Understanding API Types and Choosing the Right One

Rapid

Table of Contents

Introduction

Application Programming Interfaces (APIs) emerged as a key component for modern software development practices, enabling development teams to more efficiently create applications and microservices without having to rewrite common functionality from scratch. It is not surprising that API adoption is on the rise. In a recent survey conducted by RapidAPI in 2019, respondents reported that their organization will continue to increase API usage over the course of 2020. ***In fact, almost 67% expect to use APIs more this year than previously in 2019.*** The survey also notes that larger development organizations are already using hundreds of Internal APIs. In fact companies with 5000 to 10,000 software developers use, on the average, more than 300 internal APIs.

As the number of APIs continues to increase, the types of APIs being used in applications are becoming more varied. Choosing the right type of API for a project requires understanding the different available types and the best use cases for each one. While REST APIs have been popular in recent years, developers use many different kinds of APIs to build their applications depending on their specific requirements. Some may choose APIs based on performance, scalability, or reliability — while others might choose an API for its role in simplifying the development process or enabling developers to more easily modify the code. It is also important to understand that there are many differences, beginning with the very nature of the API as some are architectural styles while others are frameworks, or query languages (like GraphQL).

This guide will highlight the different API formats and specifications, describe the background of each API, detail the benefits, and provide the optimal use case for selecting different APIs for your project.

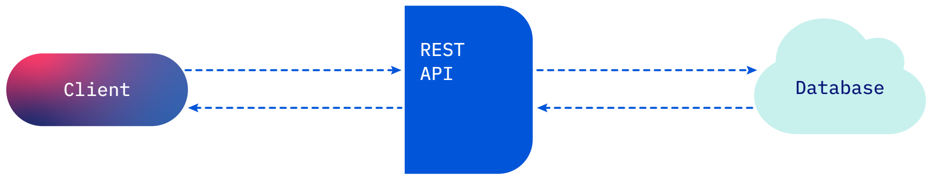
REST

REST (REpresentative State Transfer) is an architectural style that is based on the open internet philosophy. It was introduced in 2000 in Roy Fielding's doctoral dissertation, Architectural Styles and the Design of Network-based Software Architectures. REST APIs are used to call resources, and allow software to communicate based on standardized principles, properties, and constraints.

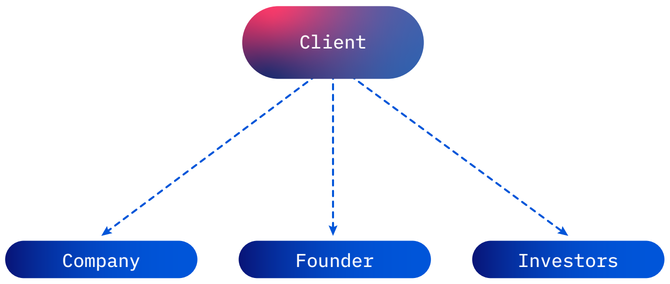
Today, the growth of the API economy is largely driven by REST APIs, and REST has become the default standard for web APIs. REST APIs are now a core business feature for many software companies, or even the core product for companies like Stripe and Twilio.

How REST Works

REST APIs operate on a simple request/response system. The request includes the HTTP method (GET, POST, PUT, PATCH, DELETE), endpoint, headers, URL parameters, and the body. The response returns the relevant data, which can be formatted as JSON, XML, plain text, images, HTML, and more.



RESTful APIs can also be designed with many different endpoints that return different types of data. Accessing multiple endpoints with a REST API requires multiple calls.



A true RESTful API will also conform to the REST architectural constraints outlined by Fielding's dissertation including:

1. **Client-Server Architecture:** The client and server are decoupled to improve scalability and allow the components to evolve independently.
2. **Statelessness:** Each request contains the information necessary to service the request
3. **Cacheability:** Responses can be explicitly or implicitly defined as cacheable or non-cacheable to improve scalability and performance.
4. **Layering:** Different layers of the API architecture should work together, creating a scalable system that is easy to update or adjust.
5. **Uniform Interface:** Communication between the client and the server must be done in a standardized language that is independent of both. This improves scalability and flexibility.

When to Use REST

As you can see from the constraints of the REST architectural style, RESTful APIs are a good fit for projects that need to be **flexible, scalable, and fast**. These characteristics make RESTful APIs particularly well-suited for web applications. In addition, if you are looking for the following characteristics, REST is a strong option:

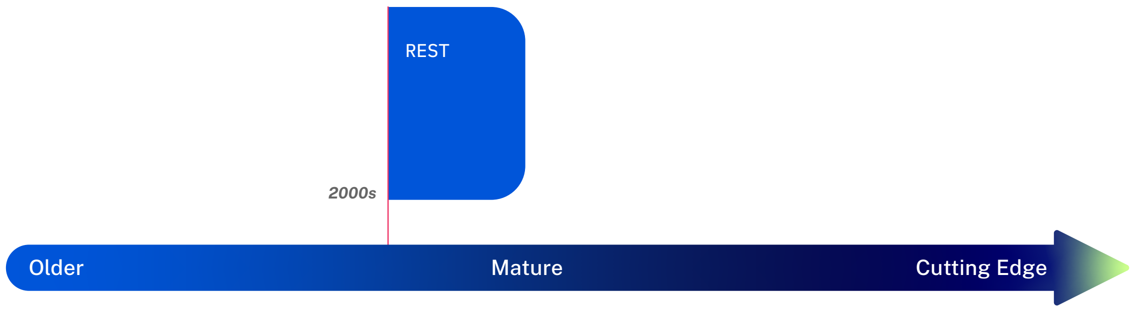
Familiarity: most people in your engineering team have already used — or at least have seen — a RESTful API. This type of API, being the most common, will have the shortest learning curve.

