HANDS-ON ACTIVITIES

INTEGRATION ARCHITECTURE

FEBRUARY, 2018

1. FIRE AND FORGET

Leverage the features and capabilities of the platform appropriately within the solution.

USE CASE

A company, Universal Containers, uses several systems as part of their enterprise system landscape, including Salesforce.com and an ERP system. Salesforce.com is the master system for CRM data, such as Accounts and Opportunities. The ERP system is the master system for customer orders and shipping and billing information.

DETAILED REQUIREMENTS

As part of their business process flow, once a sales opportunity is successfully closed ("Closed Won") an order needs to be created with the details from the opportunity for shipping and subsequent invoicing. The opportunity is maintained in Salesforce.com, but the order needs to be created in the on premise ERP system, as it is the order master. Once the order is created in the ERP system, the opportunity status must be changed to indicate that the order was created.

ASSUMPTIONS

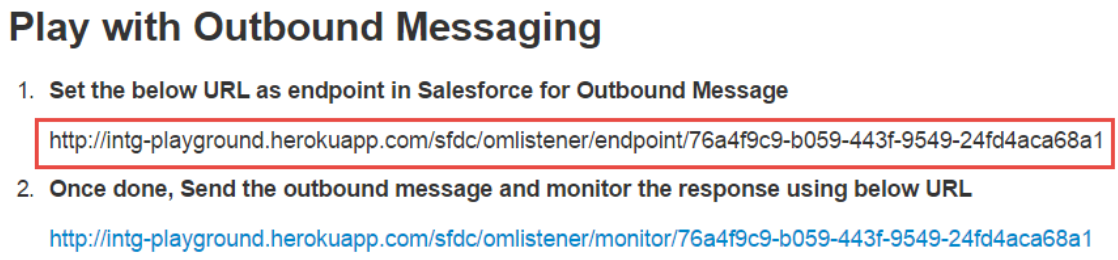
- Universal Containers has middleware that integrates with the on premise ERP system.
- This middleware can host a SOAP endpoint using a WSDL from Salesforce.
- The user doesn't need to be immediately notified of the order number after the opportunity converts to an order.
- The solution must only use declarative features of Salesforce.com.

PREREQUISITE SETUP STEPS

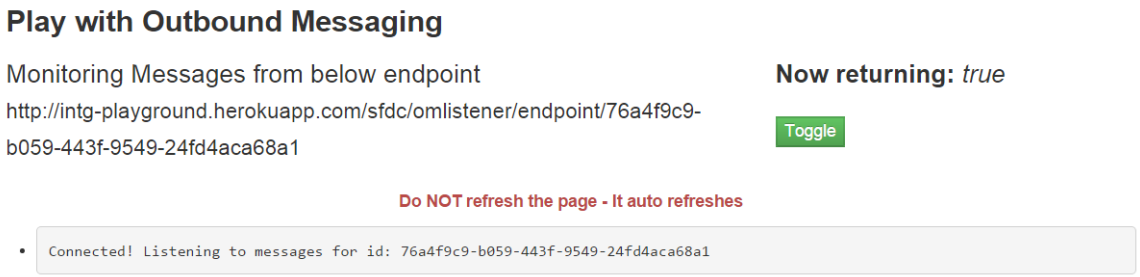
Get a proxy endpoint for a proxy service (Outbound Message Listener):

1. Go to the Integration Playground for Outbound Messaging here: http://intg-playground.herokuapp.com/sfdc/omlistener

2. Copy the unique endpoint that is randomly generated. This helps to uniquely monitor the messages coming in from an outside system.



3. Open the monitor URL in a new tab. This helps to monitor all the messages coming to this endpoint and provides an ability to play around with the responses being sent.



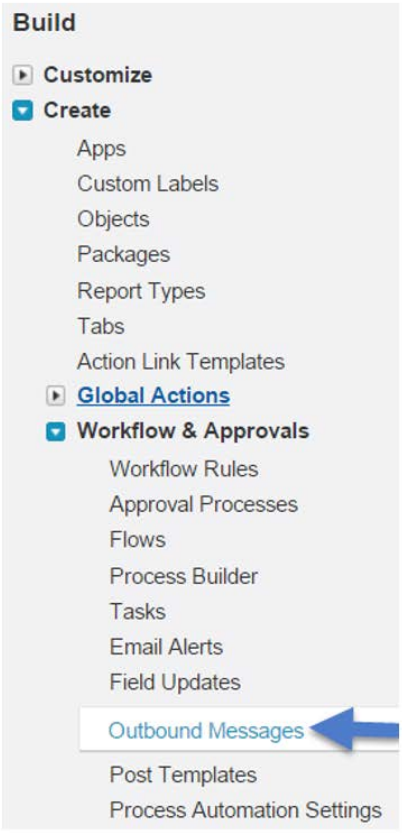
CONSIDERATIONS

- What are the patterns that can be used?
- What is the integration frequency: real-time vs. batch?
- What are the Salesforce.com platform limits?
- How are errors handled both in Salesforce.com and on the third-party platform?
- Can you send information from multiple related objects?
- Which web-services protocols can be used?
- How is security handled (2-way SSL, authentication)?

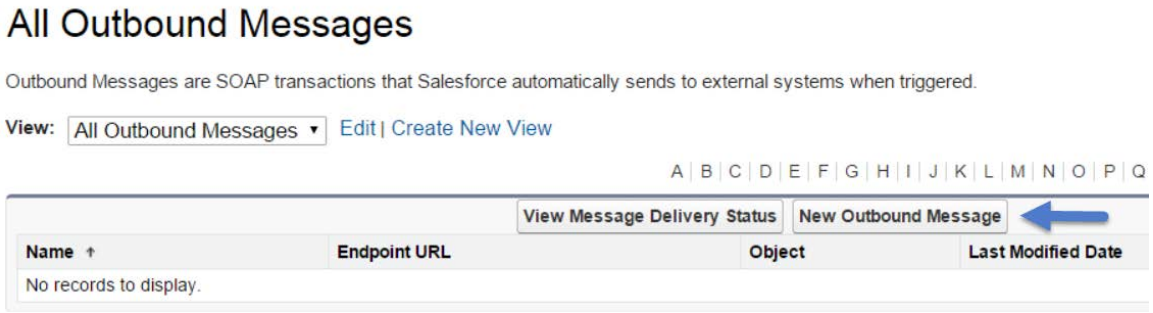
SOLUTION

Log in to the Salesforce org with your credentials.

