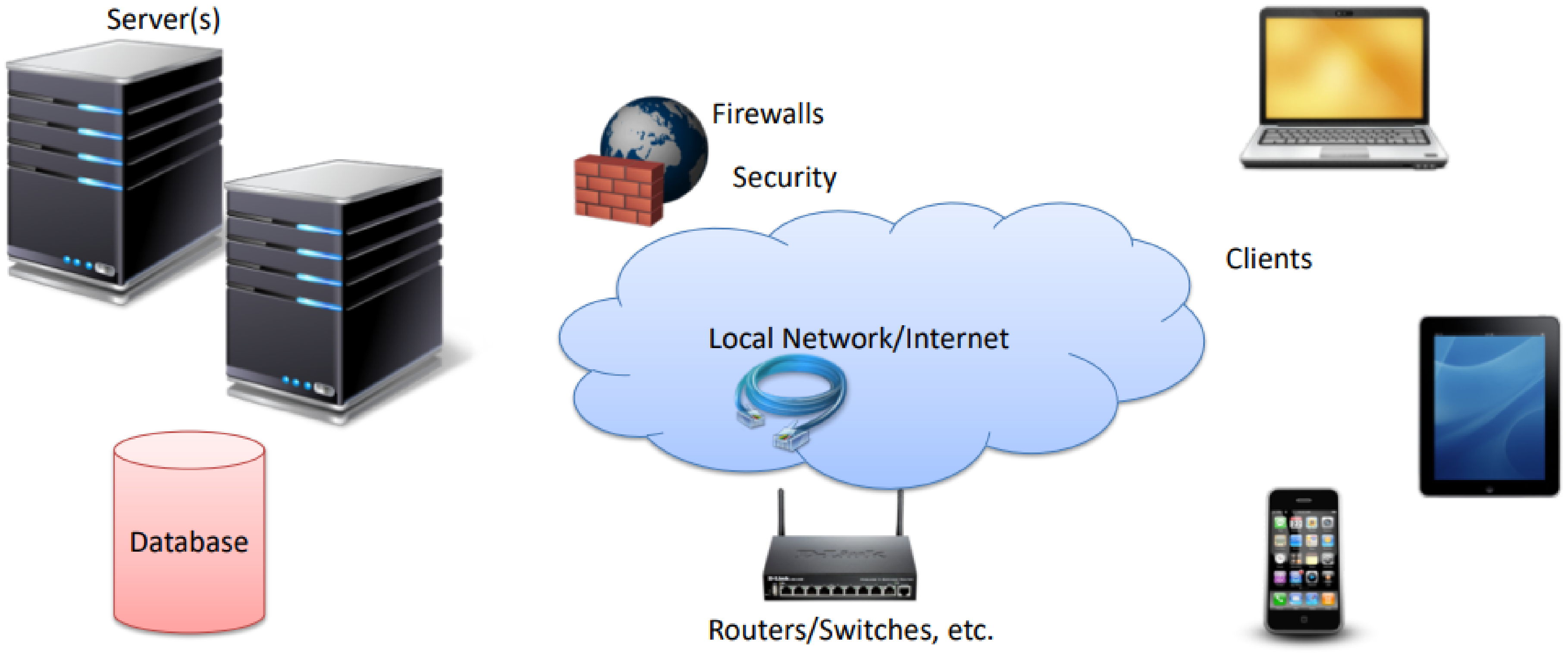
Unit -4

Java Web Services

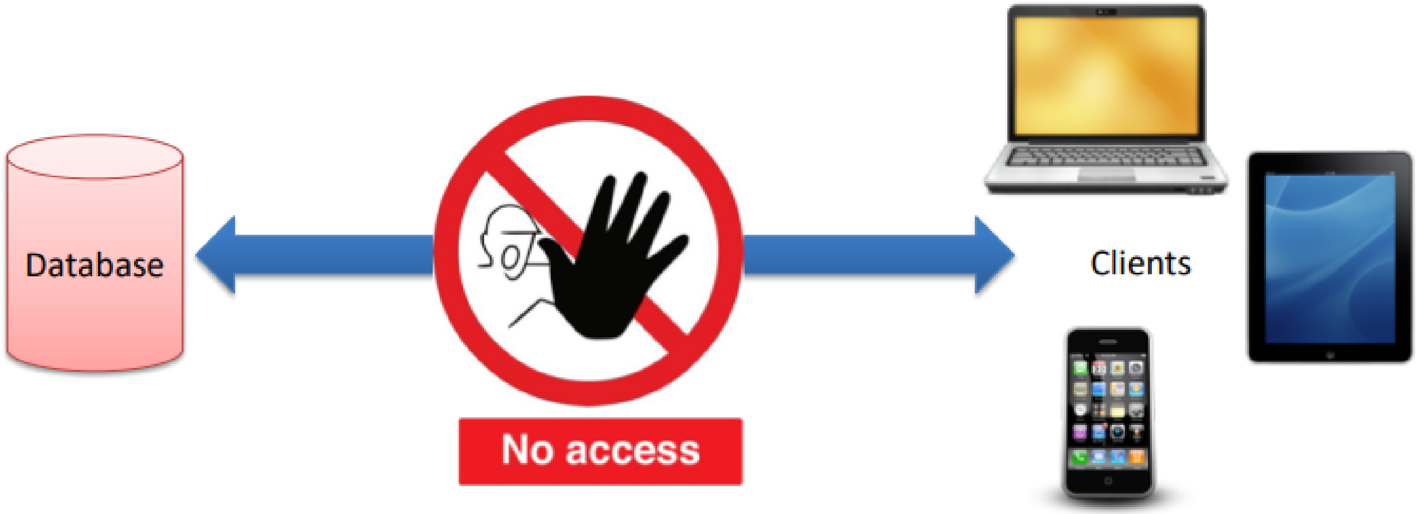
Problem

How to Share Data between Devices in a Network?



