Web Services:
SOAP, REST and OAS

Final Project Presentation

Comp. Sci. 9831B : Software Engineering for Cyber-Physical Systems

Instructor: Prof. Kostas Kontogiannis

Aasminpreet Singh Kainth

Department of Computer Science

Middlesex College, Western University

Agenda

1. Web Services
2. SOAP
3. REST
4. SOAP vs REST
5. Selection of Soap and Rest protocols
6. OpenAPI Specification (OAS)
7. Introduction to serverless architecture

Web Service

A **Web service** is a way for two machines to communicate with each other over a network.

A web server running on a computer listens for requests from other computers. When a request from another computer is received, over a network, the Web service returns the requested resources. This resource could be JSON, XML, an HTML file, Images, Audio Files, etc.

For ex. Weather application

According to W3C, “Web Services are the message-based design frequently found on the Web and in enterprise software. The Web of Services is based on technologies such as HTTP, XML, SOAP, WSDL, SPARQL, and others.”

Sometimes the term “Web service” is mistaken for the term “Website.”

Web services do not provide any GUI (Graphical User Interface) for the user, as Websites do.

A Website is a collection of Web pages and a Web page can contain multiple Web services.

Web Service

Components of Web Services

The web services must be able to fulfill the following conditions:

Accessible over the internet

Discoverable through a common mechanism like *UDDI*

Interoperable over any programming language or Operating System

Uses of Web Services

Reusing the code and connecting the existing program

Linking data between two different platforms

Interoperability between disparate applications.

SOAP as a Web Service

SOAP

Lightweight protocol designed for the exchange of information

Based on XML (enables messages to be exchanged between diverse parties)

Messages can be exchanged over different protocols such as HTTP, SMTP and FTP

Structure of a SOAP message

SOAP is based on message exchanges

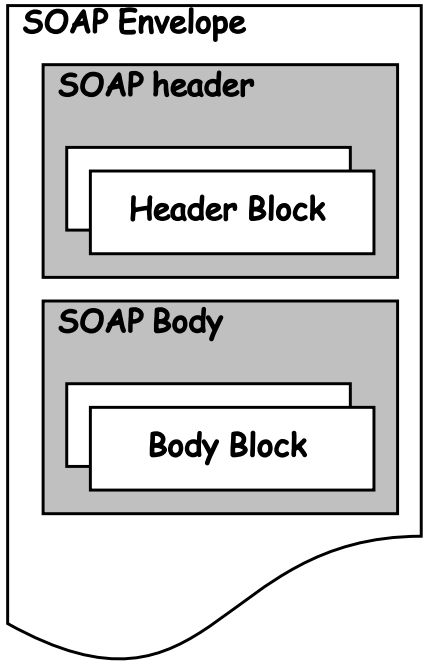
Messages are seen as envelopes where the application encloses the data to be sent

A message has two main parts

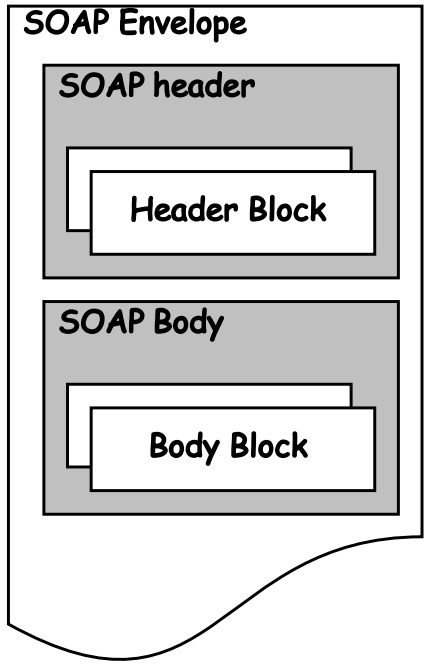
1. header 2. body

SOAP does not specify what to do with the header and the body, it only states that the header is optional, and the body is mandatory

The use of header and body is implicit.



