Red Hat

INTEGRATION

What is an API?

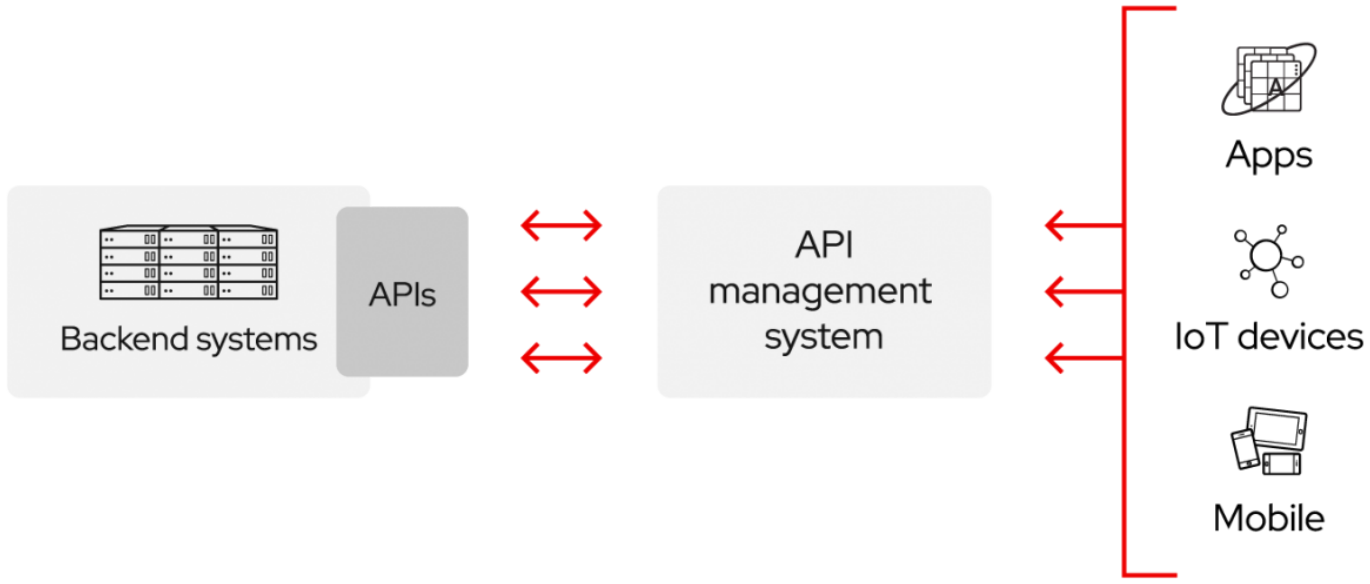
An API is a set of definitions and protocols for building and integrating application software. API stands for application programming interface.

APIs let your product or service communicate with other products and services without having to know how they're implemented. This can simplify app development, saving time and money. When you're designing new tools and products—or managing existing ones—APIs give you flexibility; simplify design, administration, and use; and provide opportunities for innovation.

APIs are sometimes thought of as contracts, with documentation that represents an agreement between parties: If party 1 sends a remote request structured a particular way, this is how party 2's software will respond.

Because APIs simplify how developers integrate new application components into an existing architecture, they help business and IT teams collaborate. Business needs often change quickly in response to ever shifting digital markets, where new competitors can change a whole industry with a new app. In order to stay competitive, it's important to support the rapid development and deployment of innovative services. Cloud-native application development is an identifiable way to increase development speed, and it relies on connecting a microservices application architecture through APIs.

APIs are a simplified way to connect your own infrastructure through cloud-native app development, but they also allow you to share your data with customers and other external users. Public APIs represent unique business value because they can simplify and expand how you connect with your partners, as well as potentially monetize your data (the Google Maps API is a popular example).



For example, imagine a book-distributing company. The book distributor *could* give its customers a cloud app that lets bookstore clerks check book availability with the distributor. This app could be expensive to develop, limited by platform, and require long development times and ongoing maintenance.

Alternatively, the book distributor could provide an API to check stock availability. There are several benefits to this approach:

- Letting customers access data via an API helps them aggregate information about their inventory in a single place.
- The book distributor can make changes to its internal systems without impacting customers, so long as the behavior of the API doesn't change.
- With a publicly available API, developers working for the book distributor, book sellers or third parties could develop an app to help customers find the books they're looking for. This could result in higher sales or other business opportunities.

