.Net Serverless ASP.NET Core web application

Self-Paced Lab

Version 1.0

Duration: 30 minutes

Purpose & Background

In this lab, you will create a simple ASP.NET Core Web, deploy to AWS Lambda using the Visual Studio leveraging the AWS Serverless Application Model (AWS SAM) to create the lambda function and the Amazon API Gateway as a proxy layer in front of the Web App.

A significant change with ASP.NET Core 2.0 is that Razor Pages are now precompiled at publish time. It means when our serverless Razor Pages are first rendered, Lambda compute time isn't spent compiling the Razor Pages from cshtml to machine instructions; hence it makes possible to execute web app on top of lambda functions.

Lab Exercises

The following exercises should be completed in order for this lab:

1. Create an AWS Serverless Application (.NET Core) Project
2. Deploy to AWS Lambda
3. Check the AWS resources created
4. Change the .Net Razor and Re-deploy the lambda function
5. Remove all the Resources deployed

Prerequisites

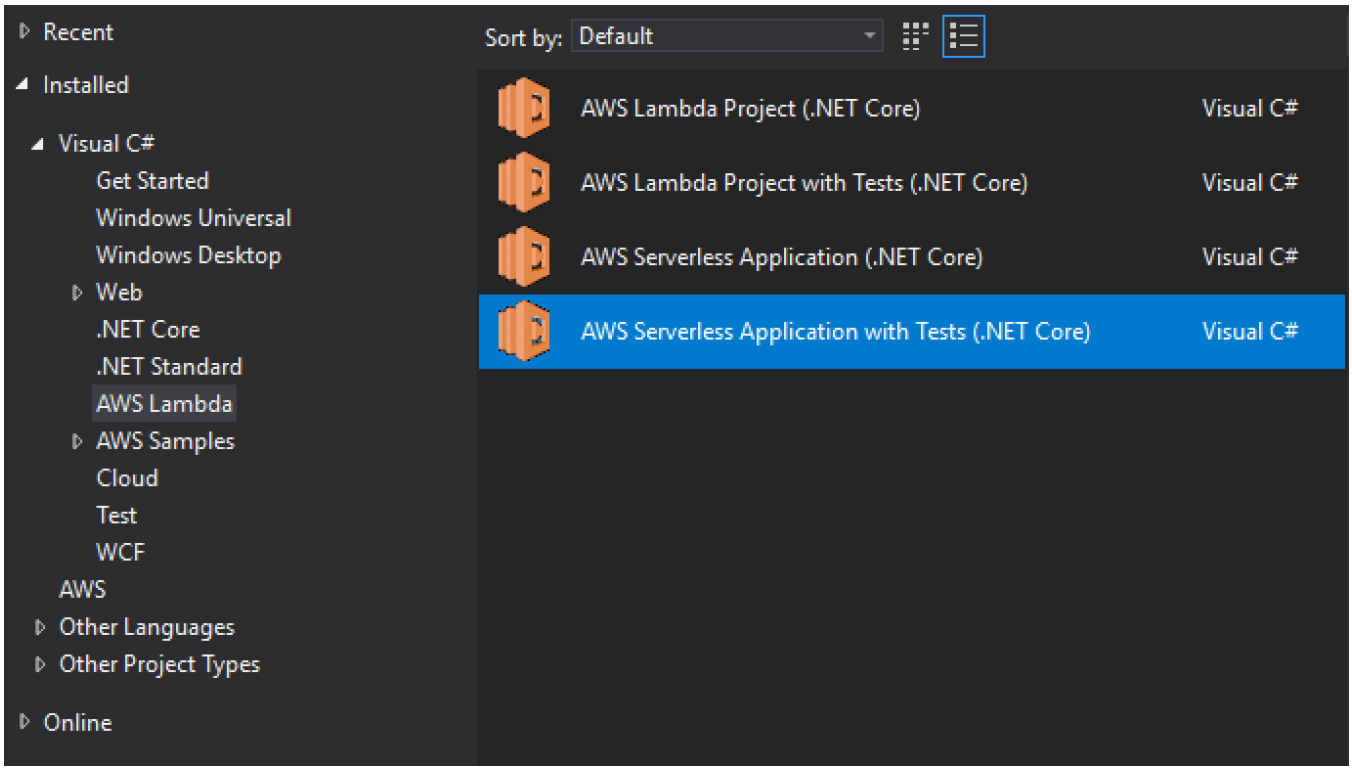
The following are the prerequisites required in order to complete the lab:

