GraphQL WG August 2020

Watch the replay: GraphQL Working Group Meetings on YouTube

Agenda

- Agree to Membership Agreement, Participation Guidelines and Code of Conduct (1m, Lee)
    - Specification Membership Agreement
    - Participation Guidelines
    - Code of Conduct
- Introduction of attendees (5m, Lee)
- Determine volunteers for note taking (1m, Lee)
- Review agenda (2m, Lee)
- Review previous meeting's action items (5m, Lee)
    - All action items
- Custom Scalar Update (5m, Andi)
- `@defer`/`@stream` (15m, Rob/Liliana)
    - RFC Update isFinal => hasNext
    - GraphQL-JS support AsyncIterable resolvers
    - GraphQL-JS support for `@defer`/`@stream`
    - Spec edit PR
    - express-graphql PR
    - GraphQL-over-HTTP RFC PR
- Field Coordinates RFC (15m, Mark)
    - RFC / Discussion Issue
- GraphQL over WebSocket refresh, existing issues and security implications (10m, Denis)
    - graphql-subscriptions-ws
- Adding generics to `DocumentNode` and `Source` to allow TypeScript type inference (10m, Dotan)
    - Discussion
    - Pull Request
- TypeScript migration plan progress and next steps (10m, Dotan)
    - graphql/graphql-js/issues/2104
    - graphql/graphql-js/pull/2609

    - graphql/graphql-js/pull/2616
    - graphql/graphql-js/pull/2634
- Tagged type RFC (30m, Benjie)
    - Spec RFC

Determine volunteers for note taking (1m, Lee)

- Benjie
- Stephen

Review previous meeting's action items (5m, Lee)

- All action items
- [**ACTION** - Lee] - File the actions for this (August 2020) meeting
- Custom Scalar editorial pass - Waiting on Lee who is waiting on legal paperwork.
- Introspection shortcuts - open and pending
- Input union RFC - Benjie, Vince or Lee to make good on this

Review agenda (2m, Lee)

- Tagged types moved to end (due to duration)

Custom Scalar Update (5m, Andi)

- New repo in graphql github org: https://github.com/graphql/graphql-scalars
- Started a pull request based on Gatsby, to generate the scalar URLs, add governance around the custom scalar specs. Not sure if this is the right technology solution, input welcome.
- [**ACTION** - anyone] - Join in with Andi on this repo
- Rikki - I'm keen to implement this in GraphiQL 1 and 2.

@defer/@stream (15m, Rob/Liliana)

- RFC Update isFinal => hasNext
- GraphQL-JS support AsyncIterable resolvers
- GraphQL-JS support for `@defer`/`@stream`
- Spec edit PR
- express-graphql PR
- GraphQL-over-HTTP RFC PR

- Rob - no major updates since the last meeting; see links above.
- Rob - roadblock regarding adding defer/stream to GraphQL.js because it breaks the types (allowing AsyncIterable(?) for resolvers in GraphQL.js), so doing it on a branch currently
- Lee - merged first PR (straightforward)
- Rob - opened up a PR to GraphQL-over-HTTP; more eyes desired
- [**ACTION** - GraphQL-over-HTTP WG] - Review https://github.com/graphql/graphql-over-http/pull/124
- Lee - potential solution is to add a different function for now so as to not break types, then merge the functions when they're ready.
- Ivan - we could use a branch and separate package (or npm tag). My issue is that execute function becomes too polymorphic - `data`, `Promise<data>` or `AsyncIterable<data>`. GraphQL as a peerDependency allows for people to choose their own version
- Lee - I'm wary of long-lived branches. I hope we can find a way to maintain one history. Separate packages could work. Good point regarding adding additional capabilities causing execute/etc to be more polymorphic. Had we know about this at the start, we might have required AsyncIterable from the start. Maybe introduce executeSync/executeAsync/executeStream such that the response type is not polymorphic. This might not be right, but is worth exploring. Then we get a single version of a package, and developers can choose which versions they want to support by choosing which functions they import. Long time hopefully the reasonable default is AsyncIterables. We want to ease maintenance and give developers choice. Eventually there'll be breaking changes, but we need to outline a path.
- Ivan - how about "experimental" folder inside the package. E.g. `import "graphql/experimental/execute"` or `/executeStream` etc. Normally when we mark something as experimental, people don't check the documentation and don't realise it's experimental.
- Lee - agree - needs to be in the function itself or the package - users must type "experimental"
- Andreas - encourage to make it clear that these are "non-spec" features to help other GraphQL implementations to know they don't have to support it yet.
- Ivan - we want to release a new version by the end of the summer. I agree with you - it's the reference implementation, so if we merge it we're promoting it.
- Lee - sounds like a good idea; GraphQL.js is both the reference implementation and how we test out new ideas, so using the folder structure to separate out the features sounds totally fine.
- Benjie - with ES6 modules `import * as GraphQL from "graphql"` would allow for libraries to "sniff" if `GraphQL.experimental_executeStream` is present; but try/catch can't be wrapped around `import ... from "graphql/experimental";` so easily for libraries (need to drop to commonJS or use async functions)