Interoperability: Due to the popularity of REST, nearly every platform and framework has a built in library capable of interfacing with a REST API that has wide support.

Development efficiencies: REST APIs are reusable, enabling developers to easily create independent microservices that work independently of one another as they are decoupled from clients and accessible by multiple applications.

REST Maturity

In recent years, RESTful APIs have become the most popular type of API, due to their flexible and fast performance. In our survey of over 2000 developers, 62.5% of respondents reported using REST in production — making it the most popular technology included in our survey.[*] An additional 20% reported POCing or investigating REST for future use.



Based on these stats — and the increasing development of RESTful APIs as a core business service for many companies — we anticipate REST APIs will continue to be a popular choice for years to come.

[*] Our developer survey was run independently of this document and did not cover all types of APIs discussed in this ebook. The survey question that asked developers about their familiarity with certain technology trends covered GraphQL, REST, gRPC, Webhooks, and Serverless & FasS.

SOAP

SOAP (**S**imple **O**bject **A**ccess **P**rotocol) is a standardized protocol that relies on XML to make requests and receive responses. SOAP APIs make data available as a service and are typically used when performing transactions that involve multiple calls or for applications where security is the main consideration.

SOAP was initially developed for Microsoft in 1998 to provide a common mechanism for integrating services on the internet regardless of operating system, object model, or programming language.

How SOAP Works

The “S” in SOAP stands for Simple, and for good reason — SOAP can be used with less complexity as it requires less coding in the app layer for transactions, security, and other functions.

SOAP has three primary characteristics:

1. **Extensibility:** SOAP allows for extensions that introduce more powerful features, such as Windows Server Security, Addressing, and more.
2. **Neutrality:** SOAP is capable of operating over a wide range of protocols, like UDP, JMS, SMTP, TCP, and HTTP.
3. **Independence:** SOAP is compatible with nearly any programming language.

SOAP-based requests and responses can be combined with a transport protocol, like HTTP, for use in web services.

When to Use SOAP

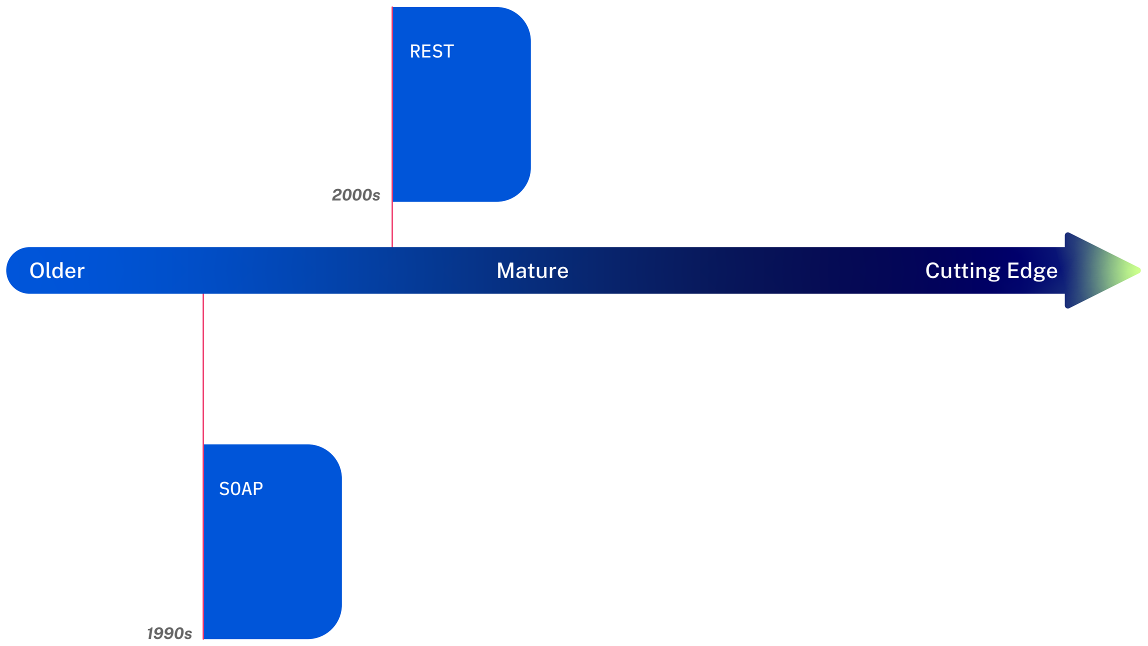
Developers continue to debate the pros and cons of using SOAP and REST. The best one for your project will be the one that most aligns with your needs. For example, SOAP remains a top choice for corporate entities and government organizations that prioritize security, even though REST has largely dominated web applications.

SOAP is a good choice when security is a priority because its standard HTTP protocol makes it easier to operate across firewalls/proxies without making any modification.

It also has greater transactional reliability, which is another reason why SOAP historically has been favored by the banking industry and other large entities.

SOAP Maturity

SOAP's introduction in 1998 means it is quite a bit older than most of the other technology and API types we are looking at in the guide. SOAP dominated the API space for many years but waned in popularity with the rise of REST.



However, that isn't to say REST completely replaces SOAP. There are still specific applications where SOAP remains the best option, and it will likely continue to be a common choice for enterprise applications and projects that require enhanced security.

