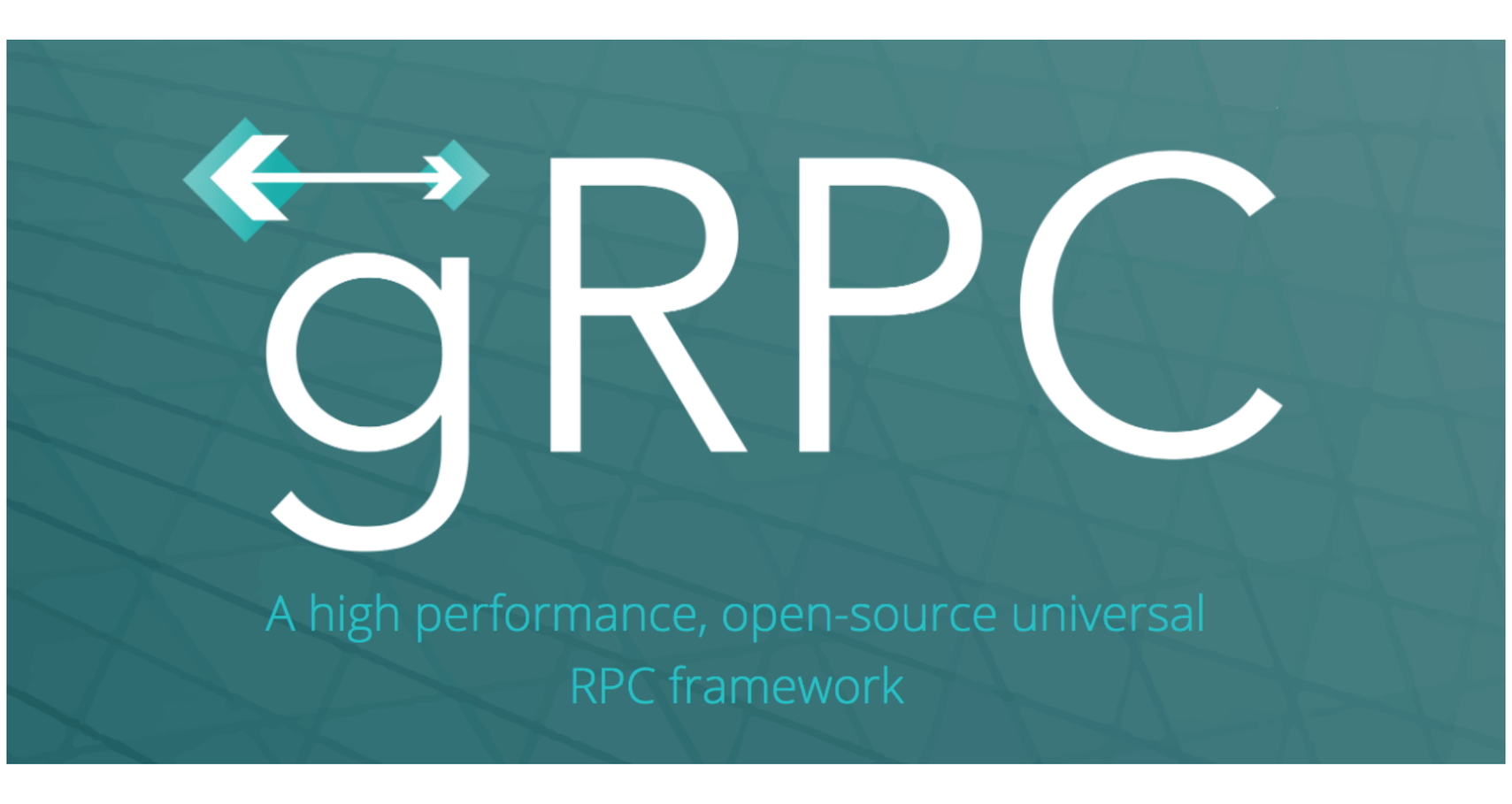
Understanding gRPC

And the differences between REST vs. RPC architectures

Arun Mathew Kurian Follow
Mar 6 · 6 min read



The objective of this article is to have a high-level idea of gRPC. It will also explain the similarities and differences between gRPC and existing protocols and architectures followed for the communication of web applications.