- Andi - We've set a high bar for merging features to master
- Lee - agree - whatever's on master should be the continual point.
- Lee - I've modelled GraphQL's releases on ECMA's releases - i.e. once per year officially, but implementations can pick them up as they go.
- Lee - anything in "draft" is safe to use in production because we have a high bar for that.
- Lee - TypeScript has a good model for this - balance between introducing features early but making sure things remain stable. We don't necessarily need a fine-grained terminology for this, we can just use "experimental" as a catch-all.
- Lee - in the rare case where something in draft changes in a significant way, we'd use this meeting to figure out how to mitigate the impact of this.
- Next steps: work through these PRs and make it easier for people to test them.
- Liliana - experimental flag or experimental directory or both?
- Lee - both? So long as they have to type "experimental" they'll have to realise - whether it's a flag passed, a function name, or a module path.

Field Coordinates RFC (15m, Mark)

- RFC / Discussion Issue
- Mark - Elevator pitch: want to write "User.email" to refer to the "email" field on the "User" type. RFC is to officially ordain this so there's no confusion.
- Mark - This is my first GraphQL spec related thing, so I want to know what next steps would be, etc.
- Mark - I've written a library ...
- Lee - please explain the mapping more
- Mark - https://github.com/sharkcore/extract-field-coordinates
- Function takes 2 parameters: SDL and query. Given a query, you can go from the query to a set of field coordinates. There's non-obvious features about pointing to the types that the fields belong to.
- Dan - I like the name "field coordinates". First question: delimiter. Second question: how does it appear in the spec? Probably a non-normative note would be non-controversial if the delimiter is non-controversial. I'm supportive of this.
- Lee - I suggest keeping the utility separate - what happens if you query an interface type - does it give you the interface type itself or all of the members?
- Lee - If we can refer to a field on a type, we'd might want to refer to any other thing in a schema. Is that within scope for this proposal?
    - Is the syntax for a field on an input object the same as fields on regular objects? Do we have a syntax for referring to arguments?
- Lee - Agree with Dan w.r.t non-normative notes
- Dan - do we already use this in the spec?
- Lee - not sure!

- Rikki - what if there's a GraphQL type literal notation; e.g. we could define that a type for a field was the same type as from another field. (`type User {nameEn: String, nameFr: User.nameEn}`)
- Mark - I haven't considered referencing other things for the first draft, but I'd like to think this is something we can iterator on in the RFC
- Rikki - Dotan et al are using something similar in GraphQL Mesh.
- Dotan - we're using the simplest form of that: type by name -> fields.
- Mark - it's mostly used for monitoring/alerting -> this field has been referenced x thousand times.
- Dotan - we also support wildcards, e.g. `Query.*` to get all fields, so it has the potential to get complicated.
- Dan - we have the same thing as Mark - we have a counter for `User.name` and when it's referenced we bump it. Seems like this is already a de facto standard, so adding it as a non-normative note as it is seems like a good thing. If we want to expand it to other things over time that'd be great too, but I think it's useful as is.
- Lee: bikeshedding punctuation, I think we should use the dot because it's what GraphiQL and other tooling already uses. And because it's non-normative people who use a different convention won't be non-conformant.
- Dan/Lee: input types, objects types, interfaces and unions all can use this same pattern.
- Lee - For RFC to move forward, need to be clear about what problem we're solving. Specify what this is for, and what this is not intended to be used for.
- [**ACTION** - Mark] - clarify the above
- Lee - error reports in GraphQL.js is where we're already doing this?
- Mark - part 1 is clear. Part 2 is how do we go from a query to a list of used fields so that we can increment the relevant counters for analytics.
- Lee - crisp up your problem statement as "you want to be able to uniquely identify some component of your schema (type, arg, field)" and explain why - a list of use cases; e.g. "you want a counter service that tracks fields/arguments"; "documentation: you want to hyperlink to part of the schema"; etc. Come up with a handful. Package it up as "unique identifiers of components of schemas"
- Rikki [chat] - great for cache keys as well!
- Andreas - "what is this thing called" is an important feature of this RFC.
- Lee - if we scope in arguments (which we should) is "field coordinates" the right name?
- Rikki [chat] - would this be called "dot notation"
-
-
-

