Consume SOAP web services

For the app you want to consume a SOAP web service, complete the steps under the following sections:

1. Setup your Visual Studio Code project
2. Prepare your SOAP Request to be consumed by a REST
3. Configure AWS lambda to consume the SOAP web service
4. Create the REST in ODC Studio to consume the Lambda function
5. Setup the AWS Signature configurations in ODC Portal

To consume a SOAP web service, follow these steps:

Setup your Visual Studio Code project

1. If you don't have .NET 6.0 SDK on your laptop, you need to install it. Click here.

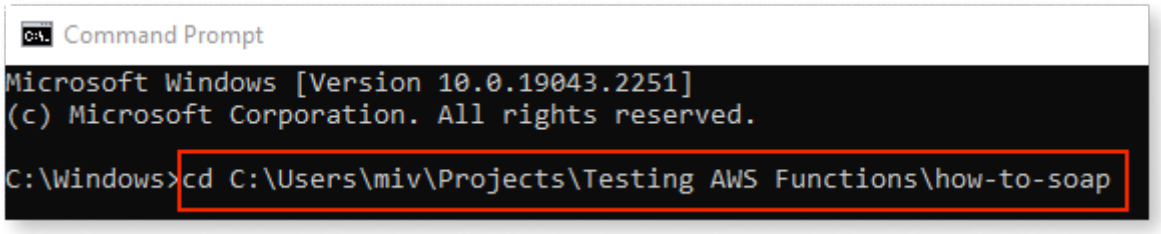
2. If you don't have Visual Studio Code installed on your laptop. Click here.

3. Inside visual studio code, go to the extensions marketplace, search for and install the following extensions:

    1. `C#` (extension id: ms-dotnettools.csharp)
    2. `AWS ToolKit` (extension id: amazonwebservices.aws-toolkit-vscode)

4. In your laptop, select the path of the folder you want to create your SOAP integration.

5. In the command line type `cd <YOUR PATH FOLDER>` and then you will follow a few steps that are available here. In the example bellow a folder was created with the name how-to-soap and it will be from here you will start.



    1. To work with SOAP web services in the command line, you need to install the `WCF dotnet-svcutil` tool for .NET. In our case the `donet-svcutil`is already installed as you can see from the picture bellow. However, to ensure that you have it installed run always the following command.

    ```
    dotnet tool install --global dotnet-svcutil
    ```

```
Command Prompt
Microsoft Windows [Version 10.0.19043.2251]
(c) Microsoft Corporation. All rights reserved.

C:\Windows>cd C:\Users\miv\Projects\Testing AWS Functions\how-to-soap

C:\Users\miv\Projects\Testing AWS Functions\how-to-soap>dotnet tool install --global dotnet-svcutil
Tool 'dotnet-svcutil' is already installed.
```

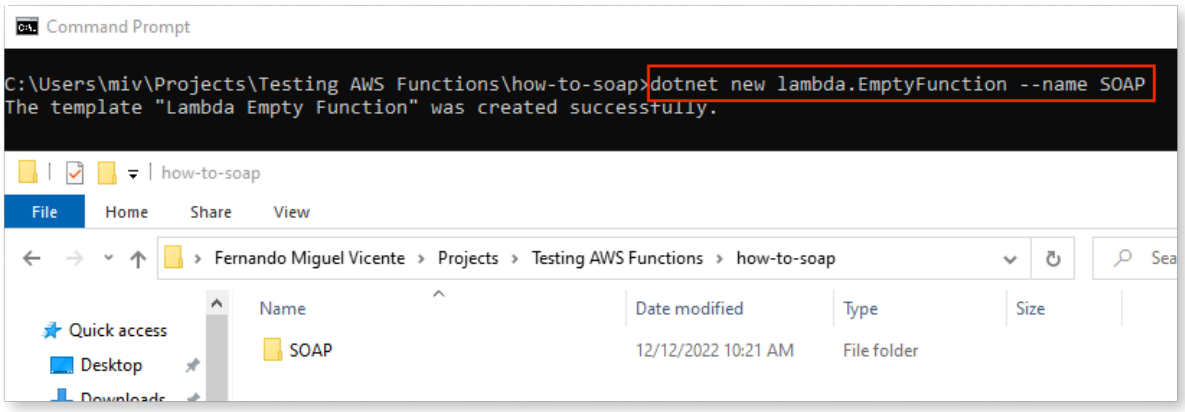
2. Lambda offers additional templates via the Amazon.Lambda.Templates NuGet package. Install the latest templates to get .NET 6 support. To install this package, run the following command:

```
dotnet new -i Amazon.Lambda.Templates
```

```
Command Prompt

C:\Users\miv\Projects\Testing AWS Functions\how-to-soap>dotnet new -i Amazon.Lambda.Templates
The following template packages will be installed:
   Amazon.Lambda.Templates

Amazon.Lambda.Templates is already installed, version: 6.8.1, it will be replaced with version .
Amazon.Lambda.Templates::6.8.1 was successfully uninstalled.
Success: Amazon.Lambda.Templates::6.9.0 installed the following templates:
Template Name                                Short Name                                   Language  Ta
-------------------------------------------  -------------------------------------------  --------  --
Empty Top-level Function                     lambda.EmptyTopLevelFunction                 [C#]      Aw
Lambda Annotations Framework (Preview)       serverless.Annotations                       [C#]      Aw
Lambda ASP.NET Core Minimal API              serverless.AspNetCoreMinimalAPI              [C#]      Aw
Lambda ASP.NET Core Web API                  serverless.AspNetCoreWebAPI                  [C#],F#   Aw
Lambda ASP.NET Core Web API (.NET 6 C...     serverless.image.AspNetCoreWebAPI            [C#],F#   Aw
Lambda ASP.NET Core Web Application w...     serverless.AspNetCoreWebApp                  [C#]      Aw
Lambda Custom Runtime Function (.NET 7)      lambda.CustomRuntimeFunction                 [C#],F#   Aw
Lambda Detect Image Labels                   lambda.DetectImageLabels                     [C#],F#   Aw
Lambda Empty Function                        lambda.EmptyFunction                         [C#],F#   Aw
Lambda Empty Function (.NET 7 Contain...     lambda.image.EmptyFunction                   [C#],F#   Aw
Lambda Empty Serverless                      serverless.EmptyServerless                   [C#],F#   Aw
Lambda Empty Serverless (.NET 7 Conta...     serverless.image.EmptyServerless             [C#],F#   Aw
Lambda Function project configured fo...     lambda.NativeAOT                             [C#],F#   Aw
Lambda Giraffe Web App                       serverless.Giraffe                           F#        Aw
Lambda Simple Application Load Balanc...     lambda.SimpleApplicationLoadBalancerFunction [C#]      Aw
Lambda Simple DynamoDB Function              lambda.DynamoDB                              [C#],F#   Aw
Lambda Simple Kinesis Firehose Function      lambda.KinesisFirehose                       [C#]      Aw
Lambda Simple Kinesis Function               lambda.Kinesis                               [C#],F#   Aw
Lambda Simple S3 Function                    lambda.S3                                    [C#],F#   Aw
Lambda Simple SNS Function                   lambda.SNS                                   [C#]      Aw
Lambda Simple SQS Function                   lambda.SQS                                   [C#]      Aw
Lex Book Trip Sample                         lambda.LexBookTripSample                     [C#]      Aw
Order Flowers Chatbot Tutorial               lambda.OrderFlowersChatbot                   [C#]      Aw
Serverless Detect Image Labels               serverless.DetectImageLabels                 [C#],F#   Aw
Serverless project configured for dep...     serverless.NativeAOT                         [C#],F#   Aw
Serverless Simple S3 Function                serverless.S3                                [C#],F#   Aw
Serverless WebSocket API                     serverless.WebSocketAPI                      [C#]      Aw
Step Functions Hello World                   serverless.StepFunctionsHelloWorld           [C#],F#   Aw


C:\Users\miv\Projects\Testing AWS Functions\how-to-soap>
```

