GraphQLConf

September 10-12, 2024 | San Francisco, CA

hosted by GraphQL Foundation

Dynamically Serving a GraphQL API with Custom Types at Runtime

Emily Li
Software Engineer, Benchling

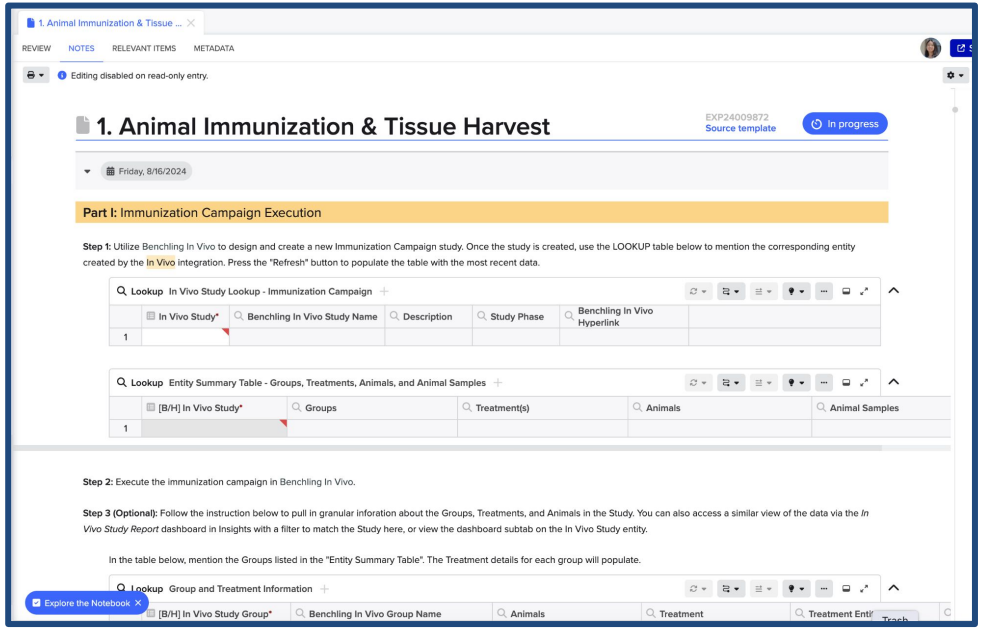
#GraphQLConf

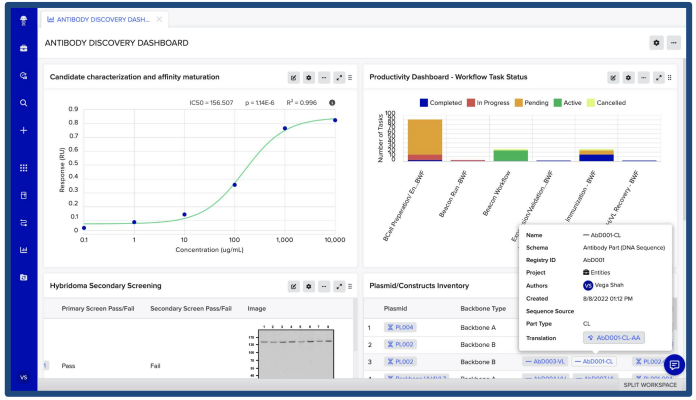
Agenda

1. Benchling use case
2. Existing GraphQL APIs
3. Dynamic graph generation
4. Technical challenges
5. Q&A

Benchling Use Case

What is Benchling?





Biology-first platform for scientific data, collaboration, and insights.

Biomolecule Design Application

pSpCas9(BB)-2A-GFP_(PX4

SEQUENCE MAP | LINEAR MAP | PLASMID | DESCRIPTION | METADATA | RELEVANT ITEMS | Share

Create | Analyze | Copy | Create PDF

gagggcctatttcccatgattccttcatatttgcatatacgatacaaggctgttagagagataattggaattaatttga
ctcccggataaagggtactaaggaagtataaacgtatatgctatgttccgacaatctctctattaaccttaattaaact

hU6_promoter

10 20 30 40 50 60 70

ctgtaaacacaaagatattagtacaaaatacgtgacgtagaaagtaataatttcttgggtagtttgcagttttaaaatt
gacatttgtgtttctataatcatgttttatgcactgcatctttcattattaaagaacccatcaaacgtcaaaattttaa

hU6_promoter

80 90 100 110 120 130 140 150

NdeI

atgttttaaaatggactatcatatgcttaccgtaacttgaaagtatttcgatttcttggctttatatatcttGTGGAAA
tacaaaattttacctgatagtatacgaatggcattgaactttcataaagctaaagaaccgaaatatatagaaCACCTTT

