Defining SOAP and REST

SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are both web service communication protocols. SOAP was long the standard approach to web service interfaces, although it's been dominated by REST in recent years, with REST now representing more than 70% of public APIs according to Stormpath. Understand the primary differences between SOAP vs. REST and how each can benefit your organization's goals.

SOAP vs REST: Primary Differences

REST operates through a solitary, consistent interface to access named resources. It's most commonly used when you're exposing a public API over the Internet. SOAP, on the other hand, exposes components of application logic as services rather than data. Additionally, it operates through different interfaces. To put it simply, REST accesses data while SOAP performs operations through a more standardized set of messaging patterns. Still, in most cases, either REST or SOAP could be used to achieve the same outcome (and both are infinitely scalable), with some differences in how you'd configure it.

SOAP was originally created by Microsoft, and it's been around a lot longer than REST. This gives it the advantage of being an established, legacy protocol. But REST has been around for a good time now as well. Plus, it entered the scene as a way to access web services in a much simpler way than possible with SOAP by using HTTP.

Benefits of REST Over SOAP

In addition to using HTTP for simplicity, REST offers a number of other benefits over SOAP:

- REST allows a greater variety of data formats, whereas SOAP only allows XML.
- Coupled with JSON (which typically works better with data and offers faster parsing), REST is generally considered easier to work with.
- Thanks to JSON, REST offers better support for browser clients.
- REST provides superior performance, particularly through caching for information that's not altered and not dynamic.
- It is the protocol used most often for major services such as Yahoo, Ebay, Amazon, and even Google.
- REST is generally faster and uses less bandwidth. It's also easier to integrate with existing websites with no need to refactor site infrastructure. This enables developers to work faster rather than spend time rewriting a site from scratch. Instead, they can simply add additional functionality.

Still, SOAP remains the preferred protocol for certain use cases. The general consensus among experts these days is that REST is the typically preferred protocol unless there's a compelling reason to use SOAP (and there are some cases in which SOAP is preferred).

Benefits of SOAP Over REST

Because you can achieve most outcomes using either protocol, it's sometimes a matter of personal preference. However, there are some use cases that SOAP tends to be better-suited for. For instance, if you need more robust security, SOAP's support for WS-Security can come in handy. It offers some additional assurances for data privacy and integrity. It also provides support for identity verification through intermediaries rather than just point-to-point, as provided by SSL (which is supported by both SOAP and REST).

Another advantage of SOAP is that it offers built-in retry logic to compensate for failed communications. REST, on the other hand, doesn't have a built-in messaging system. If a communication fails, the client has to deal with it by retrying. There's also no standard set of rules for REST. This means that both parties (the service and the consumer) need to understand both content and context.

Other benefits of SOAP include:

- SOAP's standard HTTP protocol makes it easier for it to operate across firewalls and proxies without modifications to the SOAP protocol itself. But, because it uses the complex XML format, it tends to be slower compared to middleware such as ICE and COBRA.
- Additionally, while it's rarely needed, some use cases require greater transactional reliability than what can be achieved with HTTP (which limits REST in this capacity). If you need ACID-compliant transactions, SOAP is the way to go.
- In some cases, designing SOAP services can actually be less complex compared to REST. For web services that support complex operations, requiring content and context to be maintained, designing a SOAP service requires less coding in the application layer for transactions, security, trust, and other elements.
- SOAP is highly extensible through other protocols and technologies. In addition to WS-Security, SOAP supports WS-Addressing, WS-Coordination, WS-ReliableMessaging, and a host of other web services standards, a full list of which you can find on W3C.

At the end of the day, the best protocol is the one that makes the most sense for the organization, the types of clients that you need to support, and what you need in terms of flexibility. Most new APIs are built using REST and JSON, simply because it typically consumes less bandwidth and is easier to understand both for developers implementing initial APIs as well as other developers who may write other services against it. Because it's more easily consumed by most of today's web browsers, REST+JSON has become the defacto technology for the majority of public APIs. However, SOAP remains a valuable protocol in some circumstances. Plus, you don't have to look far to find die-hard fans advocating for SOAP for certain use cases.