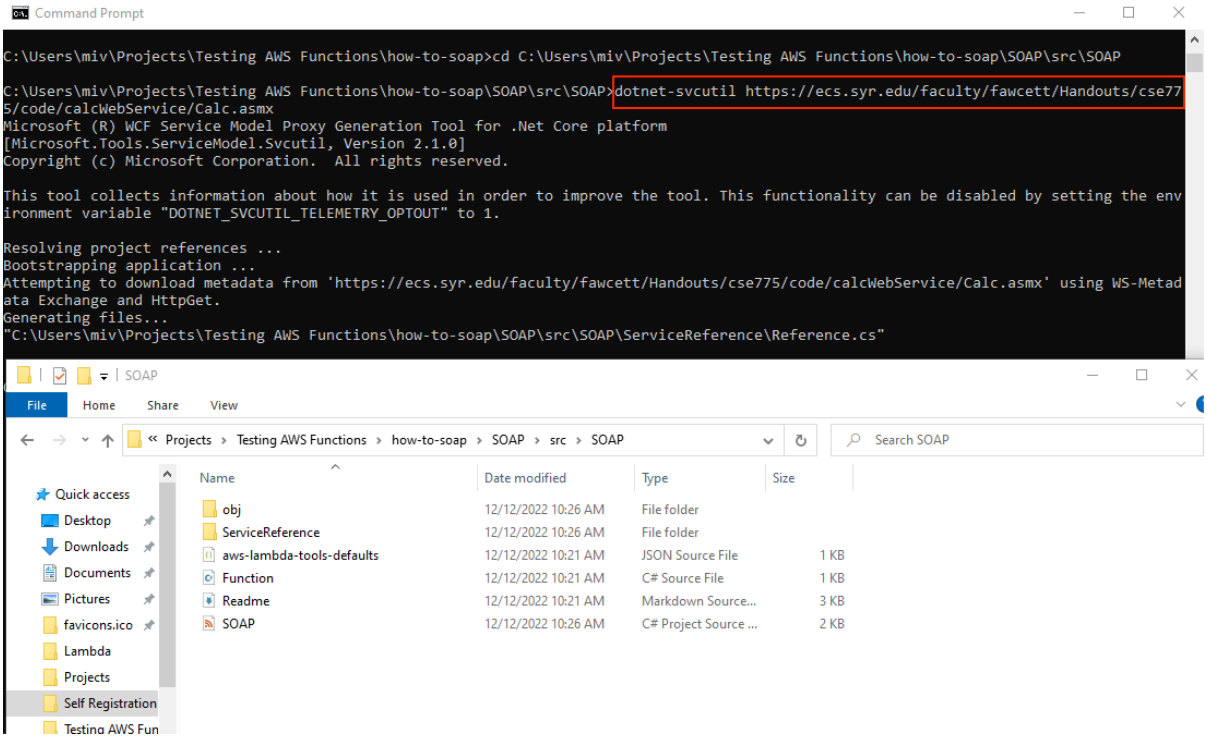
3. Now create a simple application for the Lambda function and use the command line and type the following command. In this example it was named SOAP.

```
dotnet new lambda.EmptyFunction --name <YOUR FUNCTION NAME>
```



4. Then in the command line open your project go to the directory `.\FunctionNameExample\src\FunctionNameExample\` and run the following command to install your web service using the SVCUtil. **Note:** *For testing purpose you can use a free SOAP WSDL available here.*

```
dotnet-svcutil https://<YOUR WSDL>
```



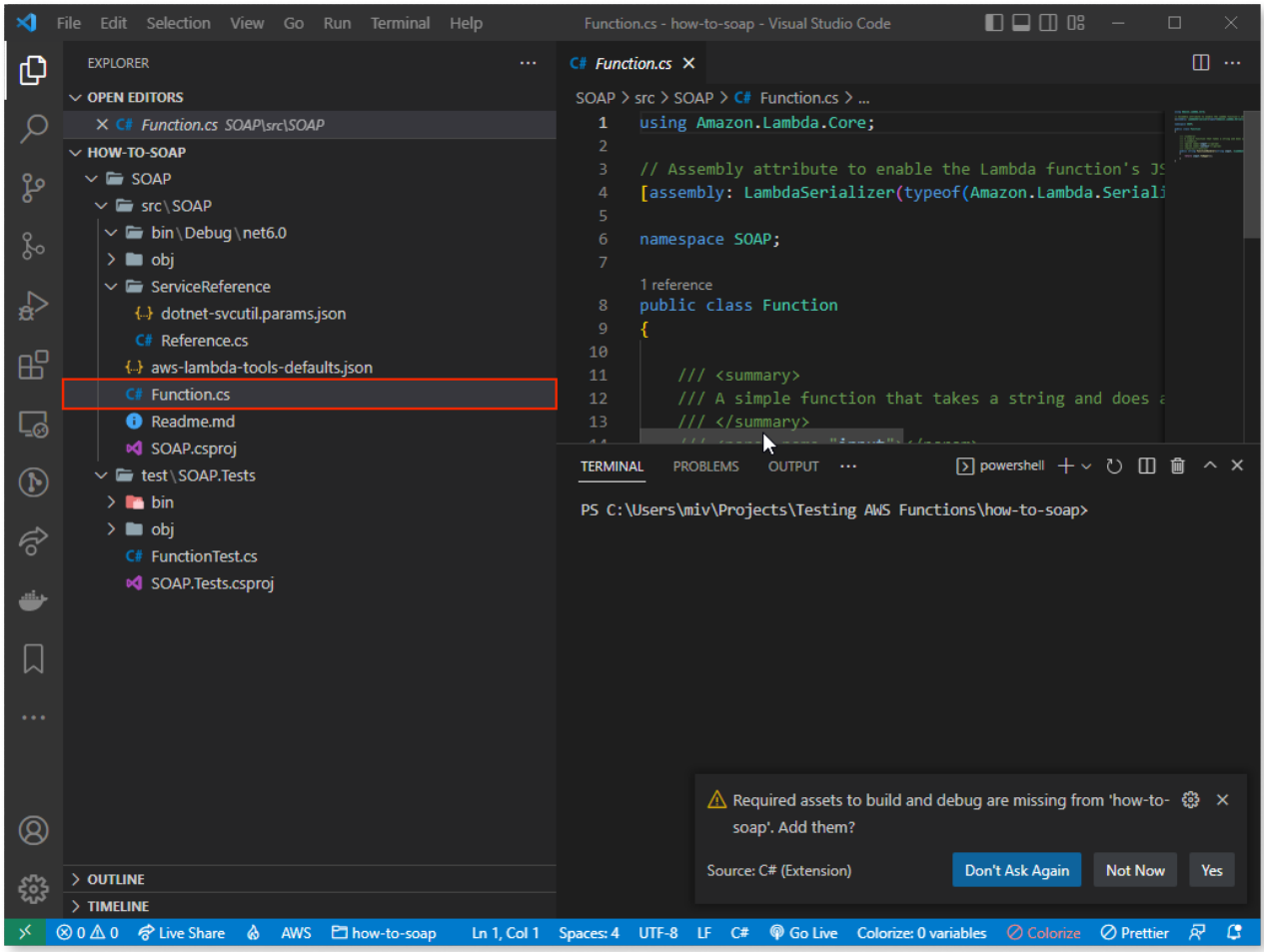
5. Install Nuget APIGatewayEvents

```
dotnet add package Amazon.Lambda.APIGatewayEvents --version 2.5.0
```

Prepare your SOAP Request to be consumed by a REST

1. Start by opening the Visual Studio Code > File > Open Folder then select your folder that contain your project. Note it must be the first folder that contains your project. See the example bellow.

2. Then Visual Studio Code will show a popup notifying you for required assets to build and debug the how-to-soap project. Select `YES`. It will create a new folder `.vscode` which will be useful in a few steps later.



3. Now in the `Functions.CS` you have a the `FunctionHandler` method which is very simple.

```
public string FunctionHandler(string input, ILambdaContext context)
{
    return input.ToUpper();
}
```

Add the calculator SOAP Web Service

1. Just before the FunctionHandler add the following code:

```
private static string ENDPOINTADDRESS =
"https://ecs.syr.edu/faculty/fawcett/Handouts/cse775/code/calcWebService/Calc.asmx";

private static int TIMEOUT = 1000;
```

```
SOAP > src > SOAP > C# Function.cs > ...
using Amazon.Lambda.Core;
using Amazon.Lambda.APIGatewayEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SOAP;

1 reference
public class Function
{
    0 references
    private static string ENDPOINTADDRESS = "https://ecs.syr.edu/faculty/fawcett/Handouts/cse775/code/calcWebService/Calc.asmx";
    0 references
    private static int TIMEOUT = 1000;

    0 references
    public string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }

}
```

Altering the Lambda function to accept the Function URL request

1. When the Lambda function is triggered from the Function URL the incoming request will contain all the HTTP info you would expect, query-string, body, headers, method, etc, this will be passed to Lambda function as `JSON`. To accept this, you will change the first parameter in the `FunctionHandler` method from a `string` to an `APIGatewayHttpApiV2ProxyRequest`.

2. But first, add the `Amazon.Lambda.APIGatewayEvents` package to the project.

    1. Right-click on top of SOAP folder and select the option `"Open in Integrated Terminal"`
    2. Then type in the terminal the following command:

    ```
    dotnet add package Amazon.Lambda.APIGatewayEvents
    ```

3. Add a using statement to the `Function.cs` file.

    ```
    using Amazon.Lambda.APIGatewayEvents;
    ```

4. Now replace the `FunctionHandler` signature from:

    ```
    public string FunctionHandler(string input, ILambdaContext context)
    ```

    To:

    ```
    public string FunctionHandler(APIGatewayHttpApiV2ProxyRequest request,
    ILambdaContext context)
    ```

1. With this change, the incoming request will be deserialized into the request object. Now you can extract the query string, path, etc from the request object. However, if you want to use the body of the request, another deserialization must be performed.

