What is an Application Programming Interface (API)

Application programming interfaces, or APIs, simplify software development and innovation by enabling applications to exchange data and functionality easily and securely.

Manage your APIs with IBM

What is an application programming interface (API)?

An application programming interface, or API, enables companies to open up their applications' data and functionality to external third-party developers, business partners, and internal departments within their companies. This allows services and products to communicate with each other and leverage each other's data and functionality through a documented interface. Developers don't need to know how an API is implemented; they simply use the interface to communicate with other products and services. API use has surged over the past decade, to the degree that many of the most popular web applications today would not be possible without APIs.

How an API works

An API is a set of defined rules that explain how computers or applications communicate with one another. APIs sit between an application and the web server, acting as an intermediary layer that processes data transfer between systems.

Here's how an API works:

While the data transfer will differ depending on the web service being used, this process of requests and response all happens through an API. Whereas a user interface is designed for use by humans, APIs are designed for use by a computer or application.

APIs offer security by design because their position as middleman facilitates the abstraction of functionality between two systems—the API endpoint decouples the consuming application from the infrastructure providing the service. API calls usually include authorization credentials to reduce the risk of attacks on the

server, and an API gateway can limit access to minimize security threats. Also, during the exchange, HTTP headers, cookies, or query string parameters provide additional security layers to the data.

For example, consider an API offered by a payment processing service. Customers can enter their card details on the frontend of an application for an ecommerce store. The payment processor doesn't require access to the user's bank account; the API creates a unique token for this transaction and includes it in the API call to the server. This ensures a higher level of security against potential hacking threats.

Why we need APIs

Whether you're managing existing tools or designing new ones, you can use an application programming interface to simplify the process. Some of the main benefits of APIs include the following:

- **Improved collaboration:** The average enterprise uses almost 1,200 cloud applications (link resides outside of IBM), many of which are disconnected. APIs enable integration so that these platforms and apps can seamlessly communicate with one another. Through this integration, companies can automate workflows and improve workplace collaboration. Without APIs, many enterprises would lack connectivity and would suffer from informational silos that compromise productivity and performance.
- **Easier innovation:** APIs offer flexibility, allowing companies to make connections with new business partners, offer new services to their existing market, and, ultimately, access new markets that can generate massive returns and drive digital transformation. For example, the company Stripe began as an API with just seven lines of code. The company has since partnered with many of the biggest enterprises in the world, diversified to offer loans and corporate cards, and was recently valued at USD 36 billion (link resides outside of IBM).
- **Data monetization:** Many companies choose to offer APIs for free, at least initially, so that they can build an audience of developers around their brand and forge relationships with potential business partners. However, if the API grants access to valuable digital assets, you can monetize it by selling access (this is referred to as the API economy). When AccuWeather (link resides outside of IBM) launched its self-service developer portal to sell a wide range of API packages, it took just 10 months to attract 24,000 developers, selling 11,000 API keys and building a thriving community in the process.
- **Added security:** As noted above, APIs create an added layer of protection

between your data and a server. Developers can further strengthen API security by using tokens, signatures, and Transport Layer Security (TLS) encryption; by implementing API gateways to manage and authenticate traffic; and by practicing effective API management.

Common API examples

Because APIs allow companies to open up access to their resources while maintaining security and control, they have become a valuable aspect of modern business. Here are some popular examples of application programming interfaces you may encounter:

- **Universal logins:** A popular API example is the function that enables people to log in to websites by using their Facebook, Twitter, or Google profile login details. This convenient feature allows any website to leverage an API from one of the more popular services to quickly authenticate the user, saving them the time and hassle of setting up a new profile for every website service or new membership.
- **Third-party payment processing:** For example, the now-ubiquitous "Pay with PayPal" function you see on ecommerce websites works through an API. This allows people to pay for products online without exposing any sensitive data or granting access to unauthorized individuals.
- **Travel booking comparisons:** Travel booking sites aggregate thousands of flights, showcasing the cheapest options for every date and destination. This service is made possible through APIs that provide application users with access to the latest information about availability from hotels and airlines. With an autonomous exchange of data and requests, APIs dramatically reduce the time and effort involved in checking for available flights or accommodation.
- **Google Maps:** One of the most common examples of a good API is the Google Maps service. In addition to the core APIs that display static or interactive maps, the app utilizes other APIs and features to provide users with directions or points of interest. Through geolocation and multiple data layers, you can communicate with the Maps API when plotting travel routes or tracking items on the move, such as a delivery vehicle.
- **Twitter:** Each Tweet contains descriptive core attributes, including an author, a unique ID, a message, a timestamp when it was posted, and geolocation metadata. Twitter makes public Tweets and replies available to developers and allows developers to post Tweets via the company's API.

Types of APIs

Nowadays, most application programming interfaces are web APIs that expose an application's data and functionality over the internet. Here are the four main types of web API:

- **Open APIs** are open source application programming interfaces you can access with the HTTP protocol. Also known as public APIs, they have defined API endpoints and request and response formats.
- **Partner APIs** are application programming interfaces exposed to or by strategic business partners. Typically, developers can access these APIs in self-service mode through a public API developer portal. Still, they will need to complete an onboarding process and get login credentials to access partner APIs.
- **Internal APIs** are application programming interfaces that remain hidden from external users. These private APIs aren't available for users outside of the company and are instead intended to improve productivity and communication across different internal development teams.
- **Composite APIs** combine multiple data or service APIs. These services allow developers to access several endpoints in a single call. Composite APIs are useful in microservices architecture where performing a single task may require information from several sources.

