# Step-by-Step Tutorial: Developing an AWS Lambda Function using .NET

This guide will walk you through creating, deploying, and testing a simple "Hello World" AWS Lambda function using .NET 8 and Visual Studio.

**Prerequisites**

1. **Visual Studio 2022** installed on your machine.
2. **AWS Toolkit for Visual Studio** installed (can be added via the Visual Studio Installer).
3. An **AWS account** and IAM user credentials with appropriate permissions (programmatic access enabled) configured in the AWS Toolkit profile.

**Step 1: Create a New AWS Lambda Project in Visual Studio**

1. Open Visual Studio and click **"Create a new project"**.
2. In the search bar, type **"AWS Lambda"**. From the list of templates, select AWS Lambda Project (.NET Core - C#) and click **Next**.

A screenshot of a project

AI-generated content may be incorrect.

1. **Configure your new project**:
   * **Project name:** Enter FirstLambda.
   * **Solution name:** It will auto-populate as LambdaDemo. You can choose to place the solution and project in the same directory.
   * Click **Create**.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 2: Choose a Lambda Blueprint**

1. The **"New AWS Lambda C# Project"** wizard will open. This is where you select a blueprint—a pre-configured code template for your function.
2. For this tutorial, select the Empty Function blueprint. This gives you a minimal, clean starting point.
3. Click **Finish**.

A screenshot of a computer

AI-generated content may be incorrect.

*Alternatively, you can create this from the command line as shown in one of screenshots:*

bash

dotnet new Lambda.EmptyFunction --name TestLambda1

**Step 3: Examine the Generated Code**

Visual Studio will generate a project with a Function.cs file. This contains the essential handler method:

csharp

using Amazon.Lambda.Core;

[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace FirstLambda;

public class Function

{

public string FunctionHandler(string input, ILambdaContext context)

{

return input?.ToUpper();

}

}

* FunctionHandler: This is the entry point that AWS Lambda calls when your function is invoked.
* input: The input data passed to the function (e.g., from a test event).
* ILambdaContext: Provides information about the invocation, function, and execution environment.
* The method simply takes the input string and returns it in uppercase.

**Step 4: Deploy the Function to AWS Lambda using Visual Studio**

1. In **Solution Explorer**, right-click on your project (FirstLambda) and select **Publish to AWS Lambda...**.

A screenshot of a computer

AI-generated content may be incorrect.

1. The **"Upload Lambda Function"** window will appear. Fill in the details:
   * **AWS Credentials:** Select your configured AWS profile (e.g., superuser).
   * **Region:** Select your desired AWS region (e.g., Asia Pacific (Mumbai)).
   * **Function Name:** Select **"Create new function"** and enter firstLambda.
   * **Handler:** This should already be correctly populated as FirstLambda::FirstLambda.Function::FunctionHandler. This tells AWS Lambda which method to execute.
   * Click **Next**.

A screenshot of a computer

AI-generated content may be incorrect.

1. On the **"Advanced Function Details"** screen, you can configure permissions. For this simple function, the default AWSLambdaBasicExecutionRole is sufficient, as it only needs permission to write logs to Amazon CloudWatch.
2. Click **Upload**.

A screenshot of a computer

AI-generated content may be incorrect.

1. Visual Studio will now build the project, package it into a ZIP file, and create the function in your AWS account. You can see the progress in the **Output** window.

A computer screen shot of a computer screen

AI-generated content may be incorrect.

**Step 5: Test the Function in the AWS Console**

1. Once deployment is successful, the **AWS Lambda - Publish** window in Visual Studio will show your function's status and a **"Test Function"** button.
2. Click **Test Function**. This will open a dialog to configure a test event.
3. Enter a simple JSON string like "codesizzler" and click **Invoke**.

A screenshot of a computer

AI-generated content may be incorrect.

1. You will see the execution results:
   * The **Response** will show "CODESIZZLER" (the input converted to uppercase).
   * The **Summary** will show details like duration, billed duration, and memory used.
   * The **Log output** will show a link to the full logs in Amazon CloudWatch.

A screenshot of a computer

AI-generated content may be incorrect.

1. You can also go to the **AWS Management Console**, navigate to the **Lambda service**, and see your deployed function listed there.

A screenshot of a computer

AI-generated content may be incorrect.

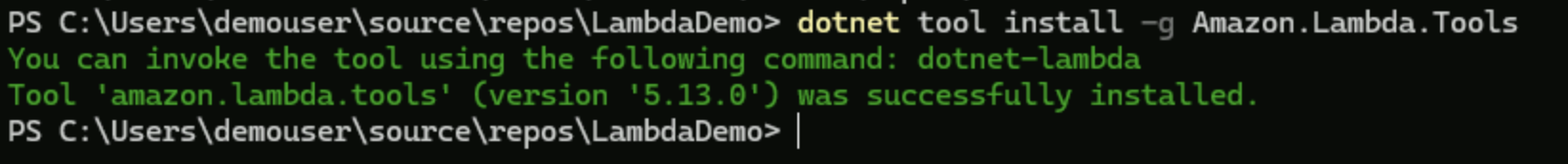
**Step 6: (Optional) Deploy using the .NET CLI Tools**

The screenshots show you also used the command line. This is a powerful alternative.

1. **Install the AWS Lambda Global Tool:**

bash

dotnet tool install -g Amazon.Lambda.Tools



1. **Navigate to your project folder** and deploy the function:

bash

cd C:\Users\demouser\source\repos\LambdaDemo\FirstLambda

dotnet lambda deploy-function

The tool will ask for the function name and other settings. Since you deployed already, it will detect the existing aws-lambda-tools-defaults.json file and **update** the existing function.

A screenshot of a computer screen

AI-generated content may be incorrect.

**Step 7: (Optional) Create a Public Function URL**

You can invoke your function via HTTPS without using the AWS API.

1. In the **AWS Lambda Console**, go to your function's **Configuration** tab.
2. Select **Function URL** and click **Create function URL**.
3. Choose an **Auth type**:
   * AWS\_IAM: Secure. Only authenticated IAM users/roles can call it.
   * NONE: Public. Anyone with the URL can invoke it. A resource-based policy is auto-created to allow public access.
4. Click **Save**. You will get a unique URL endpoint.
5. You can now test your function using curl or a web browser:

bash

curl "https://your-generated-url.lambda-url.ap-south-1.on.aws/"

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 8: Monitor Your Function**

1. AWS Lambda automatically logs all function invocations to **Amazon CloudWatch**.
2. In the AWS Console, navigate to **CloudWatch > Log groups > /aws/lambda/firstLambda**.
3. Here you can see all log streams, find specific invocations, and debug any issues.

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion**

You have successfully:

1. Created a .NET 8 Lambda function from an Visual Studio template.
2. Deployed it to AWS using both the Visual Studio GUI and the .NET CLI.
3. Tested the function in the AWS Console.
4. (Optionally) created a public URL for it and viewed its logs.

This foundational workflow is the first step toward building complex serverless applications on AWS using .NET. You can now modify the FunctionHandler to perform any task you need, such as processing data from Amazon S3, API Gateway, or DynamoDB.