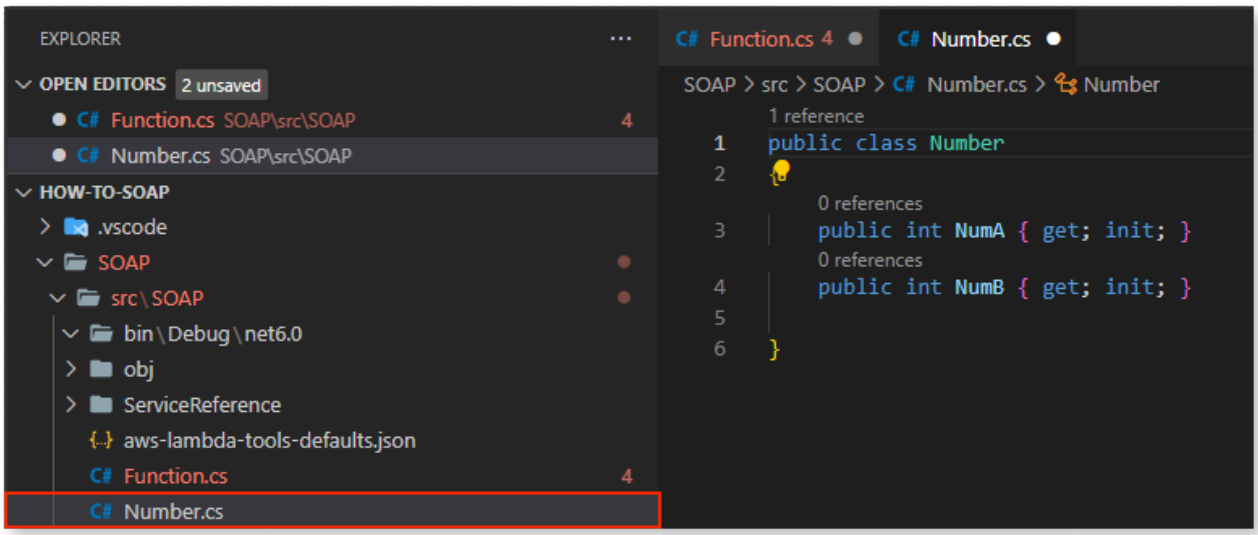
```
using Amazon.Lambda.Core;
using Amazon.Lambda.APIGatewayEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SOAP;

1 reference
public class Function
{
    0 references
    private static string ENDPOINTADDRESS = "https://ecs.syr.edu/faculty/fawcett/Handouts/cse775/code/calcWebService/Calc.asmx";
    0 references
    private static int TIMEOUT = 1000;

    0 references
    public string FunctionHandler(APIGatewayHttpApiV2ProxyRequest request, ILambdaContext context)
        {
            return input.ToUpper();
        }

}
```

Deserializing the body of the request

1. The `APIGatewayHttpApiV2ProxyRequest.Body` contains the body of any request that supports a body (PUT, POST, PATCH, etc). But this is a string and has to be explicitly deserialized into a type you define.

2. In this example, you will deserialize a Numbers from the body of the request. So start by create a new file and name it `Numbers.CS` and inside add the following code:

```
public class Numbers
{
    public int NumA { get; init; }
    public int NumB { get; init; }

}
```



3. Back in `Function.cs`, add two `using` statements just after the `APIGatewayEvents`;

```
using System.Text.Json;
using System.Text.Json.Serialization;
```

4. And change the `FunctionHandler` method to this

```
public int FunctionHandler(APIGatewayHttpApiV2ProxyRequest request,
ILambdaContext context)
{
    var serializationOptions = new JsonSerializerOptions
    {
        PropertyNameCaseInsensitive = true,
        NumberHandling= JsonNumberHandling.AllowReadingFromString
    };

    Number number = JsonSerializer.Deserialize<Number>(request.Body,
serializationOptions);
    int numA = number.NumA;
    int numB = number.NumB;

}
```

```
using Amazon.Lambda.Core;
using Amazon.Lambda.APIGatewayEvents;
using System.Text.Json;
using System.Text.Json.Serialization;

// Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SOAP;

1 reference
public class Function
{
    0 references
    private static string ENDPOINTADDRESS = "https://ecs.syr.edu/faculty/fawcett/Handouts/cse775/code/calcWebService/Calc.asmx";
    0 references
    private static int TIMEOUT = 1000;
    1 reference
    public string FunctionHandler(APIGatewayHttpApiV2ProxyRequest request, ILambdaContext context)
    {
        var serializationOptions = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true,
            NumberHandling= JsonNumberHandling.AllowReadingFromString
        };

        Number number = JsonSerializer.Deserialize<Number>(request.Body, serializationOptions);
        int numA = number.NumA;
        int numB = number.NumB;

    }
}
```

1. Setting `serializationOptions` is not necessary but makes it easier for your function to handle capitalization issues and the presence of numbers as strings in the request.

Build the SOAP Request

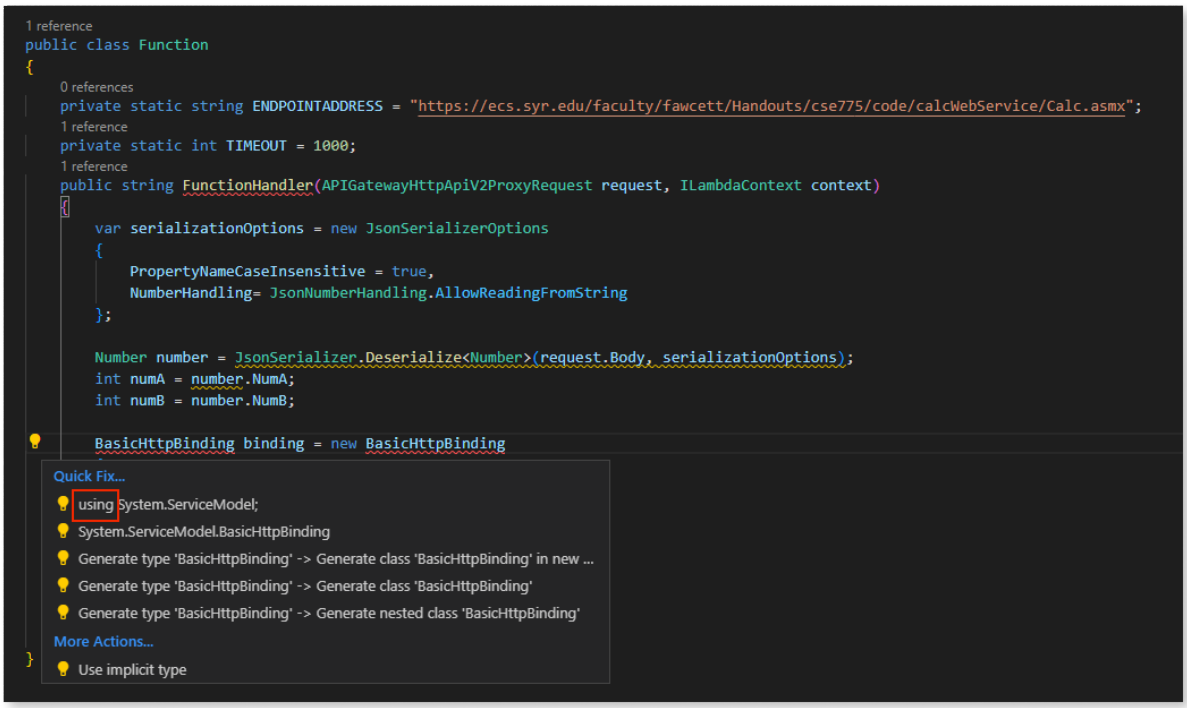
1. In a way you can start creating the request you need to start by adding the properties for the `BasicHttpBinding`.

    1. The `BasicHttpBinding` uses `HTTP` as the transport for sending `SOAP 1.1` messages. A service can use this binding to expose endpoints that conform to `WS-I BP 1.1`, such as those that `ASMX` clients access. Similarly, a client can use the `BasicHttpBinding` to communicate with services exposing endpoints that conform to `WS-I BP 1.1`, such as `ASMX` Web services or `Windows Communication Foundation (WCF)` services configured with the BasicHttpBinding.

