Published in Towards AWS

Aaqib Hafeez Follow
Apr 16, 2021 · 7 min read · Listen

Access Swagger UI in AWS Lambda via API Gateway — Deployed with the Serverless Framework

Access Swagger UI endpoint for a .NET Core Web API hosted as an AWS Lambda with an API Gateway (Infrastructure As Code)

Swagger UI — What, Why, and Who?

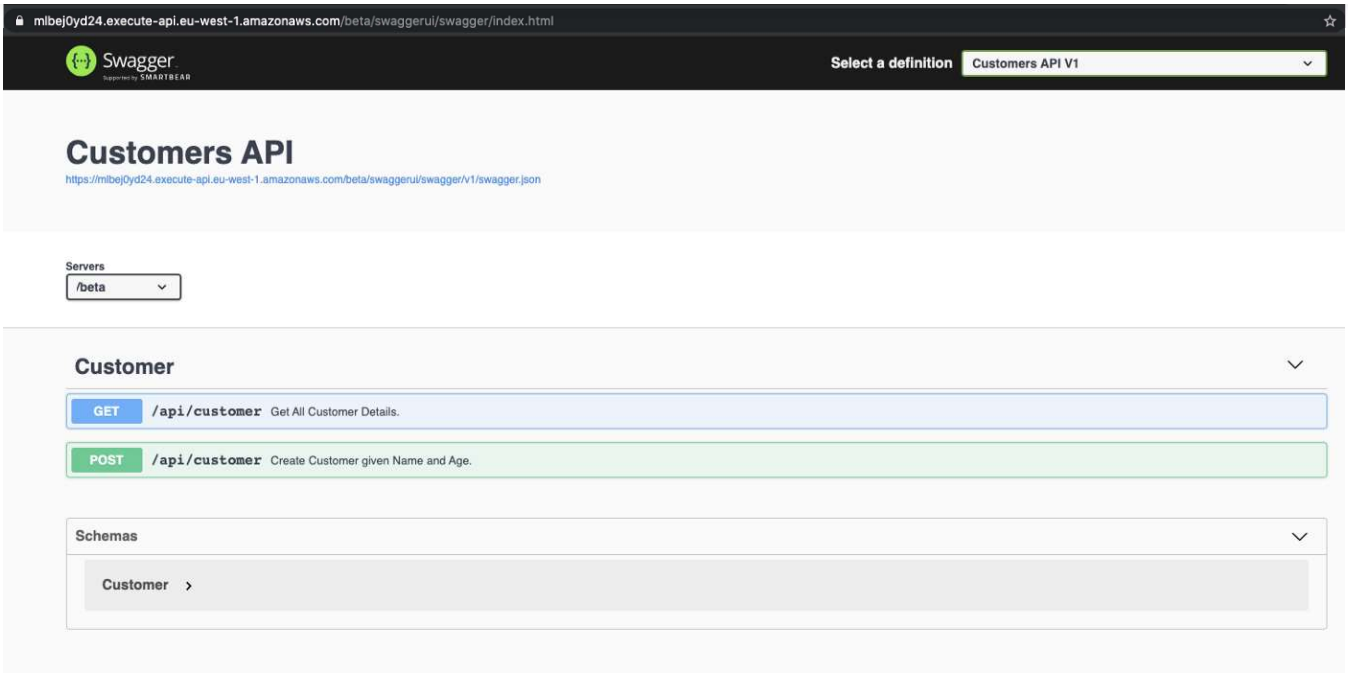
Well, why Swagger UI? Swagger UI gives users interactive API documentation and lets you invoke the API very easily by using the web interface without having to gather what the endpoint, method, request headers, body parameters, and so on actually are. You simply fill in the request parameters in the UI and off you go!— Invoke the UI!

Why?

