**Notes**

**API**

**What are the different types of APIs?**

When referencing APIs, we’re usually talking about a subcategory of APIs called web APIs. Web APIs are APIs that are accessed using the Hypertext Transfer Protocol (HTTP), the same protocol used for fetching and displaying web pages in browsers.

We can divide Web APIs into groups by intended level of access and scope of use. There are four widely agreed-upon types of web APIs: open APIs, partner APIs, internal APIs, and composite APIs.

Open APIs

Open APIs, also known as public APIs or external APIs, are available to use by any developer. As a result, open APIs typically have relatively low authentication and authorization measures, and are often restricted in the assets they share. While [**some open APIs are free**](https://blog.hubspot.com/website/free-open-apis), others require a subscription fee to use, which is often tiered based on the number of calls made to the API.

There are several advantages to making APIs public, the biggest being the ability to share data openly. This encourages any external business or developer to integrate with the app that owns the API, making both the third-party software and the API more valuable. Because of the lack of restrictions and easy implementation allowed by the open API, third parties can quickly leverage the data it provides.

For example, the traffic app Waze uses public APIs provided by municipalities and other partners about road closures, accidents, construction delays, and service vehicles. In turn, Waze makes cities easier to navigate, which pleases residents and attracts more visitors.

Partner APIs

Partner APIs are shared externally, but only among those who have a business relationship with the company providing the API. Access is limited to authorized clients with official licenses, and thus [**security measures**](https://blog.hubspot.com/website/api-security) tend to be stronger with partner APIs than with public APIs.

Some businesses favor partner APIs because they want (1) greater control over who can access their resources and (2) more say in how those resources are used. For example, Pinterest adopted a submission-based approach to providing access to new data services via its API, requiring partners to submit a request detailing how they would like to use the API before being granted access.

Internal APIs

Unlike open APIs and partner APIs, internal APIs (also called private APIs) are not intended for use by third parties. Internal APIs are only made available for use inside a company and are meant to streamline data transfers between teams and systems. Developers working for the company can use these APIs, but external developers can’t.

Because internal APIs aren't documented in a publicly released software development kit (or at all in some cases), they are often completely hidden from the public. However, many companies do eventually go public with their internal APIs.

Using APIs for internal data transfers is regarded as more efficient, secure, and traceable. It’s also a scalable solution — when a business introduces a new internal system, this system can communicate with existing systems via their APIs.

Composite APIs

Composite APIs combine multiple APIs allowing developers to bundle calls or requests and receive one unified response from different servers. If you need data from different applications or data sources, you would use a composite API. Alternatively, you can use a composite API to set off an automatic chain of calls and responses without requiring your intervention.

Because they reduce the number of total API calls, composite APIs can result in less server load and overall faster systems, as well as reduced complexity in the system. They’re commonly deployed in [**microservices**](https://blog.hubspot.com/website/microservices-vs-api) in which one job may require data from many internal APIs to complete.

Take this example from Stoplight: Say you want to create an order within a shopping cart API. You might think that this takes just one request. But, in fact, several requests must be made. First, you need to create a customer profile. Then, you need to create the order, add an item, add another, and change the status of the order. Instead of making five separate API calls in succession, you can make just one with a composite API.

**Types of API Architectures**

We can also understand APIs in terms of their architecture. An API’s architecture consists of the rules that guide what information an API can share with clients and how it shares the data. REST, SOAP, and RPC are the most popular API architectures in use today — let’s unpack each one in more detail.

REST

Today, the majority of web APIs are built on REST. REST, which stands for representational state transfer, is a set of guidelines for scalable, lightweight, and easy-to-use APIs. A REST API (or “RESTful” API) is an API that follows REST guidelines and is used for transferring data from a server to a requesting client.

For a more in-depth look at REST guidelines, see our full [**guide to REST APIs**](https://blog.hubspot.com/website/what-is-rest-api). Briefly, these guidelines are:

* **Client-Server Separation:** All client-server interactions must be in the form of a request from the client, followed by a response from the server. Servers can’t request and clients can’t respond.
* **Uniform Interface:** All requests and responses must use HTTP as the communication protocol and be formatted in a specific way to ensure compatibility between any client and any server. Server responses are formatted in JavaScript Object Notation (JSON).
* **Stateless:** Each client-server interaction is independent of every other interaction. The server stores no data from client requests and remembers nothing from past interactions.
* **Layered system:** Requests and responses must always be formatted the same way, even when passed through intermediate servers between the client and the API.
* **Cacheable:** Server responses should indicate whether a provided resource can be cached by the client and for how long.

