GraphQL - a new kind of API

- APIs are for programmatic interaction with a web service
  - Many types of API (SOAP, REST, etc)
  - REpresentation State Transfer defined by Roy Fielding in ~2000
  - We currently have a Level Four Rest API™
  - Our frontend is increasingly a Vue-based web application for it
  - Central promise of REST is a universal client for every API
- GraphQL is a Level Zero Rest API™
  - Rejects ~20 years of Best Practice™
  - We give up on the single-client-for-everything aspiration
  - Anything (else) REST can do, GraphQL can do… better?
  - Focus on making an API that can serve dedicated clients better
- GraphQL optimizes for flexible queries, served efficiently
“Why would you do that?”

- MR tries to improve performance of elasticsearch-backed commit search
- Existing REST API intervenes...
  - Fixed set of fields, returned for every request. Projections difficult to add
  - Compatibility guarantee, even though it’s likely nobody even uses these fields
  - We can’t actually tell if anyone even uses these fields
  - Hard to introduce lazy evaluation to resolve underlying N+1 issue
- If this were GraphQL...
  - Dynamic set of fields, selected only if client needs them. No need for projections
  - Living API - we could deprecate & remove these fields if we absolutely had to
  - Instrumentation can give us metrics on field use to inform deprecation decisions
  - Lazy evaluation built-in from the start
GraphQL at Facebook and GitLab

- Started in 2012 following shift from web to native applications on mobile
- Address challenges encountered using REST: “Slow, Fragile, Tedious”
  - A great talk by Lee Byron: https://www.youtube.com/watch?v=F-OizdRJh1U
- React.js: open-sourced in 2012
- Relay + GraphQL: open-sourced in 2015. Specification + reference implementation
- We set a plan for adoption in 2017
  - https://gitlab.com/gitlab-org/gitlab-ce/issues/34754
- Issues with lack of patent grant in specification raised & resolved in 2017
  - https://github.com/facebook/graphql/issues/351
- **Alpha** support merged into GitLab in 2018
  - https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/19008
- 12+ independent implementations by 2019.
- GraphQL foundation forming in March 2019
- Issue suggestions feature uses GraphQL!
GraphQL basics

- **Fields**
  - a. Everything is a field
  - b. Fields can take arguments

- **Types**
  - a. Every field has a type
  - b. Some built-in, some user-defined
  - c. QueryType and MutationType are special
  - d. Types (can) have fields, forming a graph

- **Schema**
  - a. Specifies available fields and types
  - b. Is a field. I wasn’t kidding about this bit

- **Queries**
  - a. Special mini-language, think SQL
  - b. Specifies the fields the client wants
  - c. Can be parameterized

https://gitlab.com/-/graphql-explorer
Example query - basic

```
1  {
  project(fullPath:"gitlab-com/www-gitlab-com") {
    id
    issue iid:1 {
      webUrl
    }
  }
  gitlab_ce:project(fullPath:"gitlab-org/gitlab-ce") {
    id
  }
  gitlab_ee:project(fullPath:"gitlab-org/gitlab-ee") {
    id
  }
}
```

- `project` is a field of **QueryType**, type **Project**
- `gitlab_ce:project` is the same, renamed
- `fullPath` is an argument selecting one project
  - Specify exactly the fields you want
- `issue` is a field of **Project**, type **Issue**
  - Specify exactly the fields you want
- Turned into JSON and transmitted to server
- Can load all 3 projects in one query(ish)

- Response is JSON
- Closely mirrors request semantics
- Only what I asked for is there
- Schema guarantees the server has what I want
- We can **restrict the complexity of queries**
Example query - pagination & caching

- `issues` is of type `IssueConnection`
- Includes `pageInfo`, `edges`
- Each `edge` has its own `cursor` too
- Support offset, keyset, etc, pagination
- `endCursor` to page forwards
- `startCursor` to page backwards
- REST API puts this into headers instead

- No REST, so normal caching is out
- We haven’t handled this yet
- GraphQL way:
  - Everything gets a GUID
  - Clients build their own caches
Example query - parameters and fragments

```graphql
fragment projectFields on Project {
  id
  forksCount
}

query ProjectsWithGitLab($path:ID!) {
  project(fullPath:$path) {
    ...projectFields
    issue(iid:1) {
      iid
      createdAt
    }
  }
  gitlab_ce:project(fullPath:"gitlab-org/gitlab-ce") {
    ...projectFields
  }
  gitlab_ce:project(fullPath:"gitlab-org/gitlab-ce") {
    ...projectFields
  }
  gitlab_e:project(fullPath:"gitlab-org/gitlab-e") {
    ...projectFields
  }
}

QUERY VARIABLES

{
  "path": "gitlab-com/www-gitlab-com"
}
```
Example query - directives

- Make some fields conditional.
  - a. include if
  - b. include unless
  - c. skip if
  - d. skip unless
- They’re in the schema too
- Fewer, more-general, static queries
Example mutation

- We only have this one right now!
- Very RPC-like, which is by design
- Can run multiple mutations too
- We’ve got a lot to learn here
Subscriptions

- Websocketty
- I can’t really dive deeply into this, I’ve never touched it
- Support for Relay and ActionCable out of the box, though \o/
- Historically, we’ve been unable to support long-lived connections to gitlab backend
- Move to Puma might unlock some of these capabilities
Authorization & Authentication

- **Authentication**
  - Similar to REST API
  - Supports cookies and token-based authentication. CSRF protection for cookie auth

- **Authorization**
  - Individual fields are authorizable
  - Efficiently authorizing arbitrarily complex queries can be a challenge
  - Requests can partially succeed, leaving unauthorized fields blank, or completely fail
  - You can query your own permissions for a field
Adding a new GraphQL endpoint

- **Important files:**
  - `app/assets/javascripts/lib/graphql.js`
  - `app/assets/javascripts/**/*.graphql`
  - `app/controllers/graphql_controller.rb`
  - `app/graphql/gitlab_schema.rb`
  - `app/graphql/functions/*`
  - `app/graphql/mutations/*`
  - `app/graphql/resolvers/*`
  - `app/graphql/types/*`
  - `lib/gitlab/graphql/*`
  - `spec/graphql/*`
  - `spec/requests/api/graphql/*`

- **EE-only support is coming in 11.10**

- You’re adding *fields* (of course)
- New top-level query fields in **QueryType**
- Otherwise added to another type
- Resolvers gather data without N+1 issues
- Functions are simple resolvers for a few fields
- Specs are composable, just like the fields
  - Request specs can be simpler
- Frontend defines a static query in `.graphql` file
- Applies it to the `graphql` client to do the thing
- Reimplement REST API in terms of GraphQL

- **How does that search API look?**
  - GitHub’s (sneaky peek)
- Ouch. OK, let’s look at file templates instead

- [https://graphql.org/](https://graphql.org/)
- [https://graphql-ruby.org/](https://graphql-ruby.org/)