    2. So add the following code:

    ```
    BasicHttpBinding binding = new BasicHttpBinding
    {
        SendTimeout = TimeSpan.FromSeconds(TIMEOUT),
        MaxBufferSize = int.MaxValue,
        MaxReceivedMessageSize = int.MaxValue,
        AllowCookies = true,
        ReaderQuotas = XmlDictionaryReaderQuotas.Max
    };
    ```

    3. Now you will get errors due to missing dependencies. So if you click on top of the `BasicHttpBiding` lamp to `Show Code Actions` select the first option `“using Sistem.ServiceModel”` and automatically will add a new using on your code.

    4. You can do the same for `XmlDictionaryReaderQuotas` and it will add a new dependency `“using System.xml”`.

2. Define the `security mode` and the `endpoint` address. So add the following lines after the `BasicHttpBinding` object.

```
binding.Security.Mode = BasicHttpSecurityMode.Transport;

EndpointAddress address = new EndpointAddress(ENDPOINTADDRESS);
```

3. Now is the time to create the channel based on the `basichttpbinding` and the endpoint defined previously. So add the following code.

```
CalculatorWebServiceSoapChannel proxy = new
ChannelFactory<CalculatorWebServiceSoapChannel>(binding,
address).CreateChannel();
```

   1. You will get an error in the `CalculatorWebServiceSoapChannel` due to a missing dependency. So if you click on the top of the `CalculatorWebServiceSoapChannel` lamp to `Show Code Actions` select the first option `“using ServiceReference”` and automatically will add a new using to your code.

4. As the last step you will return the method AddSync from the calculator with two input parameters which are the numA and numB to calculate.

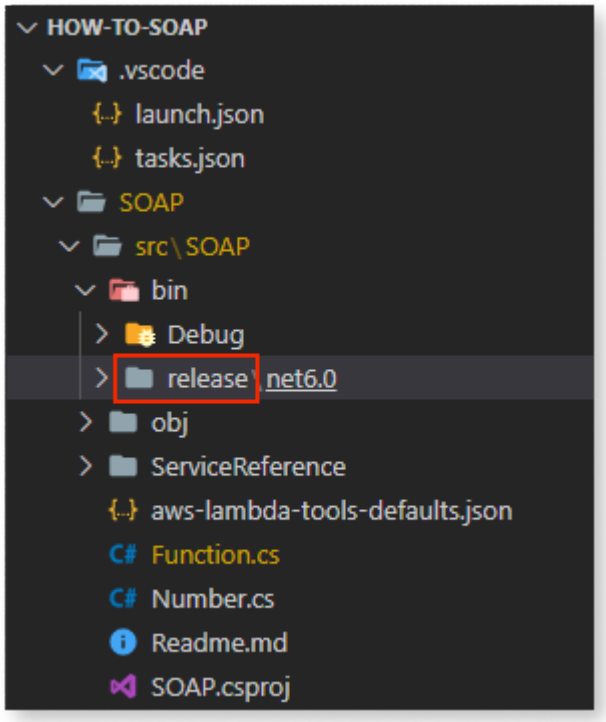
```
return proxy.AddAsync(numA, numB).Result;
```

5. Now to build in release mode, you need to open the folder `.vscode` > `tasks.json` and inside the `task` > `label` build in the arguments and add the following arguments and save the `JSON` file.

   1. `“-c”,`
   2. `“release”`

```
"version": "2.0.0",
"tasks": [
    {
        "label": "build",
        "command": "dotnet",
        "type": "process",
        "args": [
            "build",
            "${workspaceFolder}/SOAP/test/SOAP.Tests/SOAP.Tests.csproj",
            "/property:GenerateFullPaths=true",
            "/consoleloggerparameters:NoSummary",
            "-c",
            "release"
        ],
        "problemMatcher": "$msCompile"
    },
```

6. Before making a project build ensure you have your tests prepared and available on the test folder > `FunctionTests.CS`. Then do a `CTRL + SHIFT + B` to make a build in release mode. After that, you will have inside the bin folder a release folder.

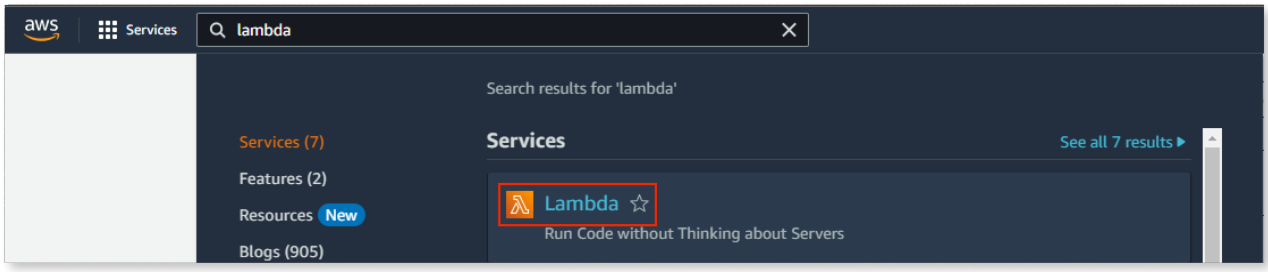


Configure AWS lambda to consume the SOAP web service

Create a Lambda Function in AWS

1. Create an AWS account in the first place and set the MFA to reduce the risk of attacks on your account.

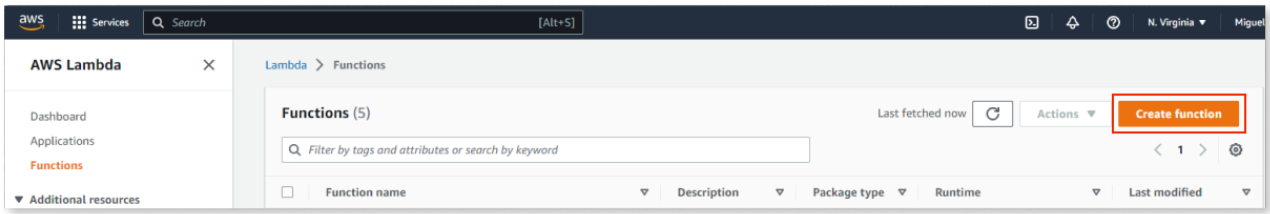
2. After you create an account and log in, at the top, search and select the Lambda option.



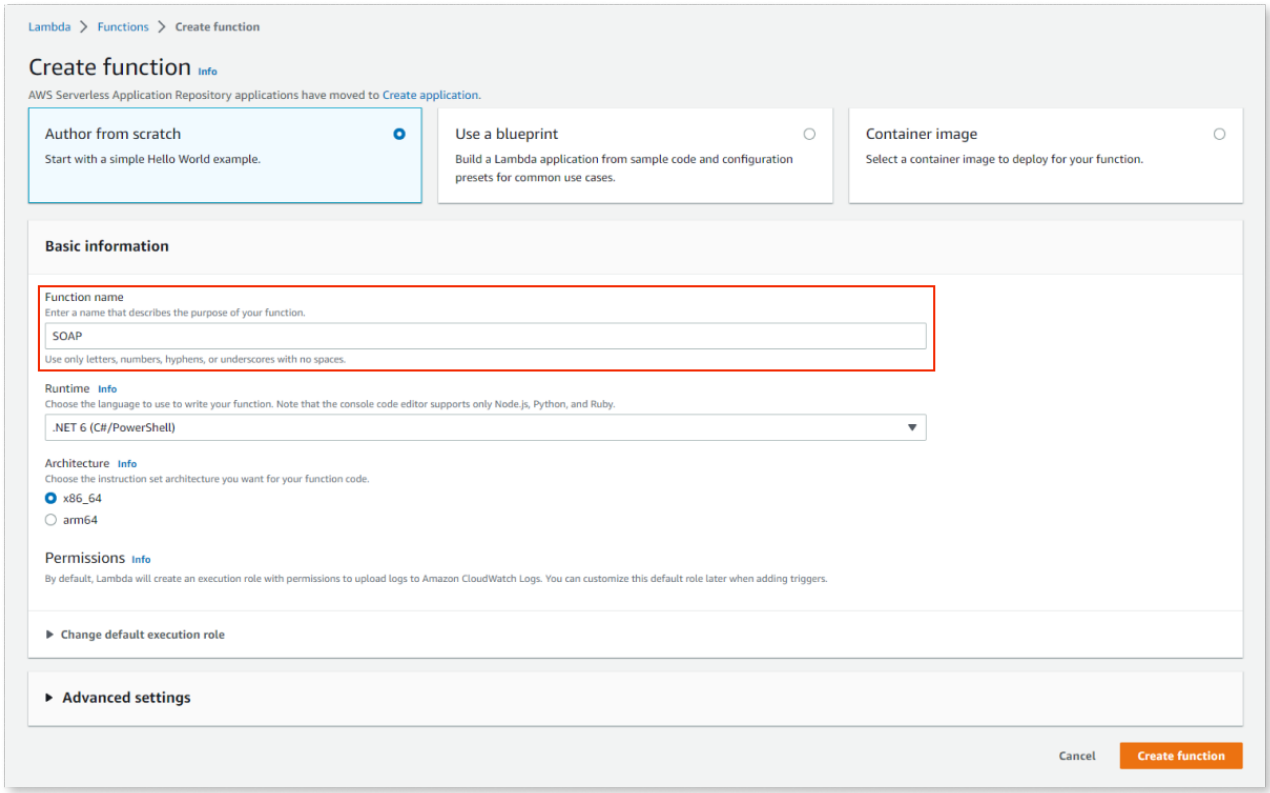
3. Now select the option create function.

