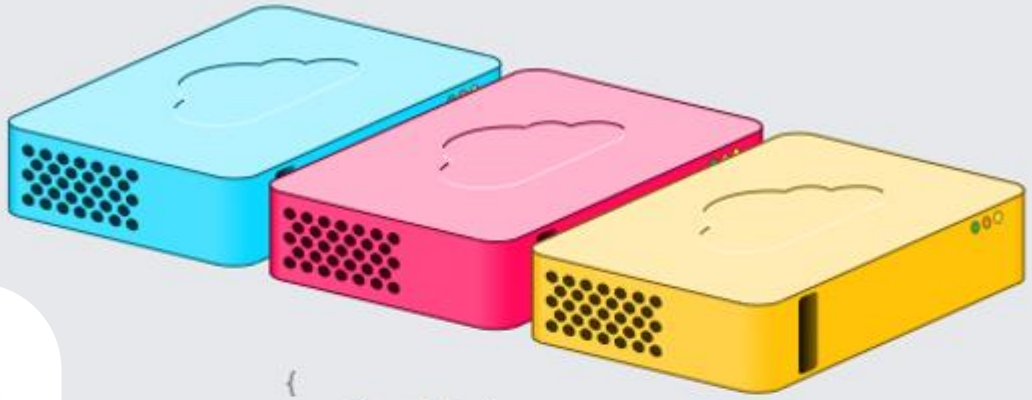
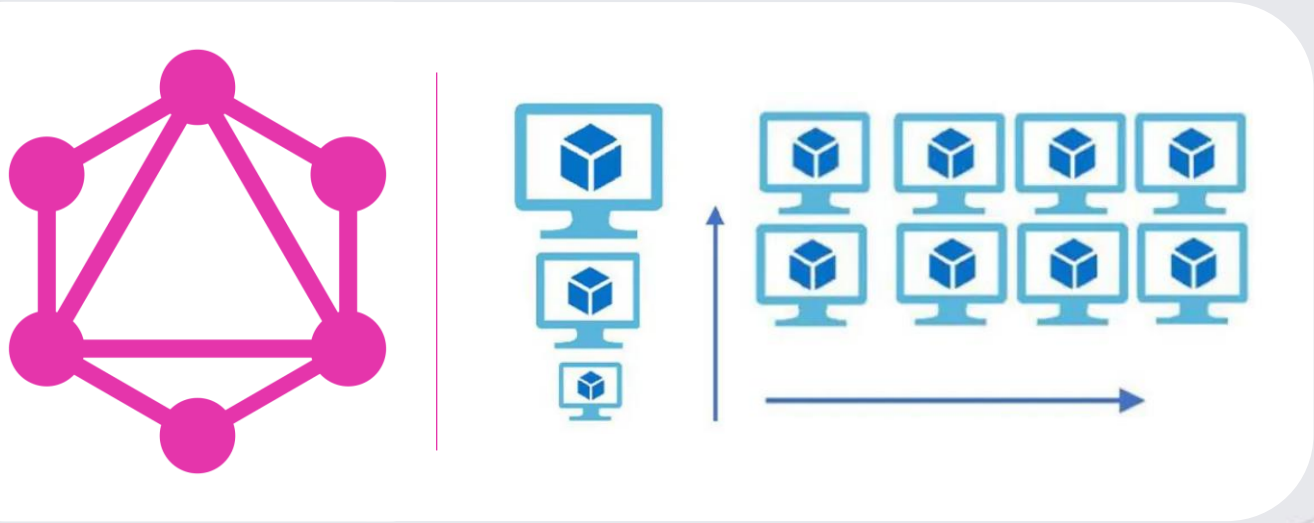
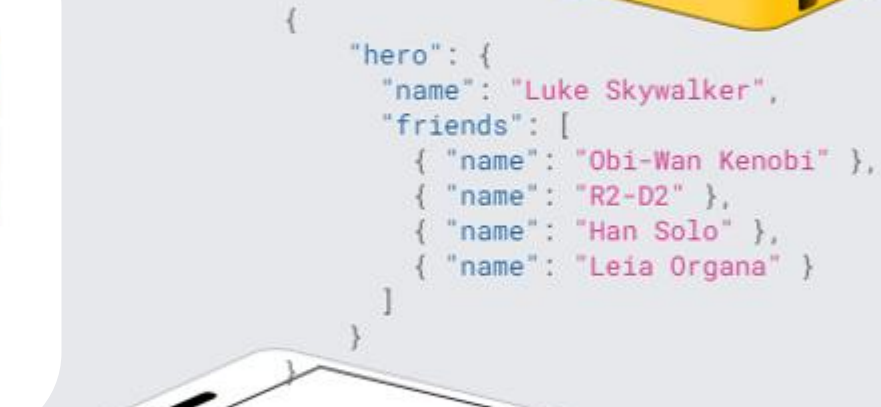
ELCA
We make it work.

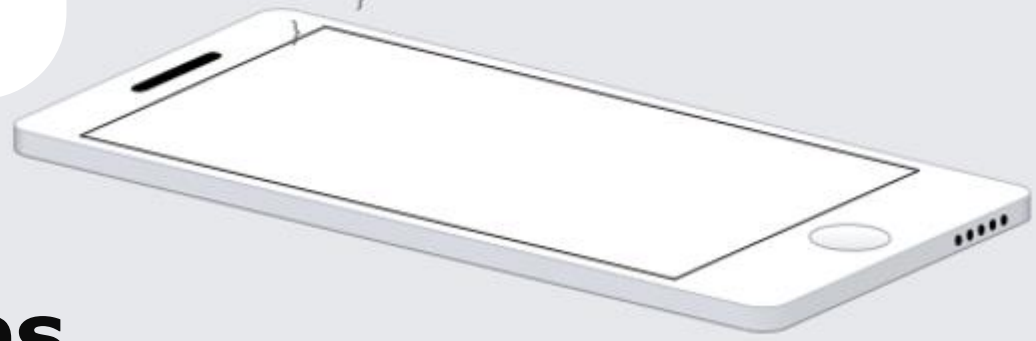
Granada – 08.Feb.2024

ELCA
We make it work.









GraphQL At Scale

Presented by Miguel Rosales

About Me

Miguel Ángel Rosales Navarro

- Born and living on a little village at center-west of Granada, called **Purullena** **(Pure & Full** water**).**
- Telecommunications Engineer & Computer Science Engineer.
- Head of ELCA Spain Engineering Delivery Unit.
- Technical Expert & Senior Project Manager.
- Chispi’s father.

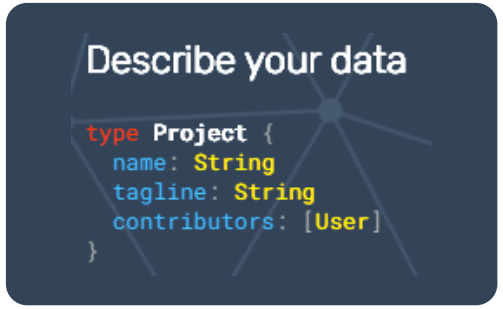
Agenda

1. **What Is GraphQL**
2. When to use GraphQL vs REST
3. Why Scale GraphQL
4. Why Scale GraphQL Horizontally?
   - 4.1. Horizontal GraphQL Techniques
   - 4.2. Horizontal GraphQL Best Practices
5. Why Scale GraphQL Veritcally?
   - 5.1. Vertical GraphQL Techniques
   - 5.2. Vertical GraphQL Best Practices
6. Other Best Practices
7. Real World Example
8. Conclusions

1. What Is GraphQL

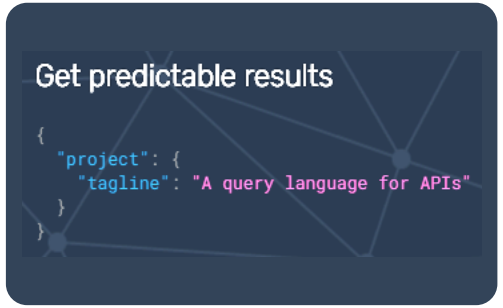
GraphQL is a query language for APIs that was developed by Facebook in 2012 to address some of the limitations of traditional APIs. Some of its key features are:

- Allows developers to request all the data they need in a single query.
- Enables developers to **define a schema that describes the data** available in their API. This makes it easier to understand and work with the data, especially when **dealing with complex relationships** between different entities.
- Allows developers to be able to build APIs that are better suited to **modern** web applications and **can be scaled more easily**.