GraphQL

GraphQL — a query language that lets the client define the structure of the data required

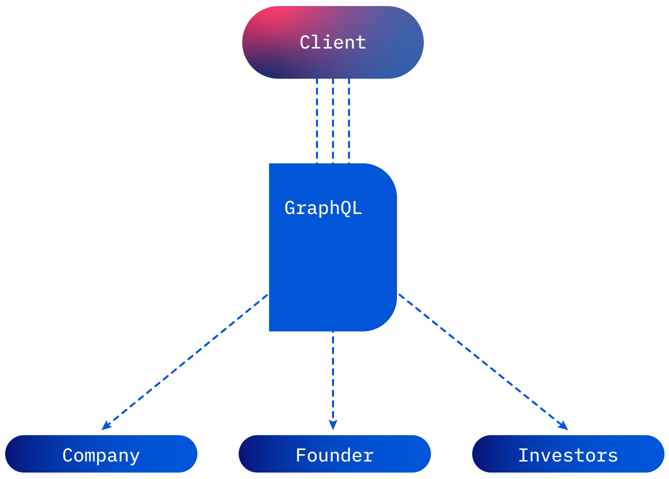
— was created by developers at Facebook in 2012. It was developed to support the complicated data structures required to show the Facebook News Feed on the mobile application. GraphQL was later open-sourced in 2015.

GraphQL APIs have two main advantages over other traditional APIs:

1. GraphQL allows you to request data from multiple resources in a single request. In comparison, traditional APIs require making multiple requests to fetch each type of resource.
2. GraphQL allows you to control exactly what information your application receives in the response of the API. With traditional APIs, you have less control which can lead to over-fetching data in certain scenarios.

How GraphQL Works

GraphQL relies on a strongly typed schema. The pre-defined schema is what allows the client to specify the exact shape of the data returned.



The ability to define the exact request and response structure is also a way to cut down on the resources and bandwidth required to fetch data, as only the desired data will be included in the response. This makes GraphQL a powerful tool for projects that might be

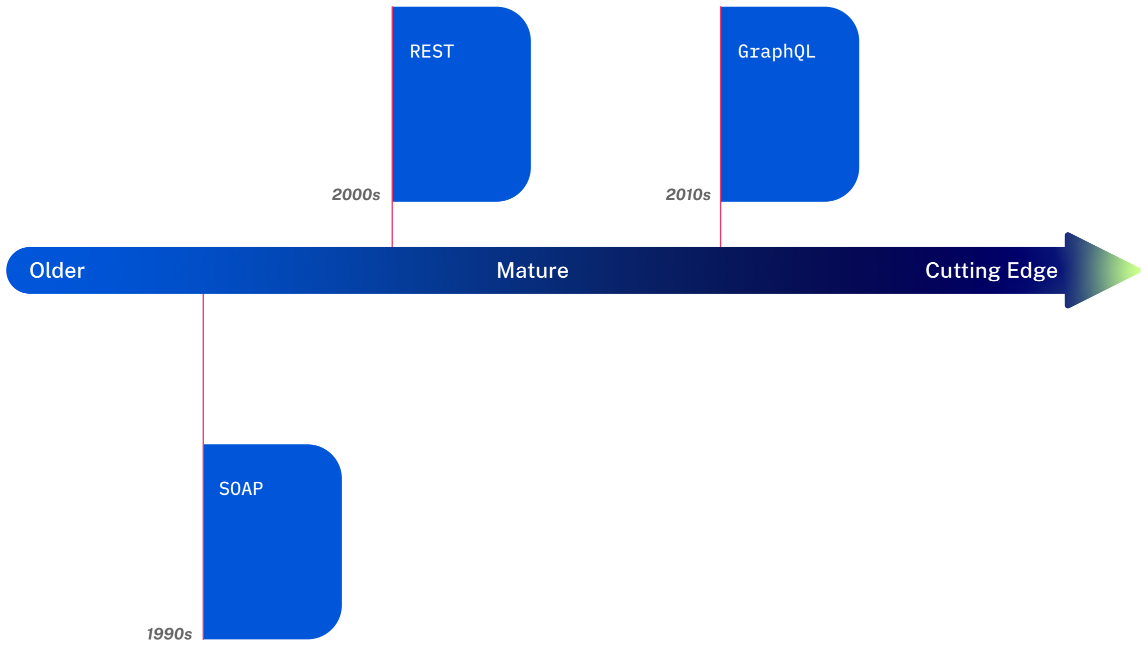
When to Use GraphQL

GraphQL was developed to overcome some of the limitations of REST APIs. Ultimately, both use different methods to accomplish similar tasks. GraphQL APIs are a great choice if you are seeking to reduce the number of queries needed to fetch the required data, when the data is being pulled from multiple resources.

Much like REST APIs, many companies — including GitHub, Yahoo, and others — have made GraphQL APIs publicly available for developers.

GraphQL Maturity

GraphQL's introduction in 2012 makes it one of the newer technologies in this list. In our recent survey of over 2000 developers, we found 19.7% of respondents were either investigating or launching GraphQL as a proof of concept. An additional 36.4% reported they are unfamiliar with GraphQL.



This indicates that GraphQL is still relatively new. Even though GraphQL was introduced for a very niche purpose, we expect to see an increase in use in the years to come as developers become more familiar with it and large companies continue to make GraphQL APIs publicly available.