LKO1_5_primer

hU6_promoter

160 170 180 190 200 210 220 230

BbsI BbsI

GGACGAAACACCggGTCTTCgaGAAGACctgttttagagctaGAAAtagcaagttaaaataaggctagtccgttatcaa
CCTGCTTTGTGGccCAGAAGctCTTCTGgacaaaatctcgatCTTTatcgttcaattttattccgatcaggcaatagtt

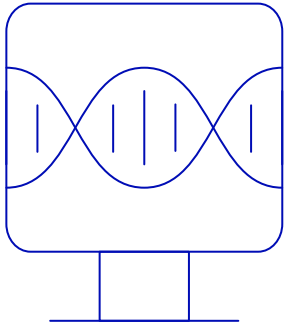
240 250 260 270 280 290 300 310

cttgaaaaagtggcaccgagtcggtgcTTTTTTgttttagagctagaaatagcaagttaaaataaggctagtccgtTTT

pGEX_3_primer, AmpR_promoter, +4
BsmBI
PvuI
FspI
SacII
PsiI, SspI
f1_origin, DraIII
XmaI, SmaI, SrfI, +5
TspMI
SphI, EagI, +5
+3
EcoRI
+2
pBR322_origin
7500
Ampicillin
bGH_PA_terminator
EGFP
AflIII
PciI
+3
BbsI
BbsI
+3
+2
+1
+1
+1
+1
+1
+1
+1
pSpCas9(BB)-2A-GFP_(PX4
9289 bp
NLS
SV40 NLS
SV40 NLS
+3
+3
BstXI
BsmI
EagI
XcmI
BarI, +1
KflI
SphI
AjuI, BspEI
BmgBI
PmlI, BspEI
CspCI
PflMI
AjuI, Eco53kI, SacI, +1
5000
BglII
PflMI
EcoNI
BmgBI
BstXI
EcoNI
BstAPI
2500
PspOMI, ApaI, BstAPI, +1
PasI
EcoRV