— You can describe the data you want to create, use, modify, etc … in your application.

— You will be able to ask what you want to recover from the data you have already described.



— It ensures that the data received in the query will always have the same structure.

1. What Is GraphQL

— **The GraphQL operations** are simple string that a server can parse and respond to with data in a specific format.

— **A GraphQL query:** is used to read or **fetch values:**

```
query HeroNameAndFriends {
  hero {
    name
    friends {
      name
    }
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
```

— **ELCA** projects examples:

COMPASSANA

SwissLife

OXYNADE
SELF-SERVICE
SAAS TICKETING

17 96
LOMBARD ODIER
LOMBARD ODIER DARIER HENTSCH

— **A GraphQL mutation:** is used to **write** or post values:

```
mutation CreateReviewForEpisode($ep: Episode
  createReview(episode: $ep, review: $review
    stars
    commentary
  }
}
```

VARIABLES

```
{
  "ep": "JEDI",
  "review": {
    "stars": 5,
    "commentary": "This is a great movie!"
  }
}
```

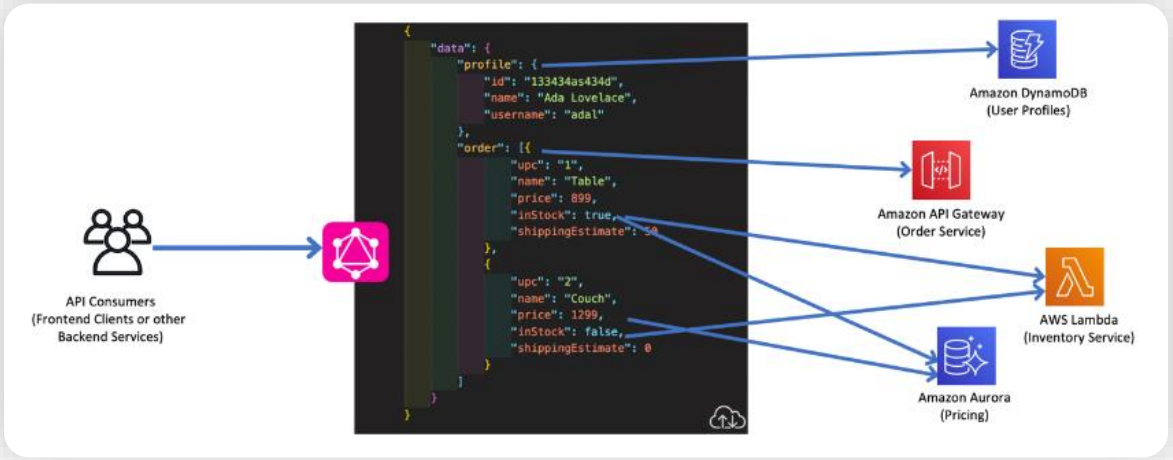
```
{
  "data": {
    "createReview": {
      "stars": 5,
      "commentary": "This is a great movie!"
    }
  }
}
```

Agenda

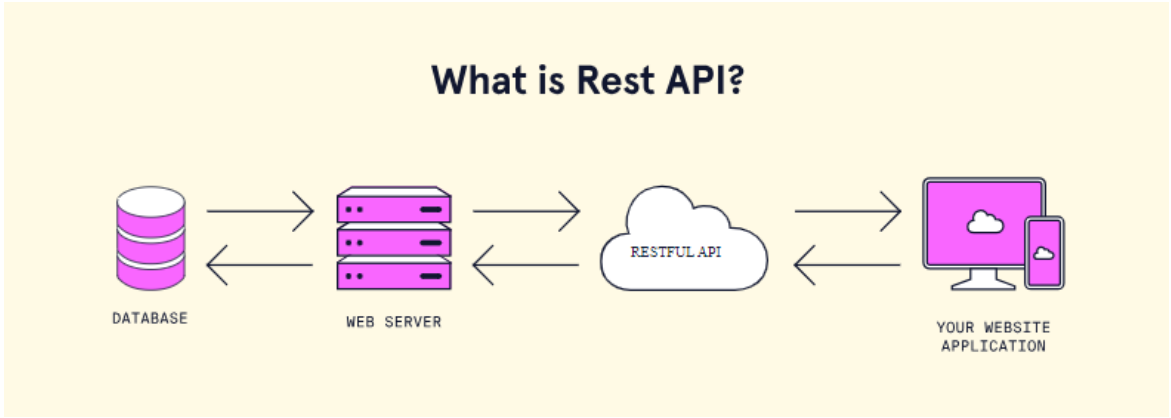
1. What Is GraphQL
2. **When to use GraphQL vs REST**
3. Why Scale GraphQL
4. Why Scale GraphQL Horizontally?
   - 4.1. Horizontal GraphQL Techniques
   - 4.2. Horizontal GraphQL Best Practices
5. Why Scale GraphQL Veritcally?
   - 5.1. Vertical GraphQL Techniques
   - 5.2. Vertical GraphQL Best Practices
6. Other Best Practices
7. Real World Example
8. Conclusions

2. When to use GraphQL vs. REST

- **GraphQL** is a better choice on these considerations:
  - **Limited bandwidth**, so need to minimize the number of requests and responses.
  - **Multiple data sources**, to combine them at one endpoint.
  - Client **requests that vary significantly**, and expectation of very different responses.



- **REST** is a better choice on these considerations:
  - For **smaller applications** with less complex data.
  - Data and operations that all clients **use similarly**.
  - **No** requirements for **complex data** querying.



3. Why Scale GraphQL?

- **Scaling GraphQL** can bring a multitude of benefits to your system. By reducing the number of requests needed to fetch data or increasing the server/s capacity, you can **improve the performance** and user experience of your application.
- Additionally, scaling GraphQL can help with versioning. With other APIs, versioning can become difficult as changes to the API can break client applications. However, with GraphQL, **clients can specify exactly what data** they need, **eliminating the need for versioning** altogether.

Some real-world examples:

Facebook (2012) reported that after implementing GraphQL, they saw a **50% reduction in the number of API requests** made by their mobile apps.

Netflix (2015) were able to reduce the number of API requests by up to **60%**, resulting in faster load times and improved user experience.

Airbnb (2017) were able to **reduce the size of their API responses by up to 90%,** resulting in faster load times and reduced data usage for their users. They also saw a significant **decrease in server costs** due to the optimized queries provided by GraphQL.

Scaling Challenges with GraphQL

— As GraphQL applications scale, they can face several challenges, especially as the complexity of your application grows:

- Deal with **large number of concurrent queries** and mutations that can put a strain on the server's resources.
- Ensure that the **schema design can handle the increased traffic** without compromising performance.
- **Careful planning and implementation** to ensure that the application **can continue to function effectively** at scale.

