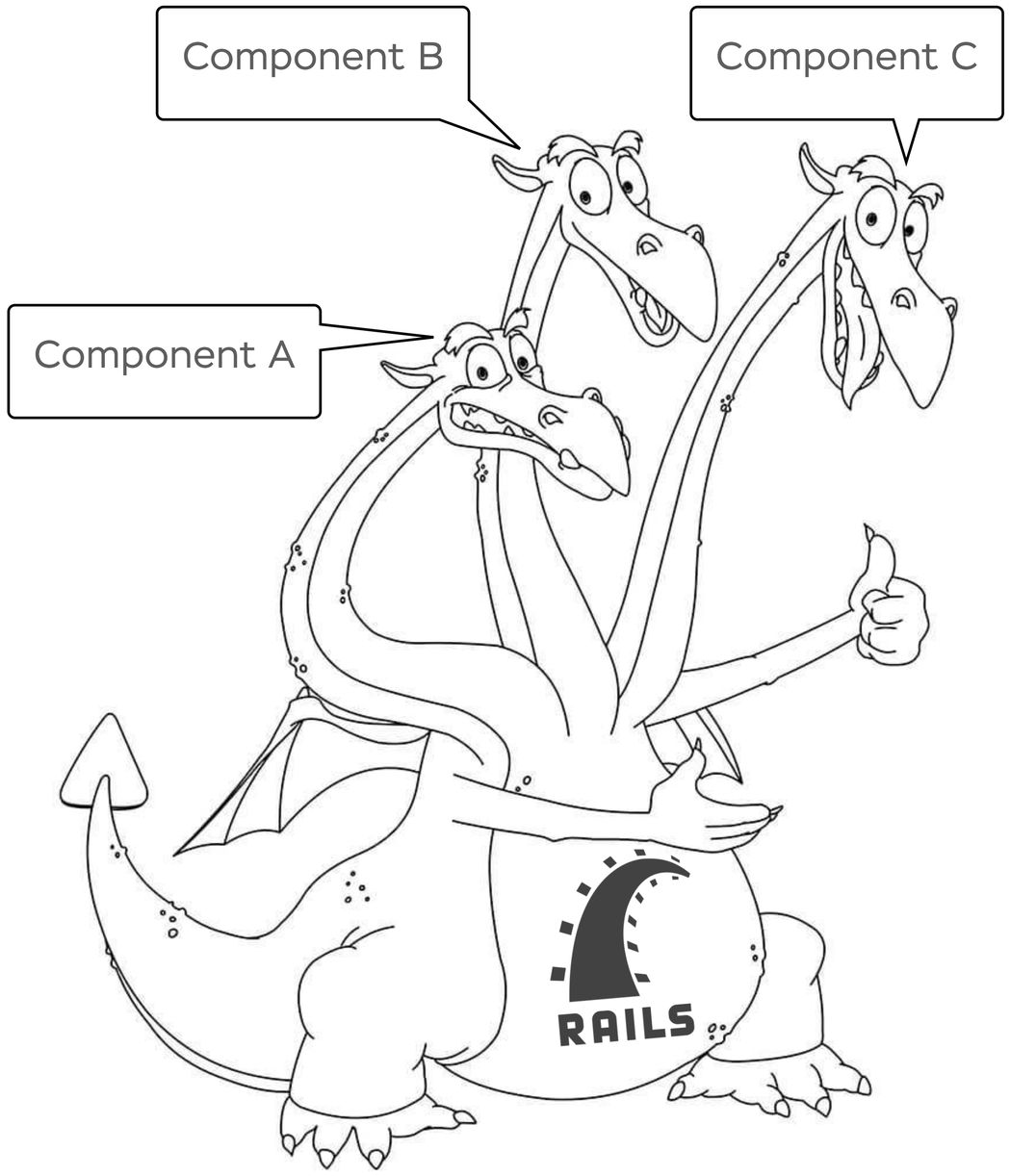
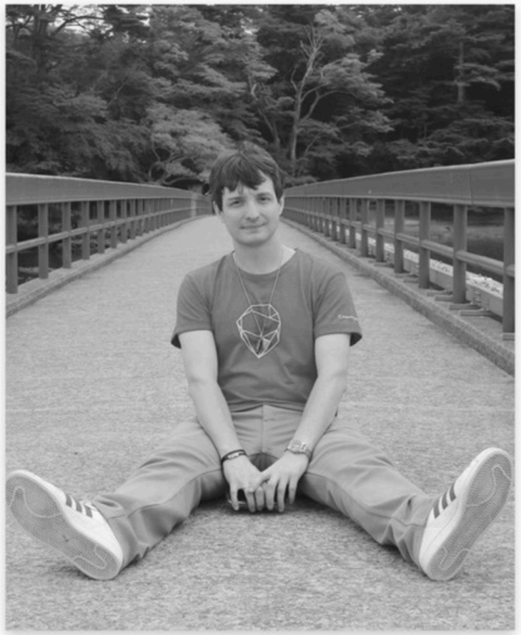
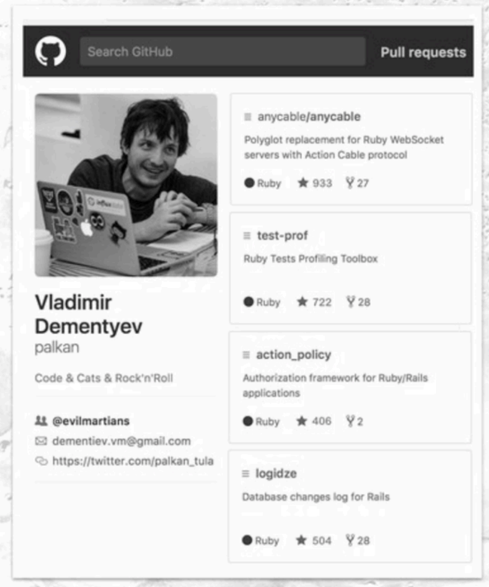