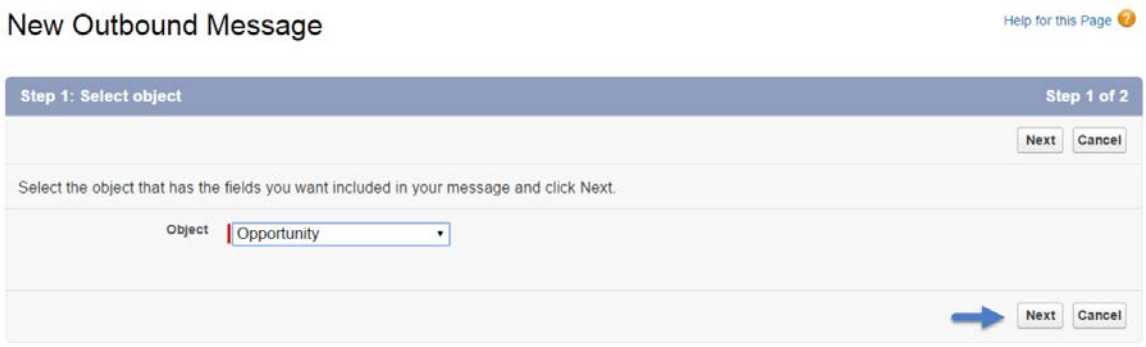
1. **Create a new outbound message.**
   - From Setup, click **Create | Workflow & Approvals | Outbound Messages**.



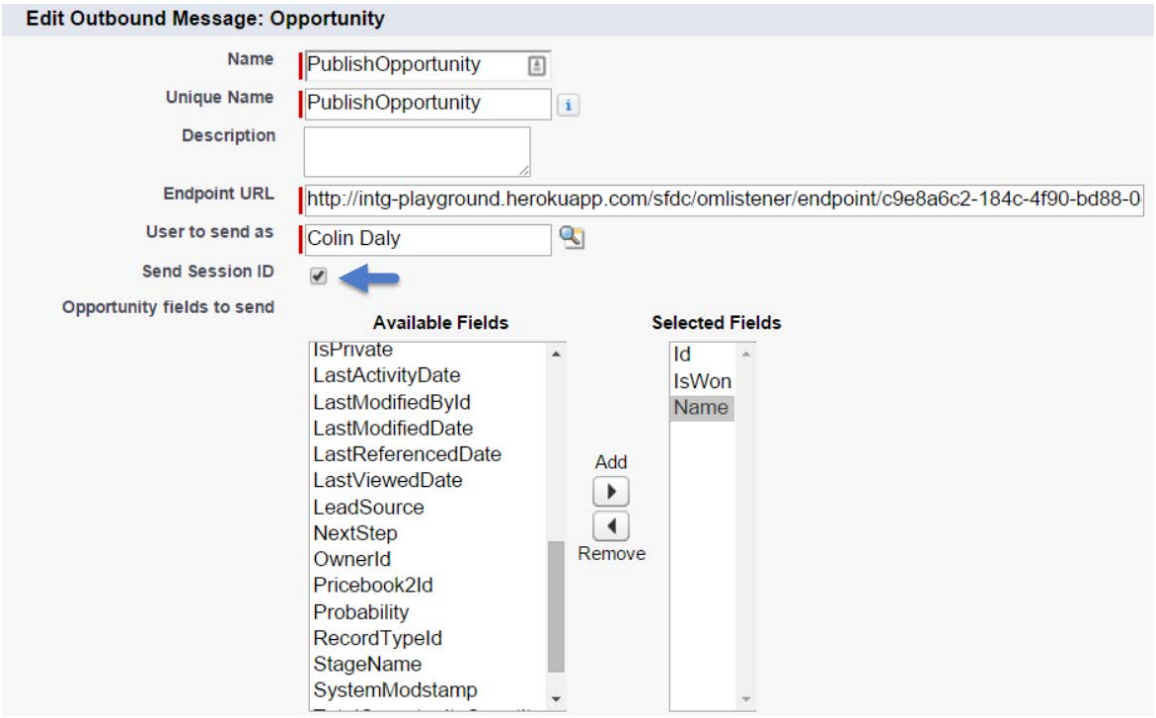
   - Click **New Outbound Message**.



- Choose the object that has the information you want included in the outbound message, in this case **Opportunity**, and click **Next**.

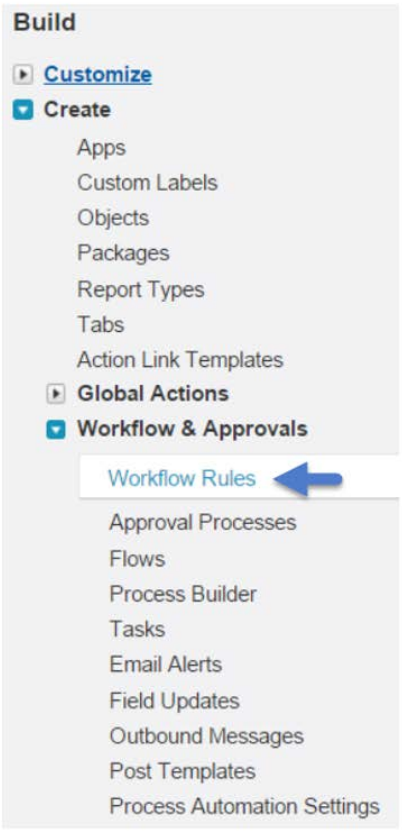


- Enter a name and description for this outbound message.
- Paste the endpoint URL that was copied in prerequisite step.
- Check **Send Session ID**.



- Choose fields to be sent in the outbound message and click **Save**.