BASES 9289 | ASSEMBLY WIZARD | SPLIT WORKSPACE

Example Benchling Application



Biomolecule Design

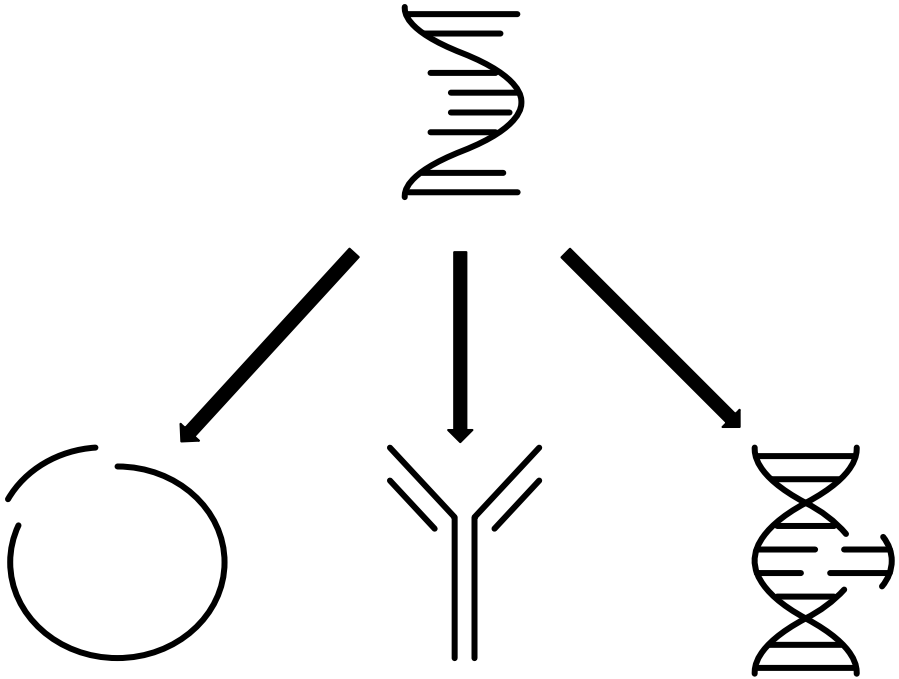
Sequence
id bases

Sequence DB table

```
type BenchlingSequence
  id: String
  bases: String
```

GraphQL SDL type

Extensible Types



Benchling DNA Sequence

Custom DNA Sequences

Defining Custom Types

Prefix*

Name*

☐ Use Registry ID as display label

☐ Include Registry ID in chips

Example | Example ID

Entity type*

Select a type

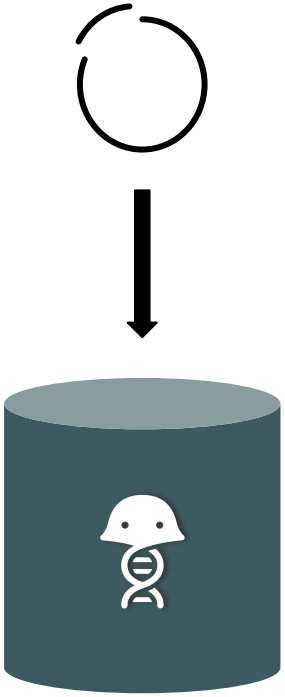
Constraint +

This schema does not have a constraint

Fieldsets +

No fieldsets implemented

Entity fields +	Warehouse name	Required	Integration	Multi-select	Definition
This schema does not have any entity fields					



Recording Data with Custom Types

Name	CHO002
Schema	Plasmid
Registry ID	CL002
Authors	TM Tom McCrory
Created	08/17/2023 08:08 AM
Parent Cell Line	CHO001
Cell Type	CHO
Plasmid Prep(s)	Backbone A-LC001-001 Backbone A-HC001-003
Expressed Antibody Complex	IgG001

Custom type

Built-in fields

Custom fields

Why build a GraphQL API?

← Choose different creation method

Registry schemas

1 Select registry schema

Entity schema

Select a schema... Filters

2 Select columns

3 Add Results

4 Name table

Table name

Cancel Add table

No table preview available yet

Select a registry schema to see a preview of your table.

Custom Types

```
interface SequenceInterface {
  id: String
  bases: String
}

type BenchlingSequence implements SequenceInterface {
  id: String
  bases: String
}
```

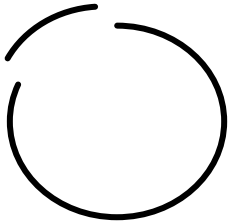
Benchling built-in type

```
type Plasmid implements SequenceInterface {
  id: String
  bases: String
  weight: Float
}
```