# Engine-ering Rails Applications

Vladimir Dementyev,
Saint-P Ruby 2019



# Vladimir Dementyev




Vladimir Dementyev

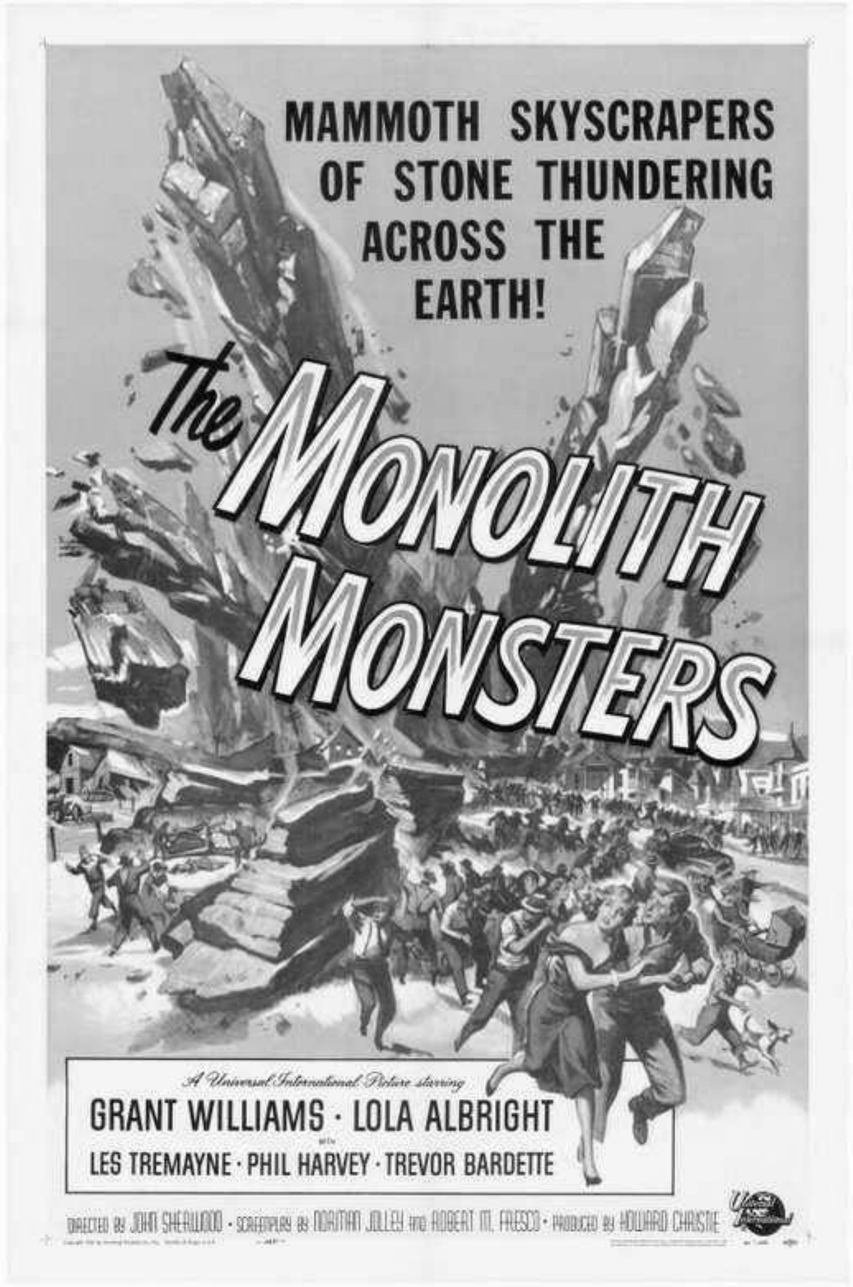
@palkan

DEV @palkan_tula



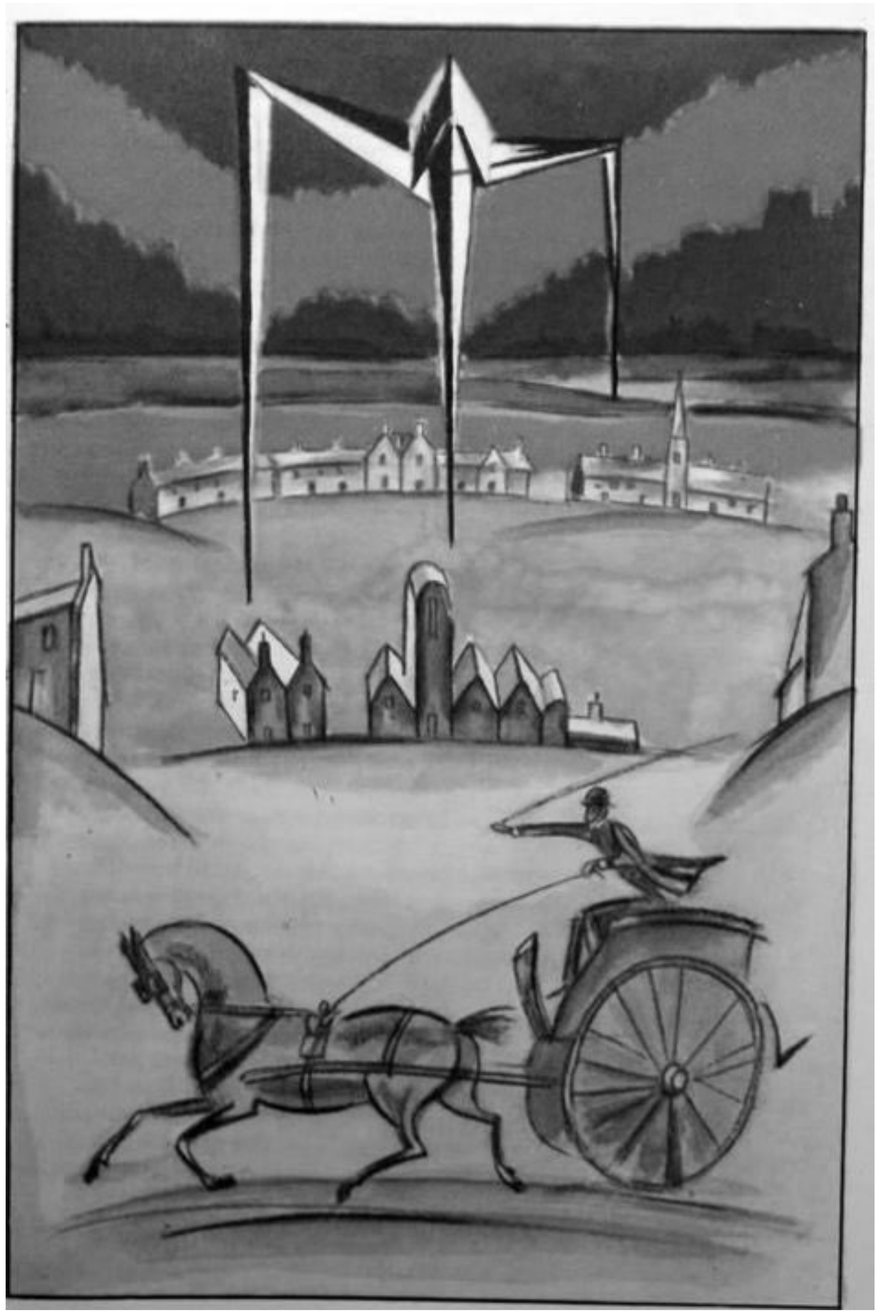
EVIL MARTIANS


TEAM LEAD



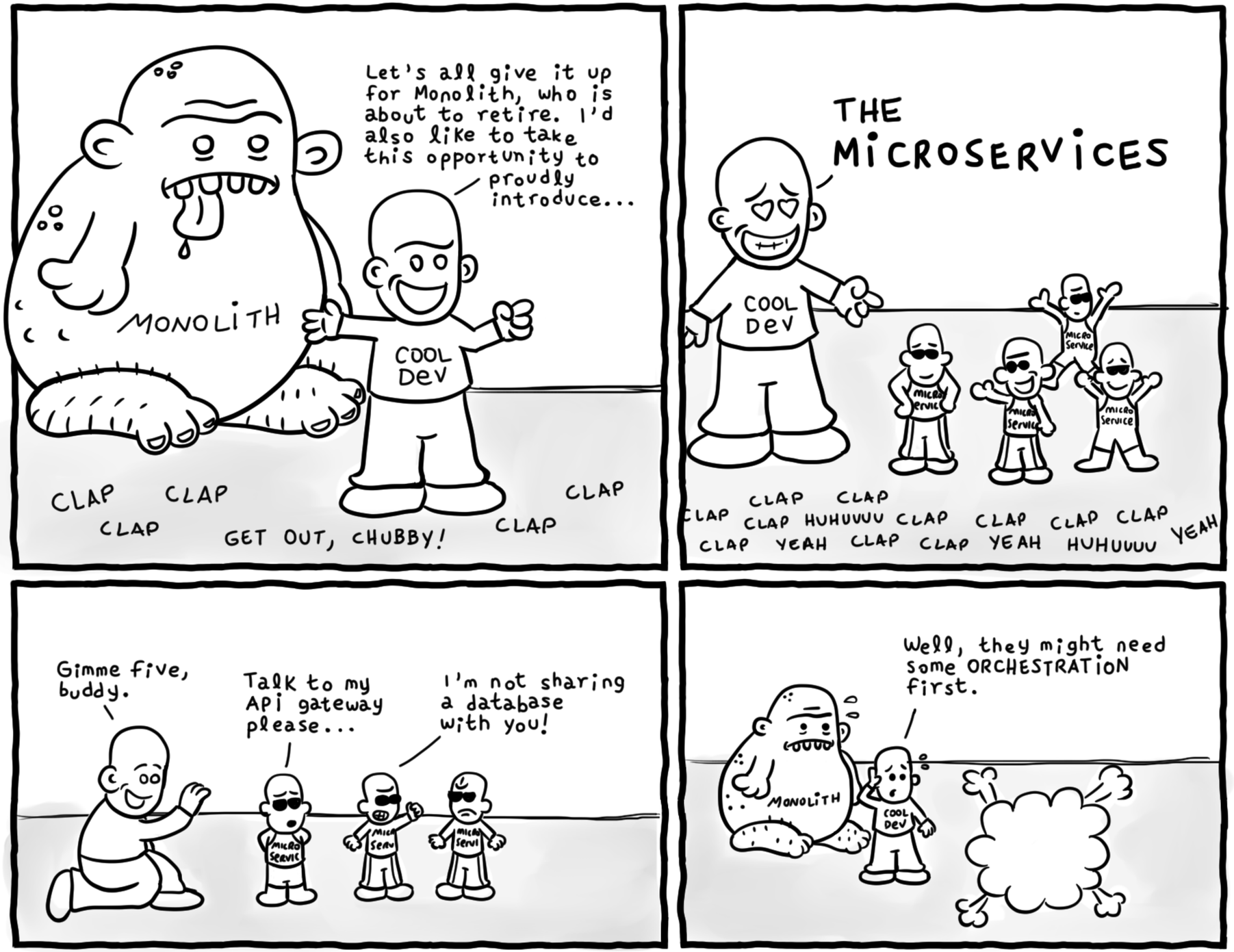

# We* are Doomed

* Rails developers



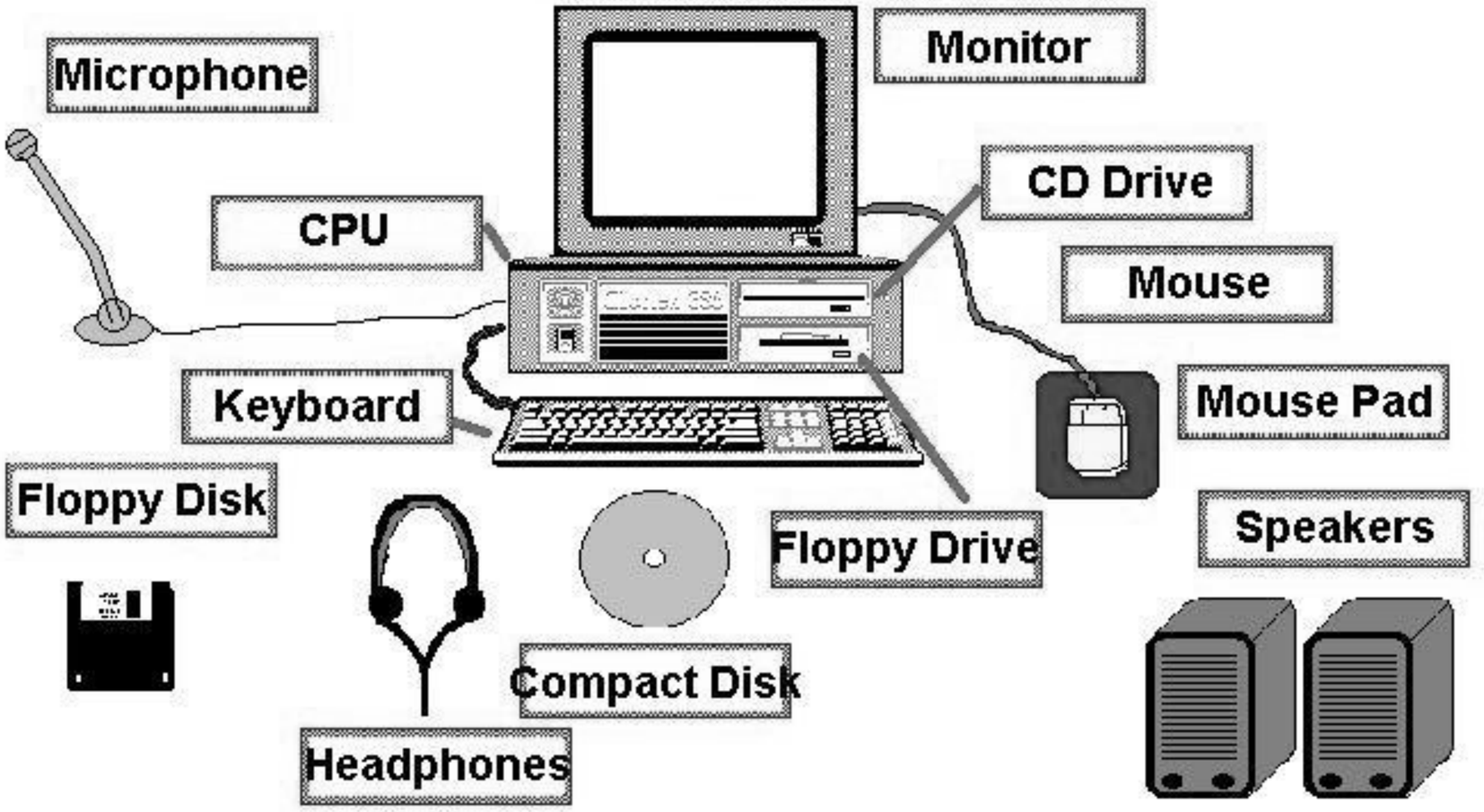
V.Yurlov, for “The war of the worlds” by H.G.Wells

# Can you escape your destiny?




Daniel Stori {turnoff.us}

# Components


Microphone
Monitor
CD Drive
CPU
Mouse
Keyboard
Mouse Pad
Floppy Disk
Floppy Drive
Speakers
Compact Disk
Headphones

# Monolith

😞

# Hanami

## Monolith First

Our default application `Web` can be used as a UI interface for our customers. At a certain point in our story, we want to manage our users with an admin panel.

We know that the set of features that we're going to introduce doesn't belong to our main UI (`Web`). On the other hand, it's **too early** for us to implement a microservices architecture, only for the purpose of helping our users reset their password.