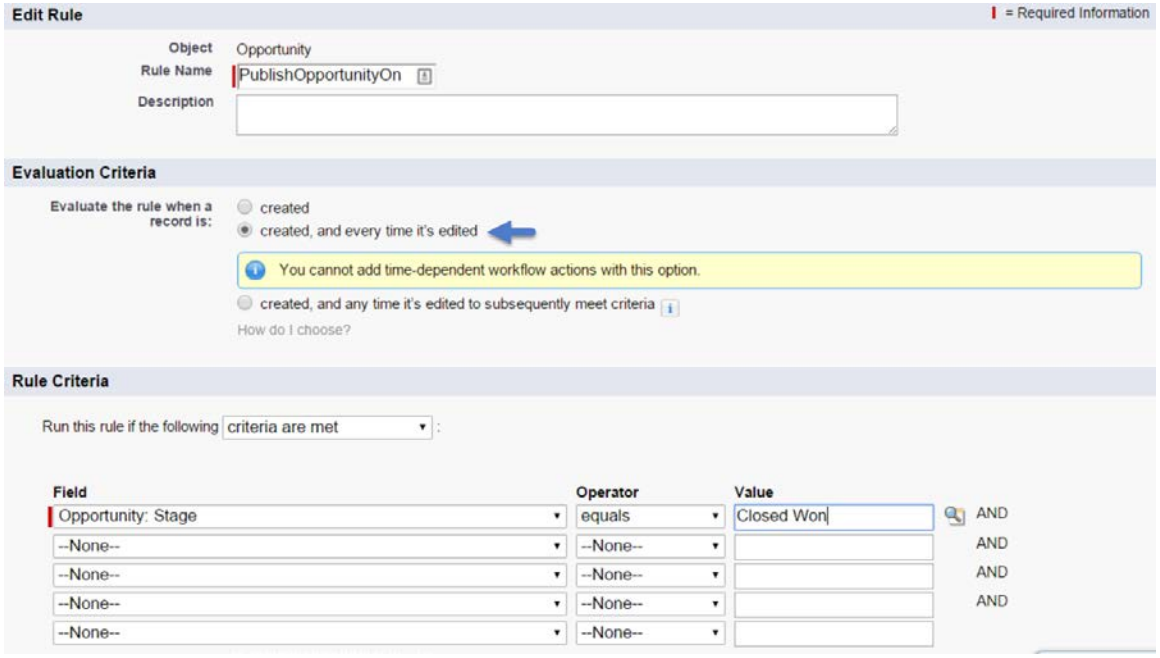
2. **Set up a Workflow rule.**
   - From Setup, click **Create | Workflow & Approvals | Workflow Rules**.



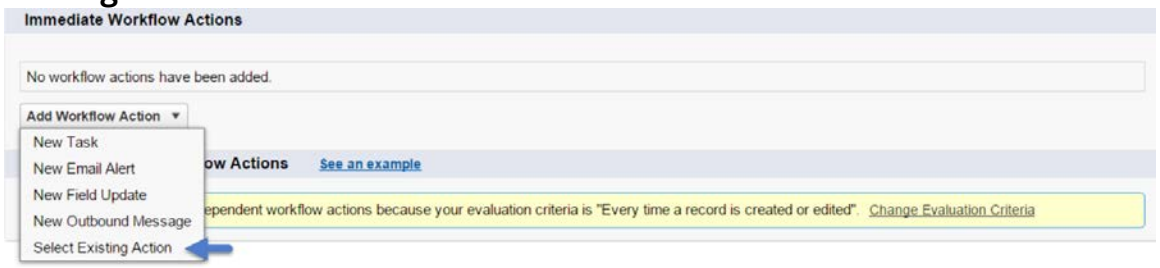
   - Click **New Rule**.

   - Choose the object that has the information you want included in the outbound message, in this case **Opportunity**, and click **Next**.

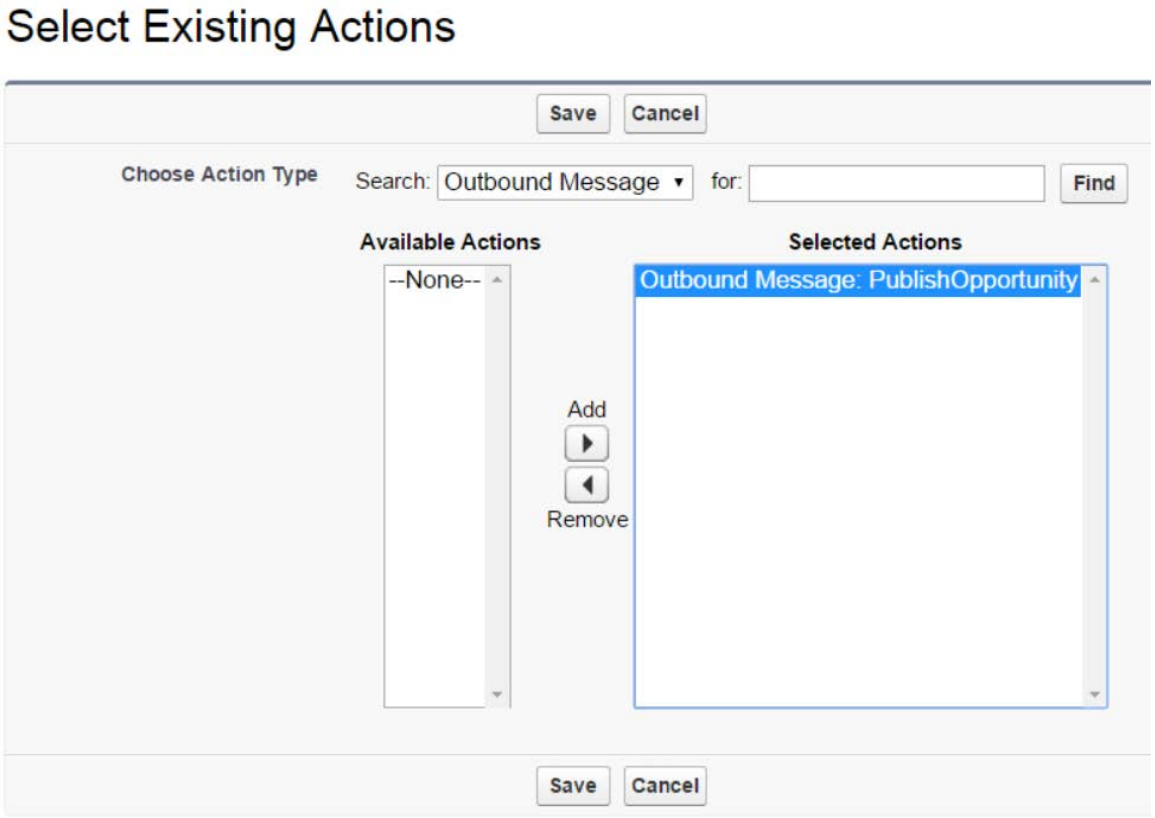
- Enter the Rule Name, choose the Evaluation Criteria **created, and whenever it's edited**, and set the Rule Criteria: in this case, **Opportunity: Stage equals Closed-Won**.