- Microsoft Visual Studio 2017 or above installed on your computer
- AWS Toolkit for Visual Studio
- Internet connection
- AWS Account

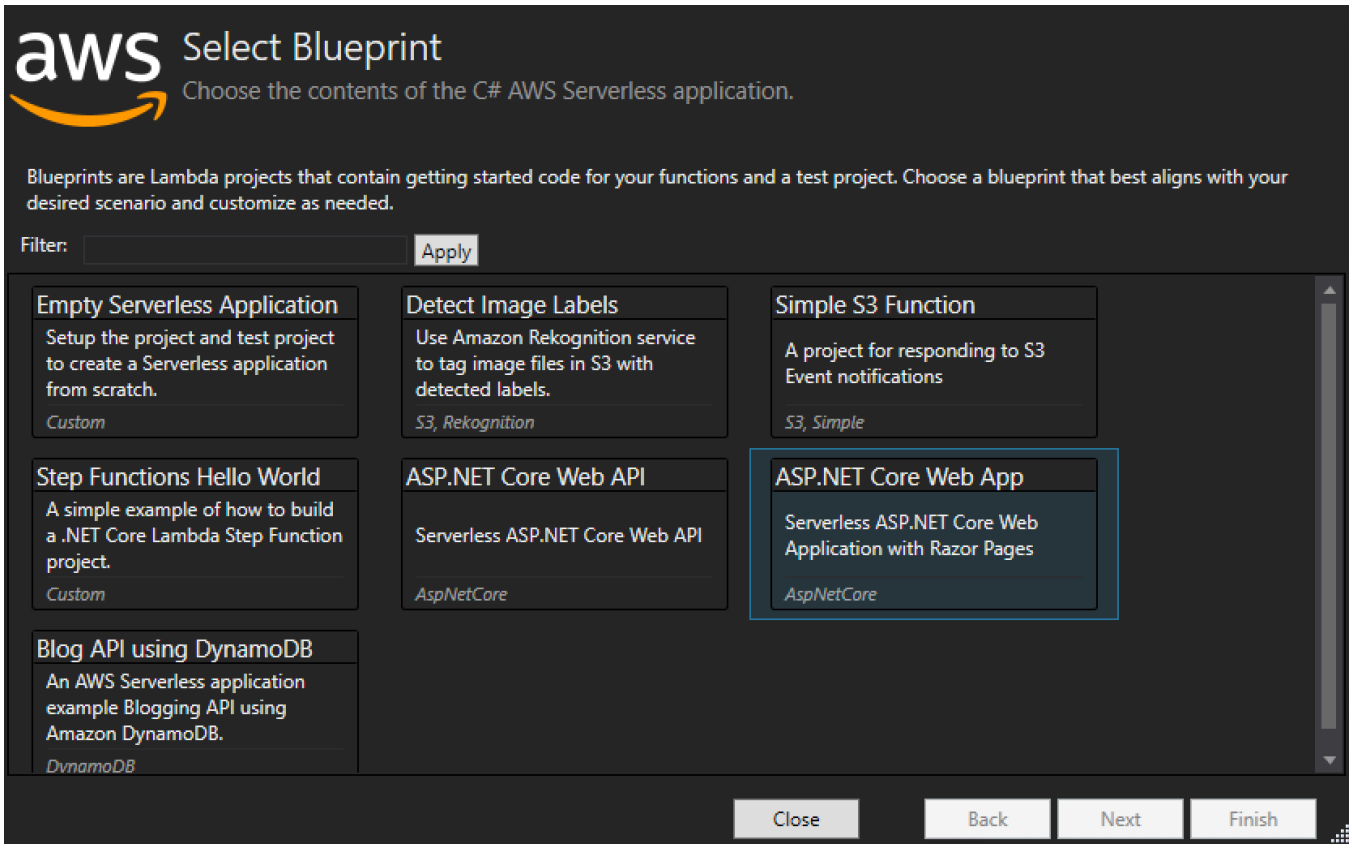
Part 1 – Create an ASP.NET Web Project

Follow the steps below to create and customize an ASP.NET Web Project in Visual Studio.

1. In Visual Studio, use File -> New -> Project to open the New Project dialog.
2. Under the Web project node, select AWS Lambda and the "AWS Serverless Application with Tests (.Net Core)" template, type **WebApp-Lambda-Userid** as the name for your project, then click the OK button.



3. In the next dialog, select the "API" blueprint, and select "ASP.NET Core Web App", then click the Finish button to generate the project.