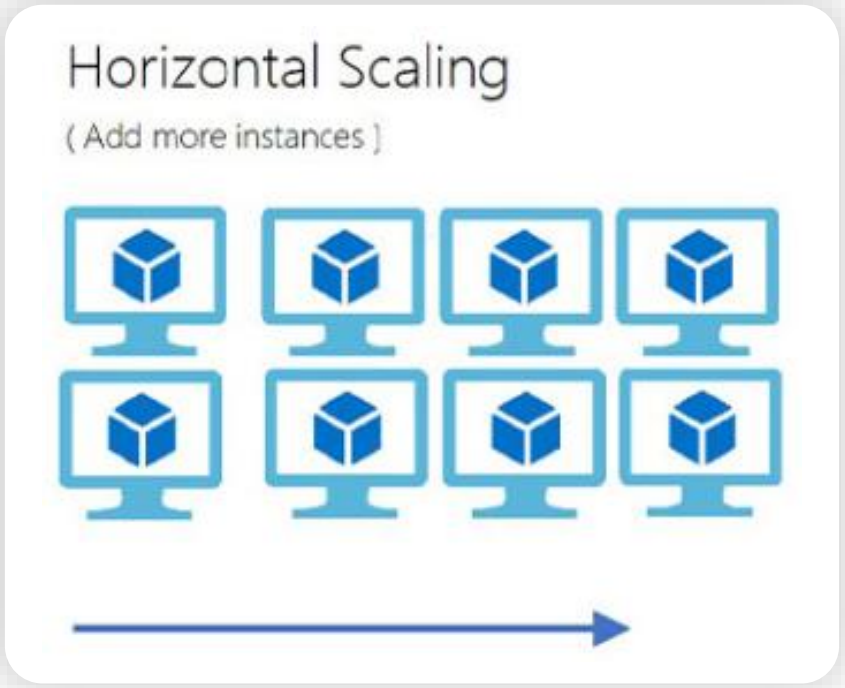
— Fortunately, there are several techniques that can be used to address these challenges in both horizontal and vertical scaling.

Agenda

1. What Is GraphQL
2. When to use GraphQL vs REST
3. Why Scale GraphQL
4. **Why Scale GraphQL Horizontally?**
   - 4.1. Horizontal GraphQL Techniques
   - 4.2. Horizontal GraphQL Best Practices
5. Why Scale GraphQL Veritcally?
   - 5.1. Vertical GraphQL Techniques
   - 5.2. Vertical GraphQL Best Practices
6. Other Best Practices
7. Real World Example
8. Conclusions

4. Why Scale GraphQL Horizontally?

- **Horizontal scaling** refers to increasing the capacity of processing requests **by adding multiple servers**.



- Scaling GraphQL horizontally can bring many benefits to your projects:
  - **Increase the performance** of your GraphQL API.
  - **Handle high traffic** easily.
  - **High availability**, isolation against single failures.

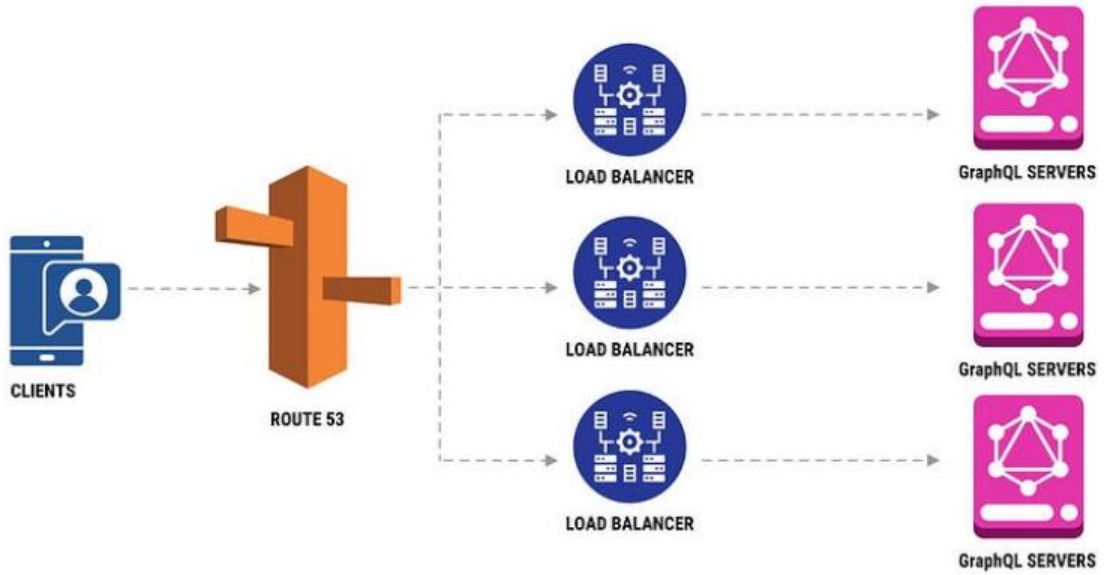
Uber

For example, **Uber** uses GraphQL to handle millions of requests per second by using horizontal scaling techniques like load balancing, sharding or clustering.

Horizontal GraphQL Scale Techniques: **LOAD BALANCING**

Load balancing is a technique used to distribute incoming network traffic across multiple servers.

- Scaling GraphQL horizontally, load balancing helps with:
  - Ensure that **no single server becomes overloaded** with requests, which can lead to slower response times and even downtime.
  - Instead, requests are spread out across multiple servers, **allowing for faster response times and better overall performance**.



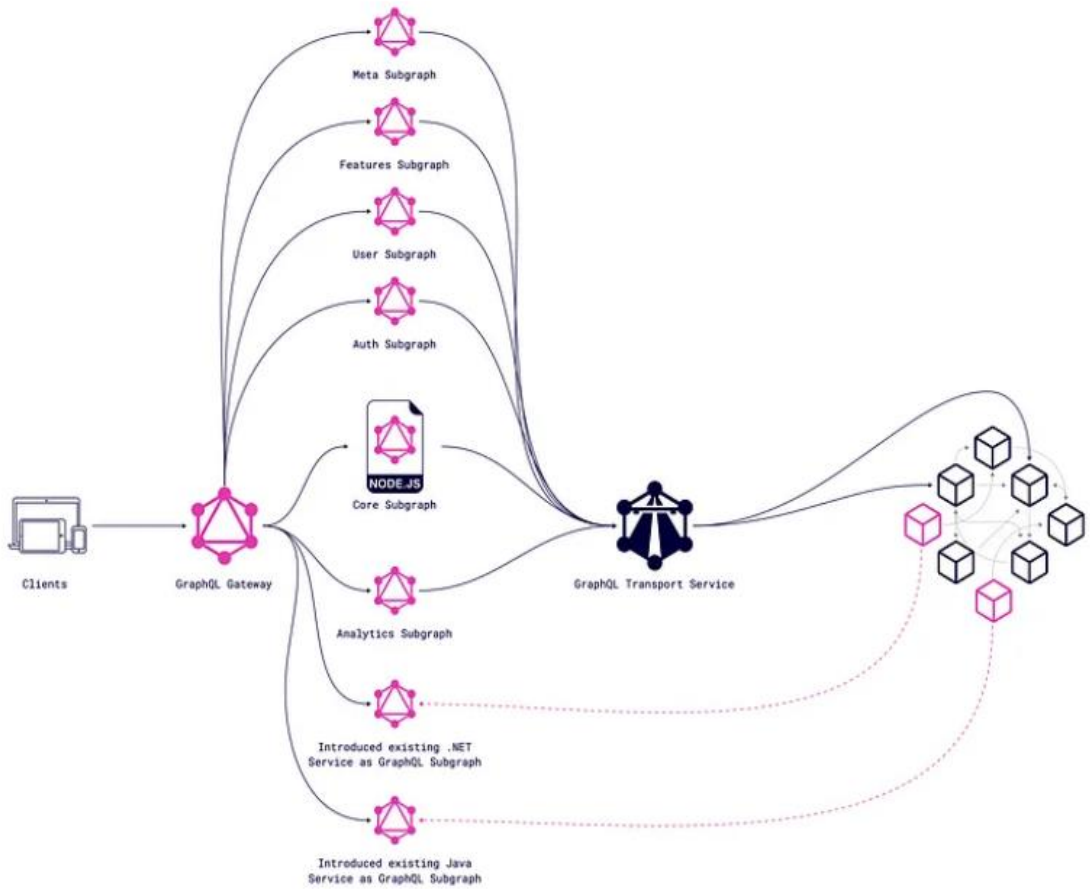
- Load balancing techniques:
  - **Round-robin** distributes requests evenly across all servers (different strategies: equal distribution, weighted servers, etc...)
  - **IP Hash** uses the client's IP address to determine which server to send the request to.
  - **Least connections** sends requests to the server with the fewest active connections.

