

What Is an API (Application Programming Interface)?

Michael Chen | Senior Writer | February 24, 2025



In This Article

- [What Is an API?](#)
- [APIs Explained](#)
- [How APIs Work](#)
- [Why Are APIs Important?](#)
- [API Components](#)
- [Benefits of APIs](#)
- [Challenges of APIs](#)
- [Common API Mistakes](#)
- [Types of APIs](#)
- [Examples of APIs](#)
- [API Use Cases](#)
- [API Protocols](#)
- [API Integrations](#)
- [APIs & Digital Transformation](#)
- [API Trends](#)
- [How Oracle Helps](#)
- [API FAQs](#)

The term “API” stands for application programming interface. APIs act as bridges between applications, letting them communicate and share data. For example, a dashboard that a marketing team uses to manage multiple social media accounts depends on APIs, which connect the company’s social platforms to the dashboard display and pull in relevant data.

A typical internet user constantly benefits from APIs, often without realizing it. APIs connect public data sources, such as weather forecasting sites, to commercial apps to warn us about upcoming storms. Developers regularly access the Google Maps API to embed maps and location services into their websites. Retailers use API-powered payment gateways such as PayPal and Stripe to securely handle financial transactions with customers.

What Is an API?

An API, or application programming interface, is a set of rules and protocols that allows applications to exchange data, perform actions, and interact in a well-documented way. When a request is made—for a weather update, say—the API processes the request, executes the necessary actions, and returns a response, typically in a standard format such as those defined by [JSON](#) or XML.

Key Takeaways

An API is a middleman that allows two software programs to talk to each other and defines how they request and receive data or functionality.

APIs are essential for building modern software applications that connect and share information.

APIs play a crucial role in enabling use of cloud services by allowing them to integrate and share data with on-premises software.

APIs Explained

With APIs, developers can access software platforms and services natively from the applications they're building. Without APIs, data would have to be manually exported from one application, prepared and transformed, and then manually imported to another application every single time a user wanted to check the weather or respond to a comment on a social media site.

In simple terms, there are three parties involved in the exchange process, as follows:

Client: The party that makes the request

Server: The party that fulfills the request

API: The intermediary that connects the two in a well-documented, predictable way

Think about a restaurant. If customers all walked into the kitchen to order their favorite dishes, chaos would ensue. In this scenario, the API provides a menu (documentation) that lists all the services (dishes) the kitchen (server application) can offer. It explains what information you as a client needs to provide and what format your order should be presented in.

The API acts as a waiter, or go between, making sure that orders are taken and delivered in a standardized way.

How APIs Work

APIs work by specifying how software components should interact, making it possible for developers to integrate different systems and share data and functionality without needing to build everything from scratch, saving time and resources. APIs typically define the methods and protocols that must be used for communication, as well as the data formats that may be exchanged.

An API defines how applications interact by providing details including:

Endpoints. Specific URLs that define where to send data and requests.

Methods. Instructions such as GET to retrieve data, POST to send data, PUT to update data, and DELETE to remove data.

Parameters. Specific details needed for the request, such as location for weather data or login credentials for social media.

Responses. The format of the data sent back by the application, such as JSON or XML.

The developer of the client application requesting data writes code to make an API call. This code specifies:

The API endpoint URL

The HTTP method

Any necessary parameters

The application sends requests to the server application's API gateway, which manages incoming requests. The API gateway routes the request to the appropriate service within the target application. The service processes the request and retrieves the data or performs another desired action.

The target service then prepares the response data according to the API definition and sends it back through the API gateway to the requesting application, which receives and parses the data and delivers the expected result to the end user.

Why Are APIs Important?

Because they provide a standardized way for developers to access other applications' and services' data and functionality, APIs let companies avoid recreating the proverbial wheel. That saves time and money. Standardization also fosters both innovation and scalability by enabling the modular addition of new features and services without disrupting the operation of existing systems.

On a business level, APIs are critical in that they allow companies to automate repetitive tasks and processes by enabling software to interact directly with other software. Given that most businesses are working to add automation to free up employees for higher-level tasks, the ability of APIs to reduce manual workloads and increase operational efficiency is a key benefit. Organizations looking to increase use of cloud services also depend heavily on APIs.

API Components

API components work together to allow different software systems to communicate and exchange data and functionality. Understanding these components is essential for successfully integrating APIs into your software. API components include the following:

An API specification provides a structured description of what the API will do and exactly how to interact with it.