Hanami has a solution for our problem: we can generate a new app that lives in the same Ruby process, but it's a separated component.

# Elixir

## Umbrella projects

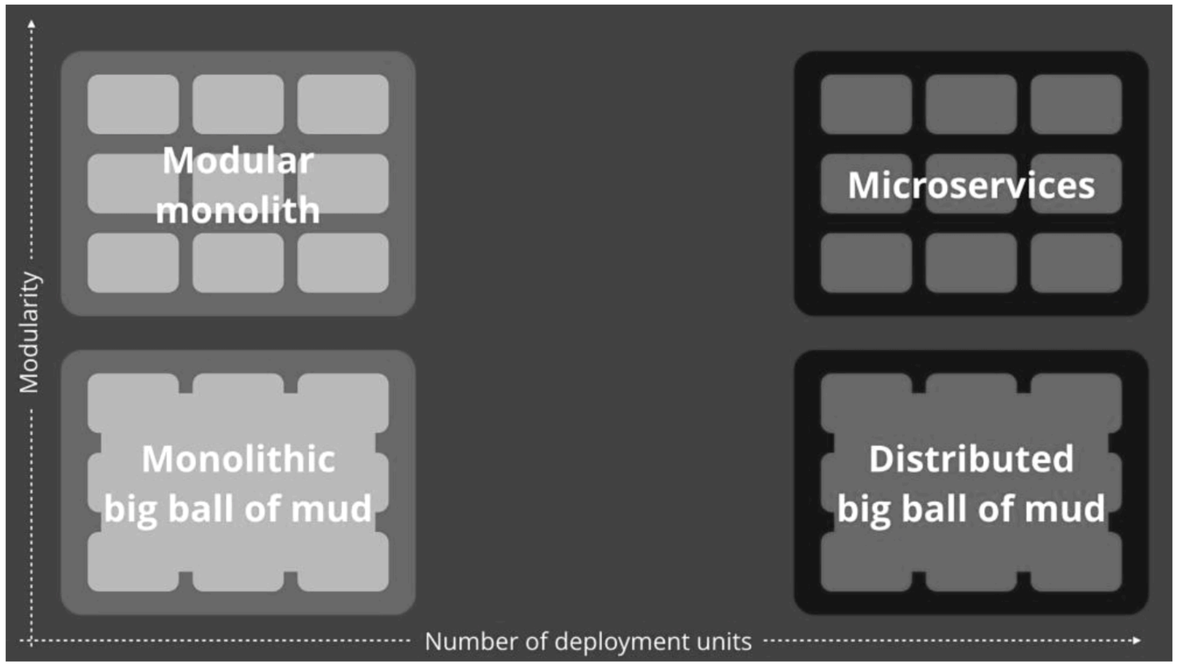
Let's start a new project using `mix new`. This new project will be named `kv_umbrella` and we need to pass the `--umbrella` option when creating it. Do not create this new project inside the existing `kv` project!

```
$ mix new kv_umbrella --umbrella
* creating README.md
* creating .formatter.exs
* creating .gitignore
* creating mix.exs
* creating apps
* creating config
* creating config/config.exs
```

# Shopify: Modular Monolith

## Modular Monoliths

We wanted a solution that increased modularity without increasing the number of deployment units, allowing us to get the advantages of both monoliths and microservices without so many of the downsides.




