What is an API?

An API, which stands for *application programming interface*, is a set of protocols that enable different software components to communicate and transfer data. Developers use APIs to bridge the gaps between small, discrete chunks of code in order to create applications that are powerful, resilient, secure, and able to meet user needs. Even though you can't see them, APIs are everywhere—working continuously in the background to power the digital experiences that are essential to our modern lives.

Here, we'll give a high-level overview of the history of APIs and how they work before reviewing the different types of APIs and how they are used. We'll also discuss some common benefits and use cases for APIs—and offer a few real-world API examples that can help you get started.

What is the history of APIs?

In order to fully understand the role that APIs play in our lives, it's important to understand how they have evolved. APIs have been around for decades, with modern web APIs first taking shape in the early 2000s. The history of APIs since that period can be roughly broken down into the following five phases:

Phase 1: Commercial APIs

In the early 2000s, web APIs emerged as a new method for emerging startups to not only make products and services available online, but to also enable partners and third-party resellers to extend the reach of their platforms. This era of APIs was defined by Salesforce, eBay, and Amazon, and these companies continue to dominate the API playing field today.

Phase 2: Social media APIs

A shift in the API landscape occurred in the mid-2000s, as a new group of companies—such as Flickr, Facebook, and Twitter—realized that APIs could change the way we share information with one another. While these APIs weren't as intrinsically linked to revenue as their commercial predecessors, they nevertheless provided significant value to their organizations. For instance, Facebook launched version 1.0 of its API in August of 2006, which allowed developers to access Facebook users' friends, photos, events, and profile information. This API played a crucial role in establishing Facebook as one of the most popular social networks in the world.

Phase 3: Cloud APIs

In 2006, Amazon introduced Amazon Simple Storage (S3), which marked yet another turning point in the history of APIs. S3 is a basic storage service in which resources are accessible via API and CLI, and its pay-as-you-go model provides a cost-efficient way for organizations to monetize digital assets in the online economy. Just six months later, Amazon released Amazon Elastic Compute (EC2), which enabled developers to use web APIs to deploy infrastructure that would power the next generation of applications. Both S3 and EC2 continue to play an essential role in application development today.