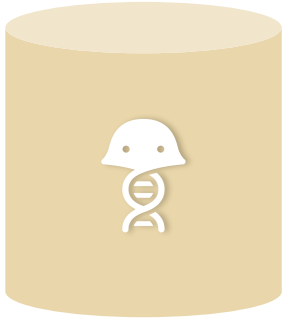
Customer-defined type

Custom Graphs

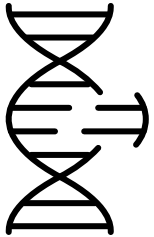
Customer A



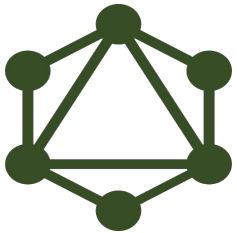
Sequence A

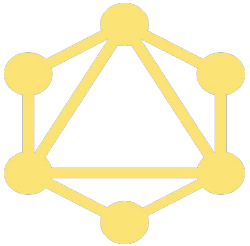


Customer B



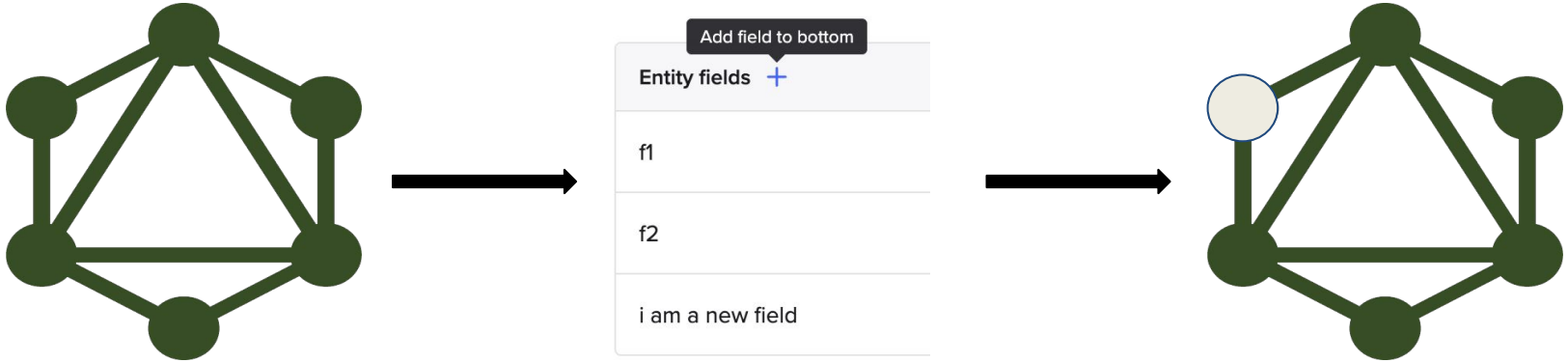
Sequence B





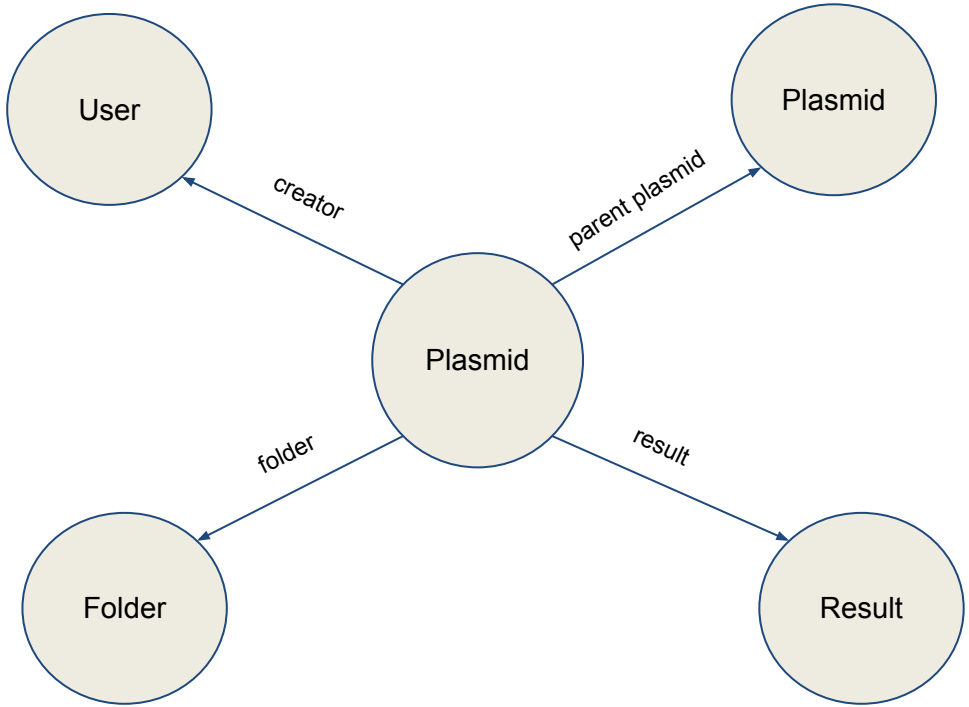
Because of custom types, each customer's graph looks different.

Mutable graphs



Types in the graph can change at any point in time.

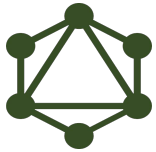
Highly Connected Graphs



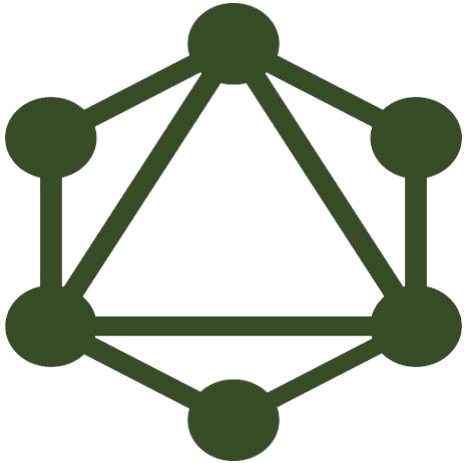
Types are often highly connected to other types.

Graph Scale

100+ built-in types



5000+ total types



40,000+ enum members

Problem Statement

We need to dynamically generate a graph per customer at runtime at scale.

Existing GraphQL APIs

Public GraphQL APIs

GraphQL API / Overview /

Explorer

For more information about how to use the explorer, see "Using the Explorer."

GitHub GraphQL API

ready to explore! Sign out

Heads up! GitHub's GraphQL Explorer makes use of your **real, live, production data.**

Viewer
Organization

+ GraphiQL

```
# Type queries here, and you will see intellige
# aware of GitHub's current GraphQL schema, equ
# syntax and validation errors highlighted with
# get you started with a simple query showing y

query {
  viewer {
    login
  }
}
```

```
{
  "data": {
    "viewer": {
      "login": "el2857"
    }
  }
}
```

Shopify Storefront API GraphiQL explorer

Start exploring Shopify's Storefront GraphQL API on our demo shop. Use the embedde

Caution

This is a **read-only** demo. To build queries and mutations install Shopify's Gra

Launch the Explorer in its own window.