Kafka-based APIs (Async APIs)

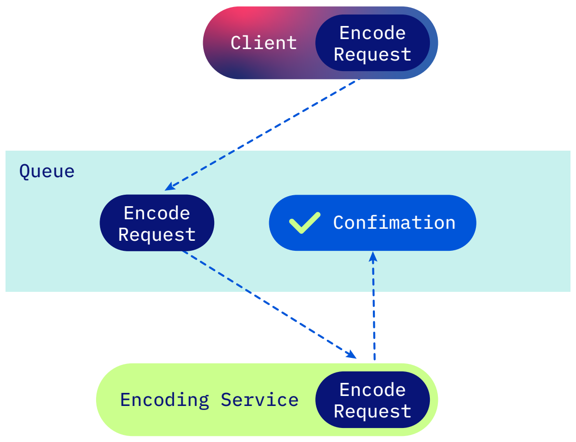
Apache Kafka is a stream-processing software platform originally developed by LinkedIn. Kafka was open-sourced in 2011 to “provide a unified, high-throughput, low latency platform for handling real-time data feeds.”

Kafka and Kafka-based APIs have become increasingly popular among large enterprises due to the unique ability to handle and process data streams and ingest and move large amounts of data quickly.

How Kafka Works

Kafka works like a pub/sub system using publishers, topics, and subscribers. There are 5 core Kafka-based APIs:

1. **Producer API:** Used to publish a stream of records to one or more Kafka topics.
2. **Consumer API:** Used to subscribe to one or more topics and process the stream of records produced to them.
3. **Streams API:** Used to consume input streams from one or more topics and produce an output stream to one or more output topics. This essentially transforms the input streams to output streams, allowing the application to act as a stream processor.
4. **Connector API:** Allows building/running reusable producers or consumers that connect Kafka topics to existing applications or data sets.
5. **Admin API:** Allows managing and inspect topics, brokers, and other Kafka objects

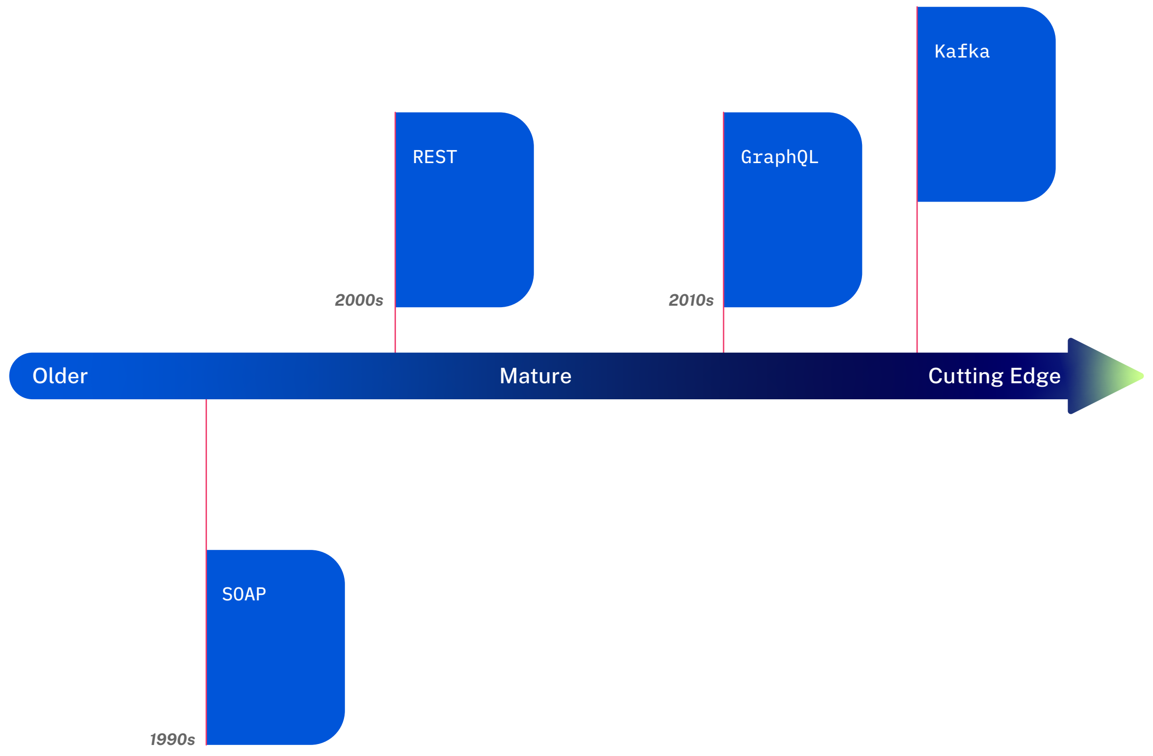


When to Use Kafka

Kafka was originally used to build a set of real-time publish-subscribe feeds on LinkedIn. Since its initial development, it has expanded to a range of uses for large scale, complex applications. Kafka shines when it comes to real-time data and analytics — in particular for streaming data pipelines that get data between systems or applications. Some of the common uses include operational monitoring data, log aggregation, event sourcing, and stream processing.

Kafka Maturity

Kafka is already being used by many of the world's leading software companies — including Airbnb, Pinterest, Cloudflare, PayPal, Spotify, and Twitter. We expect to see even more companies turn to Kafka and Kafka-based APIs as the technology matures.



Publish/Subscribe Pattern Based APIs

Publish/Subscribe (Pub/Sub) is an asynchronous messaging style used in serverless and microservices architectures. With this model, messages are not sent to a specific subscriber but are instead categorized so that they are available to all subscribers of the category.

How Pub/Sub Pattern Based APIs Work

The main characteristic of Pub/Sub APIs is the existence of publishers and subscribers, as the name implies. Publishers categorize messages, and those that are subscribed to a specified category receive the message.

As mentioned in the previous section, Kafka APIs are considered to be a type of Pub/Sub based APIs. Other types of APIs and microservices can also be built around a Pub/Sub system, including REST APIs.

