**Service-Oriented Architecture (SOA)**

Service-oriented architecture (SOA) is a software design approach that focuses on building functional, scalable software systems from individual components, called services. Services can interact with one another to perform tasks, such as allowing someone to sign in once and access a variety of business applications. In SOA, the emphasis is on modularity, reusability, and interoperability—when businesses break complex applications down into smaller, more manageable building blocks, the result is greater flexibility and scalability.

Services aren’t hard-coded into applications. Instead, services are published to a registry, and when an application wants to use a service, it asks the registry to find the latest version. In this way, services can be easily changed or updated by people without detailed knowledge of the application. An SOA approach makes it easier to adapt to changing business needs and to integrate systems that provide related functionality.

**Characteristics:**

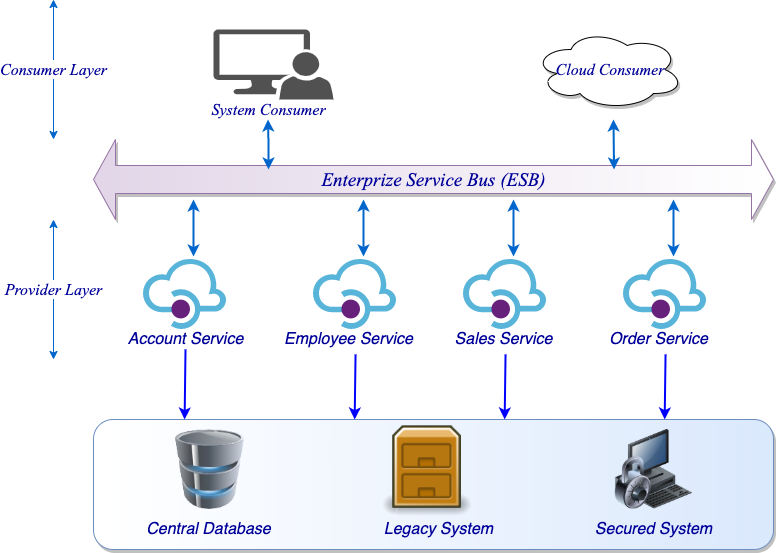
• Services are **loosely coupled** (independent and interchangeable).

• Promotes **interoperability** across different platforms and technologies.

• Uses **protocols** like SOAP, REST, and messaging systems like JMS.

• Supports **Enterprise Service Bus (ESB)** for integrating multiple services.

• Encourages **reusability** of services across various applications.



**Use Cases:**

• Large enterprise applications that integrate multiple subsystems.

• Banking, healthcare, and telecom systems requiring cross-platform communication.

• Legacy system modernization.

**How is SOA different from microservices?**

SOA and microservices have a similar goal—to provide applications with functionality through network-based function calls. However, SOA considers use on an expansive, and usually higher, level. Microservices consider use on an application level and provide service for one extremely specialized task to repeat as needed. For example, an authentication plugin for secure login would be considered part of SOA; however, a repeatable service that encrypts the password is a microservice. Microservices may be called within an SOA service or integrated in the larger project. In this example, the authentication service will use the encryption microservice in getting its job done. The main distinction between the two approaches comes down to scope. To put it simply, service-oriented architecture (SOA) has an enterprise scope, while microservices architecture has an application scope.

**What is an ESB?**

An ESB, or enterprise service bus, is an architectural pattern whereby a centralized software component performs integrations between applications. It performs transformations of data models, handles connectivity and messaging, performs routing, converts communication protocols and potentially manages the composition of multiple requests. The ESB can make these integrations and transformations available as a service interface for reuse by new applications. It is possible to implement an SOA without an ESB, but this would be equivalent to just having a bunch of services. Each application owner would need to directly connect to any service it needs and perform the necessary data transformations to meet each of the service interfaces. This is much work (even if the interfaces are reusable) and creates a significant maintenance challenge in the future as each connection is point to point. In fact, ESBs were, eventually, considered such a de facto element of any SOA implementation that the two terms are sometimes used as synonyms, creating confusion.

SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are communication protocols that enable services in an SOA to interact.

SOAP and REST are two internet data exchange mechanisms. For example, imagine that your internal accounts system shares data with your customer's accounting system to automate invoicing tasks. The two applications share data by using an API that defines communication rules. SOAP and REST are two different approaches to API design. The SOAP approach is highly structured and uses XML data format. REST is more flexible and allows applications to exchange data in multiple formats.

SOAP is an older technology that requires a strict communication contract between systems. New web service standards have been added over time to accommodate technology changes, but they create additional overheads. REST was developed after SOAP and inherently solves many of its shortcomings. REST web services are also called RESTful web services.

**SOAP (Simple Object Access Protocol)**

The Simple Object Access Protocol (SOAP) is a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. It uses **XML** for message formatting and typically relies on **HTTP, SMTP, or TCP** for transport.