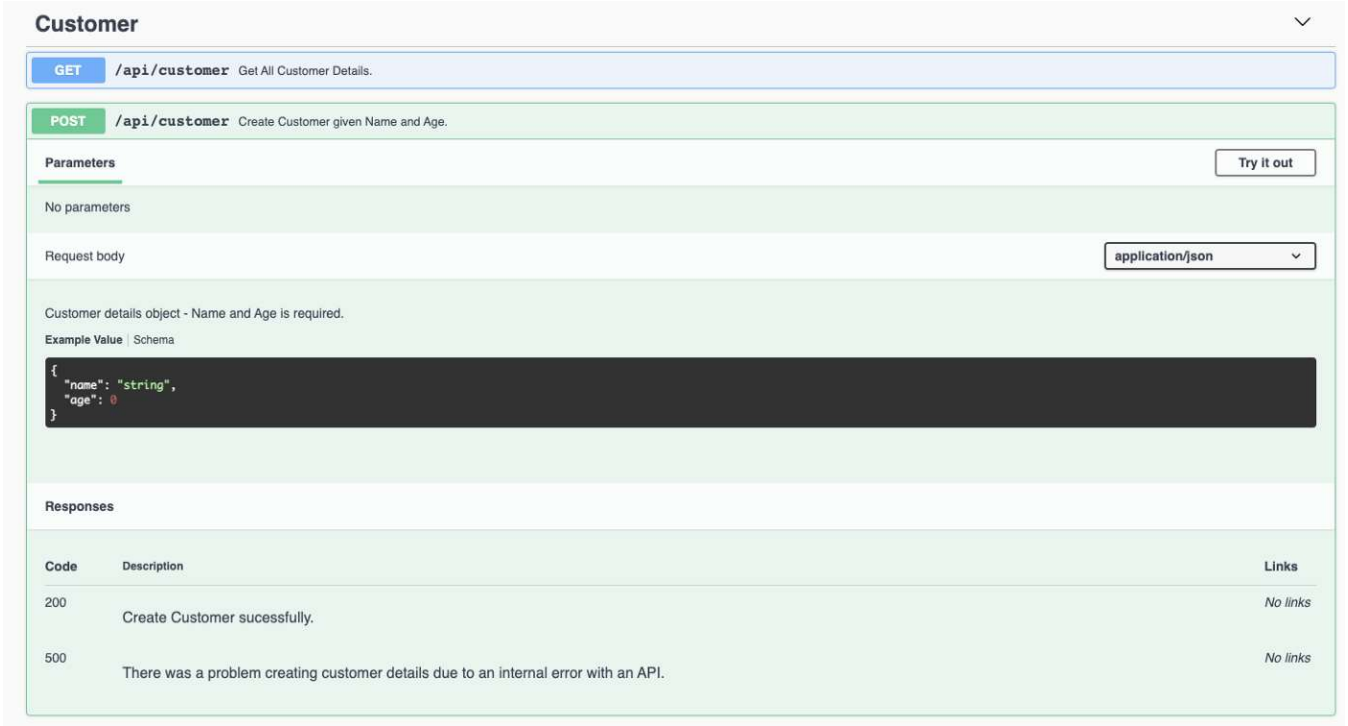
If you weren't using Swagger, then most likely you will have to contact the developers and ask them to give you all the information on how to access the API and this can be really time-consuming for both yourself and the developer. With Swagger UI this solves the problem as the developer could programmatically enable this functionality in code and keep the UI in sync with code changes so you don't need to ask the developer each time the endpoint is updated.

Who?

Who would use Swagger UI — The QA's, developers, stakeholders, or simply any client who needs to consume or test your API.



List of all Customer API endpoints that can be invoked via Swagger UI



Invoking the Create Customer endpoint via Swagger UI

Problem

There don't seem to be many solutions for accessing Swagger UI running on an AWS Lambda via API Gateway. One of the solutions that were fairly complicated required Amazon S3 where you would host the Swagger UI as static files. This was far too

What we needed was a much simpler solution — Simply documenting your C# code via XML comments and once the API has been deployed it would update the Swagger UI with the updated XML comments. Actually, this way was far easier to implement than expected :)

To achieve this, we had to use Swashbuckle and Serverless Framework (Built on top of CloudFormation and makes it easier to deploy infrastructure code to AWS) — Please visit my Github page for the solution.

However, there is a much simpler and easier solution involving Swashbuckle and API Gateway.