For example, REST APIs use POST and DELETE operations to integrate with Pub/Sub. POST operations publish messages, create subscriptions, and get messages from queues. DELETE is used to unsubscribe.

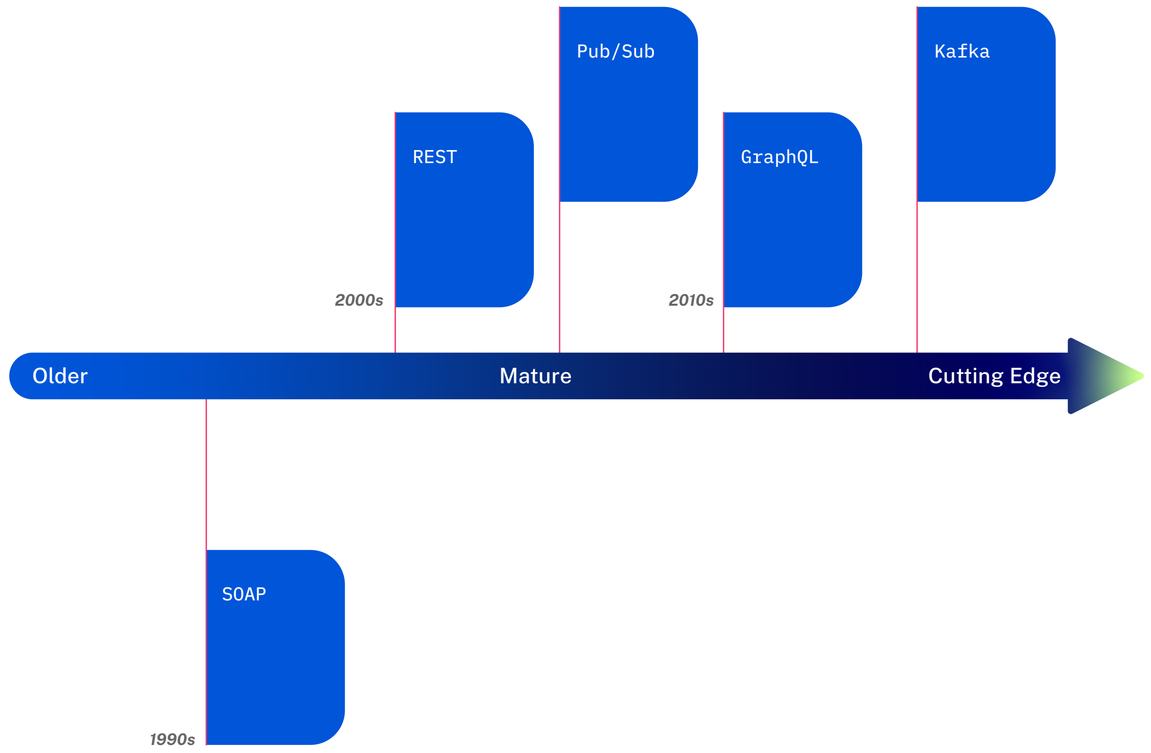
When to Use Pub/Sub Pattern Based APIs

Pub/Sub based APIs are a great addition to architecture systems that involve many independent or decoupled components. Pub/Sub based APIs can be used to provide event-driven notifications as a result of specific events that occur within the system.

This style of system is also highly scalable compared to more traditional client-server infrastructure.

Pub/Sub Pattern Based APIs Maturity

Pub/Sub systems have been around since the late 1980s. However, Pub/Sub systems and Pub/Sub-based APIs also have more modern implementations. One of the most notable is the Cloud Pub/Sub by Google Cloud. Based on this, we believe Pub/Sub-based APIs are here to stay, and these systems may inspire new types of APIs — like Kafka.



Webhooks

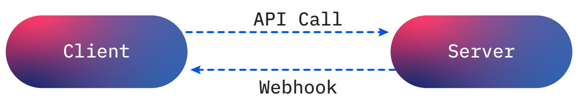
Webhooks are similar to APIs but don't technically qualify as such. With a traditional REST API, you send a request and get a response. However, no request is required for a webhook. Instead, the response is sent whenever a specified event occurs.

How Webhooks Work

When a specified event occurs, a webhook makes an HTTP request to a designated URL. This allows you to push data to your application the moment a particular event happens.