4. Then inside the Lambda function add your name function or you can use the one from the example "SOAP" and then in the Runtime option select .NET 6 (C#/PowerShell).

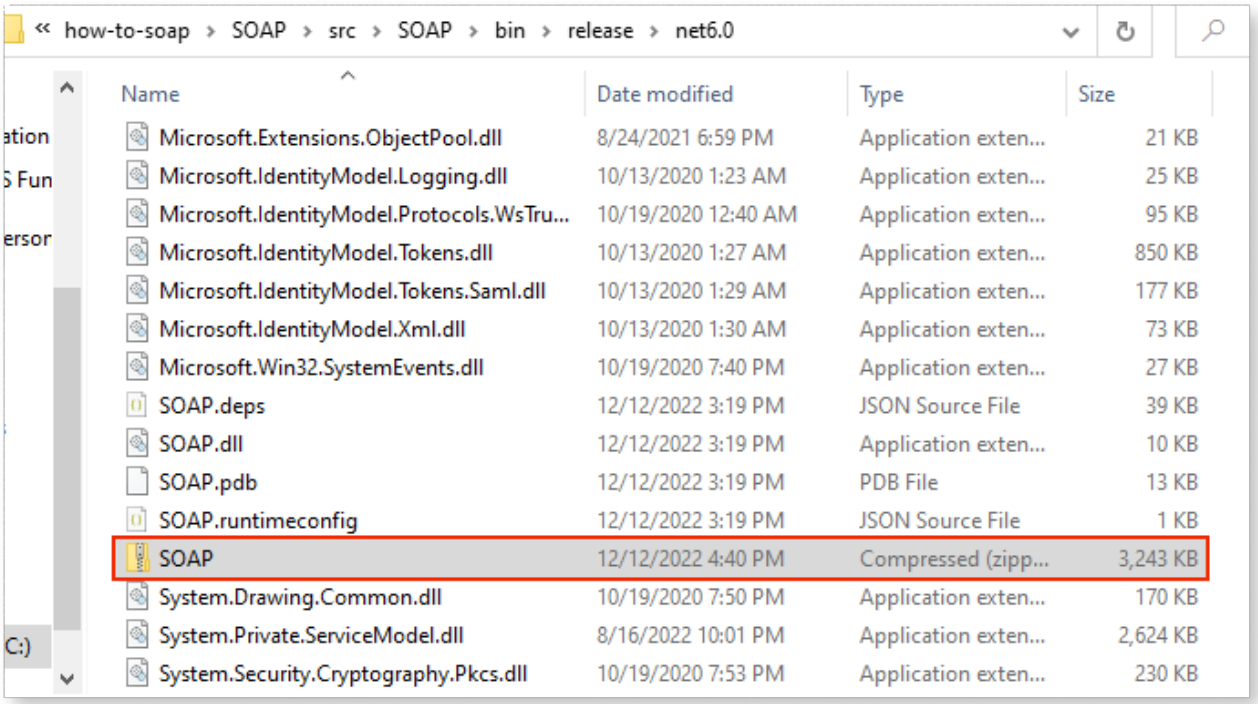
5. Click Create function.



6. You have a Lambda function created. However, you need to add the source code.

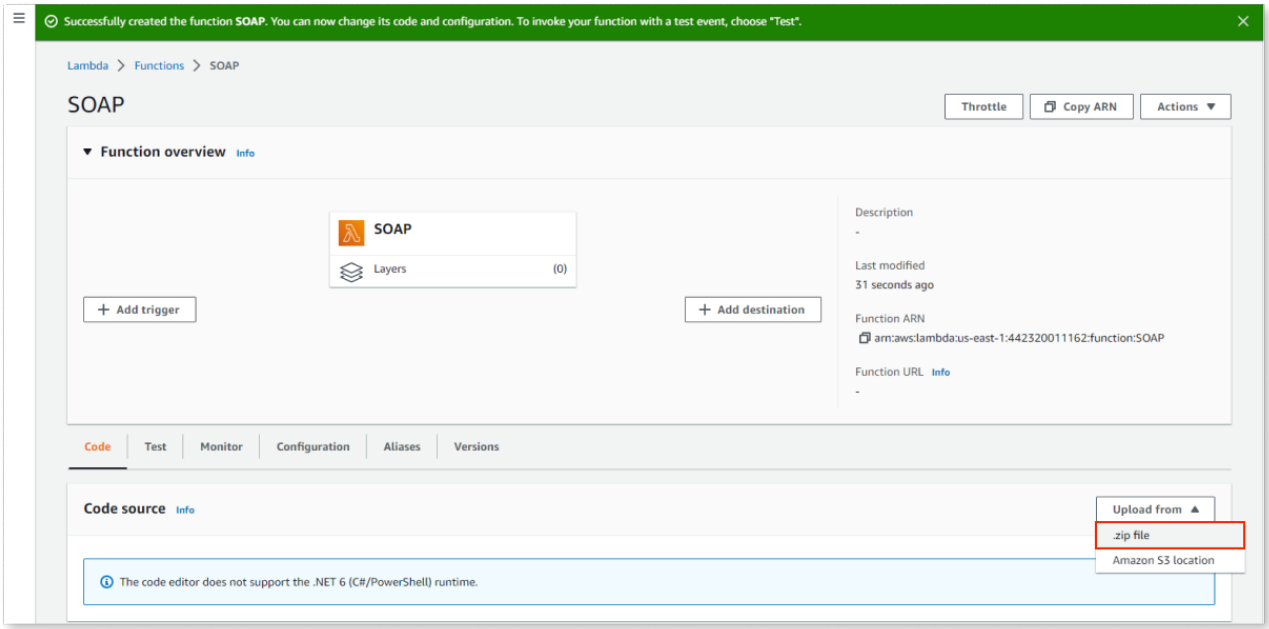


7. To get the source code of your project you can go to your project folder and select the folder inside `src` > `bin` > `release` > `net6.0`. Select all the elements inside, add them to a zip file, and name it SOAP.



8. Upload the zip file to the AWS source code.

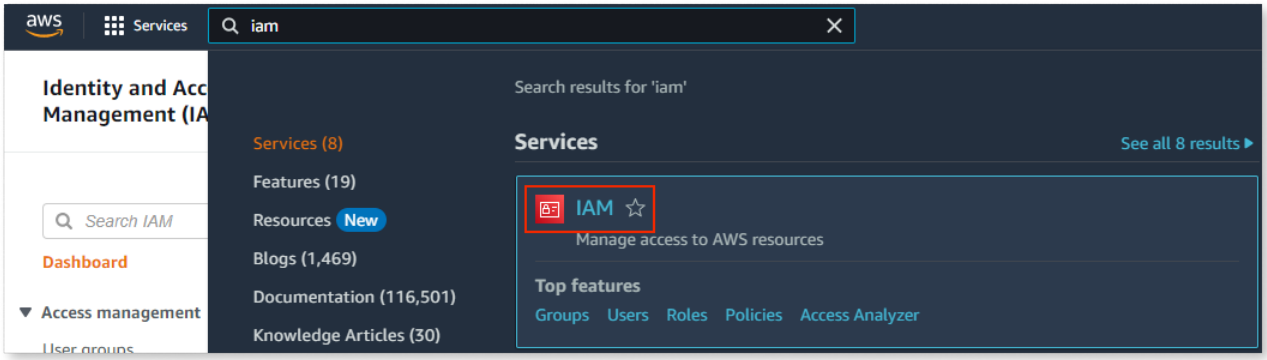
9. After uploading the source code go to Configuration Tab, and in the left menu, select the Function URL. Inside, select, the button `Create function URL`.