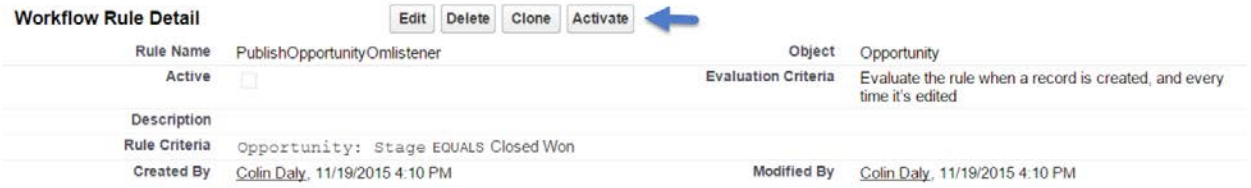
- Click **Save & Next**.
- Click **Add Workflow Action | Select Existing Action | Select Outbound Message**.



- Choose the **Outbound Message** action created in step 2. Click **Save**.

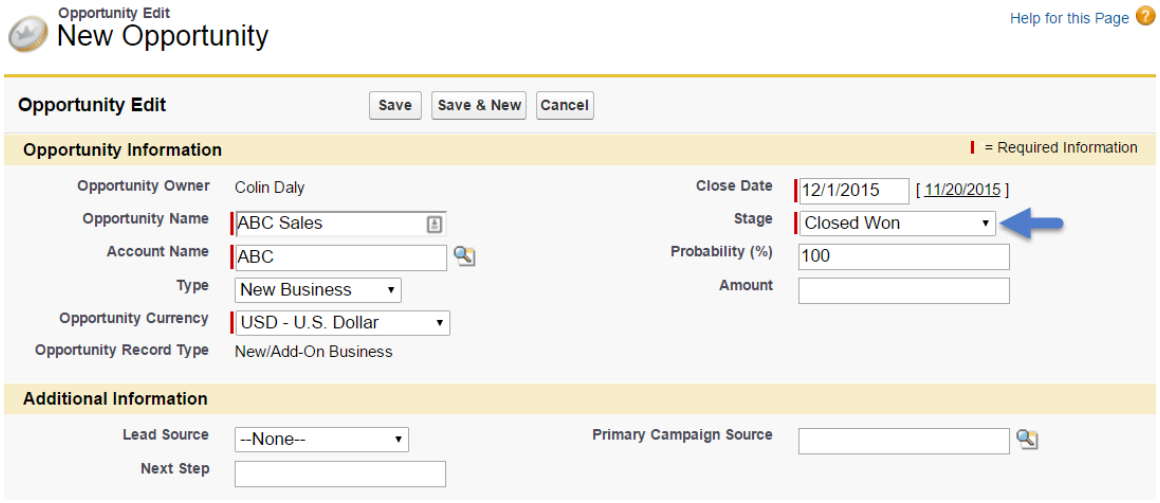


- Click **Done** and then click **Activate**.

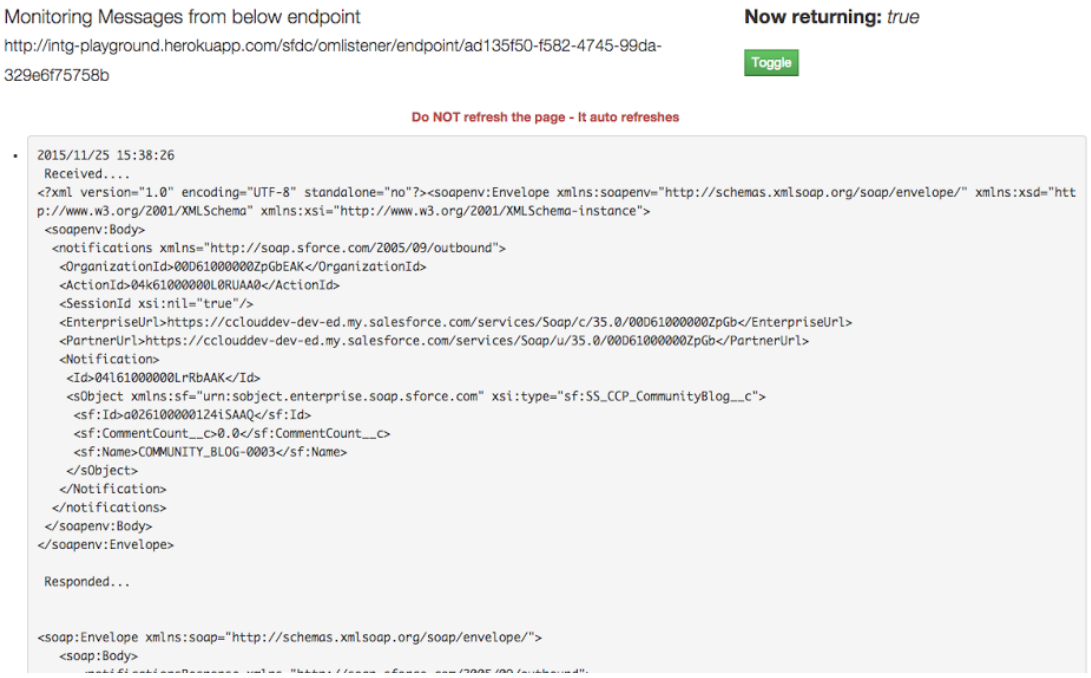


3. **Play with outbound messaging.**
   - Now that the setup is complete, it's time to test and play with outbound messaging.
   - Go to the monitor URL from step 1. Make sure the page reads "Now returning: true." That means the endpoint acknowledges outbound messages with success. By clicking the **Toggle** button, it can be switched to return false, which means that the endpoint acknowledges outbound messages with failure.

- Go to Salesforce and create a new opportunity with the stage **Closed Won**.