Problem

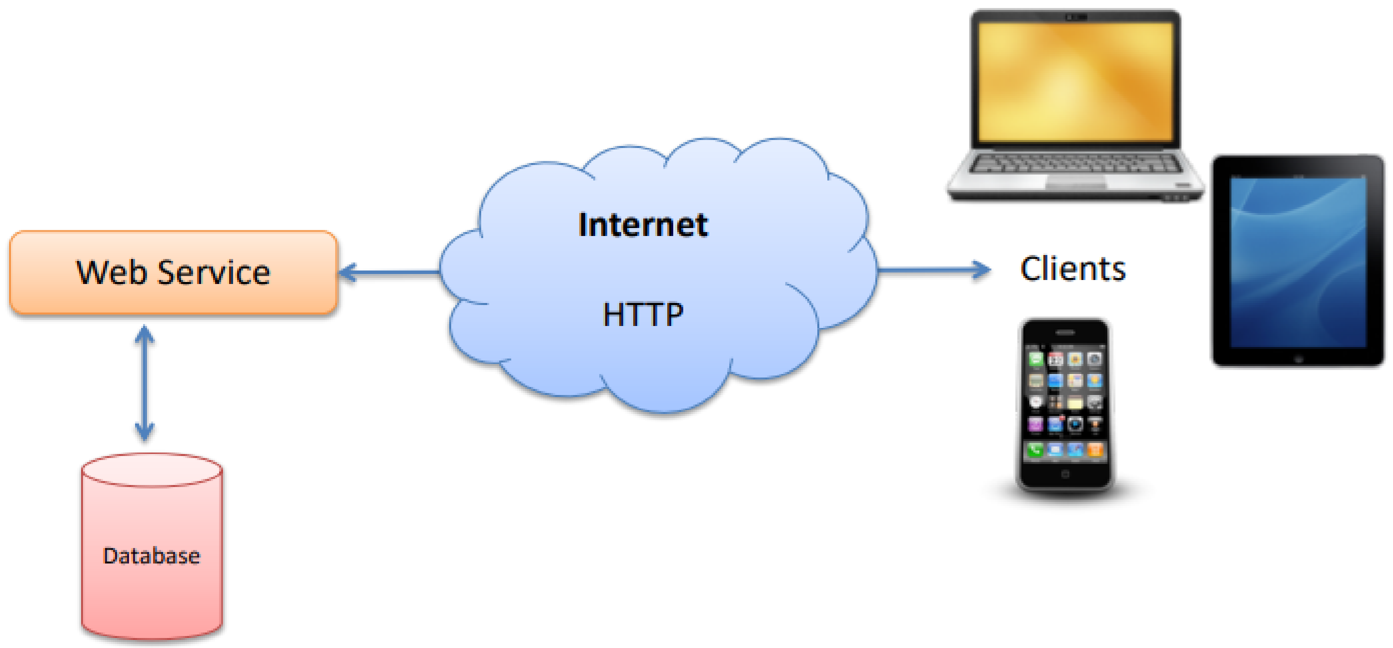
How to Share Data between Devices in a Network?



Direct Connection between the Database and the Clients that need the Data is normally not possible, due to security, compatibility issues, etc. (Firewalls, Hacker Attacks, etc.)

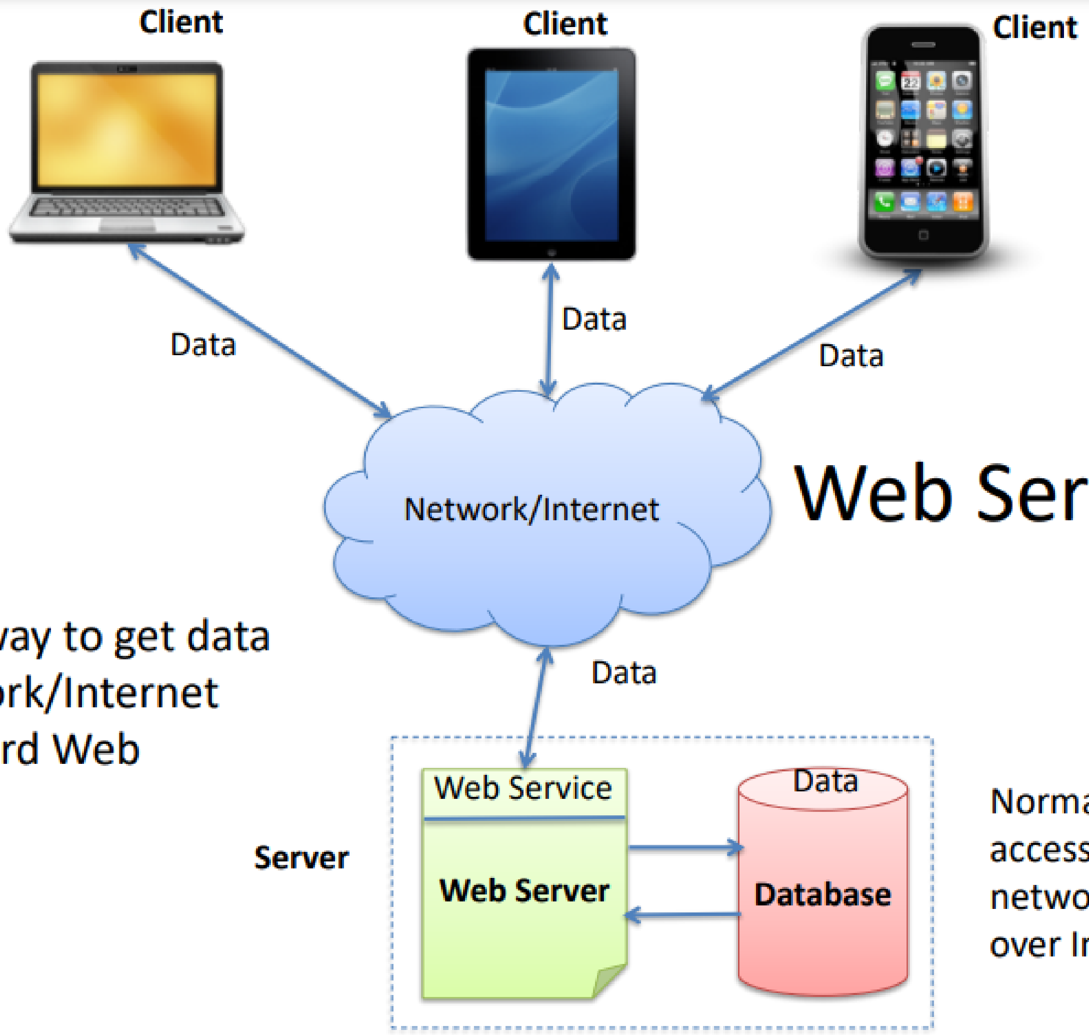
Direct Connection in a Local Network (behind the Firewall) is normally OK – but not over the Internet

