Building ASP.NET Core Web API using AWS Lambda

Objective

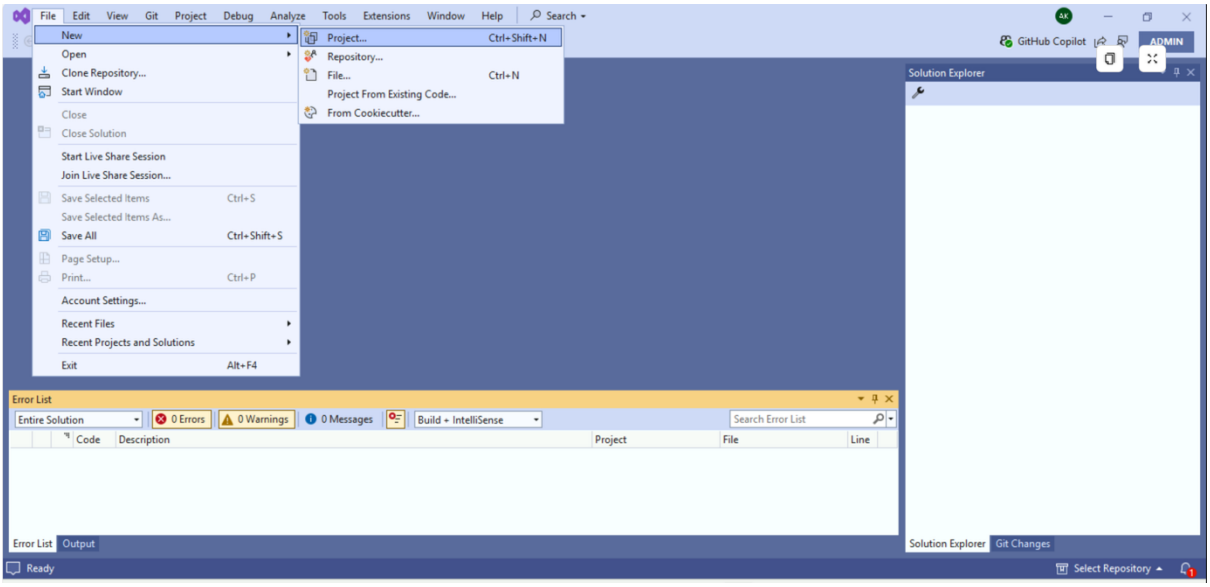
Deploy an ASP.NET Core Web API application to AWS Lambda using the serverless architecture pattern.