*Monolith vs Microservices by Simon Brown*


РГДБ
2015
И. КРЫЛОВ
SHOPIFY
И
all others
Художник В. Лихачёв
Производство студии „Диафильм“, 1962 г.

# Rails

RAILS GUIDES

Home | Guides Index | Contribute

## Getting Started with Engines

In this guide you will learn about engines and how they can be used to provide additional functionality to their host applications through a clean and very easy-to-use interface.

After reading this guide, you will know:

- **What makes an engine.**
- **How to generate an engine.**
- **How to build features for the engine.**
- **How to hook the engine into an application.**
- **How to override engine functionality in the application.**
- **Avoid loading Rails frameworks with Load and Configuration Hooks**

### Chapters


1. **What are engines?**
2. **Generating an engine**
   - Inside an Engine
3. **Providing engine functionality**
   - Generating an Article Resource
   - Generating a Comments Resource
4. **Hooking Into an Application**
   - Mounting the Engine
   - Engine setup
   - Using a Class Provided by the Application
   - Configuring an Engine
5. **Testing an engine**
   - Functional Tests


### 1 What are engines?


Addison-Wesley Professional Ruby Series
COMPONENT-BASED RAILS APPLICATIONS
Large Domains Under Control
STEPHAN HAGEMANN

# The Book?

common

# What we've done

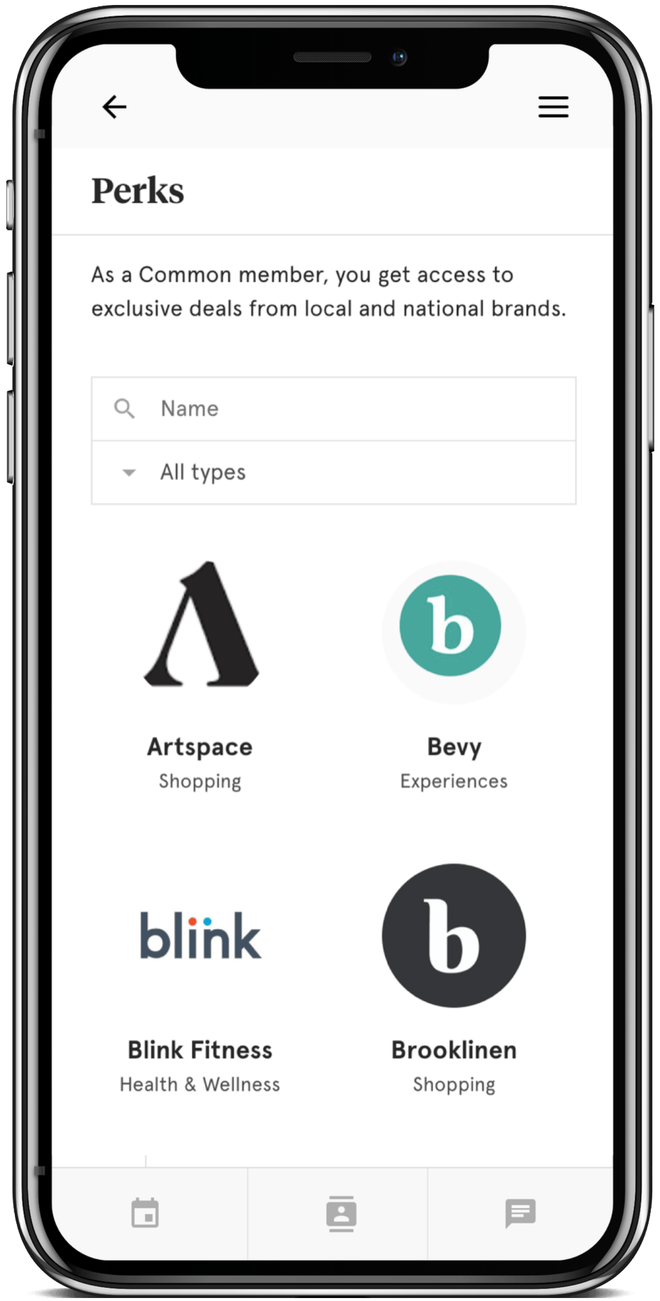
Or how we decided to go with engines

# common

Co-living rentals service

# Our mission

- **They had** a property and lease managements system (admin panel)
- **They needed** a community app for users
- **They wanted** to keep everything in the same Rails app




# Community

- Events
- Perks
- Chat
- Billing

# Phase #1: Namespaces

```
app/
  controllers/
    chat/…
  models/
    chat/…
  jobs/
    chat/…
```

# Namespaces

- Quick start
- Fake isolation

# Our mission evolves

- **There is** a property and lease management application (admin panel)
- **We need** a community app for users
- **We (they) want** to keep everything in the same Rails app
- And **we (again, they) want** to re-use the app's code in the future for a side-project

# Phase #2: Engines & Gems

```
app/…
engines/
  chat/
    app/
      controllers/…
      …
    lib/…
gems/…
```

# The Modular Monolith: Rails Architecture

Dan Manges Follow

Jan 23, 2018 · 12 min read

One of the hardest things about building a startup is handling the rapid growth in team and technology. The best way to build software with a team of three engineers is different than with ten engineers, or twenty, or fifty. Make a change to your process today, and you're doing it too soon. Wait until tomorrow, and it feels too late.

```
rails plugin new \
my_engine --mountable
```

# Engems

Building Rails apps from engines and gems

# Crash Course in Engines

```
$ rails plugin new my_engine \
--mountable  # or —full

      create
      create  README.md
      create  Rakefile
      create  Gemfile
      create  my_engine.gemspec
```

Engine is a gem

# Crash Course in Engines

```
my_engine/
  app/
    controllers/…
    …
  config/routes.rb
  lib/
      my_engine/engine.rb
      my_engine.rb
```

Added to paths

# Crash Course in Engines

```ruby
# my_engine/lib/my_engine/engine.rb
module MyEngine
  class Engine < ::Rails::Engine
    # only in mountable engines
    isolate_namespace MyEngine
  end
end
```

# Crash Course in Engines

```
# my_engine/config/routes.rb
MyEngine::Engine.routes.draw do
  get “/best_ruby_conference”,
      to: redirect("https://spbrubyconf.ru")
end
```