GraphQL over WebSocket refresh, existing issues and security implications (10m, Denis)

- graphql-subscriptions-ws
- Denis - I want to discuss some things about the existing implementation/protocol
- How many custom implementations are we aware of. Realtime seems important in the development of apps.
- Rikki - I've worked on apps where most features were in subscriptions
- Denis - there's so many people using this, but no-one is addressing the issues/PRs.
- Denis - I've listed out the various issues that are really critical, such as security issues such as bypassing the onConnect event. I want to tackle the security issues first.
- Denis - this protocol is not just about subscriptions, but queries and mutations too.
- Benjie - queries and mutations over websockets can be beneficial due to reduced overhead vs HTTP1.1/ query batching; but HTTP2/etc can solve this better. There are projects that only do subscriptions, and some that do all over it.
- Robert - I'm not sure supporting queries and mutations over subscriptions is the best thing to do over time, HTTP3 have head of line blocking, etc which make websocket a poor choice.
- Lee - Facebook maintains stateless requests over HTTP whilst streaming data from MQTT/etc.
- Robert - they use MQTT-over-websockets
- Robert - scaling a realtime gateway has very different needs to scaling a stateless gateway. This is one of the reasons that Facebook don't merge the two. I think (and I may be over-reaching) people wonder why they have to do HTTP for query/mutation; why can't I do everything over websockets - isn't it more efficient? I'd like to see evidence that this is causing an issue somewhere.
- Robert - there's potentially race conditions when you do things over two different transports, so this could be a compelling reason
- Stephen - GraphQL spec is agnostic regarding transport because there's lots of different use cases. For a lot of use-cases scaling isn’t a concern and a simpler implementation is sufficient. Would this be in scope for the graphql-over-http working group?
- Robert - we should definitely bring this under the umbrella of the foundation.
- Dotan - there is no maintainer currently. It's not highly maintained, it does work, and personally I wouldn't define it as "production ready" because of security issues and some things are not handled correctly (such as authentication over websockets). Many people are using it as GraphQL transport rather than GraphQL subscriptions transport. Needs maintenance, updating dependencies, making the protocol more robust.
- Denis - I think this should be discussed over the GraphQL-over-HTTP WG; security is the biggest concern. It can swallow errors or do weird things, causing all sorts of issues.

Some of the problems are hard to solve due to not being able to set custom headers; URLs are exposed so putting JWT token in URL isn't safe. We should kick clients off after a timeout.

- Denis - do you know how many people are conforming to the current protocol? How much wiggle room is there? I'd like to re-write it from scratch with full RFCs.
- Lee - I think you should do that. The [GraphQL] spec itself says very little about the protocol, it only talks about the abstract mechanics.
- Denis - to be clear I'm talking about the Apollo websockets spec
- Dotan - the protocol was written when we wrote the library, so the subscriptions client/server are the only parts aware of this protocol. From my point of view feel free to rewrite it from scratch.
- Rikki [chat] - some of the most commonly used thirdy party GraphQL tooling is some of least maintained. the more of us that care, the better
- Dotan [chat] - Related: https://github.com/apollographql/subscriptions-transport-ws/issues/777
- Rikki [chat] - i think security concerns are a great context in which a protocol related spec becomes important
- Lee - you should run with this; I like your example for Relay in the README - we wrote Relay in this way to make it flexible. Facebook's websocket implementation is completely different from Apollo's.
- Lee - one thing to consider: pitch this as an improvement / something to use instead of the existing spec -> give a migration path. Do you change the server first? Client first? Do you set up different URLs? if this is at an impasse, then that's still okay.
- Denis - I want to keep the server API mostly the same
- Robert - think of this as 3 separate layers: transport / protocol / graphql.
    - connection open/closed/suspended, etc should be separated from protocol itself
    - allows for protocol to be used over different transports (even non-bidirectional ones)
    - if websocket server is under too much load, we could mark that the stream is suspended
    - can we extract the protocol and write a spec around the protocol itself only, using examples of subscriptions and live queries (and less so defer and stream)
    - tell the client they should expect a stream of results to follow, but allow the server to pause the stream, handle errors, deliver healthy payloads.
- Denis - next steps: writing the protocol first, then the migration.
- Benjie - for people using other websocket libraries - what are you using?
- Stephen - Netflix - we use server-sent events (due to layer 7 proxy); there's a JS client library for that, but we're using an internal fork of it (which we can open up) that has some protocol changes. Websockets: We also support WS, but mainly for dev usage. We're matching the OSS protocol for GraphiQL support OOTB.