Solution: Web Service



Web Services uses standard web protocols like HTTP, etc.
HTTP is supported by all Web Browser, Servers and many Programming Languages

Web Services



Web Services:

- A Standard way to get data over a network/Internet
- Using standard Web protocols

Normally you don't have direct access to a Database over a network, and especially not over Internet

What is web service ?

- ✔ Web services allow communication between different applications over the internet.
- ✔ They enable interoperability between different programming languages and platforms.
- ✔ Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
- ✔ A web service is a collection of open protocols and standards used for exchanging data between applications or systems.

What is web service ?

- ❖ In sort, a complete web service is, therefore, any service that –
  - ✔ Is available over the Internet or private (intranet) networks
  - ✔ Uses a standardized XML messaging system
  - ✔ Is not tied to any one operating system or programming language
  - ✔ Is self-describing via a common XML grammar
  - ✔ Is discoverable via a simple find mechanism

What is web service ?

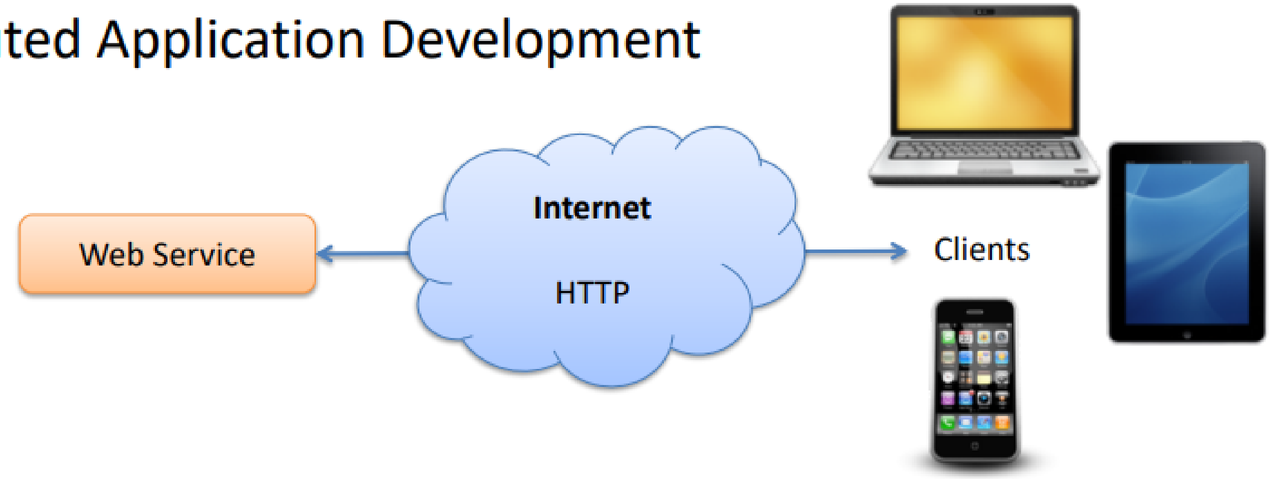
- A Web service is a method of communications between two devices over the World Wide Web.
- Web API
- Standard defined by W3C
- Cross-platform
- Web Services can be implemented and used in most Programming Languages (C#/ASP.NET, PHP, LabVIEW, Objective-C, Java, ...)
- Uses standard Web technology (Web protocols)
  - HTTP, REST, SOAP, XML, WSDL, JSON, ...

Why Web Services?