# Crash Course in Engines

```
# <root>/Gemfile
gem "my_engine", path: "my_engine"
```

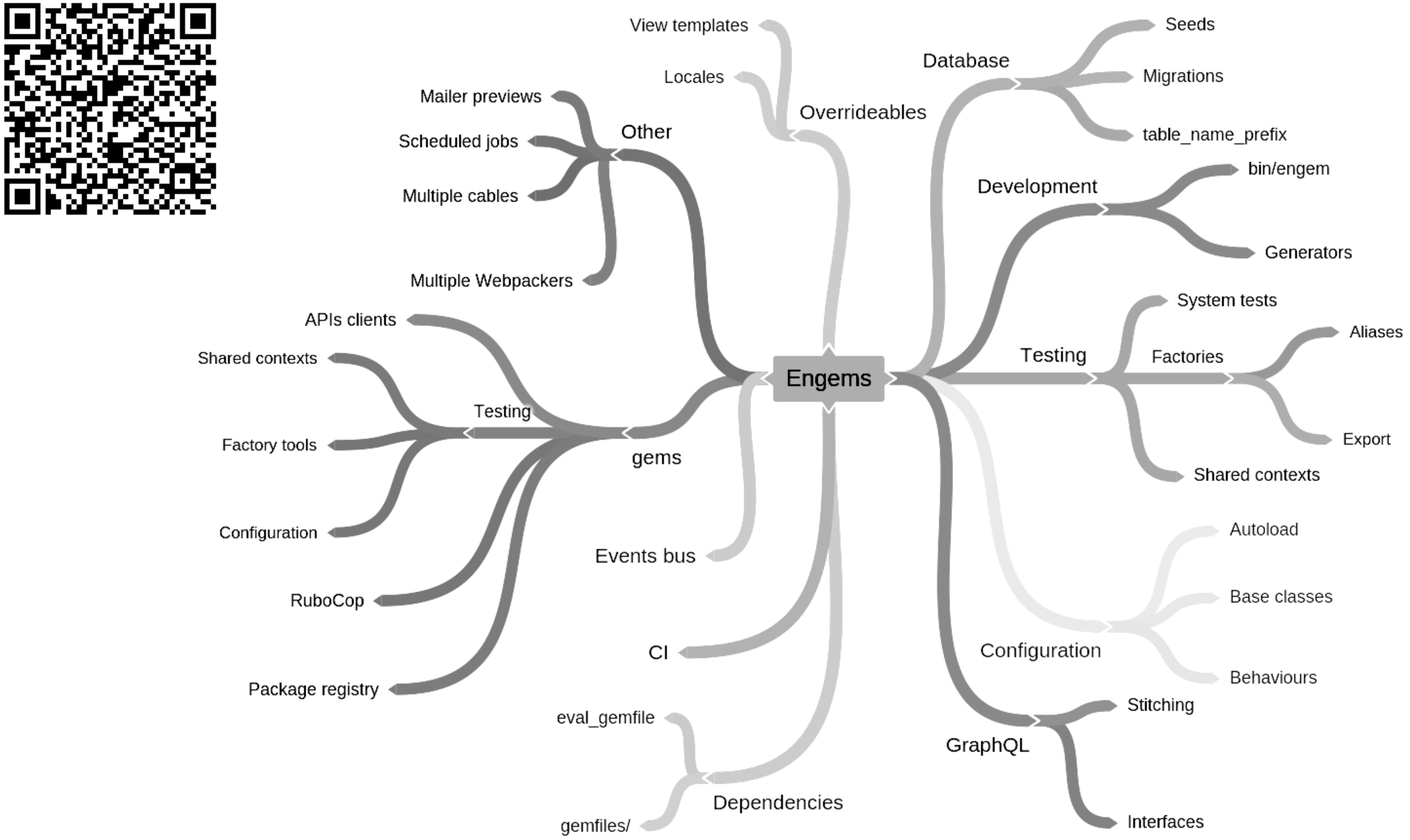
# Crash Course in Engines

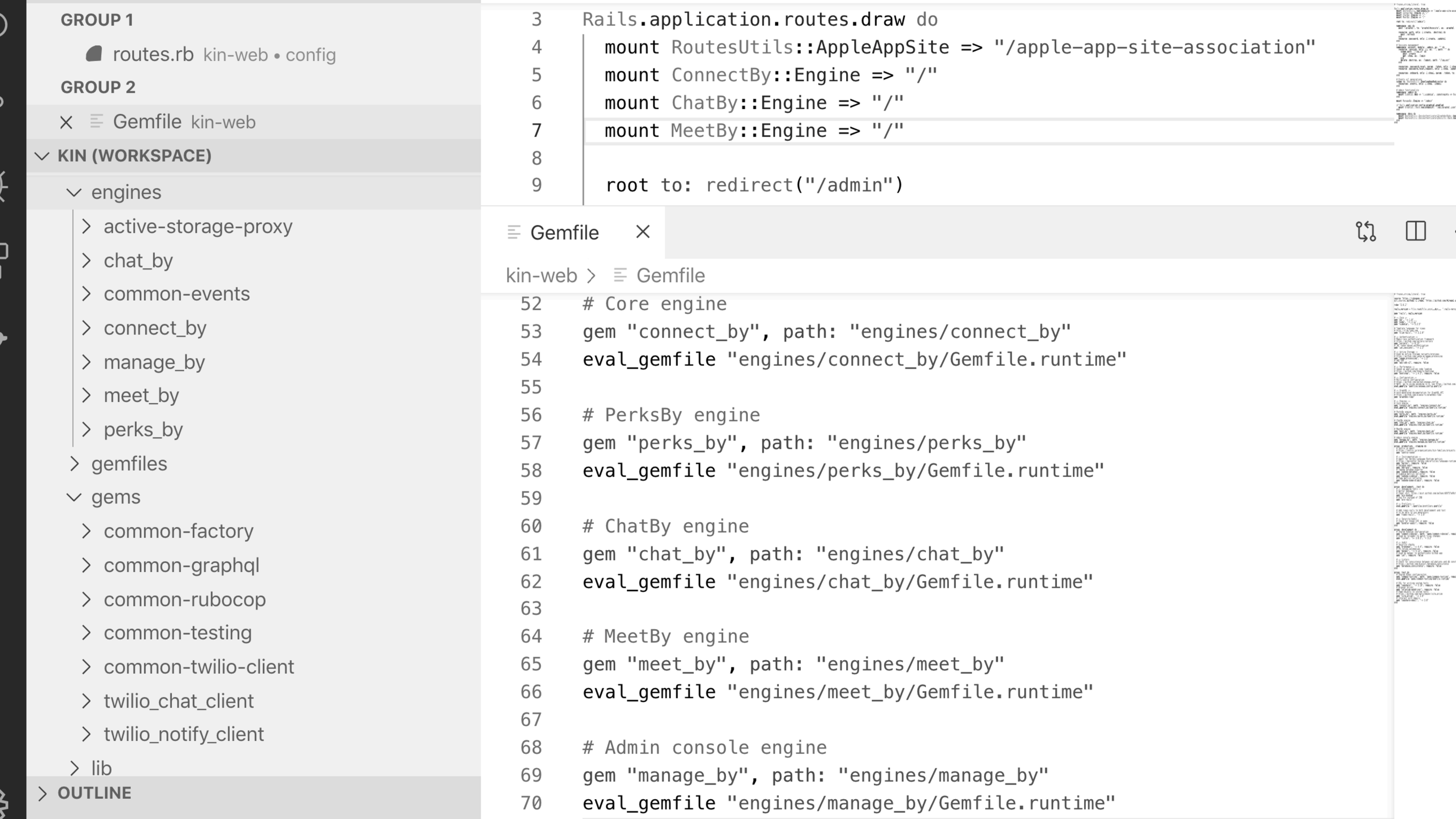
```ruby
# <root>/config/routes.rb
Rails.application.routes.draw do
  mount MyEngine::Engine ⇒ "/my_engine"
end
```