The API designer is a utility that helps developers create APIs. An [API designer](#) can be as simple as a plugin for a development environment or a highly specialized tool. The goal is to have built in rules for validating and formatting the API, to save time and aggravation.

API portals are where developers find and share published APIs and peruse specifications to understand if an API can help them and how it may be used. The portals for public APIs are often embedded within a website of supporting material, such as legal terms and conditions.

The API backend is the software that translates an API call into action for the client.

API gateways provide the URL for an API, apply rules that govern the use of that API, and direct the API call to the relevant backend. Typically, a gateway will know both the API specification and details of the rules it should apply. Rules may address authentication and authorization, certificate management, rate limiting and throttling, payload inspection and validation, intelligent routing based on the header or payload content, and more.

APIs may also include rate limiting, error handling, and documentation for developers. [Writing a solid API](#) involves a series of decisions, from architectural style to design tools, and is an invaluable skill for organizations eyeing a cloud-native future.

Benefits of APIs

By using APIs, developers can connect distributed applications—for example, a smartphone application to a social media website, or a payroll system to a business bank account. Because APIs enable the building of handy applications from small, individual, connected services, they pay benefits in robustness and scalability.

If one service breaks, most of the app can carry on. Additional benefits include:

Agility. APIs let developers select the very best technology for each problem that needs to be solved.

Faster development. APIs allow developers to plug in existing functionality instead of building everything from scratch.

Innovation. APIs foster collaboration and experimentation by allowing developers to discover new services and try them out without a huge amount of investment.

More control. APIs can be outfitted with strict authorization controls to granularly limit what data or actions an application can access.

Scalability. APIs enable applications to handle increased demand by outsourcing tasks to other services. For example, instead of maintaining its own payment processing system, a small retailer can opt for a payment API such as Stripe or PayPal. This offloads a complex task. Now the seller can focus on growing its core business while leaving payment processing to the experts and increasing customer confidence.

Challenges of APIs

For all the upsides of APIs, there are challenges around complexity, cost, and security to consider when designing applications that make use of API calls, and when building your own APIs. Software that depends

on multiple APIs can become difficult to manage and maintain—especially if the API provider makes frequent updates or changes.

Specific challenges to tackle include:

API selection. With the huge number of APIs available, choosing the best fit can be a daunting task. In fact, there might not be a single, perfect API—you might need to cobble together data and functionality from multiple sources.

Cost. While many APIs are free to use, watch for limits on calls and features. Depending on your application and audience, you may require paid subscriptions for certain functions or capacity. Will you pay a flat or usage-based fee? The ongoing cost of maintaining your API connections also needs to be factored into architectural decisions.

If you use a significant number of APIs, or have major volume on a few, look for an [API usage plan](#) to keep costs under control.

Integration complexity. Even after finding a suitable API, integrating it with your application can be a complex job. APIs from different providers may have different protocols, data formats, and authentication mechanisms. Bridging the gaps can require significant development effort.

Performance. The performance—or not—of an API can frustrate people using your application. APIs can introduce latency that causes slow response times and bogs down data processing. Remember, the employee or customer won't blame the API provider. It's your company's name on the application.

Security. Making APIs easier to discover increases the risk of misuse, so companies need to be mindful of security. Fortunately, with the right tools, creating secure APIs is reasonably straightforward. Authentication mechanisms, such as API keys, tokens, or other credentials, can make sure only authorized applications access systems. Be sure to review the API's data encryption standards. In addition, a well-designed API will conceal how its backend is implemented, allowing teams to make changes without adversely impacting the client.

Vendor lock-in. Relying heavily on a specific API provider for functionality important to your application can lock you into that ecosystem. If you wish to switch API providers in the future, that can be an expensive and disruptive operation.

Versioning issues. Like most software, APIs are not static. They evolve to add new functionality and address security and technical changes. New versions might introduce code changes that disrupt your application. And even if there are no malfunctions, keeping a record of the different API versions and integrations in use can be a major burden.

Related, not all API developers issue the clear and comprehensive documentation that's essential for your developers to use and integrate an API, so choose provider partners carefully.

Common API Mistakes

For those looking to develop APIs, there are some gotchas, particularly around choice of specifications and underestimating demand. A tenet of good API design is to abstract and protect the consumer from changes to how you implement the backend. Your API design directly reflects the underlying data storage, for example, so if the internal data structures change, the API is impacted, which may disrupt the API client.

Other mistakes to avoid include:

Poor documentation. Clear, detailed documentation is essential to the success of your API. For example, when you describe a date, you need to be clear on format. In Europe, a date is typically represented as day, month, then year, while in North America, the order is month, day, year. Not being explicit in such details can result in data quality problems and, worst case, your API breaking an application.