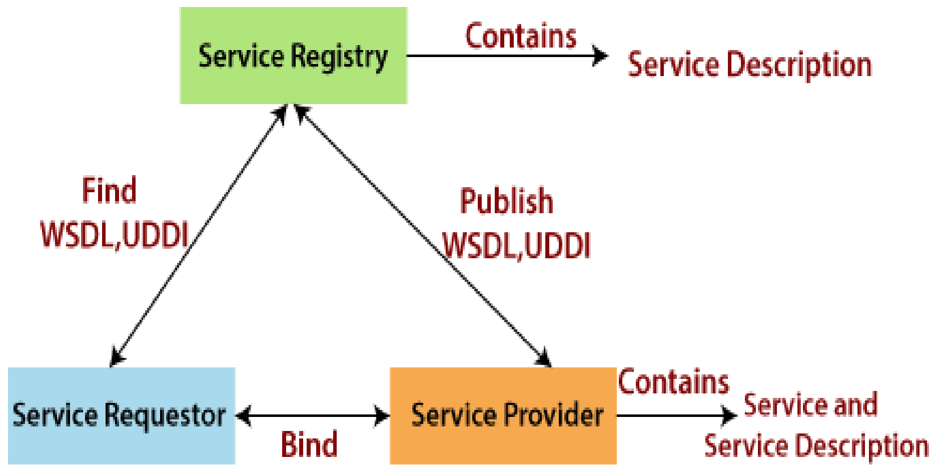
- Exposing the Existing Function on the network
- Interoperability
- Standardized Protocol
- Low Cost Communication

Why Web Services?

- Today Web Services have been very popular
- Easy Data sharing over Internet
- Platform-independent Communication
- Makes it possible of integration of different systems and platforms
- Distributed Application Development



Architecture



Three Roles

- ✔ Service Provider
- ✔ Service Requestor
- ✔ Service Registry

Operations in a Web Service Architecture

- ✔ Publication of service descriptions **(Publish)**
- ✔ Finding of services descriptions **(Find)**
- ✔ Invoking of service based on service descriptions **(Bind)**

Architecture

- **Client**: The application that sends requests to a web service.
- **Web Service**: The server-side application providing functionalities.
- **Registry (UDDI)**: A directory for discovering web services.
- **Transport (HTTP, SMTP, FTP)**: The communication protocol used.
- **Messaging (SOAP, REST)**: Defines how data is formatted and exchanged.

continue..

- The Web Services architecture describes how to instantiate the elements and implement the operations in an interoperable manner.
- The architecture of web service interacts among three roles: **service provider**, **service requester**, and **service registry**.
- The interaction involves the three operations: **publish, find**, and **bind**. These operations and roles act upon the **web services artifacts**.
- The web service artifacts are the web service software module and its description.

continue..

- The service provider hosts a network-associable module (web service).
- It defines a service description for the web service and publishes it to a service requestor or service registry.
- These service requestor uses a find operation to retrieve the service description locally or from the service registry.
- It uses the service description to bind with the service provider and invoke with the web service implementation.

continue..

- **Service Provider**

From an architectural perspective, it is the platform that hosts the services.

- **Service Requestor**

Service requestor is the application that is looking for and invoking or initiating an interaction with a service. The browser plays the requester role, driven by a consumer or a program without a user interface.

- **Service Registry**

Service requestors find service and obtain binding information for services during development.

Operations in a Web Service Architecture

- **Three behaviors that take place in the microservices:**
- Publication of service descriptions **(Publish)**

  a service description must be published so that a service requester can find the service.

- Finding of services descriptions **(Find)**

  the service requestor retrieves the service description directly. It can be involved in two different lifecycle phases for the service requestor:

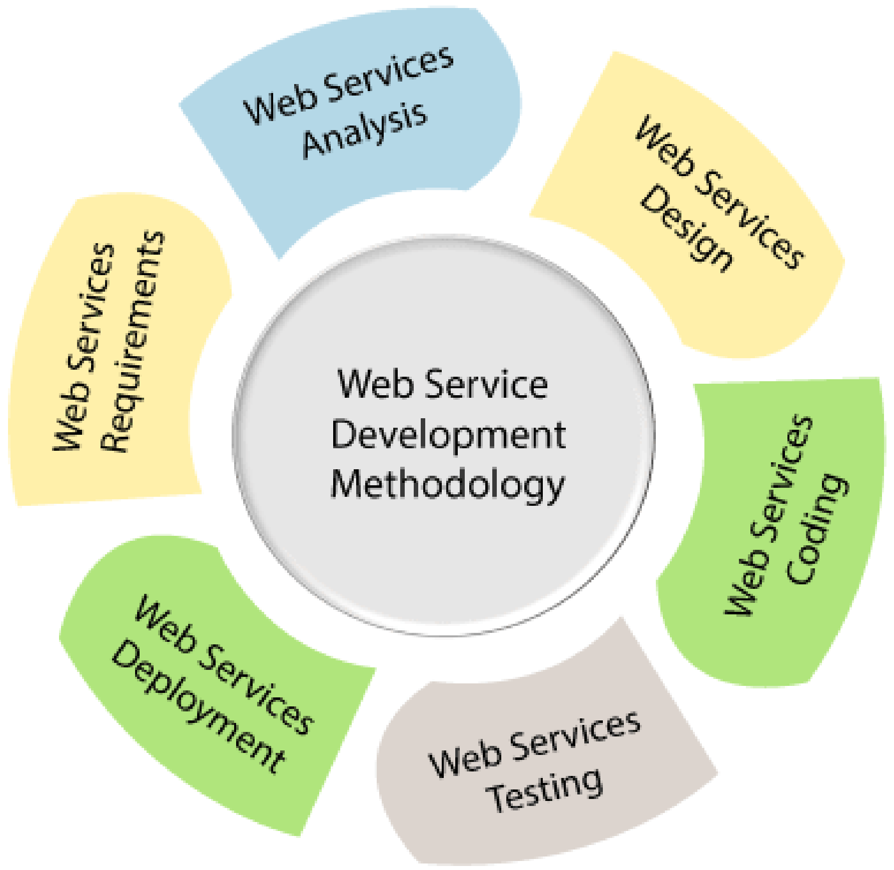
- At design, time to retrieve the service's interface description for program development.
- And, at the runtime to retrieve the service's binding and location description for invocation.

continue..