# Crash Course in Engines

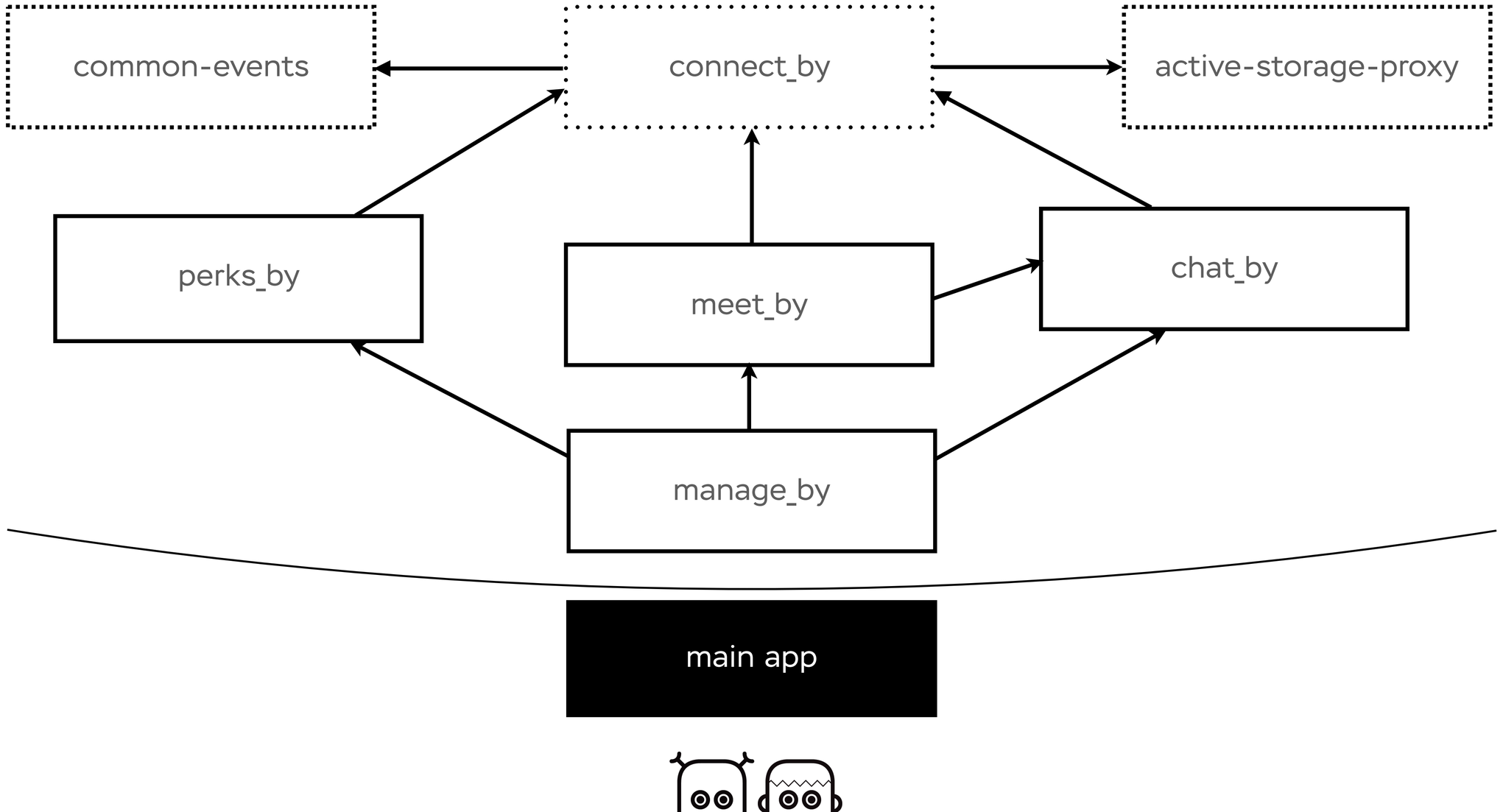
```
$ rake routes

Prefix Verb   URI Pattern     Controller#Action
…
  my_engine   /my_engine      MyEngine::Engine

Routes for MyEngine::Engine:
best_ruby_conference GET /best_ruby_conference(:format)
```

# The end


Engems
Overrideables
View templates
Locales
Other
Mailer previews
Scheduled jobs
Multiple cables
Multiple Webpackers
gems
Testing
APIs clients
Shared contexts
Factory tools
Configuration
RuboCop
Package registry
Events bus
CI
Dependencies
eval_gemfile
gemfiles/
Database
Seeds
Migrations
table_name_prefix
Development
bin/engem
Generators
Testing
System tests
Factories
Aliases
Export
Shared contexts
Configuration
Autoload
Base classes
Behaviours
GraphQL
Stitching
Interfaces


GROUP 1
routes.rb kin-web • config
GROUP 2
Gemfile kin-web
KIN (WORKSPACE)
engines
active-storage-proxy
chat_by
common-events
connect_by
manage_by
meet_by
perks_by
gemfiles
gems
common-factory
common-graphql
common-rubocop
common-testing
common-twilio-client
twilio_chat_client
twilio_notify_client
lib
OUTLINE
3 Rails.application.routes.draw do
4 mount RoutesUtils::AppleAppSite => "/apple-app-site-association"
5 mount ConnectBy::Engine => "/"
6 mount ChatBy::Engine => "/"
7 mount MeetBy::Engine => "/"
8
9 root to: redirect("/admin")
Gemfile
kin-web > Gemfile
52 # Core engine
53 gem "connect_by", path: "engines/connect_by"
54 eval_gemfile "engines/connect_by/Gemfile.runtime"
55
56 # PerksBy engine
57 gem "perks_by", path: "engines/perks_by"
58 eval_gemfile "engines/perks_by/Gemfile.runtime"
59
60 # ChatBy engine
61 gem "chat_by", path: "engines/chat_by"
62 eval_gemfile "engines/chat_by/Gemfile.runtime"
63
64 # MeetBy engine
65 gem "meet_by", path: "engines/meet_by"
66 eval_gemfile "engines/meet_by/Gemfile.runtime"
67
68 # Admin console engine
69 gem "manage_by", path: "engines/manage_by"
70 eval_gemfile "engines/manage_by/Gemfile.runtime"

# Common Engines



# Dependencies

- How to use non-Rubygems deps?
- How to share common deps?
- How to sync versions?

# Path/Git problem

```
# engines/my_engine/Gemfile
gem "local-lib", path: "../local-lib"
gem "git-lib", github: "palkan/git-lib"

# <root>/Gemfile
gem "my_engine", path: "engines/my_engine"
gem "local-lib", path: "../local-lib"
gem "git-lib", github: "palkan/git-lib"
```

# eval_gemfile

```
# engines/my_engine/Gemfile
eval_gemfile "./Gemfile.runtime"

# engines/my_engine/Gemfile.runtime
gem "local-lib", path: "../local-lib"
gem "git-lib", github: "palkan/git-lib"

# <root>/Gemfile
gem "my_engine", path: "engines/my_engine"
eval_gemfile "engines/my_engine/Gemfile.runtime"
```

# eval_gemfile

## Gemfiles and gemspecs

We define gems/engines dependencies and requirements using multiple files:

- `<gem>/<gem>.gemspec` – this is Gem specification; it defines **requirements** for this gem (for runtime and development); all required libraries (even *local*) must be specified in the gemspec: we use this information when checking *dirtyness* of gems on CI (see `/.circleci/is-dirty`)
- `<gem>/Gemfile.runtime` – defines where to find non-RubyGems **runtime** dependencies (local and GitHub); this file is used by other gems via the `eval_gemfile` method.
- `<gem>/Gemfile` – defines where to find non-RubyGems **development** dependencies (local and GitHub); always *includes* (via `eval_gemfile`) the `Gemfile.runtime` file.

# Shared Gemfiles

```
gemfiles/
  profilers.gemfile
  rails.gemfile
```

```
gem "stackprof", "0.2.12"
gem "ruby-prof", "0.17.0"
gem "memory_profiler", "0.9.12"
```

```
gem "rails", “6.0.0.rc1”
```

# Shared Gemfiles

```
# engines/my_engine/Gemfile
eval_gemfile "../gemfiles/rails.runtime"
eval_gemfile "../gemfiles/profilers.runtime"

# <root>/Gemfile
eval_gemfile "gemfiles/rails.runtime"
eval_gemfile "gemfiles/profilers.runtime"
```

# Keep Versions N'Sync

```
gem 'transdeps'
```

*A gem to find inconsistent dependency versions in component-based Ruby apps.*


# DB vs. Engines

- How to manage migrations?
- How to write seeds?
- How to namespace tables?

