Getting Started with GraphQL Ruby

An introduction to the server-side and client-side of GraphQL Ruby including best-practices for security and scalability.

Trevor Turk

Freelance software engineer with recent experience at:

- Outvote
- IFTTT
- Clearbit
- Basecamp

And for fun:

- Hello Weather

Why is GraphQL interesting?

- You get a standardized, documented, adaptable schema for your data
- Clients get an endpoint where they can make a single request to get what they need
- This solves for client complexity, multiple requests, and under/over-fetching

Simple example

```
{
  post {
    title
  }
}

{
  "post": {
    "title": "Hello, World"
  }
}
```

Example with multiple models

```
{
  post {
    title
    body
    author {
      name
    }
  }
}

{
  "post": {
    "title": "Hello, World",
    "body": "This is a post"
    "author": {
      "name": "Trevor"
    }
  }
}
```

Example schema

```
type Post {
  title: String
  body: String
  author: Author
}

type Author {
  name: String
  posts: [Post]
}
```

GraphQL Rubygems

- graphql-ruby
- graphiql
- graphql-batch
- graphql-client

Data model

```ruby
class Post < ApplicationRecord
  belongs_to :author
  scope :published, -> { where(published: true) }
end

class Author < ApplicationRecord
  has_many :posts
end
```

Database migrations

```
create_table :posts do |t|
  t.string :title
  t.boolean :published, default: false
  t.references :author
end

create_table :authors do |t|
  t.string :name
  t.string :auth_token
end
```

Install graphql-ruby

```
gem 'graphql'

rails g graphql:install --batch
      create  app/graphql/types/base_object.rb
      create  app/graphql/types/query_type.rb
      create  app/graphql/intro_to_graphql_ruby_schema.rb
add_root_type  query
      create  app/graphql/mutations/base_mutation.rb
      create  app/graphql/types/mutation_type.rb
add_root_type  mutation
      create  app/controllers/graphql_controller.rb
       route  post "/graphql", to: "graphql#execute"
       gemfile  graphql-batch
        create  app/graphql/loaders
       gemfile  graphiql-rails
         route  graphiql-rails
```

Gemfile

```
gem 'graphql'
gem 'graphiql-rails', group: :development
gem 'graphql-batch'
gem 'graphql-client'
```

Note this graphiql-rails/sprockets install issue

```
// app/assets/config/manifest.js

//= link graphiql/rails/application.css
//= link graphiql/rails/application.js
```

Routes

```
Rails.application.routes.draw do
  if Rails.env.development?
    mount GraphiQL::Rails::Engine, at: "/graphiql", graphql_path: "/graphql"
  end
  post "/graphql", to: "graphql#execute"
end
```

Controller

```
class GraphqlController < ApplicationController
  def execute
    result = MySchema.execute(
      params[:query],
      variables: params[:variables],
      context: { current_user: nil },
      operation_name: params[:operationName]
    )

    render json: result
  end
```

Schema

```
class MySchema < GraphQL::Schema
  mutation(Types::MutationType)
  query(Types::QueryType)
  use GraphQL::Batch
end
```

PostType

```ruby
# rails g graphql:object Post

class PostType < Types::BaseObject
  field :id, ID, null: true
  field :title, String, null: true
  field :published, Boolean, null: true
  field :author, Types::AuthorType, null: true
end
```

AuthorType

```ruby
# rails g graphql:object Author

class AuthorType < Types::BaseObject
  field :id, ID, null: true
  field :name, String, null: true
  field :posts, [Types::PostType], null: true
end
```

QueryType

```ruby
class QueryType < Types::BaseObject
  field :posts, [Types::PostType], null: true, description: "All published Posts"
  field :authors, [Types::AuthorType], null: true, description: "All Authors"

  def posts
    Post.published.all
  end

  def authors
    Author.all
  end
end
```

Seed data

```
# rake db:seed

author_1 = Author.create! name: "John", auth_token: "john_token"
author_2 = Author.create! name: "Jane", auth_token: "jane_token"

author_1.posts.create!(title: "John's post", published: true)

author_2.posts.create!(title: "Jane's post", published: true)
author_2.posts.create!(title: "Jane's draft post", published: false)
```

