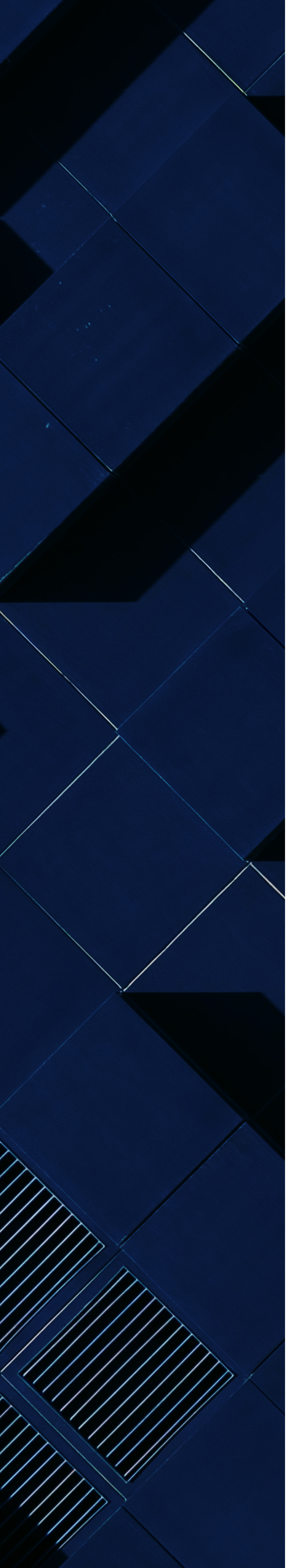


New Trends in API Architecture GraphQL & gRPC

By Aashish Pathak
Director, API Management & Integration, Blue Altair

Effective communication defines the success of the business and in this ever-changing digital world, API is acting as the backbone of the interaction among systems. While REST APIs have served well for long time, modern day challenges and complexities require efficient and high performing solutions.

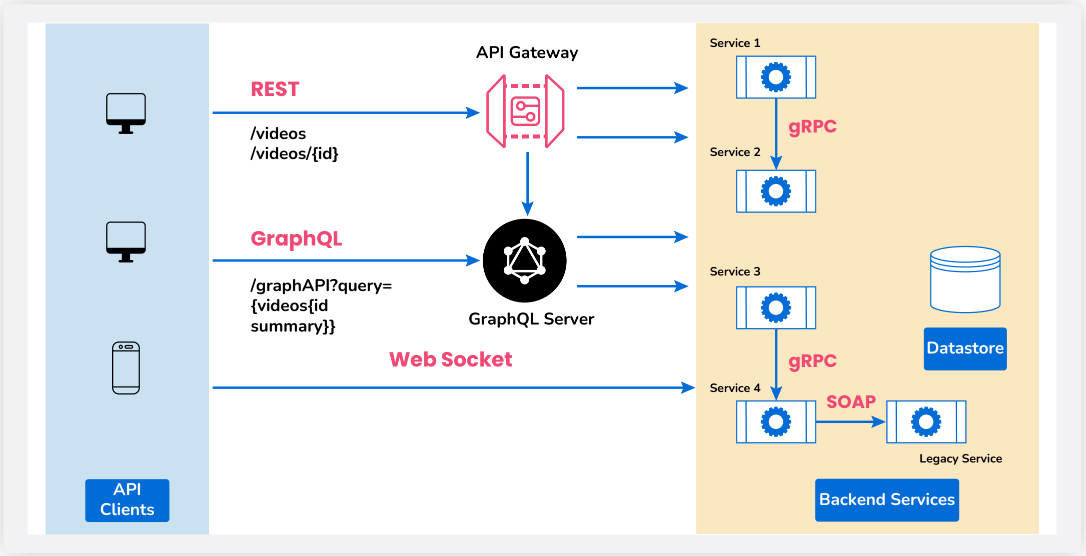
In this blog post, I will delve into two such effective API architectures: GraphQL and gRPC. These technologies offer more flexibility, speed, and control than traditional REST APIs, making them powerful tools for building scalable and responsive applications. Also, I will dive into their use cases, benefits, limitations, and where they fit best in today's API landscape.

Evolution of API Architectures

Rise of RESTful APIs

Representative State Transfer (REST) has become the dominant API architecture style, characterized by its use of standard HTTP methods and stateless communication, and has increased in popularity due to its simplicity, scalability, and ease of integration. However, in a short time, applications have become more complex, resulting in RESTful APIs facing challenges such as over-fetching and under-fetching, lack of flexibility in browsing structure, and difficulty in developing APIs without disrupting existing clients.