Version: 2024-07 (Latest)

Shop
Product

GraphiQL Prettify Explorer

```
{
  shop {
    name
  }
}
```

```
{
  "data": {
    "shop": {
      "name": "graphql"
    }
  }
}
```

Github

Shopify

Contain on the order of 100s of types in the schema.

Static Schemas

Public schema

Download the public schema for the GitHub GraphQL API.

You can perform introspection against the GraphQL API directly.

Alternatively, you can download the latest version of the public schema here:

schema.docs.graphql

```
"""
The connection type for ProjectV2Item.
"""
type ProjectV2ItemConnection {
  """
  A list of edges.
  """
  edges: [ProjectV2ItemEdge]

  """
  A list of nodes.
  """
  nodes: [ProjectV2Item]

  """
  Information to aid in pagination.
  """
  pageInfo: PageInfo!

  """
  Identifies the total count of items in the connection.
  """
  totalCount: Int!
}

"""
Types that can be inside Project Items.
"""
union ProjectV2ItemContent = DraftIssue | Issue | PullRequest

"""
An edge in a connection.
"""
type ProjectV2ItemEdge {
  """
  A cursor for use in pagination.
  """
```

Github

GraphQL Frameworks

```python
from ariadne import gql, load_schema_from_path

# load schema from file...
schema = load_schema_from_path("schema.graphql")

# ...directory containing graphql files...
schema = load_schema_from_path("schema")

# ...or inside Python files
schema = gql("""
  type Query {
    user: User
  }

  type User {
    id: ID
    username: String!
  }
""")
```

Ariadne

```python
@strawberry.type
class Book:
    title: str
    author: str


@strawberry.type
class Query:
    books: typing.List[Book]
```

Strawberry

Building with Strawberry

```python
@strawberry.type
class Book:
    title: str
    author: str


@strawberry.type
class Query:
    books: typing.List[Book]
```

1. Define type

```python
def get_books():
    return [
        Book(
            title="The Great Gatsby",
            author="F. Scott Fitzgerald",
        ),
    ]
```

2. Define dataset

```python
@strawberry.type
class Query:
    books: typing.List[Book] = strawberry.field(resolver=get_books)
```

3. Define resolver

```python
schema = strawberry.Schema(query=Query)
```

4. Create schema

Examples taken from https://strawberry.rocks/docs

Dynamic Graph Generation

Type Definition via GraphQL SDL

```
type Plasmid implements SequenceInterface {
  id: String
  bases: String
  weight: Float
}
```

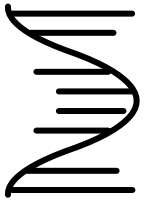
✅ Pros:

- Spec-first development
- Clear and declarative syntax
- Flexibility and extensibility

❌ Cons:

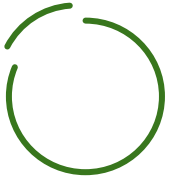
- Can be verbose
- Can't specify nullable but required values
- No directives on enum members

Combined SDL



Built-in types

+



Custom types

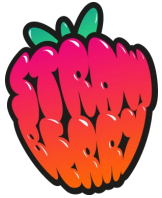
=

```
interface SequenceInterface {
    id: String
    bases: String
}


type BenchlingSequence implements SequenceInterface {
    id: String
    bases: String
}

type Plasmid implements SequenceInterface {
    id: String
    bases: String
    weight: Float
}
```

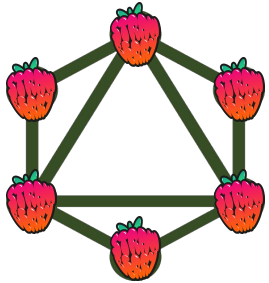
GraphQL SDL to Strawberry Dataclass



```
type Plasmid implements SequenceInterface {
  id: String
  bases: String
  weight: Float
}
```

```
annotations = {
    "__annotations__": {
        "id": str,
        "bases": str,
        "weight": float},
}
Plasmid = type("Plasmid", (object,), annotations)
```

Generating Types



```
type Plasmid implements SequenceInterface {
  id: String
  bases: String
  weight: Float
}
```

```
annotations = {
    "__annotations__": {
        "id": str,
        "bases": str,
        "weight": float},
}
Plasmid = type("Plasmid", (object,), annotations)
```