In short, APIs let you open up access to your resources while maintaining security and control. How you open access and to whom is up to you. API security is all about good API management, which includes the use of an API gateway. Connecting to APIs, and creating applications that consume the data or functionality exposed by APIs, can be done with a distributed integration platform that connects everything—including legacy systems, and the Internet of Things (IoT).

There are three approaches to API release policies.

Private

The API is only for use internally. This gives companies the most control over their API.

Partner

The API is shared with specific business partners. This can provide additional revenue streams without compromising quality.

Public

The API is available to everyone. This allows third parties to develop apps that interact with your API and can be a source for innovation.

Innovating with APIs

Exposing your APIs to partners or the public can:

- Create new revenue channels or extend existing ones.
- Expand the reach of your brand.
- Facilitate open innovation or improved efficiency through external development and collaboration.

Sounds great, right? But how can APIs do all that?

Let’s return to the example of the book distributing company.

Suppose one of the company's partners develops an app that helps people find books on bookstore shelves. This improved experience brings more shoppers to the bookstore—the distributor's customer—and extends an existing revenue channel.

Maybe a third party uses a public API to develop an app that lets people buy books directly from the distributor, instead of from a store. This opens a new revenue channel for the book distributor.

Sharing APIs—with select partners or the whole world—can have positive effects. Each partnership extends your brand recognition beyond your company's marketing efforts. Opening technology to everyone, as with a public API, encourages developers to build an ecosystem of apps around your API. More people using your technology means more people are likely to do business with you.

Making technology public can lead to novel and unexpected outcomes. These outcomes sometimes disrupt entire industries. For our book distributing company, new firms—a book borrowing service, for example—could fundamentally change the way they do business. Partner and public APIs help you use the creative efforts of a community larger than your team of internal developers. New ideas can come from anywhere, and companies need to be aware of changes in their market and ready to act on them. APIs can help.

An extraordinarily brief history of APIs

APIs emerged in the early days of computing, well before the personal computer. At the time, an API was typically used as a library for operating systems. The API was almost always local to the systems on which it operated, although it sometimes passed messages between mainframes. After nearly 30 years, APIs broke out of their local environments. By the early 2000s, they were becoming an important technology for the remote integration of data.

Remote APIs

Remote APIs are designed to interact through a communications network. By "remote," we mean that the resources being manipulated by the API are somewhere outside the computer making the request. Because the most widely used communications network is the internet, most APIs are designed based on web standards. Not all remote APIs are web APIs, but it's fair to assume that web APIs are remote.

Web APIs typically use HTTP for request messages and provide a definition of the structure of response messages. These response messages usually take the form of an XML or JSON file. Both XML and JSON are preferred formats because they present data in a way that's easy for other apps to manipulate.

What's been done to improve APIs?

As APIs have developed into the now-ubiquitous web API, several efforts have been made to make their design a little easier and their implementation more useful.

A little SOAP, a lot of REST

As web APIs have spread, a protocol specification was developed to help standardize information exchange: Simple Object Access Protocol, more casually known as SOAP. APIs designed with SOAP use XML for their message format and receive requests through HTTP or SMTP. SOAP makes it easier for apps running in different environments or written in different languages to share information.

Another specification is Representational State Transfer (REST). Web APIs that adhere to the REST architectural constraints are called RESTful APIs. REST differs from SOAP in a fundamental way: SOAP is a protocol, whereas REST is an architectural style. This means that there's no official standard for RESTful web APIs. As defined in Roy Fielding's dissertation "Architectural Styles and the Design of Network-based Software Architectures," APIs are RESTful as long as they comply with the 6 guiding constraints of a RESTful system:

- **Client-server architecture:** REST architecture is composed of clients, servers, and resources, and it handles requests through HTTP.
- **Statelessness:** No client content is stored on the server between requests. Information about the session state is, instead, held with the client.
- **Cacheability:** Caching can eliminate the need for some client-server interactions.
- **Layered system:** Client-server interactions can be mediated by additional layers. These layers could offer additional features like load balancing, shared caches, or security.
- **Code on demand (optional):** Servers can extend the functionality of a client by transferring executable code.
- **Uniform interface:** This constraint is core to the design of RESTful APIs and includes 4 facets:
    - **Resource identification in requests:** Resources are identified in requests and are separate from the representations returned to the client.

