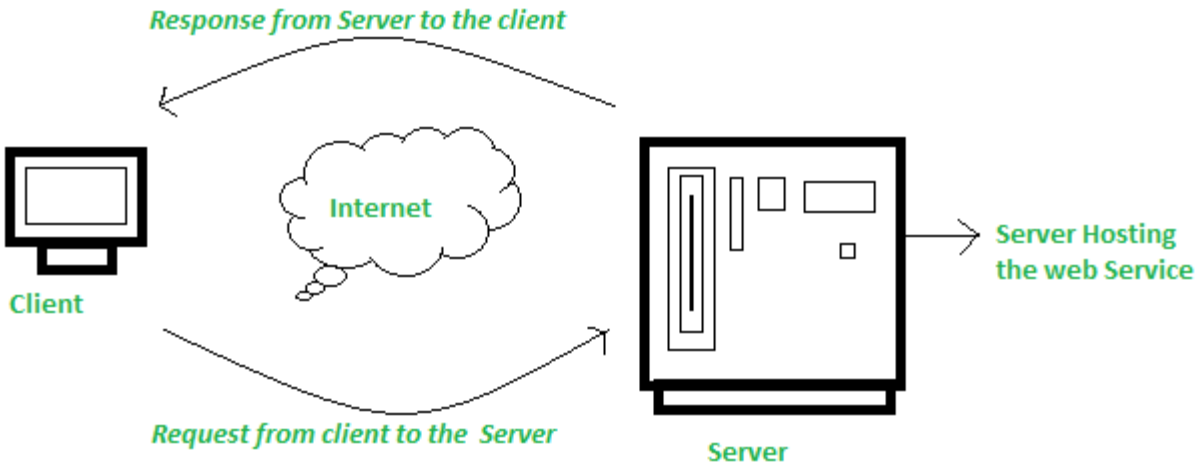
What is Web Service

- A web service is a software module that is intended to carry out a specific set of functions.
- A web service is a standardized method for propagating messages between client and server applications.
- A Web services can be invoked by client over the network.
- The web service would be able to deliver functionality to the client that invoked the web service.
- A web service is a set of open protocols and standards that allow data to be exchanged between different applications or systems.
- Web services can be used by software programs written in a variety of programming languages and running on a variety of platforms to exchange data via computer networks
- Any software, application, or cloud technology that uses standardized web protocols (HTTP or HTTPS) to connect, interoperate, and exchange data messages – commonly XML (Extensible Markup Language) and JSON – across the internet is considered a web service.
- A client invokes a web service by submitting an XML/JSON request, which the service responds with an XML/JSON response.

How Does Web Service Work?

- The diagram depicts a very simplified version of how a web service would function. The client would use requests to send a sequence of web service calls to a server that would host the actual web service.



What is REST

- REST stands for Representational State Transfer
- Representational State Transfer (REST) is an architectural style that defines a set of constraints to be used for creating web services.
- The web services that follows the REST architectural style is called RESTful Web Services.
- REST is an architectural pattern used for exchanging data over a distributed environment.
- At rest the data will be exchanged between the client and server over a distributed environment.
- A Distributed Environment means the client can be on any platform like Java, .NET, PHP, etc.; the server can also be on any platform like Java, .NET, PHP, etc.
- The REST architectural pattern treats each service as a resource, and a client can access those resources by using HTTP Protocol methods such as GET, POST, PUT, PATCH, and DELETE.

What are the REST Principles?

- The REST architectural pattern specifies a set of constraints that a system should adhere to. Here are the REST constraints or principles.
- 1. Client-Server Constraint
- 2. Stateless Constraint
- 3. Cacheable Constraint
- 4. Uniform Interface Constraint
- 5. Content Negotiation
- 6. Layered System

Constraints for RESTful Architecture

- **Client-Server Constraint:**
- This constraint specifies that a client sends a request to the server, and the server sends a response back to the client. This separation of concerns supports the independent development of client- and server-side logic. That means client applications and server applications should be developed separately without any dependency on each other.
- A client should only know resource URIs, and that's all.

Constraints for RESTful Architecture

- **Stateless Constraint:**
- The stateless constraint specifies that the client and server communication must be stateless between requests. That means the server should not be storing anything related to the client on the server. The request from the client should contain all the necessary information so that the server can identify the client and process that request.
- This ensures that each request can be treated independently by the server

Constraints for RESTful Architecture

- **Cacheable Constraint:**
- In real-time applications, some data provided by the server is not changed that frequently, like the list of Countries, the list of States, the list of Cities, and some master data. RESTful APIs can take advantage of HTTP caching mechanisms. Responses can include cache directives to enable client-side caching, reducing the load on the server and improving performance.

Constraints for RESTful Architecture

- **Layered System**
- RESTful architectures can be composed of multiple layers, where each layer provides specific functionality. For example, REST allows us to use a layered system architecture where we deploy the APIs in server A, store data on server B, and authenticate requests in server C.
- A client cannot simply tell whether it is connected directly to the server or to an intermediary along the way. This allows for scalability, separation of concerns, and ease of maintenance.
- **Content Negotiation:**
- One of the constraints of the REST service is that the client should be able to decide in which format they want the response – whether they want the response in XML or JSON, etc. This is called Content Negotiation.

Constraints for RESTful Architecture

- **Uniform Interface** - The uniform interface constraint defines the interface between the client and the server.
- To understand the uniform interface constraint, we need to understand what a resource is and the HTTP verbs such as GET, PUT, POST & DELETE.
- In the context of a REST API, resources typically represent data entities. Product, Employee, Customer,Country,State etc are all resources.
- The HTTP verb (GET, PUT, POST, DELETE) that is sent with each request tells the API what to do with the resource.
- Each resource is identified by a specific URI (Uniform Resource Identifier). The following table shows some typical requests that you see in an API

Resource	Verb	Outcome
/Employees	GET	Gets list of employees
/Employee/1	GET	Gets employee with Id = 1
/Employees	POST	Creates a new employee
/Employee/1	PUT	Updates employee with Id = 1
/Employee/1	DELETE	Deletes employee with Id = 1

Constraints for RESTful Architecture

- By following these principles, RESTful Web APIs provide a standard and scalable approach to designing and developing web services. They promote loose coupling between clients and servers, enabling easy integration with various platforms and technologies. RESTful APIs are widely used in web and mobile application development, enabling clients to interact with server resources in a flexible and efficient manner.

What are the Differences Between REST and SOAP Services?

- SOAP stands for Simple Object Access Protocol, whereas REST stands for Representational State Transfer.
- SOAP is an XML-based protocol, whereas REST is not a protocol. Rather, it is an architectural pattern, i.e., resource-based architecture.
- SOAP has stateless and state-full implementation specifications, whereas REST is completely stateless.
- SOAP enforces the message format XML, whereas REST does not enforce the message format XML or JSON.
- The SOAP message consists of an envelope with SOAP headers and a body to store the information we want to send. In contrast, REST uses the HTTP build-in headers (with various media types) to store the information and uses HTTP Methods such as GET, POST, PUT, PATCH, and DELETE to perform CRUD operations.
- SOAP is operation-based, where services are exposed as operations, while REST is resource-based, where services are exposed as resources identified by URLs.
- REST is more flexible and offers better performance due to its stateless nature and support for a broader range of data formats. SOAP, with its rigid structure and XML format, is slower. SOAP performance is slow compared to REST.
- REST is generally considered easier to work with and is more commonly used in modern web service applications, especially for public APIs.