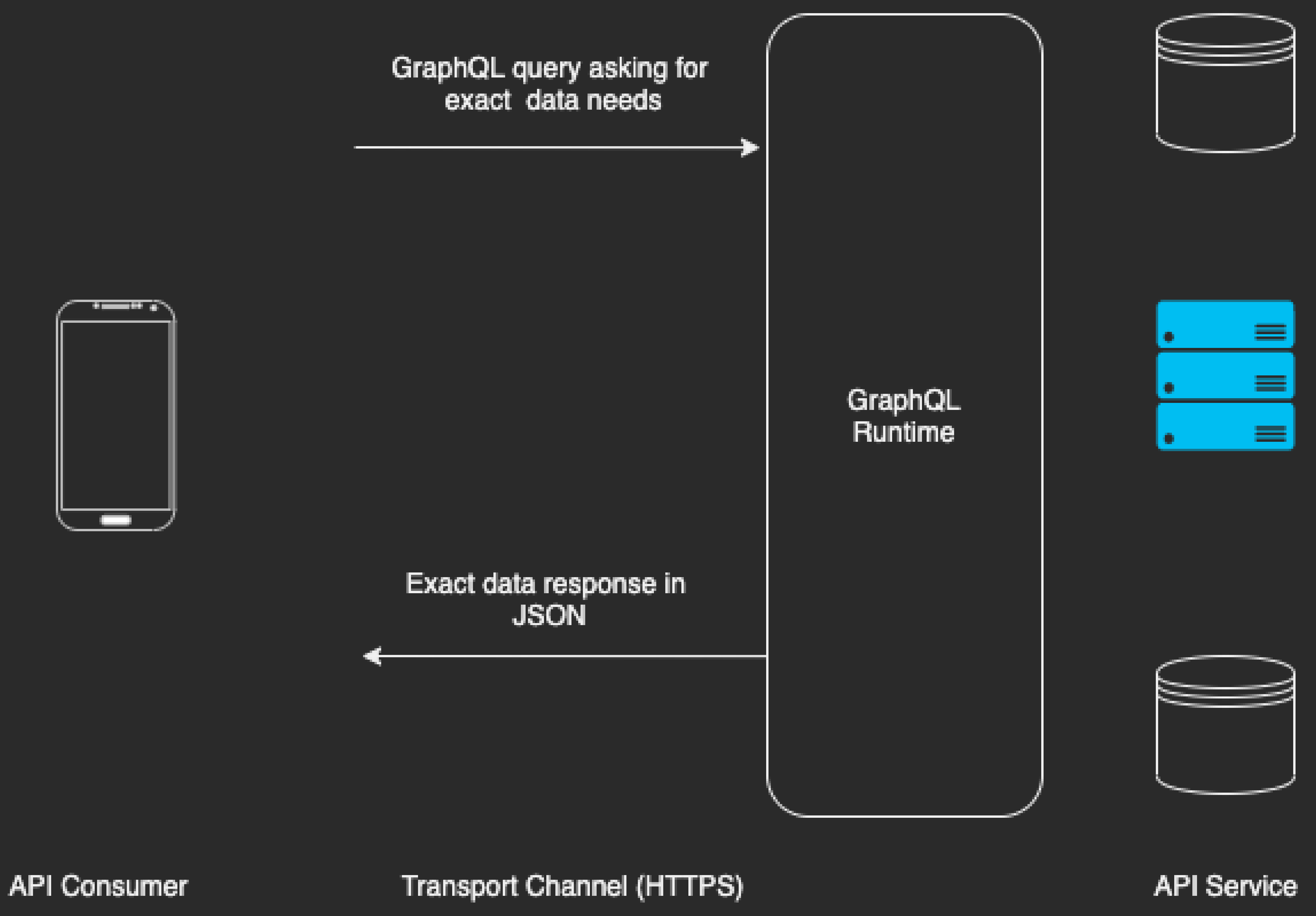
GraphQL 101
(Ruby Edition)

Introduction

- **Graph**: data structure **QL**: query language.
- Query language for data API.
- Declarative, flexible, and efficient.
- Optimize data communication between client and server.
- JSON is commonly used for data communication.
- Allows API introspection.

Overview


GraphQL query asking for exact data needs
GraphQL Runtime
Exact data response in JSON
API Consumer
Transport Channel (HTTPS)
API Service

GraphQL Way

TYPED SCHEMA

- All fields have types, either primitive or custom.

DECLARATIVE LANGUAGE

- The client language is declarative instead of imperative.

SINGLE ENDPOINT AND CLIENT LANGUAGE

- These allows easy client access, without under-fetching or over-fetching.

SIMPLE VERSIONING

- Versioning is avoided all together.
- To maintain compatbility add new fields and types without removing the old ones.

Cons

SECURITY

- Overly complex queries can be used for resource-exhaustion attacks.
- Implement cost analysis on the query in advance.
- Enforce limits on the amount of data that can be consumed.
- Implement timeout to kill long running requests.
- Implement other general API security restrictions like whitelist, rate limiting etc.

CACHING AND OPTIMIZING

- REST APIs can easily be cached because of dictionrary like nature, using URL as the cache key.
- In graphql we can use the query text as key to cache its response.
- We can normalize a query response into a flat collection of records with a global unique ID.

LEARNING CURVE

- Steep learning curve.
- Have to learn the syntax of a new declarative language.
- Have to understand concepts like schemas, resolvers etc.