Structure of a SOAP message



SOAP Envelope element

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <WesternWebService xmlns="http://tempuri.org/">
      <TutorialID>int</TutorialID>
    </Guru99WebService>
  </soap:Body>
</soap:Envelope>
```

SOAP Body element

Name of the web service

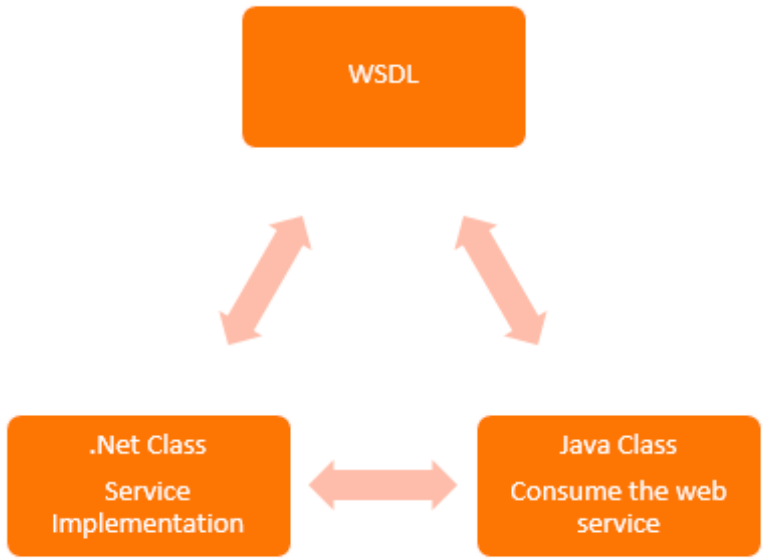
Parameter required by the web service

Web Services Description Language (WSDL)

Web services have a description language known as WSDL, which is used to describe the web service.

The WSDL file is what binds everything together.

WSDL file is in XML, which can be understood by any programming language



Restful Web Service

REST (REpresentational State Transfer)

Architecture for developing Web services

Makes use of the same architecture that use HTTP or similar protocols

Interacting with stateful resources, rather than messages or operations.

Restful Web Service

Lightweight, maintainable, and scalable service that is built on the REST architecture.

Expose API from the application in a secure, uniform, stateless manner to the calling client.

The calling client can perform predefined operations using the Restful service.

RESTFul Principles and Constraints

RESTFul Client-Server
The server will have a RESTful web service which would provide the required functionality to the client. The client send's a request to the web service on the server.

Stateless
The server should not maintain any sort of information between requests from the client.

Cache
The cache is a concept implemented on the client to store requests which have already been sent to the server. So if the same request is given by the client, instead of going to the server, it would go to the cache and get the required information.

Layered System
The concept of a layered system is that any additional layer such as a *middleware layer* can be inserted between the client and the actual server hosting the RESTFul web service.

Interface/Uniform Contract
RESTful basically works on the HTTP web layer and uses the below key verbs to work with resources on the server
POST - GET - PUT - DELETE

SOAP Vs. REST

	Metric	SOAP	REST
direct / measurable	Cost (lower)		
	Effort (lower)		
	Server side		
	Client side		
	Lines of code (fewer)		
	Execution speed (faster)		
	Memory (low consumption)		
	Errors (less)		
indirect / non-measurable	Functionality		
	Protocol independent		
	Complexity (lower)		
	Efficiency		
	Performance (better)		
	Reliability		

SOAP Vs. REST

Evaluation metrics:

Cost

Effort

Lines of code

Execution Speed

Memory

Errors

Functionality

Complexity

Efficiency

Reliability

Metric	SOAP	REST
Cost (lower)		✓
Effort (lower)		
Server side		✓
Client side	✓	
Lines of code (fewer)	✓	
Execution speed (faster)		✓
Memory (low consumption)		✓
Errors (less)	✓	
Functionality		
Protocol independent	✓	
Complexity (lower)		✓
Efficiency		
Performance (better)		✓
Reliability	✓	

Selection of Soap and Rest protocols

Metric	SOAP	REST
Cost (lower)		✓
Effort (lower)		
Server side		✓
Client side	✓	
Lines of code (fewer)	✓	
Execution speed (faster)		✓
Memory (low consumption)		✓
Errors (less)	✓	
Functionality		
Protocol independent	✓	
Complexity (lower)		✓
Efficiency		
Performance (better)		✓
Reliability	✓	

	Main criteria	Project example
REST	Greater scalability; Compatibility; Performance; Simplicity; Point-to-point communication model; Limited bandwidth.	Mobile app integration with the information systems; Simple client-server data exchange solutions.
SOAP	Higher security and reliability; Lower number of errors; Asynchronous requests; Distributed computing; Support from other standards (WSDL, WS).	Business information systems (B2B); Banking information systems; Payment systems.

OpenAPI Specification (OAS)

OpenAPI Specification (formerly known as **Swagger Specification)**

API description format for REST APIs

Used to design new APIs and to document the existing APIs

Describe your entire API, including the available endpoints, operations, request and response formats,…

OpenAPI definitions can be written in *YAML* or *JSON*.

SwaggerHub Editor uses *YAML*, but you can import and download both YAML and JSON.

Swagger Tools consists of:

1. Swagger Editor
2. Swagger UI
3. Swagger Codegen

SwaggerHub combines these tools for a seamless experience, and adds hosted storage, access control and collaboration features on top of it.

Where does OAS fit in?

Requirements, Analysis and Design

Used by front and back-end developers to satisfy business and architectural requirements

Formalizes the output of that design process into code that can be used in subsequent development phases

Formalizes security concepts, such as publicly accessible or requiring authenticated access

Code and Test

With an OpenAPI document, tests can be generated from the specification

Developers and testers can iteratively work through testing and developing

Both client and server side can leverage the validation of data models through the use of the included JSON

Production and Maintenance

Verification

Testing

Documentation

Introduction to serverless architecture

Serverless architectures

Build and run applications and services without having to manage infrastructure
All the server management is done by AWS
No longer have to provision, scale, and maintain servers to run your applications, databases, and storage systems

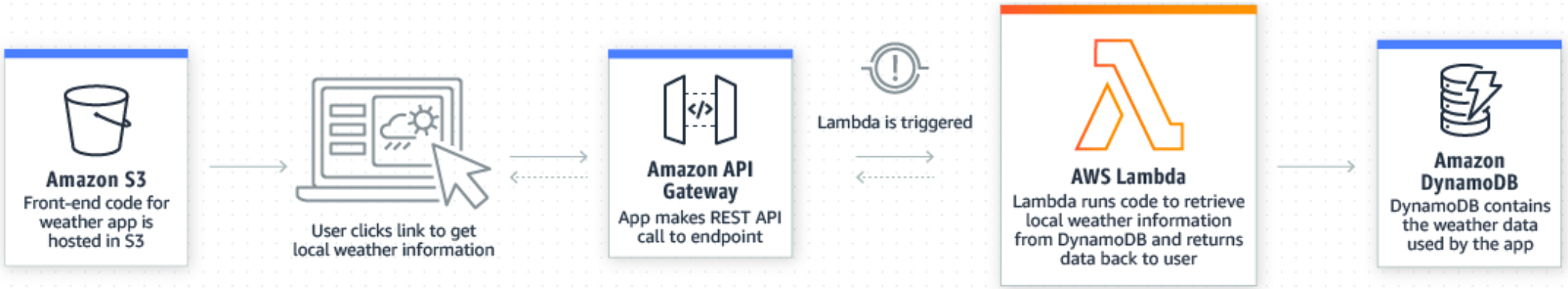
Why use serverless architectures?

Focus on their core product instead of worrying about managing and operating
Reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.

Serverless application use cases

Example: Weather application

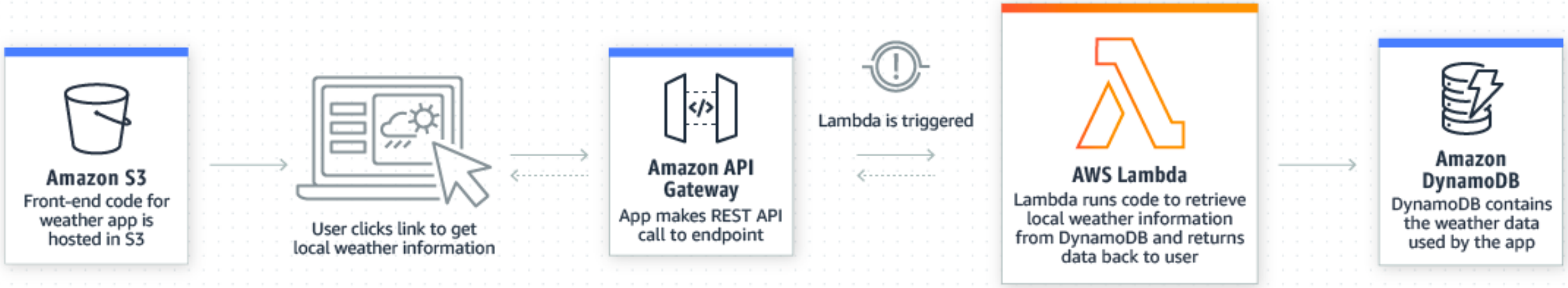
Introduction to serverless architecture



By using a serverless architecture, your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.

Serverless application use cases

Example: Weather application