10. Select the authentication type AWS_IAM and click Save.

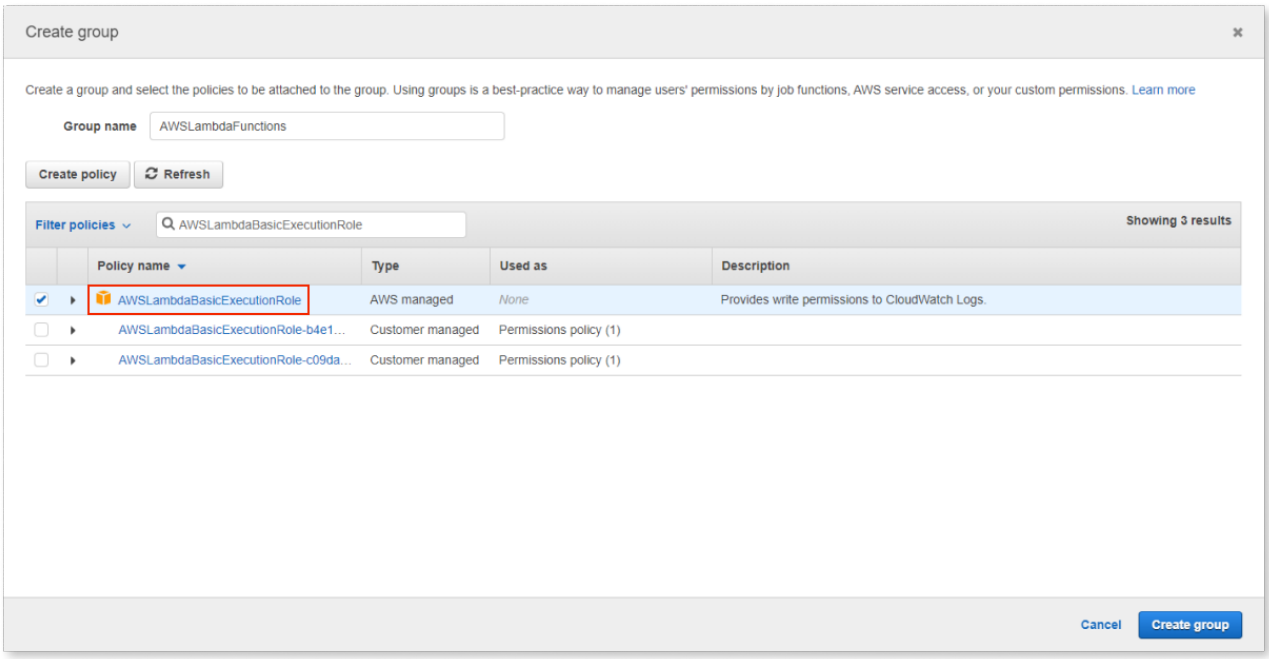
Create an IAM user in AWS and manage credentials

1. At the top of the page, search for `IAM` and select the IAM option. Then select the option `Add user` to create a new user.



2. Now, in the user wizard, name a user name, and in the access, type selects the first option with the `Access key - Programmatic access`.

3. Then, if you don't have a group of permissions, create a new group and name it `AWSLambdaFunction`. Then in the permissions section search and select the `AWSLambdaBasicExecutionRole` but it's up to you which permissions you would like to add to that specific user. This permission it's our recommendation if you will be using only for Lambda functions to protect your account.



4. You can bypass the Tags step and, in the last step, copy a safe place for the `Access Key Id` and the `Secret access key`. ***Note:*** *the secret access key is only available one time when created for you to copy.*

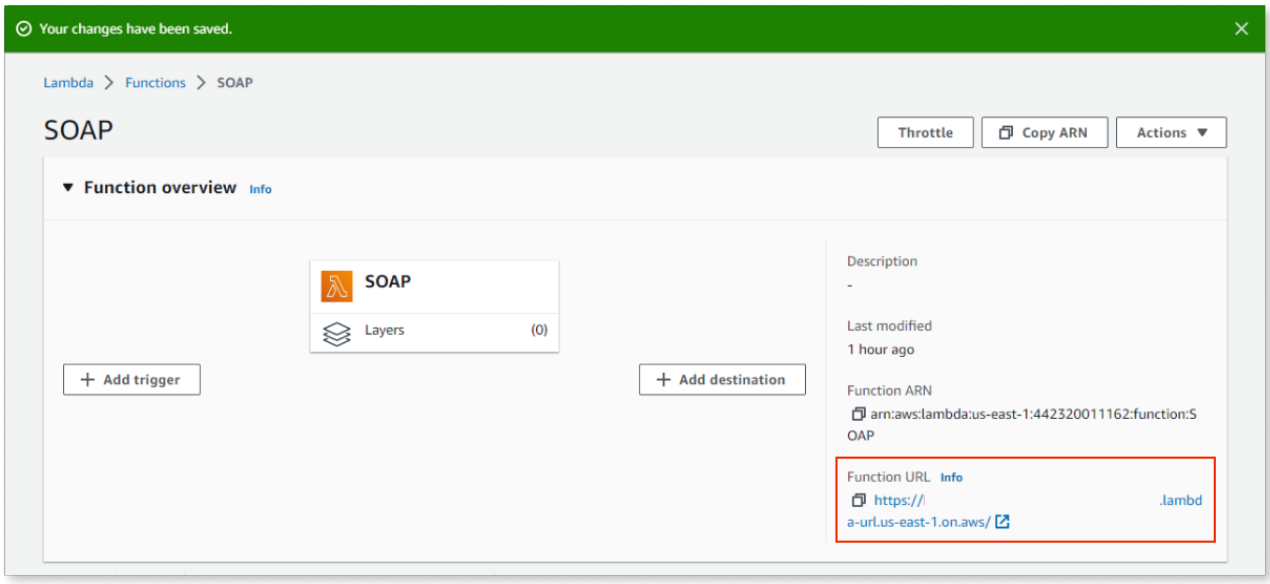
Create the REST in ODC Studio to consume the Lambda function

1. In your organization, install the `AWS Signature` forge component.

2. Now, create or use an existing application in OutSystems Developer Cloud (ODC).

3. Go to `Logic tab` > `integrations` > `REST`, then right-click on top of the REST and select the option to consume `REST API` > Add single method, and in the Method, URL select `POST` option.

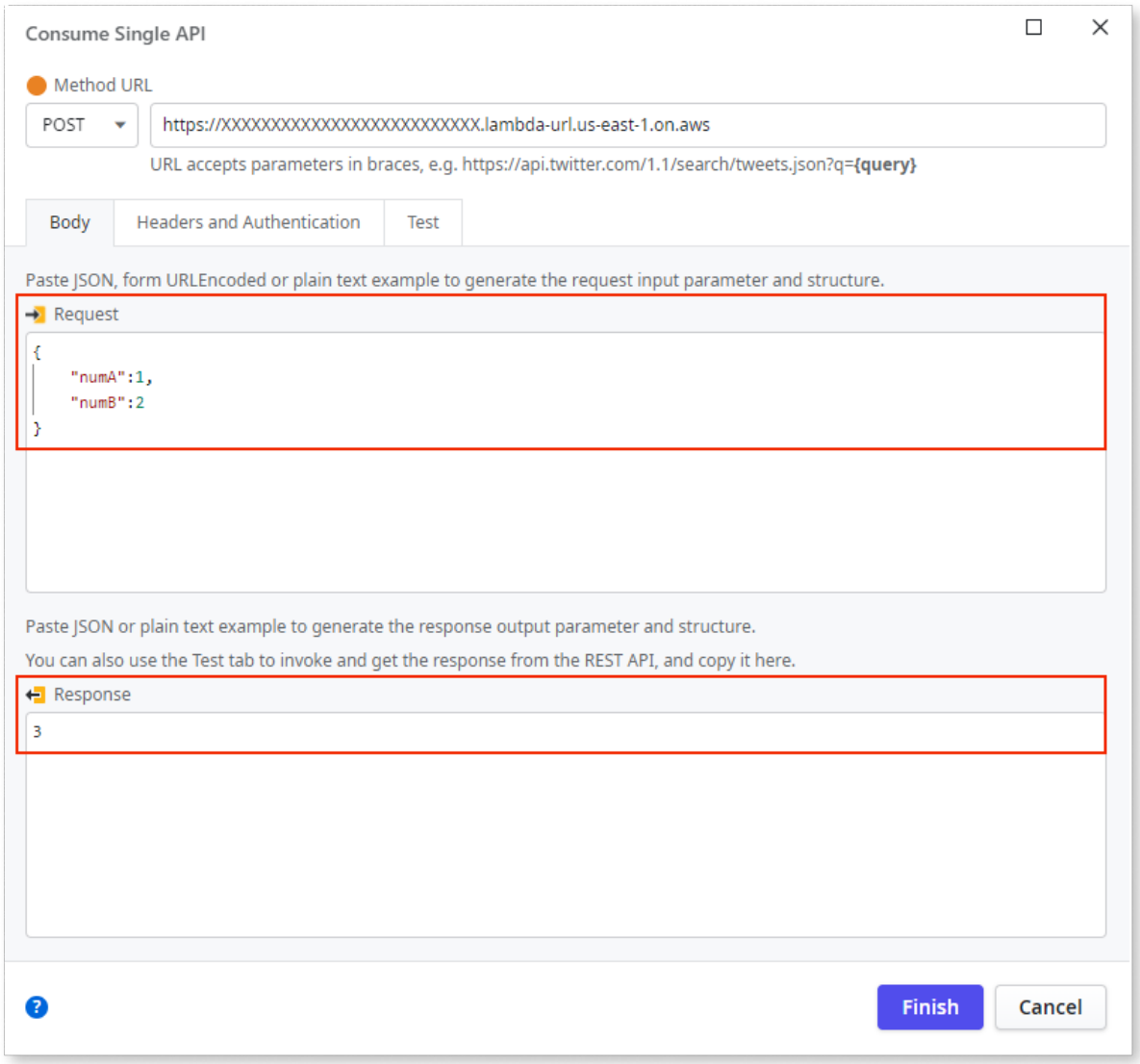
4. Go back to `AWS SOAP` Lambda, and in the function overview, copy the `Function URL`.



