# Introduction

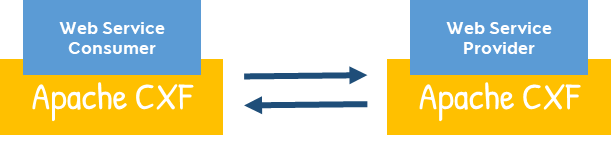
* A web services stack or engines provide us with various tools that we use to develop our web services.
* They also serialize and deserialize the upcoming requests into language objects and back on the wire to the **XML** or **JSON** formatted messages.
* They dispatch the incoming web services requests from the clients to the appropriate web services classes and methods that we code (to handle the requests).

There are many of them but we are interesting in **Apache** **CXF** and we will learn how to create maven project using **Apache** **CXF** from scratch.

# Apache CXF Overview

## Web Service Engine

The various tools provided by web services stacks or engines allow us:



* To build and run web services.
* As developers to focus on the business logic and the application itself.

Using **Apache** **CXF**, we can develop both the web services **providers** and web services **consumers** for both **SOAP** as well as **restful** web services.

## What is CXF?

Apache CXF is one of the powerful and popular services engines in the java space because it implements both **JAXWS** and **JAXRS**. Using **Apache** **CXF**, we can develop both the web services providers and web services consumers for both **SOAP** and **Restful** web services.

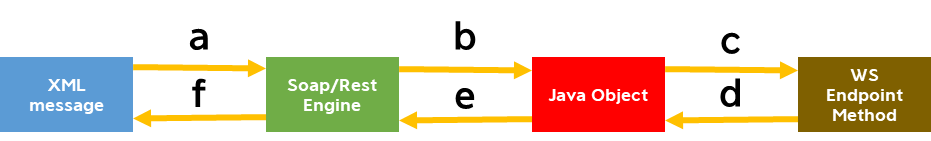
## Why CXF?

CXF comes with a **SOAP** and **Rest** engine, which at a minimum do two things:

1. It serializes the **XML** or **JSON** or any other formatted messages requests and responses into **Java** objects. It deserialize the Java objects to **XML** or **JSON**.
2. It dispatches the incoming request to the appropriate web service endpoint.

When an **XML** message comes in:

1. The engine takes that web service message request.
2. It converts the message to Java object.
3. Hands the Java object to an appropriate method on the web service endpoint.
4. It then takes the response from these web services classes method
5. It converts it back into **XML** or **JSON** or any other formatted format.
6. Finally, it send it back to the client.



## Web Service Standards

It implements almost all the Web services standards out there like:

* WS-Security
* WS-Transaction Management
* WS-Policy

We can simply configure these standards to our application using configuration files.

## Tools

We need tools to develop web services providers and consumers.

|  |  |
| --- | --- |
| **WSDL2JAVA** | Allow us to generate code from a **WSDL** file and implement our web services provider. It will also allow us to implement a web service client from these stubs that get generated using **WSDL2JAVA**. |
| **JAVA2WSDL** | Allow us to code first, development from **Java** to **WSDL**. |
| **Note** | |
| We can build our stubs automatically using mavin plugin. | |

## Spring configuration

It can be done with two ways:

1. By annotation
2. **CXF** uses **Spring** to configure all the web services endpoints and the other features.

## Extend and Customize

It also easy to extend and customize **CXF** using the interceptors or handlers **JAX**-**WS** handlers. If we want to add custom things, that **CXF** does not have.

## Documentation and Samples

**CXF** comes with a lot of documentation on their website and it uses a lot of samples examples. We can visit the offical web site for more information. (http://cxf.apache.org/)

# CXF and Spring Boot

Apache **CXF** makes it super easy to create web service applications by supporting the use of **spring boot.** Once we start the use of **spring** boot with **CXF** we have:

1. We simply add a **CXF Spring** dependency (a maven dependency) to the main pom.xml. Once we add this dependency and automatically all the other jar files that are required to build **RESTful** services or **SOAP** services will be transitively dragged in.
2. It also makes the publishing of the **RESTful** endpoint very easy. We simply add a property called

**cxf.jaxrs.component-scan=true**. Once we do that in the application.properties (which is the spring boot properties file), automatically all your web services endpoints will be discovered and published. Therefore, there is no need for **Java** configuration.

1. We can add the server context path. Using this, you can define a web application context for your **RESTful** or **SOAP** application.

# Spring Boot

# Create the SOAP project

In this and the next few lectures, we will create a simple Hello World SOAP web service in four simple steps:

1. Create the Project (by creating this **spring** **boot** project).
2. Create the web service endpoint class itself.
3. Create the configuration via **Spring** **Java** configuration class (the class will publish our endpoint).
4. We run the application and view the **WSDL** getting generated.

# Create the endpoint

To create a web service endpoint:

1. Inside the main source folder, we need to create a Java class named **HelloWorldWebService**.
2. We add to the last class a **String** method with the name **sayHelloWorld,** which returns a string (“**Hello World from HelloWorldWebService**”).

**Note:** The class is just a simple Java class (POJO),

1. To make it a SOAP endpoint, we mark it using **@WebService** annotation from the **JAX-WS** **API**.
2. Now we have to annotate the method with **@WebMethod** always from the **JAX-WS** **API**.

**Note:** if you go to and look at the maven dependencies, you will see all the **CXF** dependencies.

# Create the configuration class

# Run the application

# Change the web application context

# Test using SoapUI

# Enable Logging Feature

# Using Spring Boot 2.X

# Change the Context Property

# Section Summary