- Lee - Facebook are using MQTT over websockets, for historical reasons. Does direct MQTT to mobile devices.
- Stephen - does facebook still use long polling?
- Lee - sometimes; they do both - depending on the backing service. Chat/messaging uses async pushy. Other things where a polling model is more efficient.
- Rob - SSE/websockets -> we don't want the details to leak through if we can avoid it.
- Rikki [chat] - also there are some GraphiQL users who use serial communications for subscriptions

Adding generics to DocumentNode and Source to allow TypeScript type inference (10m, Dotan)

- Discussion
- Pull Request
- [walk-through of PR]
- Rikki - this seems amazing! I couldn't figure out how to use it with queries that were not defined in `.graphql` files.
- Dotan - if you're using Babel, there's a plugin that [...]
- Dotan - the ideal use case is with .graphql files for graphql-code-gen. Manual typing can work too. Apollo have merged a PR for it to the next version; this might be a good way to make GraphQL and TypeScript work well together.
- Rikki - I've been spending a lot of time adding embedded typescript support in various places. So long as we have a tagged template/pattern it makes it easy to add language features/completion/etc.
- Dotan - if operations are inlined, you normally use something e.g. graphql-tag to parse it.
- Dotan - this shouldn't be any breaking changes for anyone
- Lee - looks great; so long as there's no breaking changes and the DX is at worst the same, then there's no down-side. You don't have to use these, it's just yet another way to use the types. Additional complexity is a potential downside, but the PR seems small so this isn't much of a concern. It's up to you and Ivan to progress it.
- Ivan - we need a separate call to discuss graphql-js because not everyone on WG is using JS/TS.
- Ivan - how TypeScript specific should GraphQL.js be? It's not necessarily portable to other languages. People open other PRs such as inferring types for resolvers into the schema, to create something like GraphQL-Nexus type inference; I'm worried about becoming too TypeScript specific. Question: why cannot it be a separate function called typescriptExecute?
- Dotan - it's the same function, the signature's the same.
- Ivan - if we merge this, people are going to want to merge other TypeScript-specific functionality. My point was always that types should reflect runtime behaviour. Since we

cannot guarantee that the return value will match (it's ensured by codegen not GraphQL JS) so it's basically glorified type conversion. I'm on the fence about going too much TypeScript. I'm open to add hooks
- Lee - typescript generation should not live in GraphQL-js, so that's a boundary. We should enable changes to TypeScript that enable capabilities (like hooks) but not add things that narrow capabilities.
- Dotan - the major benefit of this feature is not GraphQL-js, it's all the libraries that consume GraphQL js because they get free type inference for hooks, components, etc.
-
-
-

TypeScript migration plan progress and next steps (10m, Dotan)

- graphql/graphql-js/issues/2104
- graphql/graphql-js/pull/2609
- graphql/graphql-js/pull/2616
- graphql/graphql-js/pull/2634
- Dotan - maybe we should rethink the next steps and break this down into smaller tasks. I started looking at migrating the tests and the linting.
- Ivan - we made the decision to move to TypeScript a year ago, everyone was involved (Apollo, etc). Core issue here is that I am a bottleneck here. We have a very high bar for quality, so I need to review every code line, which is a lot of effort especially for mass-conversion. One a technical side I'm trying to figure out a way to separate out syntax changes from meaningful changes. Core issue is the bus factor; this month I had knee surgery so I was in hospital so not communicating/merging. This blocked a lot of people - stream and defer/etc. I'm trying to onboard new maintainers. Daniel helps a lot; he wants to become a maintainer. Also have a summer of code student who can help.
- Lee: lets put a plan together for increasing bus factor. Two actions:
    - Start a Graphql.js specific call - focussed on strategy, not specific code
    - List our contributor to GraphQL.js; specify “owners”. Who do we want to give review, but not yet merge capabilities? Keep it to a tight core set.
- Ivan - Don’t feel like we need to formalize it too much. Switching from 1 to 2 maintainers is a bigger shift than from 2 to 3.
- Lee - set up the call first, and go from there. [ACTION, Lee, Ivan]

Tagged type RFC (30m, Benjie)