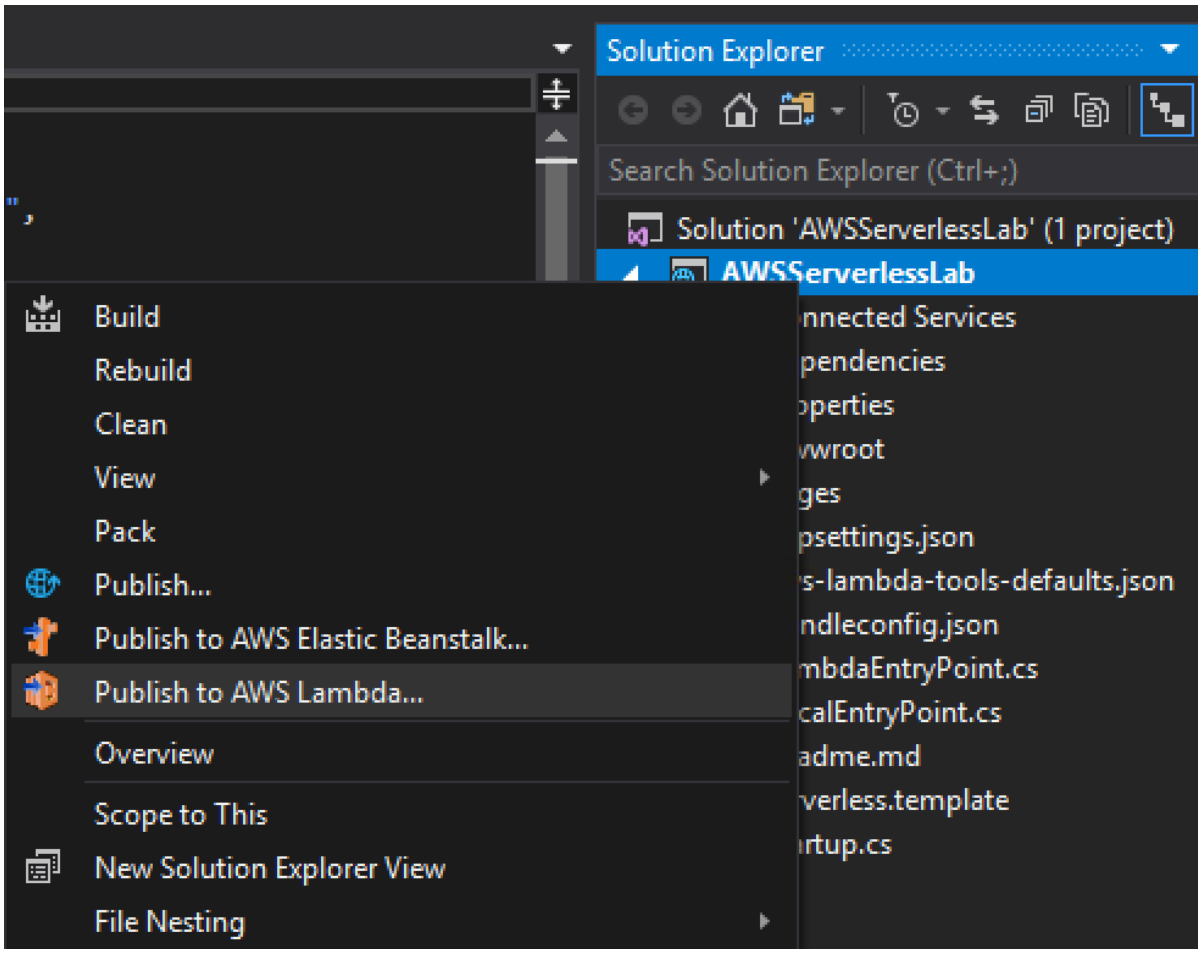
4. Open the serveless.template file from your project. This file defines the Lambda and API Gateway configuration. Find the lambda timeout configuration and change it from 30 seconds to 120 seconds, as for the excerpt below:

```
AspNetCoreFunction" : {
    "Type" : "AWS::Serverless::Function",
    "Properties": {
     "Handler":
"AWSServerlessLab::AWSServerlessLab.LambdaEntryPoint::FunctionHandlerAsync",
      "Runtime": "dotnetcore2.1",
      "CodeUri": "",
      "MemorySize": 512,
      "Timeout": 120,
      "Role": null,
      "Policies": [ "AWSLambdaFullAccess" ],
      "Environment" : {
       "Variables" : {
       }
},
```