Figure 1



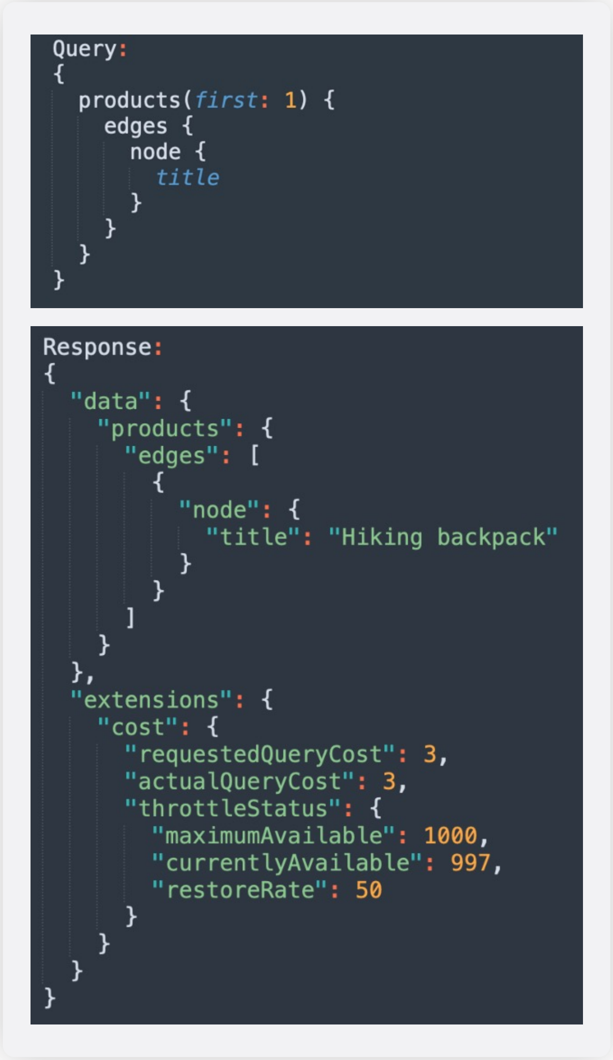
Emergence of New API Paradigm

The emergence of GraphQL addresses these limitations by allowing customers to clearly specify what information is required in a request. For example, a news aggregation application might use GraphQL to pull only news headlines and summaries without loading them. This results in more efficient data recovery and application performance.

One of the most powerful features of GraphQL is its ability to manage real-time data updates using subscriptions. This makes it ideal for applications that require live updates, such as chat platforms, sports apps displaying live scores, or stock market dashboards showing real-time price changes. For example, Shopify's GraphQL API leverages this capability to deliver dynamic updates efficiently. Figure 2 is a sample of a GraphQL request and response from the Shopify API.

On the other hand, gRPC uses Protocol Buffers (Protobuf) for efficient data serialization. Because it is a binary protocol, it has high performance and low latency. This makes it suitable for microservice architectures. Video streaming applications such as Netflix typically use gRPC to ensure minimal latency and high reliability. Bi-directional streaming and built-in authentication make IoT [Internet of Things] applications and real-time collaborative devices more attractive. With the overall shift towards microservices architecture, both GraphQL and gRPC meet the increasing demand for real-time data processing and efficient communication. It addresses the performance bottleneck of traditional RESTful APIs.

Figure 2



```
Query:
{
  products(first: 1) {
    edges {
      node {
        title
      }
    }
  }
}
```

```
Response:
{
  "data": {
    "products": {
      "edges": [
        {
          "node": {
            "title": "Hiking backpack"
          }
        }
      ]
    }
  },
  "extensions": {
    "cost": {
      "requestedQueryCost": 3,
      "actualQueryCost": 3,
      "throttleStatus": {
        "maximumAvailable": 1000,
        "currentlyAvailable": 997,
        "restoreRate": 50
      }
    }
  }
}
```

Trends and Future Directions

Both GraphQL and gRPC are experiencing important developments and trends that are shaping the future. Tracking trends reflect the search for efficiency, scalability, and a continuously improved developer and user experience.

Trends in GraphQL

Federated GraphQL

In a Federated GraphQL, a single large GraphQL schema is split into multiple smaller schemas that are managed by different teams or services. Companies are increasingly turning to Federated GraphQL to manage their apps. For complex and large-scale apps, Apollo Federation is a popular implementation tool, that allows the composition of multiple GraphQL services in a single unified API.