    - **Resource manipulation through representations:** Clients receive files that represent resources. These representations must have enough information to allow modification or deletion.
    - **Self-descriptive messages:** Each message returned to a client contains enough information to describe how the client should process the information.
    - **Hypermedia as the engine of application state:** After accessing a resource, the REST client should be able to discover through hyperlinks all other actions that are currently available.

These constraints may seem like a lot but they're much simpler than a prescribed protocol. For this reason RESTful APIs are becoming more prevalent than SOAP.

In recent years, the OpenAPI specification has emerged as a common standard for defining REST APIs. OpenAPI establishes a language-agnostic way for developers to build REST API interfaces so that users can understand them with minimal guesswork.

Another API standard to emerge is GraphQL, a query language and server-side runtime that's an alternative to REST. GraphQL prioritizes giving clients exactly the data they request and no more. As an alternative to REST, GraphQL lets developers construct requests that pull data from multiple data sources in a single API call.

SOA vs. microservices architecture

The 2 architectural approaches that use remote APIs most are service-oriented architecture (SOA) and microservices architecture. SOA, the oldest of the 2 approaches, began as an improvement to monolithic apps. Whereas a single monolithic app does everything, some functions can be supplied by different apps that are loosely coupled through an integration pattern, like an enterprise service bus (ESB).

While SOA is, in most respects, simpler than a monolithic architecture, it carries a risk of cascading changes throughout the environment if component interactions are not clearly understood. This additional complexity reintroduces some of the problems SOA sought to remedy.

Microservices architectures are similar to SOA patterns in their use of specialized, loosely coupled services. But they go even further in breaking down traditional architectures. The services within the microservices architecture use a common messaging framework, like RESTful APIs. They use RESTful APIs to communicate with each other without difficult data conversion transactions or additional integration layers. Using RESTful APIs allows, and even encourages, faster delivery of new features and updates. Each service is discrete. One service can be replaced, enhanced, or dropped without affecting any other service in the architecture. This lightweight architecture helps optimize distributed or cloud resources and supports dynamic scalability for individual services.

Related resource

- E-book: Kubernetes Patterns

Keep reading

- Article: Why choose Red Hat for APIs?
- Article: What is the Kubernetes API?
- Article: What is a service registry?

Free training

Red Hat Agile Integration Technical Overview (DO040)

Learn the basics of agile integration

APIs and Red Hat

Red Hat
3scale API
Management

Make it easy to share, secure, distribute, control, and monetize your APIs for internal or external users.

Learn more

Red Hat
Fuse

A distributed, cloud-native integration platform that connects APIs—on-premise, in the cloud, and anywhere in between.

Learn more

Red Hat
OpenShift API
Management

A hosted and managed API management service delivered as an add-on to Red Hat OpenShift Dedicated.

Learn more

Get started with Red Hat OpenShift API Management

Try it

Talk to a Red Hatter

ABOUT

We're the world's leading provider of enterprise open source solutions, using a community-powered approach to deliver high-performing Linux, cloud, container, and Kubernetes technologies. We help you standardize across environments, develop cloud-native applications, and integrate, automate, secure, and manage complex environments with award-winning support, training, and consulting services.

Company information	Jobs
Locations	Development model
Events	Newsroom
Blog	Cool Stuff Store

PRODUCTS

Red Hat Ansible Automation Platform

Red Hat Enterprise Linux

Red Hat OpenShift

Red Hat OpenShift Data Foundation

Red Hat OpenStack Platform

See all products

TOOLS

My account

Customer support

Partner resources

Developer resources

Training and certification

Red Hat Ecosystem Catalog

Resource library

TRY, BUY, SELL

Product trial center

Red Hat Store

Red Hat Marketplace

Find a partner

Contact sales

Contact training

Contact consulting

COMMUNICATE

Contact us

Feedback

Social

Red Hat newsletter