Not considering production data volumes. During APIs development, testing uses relatively small datasets. In production, data volumes are often far larger, resulting in API calls trying to communicate massive amounts of data in a single request. This can result in a range of problems, depending on the network between the client and backend. Worst case, the request may place excess demand on the API backend, which may result in the API call failing.

Mistakes may also be made when setting policies for the API gateway. These errors typically involve not providing sufficient security, which can allow malicious actors to change or improperly access data or even use the API as a way to attack the network. These kinds of issues are analyzed and collected by OWASP Foundation, with the most common mistakes being reported in their well-known [Top 10 API Security Risks list](#).

Confusing the roles of an API gateway and the API backend is another common mistake. Both capabilities need to process APIs as they're received, and it's easy to mix the two elements together. However, the gateway's job is to screen and route requests to the right place very quickly. The API backend will need longer to process each request as it's delivering business logic.

Remember, the relationship between API calls and the API backend isn't one-to-one.

Types of APIs

There are four main types of APIs. Which you choose will depend on your use case. Consider near- and longer-term plans for the application before settling on a model—swapping in a different API is doable but increases costs and complexity.

Public APIs can be used by anyone to access a server's data or other services from a client application. Common uses of public APIs include retrieving traffic and weather data and managing third-party login processes. Public APIs are generally intended to let any application use a service. This access could be a simple action, such as retrieving the current time, or something more complex, like retrieving a weather radar image or detailed list of directions from Point A to Point B. Because public APIs tend to be widely used, great care is taken not to change them unless absolutely necessary so as not to break the functionality of applications.

Private APIs are developed for internal use only and aren't widely published. Typically, private APIs let a vendor's applications communicate with that vendor's servers. For example, the banking application on your phone uses private APIs to access the unique services of your specific bank.

Partner APIs are developed for use between specific organizations. The details of the API are disclosed to a limited set of partners. For example, a cloud database platform may agree to partner with a set number of analytics providers. That will lead to partner APIs that efficiently [connect the database with the analytics platforms](#).

Composite APIs are chained together for a particular function and might be a combination of public, private, and partner APIs. An example of a chained API using public and private APIs is the integration between a weather app and a fitness tracker app. The weather app's public API provides data such as temperature and humidity to the fitness tracker app. Its private API brings data on the owner's pace and distance covered and combines it with environmental factors to calculate calories burned.

Examples of APIs

Most people are familiar with consumer APIs, such as for weather or location. But there's a universe of sophisticated APIs that enable enterprises to take advantage of functionality from cloud services to databases to robust business applications.

For example, Oracle offers a wide range of APIs across its services. Companies using Oracle Cloud Infrastructure (OCI) can leverage APIs for programmatic management of virtual networks, including creating, configuring, and managing subnets, security lists, and route tables. A Compute API enables admins to start, stop, reboot, and configure compute instances in OCI. Other APIs connect IT teams with object storage and identity and access management functions.

Innovative startups are using APIs, too. For example, Inworld.ai offers AI-driven virtual characters for role-playing online games. Using APIs, developers can create nonplayable characters (NPCs) that interact with players in a realistic and engaging way. The APIs let game designers specify a character's attributes, personality, and behaviors, allowing them to customize NPCs to add depth and variety to their games. Virtual characters can comprehend and respond to text or voice inputs, all via APIs.

From Dominoes' use of APIs to integrate with voice assistants so customers can order pizza without touching a device to Uber using APIs to link to real-time data and dynamically adjust ride prices based on demand and traffic conditions, this technology is driving real innovation now.

API Use Cases

For the average person, the APIs that enable social media integration and payment processing will be familiar. Many websites and applications use APIs to enable popular social media functions, like sharing content, while e-commerce platforms use APIs to connect with payment gateways like Stripe or PayPal.

But that's not the only way APIs make our everyday lives easier. They enable the geolocation services used by apps that provide ride-sharing or food delivery services that depend on mapping APIs to find the location of a customer's home or destination.

On the business side, API use cases include allowing teams to interact with cloud resources, such as the applications they use for financial or customer service functions. APIs are also what power communication and data exchange between IoT devices and their control systems.

If you work in a smart office where lights and temperature are adjusted automatically, that's an API use case.

API Protocols

There are several protocols, or architectural styles, for exposing APIs to developers. These approaches let the developer know how they should expect a set of APIs to function and generally what mechanism they'll use to access the API from their own programs.

