Application Programming Interface (API)

APIs, or application programming interfaces, make software development and innovation easier by allowing apps to simply and securely share data and functionality.

What is an application programming interface (API)?

Companies can open up their applications' data and functionality to external third-party developers, business partners, and internal departments through an application programming interface, or API. Through a specified interface, services and products can communicate and benefit from each other's data and functionality. Developers don't need to understand how an API works; they just utilise it to communicate with other products and services. API usage has exploded in recent years, to the point where many of today's most popular online apps would be impossible to create without them.

How an API works?

An API is a set of documented rules that describe how computers and applications interact. APIs serves as an intermediary layer between an application and a web server, processing data flow between systems.

Here’s how an API works:

1. **A client application initiates an API call:**To retrieve information, a client application makes an API call, often known as a request. This request, which contains a request verb, headers, and sometimes a request body, is sent from an application to the web server via the API's Uniform Resource Identifier (URI).
2. **After receiving a valid request:** The API makes a call to the external programme or web server after receiving a valid request.
3. **The server sends a*response:***The API makes a call to the external programme or web server after receiving a valid request.
4. **The API transfers the data:**The data is transferred from the API to the requesting application.

While the data communication method varies based on the online service, the requests and responses are all handled by an API. APIs are designed for computers or applications to use, whereas user interfaces are built for humans to use.

Because of their role as a middleman, APIs permit the abstraction of functionality between two systems—the API endpoint decouples the consuming application from the infrastructure that provides the service. To lessen the danger of server assaults, API calls normally include authorization credentials, and an API gateway can limit access to minimise security vulnerabilities. HTTP headers, cookies, and query string parameters provide extra security layers to the data throughout the exchange.

Consider the API provided by a payment processing service. Customers can enter their credit card information on the frontend of an ecommerce application. The payment processor does not need access to the user's bank account; instead, the API generates a unique token for this transaction and sends it along with the API call to the server. This provides a better level of protection against hacking attempts.

Why we need APIs?

You can use an application programming interface to simplify the process of managing existing tools or building new ones. The following are some of the most significant advantages of APIs:

* **Improved collaboration:** Nearly 1,200 cloud applications are used by the average company, with many of them being disconnected. APIs allow these platforms and apps to communicate with one another in a seamless manner. Companies can use this integration to automate procedures and boost workplace cooperation. Many businesses would be disconnected without APIs, resulting in information silos that jeopardise productivity and performance.
* **Easier innovation:** APIs provide flexibility by allowing businesses to connect with new business partners, provide new services to their existing customer base, and, ultimately, get access to new markets that can produce significant returns and accelerate digital transformation. Stripe, for example, began as a simple API with only seven lines of code. Since then, the company has worked with many of the world's largest corporations, expanded to offer loans and corporate cards, and has recently been valued at USD 36 billion.
* **Data monetization:** Many businesses prefer to provide APIs for free, at least at first, in order to build a developer community around their brand and establish ties with possible business partners. You may monetize an API that offers access to valuable digital assets by selling access (this is referred to as the API economy). AccuWeather's self-service developer portal, which sells a variety of API packages, attracted 24,000 developers in just 10 months, selling 11,000 API keys and forming a lively community in the process.
* **Added security:** APIs, as previously said, give an extra layer of security between your data and a server. Using tokens, signatures, and Transport Layer Security (TLS) encryption, as well as establishing API gateways to control and authenticate traffic and adopting good API administration, developers can further increase API security.

Common API examples

APIs have become a valuable component of modern business because they allow organisations to open up access to their resources while preserving security and control. Here are a few instances of application programming interfaces that you might come upon:

* **Universal logins:** The function that allows people to log in to websites using their Facebook, Twitter, or Google profile login data is a common API example. This useful feature enables any website to use an API from one of the more popular services to easily authenticate users, saving them the time and effort of creating a new profile for each new service or subscription.
* **Third-party payment processing:**The now-ubiquitous "Pay with PayPal" function on ecommerce websites, for example, is based on an API. This helps customers to pay for things online without revealing important information or giving unauthorised individuals access.
* **Travel booking comparisons:**The cheapest options for every date and destination are displayed on travel booking sites, which aggregate thousands of flights. This service is made possible via APIs, which give app users access to the most up-to-date availability data from hotels and airlines. APIs drastically minimise the time and effort required to search for available flights or lodging by allowing data and requests to be exchanged autonomously.
* **Google Maps:**The Google Maps service is one of the most well-known instances of an excellent API. The app uses different APIs and capabilities to offer users with directions or areas of interest in addition to the main APIs that display static or live maps. When planning travel routes or tracking goods on the move, such as a delivery van, you can interface with the Maps API using geolocation and numerous data layers.
* **Twitter:** An author, a unique ID, a message, a date when it was posted, and geographical metadata are all included in each Tweet's fundamental properties. Developers have access to public Tweets and replies, as well as the ability to post Tweets using Twitter's API.