. . .

What Is gRPC?

gRPC is an open-source Remote Procedure Call framework that is used for high-performance communication between services. It is an efficient way to connect services written in different languages with pluggable support for load balancing, tracing, health checking, and authentication. By default, gRPC uses protocol buffers for serializing structured data. Generally, gRPC is considered as a better alternative to the REST protocol for microservice architecture. The 'g' in gRPC can be attributed to Google, who initially developed the technology.

Before going into any more details on gRPC, let's take a look at the microservice architecture.

. . .

Microservices vs. Monoliths

Monolithic architecture was the traditional way in which applications were designed. It contains a single indivisible codebase that serves the client-side user interface, server-side application, and the database. All the developers working in the project will contribute the code to the same repository. One of my favorite analogies related to a monolith is to think of it as a studio apartment. A single room will be divided into various spaces according to the need.

The advantage of monolithic architecture is that since there is only a single unit, operations like logging, performance monitoring, and caching can be done easily. Also, it is simple to develop, test, debug, and deploy.

But as the application grows, it becomes difficult to maintain, scale, and even understand. Also, it can become so complicated that a small change in code can affect the whole application.

Another important disadvantage of monoliths is that it is a rigid commitment to a single technology. The adoption of a new framework or language may need a full system rewrite.

Enter microservice architecture!

If monolithic architecture is a studio apartment, then microservice architecture can be considered as a house with many rooms. That means the whole application will be subdivided into multiple smaller applications or services.

This gives developer teams the flexibility to select the technology best suited for their needs and can let them scale their services independently. Any fault in a microservice application affects only a particular service and not the whole application.

These services can be developed, maintained, and deployed independently, and they communicate with each other through defined methods called APIs (Application Programming Interfaces).

The communication between the microservices over HTTP can be done in multiple ways. The most widely used way is to follow the REST protocol. gRPC is another way to perform this communication. It is built to overcome the limitations of REST in microservice communication.

. . .

REST Architecture

REST is a web architecture that uses HTTP protocol. It is widely used for the development of web applications. Simply put, REST is a client-server relation where back-end data is made available via simple representations like JSON/XML to the client. REST stands for REpresentational State Transfer, as described by Roy Fielding. REST is a protocol that does not enforce any rules about how it should be implemented at a lower level. It provides guidelines for high-level architecture implementation.

In order to make any application truly RESTful, six architectural constraints must be followed:

1. Uniform interface: Meaning API interfaces must be present to the resources in the web application to the consumers of the API.
2. Client-server: The client and server must be independent of each other, and the client should only know the URIs to the resource.
3. Stateless: The server must not store anything related to the client request. The client is responsible for maintaining the state of the application.
4. Cacheable: The resources must be cacheable.
5. Layered system: The architecture must be layered, meaning the components of the architecture can be in multiple servers.
6. Code on demand: The client must be able to get executable code as a response. This is an optional constraint.

Web services based on REST are known as RESTful web services. In these applications, every component is a resource and these resources can be accessed by a common interface using HTTP standard methods. The following four HTTP methods are commonly used in REST-based architecture:

- `GET` — Read-only access to a resource.
- `POST` — Create a new resource.
- `DELETE` — Remove a resource.
- `PUT` — Update an existing resource/create a new resource.

. . .

RPC Architecture

RPC stands for Remote Procedure Call. As the name suggests, the idea is that we can invoke a function/method on a remote server. RPC protocol allows one to get the result for a problem in the same format regardless of where it is executed. It can be local or in a remote server using better resources.

RPC is a much older protocol than REST. It has been used since the time of ARPANET in the 1970s to perform network operations. The term RPC was first coined by Bruce Jay Nelson in 1981. But as we are going to see, RPC is still relevant and implemented in API-based modern applications in different ways.