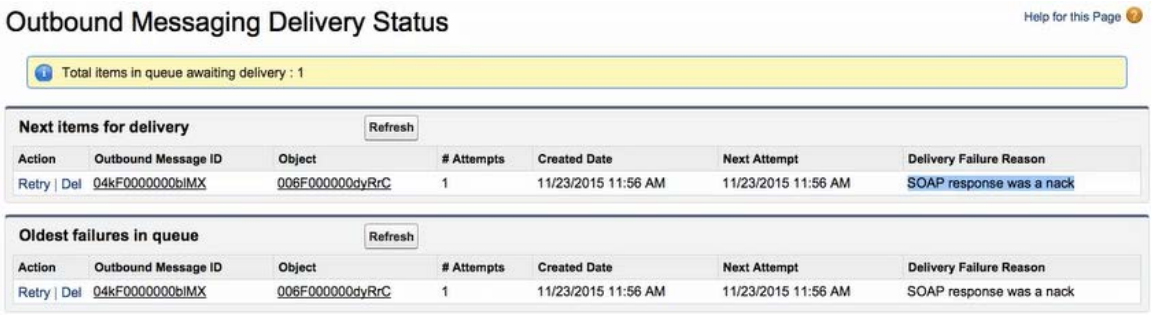
- Switch to the monitor URL and you should see messages being received by endpoint.



ALERT:

Alert: Please have only one browser listener window open at any point of time.

4. **Play with error scenarios.**
    - Go to the Monitor URL and click the **Toggle** button to make the endpoint return false. This will throw an error back to the Salesforce outbound message, which can be monitored and retried from Salesforce.
    - After toggling the endpoint to return false, go to Salesforce and create a new opportunity with the stage as **Closed Won**.
    - Go to **Setup | Monitor | Outbound Messages**. It should display errors as shown in the screenshot below. You can also see it keeps retrying.



- Go back to the Monitor URL and toggle the response back to true.
- Now through automated retry, the messages are delivered again. It can also be manually retried by clicking the **Retry** button.

2. REQUEST AND REPLY

Provide details on the integration components involved in a flow, describe Salesforce integration patterns, and discuss considerations pertinent to transaction management and error and exception handling.

USE CASE

Universal Containers uses Salesforce, but has a separate system that contains customer order, billing, and shipping information. Salesforce.com is the master data source for Accounts and Opportunities.

Universal Containers wants to display the order status and billing history for a customer account without having to store that data in Salesforce because of NPI data security concerns. They have an existing REST/Web Service that can return order status and shipping information given an account number, but cannot otherwise display this data in a browser.

PREREQUISITE SETUP STEPS

Set up remote site setting: http://intg-playground.herokuapp.com/.

CONSIDERATIONS

- Does the call to the remote system require Salesforce to wait for a response before continuing processing? In other words, is the call to the remote system a synchronous request-reply or an asynchronous request?
- If the call to the remote system is synchronous, does the response need to be processed by Salesforce as part of the same transaction as the initial call?
- Is the message size relatively small or large?
- Is the integration based on the occurrence of a specific event, such as a button click in the Salesforce user interface, or DML-based events?

SOLUTION

In the Request-Reply pattern, Salesforce makes a call to the remote system to either perform an operation or fetch data. It then waits for successful completion of that call. To signify successful completion, the remote system synchronously replies with the requested data or a success indicator code. The remote system exposes a web service (REST or SOAP) that allows other systems, such as Salesforce, to integrate with it.

The high-level solution steps are as follows:

- Salesforce consumes the Order status and billing/shipping service from Universal Containers.
- Use Order external Id in Salesforce to fetch order status and billing information (NPI security data) from the external system.
- Create a Visualforce page and extension controller to call REST/ Web service with required parameters.
- Create the custom controller that parses the returned values from the REST callout or Web Service.
- The Visualforce page subsequently renders the order and billing details to the user.

Best Solution Overview

The steps necessary to create the integration pattern are explained in detail in the following sections.

1. **Create an Object.**

   Create an Object (iOrder) with following field and API names:

Field Name	API Name	Field Type
Order Name	Name	Text(80)
Invoice Amount	Invoice_Amount__c	Currency(18, 0)
Invoice Number	Invoice_Number__c	Text(20)
Opportunity	Opportunity__c	Master-Detail(Opportunity)
Order External ID	Order_External_ID__c	Text(20) (External ID)
Order Status	Order_Status__c	Text(20)
Product Name	Product_Name__c	Text(30)
Shipping Number	Shipping_Number__c	Text(30)

Note the use of field "External ID" to store reference to the OrderID in the external system. Furthermore, you can create your own object and appropriate relationships. If you have your own object model, use and carry the appropriate API names in Apex and Visualforce pages.

2. **Create Apex classes to invoke REST/ Web Service.**

   Create an Apex class – OrderShipInfo with getOrderShipInfo method to invoke the external REST/ Web Service.

   The getOrderShipInfo method invokes the REST/Web Service by appending Order ID to the End Point URL. It then de-serializes the JSON string received in the HTTP response to populate the ShipInfo object, which is then used by the VisualForce page to display the order status.

```
1  public with sharing class OrderShipInfo {
2    private final String EXTERNAL_END_POINT_URL = 'http://intg-
   playground.herokuapp.com/services/rest/iOrder';
3     /*
4      *  Ship Info Class to hold REST API response data
5     */
6    public class ShipInfo
7    {
8      public String id{get;set;}
9      public String status{get;set;}
10     public String comments{get;set;}
11     public String orderOwner{get;set;}
12   }
13  public ShipInfo getOrderShipInfo(String externalID){
14   try{
15      HttpRequest req = new HttpRequest();
16      req.setEndpoint(EXTERNAL_END_POINT_URL + '/' + externalID);
17      req.setMethod('GET');
18      req.setHeader('Content-Type', 'application/x-www-form-urlencoded');
19      req.setTimeout(60000);
20
21      Http http = new Http();
22      HttpResponse res = http.send(req);
23      ShipInfo theStatus =
   (ShipInfo)JSON.deserialize(res.getbody(),ShipInfo.class);
24      return theStatus;
25   }catch(CalIOutException ce){
26      throw ce;
27   }
28   return null;
29  }
30 }
```

3. **Create an Apex Controller for use in the Visualforce page.**

```
1  public class iOrderControllerExtension
2  {
3  private iOrder__c myOrder;
4  OrderShipInfo myOrderShipInfo;
5  OrderShipInfo.ShipInfo theShipInfo;
6  private ApexPages.StandardController controller {get; set;}

7  public iOrderControllerExtension(ApexPages.StandardController
   controller) {
8  //initialize the stanrdard controller
9  this.controller = controller;
10 this.myOrder = (iOrder__c)controller.getRecord();
11 }
12 public void refreshOrderShipInfo()
13 {
14 myOrderShipInfo = new OrderShipInfo();
15 theShipInfo =
   myOrderShipInfo.getOrderShipInfo(this.myOrder.Order_External_ID__c);

16 myOrder.Order_Status__c = theShipInfo.status;
17 update myOrder;
18 }
19 public String getOrderStatus()
20 {
21 return theShipInfo.status;
22 }
23 public String getOrderComments()
24 {
25 return theShipInfo.comments;
26 }
27 public String getOrderOwner()
28 {
29 return theShipInfo.orderOwner;
30 }
31 }
```

