Article

An introduction to APIs and messaging

When to use APIs, when to use messaging, or when to use both

Save Like

Site feedback

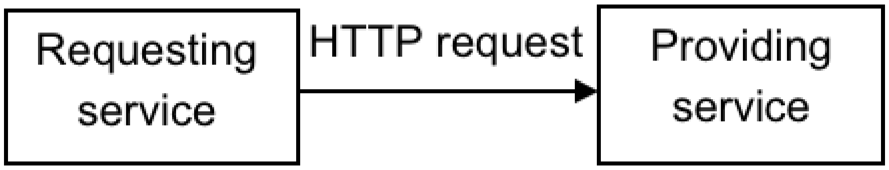
By Callum Jackson

Updated April 21, 2021 | Published June 13, 2019

APIs (application programming interfaces) have existed since the beginning of computers, but as the Web became ubiquitous, APIs have been repurposed to refer to HTTP REST-based services. REST (representational state transfer) is a commonly used concept that provides access to resources using standard HTTP operations such as GET, PUT, POST, or DELETE. For example, a system can request information regarding a book by sending the following request:

```
GET http://mybookstore/books/mybook
```

In this example the GET operation is used to declare that we are attempting to read the current state. The URL `http://mybookstore/books/mybook` is then used to identify the resource to be retrieved. The following figure shows a basic REST API.



Messaging encourages a decoupled architecture, where an intermediary – often referred to as a *messaging provider* – is placed between two applications, systems, or services. The messaging provider facilitates communication between the sender and receiver of messages and provides many benefits such as reliable delivery, work load balancing, and security.

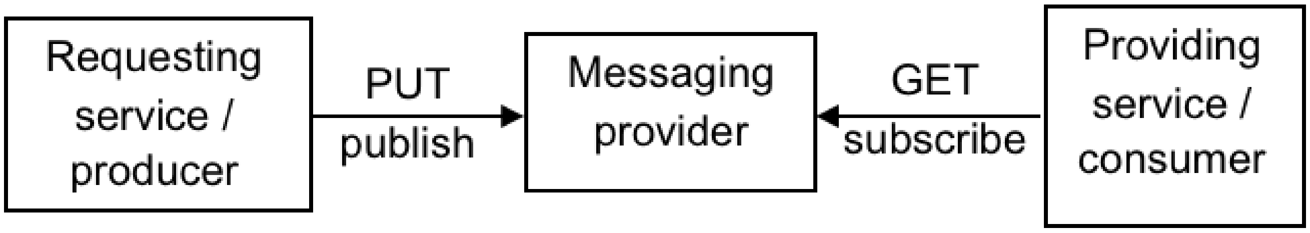
Some examples of messaging providers include:

- IBM MQ
- RabbitMQ
- ActiveMQ
- Apache Kafka

A *messaging client* is used to communicate with a messaging provider. A client can take the role of either or both of these:

- Requesting service or producer
- Providing service or consumer

Requesting services publish (PUT in REST terminology) a message to request processing, while the providing service subscribes (GET in REST terminology) to the message to complete the processing.



Characteristics of APIs and messaging

APIs and Messaging are widely used technologies, both individually and together. The key to success is understanding their characteristics so that you can select the right option based on the communication requirements:

Characteristic	APIs	Messaging
Interaction Styles	HTTP is **synchronous**. Each request message has a corresponding response; that is, an acknowledgement of the request message. This makes **request/response** solutions easy to implement.	Messaging is **asynchronous**, and allows a full range of interaction patterns, including: **fire and forget**, where requesting messages do not have corresponding responses; **request/response**, where each request message has a corresponding response; and **publish/subscribe**, where applications, systems, or services can register for messages on a defined topic. The messages are forwarded to all subscribers when publishers emit events.
Application Usage	**Ubiquitous.** Almost all companies have IT infrastructures that support HTTP traffic, and most programming languages provide built-in support for HTTP.	**Simplified application development**. Messaging frees you from coding complex delivery mechanisms to **assure messages are not lost**, freeing you to focus on business logic.
Coupled	**Simple**. Communication flows directly from the requester to the providing service (unless the architecture deliberately introduces intermediaries that control the communication).	**Decoupled**. The messaging provider acts as a shock absorber between applications, protecting the messages. If the server or receiving application goes down, or it is too busy to process more requests, the message can wait in the messaging provider until the systems are up and it can be delivered.

Site feedback

Deciding which architecture is right for your application

To help you decide whether to use APIs or messaging, let’s consider three different scenarios.

In the first scenario, a retailer exposes their product catalogue. By exposing their product catalog, a partner can quickly and easily access it on their own. The data in the catalog is read-only, so users can repeat requests in error situations. The **simple** and **synchronous** characteristics of RESTful APIs make this a natural choice.

In the second scenario, a healthcare provider updates a patient's record after they treat the patient. The health data represents a mission-critical communication to both the healthcare provider and the patient. The **assure messages are not lost** characteristic of messaging is valuable here.

Watch the start of this video to learn how messaging fits into your apps.

In the final scenario, let's consider a retailer's digital transformation. The retailers can build new engaging applications which can allow partners to sell their products. Access to the catalogue and ordering systems is required. In this situation, a combination of APIs and messaging is logical. Where **simple** and **synchronous** is important, the catalog is exposed, using APIs, and messaging provides access to the order system to **assure that messages are not lost** and **decoupling** to handle peak workloads.

Site feedback

Summary and next steps

Deciding between implementing your solution with APIs or messaging is not an either/or decision; sometimes your solutions will require both. Both APIs and messaging provide many benefits to developing robust, engaging apps.

Legend

Categories

API Management IBM MQ Messaging MQTT RabbitMQ

Table of Contents

Related

Article

API fundamentals

December 16, 2020 →

Site feedback

Video

Messaging APIs: How messaging fits into your code

April 19, 2021 →

Series

Learning path: IBM MQ Developer Essentials Badge

March 12, 2021 →

Article

Why enterprise messaging and event streaming are different

October 21, 2020

Site feedback

Build Smart
Build Secure

IBM Developer

About

FAQ

Third-party notice

Follow Us

Twitter

LinkedIn

Facebook

YouTube

Explore

Newsletters

Code patterns

APIs

Articles

Tutorials

Open source projects

Videos

Events

Cities

Select a language

English

Português

Español

한글