Twitter Thread by Sam Newman



Sam Newman
@samnewman

So a few people have asked why I have this snarky response. What is my problem with this service? Well, to be clear, it's not an issue with GraphQL, it's an issue with direct coupling with underlying datasources #thread

> Great to see AWS providing direct data coupling as a service. https://t.co/TEk8ybPTro
>
> — Sam Newman (@samnewman) January 5, 2021

The service as advertised makes it simple to map a GraphQL definition against a database. Now, what's the problem with this? Well, the devil here is in the detail. But fundamentally it comes down to how important information hiding is to you.

Information hiding is the concept whereby you expose as little information as possible to external parties. Anything you expose over a boundary becomes part of the contract between provider and consumer.

Anything I hide inside a boundary can be changed freely. Anything I expose must be maintained if I want to maintain backwards compatibility.

Why is backwards compatibility important? If I break compatibility with external consumers of an interface, then such a change can lead to the need to lock step releases of consumer and producer, or even worse accidental breakages in prod.

The concept of information hiding goes back a long way - it was originally defined by David Parnas and shared in his paper "On the criteria to be used in decomposing systems into modules
David Lorge. Parnas
Carnegie Mellon University" https://t.co/qjhfrXkHtI

Encapsulation in OO when used well is an example of information hiding in action.

Now, in a distributed system, being able to make changes safely and with confidence is important. Having a clear understanding about what changes can be made safely, without causing compatibility issues with clients is vital if you want independent deployability.

Now, this is why when we talk about Microservices we place so much importance on not allowing external parties to directly access a microservice's internal database - no information hiding is possible.

Aren’t we hiding information by wrapping a database with a GraphQL endpoint? Well, the devil is in the detail. A common pattern I’ve seen is a 1:1 mapping - think putting getters and setters on an object to access private fields. That’s not information hiding.

Now with something like a GraphQL projection, if you’re careful, you can provide a degree of information hiding - you can decide not to expose some parts of the underlying data model.

The golden rule here is never expose anything until someone actually needs it. Think outside in - what exactly do external parties need? This allows you to be selective in what you expose. Too many people think inside out, and expose everything in case it *might* be needed.

Now, let’s assume you’ve done a great job in being selective in terms of what you allow to be accessed. Well, then your GraphQL endpoint is just a wrapper on a database. Where does the logic live that defines what the allowable state transitions are? In the client.

What happens if you have more than one client? You have to either duplicate that logic or else share that logic (eg shared library). The concern here is that you might have different views about what an acceptable transition is in different clients.

What happens if different clients don’t agree about what acceptable transitions are? Fun stuff! By fun I mean painful, expensive things.

So, my concern with offerings that allow you to wrap database in GraphQL endpoints is partly it makes information hiding more difficult, and that with multiple clients makes it difficult to manage state in a consistent, correct manager.

I’d have the same concerns about other approaches like ODATA. It’s not a GraphQL issue specifically.

Sidebar: the (brief) article linked compares GraphQL and REST, which is a common thing to do but misses a HUGE amount of what REST is about. Arguably of course, if you just think of REST as being JSON over HTTP, then although that’s not correct, I understand the confusion.

So, by all means make use of GraphQL, but be aware of the challenges that can occur when you just map database tables to GraphQL endpoints.