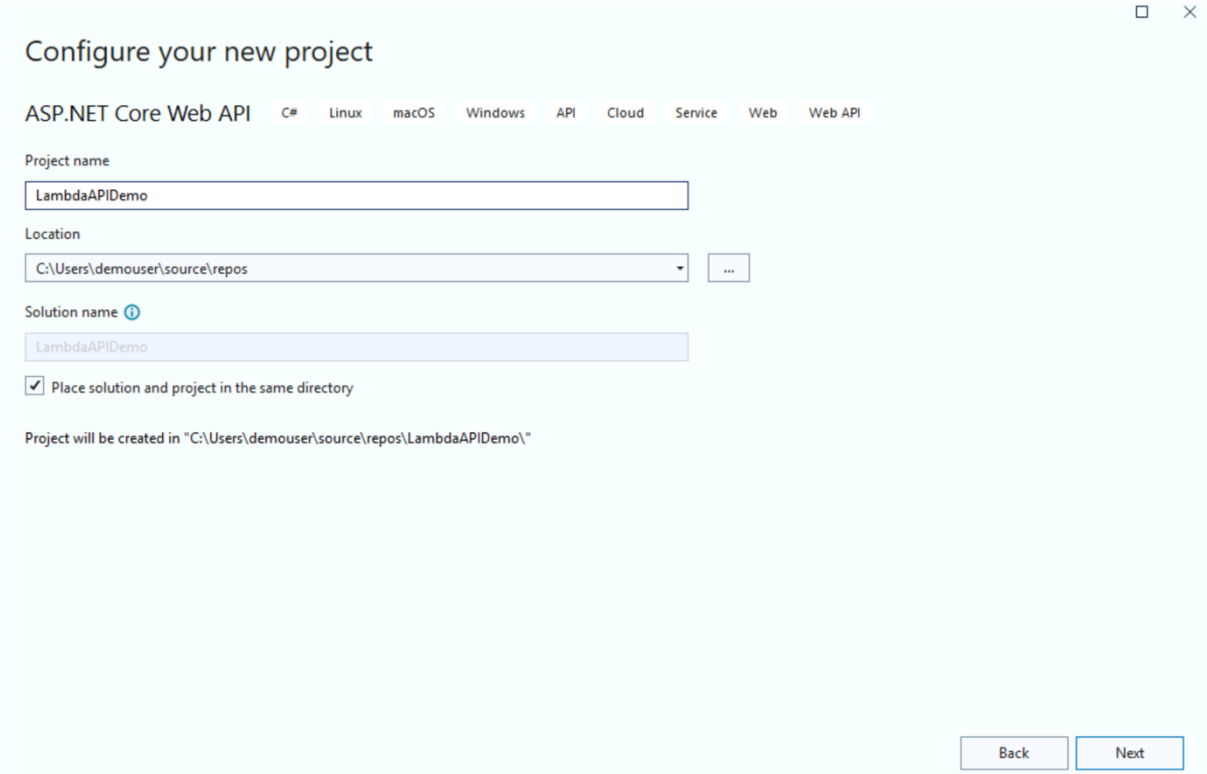
Pre-requisites

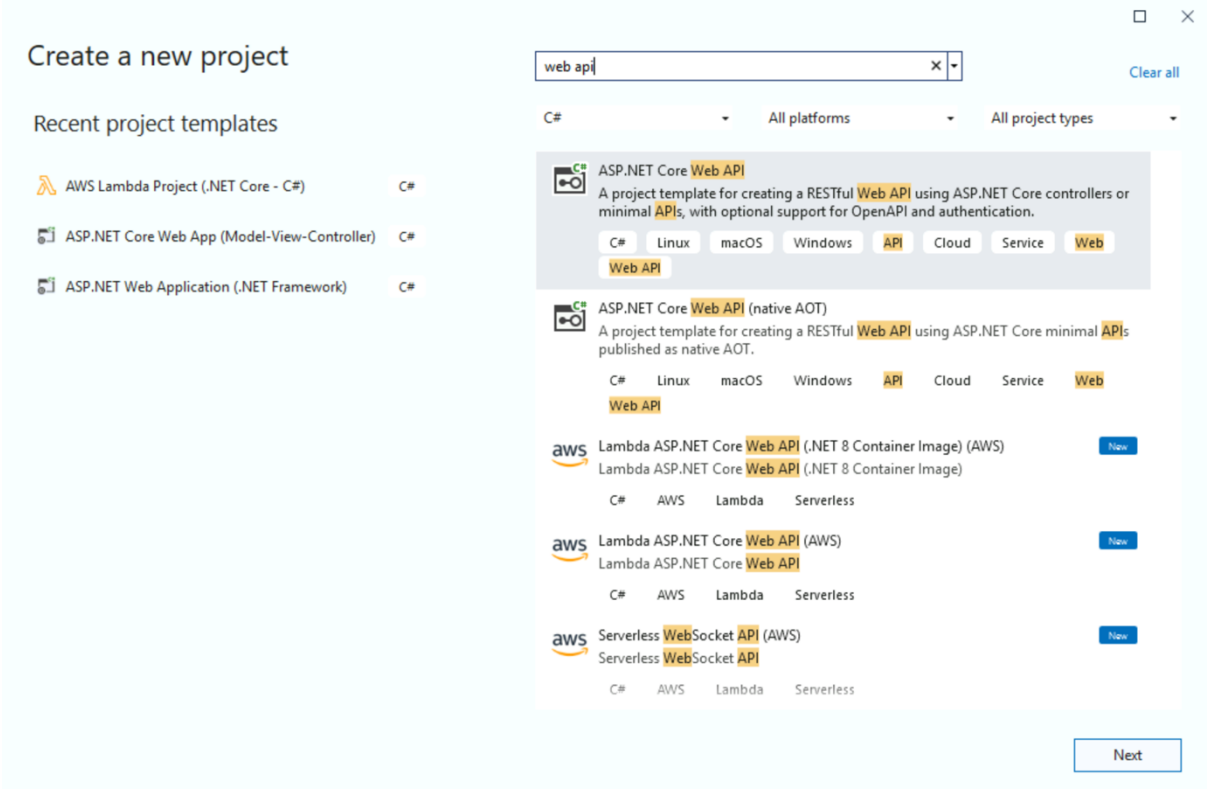
- An active AWS account
- Visual Studio
- AWS Toolkit for Visual Studio
- ASP.NET Core 6 or higher

