Introduction to Web Services and APIs

As described by the World Wide Web Consortium (W3C): *Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML.*

Web services enable applications to communicate with each other. The applications can be entirely different. Consider the following scenario:

- `One application written in Java is running on a Linux host and is using an Oracle database`
- `Another application written in C++ is running on a Windows host and is using an SQL Server database`

These two applications can communicate with each other over the internet with the help of web services.

An application programming interface (API) is a set of rules that enables data transmission between different software. The technical specification of each API dictates the data exchange.

Consider the following example: A piece of software needs to access information, such as ticket prices for specific dates. To obtain the required information, it will make a call to the API of another software (including how data/functionality must be returned). The other software will return any data/functionality requested.

The interface through which these two pieces of software exchanged data is what the API specifies.

You may think Web Services and APIs are quite similar, and you will be correct. See their major differences below.

Web Service vs. API

The terms `web service` and `application programming interface (API)` should not be used interchangeably in every case.

- `Web services are a type of application programming interface (API). The opposite is not always true!`
- `Web services need a network to achieve their objective. APIs can achieve their goal even offline.`
- `Web services rarely allow external developer access, and there are a lot of APIs that welcome external developer tinkering.`
- `Web services usually utilize SOAP for security reasons. APIs can be found using different designs, such as XML-RPC, JSON-RPC, SOAP, and REST.`
- `Web services usually utilize the XML format for data encoding. APIs can be found using different formats to store data, with the most popular being JavaScript Object Notation (JSON).`

Web Service Approaches/Technologies

There are multiple approaches/technologies for providing and consuming web services:

- `XML-RPC`
  - `XML-RPC uses XML for encoding/decoding the remote procedure call (RPC) and the respective parameter(s). HTTP is usually the transport of choice.`
  - Code: http

```http
--> POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
     <param>
           <value><i4>41</i4></value>
           </param>
        </params>
  </methodCall>

<-- HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
            <value><string>South Dakota</string></value>
            </param>
      </params>
 </methodResponse>
```

The payload in XML is essentially a single `<methodCall>` structure. `<methodCall>` should contain a `<methodName>` sub-item, that is related to the method to be called. If the call requires parameters, then `<methodCall>` must contain a `<params>` sub-item.

- `JSON-RPC`
  - `JSON-RPC uses JSON to invoke functionality. HTTP is usually the transport of choice.`

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

- Code: http

  ```http
  --> POST /ENDPOINT HTTP/1.1
   Host: ...
   Content-Type: application/json-rpc
   Content-Length: ...

  {"method": "sum", "params": {"a":3, "b":4}, "id":0}

  <-- HTTP/1.1 200 OK
   ...
   Content-Type: application/json-rpc

   {"result": 7, "error": null, "id": 0}
  ```

The `{"method": "sum", "params": {"a":3, "b":4}, "id":0}` object is serialized using JSON. Note the three properties: `method`, `params` and `id`. `method` contains the name of the method to invoke. `params` contains an array carrying the arguments to be passed. `id` contains an identifier established by the client. The server must reply with the same value in the response object if included.

- `SOAP (Simple Object Access Protocol)`
  - SOAP also uses XML but provides more functionalities than XML-RPC. SOAP defines both a header structure and a payload structure. The former identifies the actions that SOAP nodes are expected to take on the message, while the latter deals with the carried information. A Web Services Definition Language (WSDL) declaration is optional. WSDL specifies how a SOAP service can be used. Various lower-level protocols (HTTP included) can be the transport.
  - Anatomy of a SOAP Message
    - `soap:Envelope`: (Required block) Tag to differentiate SOAP from normal XML. This tag requires a `namespace` attribute.
    - `soap:Header`: (Optional block) Enables SOAP’s extensibility through SOAP modules.
    - `soap:Body`: (Required block) Contains the procedure, parameters, and data.
    - `soap:Fault`: (Optional block) Used within `soap:Body` for error messages upon a failed API call.

  - Code: http

    ```http
    --> POST /Quotation HTTP/1.0
    Host: www.xyz.org
    Content-Type: text/xml; charset = utf-8
    Content-Length: nnn

    <?xml version = "1.0"?>
    <SOAP-ENV:Envelope
      xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
       SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

      <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/quotations">
         <m:GetQuotation>
           <m:QuotationsName>MiscroSoft</m:QuotationsName>
        </m:GetQuotation>
      </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>

    <-- HTTP/1.0 200 OK
    Content-Type: text/xml; charset = utf-8
    Content-Length: nnn

    <?xml version = "1.0"?>
    <SOAP-ENV:Envelope
     xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
      SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

    <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/quotation">
        <m:GetQuotationResponse>
           <m:Quotation>Here is the quotation</m:Quotation>
       </m:GetQuotationResponse>
     </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
    ```

Note: You may come across slightly different SOAP envelopes. Their anatomy will be the same, though.

- `WS-BPEL (Web Services Business Process Execution Language)`
  - WS-BPEL web services are essentially SOAP web services with more functionality for describing and invoking business processes.
  - WS-BPEL web services heavily resemble SOAP services. For this reason, they will not be included in this module's scope.
- `RESTful (Representational State Transfer)`
  - REST web services usually use XML or JSON. WSDL declarations are supported but uncommon. HTTP is the transport of choice, and HTTP verbs are used to access/change/delete resources and use data.

  - Code: http

    ```http
    --> POST /api/2.2/auth/signin HTTP/1.1
    HOST: my-server
    Content-Type:text/xml

    <tsRequest>
      <credentials name="administrator" password="passw0rd">
        <site contentUrl="" />
      </credentials>
    </tsRequest>
    ```

  - Code: http

    ```http
    --> POST /api/2.2/auth/signin HTTP/1.1
    HOST: my-server
    Content-Type:application/json
    ```

```
Accept:application/json

{
 "credentials": {
   "name": "administrator",
  "password": "passw0rd",
  "site": {
    "contentUrl": ""
   }
  }
}
```

Similar API specifications/protocols exist, such as Remote Procedure Call (RPC), SOAP, REST, gRPC, GraphQL, etc.

Do not feel overwhelmed! In the following sections, you will have the opportunity to interact with different web services and APIs.