Dynamic GraphQL Schemas

Rob Van Gennip · @ravangen

shopify

Toronto 🇨🇦

APIs @ Shopify

@ravangen

Momo 🐶

Dynamic schemas reduce cognitive load on users

Defining Schema

Custom Behaviour

GraphQL Versioning

Schema First

- Describe GraphQL service’s type system
- Schema Definition Language (SDL)
- Interoperable, language agnostic
- Static, customize behaviour with directives

```
schema {
  query: Query
}

scalar DateTime

interface Node {
  id: ID!
}

type Query {
  product(id: ID!): Product
}

type Product implements Node {
  id: ID!
  name: String!
  description(truncateAt: Int): String!
  body: String @deprecated(reason: "Use `description` instead")
  createdAt: DateTime!
}
```

Code First

- Expressed in programming language
- Seamless code generation
- Exportable SDL artifact
- Convenient modularization

```ruby
module Types
  class ProductType < Types::BaseObject
    implements GraphQL::Types::Relay::Node

    global_id_field :id # Defines field :id, ID, null: false

    field :name, String, null: false

    field :description, String, null: false do
      argument :truncate_at, Integer, required: false
    end
    def description(truncate_at: nil)
      if truncate_at.nil?
        object.description
      else
        object.description.truncate(truncate_at, separator: /\s/)
      end
    end

    field :body, String, null: true, deprecation_reason: "Use `description` instead."

    field :created_at, GraphQL::Types::ISO8601DateTime, null: false
  end
end
```

“Dynamic” Schema

- Generated from metadata
- Code generation
- Transformation

```ruby
module Types
  class QueryType < Types::BaseObject
    # ...

    field :product, ProductType, null: true do
      argument :id, ID, required: true
    end

    field :products, ProductType.connection_type, null: false
  end
end
```

```graphql
type Query {
  product(id: ID!): Product
  products(after: String, before: String, first: Int, last: Int): ProductConnection!
}

type ProductConnection {
  edges: [ProductEdge]
  nodes: [Product]
  pageInfo: PageInfo!
}

type ProductEdge {
  cursor: String!
  node: Product
}
```

Schema Visibility

```graphql
# full-schema.graphql
directive @visibility(
  schema: SchemaType!
) on OBJECT | FIELD_DEFINITION | ARGUMENT_DEFINITION # ...

enum SchemaType { INTERNAL PUBLIC }

type Product implements Node {
  # ...

  # Server exposes field if execution context permits access to specific schema
  # Acts as if did not exist as part of the schema (not discoverable)
  costOfGoodsManufactured: Float @visibility(schema: INTERNAL)
}
```

```graphql
# internal-schema.graphql
type Product implements Node {
  # ...

  costOfGoodsManufactured: Float
}
```

```graphql
# public-schema.graphql
type Product implements Node {
  # ...

  # No costOfGoodsManufactured field
}
```