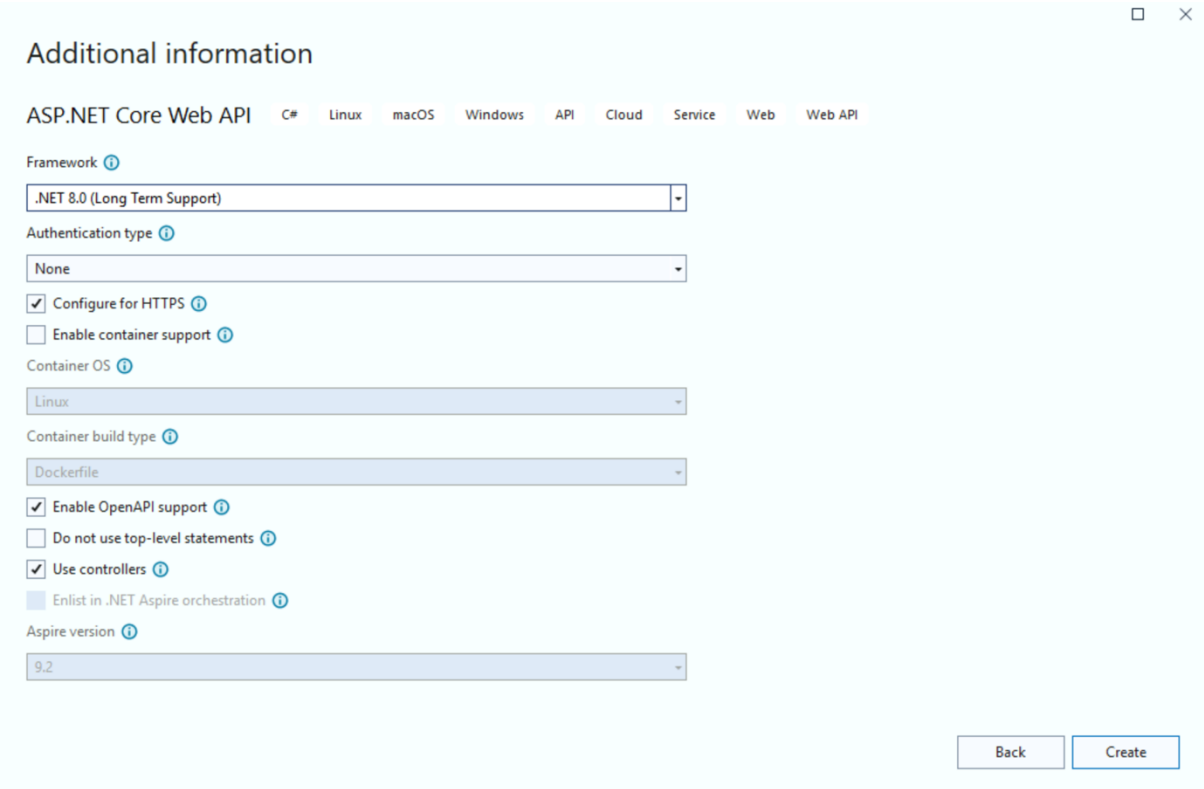
Step 1: Create ASP.NET Core Web API Project

1. Open Visual Studio and select **Create a new project**
2. Choose **ASP.NET** **Core Web API** template
3. Configure your project:
    - Project name: `YourProjectName`
    - Framework: .NET 6.0 or higher
    - Authentication: None
    - Configure for HTTPS: Enabled
    - Enable OpenAPI support: Enabled