GraphQL Mesh

GraphQL Mesh allows GraphQL to be used as a universal data layer by combining multiple APIs including REST, gRPC, SOAP, etc. into a single GraphQL schema. This trend is gaining traction as it simplifies the integration of diverse and legacy systems into modern applications.

GraphQL Subscriptions and Real-Time Data

As real-time applications continue to grow in popularity, GraphQL subscriptions are increasingly being adopted for use cases like chat applications, live sports updates, and collaborative tools. By leveraging WebSockets, subscriptions enable seamless real-time data updates. Frameworks such as Apollo and Hasura have made it easier for companies to integrate these real-time capabilities into their systems, driving broader adoption of GraphQL for dynamic, event-driven experiences.

Trends in gRPC

gRPC in Microservices

There is a growing trend to use gRPC for inter-service communication in microservices environments. To replace the traditional RESTful approach, organizations with large microservices deployments, such as Uber and Netflix, are leveraging gRPC to improve communication between services.

Future of GraphQL and gRPC

Enhanced tooling and ecosystem

Expect more advanced tools for plan management, auditing, and security. Projects like GraphQL Inspector and Apollo Studio are leading the way in providing a robust GraphQL ecosystem of tools. Projects like OpenTelemetry and Envoy Proxy enhance the gRPC ecosystem.

AI and ML Integration

Both technologies will have improved integration with AI and machine learning frameworks, to support intelligent data and query processing. This integration will enable more powerful and advanced data-driven applications.

Edge Computing and IoT

Both technologies will play a key role in the growth of Edge Computing and IoT, providing efficient and scalable solutions for real-time data collection, processing, and communication in distributed networks.

Challenges and Considerations

Although GraphQL and gRPC offer powerful capabilities for modern applications, they also come with unique challenges that developers need to solve.

Challenges with GraphQL

Designing comprehensive and effective GraphQL services can be complex. This is especially true for large applications with diverse data needs.

GraphQL APIs may be vulnerable to security issues, such as Denial-of-Service (DoS) attacks, information disclosure, and unauthorized access.

GraphQL's flexibility can sometimes cause performance overhead especially for solving complex query problems that require views of multiple data sources.

Challenges with gRPC

gRPC's use of Protocol Buffers (Protobuf) can be more complex to understand compared to REST RPC-based communication models.

Web browsers do not directly support gRPC natively. Thus, limiting its use in web applications.

Integrating gRPC into existing systems. Especially systems that use different protocols or data formats.

Considerations to Overcome Challenges

Follow a planning-based design approach to ensure clarity and consistency in design.

Use an API gateway that can switch between gRPC and other protocols such as REST or SOAP.

Implement an effective certification authorization mechanism. Consider using rate limiting Depth restriction and limiting complexity to reduce DoS attacks.

Create API Governance practices, including code reviews, management plans, and documentation standards. Use tools like GraphQL SDL to document Swagger/OpenAPI.

Leverage existing tools such as Apollo Studio, GraphQL Playground, and GraphQL Inspector for GraphQL. Use additional or enhanced tools such as Reflection, gRPC, Envoy, and OpenTelemetry to improve observability for gRPC services.

Conclusion

Both technologies come with their own challenges and considerations. Schema complexity, safety concerns and the operating costs of GraphQL require careful planning and optimization. With a steep learning curve and limited browser support, the complexity of debugging gRPC requires a focus on training, creating tools, and best practices.

In summary, choosing between GraphQL and gRPC or a combination thereof depends on the specific requirements of your application and infrastructure. By understanding the strengths, challenges, and future directions, developers can make data-driven decisions by leveraging these technologies to create powerful, scalable, and innovative APIs. Now more than ever, it is important that enterprise organizations incorporate modern API architectures that are robust and future-proof to support the increasingly interconnected digital landscape of tomorrow.

About Blue Altair

Blue Altair is a niche, industry-recognized business and technology consulting firm that assists our clients with digital transformations. We offer Assessment and Strategy, Technology Implementation, and Managed Services in API Management and Integration; Data Management; Digital Application Development; and Data Science and AI. Our Client Success capability ensures a higher-than-industry rate of successfully delivered projects, with a primary focus on program and project management, business analysis, and quality assurance. Blue Labs is our innovation hub, where we use cutting-edge technology to build offerings that deliver accelerators and solutions. Our culture is the heart of our existence, and our core values are the key drivers for our handpicked, top-tier performers.

About the Author

Aashish is the Director of API Management & Integration capability at Blue Altair. With more than 13 years of experience in solutioning and software development, Aashish has a strong track record of delivering successful API and integration projects across multiple industries.

www.bluealtair.com

bluealtair
Driving Digital Success