Lab 2

Connect the Order API to a SOAP Web Service
Overview

In this step we are going to implement the post:/orders flow. To create an order, we are going to need to

1. Call the .NET SOAP Service to create the Order.
2. Transform the request to XML and response back to JSON format.

Step 1: Implement Create Order flow

| WARNING | Make sure you are modifying the post:/orders flow! Name should look like post:/orders:application/json:api-config or similar. |

1. Remove the Transform Message component icon. To do that you just right-click on processor icon and select Delete.

2. Let’s add the Web Service Consume processor to the flow. To do that go to the Mule Palette panel and type web service in the filter you should see an icon like in the picture below.
If you don’t see this processor in your palette, follow these instructions to add it to your palette.

*Click* on the **Add Module** link in the palette.

An "Add Modules to Project" window will pop up. In this window, *type* **web service** in the search window to narrow down the module you wish to add.  
*Select* the **Web Service** Consumer Module and *drag & drop* the icon.

This will add the Module to your palette, allowing you to select and use the Web Service Consumer processor in your project.

1. *Drag and drop* the **Consume** processor into the empty flow.

2. *Double-Click* on the **Consume** processor icon to display its configuration panel.

3. Change its name to **Create Order**
4. Let's add a new connector configuration. Click on button on the **Connector Configuration**. Then this window below will pop up.

5. In the **WSDL Location** field, paste the following WSDL definition, that contains the definitions
of our .NET SOAP Service which is already deployed: http://soap-order-ws-v2.use1.cloudbhub.io/service/BasicHttpBinding_IOrderService?wsdl

Proceed and complete the rest of the configuration for **Service** and **Port** and you can leave **Address** empty.

6. **Click** the **Ok** button

7. Once the configuration was set, **Select Create Order** from the list of operations back in the basic settings section.
8. Click the button on the top right of the Properties Panel to save the changes.

Step 2: Transform the request to XML and response back to json format

Now our configuration to create the order is complete, but there is one thing missing, we need to format the message coming in and out of the Create Order icon. To do this, we need to add 2 Transform Message processors (DataWeave Elements), one before and one after the Web Service Consumer.

1. Type transform in the Mule Palette panel

2. Drag-&-drop a Transform Message before and after Create Order Web Service

3. Double-Click the first Transform Message element to open the properties.

4. Notice that we already have the input and output Metadata defined by Datasense so we know what is coming in and out of these elements
5. The idea is to map each of the incoming (input side) items with the ones that our .NET SOAP Service is expecting. For example, match the customerId from the left, to the UserId on the right:
6. After matching all the corresponding items, your mapping should look like below.

**NOTE**

For the **Status** output field, we will hard code the value instead of mapping it from the input side. To do this, double click on the **Status** field and it will show up in the Dataweave screen. From inside the Dataweave screen change default value of null to a constant value of 'new'.

The **dataweave** code should look like this: Remember not to match the “ID” fields on the right
(the ones that are “id” alone), these are auto-generated for each order created. E.g: `ns1:Id : Number?

```dw 2.0
output application/xml
ns ns0 http://tempuri.org/
var totalPrice = (if(payload.totalPrice == null) sum(payload.orderLineItems.price) else payload.totalPrice as Number)
---
{
  ns0#CreateOrder: {
    ns0#Order: {
      ns0#Items: {
        (payload.orderLineItems map ( orderLineItem , indexOfOrderLineItem
          ) -> {
          ns0#OrderItem: {
            ns0#ProductId: orderLineItem.productId,
            ns0#Quantity: orderLineItem.quantityOnHand
          }
        })
      },
      ns0>TotalPrice: totalPrice,
      ns0>UserId: payload.customerId,
      ns0>Status: "new"
    }
  }
}
```

**NOTE**

Pay attention at `var totalPrice = (if(payload.totalPrice == null) sum(payload.orderLineItems.price) else payload.totalPrice as Number)`. Here if the totalPrice doesn't come in the request, it will be calculated based on the price of each item.

7. You can copy and paste the code above on the right panel containing the DataWeave code to save time.

8. After doing this, Click the button.

9. We will now move onto the Transform Message element on the right of the Web Service Consumer processor, the one that formats the outgoing data from the web service call. Double-Click on the Transform Message element on the right hand side of the Web Service Consumer that we named “Create Order”, to bring up the properties.

10. You can see that this one is quite simple, it just returns(output) the id of the order just created.
11. Match the `CreateOrderResult` on the left, with the `id` on the right by dragging the `CreateOrderResult` field from the input side over to `id` field on the output side.

12. After this, just click outside in the white canvas to save the changes, or click the button at the top right of the element panel.

### Summary

In this lab, you completed the following steps:

- **Step 1:** Implement Create Order flow
- **[Step 2: Transform the request to XML and response back to JSON format]**

Earlier, we used the RAML specification to define our API and auto-generated a skeleton project to implement the API. In this lab we took a specific flow (POST) and implemented a real integration with an external service to Create Order. In doing so we added a Web Service Consumer component, configured it to a WSDL using a valid url. Additionally we added a Transform component before and after the Web Service Component in order to provide the proper payload needed for the web service call, and to properly handle the response coming back from the web service call. Finally we are now able to run this project, in the next lab, locally on Anypoint Studio to test out the new implementation.

**Congratulations! You have completed Lab 2.**

Please proceed to Lab 3

Take me to the TOP