Part 2 – Deploy to AWS Lambda

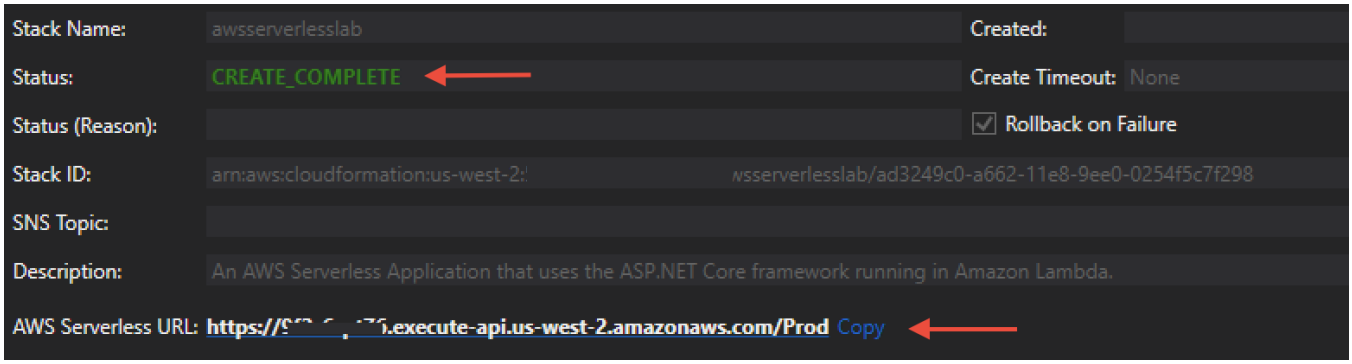
Follow the steps below to deploy the ASP.NET Core Web application to AWS Lambda.

1. Right-click your project in Solution Explorer, and select "Publish to AWS Lambda" to launch the publishing wizard. See the figure below.