Types of API protocols

As the use of web APIs has increased, certain protocols have been developed to provide users with a set of defined rules that specifies the accepted data types and commands. In effect, these API protocols facilitate standardized information exchange:

- **SOAP** (Simple Object Access Protocol) is an API protocol built with XML, enabling users to send and receive data through SMTP and HTTP. With SOAP APIs, it is easier to share information between apps or software components that are running in different environments or written in different languages.
- **XML-RPC** is a protocol that relies on a specific format of XML to transfer data, whereas SOAP uses a proprietary XML format. XML-RPC is older than SOAP, but much simpler, and relatively lightweight in that it uses minimum bandwidth.
- **JSON-RPC** is a protocol similar to XML-RPC, as they are both remote procedure calls (RPCs), but this one uses JSON instead of XML format to transfer data. Both protocols are simple. While calls may contain multiple

parameters, they only expect one result.

- **REST** (Representational State Transfer) is a set of web API architecture principles, which means there are no official standards (unlike those with a protocol). To be a REST API (also known as a RESTful API), the interface must adhere to certain architectural constraints. It's possible to build RESTful APIs with SOAP protocols, but the two standards are usually viewed as competing specifications.

APIs, web services, and microservices

A web service is a software component that can be accessed via a web address. Therefore, by definition, web services require a network. As a web service exposes an application's data and functionality, in effect, every web service is an API. However, not every API is a web service.

Traditionally, API referred to an interface connected to an application that may have been created with any of the low-level programming languages, such as Javascript. The modern API adheres to REST principles and the JSON format and is typically built for HTTP, resulting in developer-friendly interfaces that are easily accessible and widely understood by applications written in Java, Ruby, Python, and many other languages.

When using APIs, there are two common architectural approaches—service-oriented architecture (SOA) and microservices architecture.

- SOA is a software design style where the features are split up and made available as separate services within a network. Typically, SOA is implemented with web services, making the functional building blocks accessible through standard communication protocols. Developers can build these services from scratch, but they usually create them by exposing functions from legacy systems as service interfaces.
- Microservices architecture is an alternative architectural style that divides an application into smaller, independent components. Applying the application as a collection of separate services makes it easier to test, maintain, and scale. This methodology has risen to prominence throughout the cloud computing age, enabling developers to work on one component independent of the others.

While SOA was a vital evolutionary step in application development, microservices architecture is built to scale, providing developers and enterprises with the agility and flexiblity they need to create, modify, test, and deploy applications at a granular level, with shorter iteration cycles and more efficient

use of cloud computing resources.

For a deeper dive on how these architectural approaches relate, see "SOA vs. microservices: What's the difference?"

APIs and cloud architecture

It's crucial to develop APIs fit for purpose in today's world. Cloud native application development relies on connecting a microservices application architecture through your APIs to share data with external users, such as your customers.

The services within microservices architecture utilize a common messaging framework, similar to RESTful APIs, facilitating open communication on an operating system without friction caused by additional integration layers or data conversion transactions. Furthermore, you can drop, replace, or enhance any service or feature without any impact on the other services. This lightweight dynamic improves cloud resources optimization, paving the way for better API testing, performance and scalability.

APIs and IBM Cloud®

APIs will continue to be just one part of application modernization and transforming your organization as the demand for better customer experiences and more applications impacts business and IT operations.

When it comes to meeting such demands, a move toward greater automation will help. Ideally, it would start with small, measurably successful projects, which you can then scale and optimize for other processes and in other parts of your organization.

Working with IBM, you'll have access to AI-powered automation capabilities, including prebuilt workflows, to help accelerate innovation by making every process more intelligent.

Take the next step:

- Explore IBM API Connect®, an intuitive and scalable API design platform to create, securely expose, manage and monetize APIs across cloud computing systems.
- Build skills to help you create developer communities to publish and share APIs and engage with them through a self-service portal in the Solution

Developer: IBM API Connect curriculum.

- API Connect can also come integrated with other automation capabilities in IBM Cloud Pak® for Integration, a hybrid integration solution that provides an automated and closed-loop lifecycle across multiple styles of enterprise integration.
- For Business to Business API connections explore the IBM Sterling Supply Chain Business Network B2B API Gateway for secure connections between you, your customers, and your partners.
- Take our integration maturity assessment to evaluate your integration maturity level across critical dimensions and discover the actions you can take to get to the next level.
- Download our agile integration guide, which explores the merits of a container-based, decentralized, microservices-aligned approach for integrating solutions.

Get started with an IBM Cloud account today.

A client application initiates an API call to retrieve information—also known as a request. This request is processed from an application to the web server via the API's Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.**After receiving a valid request**, the API makes a call to the external program or web server.**The server sends a** responseto the API with the requested information.**The API transfers the data** to the initial requesting application.