3-2-1: Examining Web Services in DevOps

After completing this episode, you should be able to:

- Identify and explain the significance of web services components in systems integrations, given a scenario.

Description: In this episode, the learner will examine various components in web services such as REST, SOAP, RPC, websockets and their significance in systems integrations. We will explore key features, use cases, and more.

- Describe the significance of REST in system integration
    - Representational State Transfer (REST)
        - ♠ An architectural style for designing networked applications.
        - ♠ It relies on stateless, client-server communication, using standard HTTP methods.
        - ♠ Key features
            - ♠ Stateless Interactions - each request from client to server must contain all the information needed to process the request.
            - ♠ CRUD operations
                - ♠ Create - adding new resources (HTTP POST).
                - ♠ Read - retrieving resources (HTTP GET).
                - ♠ Update - modifying existing resources (HTTP PUT or PATCH).
                - ♠ Delete - removing resources (HTTP DELETE).
            - ♠ Resource-Based URLs - resources are identified by URIs.
        - ♠ Use Case - ideal for web APIs due to simplicity and scalability.
- Describe the significance of SOAP
    - Simple Object Access Protocol (SOAP)
        - ♠ A protocol for exchanging structured information in web services using XML.
        - ♠ Key features
            - ♠ Extensibility - allows for the addition of new features and functionalities without disrupting existing services.
            - ♠ Neutrality - operates over any transport protocol, such as HTTP, SMTP, or TCP.
            - ♠ Independence - functions independently of any specific programming language or platform.
            - ♠ WS-Security support - provides a framework for secure messaging through encryption, authentication, and integrity mechanisms.
        - ♠ Use case - suitable for enterprise-level applications requiring security and transactional reliability.
- REST and SOAP
    - REST uses standard HTTP methods and JSON for communication, making it simpler and more efficient for web applications.
    - SOAP, uses XML for messaging, understanding both protocols is essential for effective cloud integration and API development.
    - Remote Procedure Call (RPC)
        - ♠ Allows a program to execute a procedure on a different address space as if it were local.
        - ♠ Key features
            - ♠ Synchronous communication - executes procedures on remote systems as if they were local, requiring both systems to be available during the call.
            - ♠ Tightly coupled systems - facilitates direct and efficient communication, provided both systems are compatible and available.
        - ♠ Use case - useful for services requiring direct method calls and lower overhead for internal communications.

- Web Sockets
  - Provides full-duplex communication channels over a single TCP connection.
  - Key features
    - Real-time - enables instant data transfer between client and server.
    - Bidirectional communication - allows for data to be sent and received simultaneously.
    - Low latency - minimizes delay in communication, ensuring quick response times.
  - Use case - ideal for live updates, chat applications, and real-time gaming.
- GraphQL
  - A query language for APIs that allows clients to request exactly the data they need.
  - Key features
    - Flexible and efficient data retrieval - clients can specify exactly the data they need, optimizing data transfer.
    - Strong type system - ensures data consistency and validation through defined types.
    - Schema-based - uses a schema to define the structure of the API and the relationships between data.
  - Use case - suitable for complex data-fetching scenarios, reduces over-fetching or under-fetching of data.

Additional References

- **Example 1: Create (POST)**
  - Objective: Add a new user to a database.

```
curl -X POST https://api.example.com/users \
-H "Content-Type: application/json" \
-d '{
  "name": "John Doe",
  "email": "john.doe@example.com"
}'
```

- curl: Command-line tool for making HTTP requests.
- -X POST: Specifies the HTTP method as POST, which is used to send data to the server to create a resource.
- \https://api.example.com/users: The URL of the API endpoint.
- -H "Content-Type: application/json": Sets the HTTP header to indicate the type of data being sent (JSON in this case).
- -d '': The data to be sent in JSON format.

- **Create (SOAP)**
  - Add new user to database

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:web='
  <soapenv:Header/>
  <soapenv:Body>
    <web:AddUser>
      <web:Name>John Doe</web:Name>
      <web:Email>john.doe@example.com</web:Email>
    </web:AddUser>
  </soapenv:Body>
</soapenv:Envelope>
```

- <soapenv:Envelope>: Root element of a SOAP message.
- xmlns:soapenv: Namespace declaration for SOAP.

- <soapenv:Header/>: Optional header element (empty in this example).
- <soapenv:Body>: Contains the body of the SOAP message.
- <web:AddUser>: Specific operation being called (add a user).
- <web:Name> and <web:Email>: Parameters for the operation.

- **GraphML**

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
                            http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
    <!-- GraphML keys for node and edge data -->
    <key id="d0" for="node" attr.name="color" attr.type="string"/>
    <key id="d1" for="edge" attr.name="weight" attr.type="double"/>

    <!-- Graph definition -->
    <graph id="G" edgedefault="undirected">
        <!-- Nodes -->
        <node id="n0">
            <data key="d0">red</data>
        </node>
        <node id="n1">
            <data key="d0">green</data>
        </node>
        <node id="n2">
            <data key="d0">blue</data>
        </node>

        <!-- Edges -->
        <edge id="e0" source="n0" target="n1">
            <data key="d1">1.0</data>
        </edge>
        <edge id="e1" source="n1" target="n2">
            <data key="d1">2.0</data>
        </edge>
        <edge id="e2" source="n2" target="n0">
            <data key="d1">3.0</data>
        </edge>
    </graph>
</graphml>
```