File Edit View Git Project Debug Analyze Tools Extensions Window Help
Search
New
Open
Clone Repository...
Start Window
Close
Close Solution
Start Live Share Session
Join Live Share Session...
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Page Setup...
Print... Ctrl+P
Account Settings...
Recent Files
Recent Projects and Solutions
Exit Alt+F4
Project... Ctrl+Shift+N
Repository...
File... Ctrl+N
Project From Existing Code...
From Cookiecutter...
GitHub Copilot
ADMIN
Solution Explorer
Error List
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Search Error List
Code
Description
Project
File
Line
Error List
Output
Solution Explorer
Git Changes
Ready
Select Repository

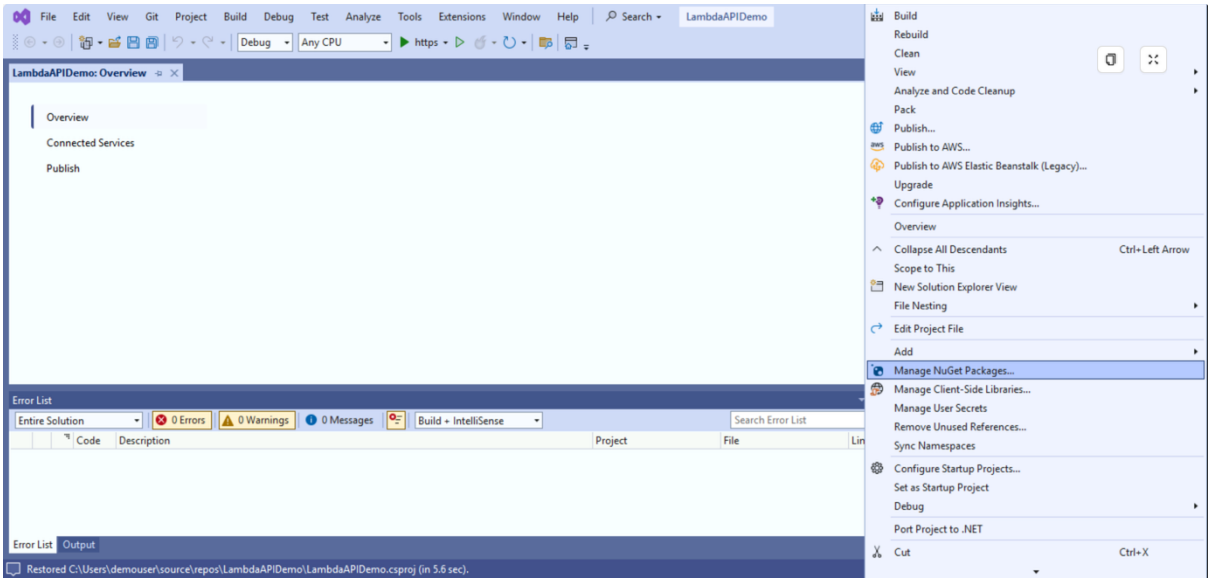

Configure your new project
ASP.NET Core Web API
C#
Linux
macOS
Windows
API
Cloud
Service
Web
Web API
Project name
LambdaAPIDemo
Location
C:\Users\demouser\source\repos
...
Solution name
LambdaAPIDemo
Place solution and project in the same directory
Project will be created in "C:\Users\demouser\source\repos\LambdaAPIDemo\"
Back
Next