Common architectural styles include the following:

Representational state transfer (REST)

This is perhaps the most popular architecture for accessing resources and services on the web. In many environments, a client goes through processes that change its state relative to the server. For instance, if you want to know your bank balance, you'll need to go from an unauthenticated state to an authenticated one. The server and client then maintain that authenticated state once it's established. By contrast, REST APIs are stateless. If a developer wants to use a REST API to check a bank balance, the request will need to include enough information to authenticate the user making the request. Once the request has been processed, no state information is retained. If the user wants to make another similar request, they must again provide authentication information with the request. One benefit of REST APIs is that servers don't need to track the state of clients, which can greatly simplify the server's architecture.

Remote procedure calls (RPCs)

In traditional applications, procedure calls, sometimes called function calls, are used to access the devices and services of the computer on which the application is running. Opening and reading files or writing to the computer's display or other devices are functions handled through procedure calls. In this way, the operating system provides a layer of abstraction between the application and the actual hardware of the computer. Application programmers don't need to know anything about the computer's display; they just use a procedure call. In the same way, procedure calls can let an application use a resource on a network. Perhaps the user's files aren't on the local computer, but instead on a network server. A remote procedure call gets the job done. Many times, the application won't know whether the resource it wants to use is local or remote. The operating system figures it out and takes the appropriate steps to fulfill the request. Generally, RPCs can use any format to access a function—conventions for how the calls work are typically the domain of the operating system.

Operating system calls are just one type of RPC. Other types can be developed to do almost anything. A company might, for instance, choose to create its own application to track worker hours. Developers could use basic networking functions to create procedures that let mobile apps report check-ins or check-outs to central servers. Various libraries make this development easier, while using a standard architecture such as REST can be helpful since other developers will be more likely to understand how the RPC works.

Simple object access protocol (SOAP)

Like REST, SOAP provides a way to access services on the internet. It uses XML to define how requests are formatted and can run on a wide variety of transport protocols, which means it can be vendor-agnostic. SOAP is most commonly used to access web services, with HTTP acting as the transport layer. If an application wished to retrieve a product description, it would create the proper XML document and send it to a web server that knows about the product. The web server would send back its own XML document, including the product information requested. Since SOAP is intended to retrieve objects, actions are limited to GET, POST, PUT, and DELETE, making the verb structure of the protocol extremely simple.

API Integrations

API integrations connect applications and allow them to exchange data and functionality. Imagine integrations as phone lines enabling open back-and-forth communications.

There are three components involved.

The APIs themselves provide the rules and specifications dictating how applications can communicate. APIs define what data can be exchanged, how to format it, and what actions may be triggered.

The server application exposes its functionalities or data via an API. For instance, a cloud service might offer an API that helps IT teams quickly spin up new instances or add seats.

The client application uses APIs to request data or functionality from the server application. A rideshare app, for example, uses a weather service's API to adjust prices when it's raining or above or below certain temperature thresholds.

The actual process involves a few steps, beginning with the developer of the client application selecting a suitable API. The client uses API keys, tokens, or other credentials to authenticate with the desired API and obtain authorization to access specific data or actions. It then makes requests, or calls, to the server's API requesting the exact data or action desired.

The serving application processes the request and, if authorized, performs the action or retrieves the data and sends it back to the client, via the API, in a structured format, such as JSON or XML.

APIs & Digital Transformation

Digital transformation is all about the cloud, and APIs are a cornerstone of cloud-native architectures. APIs allow for integration of services and systems within the cloud, and they let enterprises connect legacy applications with their new cloud services, thus enabling a gradual transition into a digital future without disrupting operations. And with APIs, enterprises can quickly respond to market changes and opportunities. Think about building modern services, such as payment gateways, social media platforms, and analytics tools, directly into your applications.

Another transformative, and API-driven, technology is microservices, an architectural approach to modern application development favoring independent services and functions. In a microservices architecture, an application is broken up into contained building blocks that efficiently perform a single task. Microservices communicate with other applications or services using APIs. An application could have just a few microservices, or it could be made up of hundreds or even thousands of moving parts. Microservices-based applications scale faster by keeping individual elements independent. This provides the agility and flexibility required for digital transformation initiatives that may be hindered by the monolithic architectures used in legacy software development.

Cloud native companies that **embrace microservices** can move quickly to seize new opportunities and embrace automation. APIs underpin that strategy.

API Trends

Companies are developing APIs to power the continued growth of cloud, artificial intelligence (AI) and machine learning (ML), with requests and results being communicated through APIs. But APIs themselves are also seeing advancements.