GraphiQL

Query for posts

```
query {
  posts {
    title
    author {
      name
    }
  }
}
```

```
{
  "data": {
    "posts": [
      {
        "title": "John's post",
        "author": {
          "name": "John"
        }
      },
      {
        "title": "Jane's post",
        "author": {
          "name": "Jane"
        }
      }
    ]
  }
}
```

Query for authors

```
query {
  authors {
    posts {
      title
    }
  }
}
```

```
{
  "data": {
    "authors": [
      {
        "posts": [
          {
            "title": "John's post"
          }
        ]
      },
      {
        "posts": [
          {
            "title": "Jane's post"
          },
          {
            "title": "Jane's draft post"
          }
        ]
      }
    ]
  }
}
```

Protect draft posts

```ruby
class AuthorType < Types::BaseObject
  field :posts, [Types::PostType], null: true

  def posts
    if context[:current_user] == object
      object.posts.all
    else
      object.posts.published.all
    end
  end
end
```

Authorization

```ruby
class GraphqlController < ApplicationController
  def execute
    result = MySchema.execute(
      params[:query],
      variables: params[:variables],
      context: { current_user: current_user },
      operation_name: params[:operationName]
    )

    render json: result
  end

  private

  def current_user
    if auth_token = request.headers[:Authorization]
      Author.find_by_auth_token(auth_token)
    end
  end
```

GraphiQL setup

- See GitHub's GraphQL API

```
# config/initializers/graphiql.rb

GraphiQL::Rails.config.headers['Authorization'] = -> (context) { "jane_token" }
```

```
query {
  authors {
    posts {
      title
      published
    }
  }
}
```

```
{
  "data": {
    "authors": [
      {
        "posts": [
          {
            "title": "John's post",
            "published": true
          }
        ]
      },
      {
        "posts": [
          {
            "title": "Jane's post",
            "published": true
          },
          {
            "title": "Jane's draft post",
            "published": false
          }
        ]
      }
    ]
  }
}
```

Nullifying fields

```
query {
  posts {
    id
    title
  }
}

{
  "data": {
    "posts": [
      {
        "id": "1",
        "title": "John's post"
      },
      {
        "id": "2",
        "title": "Jane's post"
      }
    ]
  }
}
```

Require current user

```
class PostType < Types::BaseObject
  field :id, ID, null: true

  def id
    if context[:current_user] == object.author
      object.id
    end
  end
```

```
query {
  posts {
    id
    title
  }
}

{
  "data": {
    "posts": [
      {
        "id": null,
        "title": "John's post"
      },
      {
        "id": "2",
        "title": "Jane's post"
      }
    ]
  }
}
```

Where to protect data?

- Per-field in `PostType`, `AuthorType`, etc
- Top-level in `QueryType`

Top-level protection

```ruby
class QueryType < Types::BaseObject
  field :authors, [Types::AuthorType], null: true, description: "All Authors, admin only"

  def authors
    if context[:current_user].admin?
      Author.all
    end
  end
end
```

Fields with arguments

```ruby
class QueryType < Types::BaseObject
  field :post, Types::PostType, null: true do
    argument :id, ID, required: true
  end

  def post(id:)
    Post.find_by_id(id)
  end
end
```

```
query {
  post(id: 1) {
    title
  }
}

{
  "data": {
    "post": {
      "title": "John's post"
    }
  }
}
```

```
query {
  post(id: 999) {
    title
  }
}

{
  "data": {
    "post": null
  }
}
```

Consider protecting every field

```ruby
class QueryType < Types::BaseObject
  field :post, Types::PostType, null: true do
    argument :id, ID, required: true
  end

  def post(id:)
    Post.published.find_by_id(id)
  end
end
```

N+1s

```
query {
  posts {
    title
    author {
      name
    }
  }
}

{
  "data": {
    "posts": [
      {
        "title": "John's post",
        "author": {
          "name": "John"
        }
      },
      {
        "title": "Jane's post",
        "author": {
          "name": "Jane"
        }
      }
    ]
  }
}
```

```
SELECT "authors".* FROM "authors" WHERE "authors"."id" = ? LIMIT ?  [["id", 1], ["LIMIT", 1]]
SELECT "authors".* FROM "authors" WHERE "authors"."id" = ? LIMIT ?  [["id", 2], ["LIMIT", 1]]
```