Built-in + Custom types → GraphQL SDL document → Strawberry dataclasses

Using forward refs

```
@strawberry.type
class Folder:
    id: str
    creator: ForwardRef
```

```
@strawberry.type
class User:
    id: str
```

```
@strawberry.type
class Folder:
    id: str
    creator: User
```

Replace links to other types with ForwardRef → Finish generating all types → Replace ForwardRef with actual types

Generate resolvers

```
def list_items_resolver(type_identifier: TypeIdentifier) -> StrawberryResolver:
    new *
    def resolver(info: Info) -> list[str]:
        return get_items(info, type_identifier)

    return StrawberryResolver(resolver)
```

Generate resolver functions for each type on the fly

Generate schema

```python
def generate_schema():
    fields = []
    for type_identifier in strawberry_graph:
        fields.append(strawberry.field(resolver=type_identifier))

    query = create_type( name: "Query", fields)
    return strawberry.Schema(query=query)
```

- Loop over graph of types
- Dynamically generate strawberry fields and resolvers
- Attach generated fields to query

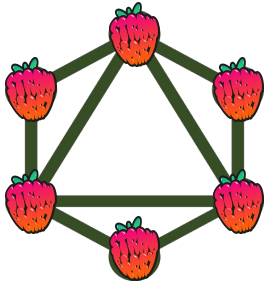
Query for Custom Types

```
query PlasmidQuery {
  stagingtx__PlasmidItems(id: {anyOf: ["seq_9RhtI332"]}) {
    edges {
      node {
        id
        bases
        weight
      }
    }
  }
}
```

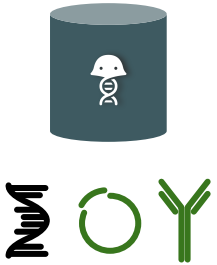
```
{
  "data": {
    "stagingtx__PlasmidItems": {
      "edges": [
        {
          "node": {
            "bases": "GATTAG",
            "id": "seq_9RhtI332",
            "weight": 10
          }
        }
      ]
    }
  }
}
```

Alternatives for Dynamic Generation

Built-in + Custom types



Strawberry dataclasses



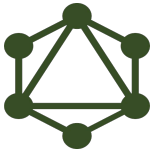
Built-in + Custom types

Ariadne spec

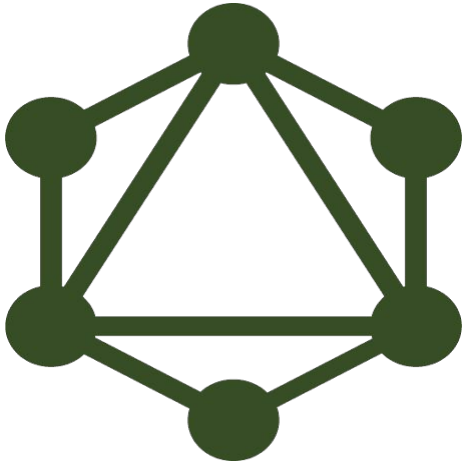
Technical Challenges

Graph Scale

100+ built-in types



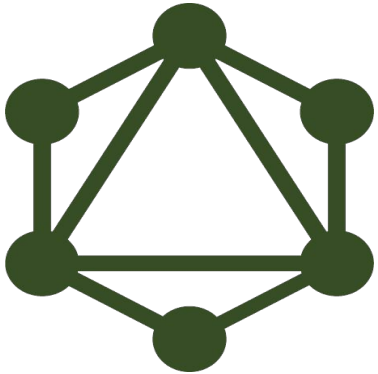
5000+ total types



40,000+ enum members

Much greater scale than typical GraphQL APIs.

Introspection Performance



> 60 seconds
in worst case

Requires building the entire graph to introspect the types.

Performance Breakdown

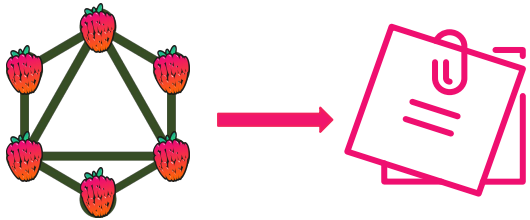
overall_construction	get_combined_sdl	strawberry_query_generator	generate_schema	introspection
64 sec	43 sec	5 sec	12 sec	4 sec

- 67% of time constructing combined GraphQL SDL document
- Remainder of time spent parsing SDL and constructing dataclasses