4. **Create a custom Visualforce page.**

   Create a Visualforce page; for example “OrderPage” with sample code:

```
<apex:page standardController="iOrder__c"
extensions="iOrderControllerExtension">
<apex:pageBlock >

<apex:pageBlockSection >
<apex:pageBlockSectionItem >
<apex:outputLabel value="Order Status" for="order_status"/>
<apex:outputText value="{!iOrder__c.Order_Status__c}" id="order_status"/>
</apex:pageBlockSectionItem>

<apex:pageBlockSectionItem >
<apex:outputLabel value="External System Order ID" for="external_id"/>
<apex:outputText value="{!iOrder__c.Order_External_ID__c}"
id="external_id"/>
</apex:pageBlockSectionItem>

<apex:pageBlockSectionItem >
<apex:outputLabel value="Shipping Number" for="shipping_number"/>
<apex:outputText value="{!iOrder__c.Shipping_Number__c}"
id="shipping_number"/>
</apex:pageBlockSectionItem>
</apex:pageBlockSection>

<apex:pageBlockButtons location="bottom">
<apex:form >
<apex:commandButton action="{!refreshOrderShipInfo}"  value="Refresh Order
Status" id="theButton"/>
</apex:form>

</apex:pageBlockButtons>
</apex:pageBlock>


<apex:outputText value="Note: Clicking the Refresh Order Status button makes
a REST API callout to fetch shipping information from an external application" />
</apex:page>
```

5. **Test the Visualforce page.**

   Place the Visualforce page on the page layout of the iOrder__c object. Upon clicking **RefreshOrderPage**, the JSON response from the REST service should be displayed. Make sure that you are able to invoke the REST/ Web Service and get the response back. Click the URL to ensure that the REST service is active and JSON response is sent: http://intg-playground.herokuapp.com/services/rest/iOrder/12345.

   This pattern is described in detail in the Salesforce Integration Patterns and Practices guide.

3. REMOTE CALL-IN

Ability to leverage the features and capabilities of the platform appropriately using remote systems.

USE CASE

A company, Universal Containers, uses several systems as part of their enterprise system landscape including Salesforce.com and an ERP system. Salesforce.com is the master system for CRM data such as Accounts and Opportunities. The ERP system is the master system for customer orders, shipping and billing information.

DETAILED REQUIREMENTS

As part of their business process flow, when an order in created in the ERP system, it needs to be also created in Salesforce.com in real-time.

PREREQUISITE SETUP STEPS

1. Create an integration user in Salesforce. It is best practice to create a dedicated integration user for integration needs, with the profile permission of "API only" and a non-expiring password.
2. If you are not using a dedicated integration user, then the user's profile should have the "API Enabled" permission at a minimum.

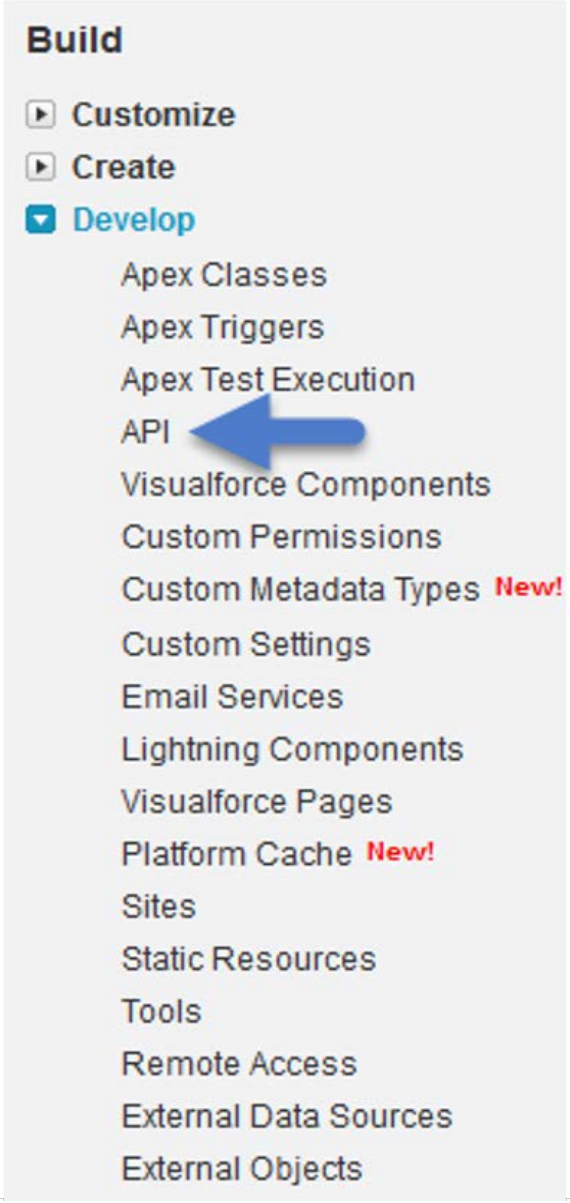
CONSIDERATIONS

What are the patterns that can be used?
Does this need to be real-time or batch?
What are the platform limits?