Types of APIs

The majority of application programming interfaces (APIs) nowadays are web APIs, which expose an application's data and capabilities over the internet. The four primary types of web APIs are listed below:

* **Open APIs:** Open APIs are open source application programming interfaces that can be accessed via HTTP. They have established API endpoints and request and response forms, and are also known as public APIs.
* **Partner APIs:** Application programming interfaces (APIs) exposed to or provided by strategic business partners. Typically, developers can use a public API developer portal to access these APIs in self-service mode. They will still need to go through an onboarding procedure and obtain login credentials in order to use partner APIs.
* **Internal APIs:** Internal APIs are application programming interfaces that are not accessible to outside users. These internal APIs aren't open to the public and are intended to boost productivity and communication across different development teams within the firm.
* **Composite APIs:** Multiple data or service APIs can be combined. These services allow developers to make a single call to several endpoints. In a microservices architecture, composite APIs are beneficial when a single job requires data from multiple sources.

Types of API protocols

As the use of online APIs has grown, different protocols have emerged to give users with a set of established rules that specify the data types and commands that are acceptable. In practise, several API protocols allow for uniform data exchange:

* **SOAP:** SOAP (Simple Object Access Protocol) is an XML-based API protocol for sending and receiving data via SMTP and HTTP. It's easier to transfer data between apps or software components that run in various settings or are built in different languages thanks to SOAP APIs.
* **XML-RPC:** XML-RPC is a protocol that transfers data using a specified XML format, whereas SOAP uses a proprietary XML format. XML-RPC is older than SOAP, but it is significantly simpler and lighter in terms of bandwidth usage.
* **JSON-RPC:** JSON-RPC is similar to XML-RPC in that it uses JSON instead of XML to transport data. Each protocol is straightforward. While calls may have numerous parameters, only one result is expected.
* **REST:** There are no formal standards for (Representational State Transfer), which is a collection of web API architecture concepts (unlike those with a protocol). The interface must follow specific architectural limitations to qualify as a REST API (also known as a RESTful API). Although RESTful APIs can be built using SOAP protocols, the two standards are typically seen as competitors.

APIs, web services, and microservices

A web service is a piece of software that may be accessed via the internet. Web services, by definition, necessitate the use of a network. Every web service is an API since it exposes the data and functionality of an application. Not every API, however, is a web service.

API used to refer to an application's interface, which could have been written in any of the low-level programming languages, such as JavaScript. Developer-friendly interfaces that are easily accessible and widely understood by applications developed in Java, Ruby, Python, and many other languages arise from current APIs that adhere to REST principles and the JSON format, and are often created for HTTP.

Service-oriented architecture (SOA) and microservices architecture are two common architectural methods when leveraging APIs.

* **SOA** is a software design paradigm in which features are separated and made available as independent network services. Web services are commonly used to implement SOA, making the functional building blocks accessible via standard communication protocols. Developers can design these services from the ground up, but they commonly do so by exposing legacy system functionality as service interfaces.
* **Microservices architecture** is a type of architecture that separates a large application into smaller, self-contained components. It is easier to test, manage, and grow the application if it is implemented as a collection of distinct services. This methodology has gained popularity in the cloud computing era since it allows developers to work on one component without affecting the others.

While SOA was an important step forward in application development, microservices architecture is built to scale, allowing developers and businesses to create, modify, test, and deploy applications at a granular level, with shorter iteration cycles and more efficient use of cloud computing resources.

See "SOA vs. Microservices: What's the Difference?" for a more in-depth look at how these architectural methods differ.

APIs and cloud architecture

In today's world, it's critical to create APIs that are suitable for purpose. Connecting a microservices application architecture through your APIs to share data with external users, such as your clients, is important to cloud native application development.

Microservices design uses a common messaging mechanism, comparable to RESTful APIs, to enable open communication across an operating system without the need for additional integration layers or data conversion operations. You can also remove, replace, or improve any service or feature without affecting the other services. This lightweight dynamic allows for better API testing, performance, and scalability by optimising cloud resources.