Mitigation Strategies Considered

Cache type data from database

Cache combined SDL document

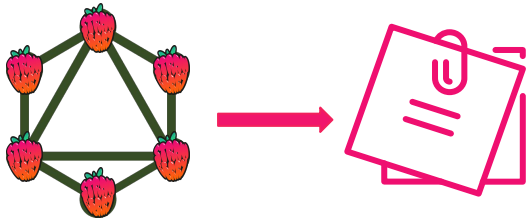


Cache SDL from Strawberry graph

Mitigation Strategies Considered

Cache type data from database

Cache combined SDL document



Cache SDL from Strawberry graph

Frontend Tooling for SDL

Registry schemas

1. Select registry schema

Entity schema

Select a schema...

Filters

2. Select columns

3. Add Results

4. Name table

Table name

apollo-client

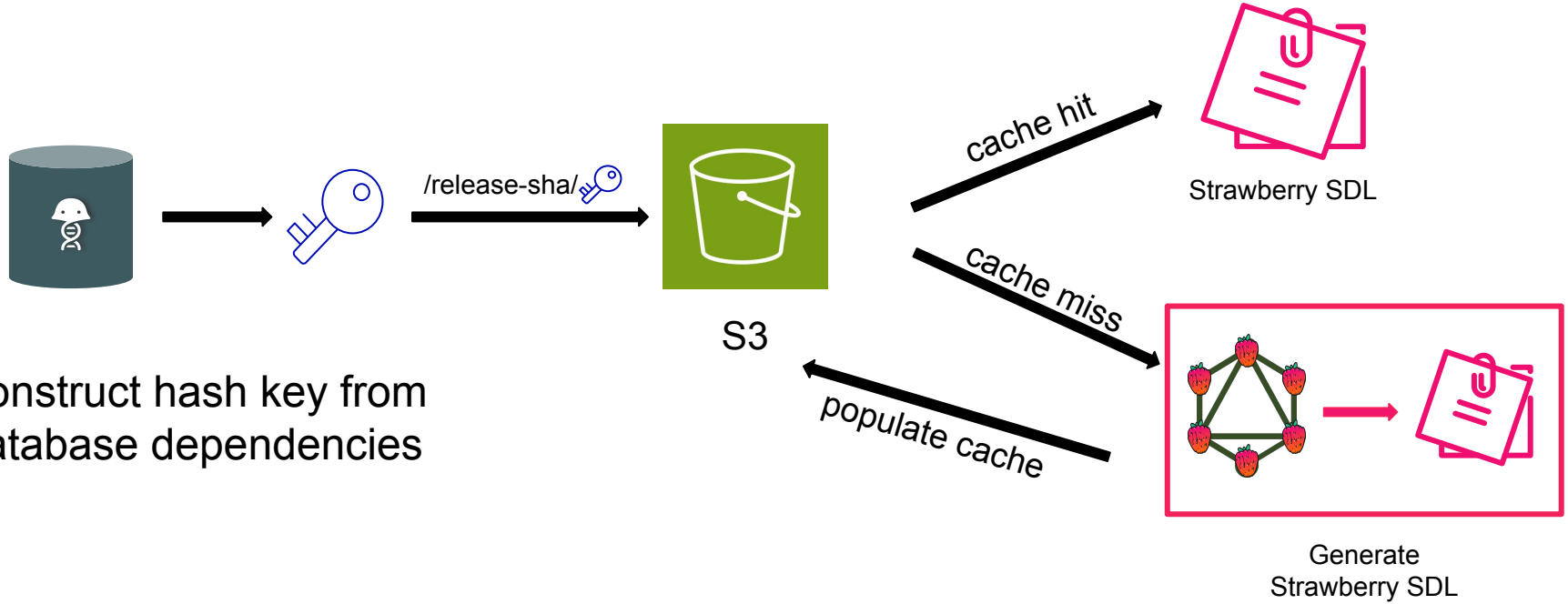
graphql-js

graphiql

relay

amplify-js

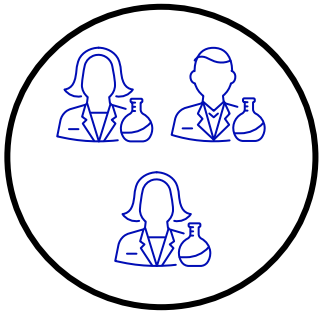
Caching Strawberry SDL



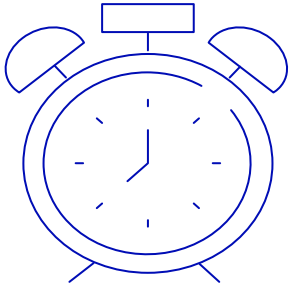
Construct hash key from database dependencies