Sample Application

A simple TODO application (Ruby/React)

- Signup
- Login
- Create a todo.
- Update a todo.
- Delete a todo.
- View realtime updates.

SOURCE CODE

- https://github.com/itsgg/todo-app

API DOCUMENTATION

- https://documenter.getpostman.com/view/13745138/TzY7cD19

Schema

Also called as

- Schema Definition Language (SDL)
- Interface Definition Language (IDL)

```
type Todo(id: Int!) {
  title: String!
  complete: Boolean!
}
```

> Exclamation(!) after the types means that they cannot be empty

Queries

Represent **READ** operations.

REQUEST

```
query {
  todo(id: 1) {
    id
    title
    complete
  }
}
```

RESPONSE

```
{
  "data": {
    "todo": {
      "id": 1,
      "title": "Buy milk",
      "complete": false
    }
  }
}
```

Mutations

Represent **WRITE**-then-**READ** operations.

REQUEST

```
mutation {
  createTodo(title: "Buy Dogecoin!", complete: true) {
    todo {
      id
      title
      complete
    }
  }
}
```

RESPONSE

```
{
  "data": {
    "todo": {
      "id": 2,
      "title": "Buy Dogecoin!",
      "complete": true
    }


  }
```

Subscriptions

Continous **READ** operations

REQUEST

```
subscription todosChanged {
  todo {
    id
    title
    complete
  }
}
```

RESPONSE

```
{
  "data": {
    "todos": [
      { "id": 1, "title": "Buy milk", "complete": false },
      { "id": 2, "title": "Buy Dogecoin!", "complete": true }
    ]
  }
}
```

Requests

DOCUMENT

- A request document contains queries, mutations, subscriptions and fragments

VARIABLES

- An object that represents values used in the document.

META-INFORMATION

- Which operation to execute if the document contains more than one operations.

```
query Todo {
  todo(id: $userId) {
    title
    ...userInfo
  }
}

fragment userInfo on User {
  email
}
```

Mutation

```ruby
module Mutations
  class Register < Types::BaseMutation
    type Types::UserType
    null false

    argument :email, String, required: true
    argument :password, String, required: true

    def resolve(email:, password:)
      User.create! email: email, password: password
    end
  end
end
```

```ruby
module Types
  class MutationType < Types::BaseObject
    field :register, mutation: Mutations::Register
  end
end
```

Query Resolver

```ruby
module Resolvers
  class TodoResolver < Types::BaseResolver
    type [Types::TodoType], null: false

    def resolve
      user = context[:current_user]
      authorize! user, to: :manage_todos?
      user.todos
    end
  end
end
```

```ruby
module Types
  class QueryType < Types::BaseObject
    field :todos, resolver: Resolvers::TodoResolver
  end
end
```

Subscriptions

```ruby
module Subscriptions
  class TodoCreated < GraphQL::Schema::Subscription
    payload_type Types::TodoType
    null true

    def subscribe
      nil
    end

    def update
      object
    end
  end
end
```

```ruby
module Types
  class SubscriptionType < Types::BaseObject
    field :todo_created, subscription: Subscriptions::TodoCreated
  end
end
```

Configuring Subscriptions

```ruby
class TodoAppSchema < GraphQL::Schema
  use GraphQL::Subscriptions::ActionCableSubscriptions

  mutation(Types::MutationType)
  query(Types::QueryType)
  subscription(Types::SubscriptionType)
end
```

```ruby
class GraphqlChannel < ApplicationCable::Channel
  def subscribed
    @subscription_ids = []
  end

  def execute(data)
    # Snip....
    transmit({ result: result.subscription? ? { data: nil } : result.to_h, more: result.subscription? })
  end

  def unsubscribed
    @subscription_ids.each do |sid|
      TodoAppSchema.subscriptions.delete_subscription(sid)
    end
  end
end
```

Triggering Subscriptions

```ruby
class Todo < ApplicationRecord
  after_commit :notify_new_todo

  private

  def notify_new_todo
    TodoAppSchema.subscriptions.trigger('todoCreated', {}, self)
  end
end
```

ONLY TO FEW CLIENTS

```ruby
subscription_scope :current_user_id
TodoAppSchema.subscriptions.trigger('todoCreated', {}, self, scope: self.user_id)
```

Testing Query Resolvers

```ruby
require 'rails_helper'

module Resolvers
  RSpec.describe TodoResolver, type: :request do
    # snip ...

    describe '.resolve' do
      it 'success' do
        post_to_graphql(query: gql, token: user.token, variables: {})
        json = JSON.parse(response.body)
        data = json['data']
        expect(data['todos'].count).to eq(todos.count)
      end

      it 'failure' do
        post_to_graphql(query: gql, token: 'INVALID', variables: {})
        json = JSON.parse(response.body)
        expect(json['errors']).to be_present
      end
    end
  end
end
```

Testing Mutations

```ruby
require 'rails_helper'

module Mutations
  RSpec.describe CreateTodo, type: :request do
    # snip ...
    describe '.resolve' do
      it 'success' do
        expect do
          post_to_graphql(query: gql, token: user.token, variables: { title: 'Change the world' })
        end.to change(Todo, :count)
      end

      it 'failure' do
        expect do
          post_to_graphql(query: gql, token: 'INVALID', variables: { title: 'Give up' })
        end.not_to change(Todo, :count)
      end
    end
  end
end
```