# Migrations. Option #1

Install migrations:

```
rails my_engine:install:migrations
```

# Migrations. Option #2

“Mount“ migrations:

```ruby
# engines/my_engine/lib/my_engine/engine.rb
initializer "my_engine.migrations" do |app|
  app.config.paths["db/migrate"].concat(
    config.paths["db/migrate"].expanded
  )
  # For checking pending migrations
  ActiveRecord::Migrator.migrations_paths +=
    config.paths["db/migrate"].expanded.flatten
end
```

# Seeds

```
# <root>/db/seed.rb
ActiveRecord::Base.transaction do
  ConnectBy::Engine.load_seed
  PerksBy::Engine.load_seed
  ChatBy::Engine.load_seed
  MeetBy::Engine.load_seed
end
```

# table_name_prefix

```
class CreateConnectByInterestTags < ActiveRecord::Migration
  include ConnectBy::MigrationTablePrefix

  def change
    create_table :interest_tags do |t|
      t.string :name, null: false
    end
  end
end
```

# table_name_prefix

```
module ConnectBy
  module MigrationTablePrefix
    def table_prefix
      ConnectBy.table_name_prefix
    end

    def table_name_options(config = ActiveRecord::Base)
      {
        table_name_prefix: "#{table_prefix}#{config.table_name_prefix}",
        table_name_suffix: config.table_name_suffix
      }
    end
  end
end
```

# table_name_prefix

```
# config/application.rb
config.connect_by.table_name_prefix = "connect_"
```

# Factories

```
# engines/my_engine/lib/my_engine/engine.rb
initializer "my_engine.factories" do |app|
  factories_path = root.join("spec", "factories")

  ActiveSupport.on_load(:factory_bot) do
    require "connect_by/ext/factory_bot_dsl"
    FactoryBot.definition_file_paths.unshift factories_path
  end
end
```

Custom load hook

# Factories: Aliasing

```
using ConnectBy::FactoryBotDSL
```

Takes engine namespace into account

```
FactoryBot.define do
  # Uses ConnectBy.factory_name_prefix + "city" as the name
  factory :city do
    sequence(:name) { |n| Faker::Address.city + " (#{n})"}

    trait :private do
      visibility { :privately_visible }
    end
  end
end
```

# Factories: Aliasing

```ruby
# spec/support/factory_aliases.rb
unless ConnectBy.factory_name_prefix.empty?
  RSpec.configure do |config|
    config.before(:suite) do
      FactoryBot.factories.map do |factory|
        next unless factory.name.to_s.starts_with?(ConnectBy.factory_name_prefix)

        FactoryBot.factories.register(
          factory.name.to_s.sub(/^#{ConnectBy.factory_name_prefix}/, "").to_sym,
          factory
        )
      end
    end
  end
end
```

# Factories: Aliasing

- Use short (local) name when testing the engine
- Use long (namespaced) when using in other engines

# Shared Contexts

```
my_engine/
  lib/
    my_engine/
      testing/
        shared_contexts/…
        shared_examples/…
      testing.rb
```

# Shared Contexts

```
# other_engine/spec/rails_helper/rb

require "connect_by/testing/shared_contexts"
require "connect_by/testing/shared_examples"
```

# How to test engines?

# Testing. Option #1

Using a full-featured Dummy app:

```
spec/
  dummy/
    app/
      controllers/
      …
    config/
    db/
    test/
    …
```

# Testing. Option #2

```
gem 'combustion'
```

*A library to help you test your Rails Engines in a simple and effective manner, instead of creating a full Rails application in your spec or test folder.*

# Combustion

```ruby
begin
  Combustion.initialize! :active_record, :active_job do
    config.logger = Logger.new(nil)
    config.log_level = :fatal
    config.autoloader = :zeitwerk

    config.active_storage.service = :test
    config.active_job.queue_adapter = :test
  end
rescue ⇒ e
  # Fail fast if application couldn't be loaded
  $stdout.puts "Failed to load the app: #{e.message}\n" \
               "#{e.backtrace.take(5).join("\n")}"
  exit(1)
end
```

# Combustion

```
spec/
  internal/
    config/
      storage.yml
    db/
      schema.rb
```

# Combustion

- Load only required Rails frameworks
- No boilerplate, only the files you need
- Automatically re-runs migrations for every test run

# CI

```
commands:
  engem:
    description: Run engine/gem build
    parameters:
      target:
        type: string
    steps:
      - run:
          name: "[<< parameters.target >>] bundle install"
          command: |
            .circleci/is-dirty << parameters.target >> || \
            bundle exec bin/engem << parameters.target >> build
```

Skip if no relevant changes

# .circleci/is-dirty

```
pattern = File.join(__dir__, "../{engines,gems}/*/*.gemspec")

gemspecs = Dir.glob(pattern).map do |gemspec_file|
  Gem::Specification.load(gemspec_file)
end

names = gemspecs.each_with_object({}) do |gemspec, hash|
  hash[gemspec.name] = []
end

# see next slide
```

# .circleci/is-dirty

```
  tree = gemspecs.each_with_object(names) do |gemspec, hash|
    deps = Set.new(gemspec.dependencies.map(&:name)) +
Set.new(gemspec.development_dependencies.map(&:name))
    local_deps = deps & Set.new(names.keys)
    local_deps.each do |local_dep|
      hash[local_dep] << gemspec.name
    end
  end

  # invert tree to show which gem depends on what
  tree.each_with_object(Hash.new { |h, k| h[k] = Set.new }) do |(name, deps), index|
    deps.each { |dep| index[dep] << name }
    index
  end
```

# .circleci/is-dirty

```
def dirty_libraries
  changed_files = `git diff $(git merge-base origin/master HEAD) --name-only`
    .split("\n")
  raise "failed to get changed files" unless $?.success?

  changed_files.each_with_object(Set.new) do |file, changeset|
    if file =~ %r{^(engines|gems)/([^\/]+)}
      changeset << Regexp.last_match[2]
    end
    changeset
  end
end
```

# CI

```
engines:
  executor: rails
  steps:
    - attach_workspace:
        at: .
    - engem:
        target: connect_by
    - engem:
        target: perks_by
    - engem:
        target: chat_by
    - engem:
        target: manage_by
    - engem:
        target: meet_by
```

# Dev Tools

- `rails plugins new` is too simple
- Use generators: `rails g engine`

# Generators

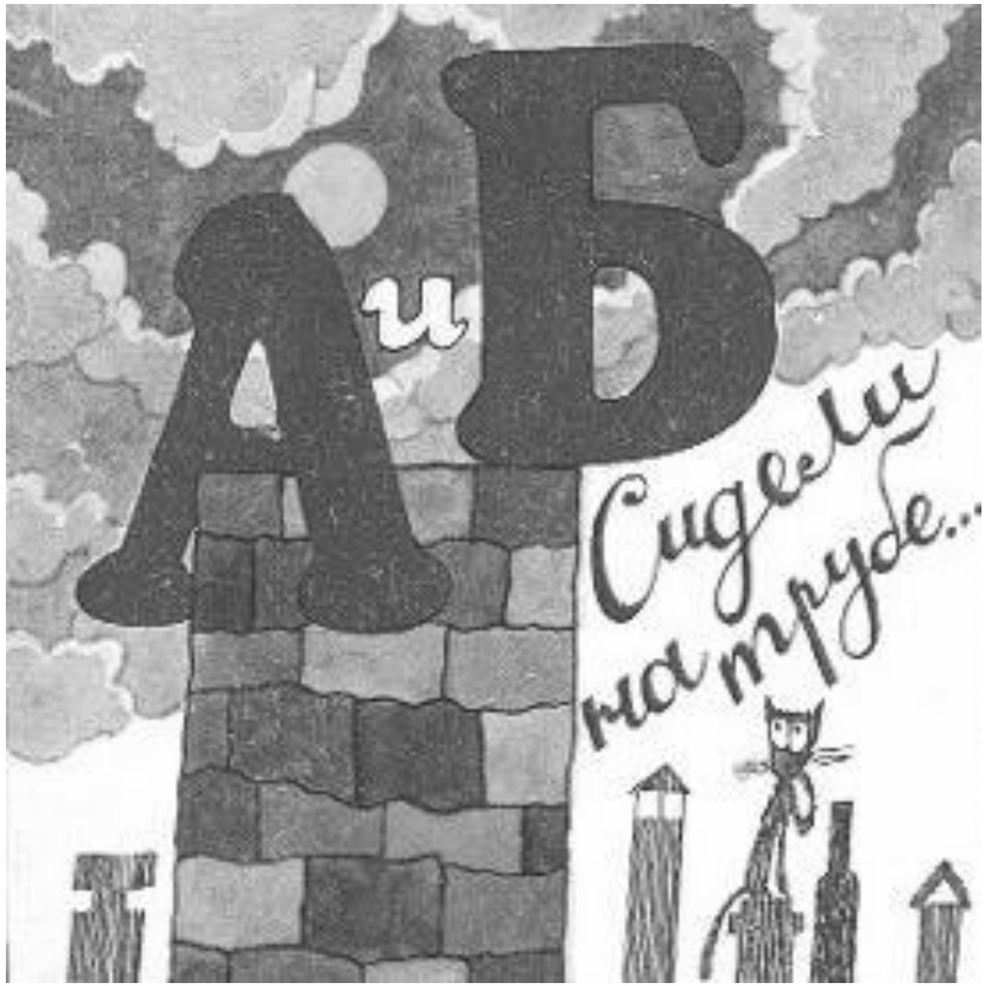
```ruby
# lib/generators/engine/engine_generator.rb
class EngineGenerator < Rails::Generators::NamedBase
  source_root File.expand_path("templates", __dir__)

  def create_engine
    directory(".", "engines/#{name}")
  end
end
```