Cache Repair

Redis Cache

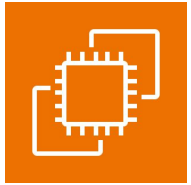


Active Users



Every 30 minutes

Async job



Refresh stale cache

Caching Results

- **49.59% Cache Hit Rate**
- **Introspection: 60 seconds → < 1 second**

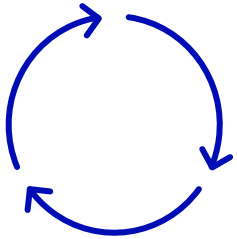
Performance Per Request

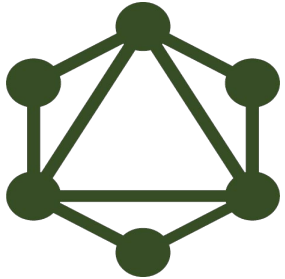
overall_construction	get_combined_sdl	strawberry_query_generator	generate_schema	introspection
64 sec	43 sec	5 sec	12 sec	4 sec

When serving requests, we need to have the most recent version of the graph for data integrity.

Performance Per Request

```
query PlasmidQuery {
  stagingtx__PlasmidItems(id: {anyOf: ["seq_9RhtI332"]}) {
    edges {
      node {
        id
        bases
        weight
      }
    }
  }
}
```

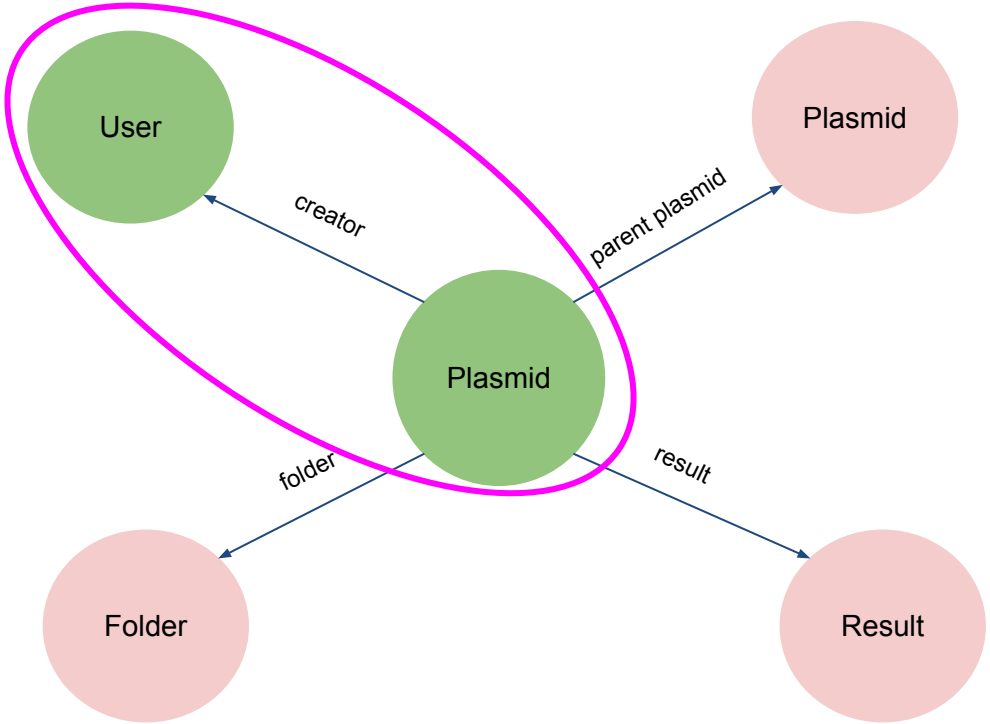




We want to avoid rebuilding the entire graph on every request.

Generating subgraph of types

```
query PlasmidQuery {
  stagingtx__PlasmidItems {
    edges {
      node {
        id
        creator {
          id
        }
      }
    }
  }
}
```



Subgraph Results

Serving requests: 60 seconds → < 1 second

Acknowledgements

Daniel Grossmann-Kavanagh
Stefan Young
Tim Jones
Mike Zhu
Daniel Vishwanath-Deutsch
Eli Levine
Jesse Coffey
Tara Lee
Eli Berkowitz
Ajay Subramanian

Q&A

P.S. We're hiring!
https://www.benchling.com/careers

GraphQLConf

hosted by  GraphQL Foundation