A web service is a standardized way for different software applications to communicate and exchange data over a network, typically the internet, using common protocols such as HTTP or HTTPS and usually formats like XML or JSON.

**Why Web Services Are Needed**

* Interoperability: They allow applications built on different platforms (Java, .NET, Ruby, etc.) to work together seamlessly.
* Reusability: Business logic can be developed once and accessed by multiple client applications across the network.
* Standardized Communication: Use of industry standards (SOAP, HTTP, XML, JSON) ensures reliable, universal communication.
* Loose Coupling: Web services are independent of the applications consuming them, enabling system updates without disrupting other components.
* Cost Efficiency: Leveraging internet-based protocols and existing web infrastructure makes them cheaper than custom integration approaches.
* Scalability: They are easy to deploy and can serve many users or clients at scale.

**Advantages of Web Services**

* Platform independence: Can be used by any client, regardless of language or technology.
* Easy integration and interoperability: Connects diverse and legacy systems.
* Reusability: Single service can be used by multiple clients.
* Standardized protocols: Reliable communication using industry standards.
* Lower costs: Utilizes the existing web infrastructure, reducing deployment and communication expenses.
* Rapid deployment: Easy to deploy and maintain, often requiring minimal investment in extra hardware.

**Disadvantages of Web Services**

* Internet dependency: Web services require a stable and reliable internet connection. Any disruption or poor connectivity affects their availability and performance.
* Performance limitations: Typically slower compared to native solutions, especially for complex transactions or high-volume processing.
* Security risks: Being publicly accessible over networks means they are exposed to security threats like hacking, data breaches, and phishing. Extra safeguards are necessary.
* Browser and compatibility issues: Clients and applications may encounter inconsistencies or need extra effort for cross-browser and cross-platform compatibility.
* Limited access to device features: They may not be able to access all hardware or system capabilities compared to native applications.
* Availability challenges: No service is available 100% of the time due to reliance on network and server infrastructure; downtime is possible.

Web services are powerful tools for enabling integration and efficiency but must be carefully implemented to overcome connectivity, security, and performance challenges

WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery, and Integration) are XML-based standards critical for the definition and discovery of web services in distributed computing

**WSDL (Web Services Description Language)**

* Purpose: WSDL is an XML-based language used to describe the functionalities, operations, and network access details of a web service.
* Features: It specifies what a service does, where it resides (network location), and how to interact with it (available operations, input and output formats, protocols).
* Usage: Clients or other services can read a WSDL file to understand how to consume or interact with a web service without knowing its implementation details.
* Components: Major elements include <types>, <messages>, <binding>, <portType>, and <service>, each detailing different aspects of service operation.
* Example: Exposes a "weather forecast" service showing which operations are available (e.g. getForecast), location, and expected input/output formats.

**UDDI (Universal Description, Discovery, and Integration)**

* Purpose: UDDI is a registry service that lets businesses publish, discover, and integrate web services in a platform-neutral way.
* Features: It acts as a directory of web services, similar to a phone book for businesses/services, making it possible for businesses or clients to look up and find needed services.
* Usage: Service providers publish their web services (typically by registering the WSDL documents) to a UDDI registry. Service consumers query UDDI to discover and bind to the required services.
* Role in SOA: UDDI helps in the discovery phase of Service-Oriented Architecture, allowing dynamic binding and integration of services.

WSDL provides the definition and interface of a web service, while UDDI enables service discovery and integration in distributed environments

| **Feature** | **SOAP API** | **REST API** |
| --- | --- | --- |
| Protocol/Style | Protocol | Architectural Style |
| Data Format | XML | JSON, XML, HTML, Plain Text |
| Transport | HTTP, HTTPS, SMTP, etc. | HTTP/HTTPS |
| State | Stateful | Stateless |
| Performance | Slower, heavier | Faster, lightweight |
| Scalability | Less scalable, complex | Highly scalable, easy to cache |
| Security | Built-in, message-level | Relies on HTTPS |
| Typical Use | Enterprise, mission-critical | Modern web & mobile applications |

SOAP and REST serve different needs, with REST dominating most new API development due to its simplicity, speed, and compatibility, while SOAP persists in scenarios that demand higher reliability and stricter security.

**Common HTTP Methods**

* GET: Retrieves data from the server. Should not modify server data and is considered safe and idempotent—repeating the same request yields the same result.
* POST: Sends data to the server, usually to create a new resource. Not idempotent; repeating the request may create multiple resources.
* PUT: Updates/replaces a resource entirely, or creates it if it does not exist. Idempotent—multiple identical requests result in the same outcome.
* DELETE: Removes a resource from the server. Idempotent; repeated calls have the same effect (resource is deleted or remains deleted).
* PATCH: Partially updates a resource. Not idempotent by default, but can be implemented to be idempotent.

**Classes of HTTP Status Codes**

* 1xx (Informational): The request was received and is being processed. It is a provisional response.
  + Examples: 100 Continue, 101 Switching Protocols, 102 Processing.
* 2xx (Success): The request was successfully received, understood, and accepted.
  + Examples:
    - 200 OK: Request succeeded.
    - 201 Created: A new resource was created.
    - 204 No Content: Request succeeded but no content returned.
* 3xx (Redirection): Further action is needed to complete the request, usually involving a new URL.
  + Examples:
    - 301 Moved Permanently
    - 302 Found
    - 304 Not Modified.
* 4xx (Client Error): The request contains bad syntax or cannot be fulfilled by the server.
  + Examples:
    - 400 Bad Request
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found.
* 5xx (Server Error): The server failed to fulfill a valid request due to an internal error.
  + Examples:
    - 500 Internal Server Error
    - 502 Bad Gateway
    - 503 Service Unavailable

| **Status Code** | **Meaning** | **Description** |
| --- | --- | --- |
| 100 | Continue | Server received request, client should continue |
| 200 | OK | Request succeeded, response returned |
| 201 | Created | New resource successfully created |
| 204 | No Content | Successful but no content to return |
| 301 | Moved Permanently | Resource has moved to a new URL |
| 302 | Found | Temporary redirection |
| 400 | Bad Request | Invalid request syntax or parameters |
| 401 | Unauthorized | Authentication required or failed |
| 403 | Forbidden | Server refuses to fulfill request |
| 404 | Not Found | Requested resource not found |
| 500 | Internal Server Error | Server encountered an unexpected condition |
| 503 | Service Unavailable | Server temporarily unable to handle request |