The idea is the same. An API is built by defining public methods. Then the methods are called with arguments. RPC is just a bunch of functions, but in the context of an HTTP API, it entails putting the method in the URL and the arguments in the query string or body.

RPC APIs will be using something like `POST /deleteResource` with a body of `{ "id": 1 }` instead of the REST approach, which would be `DELETE /resource/1`.

RPC is very popular for IoT devices and other solutions requiring custom contracted communications for low-power devices, as much of the computation operations can be offloaded to another device. Traditionally, RPC can be implemented as RPC-XML and RPC-JSON.

gRPC is the latest framework to be created on the RPC protocol. It makes use of its advantages and tries to correct the issues of traditional RPC.

. . .

What Is gRPC Again?

From whatever we have read so far, we can redefine gRPC. It is an adaptation of traditional RPC frameworks. So what makes it different from the existing RPC frameworks?

The most important difference is that gRPC uses protocol buffers as the interface definition language for serialization and communication instead of JSON/XML. Protocol buffers can describe the structure of data and the code can be generated from that description for generating or parsing a stream of bytes that represents the structured data. This is the reason gRPC is preferred for the web applications that are polyglot (implemented with different technologies). The binary data format allows the communication to be lighter. gRPC can also be used with other data formats, but the preferred one is the protocol buffers. To know more about protocol buffers in detail, check out this article.

Top highlight

Also, gRPC is built on top of HTTP/2, which supports bidirectional communication along with the traditional request/response. gRPC allows a loose coupling between server and client. In practice, the client opens a long-lived connection with the gRPC server and a new HTTP/2 stream will be opened for each RPC call.

. . .

REST vs. gRPC

Unlike REST, which uses JSON (mostly), gRPC uses protocol buffers, which are a better way of encoding data. As JSON is a text-based format, it will be much heavier than compressed data in protobuf format.

Another significant improvement of gRPC over conventional REST is that it uses HTTP 2 as its transfer protocol. HTTP 1.1, which is mainly used by REST, is basically a request-response model. (The REST can also be implemented with HTTP2) gRPC makes uses of the bidirectional communication feature of HTTP 2 along with the traditional response-request structure. In HTTP 1.1, when multiple requests come from multiple clients, they are served one by one. This can slow down the system. HTTP 2 allows multiplexing, so multiple requests and responses can be served at the same time.

We can conclude that gRPC is a great option when the use cases involve multi-language communications with idiomatic APIs or large-scale microservice communications.

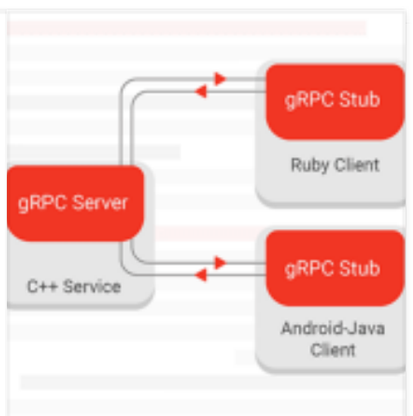
. . .

References

Microservices vs Monolith: which architecture is the best choice for your business?
By Romana Gnatyk * October 03, 2018 Having come into light just a few years ago, microservices are an accelerating…
www.n-ix.com

REST Architectural Constraints
REST is an architecture style for designing loosely coupled applications over HTTP. RESTful principles does not enforce…
restfulapi.net

Benefits and Best Practices of Adopting GRPC - XenonStack
gRPC is a high performance, open source universal RPC Framework.In simple words, it enables the server and client…
www.xenonstack.com



Thanks to Zack Shapiro.

Microservices Rpc DevOps API Programming

1K claps

WRITTEN BY
Arun Mathew Kurian Follow
Devoted reader | Ruby on Rails Developer| Movie Buff

Better Programming Follow
Advice for programmers.

See responses (2)