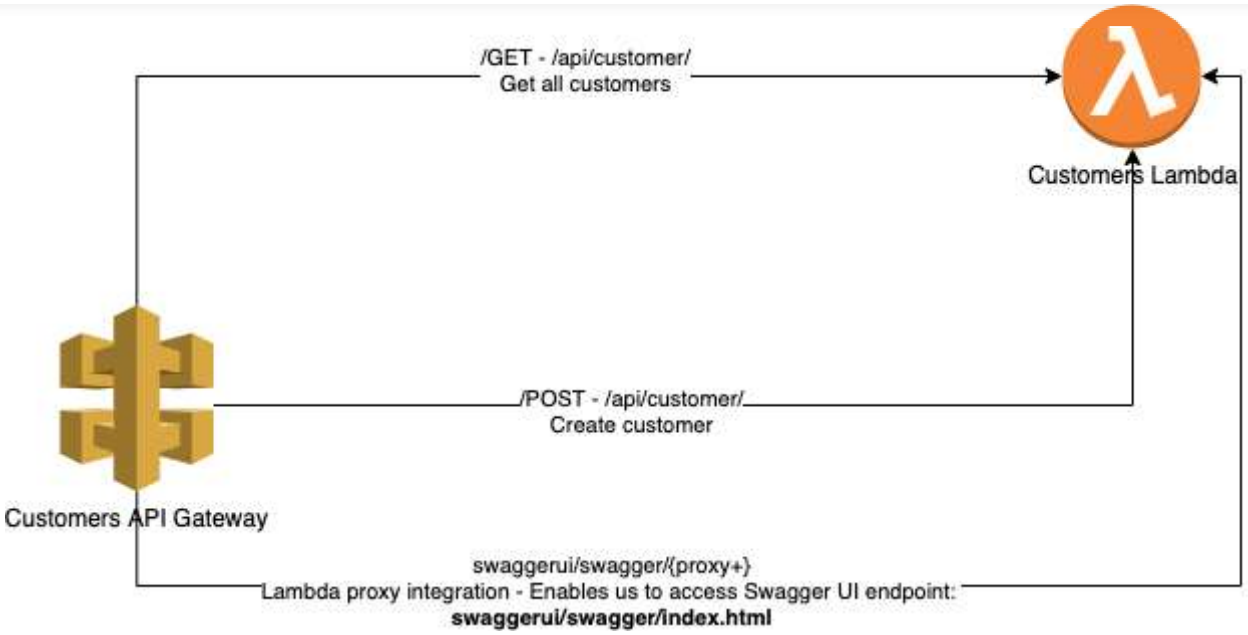
Technologies — What's involved?

- .NET Core 3.1 Web API (AWS Lambda)
- REST API on API Gateway
- AWS Serverless Framework (Abstraction layer in front of AWS CloudFormation — Makes it easier to write serverless application via infrastructure as code) — Creates the AWS Lambda Function and REST API in API Gateway. It also creates the endpoints on API Gateway so we can access the Swagger UI running in AWS Lambda.

Architecture

AWS Serverless Framework has been used to create serverless applications.

An API Gateway has been created which forward requests to the Lambda Function. The **key here** for Swagger is: Lambda proxy integration is used to forward any Swagger UI requests to the Lambda.



API Architecture

Implementation

In this example, we will create a .NET Core 3.1 Web API Application and create the Serverless Application using AWS Serverless Framework.

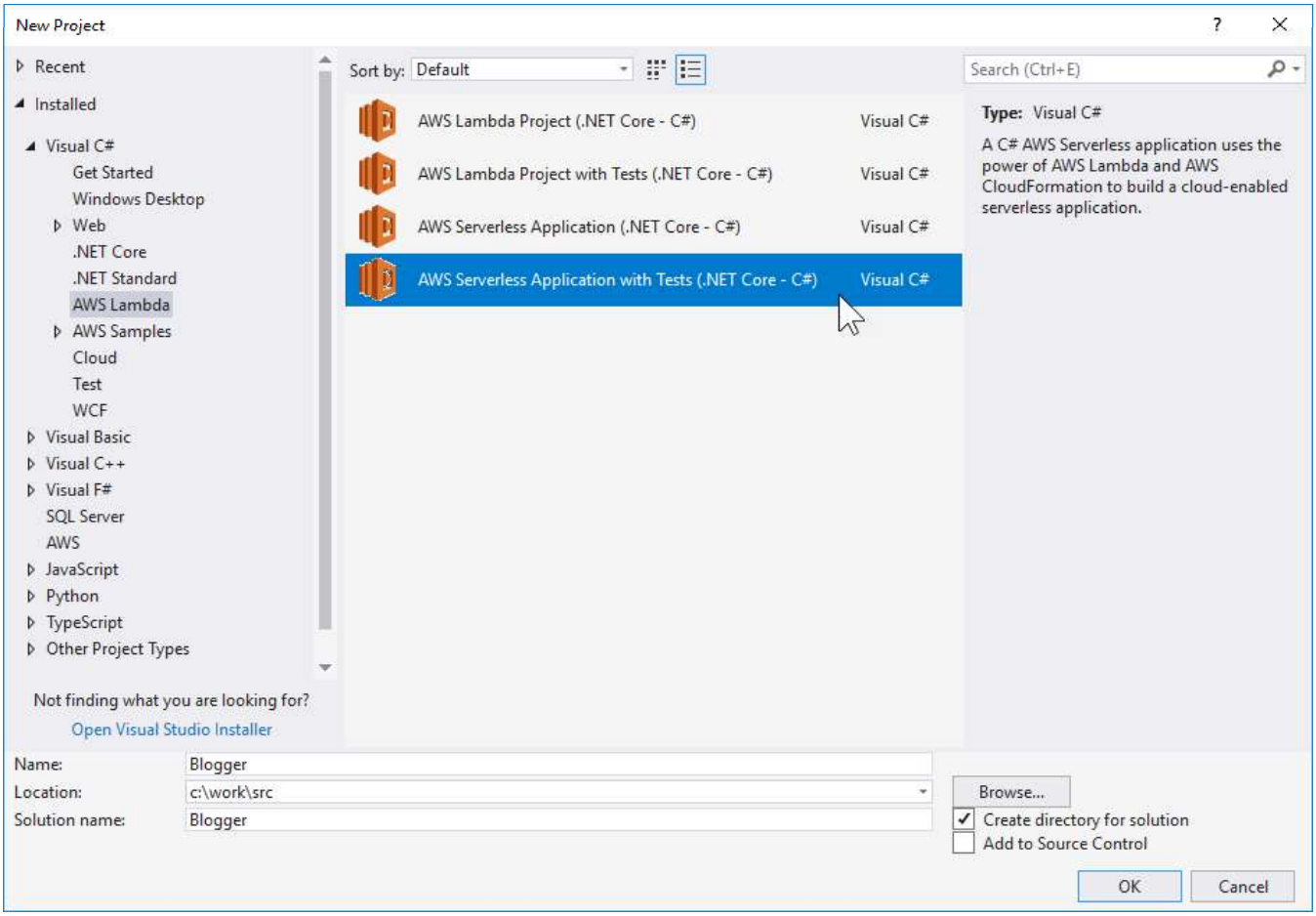
Swagger UI will be added to the .NET Core application using Swashbuckle. This is configured in the middleware of the application (Startup.cs) — see below.