Rails includes

```
Post.published.all.each { |post| post.author.name }
```

```
SELECT "authors".* FROM "authors" WHERE "authors"."id" = ? LIMIT ?  [["id", 1], ["LIMIT", 1]]
SELECT "authors".* FROM "authors" WHERE "authors"."id" = ? LIMIT ?  [["id", 2], ["LIMIT", 1]]
```

```
Post.includes(:author).published.all.each { |post| post.author.name }
```

```
SELECT "authors".* FROM "authors" WHERE "authors"."id" IN (?, ?)  [["id", 1], ["id", 2]]
```

GraphQL preloading

- GraphQL::Batch
- Dataloader
- BatchLoader
- GraphQL::Preload

GraphQL::Batch Example

```ruby
module Loaders
  class FindLoader < GraphQL::Batch::Loader
    def initialize(model)
      @model = model
    end

    def perform(ids)
      records = @model.where(id: ids.uniq)
      records.each { |record| fulfill(record.id, record) }
      ids.each { |id| fulfill(id, nil) unless fulfilled?(id) }
    end
  end
end
```

```ruby
class PostType < Types::BaseObject
  field :author, Types::AuthorType, null: true

  def author
    Loaders::FindLoader.for(Author).load(object.author_id)
  end
end
```

```
SELECT "authors".* FROM "authors" WHERE "authors"."id" IN (?, ?)  [["id", 1], ["id", 2]]
```

Complex Loaders

- Loaders are often the biggest technical challenge when using GraphQL
- Don't be afraid to experiment with different loaders and refactor them over time
- Remember you can pass the current_user etc into a loader if needed

```
Loaders::FindLoader.for(Post, current_user: context[:current_user]).load(id)
```

Avoiding Complex Loaders

- You can simplify things with arguments, namespaces, and custom fields

```
query {
  posts(include_drafts: true) {
    title
  }
}
```

```
query {
  me {
    draft_posts {
      title
    }
  }
}
```

Monitor for N+1s

- Approvals
- Bullet
- rspec-sqlimit
- n_plus_one_control
- Rails strict_loading

Pagination

- Consider limit/offset, page, or graphql-ruby's built-in Relay connections:

```
class AuthorType < Types::BaseObject
  field :posts, PostType.connection_type, null: true

  def posts
    object.posts
  end
end
```

```
query {
  authors {
    name
    paginatedPosts(first: 1) {
      edges {
        node {
          title
        }
      }
      pageInfo {
        hasNextPage
        endCursor
      }
    }
  }
}
```

```
{
  "data": {
    "authors": [
      {
        "name": "John",
        "paginatedPosts": {
          "edges": [
            {
              "node": {
                "title": "John's post"
              }
            }
          ],
          "pageInfo": {
            "hasNextPage": false,
            "endCursor": "MQ"
          }
        }
      },
      {
        "name": "Jane",
        "paginatedPosts": {
          "edges": [
            {
              "node": {
                "title": "Jane's post"
              }
            }
          ],
          "pageInfo": {
            "hasNextPage": true,
            "endCursor": "MQ"
          }
        }
      }
    ]
  }
}
```

```
query {
  authors {
    name
    paginatedPosts(first: 1, after: "MQ") {
      edges {
        node {
          title
        }
      }
      pageInfo {
        hasNextPage
        endCursor
      }
    }
  }
}
```

```
{
  "data": {
    "authors": [
      {
        "name": "John",
        "paginatedPosts": {
          "edges": [],
          "pageInfo": {
            "hasNextPage": false,
            "endCursor": null
          }
        }
      },
      {
        "name": "Jane",
        "paginatedPosts": {
          "edges": [
            {
              "node": {
                "title": "Jane's draft post"
              }
            }
          ],
          "pageInfo": {
            "hasNextPage": true,
            "endCursor": "Mg"
          }
        }
      }
    ]
  }
}
```

Complexity

```ruby
class MySchema < GraphQL::Schema
  max_complexity 200
end
```

```ruby
class PostType < Types::BaseObject
  field :author, Types::AuthorType, null: true, complexity: 10
end
```

```
{
  "errors": [
    {
      "message": "Query has complexity of X, which exceeds max complexity of X"
    }
  ]
}
```