When to Use Webhooks

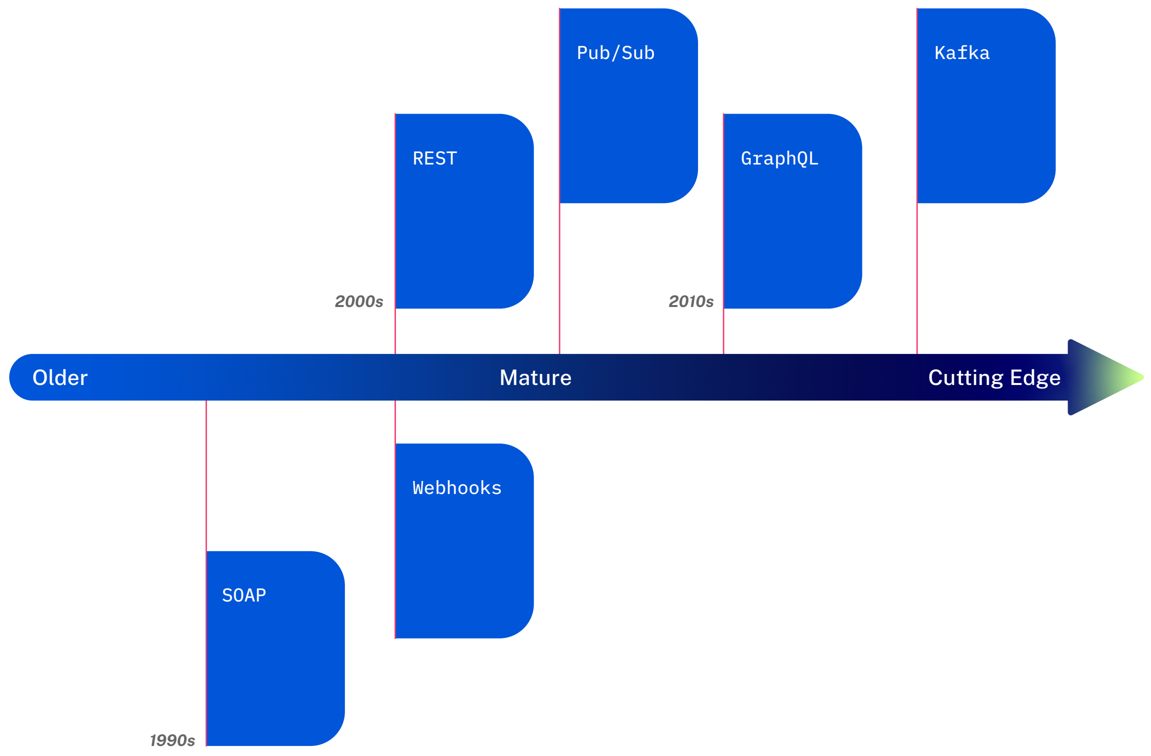
Webhooks are commonly used when real-time data is required, but the data changes relatively infrequently. Instead of sending repeated API requests to get live data, a webhook can be triggered every time there is an update. This will ensure you have accurate data without having to make frequent API requests, which can be costly or use a lot of bandwidth.



They are also used to create notifications that are triggered by a specific event, making them very common for e-commerce, communication, social media, and other platforms you probably use every day.

Webhook Maturity

In our Developer Survey, webhooks were the second most commonly used technology, right behind REST APIs. Nearly 30% of developers reported using them, with an additional 20% currently POCing or investigating the for future use. Webhooks are also used by many of today's leading software companies, including Twilio, Square, and MailChimp — so we believe they are here to stay.



RPC (Remote Procedure Call)

RPC (Remote Procedure Call) is a protocol that uses the client-server model to enable one program to request a service from another program over a network. As implied by the name, an RPC can call a function on a remote server. This typically tightly couples the client to the server, in contrast to newer API designs like REST where the client and server are decoupled.

RPC was theoretically introduced in the early 1970s, and was first used in production in the early 1980s.

How RPC Works

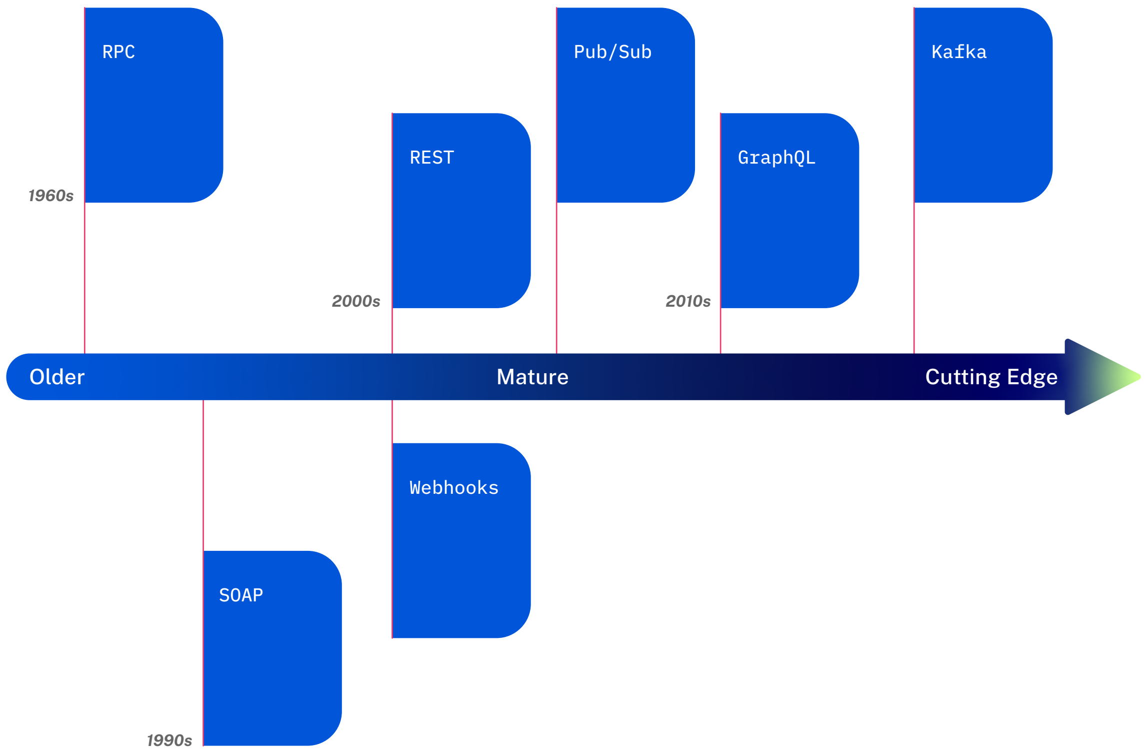
With an RPC, you can structure your call as if you are calling a function directly on the server. This familiar format makes RPC easier to start using.