This serverless file (infrastructure code) will create an API Gateway and an AWS Lambda. In the serverless file, we will set up a Lambda proxy integration in API Gateway which will forward any requests to the Swagger UI endpoint which lives in our Lambda allowing us to access Swagger UI.

There are two stages in implementing this solution:

Create .NET Core 3.1 Serverless Application — Integrate Swagger UI.

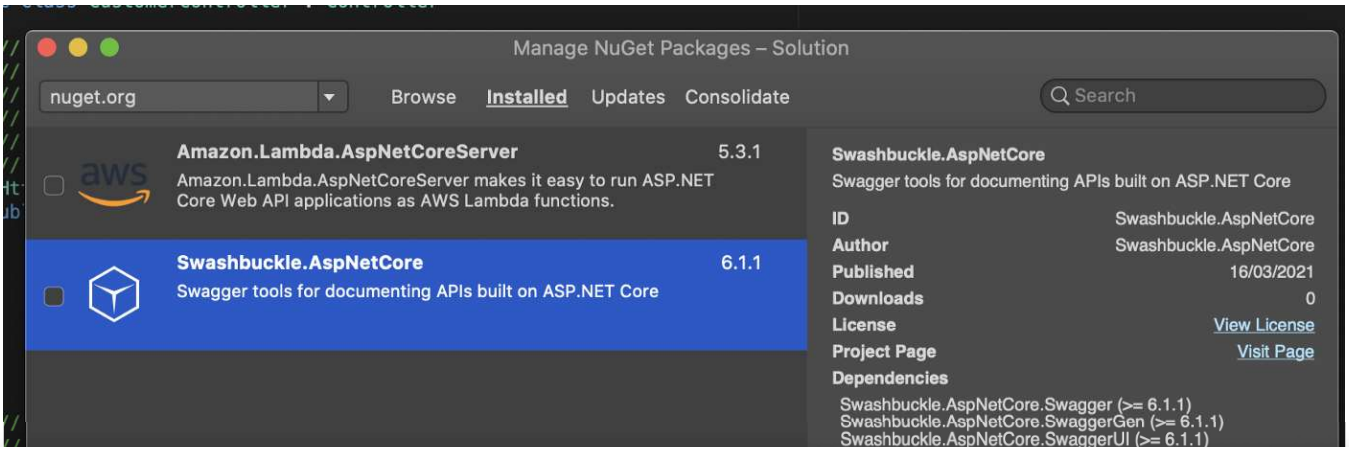
1. Create a new project in Visual Studio using the AWS Serverless Template. You will need to install **Lambda Tools Extension** for Visual Studio — You can do this via the Nuget Package Manager extension. Or you can do this via Command-Line if you are using Visual Studio on Mac OSX.



Visual Studio AWS Lambda Tools Extension Templates

2. Once the project has been created, install the 'Swashbuckle.AspNetCore' Nuget package via Nuget Package Manager in Visual Studio.

