Bap) and separating it from the ball of mud.

Background

Feedback from StripeSaf has shown that today developers are generally happy working in Ruby. However, feedback from those working in an isolated and detangled codebase has been overwhelmingly positive, as those engineers enjoy faster deploys and simpler development. As part of Skywalker, earlier this year we isolated a prototype of Stripe's first baping, a service supporting a subset of Stripe's V1 API, from Bap, Stripe's V1 API service. This first baping demonstrated a new path to Ruby services starting 2x faster, with 3x better memory utilization and benefits in running tests. Skywalker sought to expand and therefore needed to execute on one of Stripe's products end-to-end and deliver the same value for every developer that supports it. The goal was to show this was not only valuable to Stripe and our customers but reachable in general for product teams.

What we did

Ruby infrastructure and Service Platform, along with our product partners in the Capital org, started this endeavor by first creating a new baping. This new baping allowed Capital engineers to manage their own infrastructure, their own deploys, and have autonomy over how their product runs. However, this service was almost as cumbersome to run as Bap itself due to the amount of Ruby code it required to have loaded in memory.

Skywalker engineers spent considerable time identifying, categorizing, and detangling dependencies from Capital code to the ball of mud, producing a list of every place in the Capital codebase that directly or indirectly relied on the ball of mud. Over the course of several months, we burned this list down by either removing the bad call sites entirely or by diving deeper into the dependency and separating it from the ball of mud.

Assuming all these bad dependencies are either improved or removed, then Capital's baping would be guaranteed to not depend on the ball of mud. Skywalker engineers started with a list of more than 1000 such dependency violations, and burned them down to 0.

Time	Dependency Violations
Start	~700
Midpoint	~500
End	0

Reese Williams
to Reese Williams, Ruby Application Infrastructure Team
Aug 23, 2023, 10:09:53 AM

Background

Our internal API for sending emails used to be a sticking point for modularity efforts in pay-server, primarily due to its usage of subclass lists to lookup all email template implementations. Services that need to send a single email need to depend on every email template implementation at Stripe. Sending emails out-of-band via Monster consumers generally avoids these dependencies, but a few places in the API itself still had dependencies on these subclass lists, especially by way of Risk related code.

We realized that many services never actually use these dependencies and thus loaded thousands of unneeded Ruby files, which required those services to spend more time preloading code and garbage collecting.

What did we do?

As part of [Project Skywalker](#), we completely detached email subclasses from the ball of mud by finding and eliminating [hundreds](#) of dependency edges. The general shape of this work was decoupling models used by the API from dependencies that reach into Risk code. (Risk actions often involve managing and sending emails as part of reviews, despite no part of the Stripe API actually depending on these one-off Risk actions).

Given the breadth of changes involved, this project was split into several smaller workstreams, each working closely with partner teams to refactor foundational parts of pay-server. The following are only a sample of those workstreams:

- [Decoupling Safe Machine Transitions](#) - We built [a new interface for SafeMachine](#) implementations that allowed us to decouple models and their transitions, and we migrated dozens of models to this

The Results

Results

Detangled service wins

83%

Reduction in code loaded

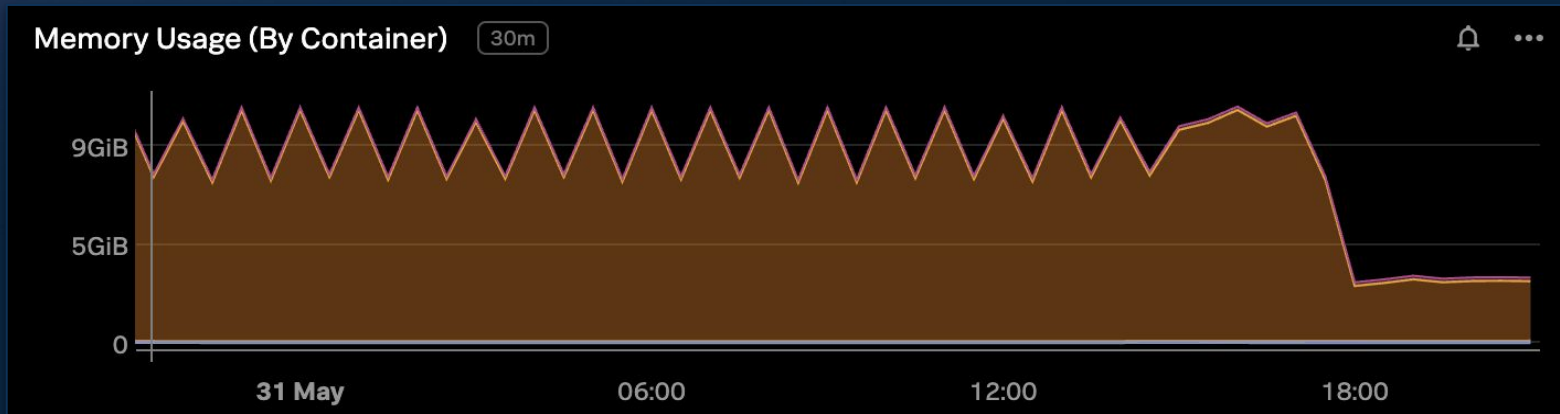
30%

Increase in throughput

3x

Faster devbox reload times

Memory Usage wins



What's next?

Takeaways

Keep an eye on signals to pay
down your tech debt

Start off small, don't bite off more
than you can chew

Celebrate the small wins to keep
up the momentum

stripe

Thanks