- Invoking of service based on service descriptions **(Bind)**

In the bind operation, the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact, and invoke the service.

Web Service Implementation Lifecycle



Web Service Implementation Lifecycle

Web Service Implementation Lifecycle

- A web service implementation lifecycle refers to the phases for developing web services from the requirement to development. An Implementation lifecycle includes the following phases:
- Requirements Phase
- Analysis Phase
- Design Phase
- Coding Phase
- Test Phase
- Deployment Phase

Requirements Phase

- The objective of the requirements phase is to understand the business requirement and translate them into the web services requirement.
- The requirement analyst should do requirement elicitation (it is the practice of researching and discovering the requirements of the system from the user, customer, and other stakeholders).
- The analyst should interpret, consolidate, and communicate these requirements to the development team.
- The requirements should be grouped in a centralized repository where they can be viewed, prioritized, and mined for interactive features.

Analysis Phase

- The purpose of the analysis phase is to refine and translate the web service into conceptual models by which the technical development team can understand.
- It also defines the high-level structure and identifies the web service interface contracts.

Design Phase

- In this phase, the detailed design of web services is done.
- The designers define web service interface contract that has been identified in the analysis phase.
- The defined web service interface contract identifies the elements and the corresponding data types as well as mode of interaction between web services and client.

Coding Phase

- Coding and debugging phase is quite similar to other software component-based coding and debugging phase.

- The main difference lies in the creation of additional web service interface wrappers, generation of WSDL, and client stubs.

Test Phase

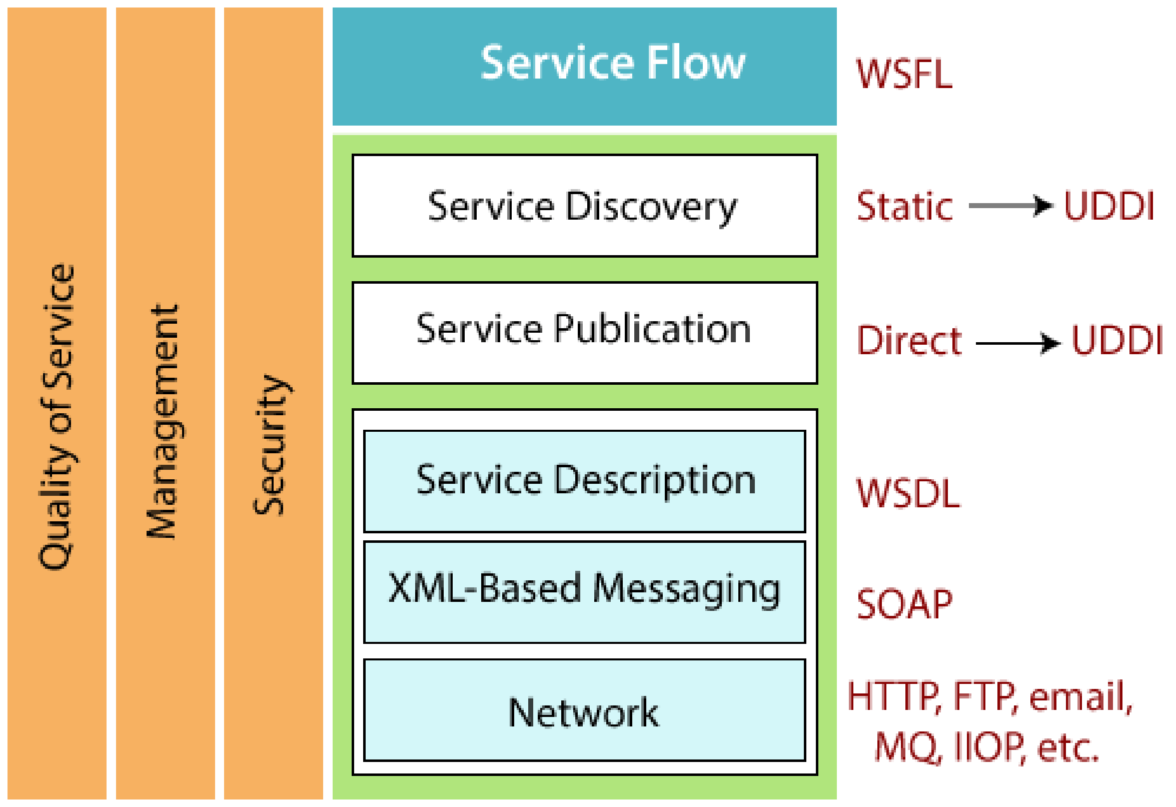
- In this phase, the tester performs interoperability testing between the platform and the client's program.
- Testing to be conducted is to ensure that web services can bear the maximum load and stress.
- Other tasks like profiling of the web service application and inspection of the SOAP message should also perform in the test phase.

Deployment Phase

- The purpose of the deployment phase is to ensure that the web service is properly deployed in the distributed system.
- It executes after the testing phase.
- The primary task of deployer is to ensure that the web service has been properly configured and managed.
- Other optional tasks like specifying and registering the web service with a UDDI registry also done in this phase.