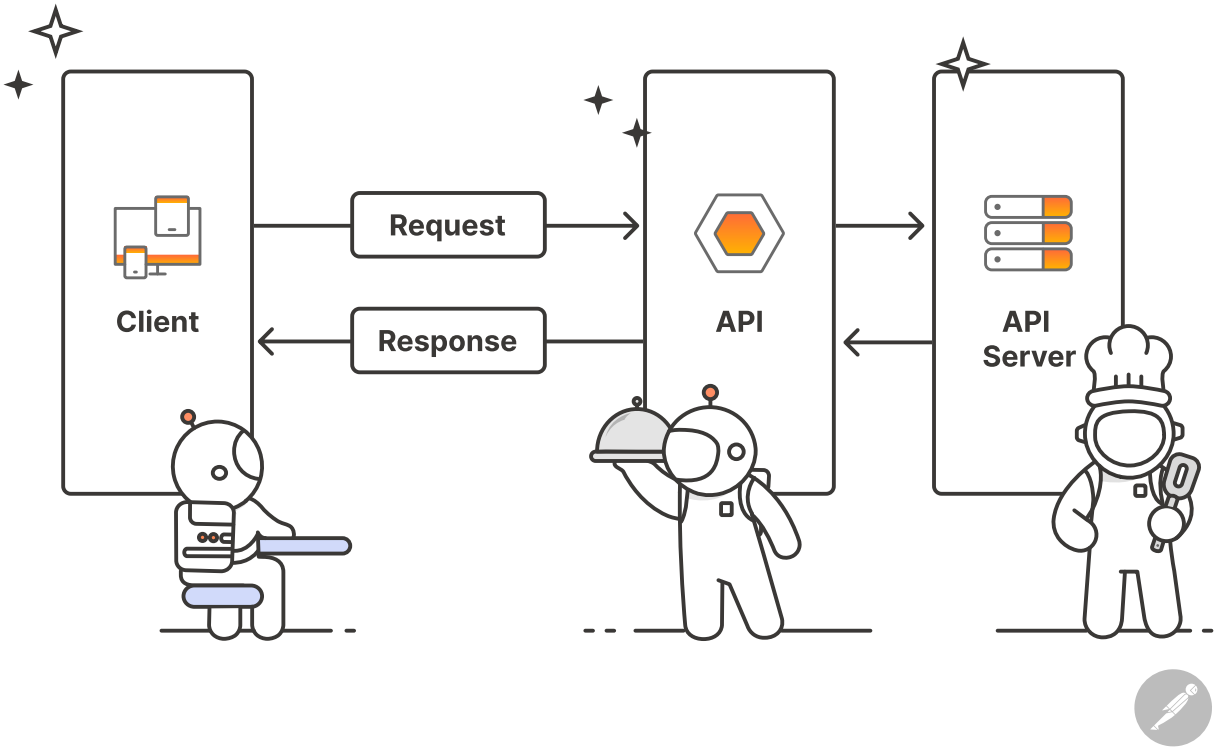
Phase 4: APIs for mobile applications

The world was introduced to Apple's iPhone and Google's Android in 2007. The ability to carry the web in our pockets radically changed how we live—and spurred a massive investment in mobile applications that are powered by APIs.

For instance, Twilio launched its API-as-a-product platform in 2007, which allowed developers to make and receive phone calls from any cloud application. Instagram then launched its photo-sharing iPhone application in October 2010, and it had one million users just three months later. Instagram did not initially provide an API, but it began work on one in early 2011 in response to user demand. These API-first companies played an essential role in creating the blueprint for how APIs are delivered today.

Phase 5: APIs for connected devices

Around 2010, some developers began using APIs to connect everyday objects—such as cameras, thermostats, speakers, microphones, and sensors—to the cloud. This next generation of devices, which includes Fitbit, Nest, Alexa, can send and receive data, content, media, and other digital resources, further changing the way we interact with the world around us.



How do APIs work?

APIs work by sharing data between applications, systems, and devices. This happens through a request and response cycle. The request is sent to the API, which retrieves the data and returns it to the user. Here's a high-level overview of how that process works.

1. API client

The API client is responsible for starting the conversation by sending the request to the API server. The request can be triggered in many ways. For instance, a user might initiate an API request by entering a search term or clicking a button. API requests may also be triggered by external events, such as a notification from another application.

2. API request

An API request will look and behave differently depending on the type of API, but it will typically include the following components:

- **Endpoint:** An API endpoint <https://blog.postman.com/what-is-an-api-endpoint/> is a dedicated URL that provides access to a specific resource. For instance, the /articles endpoint in a blogging app would include the logic for processing all requests that are related to articles.
- **Method:** The request's method indicates the type of operation the client would like to perform on a given resource. REST APIs are accessible through standard **HTTP methods**

<https://blog.postman.com/what-are-http-methods/>, which perform common actions like retrieving, creating, updating, and deleting data.

- **Parameters:** Parameters are the variables that are passed to an API endpoint to provide specific instructions for the API to process. These parameters can be included in the API request as part of the URL, in the query string, or in the request body. For example, the /articles endpoint of a blogging API might accept a "topic" parameter, which it would use to access and return articles on a specific topic.
- **Request headers:** Request headers are key-value pairs that provide extra details about the request, such as its content type or authentication credentials.
- **Request body:** The body is the main part of the request, and it includes the actual data that is required to create, update, or delete a resource. For instance, if you were creating a new article in a blogging app, the request body would likely include the article's content, title, and author.

3. API server

The API client sends the request to the API server, which is responsible for handling authentication, validating input data, and retrieving or manipulating data.

4. API response

Finally, the API server sends a response to the client. The API response typically includes the following components:

- **Status code:** HTTP status codes <https://blog.postman.com/what-are-http-status-codes/> are **three-digit codes** that indicate the outcome of an API request. Some of the most common status codes include 200 OK, which indicates that the server successfully returned the requested data, 201 Created, which indicates the server successfully created a new resource, and 404 Not Found, which indicates that the server could not find the requested resource.
- **Response headers:** HTTP response headers <https://blog.postman.com/what-are-http-headers/> are very similar to request headers, except they are used to provide additional information about the server's response.
- **Response body:** The response body includes the actual data or content the client asked for—or an error message if something went wrong.

In order to better understand this process, it can be useful to think of APIs like restaurants. In this metaphor, the customer is like the user, who tells the waiter what she wants. The waiter is like an API client, receiving the customer's order and translating it into easy-to-follow instructions for the kitchen—sometimes using specific codes or abbreviations that the kitchen staff will recognize. The kitchen staff is like the API server because it creates the order according to the customer's specifications and gives it to the waiter, who then delivers it to the customer.

What are the benefits of APIs?