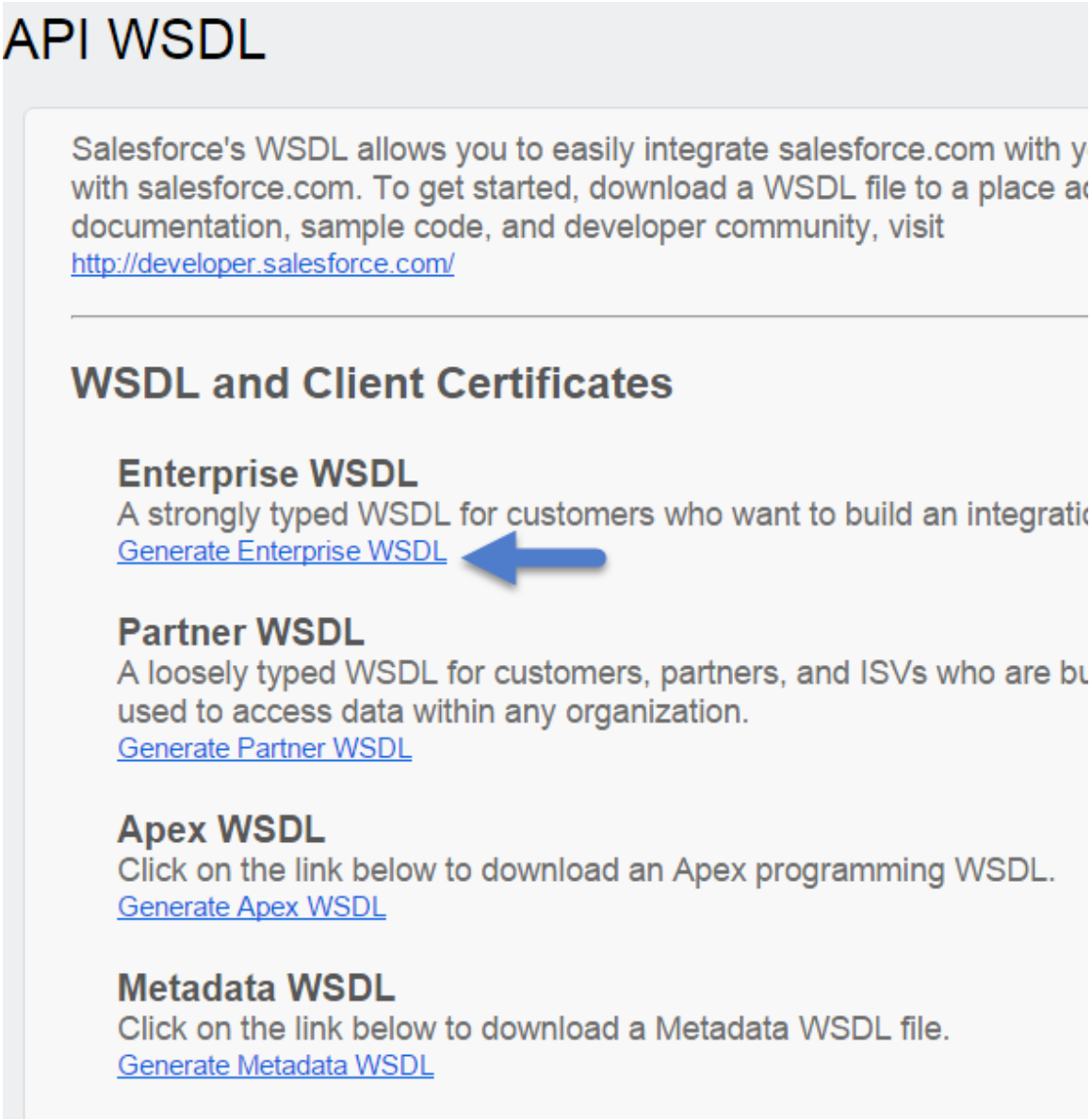
SOLUTION

1. Generate WSDL.

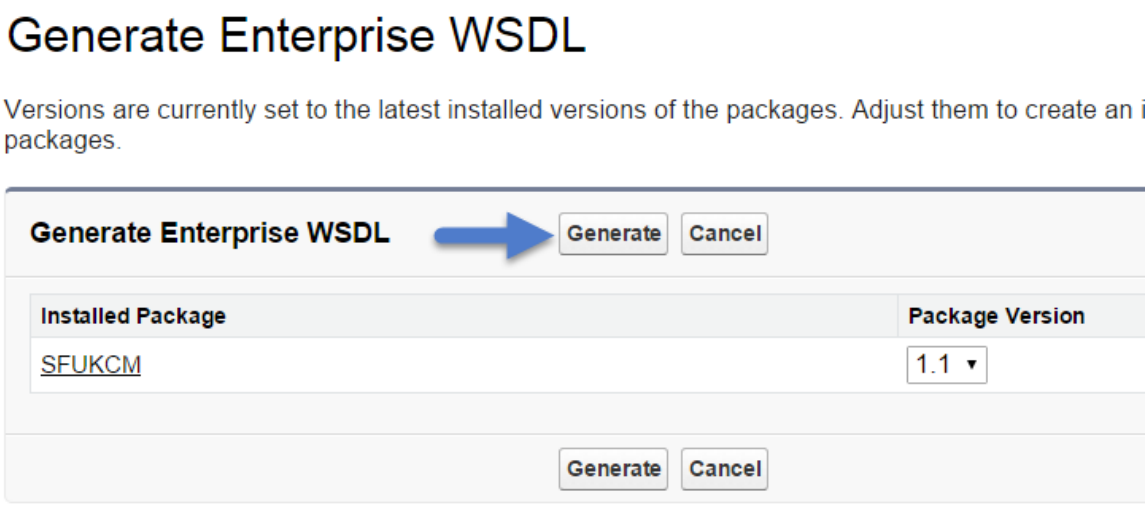
- Log in to the Salesforce Org with your credentials.
- User should have the "Modify All Data" permission to generate the WSDL.
- From Setup, click **Develop** and then **API** to display the WSDL download page.



- Download the appropriate WSDL; in this case, it's Enterprise WSDL. Click the **Generate Enterprise WSDL** link.



- If you are downloading an enterprise WSDL and you have managed packages installed in your org, click **Generate**.



- You will then be directed to a webpage displaying the document tree of an XML file. Right-click on the webpage and choose **Save as...** if you are using Chrome or **Save Page As...** if you are using Firefox. Save it to a local directory.

2. Import the WSDL into Heroku (or SOAP UI).

- Go to http://intg-playground.herokuapp.com/sfdc/inbound
- Click the link under 1. Play with Remote call in - Enterprise WSDL.
- Click **Upload Enterprise WSDL**.
- Choose the WSDL file you downloaded.
- Upon successful upload, you should see an endpoint in the right-hand side of the page.

