.NET Core CLI

PDF (lambda-dg.pdf#csharp-package-cli)
Kindle (https://www.amazon.com/dp/B07GFJLN6D)
RSS (lambda-updates.rss)

The .NET Core CLI offers a cross-platform way for you to create .NET-based Lambda applications. This section assumes that you have installed the .NET Core CLI. If you haven't, see Download .NET (https://dotnet.microsoft.com/download) on the Microsoft website.

In the .NET CLI, you use the `new` command to create .NET projects from a command line. This is useful if you want to create a project outside of Visual Studio. To view a list of the available project types, open a command line and navigate to where you installed the .NET Core runtime and run the following command:

```
dotnet new -all
Usage: new [options]
...
Templates                                         Short Name         Language          Tags
--------------------------------------------------------------------------------------------
------------------------------
Console Application                               console            [C#], F#, VB
Common/Console
Class library                                     classlib           [C#], F#, VB
Common/Library
Unit Test Project                                 mstest             [C#], F#, VB
Test/MSTest
xUnit Test Project                                xunit              [C#], F#, VB
Test/xUnit
...
Examples:
    dotnet new mvc --auth Individual
    dotnet new viewstart
    dotnet new --help
```

Lambda offers additional templates via the Amazon.Lambda.Templates (https://www.nuget.org/packages/Amazon.Lambda.Templates) nuget package. To install this package, run the following command:

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

Once the install is complete, the Lambda templates show up as part of `dotnet new`. To examine details about a template, use the `help` option.

```
dotnet new lambda.EmptyFunction --help
```

The `lambda.EmptyFunction` template supports the following options:

- `--name` – The name of the function
- `--profile` – The name of a profile in your AWS SDK for .NET credentials file (https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/net-dg-config-creds.html)
- `--region` – The AWS Region to create the function in

These options are saved to a file named `aws-lambda-tools-defaults.json`.

Create a function project with the `lambda.EmptyFunction` template.

```
dotnet new lambda.EmptyFunction --name MyFunction
```

Under the `src/myfunction` directory, examine the following files:

- **aws-lambda-tools-defaults.json**: This is where you specify the command line options when deploying your Lambda function. For example:

  ```
  "profile" : "default",
  "region" : "us-east-2",
  "configuration" : "Release",
  "framework" : "netcoreapp2.1",
  "function-runtime":"dotnetcore3.1",
  "function-memory-size" : 256,
  "function-timeout" : 30,
  "function-handler" : "MyFunction::MyFunction.Function::FunctionHandler"
  ```

- **Function.cs**: Your Lambda handler function code. It's a C# template that includes the default `Amazon.Lambda.Core` library and a default `LambdaSerializer` attribute. For more information on serialization requirements and options, see Serializing Lambda functions (./csharp-handler.html#csharp-handler-serializer) . It also includes a sample function that you can edit to apply your Lambda function code.

  ```
  using System;
  using System.Collections.Generic;
  using System.Linq;
  using System.Threading.Tasks;

  using Amazon.Lambda.Core;

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

  namespace MyFunction
  {
      public class Function
      {

          public string FunctionHandler1(string input, ILambdaContext context)
          {
              return input?.ToUpper();
          }
      }
  }
  ```

- **MyFunction.csproj**: An MSBuild (https://msdn.microsoft.com/en-us/library/dd393574.aspx) file that lists the files and assemblies that comprise your application.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0 " />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.3.0" />
  </ItemGroup>

</Project>
```

- **Readme**: Use this file to document your Lambda function.

Under the `myfunction/test` directory, examine the following files:

- **myFunction.Tests.csproj**: As noted previously, this is an MSBuild (https://msdn.microsoft.com/en-us/library/dd393574.aspx) file that lists the files and assemblies that comprise your test project. Note also that it includes the `Amazon.Lambda.Core` library, allowing you to seamlessly integrate any Lambda templates required to test your function.

```
<Project Sdk="Microsoft.NET.Sdk">
   ...

    <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0 " />
   ...
```

- **FunctionTest.cs**: The same C# code template file that it is included in the `src` directory. Edit this file to mirror your function's production code and test it before uploading your Lambda function to a production environment.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

using Xunit;
using Amazon.Lambda.Core;
using Amazon.Lambda.TestUtilities;

using MyFunction;

namespace MyFunction.Tests
{
    public class FunctionTest
    {
        [Fact]
        public void TestToUpperFunction()
        {

            // Invoke the lambda function and confirm the string was upper cased.
```

```
            var function = new Function();
            var context = new TestLambdaContext();
            var upperCase = function.FunctionHandler("hello world", context);

            Assert.Equal("HELLO WORLD", upperCase);
        }
    }
}
```

Once your function has passed its tests, you can build and deploy using the Amazon.Lambda.Tools .NET Core Global Tool (http://aws.amazon.com/blogs/developer/net-core-global-tools-for-aws/) . To install the .NET Core Global Tool, run the following command:

```
dotnet tool install -g Amazon.Lambda.Tools
```

If you already have the tool installed, you can make sure that it is the latest version using the following command:

```
dotnet tool update -g Amazon.Lambda.Tools
```

For more information about the Amazon.Lambda.Tools .NET Core Global Tool, see the AWS Extensions for .NET CLI (https://github.com/aws/aws-extensions-for-dotnet-cli) repository on GitHub.

With the Amazon.Lambda.Tools installed, you can deploy your function using the following command:

```
dotnet lambda deploy-function MyFunction --function-role role
```

After deployment, you can re-test it in a production environment using the following command, and pass in a different value to your Lambda function handler:

```
dotnet lambda invoke-function MyFunction --payload "Just Checking If Everything is OK"
```

If everything is successful, you see the following:

```
dotnet lambda invoke-function MyFunction --payload "Just Checking If Everything is OK"
Payload:
"JUST CHECKING IF EVERYTHING IS OK"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id  Duration: 0.99 ms       Billed Duration: 1 ms         Memory Size: 256
MB     Max Memory Used: 12 MB
```