Building the **One-Graph** Together

@sjparsons - sjparsons.com/p/one-graph

Sam Parsons

Software engineer

❤️ GraphQL, React, JavaScript, Data Eng

👷 @Braintree (@PayPal)

sjparsons.com

@sjparsons

Let's build a one-graph

Your company should have one unified graph, instead of multiple graphs created by each team.

— *Principled GraphQL, "One Graph"*

photo: Vibrant Hot Air Balloons by Nicholas Raymond; CC BY 2.0)

So, why a one-graph?

1. Schema and resolvers are built once
2. Clients can access everything needed from one location

Challenges building a one-graph

Technical	**Schema**
servers, infrastructure, scaling, federation, dev-tooling	mapping business model, design, types

Tools for solving the **technical challenges**

- Apollo Federation
- A monolith org-wide repo
- Something bespoke that combines multiple independent 'packages' together

As a community we have made great progress on technical challenges, but that's only half the problem

photo: Mt Rainier 2017 by sofiadonnecke; public domain

Situation

1. Organization-wide GraphQL platform
2. Vision of migrating all existing GraphQL to the one-graph, including 'legacy' GraphQL ⇒ can't begin from scratch
3. Lots of teams and contributors ⇒ consensus building
4. Private graph for internal web and mobile clients
5. Mobile clients ⇒ non-additive changes need great care

Techniques

photo: Chisels by Dave Catchpole; CC BY 2.0

Technique 1

Build a **community**

It's all about the **people**

photo: Crowd on Sydney Harbour Bridge; public domain

Step 1. **Gather the community**

Find others who are interested or already involved and start building relationships

Step 2. **Meet regularly**

Allow contributions to the agenda

Make it open to anyone and easy to join (video probably)

Keep notes

Step 3. **Create a space**

Slack (or similar) for updates and async communication

Repo/wiki for documentation

Approach to community

- Weekly meetings on zoom (participants from all over the world)
- Slack channel for contributors to the graph
- Shared documentation repo
- Breakout meetings for specific areas of the graph
- Workshops for developing consensus around different topics
- Working groups to focus on solving different problems

Technique 2

Establish a **shared vision**

photo: Mountain Road by Axel Kristinsson; CC BY 2.0

We can drive towards the same goal only when we can understand and describe that goal in the same way.

— *Howard Dierking*

Why are we doing this?

(driven from leadership)

Strategic goals that will be achieved

Business needs that will be met

How are we going to do this?

(driven from the community)

Schema design guidelines

Working agreements

Principles for success

You're on the right path if your **schema design guidelines**...

- follow best-practices of the wider GraphQL community;
- are detailed enough to answer the questions your contributors;
- encourage schema that is evolvable over the long-term.

Some high-level guidelines

1. Prefer Object types over simpler structures
2. Create more specific mutations/queries rather than multipurpose ones
3. Don't use a generic name for a specific object
4. Define your schema from the business domain model
5. Delay defining interfaces until you understand domain thoroughly

More 📚 on GraphQL schema design

- Continuous Evolution of GraphQL Schemas @ Github by Marc-André Giroux
- GraphQL Schema Design: Building Evolvable Schemas by Marc-André Giroux
- Evolving the Graph by Jon Wong

Technique 3

Manage change appropriately

Hope is not a plan

photo: The Hussaini Hanging Bridge by AdnanKakazai; CC BY-SA 4.0

Tradeoffs

```
control      |<- free form ----- informal review ----- tightly controlled ->|
velocity     |<- high ---------------------------------------------- low ->|
participants |<- few ------------------------------------------- many ->|
```

Approach to change

```
|                                [X]                                |
|<- free form ----- informal review ----- tightly controlled ->|
```

Allow for collaboration and review without slowing velocity too much

- Semi-informal communication about changes and releases
- Ad-hoc review of schema changes
- Break-out groups for different areas of the graph

Ideas for managing change

1. Get analytics on the use of your graph
2. Use CI tools to post on PRs the changes to the resulting one-graph
3. Establish a deprecation policy
4. Establish a policy for breaking changes (*a la* GitHub)

1. Build a **community**
2. Establish a **shared vision**
3. **Manage change** appropriately

Thanks for listening. Questions?

Sam Parsons

@sjparsons - sjparsons.com/p/one-graph

photo: Stairwell at Louvre by Joe deSousa; public domain