# Dev Tools

- `rails plugins new` is too simple
- Use generators: `rails g engine`
- Manage engines: `bin/engem`

# bin/engem

```
# run a specific test
$ ./bin/engem connect_by rspec spec/models/connect_by/city.rb:5

# runs `bundle install`, `rubocop` and `rspec` by default
$ ./bin/engem connect_by build

# generate a migration
$ ./bin/engem connect_by rails g migration <name>

# you can run command for all engines/gems at once by using "all" name
$ ./bin/engem all build
```




Engine

↕

Engine

↕

Main app

# How to modify other engine’s entities?

# Base & Behaviour

- Base Rails classes within an engine MUST be configurable
- They also MAY require to have a certain “interface”

# Base & Behaviour

```
module ConnectBy
  class ApplicationController < Engine.config.application_controller.constantize
    raise "Must include ConnectBy::ControllerBehaviour" unless
      self < ConnectBy::ControllerBehaviour

    raise "Must implement #current_user method" unless
      instance_methods.include?(:current_user)
  end
end
```

# Base & Behaviour

```ruby
module ConnectBy
  class ApplicationController < Engine.config.application_controller.constantize
    raise "Must include ConnectBy::ControllerBehaviour" unless
      self < ConnectBy::ControllerBehaviour

    raise "Must implement #current_user method" unless
      instance_methods.include?(:current_user)
  end
end
```

Configurable

# Base & Behaviour

```
# config/application.rb
config.connect_by.application_controller =
  "MyApplicationController"
```

# Base & Behaviour

```ruby
module ConnectBy
  class ApplicationController < Engine.config.application_controller.constantize
    raise "Must include ConnectBy::ControllerBehaviour" unless
      self < ConnectBy::ControllerBehaviour

    raise "Must implement #current_user method" unless
      instance_methods.include?(:current_user)
  end
end
```

“Interface”

# Base & Behaviour

```
class ApplicationController < ActionController::API
  include ConnectBy::ControllerBehaviour
  include JWTAuth
  include RavenContext if defined?(Raven)
end
```

# Modify Models

- Prefer **extensions** over **patches**
- Use **load hooks** (to support autoload/reload)

# Load Hooks

```ruby
# engines/connect_by/app/models/connect_by/city.rb
module ConnectBy
  class City < ActiveRecord::Base
    # ...
    ActiveSupport.run_load_hooks(
      "connect_by/city",
      self
    )
  end
end
```

# Load Hooks

```ruby
# engines/meet_by/lib/meet_by/engine.rb
initializer "meet_by.extensions" do
  ActiveSupport.on_load("connect_by/city") do
    include ::MeetBy::Ext::ConnectBy::City
  end
end
```

# Load Hooks

```ruby
# engines/meet_by/app/models/ext/connect_by/city.rb
module MeetBy
  module Ext
    module ConnectBy
      module City
        extend ActiveSupport::Concern

        included do
          has_many :events, class_name: "MeetBy::Events::Member",
                            inverse_of: :user,
                            foreign_key: :user_id,
                            dependent: :destroy
        end
      end
    end
  end
end
```

# How to communicate between engines?

# Problem

- Some “events” trigger actions in multiple engines
- E.g., user registration triggers chat membership initialization, manager notifications, etc.
- But user registration “lives” in `connect_by` and have no idea about chats, managers, whatever 🤔

# Solution

- Events
- Events
- Events

# Tools

- Hanami Events
- Rails Events Store
- Wisper (?)
- dry-events (?)

# RAILS EVENT STORE

The open-source implementation of an Event Store for Ruby and Rails

Get Started  Get Support

Rails Event Store is a library for publishing, consuming, storing and retrieving events. It's your best companion for going with an Event-Driven Architecture for your Rails application.

# Railsy RES

## Railsy Rails Events Store

Since our architecture is modularized, we need a way for our components/engines to communicate with each other.

We do this by adding a pub/sub (or *event sourcing*) layer to our application via Rails Event Store.

We do not use RES directly but through the `railsy-events` engine, which wraps RES functionality and provide its own API. That would allow us to replace RES in the future (if necessary) without changing our application code.

# Railsy RES vs RES

- Class-independent event types
- Uses RES as interchangeable adapter
- Better testing tools
- Less verbose API and convention over configuration

# Railsy RES

```ruby
class ProfileCompleted < Railsy::Events::Event
  # (optional) event identifier is used for transmitting events
  # to subscribers.
  #
  # By default, identifier is equal to `name.underscore.gsub('/', '.')`.
  self.identifier = "profile_completed"

  # Add attributes accessors
  attributes :user_id

  # Sync attributes only available for sync subscribers
  sync_attributes :user
end
```

# Railsy RES

```ruby
event = ProfileCompleted.new(user_id: user.id)

# or with metadata
event = ProfileCompleted.new(
  user_id: user.id,
  metadata: { ip: request.remote_ip }
)

# then publish the event
Railsy::Events.publish(event)
```

# Railsy RES

```
initializer "my_engine.subscribe_to_events" do
  ActiveSupport.on_load "railsy-events" do |store|
    # async subscriber is invoked from background job,
    # enqueued after the current transaction commits
    store.subscribe MyEventHandler, to: ProfileCreated

    # anonymous handler (could only be synchronous)
    store.subscribe(to: ProfileCreated, sync: true) do |event|
      # do something
    end

    # subscribes to ProfileCreated automatically
    store.subscribe OnProfileCreated::DoThat
  end
end
```

Convention over configuration

# Railsy RES

```
engines/
  connect_by/
    events/
      connect_by/
        users/
          registered.rb
  chat_by/
    subscribers/
      connect_by/
        users/
          on_registered/
            create_chat_account.rb
```

# What we implement in the main app?

# Main App

- Authentication

# Authentication

```ruby
module ConnectBy
  class ApplicationController < Engine.config.application_controller.constantize
    raise "Must include ConnectBy::ControllerBehaviour" unless
      self < ConnectBy::ControllerBehaviour

    raise "Must implement #current_user method" unless
      instance_methods.include?(:current_user)
  end
end
```

Engines don’t care about how do you obtain the current user

# Main App

- Authentication
- Feature/system tests
- Locales and mailers templates
- Instrumentation and exception handling
- Configuration

# Gems

Or stop putting everything
into lib/ folder

# Gems

- Shared non-application specific code between engines
- Isolated tests
- Ability to share between applications in the future (e.g., GitHub Package Registry)

# Gems

```
gems/
  common-rubocop/
  common-testing/
  common-graphql/
  …
```

# common-rubocop

- Standard RuboCop configuration (based on `standard` gem)
- RuboCop plugins (e.g., `rubocop-rspec`)
- Custom cops

# common-rubocop

```
# .rubocop.yml
inherit_gem:
  common-rubocop: config/base.yml
```

# common-testing

- RSpec tools bundle (`factory_bot`, `faker`, `test-prof`, formatters)
- Common `spec_helper.rb` and `rails_helper.rb`

# common-testing

```
# rails_helper.rb
require "common/testing/rails_configuration"
```

# common-graphql

- Base classes (object, mutation)
- Additional scalar types
- Batch loaders
- Testing helpers

# Why engines?

And why not

# Why engines?

- Modular monolith instead of monolith monsters or micro-services hell
- Code (and tests) isolation
- Loose coupling (no more spaghetti logic)

# Why engines?

- Easy to upgrade frameworks (e.g., Rails) component by component
- Easy to re-use logic in another app
- ...and we can migrate to micro-services in the future (if we're brave enough)

# Why not engines?

- Not so good third-party gems support
- Not so good support within Rails itself

Thanks!

Спасибо!