- Spec RFC

- Benjie - As discussed last WG, there is progress on input unions; tagged type is a leading option. Draft RFC for that is linked here.
    - Tagged type is the first type that is not a wrapper that can be valid for both input and output.
    - If all members were scalars, it would be valid for both input and output.
    - Feature summary:
        - New graphql type
        - Has fields
        - Associated types like other type fields
        - Fields don't accept arguments
        - Currently don't accept directives, but this could change
        - For querying - use a selection set and get exactly one. Not null, just "skipped"
        - For input, same behavior
- Should this actually be a separate type? Yes. It allows it to evolve separately. Doesn't have much in common with other fields (e.g. no arguments)
- How does this behave with regards to nullability and nonnull constraint? This is still an open question.
- Divergence from current gql spec - previously `__typename` value was always a concrete type. Now it can also be tagged type.
- Could look into adding introspection for __Type
- Another open question: Should we allow field aliases? Can you refer to the same field more than once? Current RFC puts no restrictions, same as normal selection set
- Lee - Should you be able to use fragment spread within tagged type?
    - Benjie - Yes. It allows to to keep in one location all possibilities
- Andreas - Clarification - you can only return a single field, correct? (Yes) Breaking changes.. The __typename is a big change. Not necessarily a blocker, but something to note.
- Lee - Also, anywhere to see {} typeset, you can ask for __typename. Would this be an exception to that?
- Benjie - I think RFC is the right approach. Many pieces of tooling don't know the schema. They might add __typename automatically. Would be a big lift to require knowing shema. What might break?
- Andreas - Maybe not technically "breaking", but it's a big change. There is a concept of normalized queries. Executed in terms of object. You can expand all fragments and unions. This changes the way to analyse the query overall. Not opposed to fundamental aspects of this proposal, just noting the potential implications
- Benjie - currently calling these fields "members". We already have "fields" in other cases. Could cause issues if we want to evolve inputs. More branches for tooling to handle, but the trade-offs are worth it to enable future expansion of graphql.

- Andreas - Looking at examples of single pet vs list of pets. Is this similar to how a fragment would work? (Yes)
- Discussion on some interesting edge cases. Likely some interesting debate on some of these.
- Lee - could be some confusion if multiple variables, one field exists but is null.
- Benjie - should all members be non-nullable?
- Stephen - would that impact evolvability of a tagged type, for example when deprecating and removing members?
- Benjie - if you deprecate a member, then you should never resolve to that value, but you should always accept it as an input. Since only one key is present, I don't think nullability ties in to deprecated in the same way as for object types.
- The only way to know what a tagged type would be is to request **all** members. Otherwise, you might not get a result. Could potentially add an introspection field to just get the field name?
- Lee - if you won't request fox, then my code doesn't have "fox". It's an else branch, and could be anything.
- Stephen - this is similar to the oneof feature in protocol buffers. Proto includes a special method for checking which value (if any) in the oneof is set.
- Benedikt - if you have object type as the member type, you might have the same type twice.
- Benjie - if a tagged contains both input and output types it would be unusable. THere's already validation in RFC for this.
- Lee - part of what we're doing to charting rules for how these would operate. Retaining the option type for later.
- Benjie - tentatively calling option type "struct". Structs could potentially be a solution for this, but there are issues. We'd lose flexibility of evolving schema over time.
- Andreas - why would I use unions if I have tags?
- Benjie - Unions have some advantages. Ex: fragment spreads. If a union member implements an interface, you spread over the interface. You can't spread interface over tagged type.
- Rikki - What if input types could use interfaces, and share them with types, this indirectly solves the problem kinda?
- Benjie - Interfaces use the output type definition of fields, inputs don't support arguments, etc.
- Benjie - using separate input/output types would add complexity
- Benedikt - it can be useful to have same types in input and output
- Lee - Any blockers to merging?
- Benjie - Some open questions, but no blockers. This could be merged as-is.
- Andreas - This solves problems for inputs, why is this extended for outputs as well? What new capabilities does it add?

- Benjie - Excellent question.
    - Easier for beginners to query
    - Symmetry between input and output
    - Ability to do unions of scalars, lists, enums, etc in addition to simple types
- Lee next steps: Start an implementation
- Benjie: Planning to let it brew for a month in the community

Any other business

- [Benjie] How to get recordings?
- Lee - Zoom provides them after a few weeks as raw video files. Eventually we want to get them on youtube.
- Ivan - The foundation guys have figured out how to publish on youtube!
- Lee - let's add youtube link to notes after the fact