**Characteristics:**

• **Strict message structure** (uses XML).

• **WS-Security** ensures secure communication.

• **Protocol-based**, independent of REST principles.

• **Slower** than REST due to XML overhead.

• **Stateful or stateless**, depending on implementation.

**SOAP Message Format (XML Example):**

| <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:web="http://www.example.com/webservice">  <soapenv:Header/>  <soapenv:Body>  <web:GetUserDetails>  <web:UserID>123</web:UserID>  </web:GetUserDetails>  </soapenv:Body>  </soapenv:Envelope> |
| --- |

**Use Cases:**

• Banking and financial services (high security required).

• Government and enterprise-level applications.

• Asynchronous processing (e.g., invoice generation).

SOAP is a protocol that defines rigid communication rules. It has several associated standards that control every aspect of the data exchange. For example, here are some standards SOAP uses:

* Web Services Security (WS-Security) specifies security measures like using unique identifiers called tokens
* Web Services Addressing (WS-Addressing) requires including routing information as metadata
* WS-ReliableMessaging standardizes error handling in SOAP messaging
* Web Services Description Language (WSDL) describes the scope and function of SOAP web services

When you send a request to a SOAP API, you must wrap your HTTP request in a SOAP envelope. This is a data structure that modifies the underlying HTTP content with SOAP request requirements. Due to the envelope, you can also send requests to SOAP web services with other transport protocols, like TCP or Internet Control Message Protocol (ICMP). However, SOAP APIs and SOAP web services always return XML documents in their responses

**REST (Representational State Transfer)**

REST is an **architectural style** for web services that leverages **HTTP methods** to interact with resources, usually in **JSON or XML format**.

**Characteristics:**

• **Stateless:** No session is stored between requests.

• **Lightweight:** Uses JSON (or XML), reducing bandwidth.

• **Cacheable:** Supports caching to improve performance.

• **Uses HTTP methods:**

• GET → Retrieve data

• POST → Create data

• PUT → Update data

• DELETE → Remove data

REST is a software architectural style that imposes six conditions on how an API should work. These are the six principles REST APIs follow:

* *Client-server architecture:* The sender and receiver are independent of each other regarding technology, platforming, programming language, and so on.
* *Layered:* The server can have several intermediaries that work together to complete client requests, but they are invisible to the client.
* *Uniform interface:* The API returns data in a standard format that is complete and fully usable.
* *Stateless:* The API completes every new request independently of previous requests.
* *Cacheable:* All API responses are cacheable.
* *Code on demand (optional):* The API response can include a code snippet if required.

**Key differences: SOAP vs REST**

SOAP is a protocol, while REST is an architectural style. This creates significant differences in how SOAP APIs and REST APIs behave.

**Design**

The SOAP API exposes functions or operations, while REST APIs are data-driven. For example, consider an application with employee data that other applications can manipulate. The application's SOAP API could expose a function called CreateEmployee. To create an employee, you would specify the function name in your SOAP message when sending a request.

However, the application's REST API could expose a URL called /employees, and a POST request to that URL would create a new employee record.

**Flexibility**

SOAP APIs are rigid and only allow XML messaging between applications. The application server also has to maintain the state of each client. This means it has to remember all previous requests when processing a new request.

REST is more flexible and allows applications to transfer data as plain text, HTML, XML, and JSON. REST is also stateless, so the REST API treats every new request independently of previous requests.

**Performance**

SOAP messages are larger and more complex, which makes them slower to transmit and process. This can increase page load times.

REST is faster and more efficient than SOAP due to the smaller message sizes of REST. REST responses are also cacheable, so the server can store frequently accessed data in a cache for even shorter page load times.

**Scalability**

The SOAP protocol requires applications to store the state between requests, which increases bandwidth and memory requirements. As a result, it makes applications expensive and challenging to scale.

Unlike SOAP, REST permits stateless and layered architecture, which makes it more scalable. For example, the application server can pass the request to other servers or allow an intermediary (like a content delivery network) to handle it.

**Security**

SOAP requires an additional layer of WS-Security to work with HTTPS. WS-Security uses additional header content to ensure only the designated process in the specified server reads the SOAP message content. This adds communication overheads and negatively impacts performance.

REST supports HTTPS without additional overheads.

**Reliability**

SOAP has error handling logic built into it, and it provides more reliability. On the other hand, REST requires you to try again in case of communication failures, and it’s less reliable.

**What are the similarities between SOAP and REST?**

You can use SOAP and REST to build APIs or communication points between diverse applications. The terms web service and API are used interchangeably. However, APIs are the broader category. Web services are a special type of API.

Here are other similarities between SOAP and REST:

* They both describe rules and standards on how applications make, process, and respond to data requests from other applications
* They both use HTTP, the standardized internet protocol, to exchange information
* They both support SSL/TLS for secure, encrypted communication