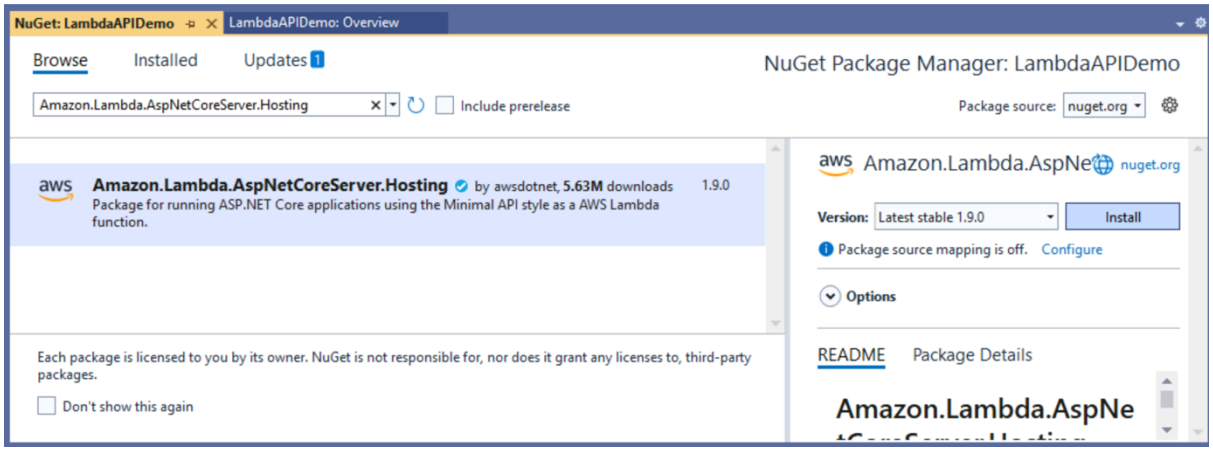

Create a new project
Recent project templates
AWS Lambda Project (.NET Core - C#)
C#
ASP.NET Core Web App (Model-View-Controller)
C#
ASP.NET Web Application (.NET Framework)
C#
web api
Clear all
C#
All platforms
All project types
ASP.NET Core Web API
A project template for creating a RESTful Web API using ASP.NET Core controllers or minimal APIs, with optional support for OpenAPI and authentication.
C# Linux macOS Windows API Cloud Service Web Web API
ASP.NET Core Web API (native AOT)
A project template for creating a RESTful Web API using ASP.NET Core minimal APIs published as native AOT.
C# Linux macOS Windows API Cloud Service Web Web API
Lambda ASP.NET Core Web API (.NET 8 Container Image) (AWS)
New
Lambda ASP.NET Core Web API (.NET 8 Container Image)
C# AWS Lambda Serverless
Lambda ASP.NET Core Web API (AWS)
New
Lambda ASP.NET Core Web API
C# AWS Lambda Serverless
Serverless WebSocket API (AWS)
New
Serverless WebSocket API
C# AWS Lambda Serverless
Next