> Enpoint:
> https://login.salesforce.com/services/Soap/c/35.0/0DFU0000000fxWX

3. **Log in to Salesforce.**

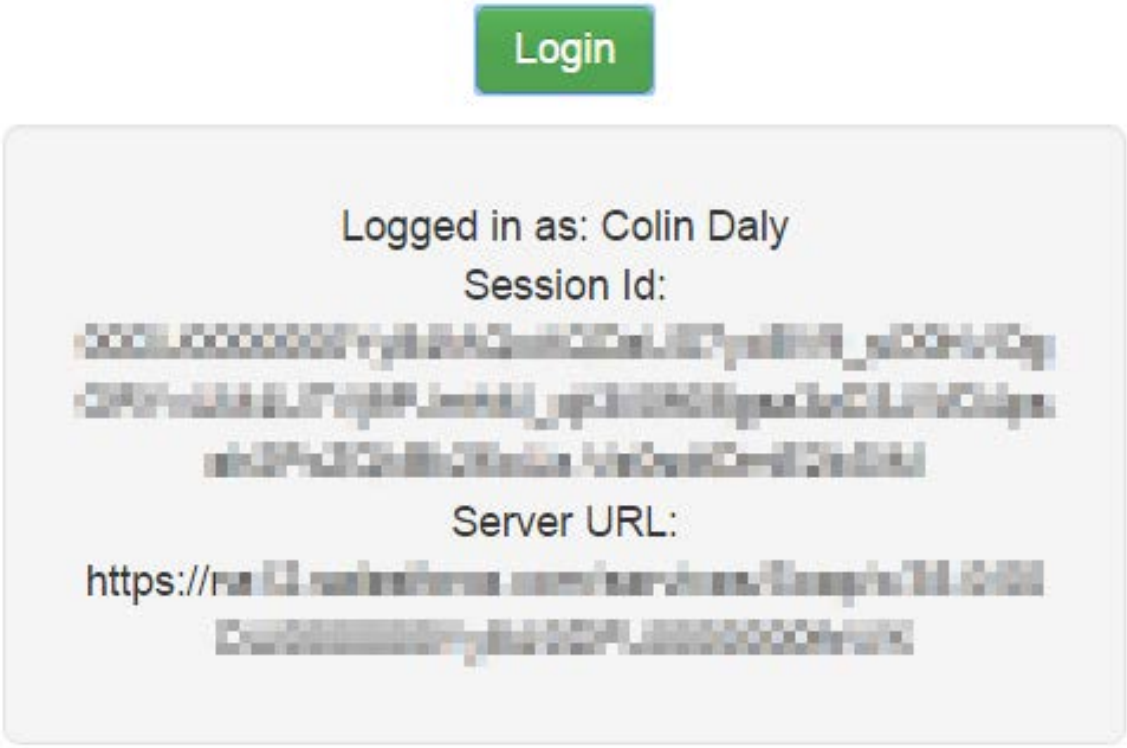
- Before doing any operation through SOAP API, a login step is necessary. This step gets the Session ID and Server URL from Salesforce, which must be passed back to Salesforce on further operations in SOAP API.
- Replace the variables in the XML data in the text area under that webpage's STEP #2 with your username and password.

STEP #2: Login to Salesforce

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:enterprise.soap.sforce.com">
  <soapenv:Body>
    <urn:login>
      <urn:username>REPLACE_WITH_USERNAME</urn:username>
      <urn:password>REPLACE_WITH_PASSWORD</urn:password>
    </urn:login>
  </soapenv:Body>
</soapenv:Envelope>
```

- Click the **Login** button.

- Upon successful login, it should display your full name as defined in Salesforce, the Session ID, and Server URL.



4. Execute SOAP Operation or, in this case, create iOrder.

- Copy the Session ID created in the previous step.

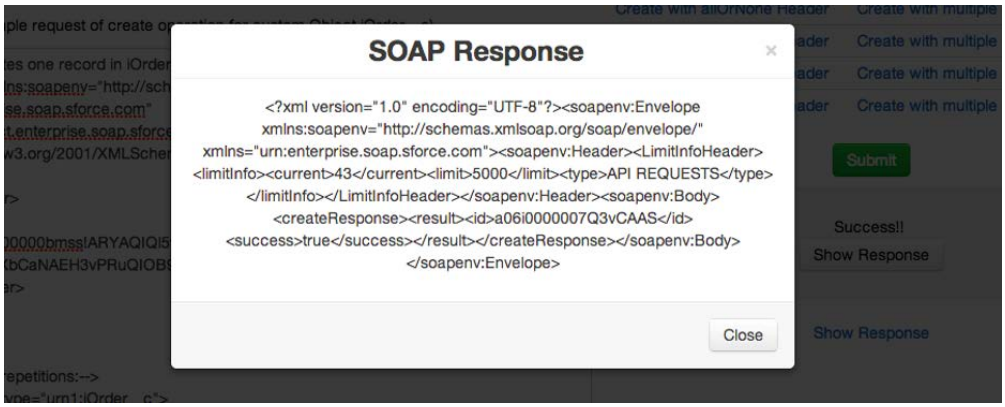
- Paste the Session ID into the XML data window so that it replaces the "REPLACE_WITH_SESSION_ID OBTAINED IN STEP 2" text.

STEP #2: Play with different operations in Enterprise WSDL

(Example below is a sample request of create operation for custom Object iOrder__c)

```
<!-- This message creates one record in iOrder custom object -->
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:enterprise.soap.sforce.com"
xmlns:urn1="urn:sobject.enterprise.soap.sforce.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <urn:SessionHeader>
      <urn:sessionId>REPLACE_WITH_SESSION_ID OBTAINED IN STEP
2</urn:sessionId>
    </urn:SessionHeader>
  </soapenv:Header>
  <soapenv:Body>
    <urn:create>
      <!--Zero or more repetitions:-->
```

- Click the **Submit** button.
- Upon successful post, the page will display a Success message.
- By clicking **Show Response**, the app shows the raw SOAP response returned by Salesforce.



For more details on the pattern, Integration Patterns and Practices.

4. DATA INTEGRATION SPECIALIST

Demonstrate your integration skills by synchronizing external data systems and Salesforce.

TRAILHEAD SUPERBADGE

Data Integration Specialist Superbadge

ADDITIONAL ACTIVITIES

To practice these activities, you can leverage one of your Playgrounds or other dev environments if needed.

1. WORKFLOW

Build a Workflow to trigger an Outbound Message on update to an Account record.

- Send the outbound message to a simple HTTP-capture service (e.g., http://requestb.in/).
- Examine the delivered messages.
- Observe the automated retry.
- Check the Outbound Message queue and cancel the failed message.

2. SOAP INTEGRATION

Using a simple SOAP integration tool (e.g., SOAPUI, Eclipse), import the SOAP WSDL and perform each of the core operations (login, query, upsert, etc.) to understand the message structure.

3. REST INTEGRATION

Using a simple REST integration tool (e.g., cURL, Postman), authenticate and perform some basic GET and POST requests to the REST API to understand the message structure.