This guide will explain how to use Swashbuckle in a fairly simple way to enable Swagger UI for your API. For more information on how to use Swashbuckle — please use the official Microsoft guide.



Set GenerateDocumentationFile to true into the PropertyGroup section. You can suppress warnings for undocumented types — see Microsoft Guide.

```
<PropertyGroup>
<GenerateDocumentationFile>true</GenerateDocumentationFile>
</PropertyGroup>
```

4. In the Startup.cs file you will need to add and configure Swagger middleware.

a. In the ConfigureServices(…) method, call the AddSwaggerGen method like below:

```
1  public void ConfigureServices(IServiceCollection services)
2  {
3          services.AddControllers();
4
5          services.AddSwaggerGen(swagger =>
6          {
7                  swagger.SwaggerDoc("v1", new OpenApiInfo { Title = "Customers API" });
8                  var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
9                  var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
10                 swagger.IncludeXmlComments(xmlPath);
11             });
12         }
13 }
```

Startup.cs hosted with ❤ by GitHub view raw

ConfigureServices method implementation in Startup.cs

b. In the Configure(..) method, call the UseSwagger and UseSwaggerUI methods.

When calling UseSwagger ensure you set the RouteTemplate path to ‘swagger/{documentName}/swagger.json’ as this will be the path where Swagger UI will be exposed.

Also, a similar implementation applies for ‘useSwaggerUI ‘— We need to set the RoutePrefix to ‘swagger’ to reflect the Swagger endpoint.

```
        }

        app.UseHttpsRedirection();
        app.UseRouting();

        //Swagger
        app.UseSwagger(c =>
        {
            c.RouteTemplate = "swagger/{documentName}/swagger.json";
        });

        app.UseSwaggerUI(c =>
        {
            c.SwaggerEndpoint("v1/swagger.json", "Customers API V1");
            c.RoutePrefix = "swagger";
         });

         app.UseAuthorization();
         app.UseEndpoints(endpoints =>
         {
            endpoints.MapControllers();
            endpoints.MapGet("/", async context =>
            {
                await context.Response.WriteAsync("Welcome to running ASP.NET Core on AWS Lambda"
            });
          });
    }
```

5. Create a new controller. In this example, we will create a Customer Controller which will have a GET and POST endpoint.

Add the XML comments to all the endpoints in the Customer Controller class. The parameters and response code types are declared for each method which is how Swagger UI interprets the information and displays this in UI form.