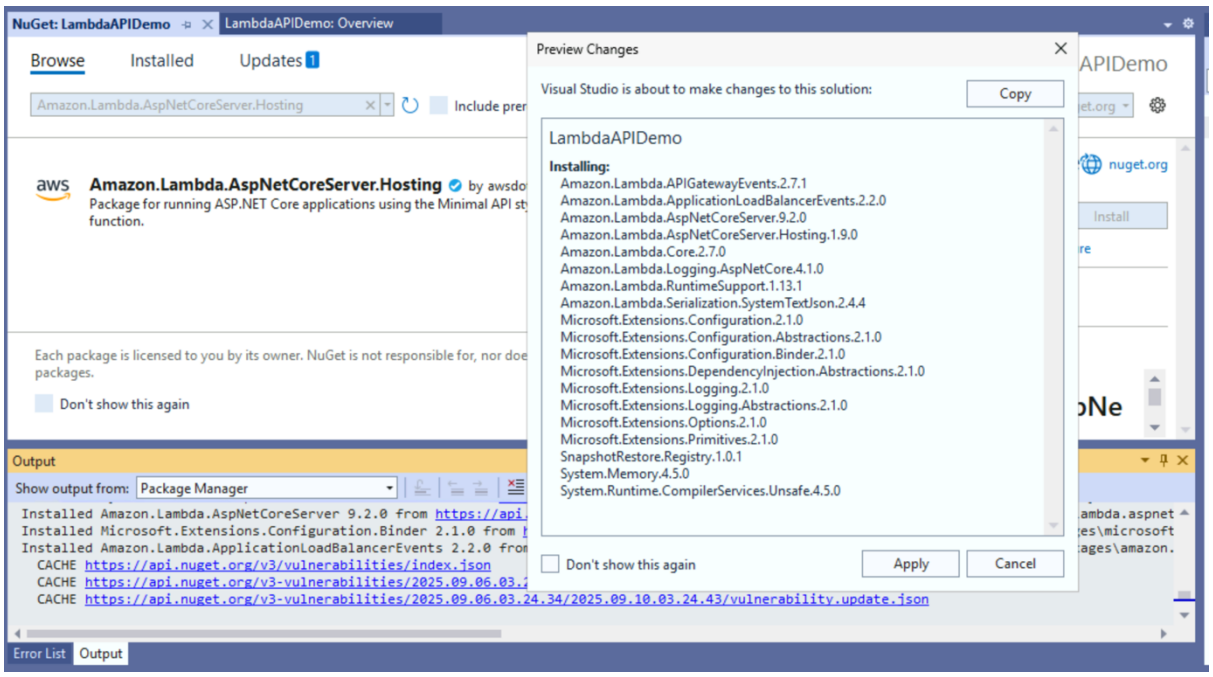


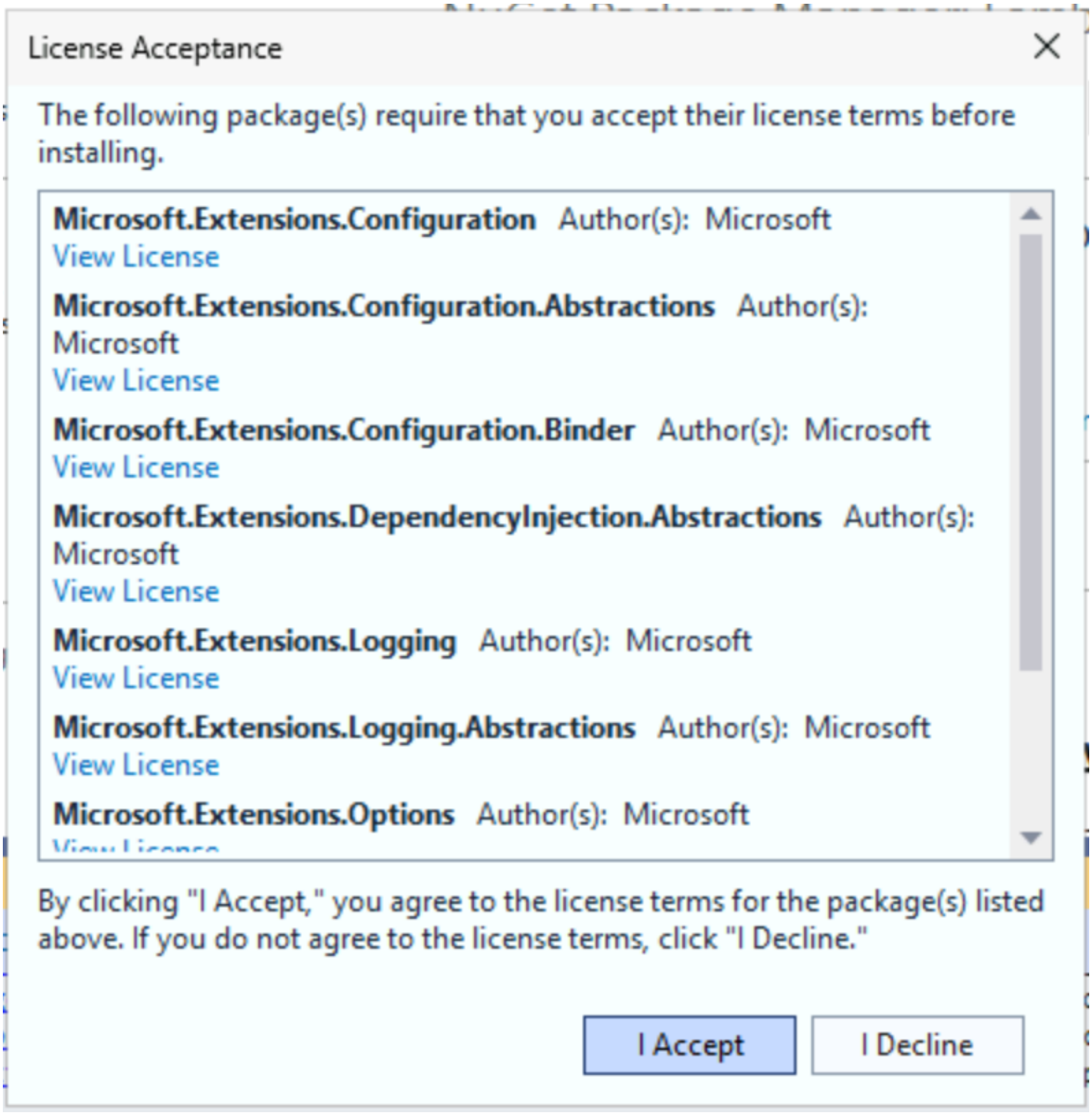
Step 2: Install Required NuGet Package

1. Right-click on your project in Solution Explorer
2. Select **Manage NuGet Packages**
3. Search for and install:
    - Amazon.Lambda.AspNetCoreServer.Hosting




NuGet: LambdaAPIDemo
LambdaAPIDemo: Overview
Browse
Installed
Updates 1
NuGet Package Manager: LambdaAPIDemo
Amazon.Lambda.AspNetCoreServer.Hosting
Include prerelease
Package source: nuget.org
Amazon.Lambda.AspNetCoreServer.Hosting by awsdotnet, 5.63M downloads
Package for running ASP.NET Core applications using the Minimal API style as a AWS Lambda function.
1.9.0
Amazon.Lambda.AspNe
nuget.org
Version: Latest stable 1.9.0
Install
Package source mapping is off. Configure
Options
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.
Don't show this again
README
Package Details
Amazon.Lambda.AspNe


NuGet: LambdaAPIDemo
LambdaAPIDemo: Overview
Browse
Installed
Updates 1
Preview Changes
Visual Studio is about to make changes to this solution:
Copy
LambdaAPIDemo
Installing:
Amazon.Lambda.APIGatewayEvents.2.7.1
Amazon.Lambda.ApplicationLoadBalancerEvents.2.2.0
Amazon.Lambda.AspNetCoreServer.9.2.0
Amazon.Lambda.AspNetCoreServer.Hosting.1.9.0
Amazon.Lambda.Core.2.7.0
Amazon.Lambda.Logging.AspNetCore.4.1.0