5. Then go back to ODC studio and paste it into the `Method URL`. In the Request field, add the following `JSON`.

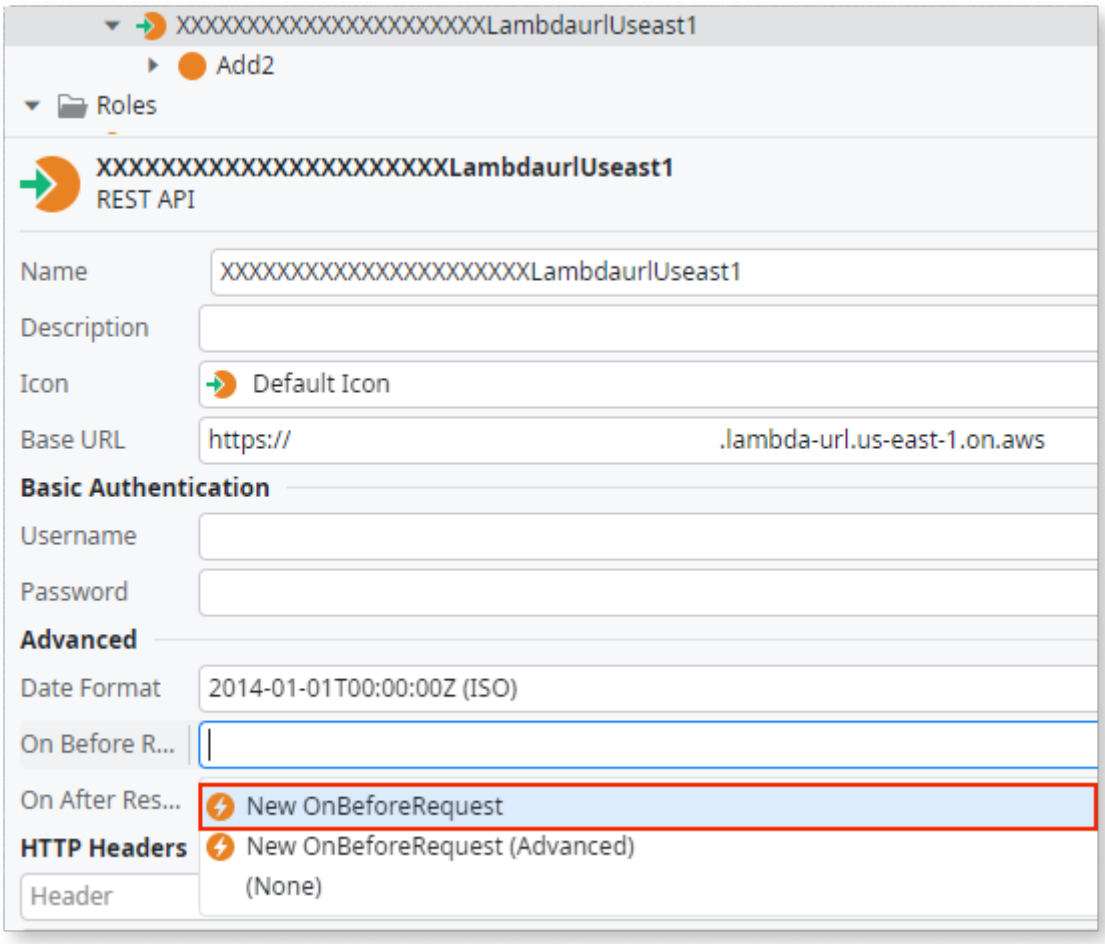
```
{
    "numA":1,
    "numB":2
}
```

1. In the Response field, add 3 and click on the Finish button.

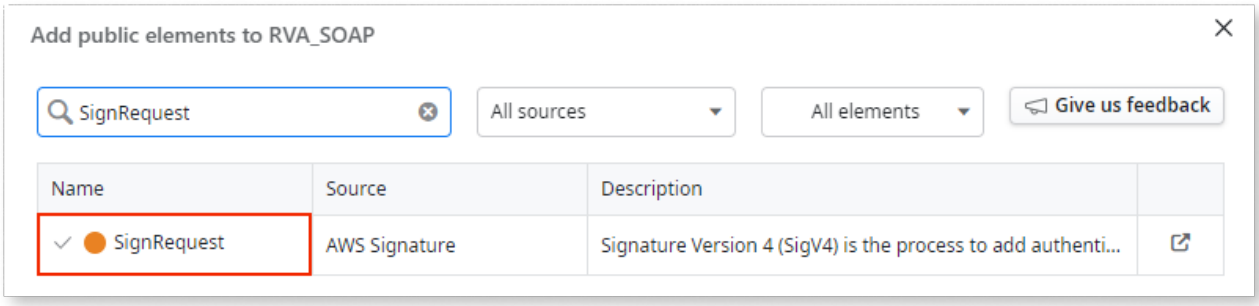


2. Then rename the `REST API` to something that you like and the method name it to `Add`.

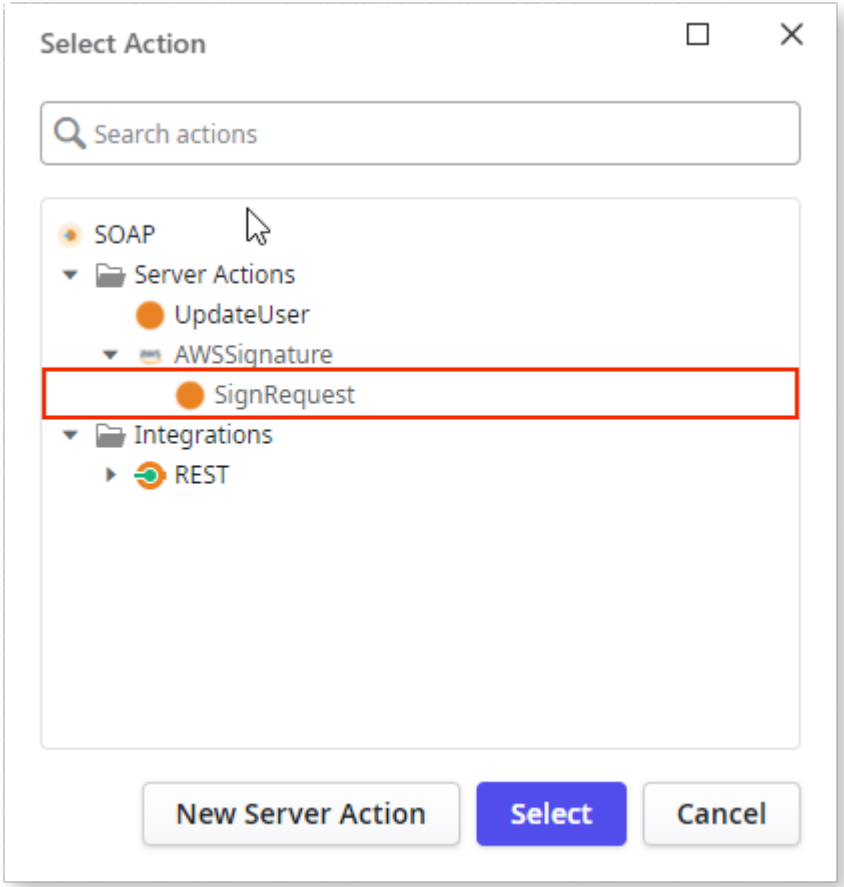
3. Now select your `REST API`, and in the advanced properties, open the option `On Before Request` and select the option `New OnBeforeRequest`.



6. Open the OnBeforeRequest REST API Callback, and add a new public element on the top of the plugin icon. Search for SignRequest from AWS Signature and add it to your application.

