LOG8430: Architecture logicielle et conception avancée

Microservices, REST and GraphQL

Automne 2017

Fabio Petrillo
Chargé de Cours

Page 2

http://nealford.com

Page 3

Page 4

Page 5

5

Page 6

Microservices

Microservices are small,

***autonomous** services that*

*work **together**.*

Building Microservices, Newman, Sam.
O'Reilly Media, Incorporated, 2015.

6

Page 7

Microservices Architecture

Page 8

Microservices

Page 9

Microservices have emerged from...

- Domain-driven design
- Infrastructure automation
- Continuous delivery
- Small autonomous teams
- Systems at scale

Page 10

Conway's Law

*"Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a **copy** of the organization's communication **structure**."*

Melvin Conway's paper How Do Committees Invent, published in Datamation magazine in April 1968

Page 11

Technology Heterogeneity

Page 12

Scaling

Page 13

The bounded context - shared and hidden models

13

Page 14

Deployment – Virtualization and lightweight containers

14

Page 15

15

Page 16

Representational State Transfer (REST)

- REST is an architectural style
- It is not a standard

- Client/Server relation
- Commonly associated with HTTP

https://spring.io/understanding/REST

Page 17

Principles of REST

- Resources expose easily understood directory structure URIs.
- Representations transfer JSON or XML to represent data objects and attributes.
- Messages use HTTP methods explicitly (for example, GET, POST, PUT, and DELETE).
- Stateless interactions do not store the client context on the server between requests.
- Web service APIs that adhere to the REST architectural constraints are called

RESTful APIs.

https://spring.io/understanding/REST

Page 18

Building a RESTful Web Service

- Tutorial : Spring Boot Framework
  - https://spring.io/guides/gs/rest-service/
- HTTP GET
  - $ curl http://localhost:8080/greeting?name=User
- Respond with a JSON
  - {"id":1,"content":"Hello, User!"}

https://spring.io/understanding/REST

Page 19

RESTful API Design Tools

- Swagger
- RAML
- Spring REST Docs
- Review of tools on
  - https://opencredo.com/rest-api-tooling-review/

Page 20

Page 21

21

Page 22

22

Page 23

What is GraphQL?

- GraphQL is a query language (using JSON) for APIs and a runtime for fulfilling those queries with your existing data.
- It was originally written and open sourced by Facebook, and can be seen as an alternative to REST.
- A GraphQL operation can be either a **query** (read operation), or a **mutation** (write operation).

Query Result

Page 24

What is GraphQL?

- GraphQL reduces network hops by allowing you to retrieve all the data you

need in a single query.

- GraphQL is WYSIWYG model, make client consumption code less error prone.
- RESTful HTTP leverages more consistency and predictability by making use of status codes and HTTP verbs.
- Hypermedia makes RESTful consumers easy to implement by allowing them to "discover" resource relations whilst using the API.
- HTTP already implements caching, whereas GraphQL does not.
- GraphQL is useful in that it provides a schema for consumers, but be warned that interface description is not API documentation.

Page 25

RESTful APIs versus GraphQL APIs

- In RESTful APIs, the language we use for the **request** is **different** than the language we use for the **response**.
- There are **no standards or agreements** about what request and response HTTP codes mean and implementers use different specifications, which makes working with different APIs unpredictable.
- To consume RESTful APIs, we use a URL to read from or write to a **single**

resource, such as a product, a person, a post, or a comment. If we need to work with **multiple resources** such as a list of posts with a list of comments, we need to use **multiple endpoints**.

Page 26

RESTful APIs versus GraphQL APIs

- GraphQL is one other alternative that **is attempting** to solve most of these issues.
- GraphQL RFC specification has been created. It's managed by Facebook, but it's open source on GitHub and anyone can contribute to it.
- GraphQL is protocol-agnostic and does not depend on anything HTTP.
- The language used for a GraphQL request is directly related to the language used for the response (JSON).
- Let's assume we have a single GraphQL endpoint exposed over HTTP as /graphql.
  - /graphql?query={...}

Page 27

RESTful APIs versus GraphQL APIs

- GraphQL reduces network hops by allowing you to retrieve all the data you need in a single query.
- GraphQL is WYSIWYG model, make client consumption code less error prone.
- RESTful HTTP leverages more consistency and predictability by making use of status codes and HTTP verbs.
- Hypermedia makes RESTful consumers easy to implement by allowing them to "discover" resource relations whilst using the API.
- HTTP already implements caching, whereas GraphQL does not.
- GraphQL is useful in that it provides a schema for consumers, but be warned that interface description is not API documentation.

27

Page 28

GraphQL - Issues

- GraphQL makes easier is resource exhaustion attacks (AKA Denial of Service attacks).
- Authentication and authorization are other concerns that we need to think about when working with GraphQL.
- GraphQL makes a bit more challenging is client data caching. RESTful APIs are easier to cache because of their dictionary nature: this location gives that data.
- Main issues -> N+1 SQL queries

Page 29

GraphQL - some resources

- http://graphql.org/
- https://dev-blog.apollodata.com/tutorial-building-a-graphql-server-cddaa023c035
- https://launchpad.graphql.com/new
- https://www.reindex.io/blog/building-a-graphql-server-with-node-js-and-sql/
- https://www.infoq.com/news/2017/07/graphql-vs-rest
- https://edgecoders.com/graphql-deep-dive-the-cost-of-flexibility-ee50f131a83d

Page 30

Microservices - No Silver Bullet

- associated complexities of **distributed systems**
- networks aren’t **reliable**

If you can't build a monolith, what makes you think microservices are the answer?

30

Page 31

31