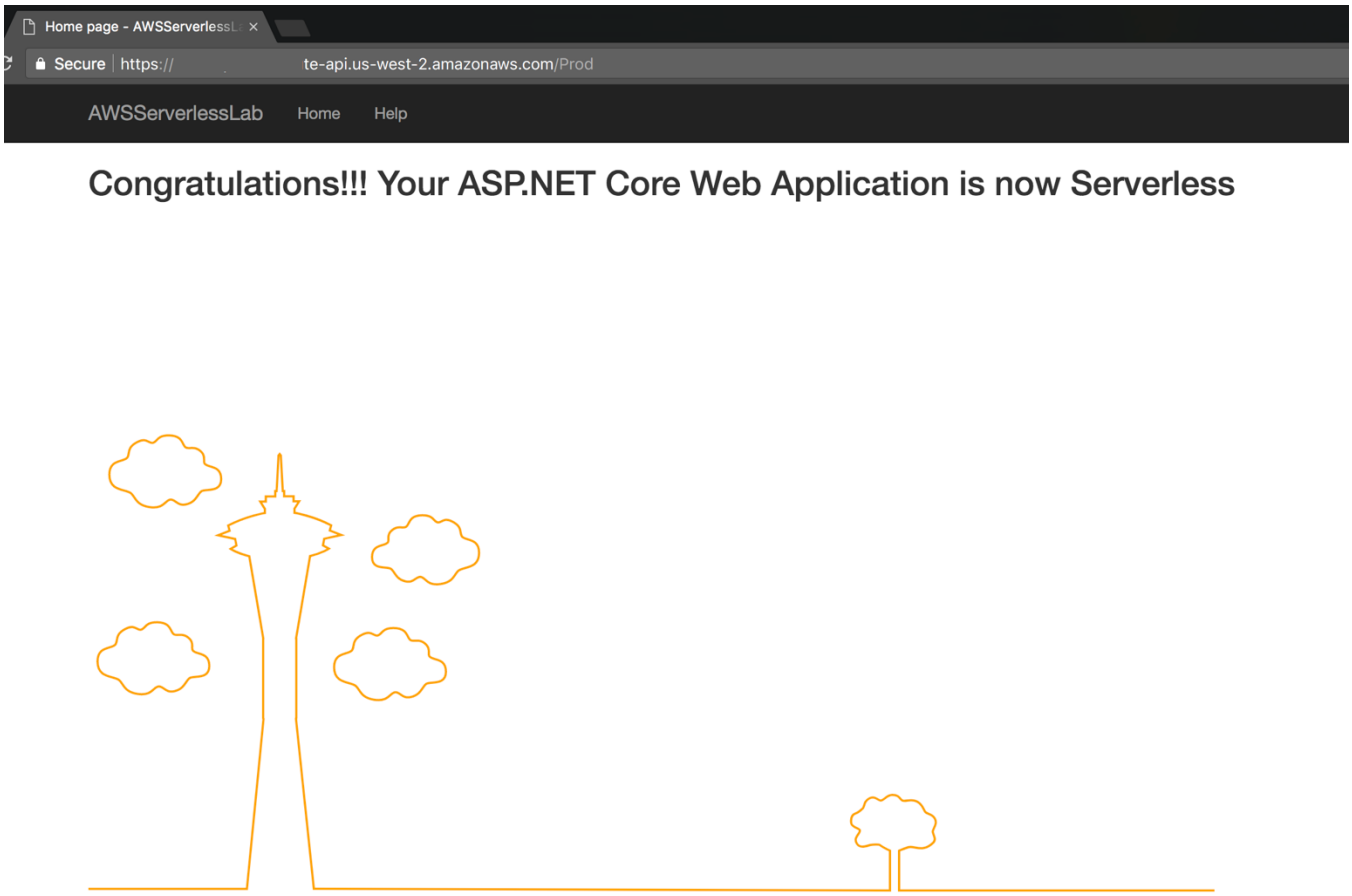


2. Ensure the "Account profile to use" drop-down and "Region" drop-down are set to the profile and region you are using for today's labs.
    a. For the CloudFormation Stack type **WebApp-Stack-Userid**.
    b. Click on New button for creating a new S3 bucket and named it as **WebApp-S3-Userid**.
    c. Click "Publish" and wait for the process to finish.

3. When the STATUS changes to **CREATE_COMPLETE** copy the AWS Serverless URL as you can see in the picture below.



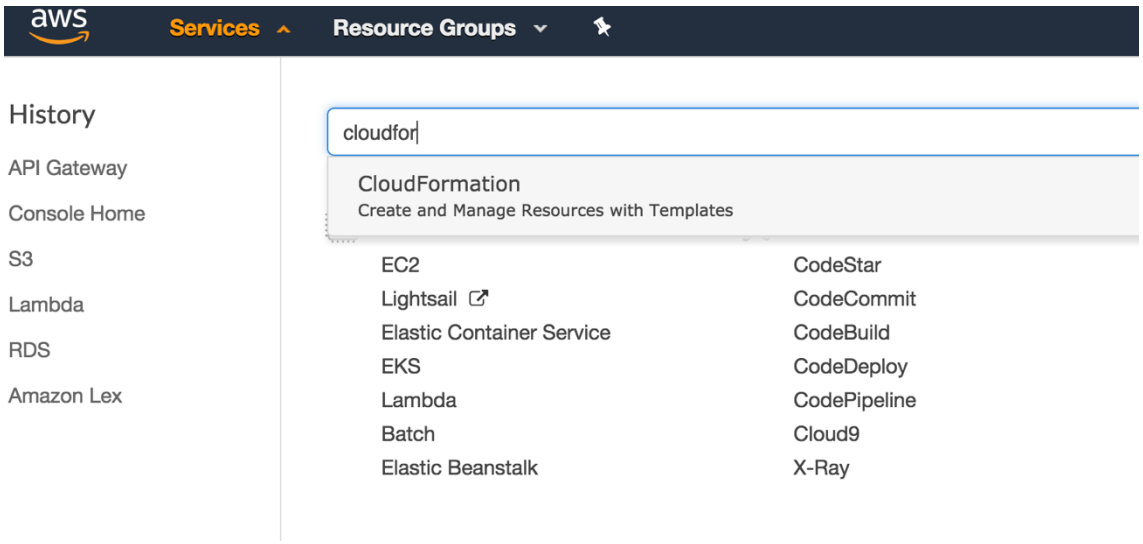
4. Paste the URL copied on the previous step on a browser to see the following result:



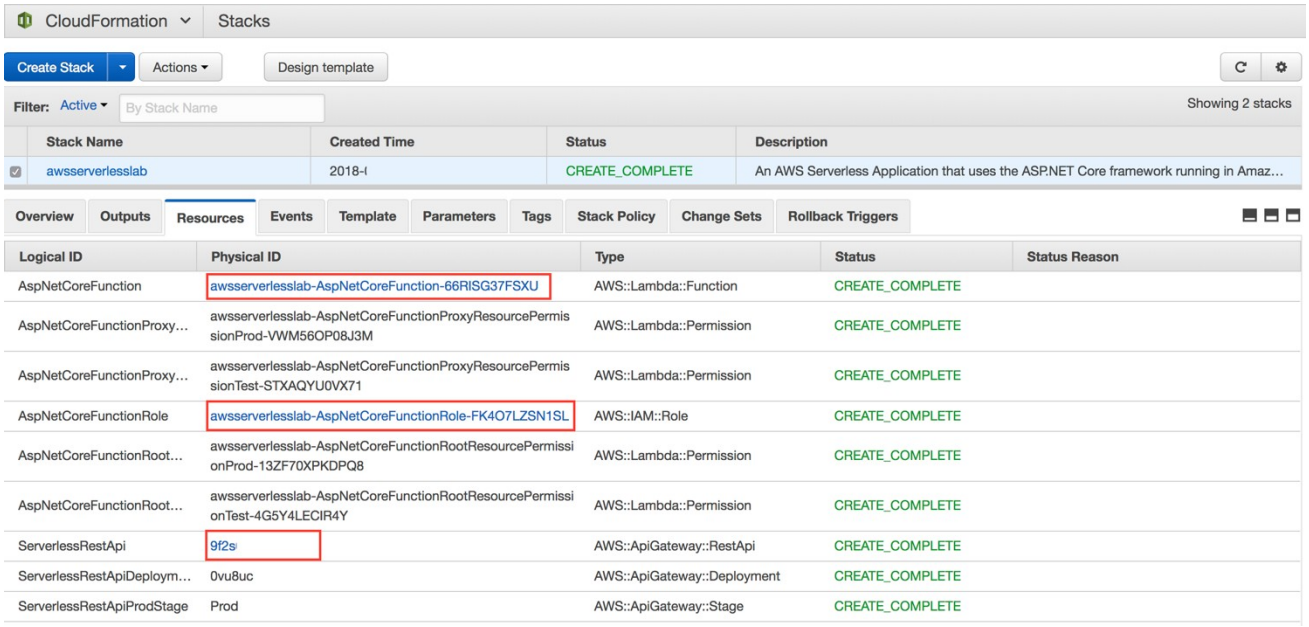
Part 3 – Check the AWS resources created.

For this lab, we'll access the AWS Management Console to check the resources published by the Visual Studio using the CloudFormation SAM template.

1. In the AWS Management Console, select the region you have deployed the lambda function and type "CloudFormation" into the search box to navigate to the API Gateway console. See the figure below.



2. Select the **WebApp-Stack-Userid** and select the tab ***Resources*** to verify what resources were created by clicking on the links at the Physical ID. You will be redirected to the *Lambda Function*, *IAM* Role and *API Gateway* resources that were created by the CloudFormation Stack launched by the visual studio:



Part 4 – Change the .Net Razor and Re-deploy the lambda function.

With your ASP.NET Core Web deployed on lambda, you can start customizing it using .Net Razor pages.

For this lab, we'll change the *index.cshtml and index.cshtml.cs* to render information on the landing page.

1. Add the code below to the *index.cshtml* file, right after the *<h2>Congratulations!!! Your ASP.NET Core Web Application is now Serverless</h2>*

```
<h2> @Model.Message </h2>
```

2. Replace the *IndexModel* class at the *index.cshtml.cs* file with the following code:

```
public class IndexModel : PageModel
   {
      public string Message { get; private set; } = "PageModel in C#";

      public void OnGet()
      {
         Message += $" Server time is { DateTime.Now }";
      }
   }
```

3. Re-deploy the application following the steps of the Part 2 – Deploy to AWS Lambda. Wait until the STATUS changes to **UPDATE_COMPLETE**, and refresh or open the Wep App URL (The URL can be found at the *AWS Serverless URL* on the form).

Congratulations, you have created and deployed an ASP.NET Web application using AWS Lambda, API Gateway and the .net core Razor Pages.

Part 5 – Remove all the Resources deployed

Finally, lets remove all the resources created by the CloudFormation stack launched by Visual Studio.

1. In the AWS Management console, navigate to CloudFormation console.
2. On the CloudFormation console, click on the Stack Name picked for you .Net Web App Lambda.

3. Click on the *Actions* DropDownButton and select *Delete Stack*. Confirm that you want to delete it on the next dialog.