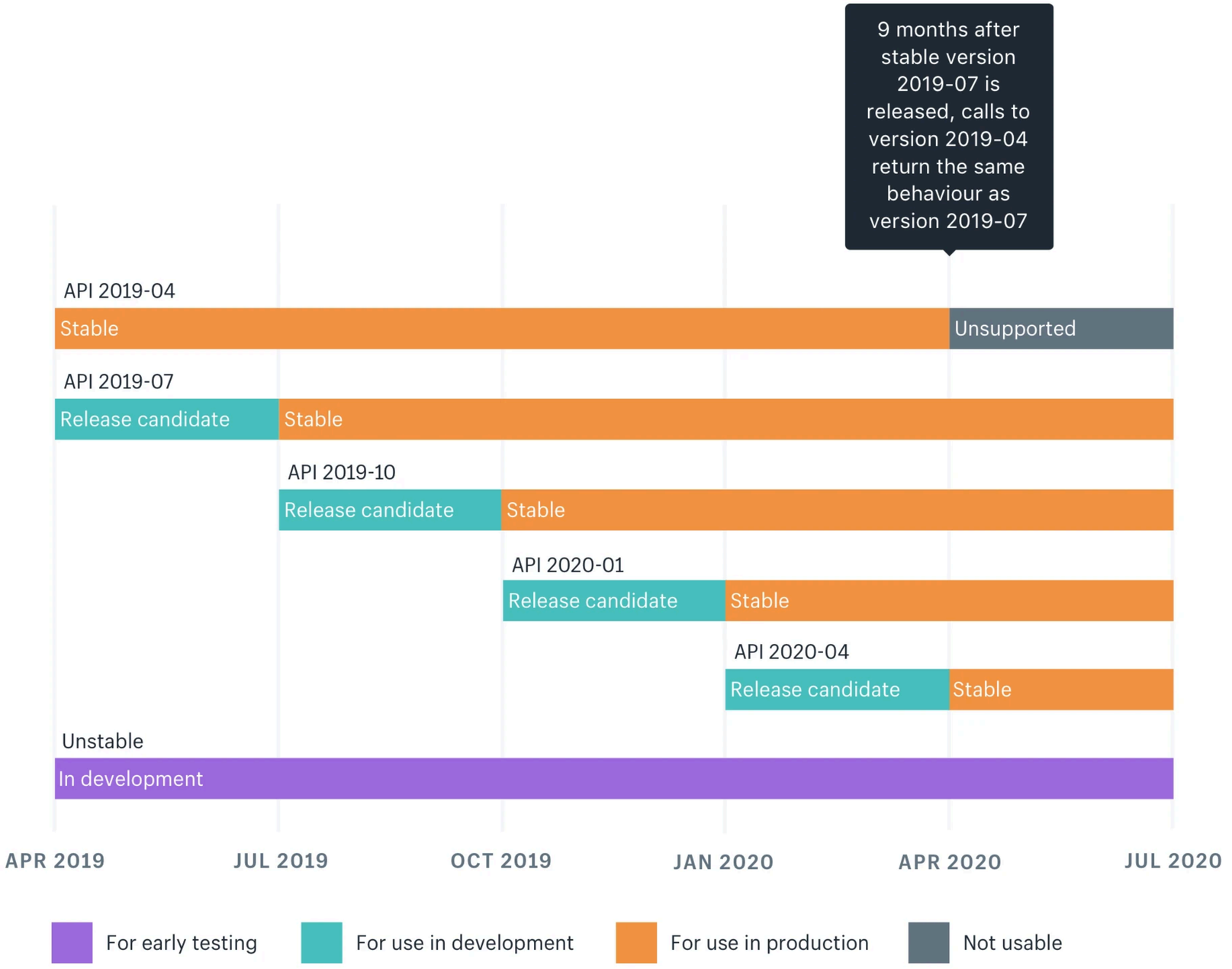
```ruby
module Types
  class BaseField < GraphQL::Schema::Field
    argument_class Types::BaseArgument

    # Pass `field ..., schema: :internal` to hide this field
    def initialize(*args, schema: nil, **kwargs, &block)
      @schema = schema
      super(*args, **kwargs, &block)
    end

    def visible?(context)
      # if `schema` argument is for internal, hide field from non-internal clients
      super && (@schema != :internal || context[:has_internal_access])
    end
  end

  class ProductType < Types::BaseObject
    # ...

    field :cost_of_goods_manufactured, Float, null: false, schema: :internal
  end
end
```

Versioning GraphQL

- APIs are forever contracts, should never break
- Default to avoid versioning
- Continuous improvement
- External impact

Shopify

- Making more things possible
- Cadence is consistent and predictable
- Actionable communication
- Developer preview


9 months after stable version 2019-07 is released, calls to version 2019-04 return the same behaviour as version 2019-07
API 2019-04
Stable
Unsupported
API 2019-07
Release candidate
Stable
API 2019-10
Release candidate
Stable
API 2020-01
Release candidate
Stable
API 2020-04
Release candidate
Stable
Unstable
In development
APR 2019
JUL 2019
OCT 2019
JAN 2020
APR 2020
JUL 2020
For early testing
For use in development
For use in production
Not usable

Drawbacks

- Shared base schema across versions
- Version creep in fields and type names
- Complicates schema stitching and federation

```
field :cost_of_goods_manufactured, Float, null: false, versions: "2019-2021"
field :cost_of_goods_manufactured_v2, Money, null: false, versions: "2021-"
```

Code Samples

github.com/ravangen/graphql-conf-2021