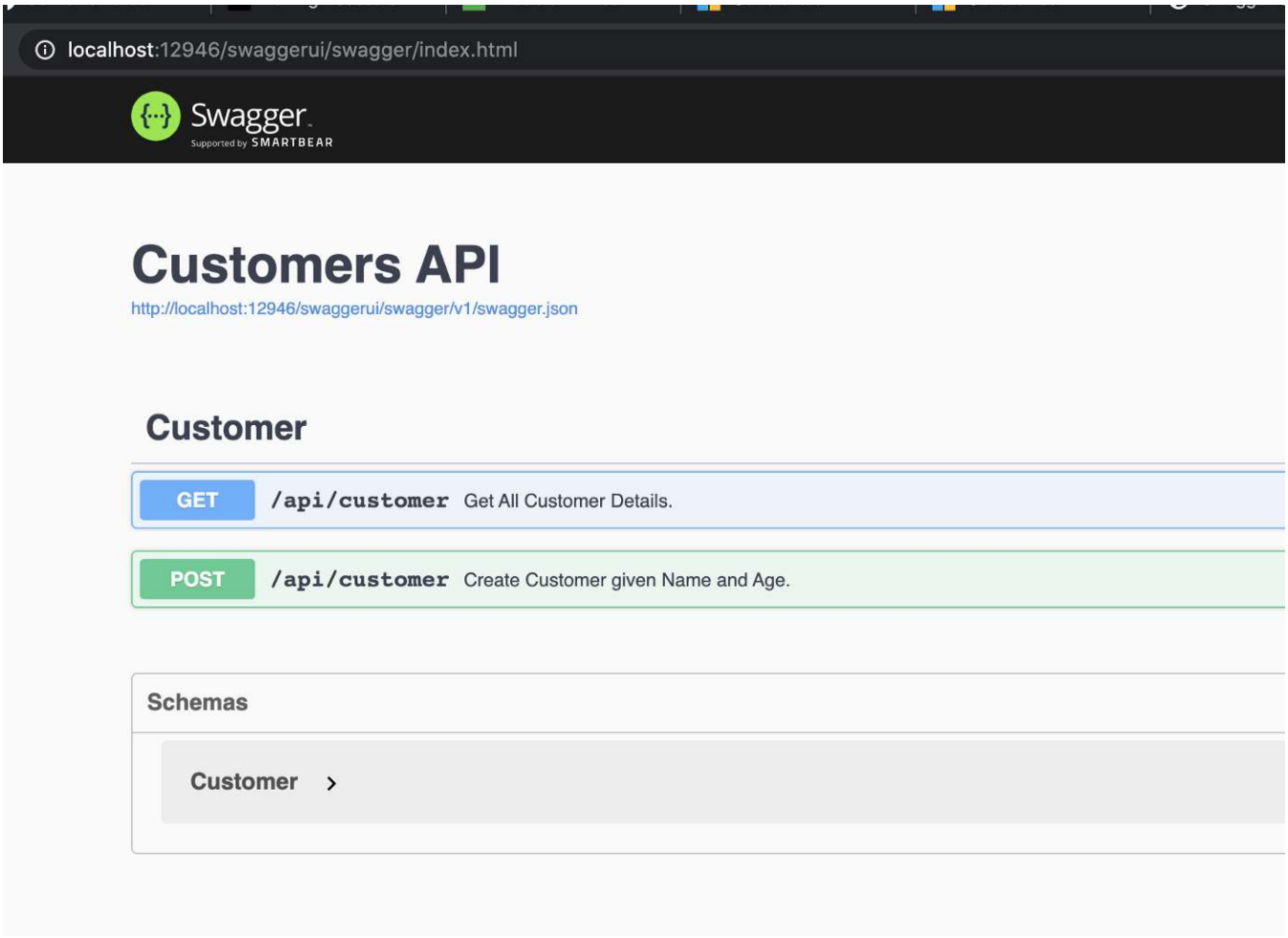
For example, for the GET method, we have specified 200 and 500 response codes with an explanation of what the response codes mean. For more information on this, please see the Microsoft guide.

```
using Microsoft.AspNetCore.Mvc;

// For more information on enabling MVC for empty projects, visit https://go.microsoft.com/fwli

namespace SwaggerAPILambda.Controllers
{
    [Produces("application/json")]
    [Route("api/customer")]
    public class CustomerController : Controller
    {
        /// <summary>
        /// Get All Customer Details.
        /// </summary>
        /// <returns> Return Customer Details</returns>
        /// <response code="200">Return Customer Details successfully.</response>
        /// <response code="500">There was a problem getting customer details due to an interna
        [HttpGet]
        public ActionResult<Customer> Get()
        {
            return new OkObjectResult(new Customer()
            {
                Name = "John",
                Age = 30
            });
        }

        /// <summary>
        /// Create Customer given Name and Age.
        /// </summary>
        /// <param name="customer">Customer details object - Name and Age is required.</param>
        /// <returns></returns>
        /// <response code="200">Create Customer sucessfully.</response>
        /// <response code="500">There was a problem creating customer details due to an interr
        [HttpPost]
        public ActionResult Create([FromBody] Customer customer)
        {
            return new OkObjectResult("Customer Created");
        }
    }

    /// <summary>
    /// Customer Details DTO.
    /// </summary>
```

```
50          [Required]
51          public string Name { get; set; }
52
53          /// <summary>
54          /// Customer Age
55          /// </summary>
56          [Required]
57          public int Age { get; set; }
58      }
59  }
```

Run project in Visual Studio and navigate to /swagger/index.html e.g. http://localhost:12946/swagger/index.html in this example.

Woohoo, you have Swagger UI running locally on your machine :)



Next steps, we need to get the API deployed onto AWS and access this via API Gateway.

In this stage, you will define all your serverless code in “serverless.yml” file. The serverless code will consist of creating the Lambda Function, API Gateway and implementing the Lambda Proxy integration for API Gateway that is required to access Swagger UI via API Gateway.

The customer resource endpoints have been defined in the serverless file — GET and POST. This will expose the endpoints in API Gateway and allow Swagger to invoke both endpoints.

Please note: API Gateway path is case sensitive so ensure your Route path in the Controller class matches the path defined in the serverless file (under HTTP → path) otherwise you may have trouble invoking your APIs from Swagger UI.

```
functions:
  orders:
    name: customers-lambda-${self:custom.stage}
    handler:
SwaggerAPILambda::SwaggerAPILambda.LambdaEntryPoint::FunctionHandler
Async
    environment:
       ASPNETCORE_ENVIRONMENT: ${self:custom.stage}
    events:
      - http:
          path: "api/customer"
          method: "get"
          cors: true
      - http:
          path: "api/customer"
          method: "post"
          cors: true
      - http:
          path: "swagger/{proxy+}"
          method: "get"
```