Amazon.Lambda.RuntimeSupport.1.13.1
Amazon.Lambda.Serialization.SystemTextJson.2.4.4
Microsoft.Extensions.Configuration.2.1.0
Microsoft.Extensions.Configuration.Abstractions.2.1.0
Microsoft.Extensions.Configuration.Binder.2.1.0
Microsoft.Extensions.DependencyInjection.Abstractions.2.1.0
Microsoft.Extensions.Logging.2.1.0
Microsoft.Extensions.Logging.Abstractions.2.1.0
Microsoft.Extensions.Options.2.1.0
Microsoft.Extensions.Primitives.2.1.0
SnapshotRestore.Registry.1.0.1
System.Memory.4.5.0
System.Runtime.CompilerServices.Unsafe.4.5.0
Don't show this again
Apply
Cancel
Output
Show output from: Package Manager
Installed Amazon.Lambda.AspNetCoreServer 9.2.0 from https://api
Installed Microsoft.Extensions.Configuration.Binder 2.1.0 from
Installed Amazon.Lambda.ApplicationLoadBalancerEvents 2.2.0 fro
CACHE https://api.nuget.org/v3/vulnerabilities/index.json
CACHE https://api.nuget.org/v3-vulnerabilities/2025.09.06.03.
CACHE https://api.nuget.org/v3-vulnerabilities/2025.09.06.03.24.34/2025.09.10.03.24.43/vulnerability.update.json
Error List
Output



Step 3: Configure AWS Lambda Support

1. Open `Program.cs` file
2. Add the following code after `var builder = WebApplication.CreateBuilder(args);`:

```
// Add AWS Lambda support
builder.Services.AddAWSLambdaHosting(LambdaEventSource.HttpApi);
```