One example is reverse (egress) gateways. Typically, an API gateway's job is to field incoming requests. However, there are situations where an API gateway operates as a specialized internet proxy, providing infrastructure to direct application network traffic while allowing IT teams to monitor what's happening. When the only way for traffic to leave a network is through the API gateway, using approved external APIs, IT can audit packets leaving the network and better understand what's happening with their data.

API gateways can also track the use of chargeable services, ensuring that the service is being used properly and that revenue is coming in as expected. Additionally, gateways can help test new versions of applications before deployment. During testing, calls may need to be directed depending on whether the request originated from the production system or the one being tested. The gateway can derive the necessary information and perform the redirection.

New API rate limiting and throttling techniques help IT teams manage API access and prevent attacks. Rate limiting restricts the number of API calls an application can make within a specific timeframe, while throttling dynamically adjusts access based on factors like server load or whether activity is tagged as possibly malicious.

Other key trends and standards that are shaping the API landscape include a focus on openness and interoperability, without sacrificing security. Companies should keep an eye on trends including:

OpenAPI Initiative. The [OpenAPI Initiative, or OAS](#), is a consortium of organizations working on a common language for describing APIs. This standardized format will hopefully add up to better documentation, discovery, and integration.

JSON Web Tokens. Another open standards effort, [JWTs](#) are becoming a popular method for authentication and authorization in APIs. They provide a secure way for applications to exchange user identity information without storing sensitive data on the API server itself.

OAuth 2.0. Because security is as critical as standardization, [OAuth 2.0](#), the industry-standard protocol for authorization, is being widely adopted. The OAuth 2.0 framework allows applications to access user data on another application's platform without requiring users to share their passwords. This gives individuals and companies more control over their data.

[OpenID Connect](#), or OIDC, builds on OAuth 2.0 and allows applications to verify a user's identity without needing separate login credentials for each application.

Finally, for companies heavily dependent on solid API performance, [API analytics](#) is an emerging must-have. By proactively monitoring and analyzing API use patterns, IT can identify performance bottlenecks, security risks, and opportunities for improvement. Look for [an API manager](#) that offers tools to gather and drill down into API usage data. Measuring which APIs are being used the most can help organizations determine where to make investments. For example, if the APIs for an aging ERP solution are getting more use than those for the CRM, it may make sense to prioritize improving or replacing the ERP.

How Oracle Helps

Oracle Cloud Infrastructure (OCI) provides [a comprehensive set of services](#) to manage the lifecycle of APIs. Built-in tools make it easy for developer teams to collaborate on prototyping, testing, and validating APIs. The Oracle Cloud Infrastructure API Gateway provides integration, acceleration, governance, and security for API and SOA-based systems, enabling teams to manage and deliver web APIs securely. In addition, usage plans and subscriptions allow API operators to monitor and monetize APIs.

Once a development team understands how APIs work, they gain insight into the hidden connections that power many of the applications and services their customers and employees use every day. Now, developers can build applications faster, better, and for less by tapping data and functionalities exposed through APIs rather than building everything from scratch.

Finance applications are major, and demanding, use cases for APIs. They can help CIOs help CFOs deliver systems that delight both employees and customers. Here are other ways to help streamline core financial processes.

[Access the ebook](#)

API FAQs

What are the four types of APIs?

The four types of APIs are public (anyone can use them), private (internally developed within an organization), partner (developed to work between software from involved organizations), and composite (various types of APIs used together).

What is an example of an API in real life?

A good example of a public API provider is NASA, which provides APIs to share research data, imagery, and event tracking information. These APIs allow developers to get a feed of selected NASA data—such as Mars Rover updates or details about NASA-tracked natural events such as volcanic eruptions—and integrate it

into their own applications. For example, a weather app might integrate Mars Rover updates into a special section promoted as a "Live from Mars" feed for users to check out.

Is creating an API easy?

Writing an API can be a straightforward process, particularly for experienced developers. APIs can be coded in almost any programming language, and existing architectures such as REST provide established guidelines to work with. A simple way to learn API development is to reverse-engineer public open source APIs to see how their architects created them.

What is REST API in simple terms?

REST, sometimes referred to as RESTful, stands for “representational state transfer” and is a standard protocol used for developing web services. REST provides a set of rules and guidelines to allow different applications to communicate over the internet in a scalable and efficient manner. REST defines how applications make requests—typically GET, PUT, POST, and DELETE—via HTTP using HTML, XML, Python, JSON, PHP, or plain text without relying on establishing a stateful relationship between the client and the server.