Accessing Swagger UI via API Gateway — Setup Lambda Proxy Integration in API Gateway

In the serverless file, you will also see a proxy path defined for “swagger/{proxy+}” (shown below) which allows any GET requests that fall under the path ‘swagger/’ to

```
- http:
    path: "swagger/{proxy+}"
    method: "get"
```

```
service:
  name: swaggerserverlesslambda

custom:
  stage: 'beta'
  environmentTag: ${opt:environmentTag, 'qa'}

provider:
  name: aws
  account: 'AWS_ACCOUNT_ID'
  endpointType: REGIONAL
  runtime: dotnetcore3.1
  memorySize: 512
  timeout: 30
  region: eu-west-1
  deploymentBucket:
    name: aaqib-serverless-deploys
    serverSideEncryption: AES256
  stage: ${self:custom.stage}
  iamRoleStatements:
    - Effect: "Allow"
      Action:
        - "logs:CreateLogGroup"
        - "logs:CreateLogStream"
        - "logs:PutLogEvents"
      Resource: "*"

package:
  include:
    - ./**

functions:
  orders:
    name: customers-lambda-${self:custom.stage}
    handler: SwaggerAPILambda::SwaggerAPILambda.LambdaEntryPoint::FunctionHandlerAsync
    environment:
      ASPNETCORE_ENVIRONMENT: ${self:custom.stage}
```

```
43        - http:
44            path: "api/customer"
45            method: "post"
46            cors: true
47        - http:
48            path: "swagger/{proxy+}"
49            method: "get"
```

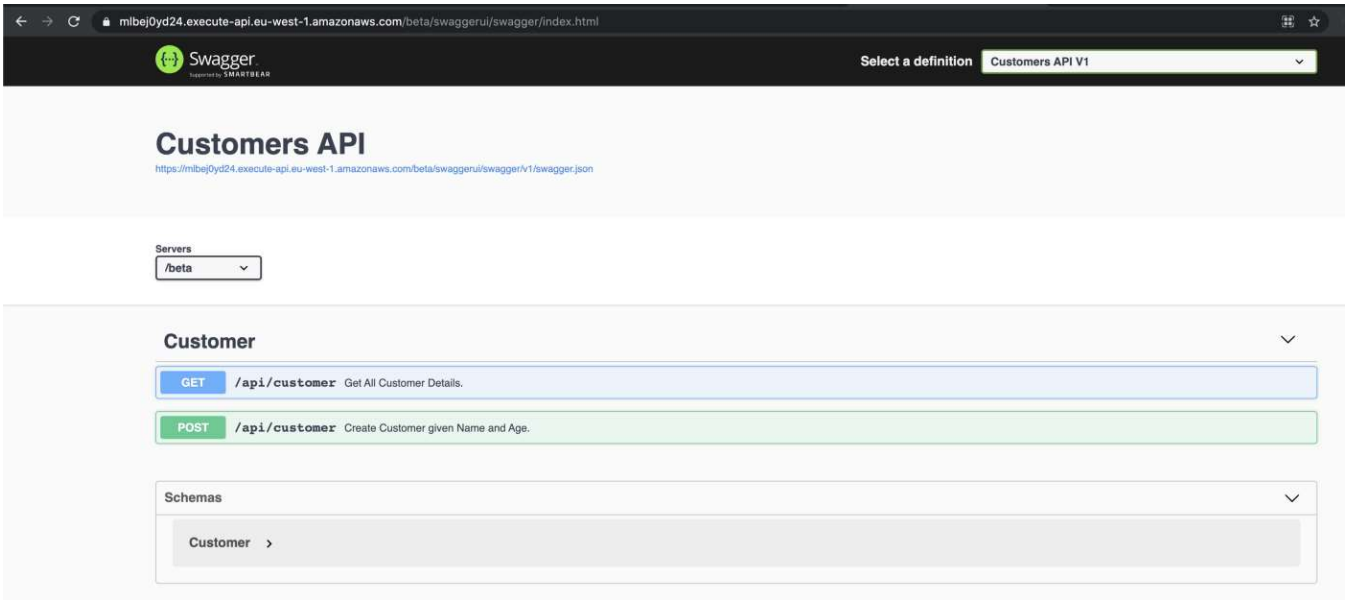
Please note: You will create to create an AWS S3 deployment bucket for your serverless code.