```
                +---------+              +----------+              +----------+
 HTTP Request   |         |   Invoke     |          |   Query      |          |
------------>>> |   API   | ----->>>     |  Lambda  | ----->>>     | DynamoDB |
<<<------------ | Gateway | <<<-----    | Function | <<<-----    |  Table   |
 HTTP Response  |         |   Result     |          |   Result     |          |
                +---------+              +----------+              +----------+
```

Example: Weather application

Introduction to Amazon API Gateway

Amazon API Gateway

Handles accepting, processing, and orchestrating API requests

Maps requests to the appropriate backend

Swagger and the API Gateway

Two approaches to deploying your API on API Gateway — either import a Swagger file from your local machine, or building the API on the AWS console itself.

How SwaggerHub Bridges the Gap Between Swagger, the API Gateway and Lambda

The Swagger team conceptualized a method in which you can push API files from *SwaggerHub* to the AWS API Gateway, without any manual plumbing, and orchestrate all of the operations involved in deploying your APIs on API Gateway from one source of truth.

Inherent drawbacks of Serverless architectures

Vendor control

Multitenancy problems

Vendor lock-in

And many more..

Summary

1. Web Services
2. SOAP
3. REST
4. SOAP vs REST
5. Selection of Soap and Rest protocols
6. OpenAPI Specification (OAS)
7. Introduction to serverless architecture

Questions?

Thank You.