Horizontal GraphQL Scale Techniques: **SHARDING**

- **Sharding** consists in splitting data into smaller, more manageable chunks called **shards**. Each shard contains a subset of the overall data and is stored on a separate server, allowing for faster response times.

- For example, the servers can be sharded based on user location or some other criteria, so that requests are routed to the appropriate shard. This can help **reduce latency** and improve response times.

- However, there are also drawbacks to consider such **as increased complexity** in managing multiple servers and potential data inconsistency.

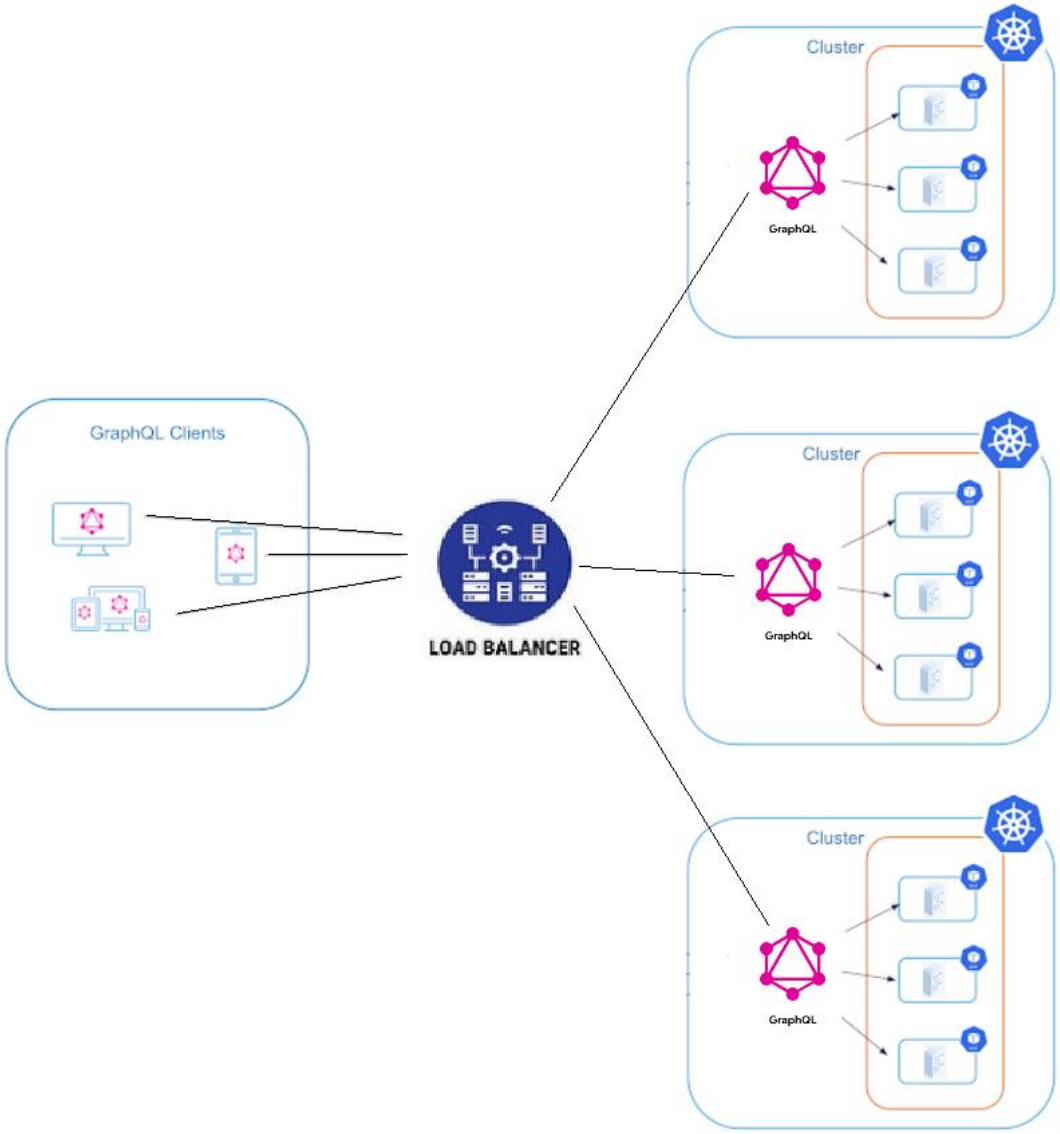


Horizontal GraphQL Scale Techniques: **CLUSTERING**

- **Clustering** consists in **grouping multiple servers together** into a cluster. These servers work together to handle incoming requests, with each server taking on a portion of the workload.

- One of the main benefits of clustering is its ability to provide **high availability and fault tolerance**. If one server in the cluster fails, the other servers can continue to handle requests without interruption.

- However, setting up and **maintaining a cluster can be complex** and time-consuming.