APIs connect various software systems, applications, and devices by allowing them to communicate with one another. This unlocks many benefits, ranging from enhanced user experiences to increased business efficiency. The most common advantages of APIs include:

- **Automation:** APIs can be used to automate repetitive, time consuming work so that humans can focus on more complex tasks. This improves productivity, especially for developers and testers.
- **Innovation:** Public APIs can be used by external engineering teams, which spurs innovation and accelerates development by enabling developers to repurpose existing functionality to create new digital experiences.
- **Security:** APIs can provide an additional layer of protection against unauthorized breaches by requiring authentication and authorization for any request to access sensitive data.
- **Cost efficiency:** APIs provide access to useful third-party tools and infrastructure, which helps businesses avoid the expense of building complex in-house systems.

What are the different types of APIs?

There are many different types of APIs and ways to categorize them. For instance, you can categorize APIs by who has access to them. This organizational framework includes:

- **Private APIs:** Private APIs, also known as internal APIs, are used to connect different software components within a single organization, and they are not available for third-party use. For instance, a social media application might have a private API that handles the login workflow, another private API that handles the feed, and yet another private API that facilitates communication between users. Some applications may include dozens or even hundreds of private APIs.
- **Public APIs:** Public APIs provide public access to an organization's data, functionality, or services, which third-party developers can integrate into their own applications. Some public APIs are available for free, while others are offered as billable products. For instance, an e-commerce application may incorporate a public payment API, such as **Stripe <https://www.postman.com/stripedev/workspace/stripe-developers/overview>** , to handle payment processing without having to build that functionality from scratch.
- **Partner APIs:** Partner APIs enable two or more companies to share data or functionality in order to collaborate on a project. They are not available to the general public and therefore leverage authentication mechanisms to ensure they are only used by authorized partners.

What are the most common API architectural styles?

You can also categorize APIs according to their architectural style, of which there are many. The most frequently used architectural styles are:

1. REST

As discussed above, REST <https://blog.postman.com/rest-api-examples/> is the most popular API architecture for transferring data over the internet. In a RESTful context, resources are accessible via endpoints, and operations are performed on those resources with standard HTTP methods <https://blog.postman.com/what-are-http-methods/> such as GET, POST, PUT, and DELETE.

2. SOAP

SOAP <https://blog.postman.com/soap-api-definition/> , which stands for **Simple Object Access Protocol**, uses XML to transfer highly structured messages between a client and server. SOAP is often used in enterprise environments or legacy systems, and while it includes advanced security features, it can be slower than other API architectures.

3. GraphQL

GraphQL <https://blog.postman.com/what-is-a-graphql-api-how-does-it-work/> is an open source query language that enables clients to interact with a single API endpoint to retrieve the exact data they need, without chaining multiple requests together. This approach reduces the number of round trips between the client and server, which can be useful for applications that may run on slow or unreliable network connections.

4. Webhooks

Webhooks are used to implement event-driven architectures, in which requests are automatically sent in response to event-based triggers. For instance, when a specific event occurs in an application, such as a payment being made, the application can send an HTTP request to a pre-configured webhook URL with the relevant event data in the request payload. The system that receives the webhook can then process the event and take the appropriate action.

5. gRPC

RPC stands for **Remote Procedure Call**, and gRPC <https://blog.postman.com/what-is-grpc/> APIs were originated by Google. In gRPC architectures, a client can call on a server as if it were a local object, which makes it easier for distributed applications and systems to communicate with one another.

What is the API-first strategy?

The API-first strategy is an approach to software development in which applications are designed and built as a collection of internal and external services that are delivered through APIs. APIs are the building blocks of these applications, and the API-first strategy helps teams prioritize their quality, security, and performance.

What is the difference between SOAP APIs and REST APIs?

SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are two of the most common architectural styles for building APIs. SOAP APIs use XML and include built-in features for security and error handling, which makes them well-suited for enterprise environments with strict standards. On the other hand, REST APIs use JSON for resource representation, which is less verbose than XML. REST APIs are usually easier to understand, consume, and integrate than SOAP APIs, but they lack some of SOAP's advanced features. Learn more about the differences between SOAP and REST <https://blog.postman.com/soap-vs-rest/> .

What is the difference between APIs and webhooks?

Webhooks are lightweight callback functions that facilitate event-driven communication between APIs. In the traditional request-response cycle, an API client actively sends a request to an API server in order to retrieve data or perform actions. In contrast, a webhook listens for a specific event, such as a new user account being created or a payment being made, and performs a pre-configured action in response. This eliminates the need for the API client to poll the server, as the server will automatically perform the appropriate action or return the relevant data when the specified event occurs.