Web Service Stack or Web Service Protocol Stack

The Conceptual Web Services Stack



Web Services conceptual Stack

Web Service Stack or Web Service Protocol Stack

- In the above figure, the top most layers build upon the capabilities provided by the lower layers.
- The three vertical towers represent the requirements that are applied at every level of the stack.
- The text on the right represents technologies that apply at that layer of the stack. A web service protocol stack typically stacks four protocols:
- Transport Protocol
- Messaging Protocol
- Description Protocol
- Discovery Protocol

(Service) Transport Protocol:

- The network layer is the foundation of the web service stack. It is responsible for transporting a message between network applications.
- HTTP is the network protocol for internet available web services. It also supports other network protocol such as **SMTP**, **FTP**, and **BEEP** (Block Extensible Exchange Protocol).

(XML) Messaging Protocol:

- It is responsible for encoding message in a common XML format so that they can understand at either end of a network connection.

- SOAP is the chosen XML messaging protocol because it supports three operations: publish, find, and bind operation.

(Service) Description Protocol:

- It is used for describing the public interface to a specific web service. WSDL is the standard for XML-based service description.
- WSDL describes the interface and mechanics of service interaction. The description is necessary to specify the **business context, quality of service**, and **service-to-service** relationship.

(Service) Discovery Protocol:

- It is a centralized service into a common registry so that network Web services can publish their location and description.
- It makes it easy to discover which services are available on the network.

Web Service Stack or Web Service Protocol Stack

- The first three layers of the stack are required to provide or use any web service. The simplest stack consists of HTTP for the network layer, SOAP protocol for the XML-based messaging, and WSDL for the service description layer.
- These three-layer provides interoperability and enables web service to control the existing internet infrastructure. It creates a low cost of entry to a global environment.
- The bottom three layers of the stack identify technologies for compliance and interoperability, the next two layer- **Service Publication** and **Service Discovery** can be implemented with a range of solutions.

Web Service Features

XML-Based

- Web services typically use XML (Extensible Markup Language) as the standard format for data exchange. XML provides a structured and platform-independent way to represent data, making it suitable for interoperability between different systems.

Web Service Features

Loosely Coupled

- Web services are designed to be loosely coupled, meaning that the client and server components can evolve independently.
- The client does not need to have prior knowledge of the server's implementation details, and changes in one component do not necessarily require changes in the other.
- A tightly coupled system means that the client and server logic are closely tied to one another, indicating that if one interface changes, then another must be updated.
- Accepting a loosely coupled architecture tends to make software systems more manageable and allows more straightforward integration between various systems.

Web Service Features

Coarse-Grained

- Web services often follow a coarse-grained approach, where the operations exposed by the service deal with significant amounts of data or perform complex tasks. This reduces the number of remote calls needed, improving efficiency and performance.
- Object-oriented technologies such as Java expose their functions through individual methods.
- Web services technology implement a natural method of defining coarse-grained services that approach the right amount of business logic.

Web Service Features

Ability to be Synchronous or Asynchronous

- Synchronicity specifies the binding of the client to the execution of the function. In synchronous invocations, the client blocks and delays in completing its service before continuing.
- In asynchronous communication, the client can continue with other tasks while waiting for a response, improving scalability and responsiveness.

Web Service Features

Supports Remote Procedure Calls (RPCs)

- Web services support remote procedure calls, allowing clients to invoke methods or functions on a remote server.
- This enables the client to request specific operations or actions to be performed on the server, enhancing the interaction between distributed systems.

Web Service Features

Supports Document Exchange

- Web services facilitate document exchange, allowing the transfer of structured information between different systems. This enables applications to share and collaborate on data, improving data integration and interoperability.

Types of web service

- Two types
  1) SOAP(JAX-RPC, JAX-WS)
  2) RESTFUL(JAX-RS)

Types of web service

Web Services 1.0

- “SOAP Web Services”
- Using the **SOAP** protocol (Simple Object Access Protocol)
- **XML** (Extensible Markup Language)

Visual Studio: ASP.NET ASMX Web Service

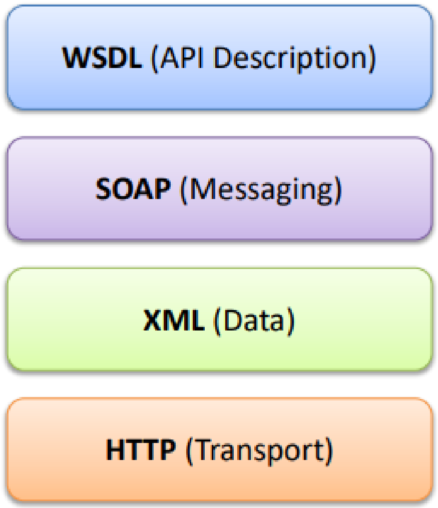
Web Services 2.0

- “RESTful Web Services”
- Using the **REST** protocol (Representational State Transfer)
- Uses standard **HTTP** methods (GET, PUT, POST, DELETE) (HTTP: Hypertext Transfer Protocol)
- Uses **JSON** (JavaScript Object Notation) or XML
- Visual Studio: ASP.NET Web API

SOAP Web Services

- SOAP stands for Simple Object Access Protocol. It is a XML-based protocol for accessing web services.
- SOAP is a W3C recommendation for communication between two applications.
- SOAP is XML based protocol. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

SOAP Web Services



- WSDL - Web Services Description Language
- SOAP - Simple Object Access Protocol
- XML – Extensible Markup Language
- HTTP - Hypertext Transfer Protocol

Advantages - Disadvantages

- **WS Security**: SOAP defines its own security known as WS Security.
- **Language and Platform independent**: SOAP web services can be written in any programming language and executed in any platform.