When to Use RPC

RPCs are comparatively simple compared to certain types of API calls, making them relatively easy to understand or update. Calls are typically lightweight and high performance, making RPC suited for situations where a large volume (think millions or even billions) is expected. These characteristics also make RPCs a popular choice for IoT environments.

RPC Maturity

Given RPC's introduction in the 70s, it is one of the oldest technologies in our list. Despite RPC's relatively old age, newer and more cutting edge implementations of RPC are still being used by many large technology companies. One example of a more modern implementation is gRPC — used by Google, Netflix, Square, and many others.



gRPC

gRPC is a high performance, open-source Remote Procedure Call framework. It was introduced by Google as an open-source version of their internal RPC technology. gRPC is based on HTTP/2 which can be more efficient than HTTP used by REST and other traditional methods.

How gRPC Works

gRPC was designed to be easy to use, and much of the setup is generated for you. First, you start by defining the messages you want to send with protocol buffers (defined in the next section).

Next, you need to add a service definition. There are a few different varieties:

1. **Unary RPC:** The client sends a request, the server sends a response. Most similar to a REST API.
2. **Client Streaming RPC:** The client sends multiple messages, the server sends one response.
3. **Server Streaming RPC:** The client sends one message, the server sends multiple messages.
4. **Bi-Directional RPC:** The client and server can independently send multiple messages to each other. Most similar to Kafka-based APIs.

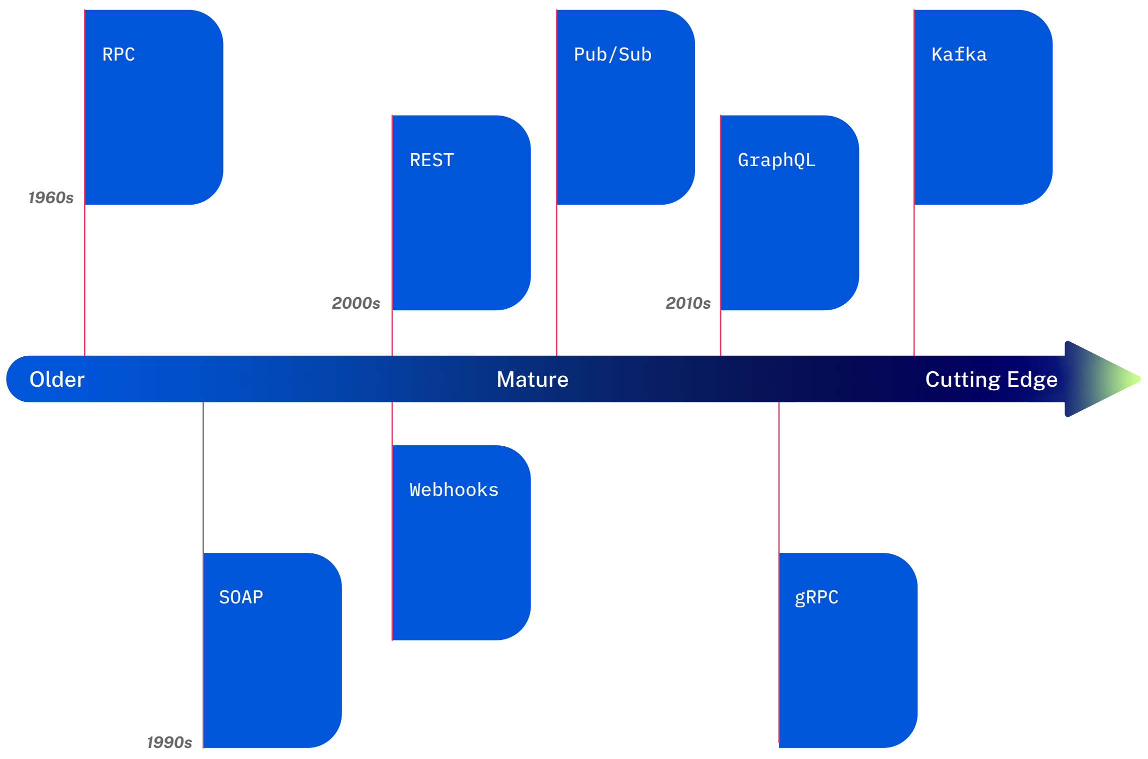
Then, you can generate the code for your application using the compiler. Once this is generated, you can focus on how the logic for the service works and integrate it into the application.

When to Use gRPC

gRPC is used to build low latency, scalable, distributed systems. One of the more common use cases for gRPC is in mobile application development, where its native streaming abilities, low latency, and high performance are beneficial.

gRPC Maturity

Over 60% of respondents in our Developer Survey reported being unfamiliar with gRPC. A mere 3.40% of respondents said they were using gRPC in production, the lowest of any item covered in the survey.



Protocol Buffers (Protobufs)

Protocol Buffers (protobufs) are “Google’s language-neutral, platform-neutral, extensible mechanism for serializing structured data — think XML, but smaller, faster, and simpler.”

As discussed in the previous section, Protobufs play an important role in gRPC and help minimize bandwidth and latency. However, they can also be used independently of gPRC.