Horizontal GraphQL Scaling Best Practices

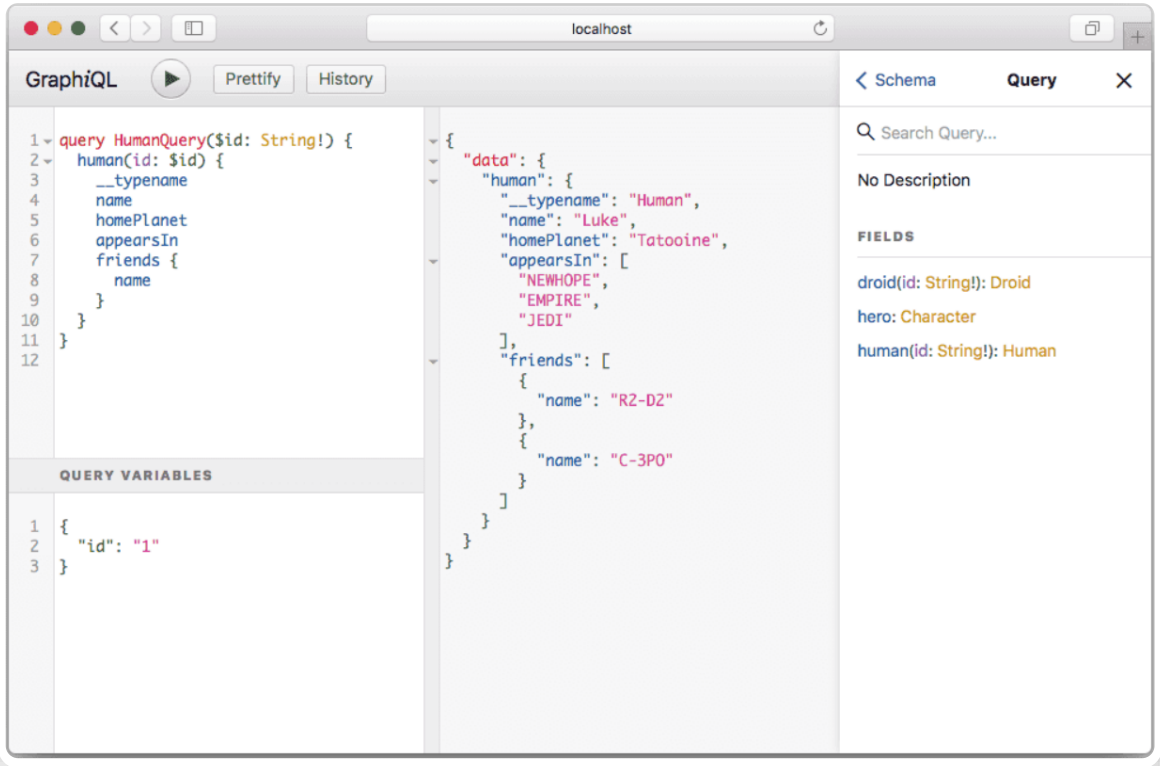
- Choosing the **right infrastructure** is crucial
  - Consider using **cloud-based services** like AWS or Google Cloud Platform, which offer **auto-scaling** and load balancing features.
  - Use a **CDN like Cloudflare** to distribute traffic and reduce latency.

- **Monitoring and optimizing** performance is key
  - Use a **distributed tracing system** that can help you identify performance bottlenecks and troubleshoot issues in real-time.
  - Implement **logging and error tracking** to capture detailed information about application errors and exceptions.

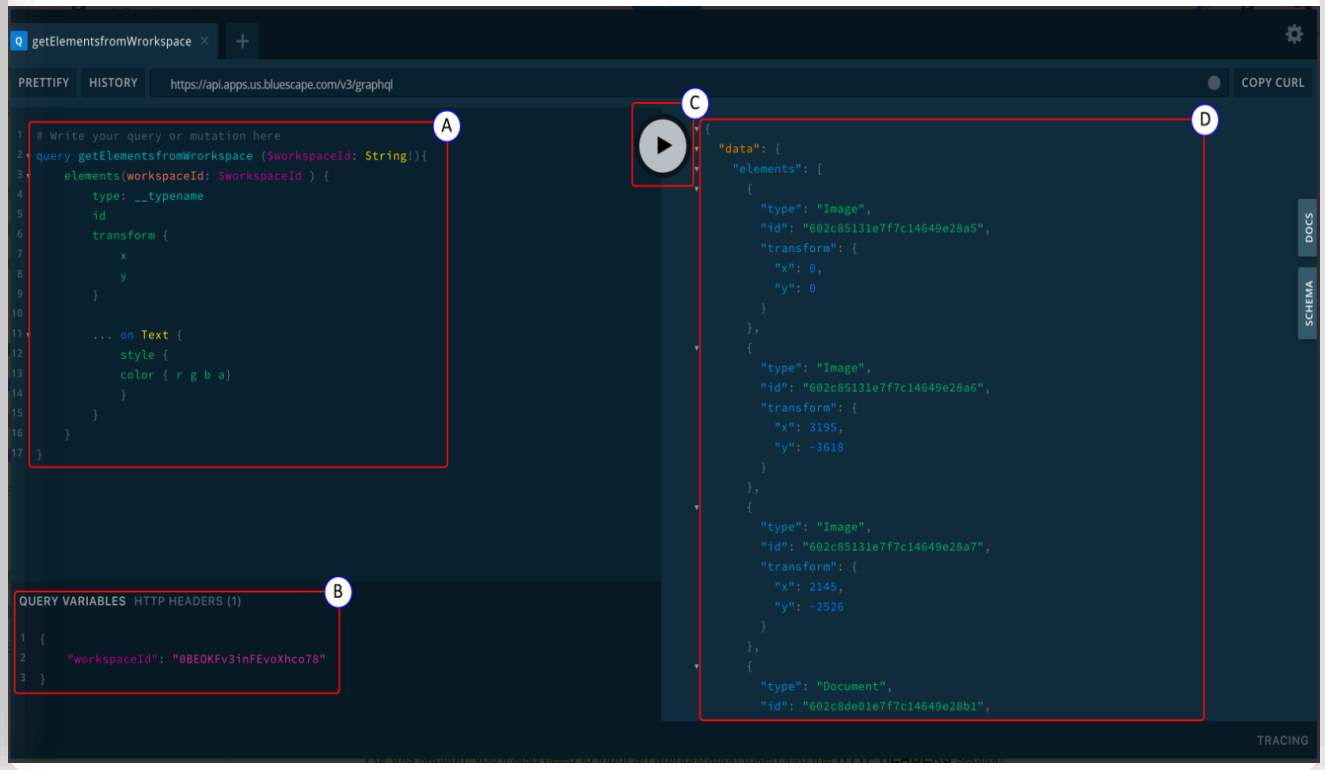
- Use **GraphQL introspection** to gain insights into the structure and behavior of your API
  - Use tools like **GraphiQL** or **GraphQL Playground**, you can explore your schema and query data in a more **interactive** and **visual** way, which can be especially helpful when **debugging complex issues**.

GraphQL Testing:

GraphiQL:



GraphQL Playground:



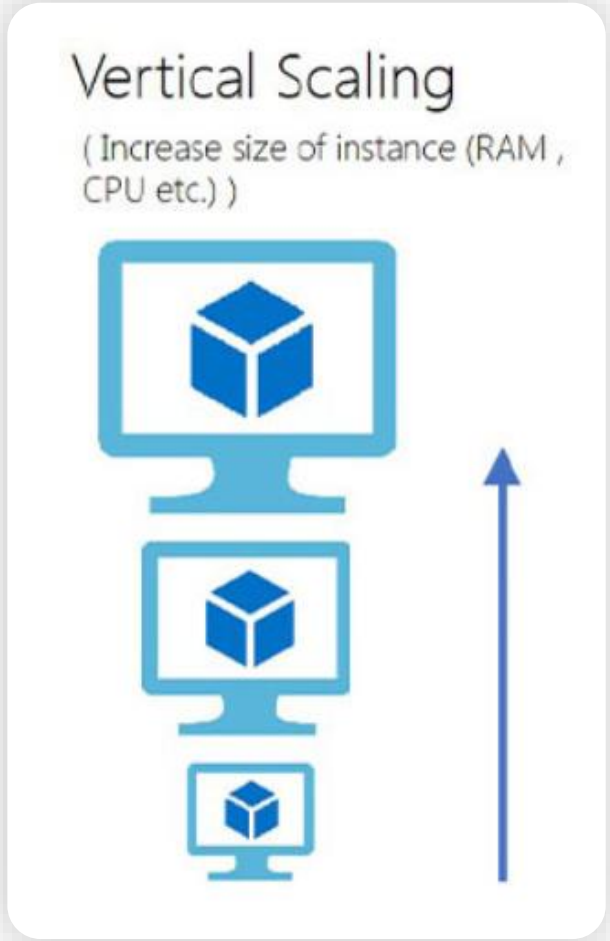
Agenda

1. What Is GraphQL
2. When to use GraphQL vs REST
3. Why Scale GraphQL
4. Why Scale GraphQL Horizontally?
   - 4.1. Horizontal GraphQL Techniques
   - 4.2. Horizontal GraphQL Best Practices
5. **Why Scale GraphQL Veritcally?**
   - 5.1. Vertical GraphQL Techniques
   - 5.2. Vertical GraphQL Best Practices
6. Other Best Practices
7. Real World Example
8. Conclusions

Why Scale GraphQL Vertically?