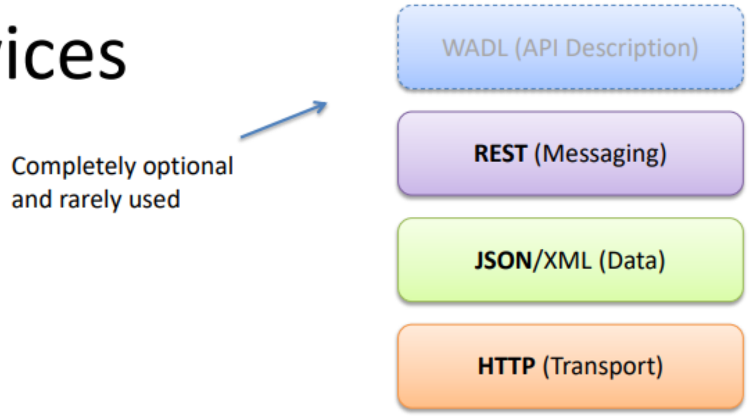
✔ **Disadvantages**

- **Slow**: SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.
- **WSDL dependent**: SOAP uses WSDL and doesn't have any other mechanism to discover the service.

RESTful Web Services

- REST stands for REpresentational State Transfer.
- REST is an architectural style not a protocol.

RESTful Web Services



- WADL – Web Application Description Language
- REST - Representational State Transfer
- XML – Extensible Markup Language
- JSON - JavaScript Object Notation
- HTTP - Hypertext Transfer Protocol

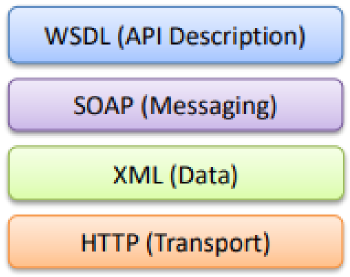
Advantages

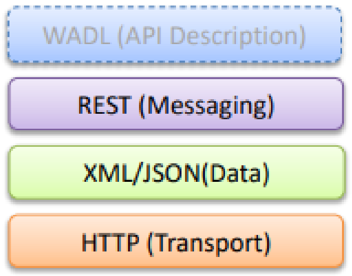
- **Fast**: RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.
- **Language and Platform independent**: RESTful web services can be written in any programming language and executed in any platform.
- **Can use SOAP**: RESTful web services can use SOAP web services as the implementation.
- **Permits different data format**: RESTful web service permits different data format such as Plain Text, HTML, XML and JSON.

SOAP vs. REST

SOAP:

- A service architecture
- XML based
- Runs on HTTP but envelopes the message
- Slower than REST
- Very mature, a lot of functionality
- Not suitable for browser-based clients
- More complicated

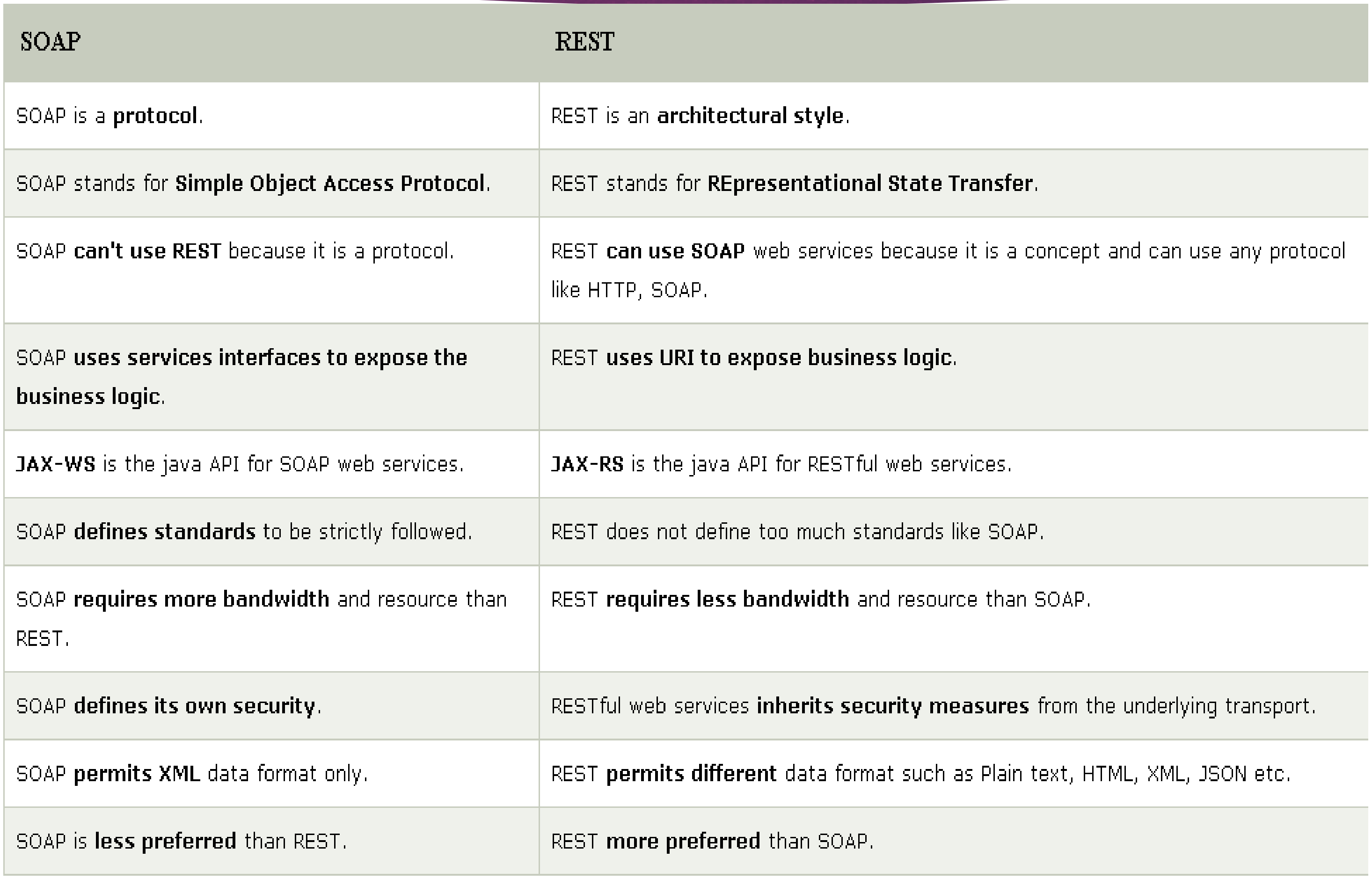




REST:

- A service architecture
- Uses the HTTP headers to hold meta information
- Can be used with XML, JSON or whatever necessary
- Usually used with JSON due to the easily parsable content
- Faster than SOAP

Soap Vs Rest



SOAP	REST
SOAP is a **protocol**.	REST is an **architectural style**.
SOAP stands for **Simple Object Access Protocol**.	REST stands for **REpresentational State Transfer**.
SOAP **can't use REST** because it is a protocol.	REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
SOAP **uses services interfaces to expose the business logic**.	REST **uses URI to expose business logic**.
JAX-WS is the java API for SOAP web services.	**JAX-RS** is the java API for RESTful web services.
SOAP **defines standards** to be strictly followed.	REST does not define too much standards like SOAP.
SOAP **requires more bandwidth** and resource than REST.	REST **requires less bandwidth** and resource than SOAP.
SOAP **defines its own security**.	RESTful web services **inherits security measures** from the underlying transport.
SOAP **permits XML** data format only.	REST **permits different** data format such as Plain text, HTML, XML, JSON etc.
SOAP is **less preferred** than REST.	REST **more preferred** than SOAP.

RESTful Web Services

- **REST (Representational State Transfer)** follows an architectural style that enables communication between client and server applications over the web.
- In REST-based web services, everything is treated as a resource, accessible via a unique URL.
- RESTful web services facilitate interaction between software applications running on different platforms and frameworks.

RESTful Web Services

- These services operate on a **code-on-demand** principle, where clients can execute logic from the server if needed.
- A RESTful web service is invoked by sending an HTTP request to a specific URL, and the response contains the result.
- The Jersey framework simplifies the development of RESTful web services in Java, making it an efficient tool for building REST APIs.

RESTful Architectural Principles

- **Statelessness:** Each request is independent and must contain all necessary information.
- **Client-Server:** Client and server are separate and communicate via standard protocols.
- **Uniform Interface:** Standardized conventions for interacting with resources.
- **Cacheability:** Responses can be cached to improve performance.

RESTful Architectural Principles

- **Layered System:** The system is composed of multiple layers, each with specific responsibilities.
- **Code on Demand (optional):** Servers can send executable code to clients for additional functionality.
- **Resource-Based:** Everything is treated as a resource, and operations are performed on resources using HTTP methods.

HTTP Methods & URI Matching

- GET: Retrieve data.
- POST: Submit data.
- PUT: Update data.
- DELETE: Remove data.
- Example URI pattern: /api/students/{id}

URI Matching

- URI Patterns:
- Collection URIs: These represent a group of resources. For example:
  - GET /users → A request to retrieve all users.
  - POST /users → A request to create a new user.
- Single Resource URIs: These represent a specific instance of a resource. For example:
  - GET /users/{id} → A request to retrieve a specific user by ID.
  - PUT /users/{id} → A request to update a specific user by ID.
  - DELETE /users/{id} → A request to delete a specific user by ID.

Continue..

- URL Parameters: Example:
  - GET /users?age=25 → Retrieve users that are 25 years old (query parameter).
- In RESTful web services:
- **HTTP methods** define the type of action (operation) to be performed on a resource.
- **URIs** identify the resource (or collection of resources) on which the operation is to be performed.
- The combination of these methods and URIs forms the foundation of how RESTful APIs operate, ensuring that operations on resources are clear, consistent, and standardized across the web.

JAX-RS Injection

- @PathParam: Extracts values from URI path.
- @QueryParam: Retrieves query parameters.
- @FormParam: Captures form input data.
- Example

```
@Path("/students")
public class StudentResource {
 @GET
@Path("{id}")
public String getStudent(@PathParam("id") int id) {
return "Student ID: " + id;     }
 }
```

Server Responses & Exception Handling

- Response Codes:
- 200 OK: Successful request.
- 201 Created: Resource created.
- 400 Bad Request: Invalid input.
- 500 Internal Server Error: Server issue.
- **Handling Exceptions**:

```
@Provider
 public class CustomExceptionMapper implements
ExceptionMapper<Exception> {
  public Response toResponse(Exception ex) {
  return Response.status(500).entity(ex.getMessage()).build();
}
  }
```

JAX-RS Client API

- Used to make REST calls from Java applications.
- **Example:**

```
Client client = ClientBuilder.newClient();
WebTarget target = client.target("http://example.com/api/
students/1");
String response = target.request(MediaType.TEXT_PLAIN).
get(String.class);
System.out.println(response);
```

JAX-RS Client API

- The **JAX-RS Client API** simplifies the task of consuming RESTful web services from a Java application.
- It abstracts the details of HTTP request/response handling and provides a clean, fluent interface to interact with REST APIs.
- With support for asynchronous requests, request customization, and easy error handling, it’s an excellent choice for integrating RESTful APIs into Java applications.

Thank You