Deploying Serverless Application To AWS

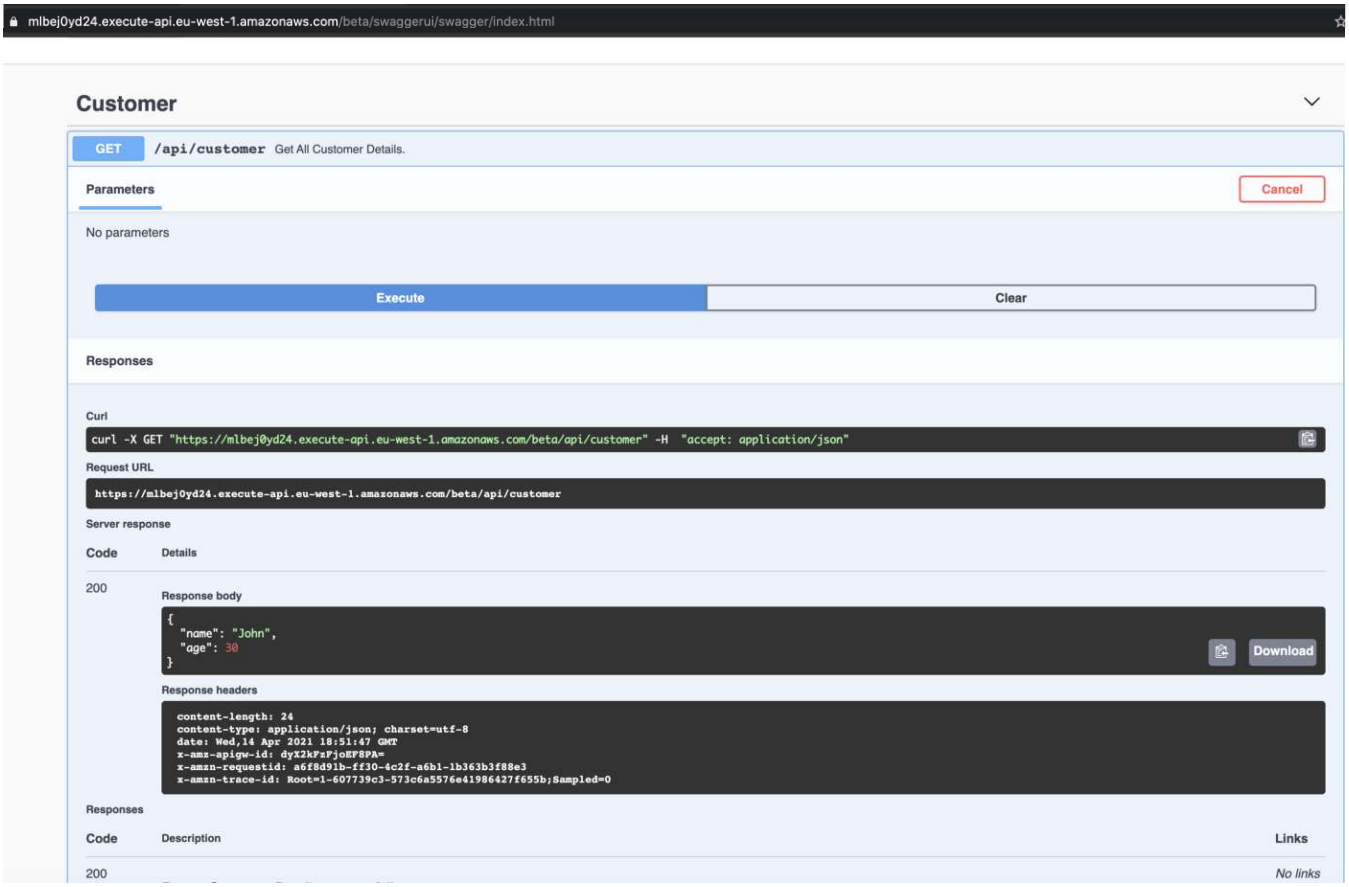
We are building and deploying the application using Bitbucket Pipelines but you can do this with any pipeline of your choice e.g. CodePipelines. Please see the sample code as an example of how Bitbucket Pipelines are used to deploy serverless code.

Once it has been deployed you should be able to access it by: https://{api-id}.execute-api.{region}.amazonaws.com/{stage}/swagger/index.html

For this example, this is the URL hosted on AWS: https://mlbej0yd24.execute-api.eu-west-1.amazonaws.com/beta/swagger/index.html



Swagger UI for AWS Lambda hosted on AWS



Invoking the GET All Customer Details endpoint using Swagger UI

That's it! You now have a Swagger UI endpoint which you can access via AWS API Gateway.

Sample Code

Please visit my Github repository for sample code.

References

- Get started SwashBuckle and ASP .NET Core — https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-5.0&tabs=visual-studio

Feel free to ask any questions or problems you may have :)

Shoutout to Ben Sloan for his advice on achieving this solution.