- **Vertical scaling** refers to **increasing the capacity of a single server** by adding more resources such as CPU, memory, or storage.
- Vertical scaling of GraphQL has several benefits:
  - **Increased performance** to handle more requests.
  - **Reduced costs** due to not adding additional servers.

— For example, **Netflix** uses vertical scaling to handle their massive amounts of traffic. They have a complex system of microservices that communicate with each other using GraphQL. By scaling vertically, **they can handle the high volume of requests while keeping costs down by using fewer servers**.



Vertical GraphQL Scale Techniques: **CACHING**

Caching is a powerful tool for scaling GraphQL vertically. It consists in **storing frequently accessed data in memory**.

At server-side, caching can significantly **reduce the number of database queries** required to fulfill requests. A cache such as **Redis or Memcached** can be used to overall performance. Warning: if there is horizontal scale, the cache should support distributed mode.

At client-side, caching allows the client to store query results in memory and **avoid redundant network requests**, reducing the number of requests sent to the server and resulting in faster and more responsive applications. Tools like **Apollo Client** can be used.

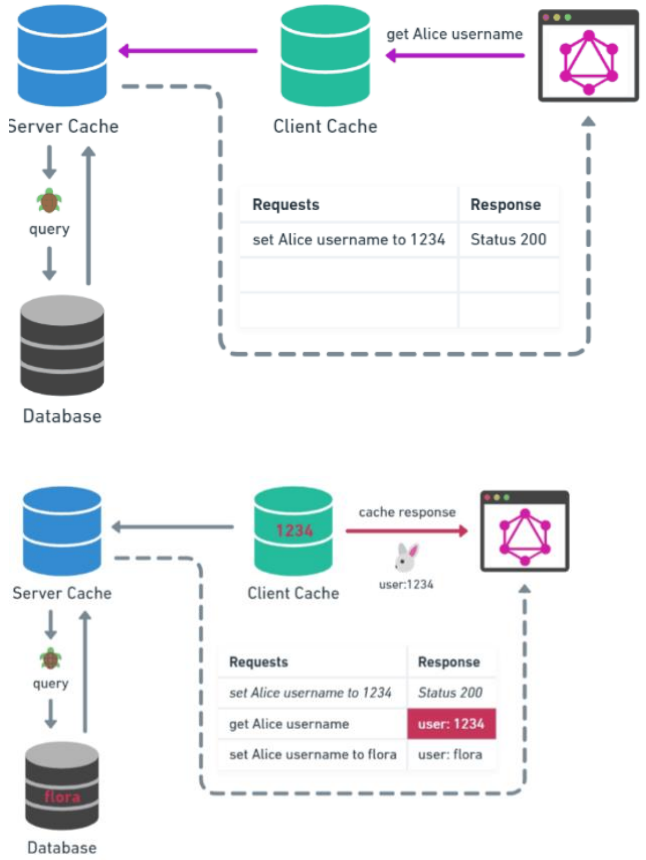
— There are several caching strategies that can be used with GraphQL, including:

- **Query-level** caching involves caching entire queries. **Parsing Level**.
- **Field-level** caching caches individual fields within a query. **Validation Level**.
- **Result-level** caching caches the results of a query, regardless of the specific fields requested. **Execution Level**.

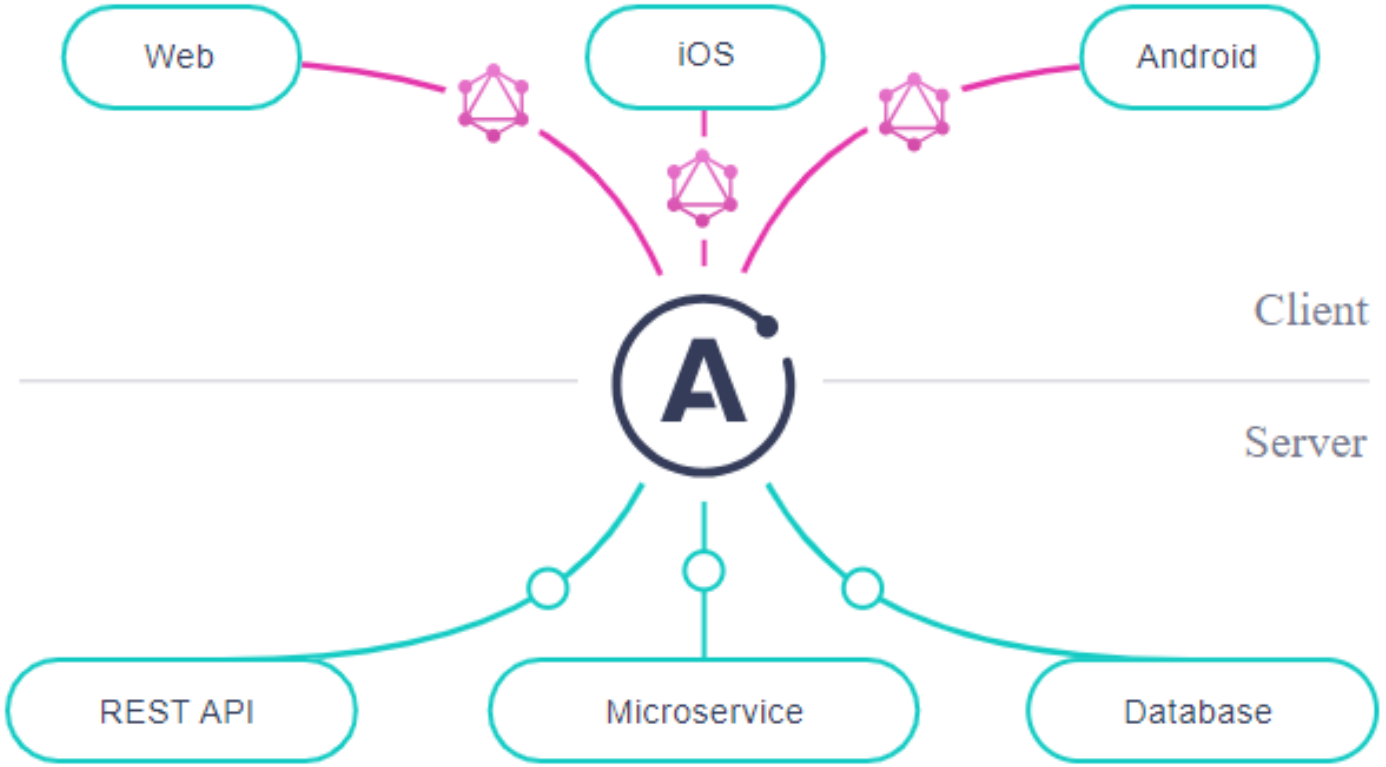
— Be sure to **set appropriate cache expiration times** to ensure data remains up-to-date.