How Protobufs Work

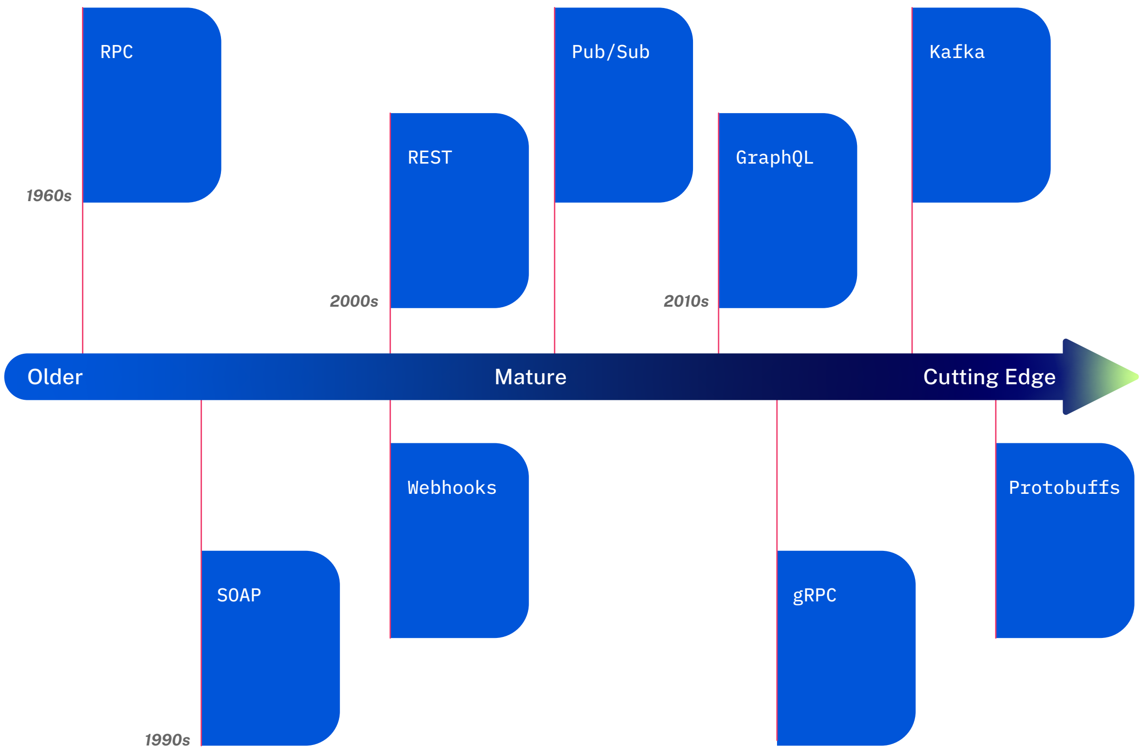
Defining protocol buffer message types in .proto files allows you to specify how the information is structured. The message format for protobufs is simple, each message has one or more uniquely numbered fields, and each field has a name and a value type. Once the messages are defined, the protocol buffer compiler is run to generate data access classes. These classes can then be used in your application.

When to Use Protobufs

Protobufs are similar to XML, but have a few unique advantages. Notably, they are simple to use, are 3 to 10 times smaller, and can be 20 to 100 times faster than XML. For this reason, protocol buffers are commonly used by tech companies — including their creator, Google — for both RPC systems and for persistent storage of data.

Protobuf Maturity

Much like gRPC, Protobufs are relatively new. Since they are typically used in conjunction with gRPC, they will likely become more widespread as gRPC matures.



A Next-Generation API Platform

As you can see based on the large variety of API styles, there is no single “best” type of API. Rather, the style you should use is dependent on your project goals and technical requirements.

Whatever APIs you decide to use, your development organization will need a centralized place to discover and connect to APIs. As part of a modern software development process, development organizations will need a flexible API platform that is runtime agnostic, supports both internal and external APIs, and accommodates multiple API types.

Although there are a variety of API runtime technologies and Gateways, a next-generation API platform that sits above your existing infrastructure. The next-generation API platform enables you to find, connect, and manage the APIs your organization is using now and those that you might want to use in the future as other API types become available.

A next-generation platform, like an API Hub, provides a centralized solution for helping developers as well as product managers, IT, and API creators to find, manage, and connect to all APIs — using a single key and SDK. A next-generation API Hub enables your organization to create new efficiencies, accelerating the software development process. Additionally, an API Hub provides management capabilities that enables you to govern and manage API consumption with enhanced visibility and control.

When choosing this next generation platform, ensure it includes the following criteria:

- **API Publishing:** Support for all standards and API types (OAS, GraphQL, etc.)
- **Discovery:** Search through all available APIs
- **Testing/Evaluation:** View documentation and test APIs easily
- **Provisioning:** Determine API access
- **Analytics and Metrics:** Obtain information about API users and their usage
- **Governance and Access Control:** Ability to provide fine-grained access to individuals and groups
- **Discussions/Support:** Get support for APIs and discuss enhancements

RapidAPI Enterprise Hub

RapidAPI is the world's largest API Marketplace, enabling millions developers to access more than 20,000 APIs using a single SDK, API Key, and API dashboard. RapidAPI Enterprise Hub is a white-labeled, internal API Marketplace used by developers, analysts, and product managers to discover and connect to internal APIs, as well as external API subscriptions.

The Enterprise API Hub is a white label solution and can connect to your internal systems, and is deployed seamlessly in the cloud. Once set up, engineering teams can publish their APIs into the hub for other developers to access and consume. RapidAPI supports all your APIs, regardless of what Gateways or API management solutions they use.

Your IT team can use a dedicated dashboard to manage who accesses the API and how the APIs get consumed. With the dashboard, they can access detailed analytics and monitoring information to ensure data security, compliance, and adherence to SLAs from a single dashboard.

Additionally, companies also use RapidAPI's Enterprise Hub as an external marketplace, enabling customers and partners to find, connect, and manage their APIs.