Depth

```
query {
  posts {
    title
    author {
      name
      posts {
        title
        author {
          name
          posts {
            title
          }
        }
      }
    }
  }
}
```

```ruby
class MySchema < GraphQL::Schema
  max_depth 20
end
```

```
{
  "errors": [
    {
      "message": "Query has depth of X, which exceeds max depth of X"
    }
  ]
}
```

- Note that you'll want to allow a large enough max complexity and depth so that your introspection query will work

Mutations

```ruby
class MutationType < Types::BaseObject
  field :create_post, mutation: Mutations::CreatePost
end
```

```ruby
class CreatePost < BaseMutation
  field :post, Types::PostType, null: true

  argument :title, String, required: true

  def resolve(title:)
    post = context[:current_user].posts.create!(title: title)
    { post: post }
  end
end
```

- Note Shopify has a recommendation for naming mutations

```
mutation createPost {
  createPost(input: { title: "Test post" }) {
    post {
      title
      author {
        name
      }
    }
  }
}

{
  "data": {
    "createPost": {
      "post": {
        "title": "Test post",
        "author": {
          "name": "Jane"
        }
      }
    }
  }
}
```

Execution errors

```ruby
class CreatePost < BaseMutation
  def resolve(title:)
    raise GraphQL::ExecutionError, "not logged in" unless context[:current_user]
  end
end
```

```
{
  "data": {
    "createPost": null
  },
  "errors": [
    {
      "message": "not logged in",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "createPost"
      ]
    }
  ]
}
```

Validation errors

```ruby
class Post < ApplicationRecord
  validates :title, uniqueness: true
end
```

```ruby
module Types
  class ErrorType < Types::BaseObject
    field :message, String, null: false
  end
end
```

```ruby
class CreatePost < BaseMutation
  field :post, Types::PostType, null: true
  field :errors, [Types::ErrorType], null: true

  def resolve(title:)
    post = context[:current_user].posts.create(title: title)

    if post.valid?
      {
        post: post,
        errors: nil
      }
    else
      {
        post: nil,
        errors: post.errors.map do |attribute, message|
          OpenStruct.new(message: "#{attribute} #{message}")
        end
      }
    end
  end
end
```

```
mutation createPost {
  createPost(input: { title: "Test post" }) {
    post {
      title
      author {
        name
      }
    }
    errors {
      message
    }
  }
}

{
  "data": {
    "createPost": {
      "post": null,
      "errors": [
        {
          "message": "title has already been taken"
        }
      ]
    }
  }
}
```

- Note graphql-ruby has a recommendation for formatting mutation errors

graphql-client

```
gem 'graphql-client'
```

```ruby
require "graphql/client"
require "graphql/client/http"

module Graph
  HTTP = GraphQL::Client::HTTP.new("http://localhost:3000/graphql")
  Schema = GraphQL::Client.load_schema(HTTP)
  Client = GraphQL::Client.new(schema: Schema, execute: HTTP)

  PostsQuery = Client.parse <<-'GRAPHQL'
    query {
      posts {
        title
      }
    }
  GRAPHQL
end
```

```
result = Graph::Client.query(Graph::PostsQuery)
result.data.posts.map(&:title)
=> ["John's post", "Jane's post"]
```

Variables

```
PostQuery = Client.parse <<-'GRAPHQL'
  query($id: ID!) {
    post(id: $id) {
      title
    }
  }
GRAPHQL
```

```
result = Graph::Client.query(Graph::PostQuery, variables: { id: "1" })
result.data.post.title
=> "John's post"
```

Context

```
result = Graph::Client.query(
  Graph::PostQuery,
  variables: { id: "3" },
  context: { current_user: Post.find(3).author }
)
result.data.post.title
=> "Jane's draft post"
```

Instrumenting by operation name

```
Processing by GraphqlController#execute as JSON
  Parameters: {
    "query"=>"query Graph__PostsQuery {\n  posts {\n    title\n  }\n}",
    "operationName"=>"Graph__PostsQuery"
  }
```

```
class GraphqlController < ApplicationController
  def execute
    NewRelic.trace_execution(params[:operationName]) do
      result = MySchema.execute(query)
    end
  end
end
```

Recommended reading

- Introduction to GraphQL
- GraphQL Ruby Guides
- Shopify GraphQL Design Tutorial

Helpful community

- GraphQL on Slack #ruby

Thanks!