May 13, 2019

NO COMMENTS

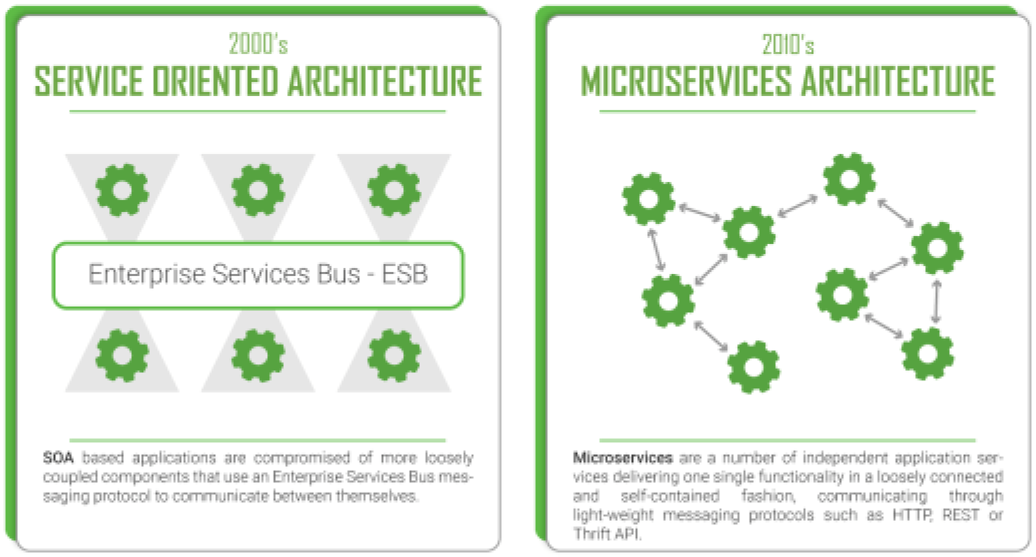
SOAP vs RESTful Microservices

Advertisements

In this article, we will discuss some points that provide the difference between SOAP versus RESTful **microservices**. Before this article, we have discussed the **Software architecture patterns and design**, and also we have discussed **how to decompose an application to the Microservices** architecture based application.

Advertisements

Let's start this article with the following diagram, that provides a lot of description about the differences:


2000's
SERVICE ORIENTED ARCHITECTURE
Enterprise Services Bus - ESB
SOA based applications are compromised of more loosely coupled components that use an Enterprise Services Bus messaging protocol to communicate between themselves.
2010's
MICROSERVICES ARCHITECTURE
Microservices are a number of independent application services delivering one single functionality in a loosely connected and self-contained fashion, communicating through light-weight messaging protocols such as HTTP, REST or Thrift API.

SOAP vs RESTful Microservices

The preceding diagram explains that SOA service based applications are compromised of more loosely coupled components that use an Enterprise Services Bus (ESB) messaging protocol to communicate between themselves.

But the Microservices based applications are a number of independent application services delivering one single functionality in a loosely connected and self-contained fashion, communicating through light-weight messaging protocols such as HTTP REST or Thrift API.

SOAP versus RESTful microservices

SOAP	RESTful microservices
An XML-based message protocol.	An architectural style.
Uses WSDL for communication between the consumer and the provider.	Use XML or JSON to send and receive data.
Invokes services by calling the RPC method.	Simply call services via the URL path.
The transfer is over HTTP. Also uses other protocols, such as SMTP or FTP.	The transfer is over HTTP only.
SOAP-based reads can't be cached.	RESTful microservice reads can be cached.
SOAP is not very scalable.	RESTful microservices are very scalable.
SOAP is more suitable for enterprise systems and high-security systems, such as a banking system.	RESTful microservices are suitable for all types of systems apart from where high security and high reliability is critical.
Doesn't support error handling.	Has built-in error handling.
Uses service interfaces to expose the business logic.	Uses URI to expose business logic.

Sources: https://stackify.com/soap-vs-rest/
https://www.dineshonjava.com/soap-vs-restful-microservices/