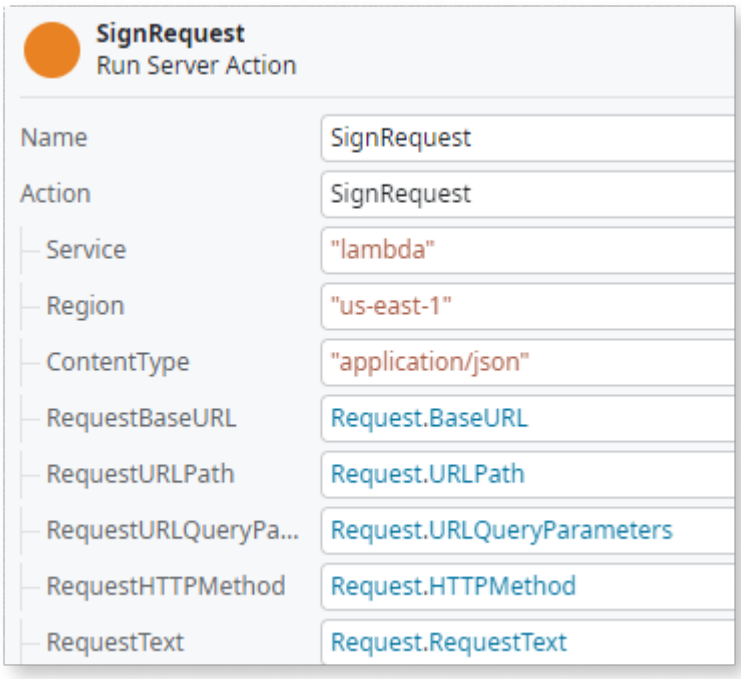


1. In the OnBeforeRequest drag and drop just before the assign a Run Server Action and search and select SignRequest server action from AWSSignature Library.

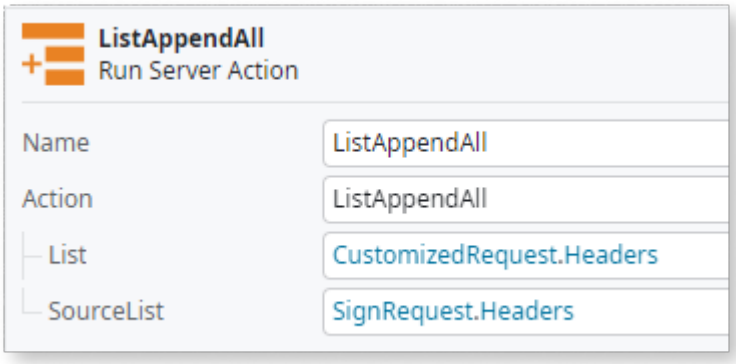


2. In the `SignRequest` add the following properties:

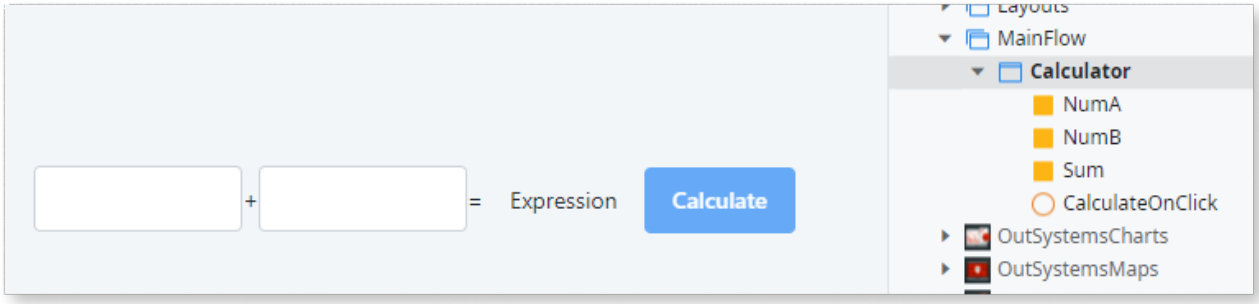
    1. Service as `"lambda"`
    2. Region as `"us-east-1"`
    3. ContentType as `"application/json"`
    4. RequestBaseURL as `Request.BaseURL`
    5. RequestURPath as `Request.URLPath`
    6. RequestURLQueryParameters as `Request.URLQueryParameters`
    7. RequestHTTPMethod as `Request.HTTPMethod`
    8. RequestText as `Request.RequestText`



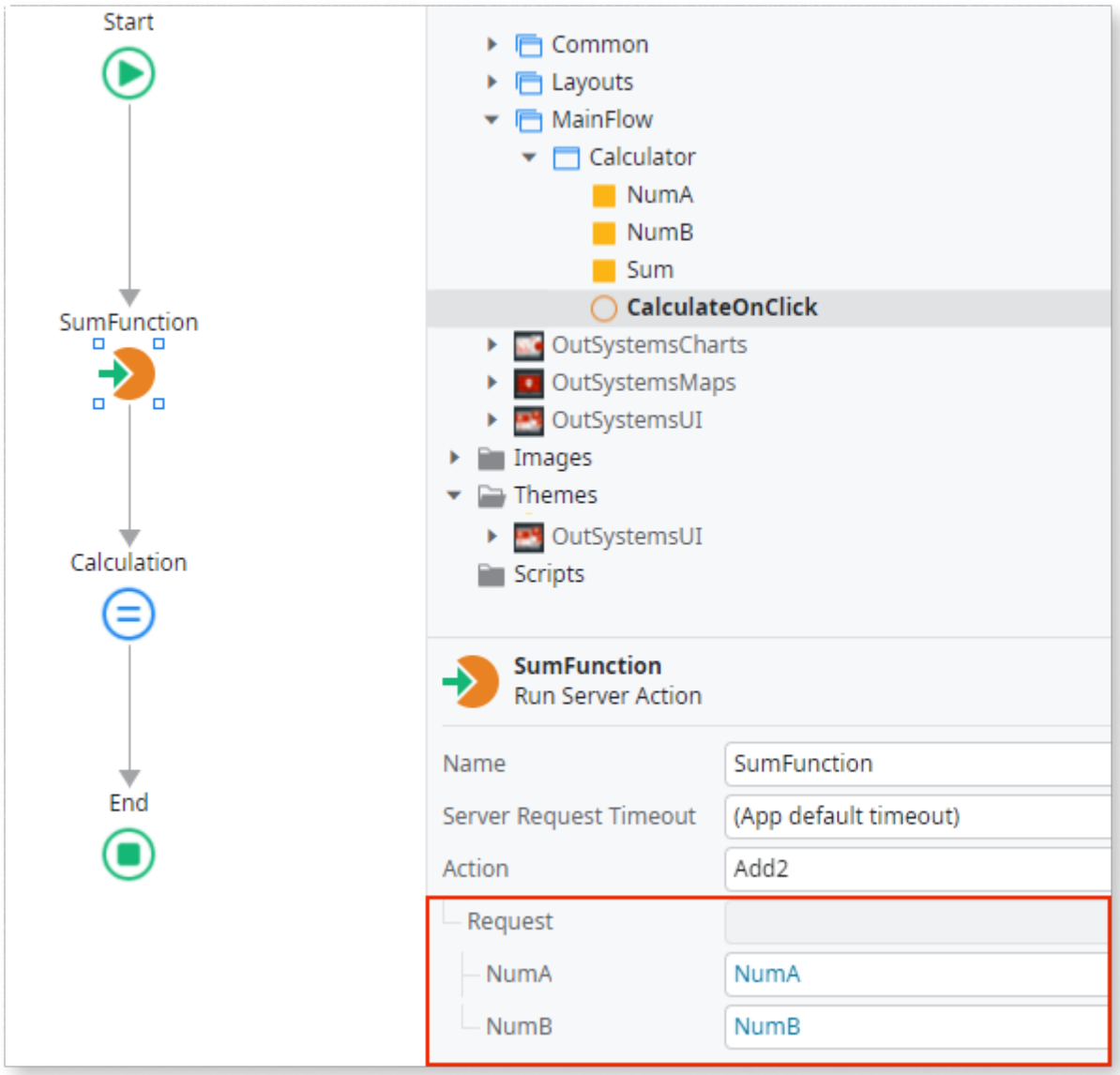
3. After the assign drag and drop another `Run Server Action` and search and select the `ListAppendAll`. Then in the `ListAppendAll` properties in the `List` add the `CustomizedRequest.Headers` and in the `SourceList` add `SignRequest.Headers`.



7. After you set the configurations in the portal you can go back to your app in `ODC Studio` and create a simple screen that contains two input fields to add any number.



8. Add a button to call the `REST API` to do the calculation. In this case, will be a `SUM`.



Setup the AWS Signature configurations in ODC Portal

1. Go to Apps and select your app and in the configurations tab add the information already provided in AWS IAM for the configurations `AWS_ACCESS_KEY_ID` and the `AWS_SECRET_ACCESS_KEY`.