Vertical GraphQL Scale Techniques: **CACHING**

Caching example:



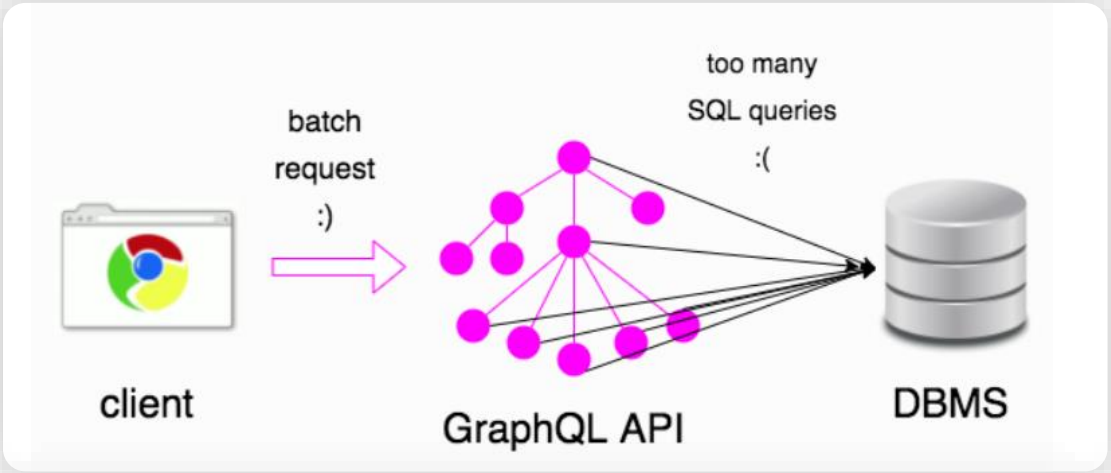
GraphQL Apollo (Client + Server):

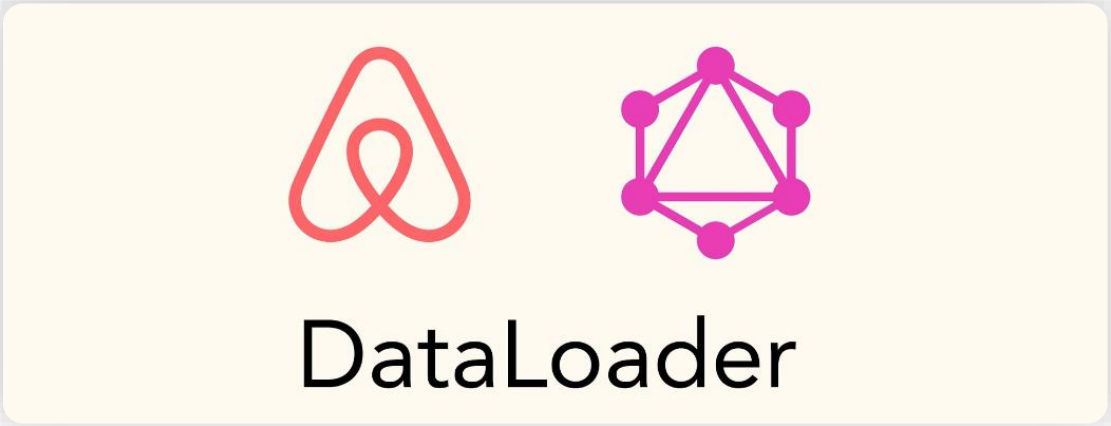


Vertical GraphQL Scale Techniques: **BATCHING QUERIES**

Batching queries consists in reduce the number of requests sent to the server. Instead of sending multiple queries one at a time, batching allows for **multiple queries to be grouped together and sent in a single request**.

- This approach can significantly **improve the performance** of GraphQL applications, particularly when dealing with large amounts of data or complex queries. **DataLoader** is a great library for batching queries.
- By reducing the number of requests, batching can also help to **minimize network overhead** and improve the overall user experience.
- It must be properly analyzed as **can introduce performance issues** at the server-side.





Vertical GraphQL Scaling Best Practices

- One of the main **challenges** of scaling GraphQL vertically is **increased complexity**. As more resources are added to a single server, it becomes **harder to manage and maintain.** This can lead to issues such as slower response times and increased downtime.
- Another challenge is the potential for **single points of failure**. If a single server goes down, it can bring down the entire system. This can be mitigated by using redundancy and failover mechanisms, but these can also **add to the complexity and cost of the system**.
- **Monitor your system** regularly to ensure that it is performing optimally. This includes monitoring CPU usage, memory usage, and network traffic. Use tools like **Prometheus or Grafana** to track metrics and identify potential issues before they become critical.
- **Test your system** to ensure that it can handle the expected traffic. Use tools like **Apache JMeter** or **Gatling** to simulate user traffic and identify potential bottlenecks. **Test both read and write operations** to ensure that your system can handle both types of traffic.

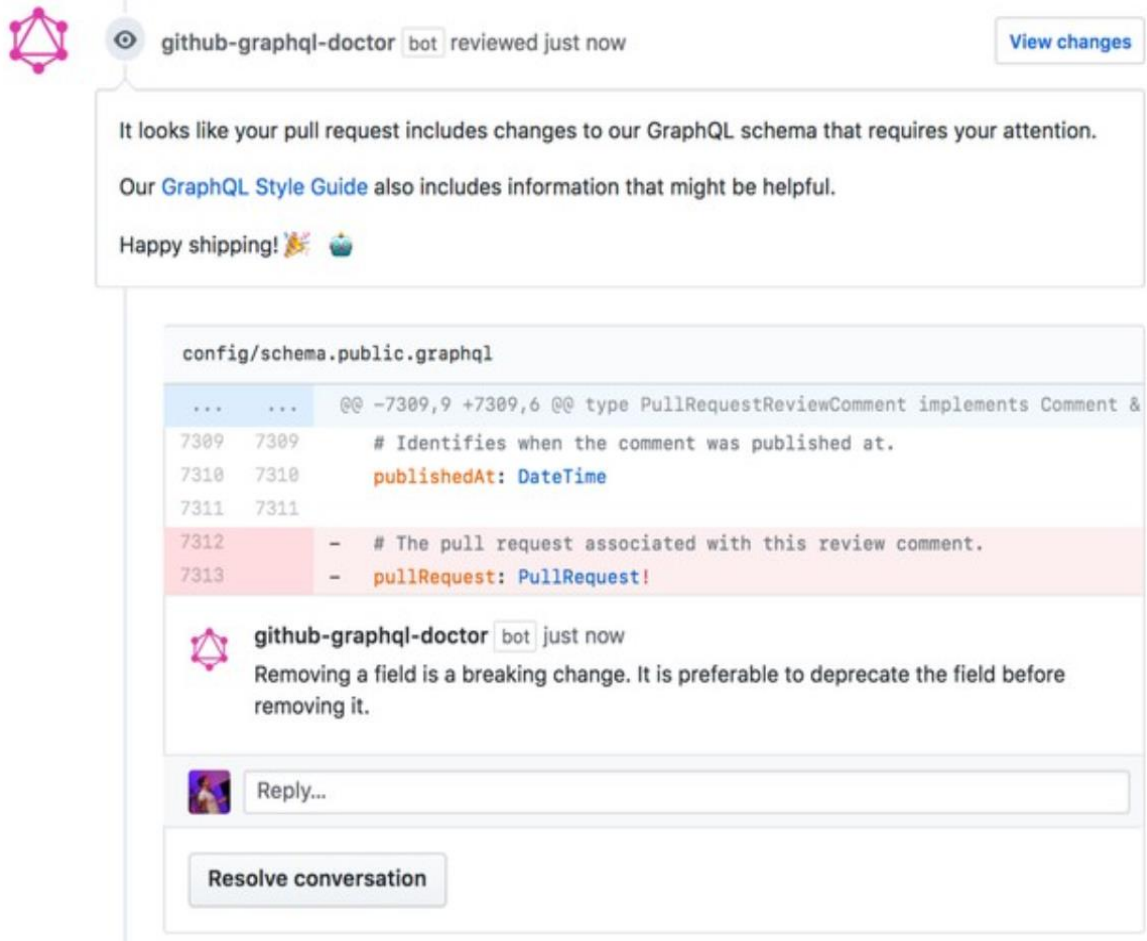
Agenda

1. What Is GraphQL
2. When to use GraphQL vs REST
3. Why Scale GraphQL
4. Why Scale GraphQL Horizontally?
   - 4.1. Horizontal GraphQL Techniques
   - 4.2. Horizontal GraphQL Best Practices
5. Why Scale GraphQL Veritcally?
   - 5.1. Vertical GraphQL Techniques
   - 5.2. Vertical GraphQL Best Practices
6. **Other Best Practices**
7. Real World Example
8. Conclusions

