SOAP vs. REST: A Look at Two Different API Styles

When talking about API (application programming interface) architectures, it's common to want to compare SOAP vs. REST, two of the most common API paradigms. Although the two are often compared as apples to apples, they're inherently different technologies and aren't easily compared on a granular level.

Why? Because **SOAP is a protocol**, and **REST is an architectural style**. A REST API can actually utilize the SOAP protocol, just like it can use HTTP. So, right off the bat, they're going to be packaged differently, function differently, and be used in different scenarios.

Now that we've gotten that out of the way, let's look at little closer at each—including some of the pros that would make you want to use one over the other for your application, if the shoe fits.

FIRST, WHAT IS AN API?

In the simplest of terms, an API is a piece of software that plugs one application directly into the data and services of another by granting it access to specific parts of a server. APIs let two pieces of software communicate, they're the basis for everything we do on mobile, and they allow us to streamline IT architectures, power savvier marketing efforts, and make easier to share data sets.

Like all software, APIs can be pretty straightforward or incredibly complex, and there are different ways to program one with different attributes that are better suited to your application. Also, with more built-in features comes more overhead—something we'll see when we look at what SOAP has to offer.

WHAT IS A REST API?

REST (Representational State Transfer) is truly a "web services" API. REST APIs are based on URIs (Uniform Resource Identifier, of which a URL is a specific type) and the HTTP protocol, and use JSON for a data format, which is super browser-compatible. (It could also theoretically use the SOAP protocol, as we mentioned above.) REST APIs can be simple to build and scale, but they can also be massive and complicated—it's all in how they're built, added on to, and what they're designed to do.

Reasons you may want to build an API to be RESTful include resource limitations, fewer security requirements, browser client compatibility, discoverability, data health, and scalability—things that really apply to web services.

Some quick REST information:

- **REST is all about simplicity, thanks to HTTP protocols.**
- **REST APIs facilitate client-server communications and architectures.** If it's RESTful, it's built on this client-server principle, with round trips between the two passing payloads of information.
- **REST APIs use a single uniform interface.** This simplifies how applications interact with the API by requiring they all interface in the same way, through the same portal. This has advantages and disadvantages; check with your developer to see if this will affect implementation changes down the road.
- **REST is optimized for the web.** Using JSON as its data format makes it compatible with browsers.
- **REST is known for excellent performance and scalability.** But, like any technology, it can get bogged down or bog down your app. That's why languages like GraphQL have come along to address problems even REST can't solve.

WHAT IS SOAP?

SOAP (Simple Object Access Protocol) is its own protocol, and is a bit more complex by defining more standards than REST—things like security and how messages are sent. These built-in standards do carry a bit more overhead, but can be a deciding factor for organizations that require more comprehensive features in the way of security, transactions, and ACID (Atomicity, Consistency, Isolation, Durability) compliance. For the sake of this comparison, we should point out that many of the reasons SOAP is a good choice rarely apply to web services scenarios, which make it more ideal for enterprise-type situations.

Reasons you may want to build an application with a SOAP API include higher levels of security (e.g., a mobile application interfacing with a bank), messaging apps that need reliable communication, or ACID compliance.

- **SOAP has tighter security.** WS-Security, in addition to SSL support, is a built-in standard that gives SOAP some more enterprise-level security features, if you have a requirement for them.
- **Successful/retry logic for reliable messaging functionality.** Rest doesn't have a standard messaging system and can only address communication failures by retrying. SOAP has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries.
- **SOAP has built-in ACID compliance.** ACID compliance reduces anomalies and protects the integrity of a database by prescribing exactly how transactions can interact with the database. ACID is more conservative than other data consistency models, which is why it's typically favored when handling financial or otherwise sensitive transactions.

SOAP VS. REST: THE KEY DIFFERENCES

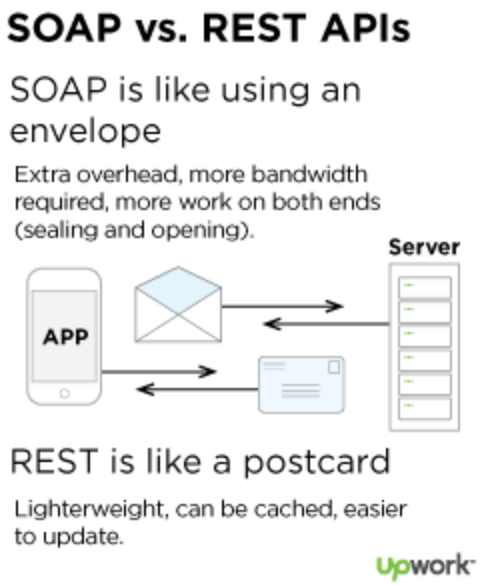
SOAP vs. REST Comparison: Which is Right for You?

Difference	SOAP	REST
Style	Protocol	Architectural style
Function	Function-driven: transfer structured information	Data-driven: access a resoruce for data
Data format	Only uses XML	Permits many data formats, including plain text, HTML, XML, and JSON
Security	Supports WS-Security and SSL	Supports SSL and HTTPS
Bandwidth	Requires more resources and bandwidth	Requires fewer resources and is lightweight
Data cache	Can not be cached	Can be cached
Payload handling	Has a strict communication contract and needs knowledge of everything before any interaction	Needs no knowledge of the API
ACID compliance	Has built-in ACID compliance to reduce anomalies	Lacks ACID compliance

Visit upwork.com to learn more

upwork

- **SOAP is a protocol. REST is an architectural style.** An API is designed to expose certain aspects of an application's business logic on a server, and SOAP uses a service interface to do this while REST uses URIs.
- **REST APIs access a resource for data (a URI); SOAP APIs perform an operation.** REST is an architecture that's more data-driven; SOAP is a standardized protocol for transferring structured information that's more function-driven.
- **REST permits many different data formats** including plain text, HTML, XML, and JSON, which is a great fit for data and yields more browser compatibility; SOAP only uses XML.
- **Security is handled differently.** SOAP supports WS-Security, which is great at the transport level and a bit more comprehensive than SSL, and more ideal for integration with enterprise-level security tools. Both support SSL for end-to-end security, and REST can use the secure version of the HTTP protocol, HTTPS.
- **SOAP requires more bandwidth; REST requires fewer resources** (depending on the API). There's a little more overhead with SOAP out of the gate, on account of the envelope-style of payload transport. Because REST is used primarily for web services, its being lightweight is an advantage in those scenarios.
- **REST calls can be cached, SOAP-based calls cannot be cached.** Data can be marked as cacheable, which means it can be reused by the browser later without having to initiate another request back to the server. This saves time and resources.
- **An API is built to handle your app's payload, and REST and SOAP do this differently.** A payload is data sent over the internet, and when a payload is "heavy," it requires more resources. REST tends to use HTTP and JSON, which lighten the payload; SOAP relies more on XML.
  SOAP is tightly coupled with the server; REST is coupled to a lesser degree. In programming, the more layers of abstraction between two pieces of technology, the less control you have over their interaction, but there's also less complexity and it's easier to make updates to one or the other without blowing up the whole relationship. The same goes for APIs and how closely they interact with a server. This is a key difference between SOAP and REST to consider. SOAP is very closely coupled with the server, having a strict communication contract with it that makes it more difficult to make changes or updates. A client interacting with a REST API needs no knowledge of the API, but a client interacting with a SOAP API needs knowledge about everything it will be using before it can even initiate an interaction.



For the most part, when it comes to APIs for web services, developers tend toward a RESTful architecture unless the SOAP path is clearly a better choice, say for an enterprise app that’s backed by more resources, needs super tight security, and has more requirements.