By following these guidelines, REST APIs can be used for quick, easy, secure data transfers, making them a popular choice among developers.

SOAP

SOAP (Simple Object Access Protocol) is a protocol for transmitting data across networks and can be used to build APIs. SOAP is standardized by the [**World Wide Web Consortium (W3C)**](https://www.w3.org/) and utilizes [**XML**](https://blog.hubspot.com/website/what-is-xml-file) to encode information.

SOAP strictly defines how messages should be sent and what must be included in them. This makes SOAP APIs more secure than REST APIs, although the rigid guidelines also make them more code-heavy and harder to implement in general.

For this reason, SOAP is often implemented for internal data transfers that require high security, and the more flexible REST architecture is deployed more commonly everywhere else. But, one more advantage to SOAP is that it works over any communication protocol (not just HTTP, as is the case with REST).

RPC

The RPC (Remote Procedural Call) protocol is the most straightforward of the three architectures. Unlike REST and SOAP that facilitate the transfer of data, RPC APIs invoke processes. In other words, they execute scripts on a server.

RPC APIs may employ either JSON (a JSON-RPC protocol) or XML (an XML-RPC protocol) in their calls. XML is more secure and more accommodating than JSON, but these two protocols are otherwise similar. Though the RPC protocol is strict, it's a relatively simple and easy way to execute code on remote networks.

RPC APIs are limited in their security and capabilities, so you likely won’t see them as often as REST or SOAP APIs on the web. However, it can be used for internal systems for making basic process requests, especially many at once.

**Choosing the Right Type of API**

To summarize, we can group web APIs into four broad categories:

* **Open APIs**, which any developer can access.
* **Partner APIs**, which only authorized developers may access.
* **Internal APIs**, which only internal teams may access.
* **Composite APIs**, which combine multiple APIs.

There are also three common types of API architectures:

* **REST**, a collection of guidelines for lightweight, scalable web APIs.
* **SOAP**, a stricter protocol for more secure APIs.
* **RPC**, a protocol for invoking processes that can be written with XML (**XML-RPC**) or JSON (**JSON-RPC**).

If you’re not someone who needs to know the gritty-gritty, this is a good baseline understanding of the types of APIs out there and how developers use them. By harnessing the right API, you’ll empower your business to partner with other applications, expanding your reach and influence.

**MongoDB**

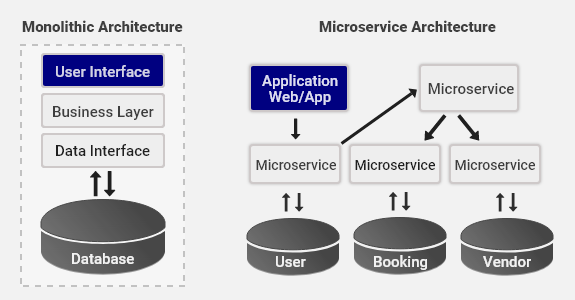
MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License which is deemed non-free by several distributions.

Difference between monolithic and microservices architecture

Monolithic architecture  is built as one large system and is usually one code-base. Monolithic application is tightly coupled and entangled as the application evolves, making it difficult to isolate services for purposes such as independent scaling or code maintainability.

It extremely difficult to change technology or language or framework because everything is tightly coupled and depend on each other.

| **Sr. No.** | **Key** | **Monolithic architecture** | **Microservices architecture** |
| --- | --- | --- | --- |
| 1 | Basic | Monolithic architecture is built as one large system and is usually one code-base | Microservices architecture is built as small independent module based on business functionality |
| 2 | Scale | It is not easy to scale based on demand | It is easy to scale based on demand. |
| 3 | Database | It has shared database | Each project and module has their own database |
| 4 | Deployment | Large code base makes IDE slow and build time gets increase. | Each project is independent and small in size. So overall build and development time gets decrease. |
| 5 | Tightly Coupled and Loosely coupled | It extremely difficult to change technology or language or framework because everything is tightly coupled and depend on each other | Easy to change technology or framework because every module and project is independent |

Microservices architecture is built as small independent module based on business functionality. In microservices application, each project and services are independent from each other at the code level. Therefore it is easy to  configure and deploy completely and also easy to scale based on demand.

LINKS

MongoDB tutorial : https://www.youtube.com/watch?v=oSIv-E60NiU

Building REST API’s with Flask : <https://www.youtube.com/watch?v=Jzv3G5iDLvw>

Web Development using FLASK and HTML : <https://www.youtube.com/watch?v=SlmualOsp6I&list=PLLhEJK7fQIxD9Yz3wljWxzkkwyIVtYC1b>