Other Best Practices for Scaling: **SCHEMA DESIGN**

- Design good GraphQL schemas helps to ensure scalability and maintainability.
- Keep the schema modular and well-organized, breaking down it into smaller, reusable components that can be easily combined to create more complex types and queries.
- Use a consistent naming convention for types and fields. This can make it easier to understand and navigate the schema, especially as it grows in size and complexity.

GraphQL-Doctor:

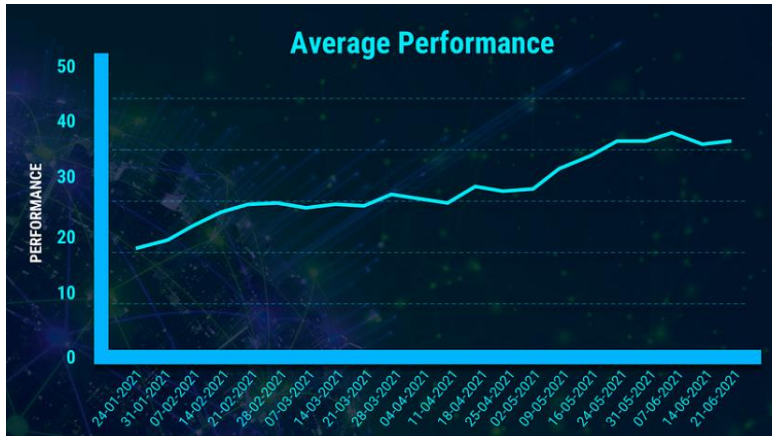


Agenda

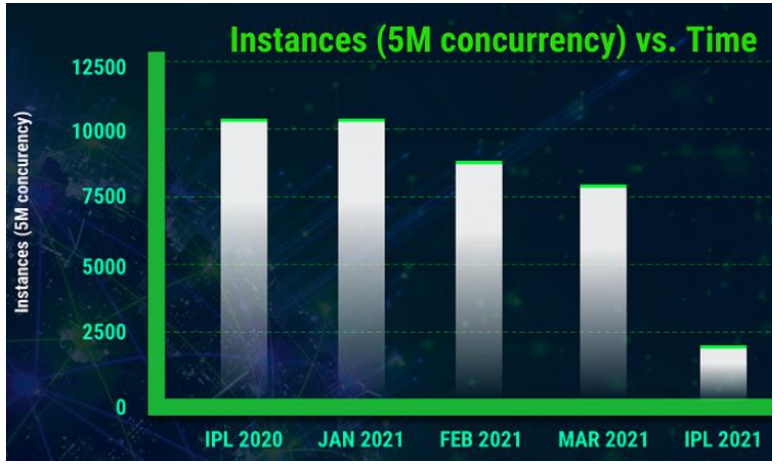
1. What Is GraphQL
2. When to use GraphQL vs REST
3. Why Scale GraphQL
4. Why Scale GraphQL Horizontally?
   - 4.1. Horizontal GraphQL Techniques
   - 4.2. Horizontal GraphQL Best Practices
5. Why Scale GraphQL Veritcally?
   - 5.1. Vertical GraphQL Techniques
   - 5.2. Vertical GraphQL Best Practices
6. Other Best Practices
7. **Real World Example**
8. Conclusions

Real World Example: **Dream11 Engineering**

- Dream11 is a company that provides a “Fantasy Sports” app that has grown from just 300,000 users in 2015 to over 110 million in 2021.



- They implemented GraphQL and Scale it using a mix of different techniques:
  - Load Balancing.
  - Batching Queries.
  - Caching (Parsing and Validating Phase).
  - Monitoring and testing including benchmarking.
  - Code Changes (removing immutable code from hot points).
  - Lazy Evaluation.
  - Infrastructure tuning (weighted DNS).

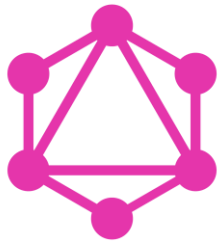


- The results were increasing the performance and reducing the costs significatively.
  https://blog.dream11engineering.com/lessons-learned-from-running-graphql-at-scale-2ad60b3cefeb

Agenda

1. What Is GraphQL
2. When to use GraphQL vs REST
3. Why Scale GraphQL
4. Why Scale GraphQL Horizontally?
   - 4.1. Horizontal GraphQL Techniques
   - 4.2. Horizontal GraphQL Best Practices
5. Why Scale GraphQL Veritcally?
   - 5.1. Vertical GraphQL Techniques
   - 5.2. Vertical GraphQL Best Practices
6. Other Best Practices
7. Real World Example
8. **Conclusions**

Conclusions



GraphQL is a powerful tool for **building scalable applications**. Scaling GraphQL is crucial for businesses that want to stay competitive in today's fast-paced digital landscape.

- Furthermore, the **benefits** of using GraphQL over traditional APIs for large projects are significant. With its ability to **reduce network overhead** and provide a **more flexible data mod**el, GraphQL is quickly becoming the preferred choice for developers across industries.
- By understanding the unique challenges that **arise when scaling GraphQL** applications, and **implementing best practices** such as caching strategies, batching queries, load balancing, sharding and good schema design, we can provide GraphQL APIs that are both fast and reliable and **create performant and maintainable applications**. Also monitoring and testing is highly recommended.

> However, it is important to keep in mind the **potential challenges that come with scaling**, such as increased **complexity** or single points of failure. **A mix** of horizontal and vertical scaling **would be the ideal solution**.


Questions?

Links

- https://speakerdeck.com/xuorig/continuous-evolution-of-graphql-schemas-at-github
- https://new.apollographql.com/Apollo-graphql-at-enterprise-scale-final.pdf
- https://frenzel-tim.medium.com/fast-cache-invalidation-for-graphql-with-quell-d6f61c0d69d2
- https://betterprogramming.pub/the-road-to-graphql-at-enterprise-scale-3325bf6b0ae4

- Apollo GraphQL Server and Client:
  https://new.apollographql.com

- Tool for caching and batching:
  https://github.com/graphql/dataloader
  https://www.youtube.com/watch?app=desktop&v=_FQ1ZEWIn2s

- Tool for schema maintenance:
  https://github.com/cap-collectif/graphql-doctor

ELCA

We make it work.

ELCA Informatique SA | Lausanne 021 613 21 11 | Genève 022 307 15 11
ELCA Informatik AG | Zürich 044 456 32 11 | Bern 031 556 63 11 | Basel 044 456 32 11

www.elca.ch