3. Open your `.csproj` file and add this inside the `<PropertyGroup>` section:

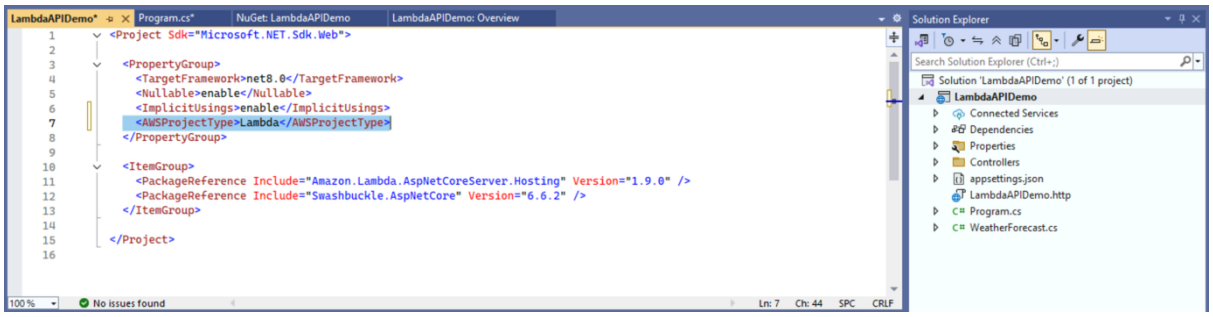
```
<AWSProjectType>Lambda</AWSProjectType>
```

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddAWSLambdaHosting(LambdaEventSource.HttpApi);
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
```

Read more [Introducing the .NET 6 runtime for AWS Lambda]

Open .csproj file and add the following tag in the **PropertyGroup** section.

<AWSProjectType>Lambda</AWSProjectType>



Now, You're done with the API stuff for this demo.

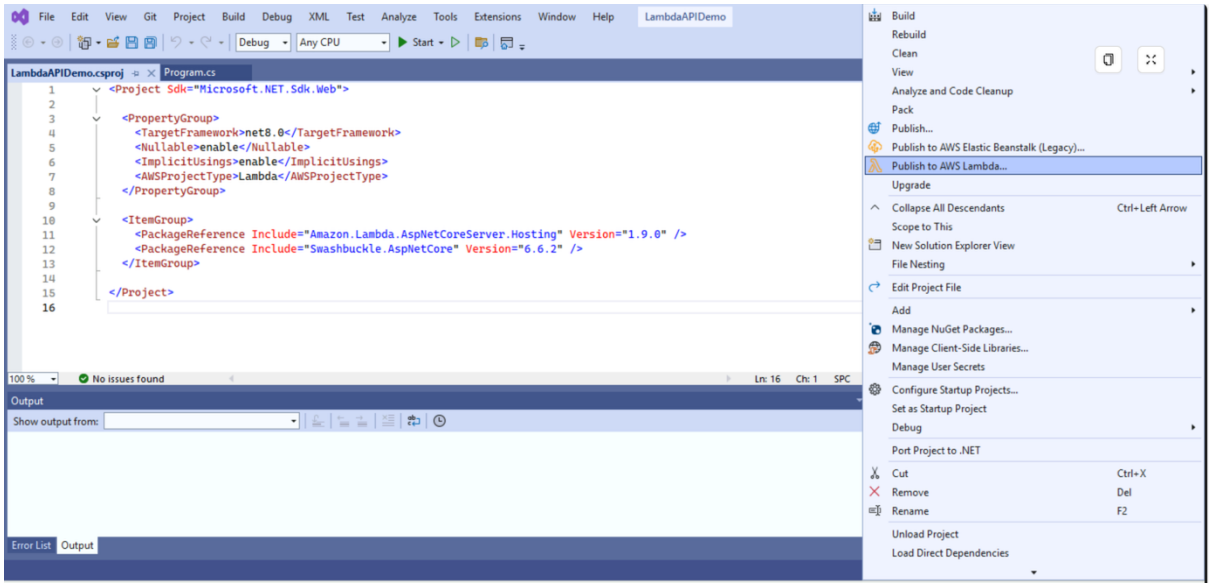
Step 4: Create Lambda Function in AWS Console

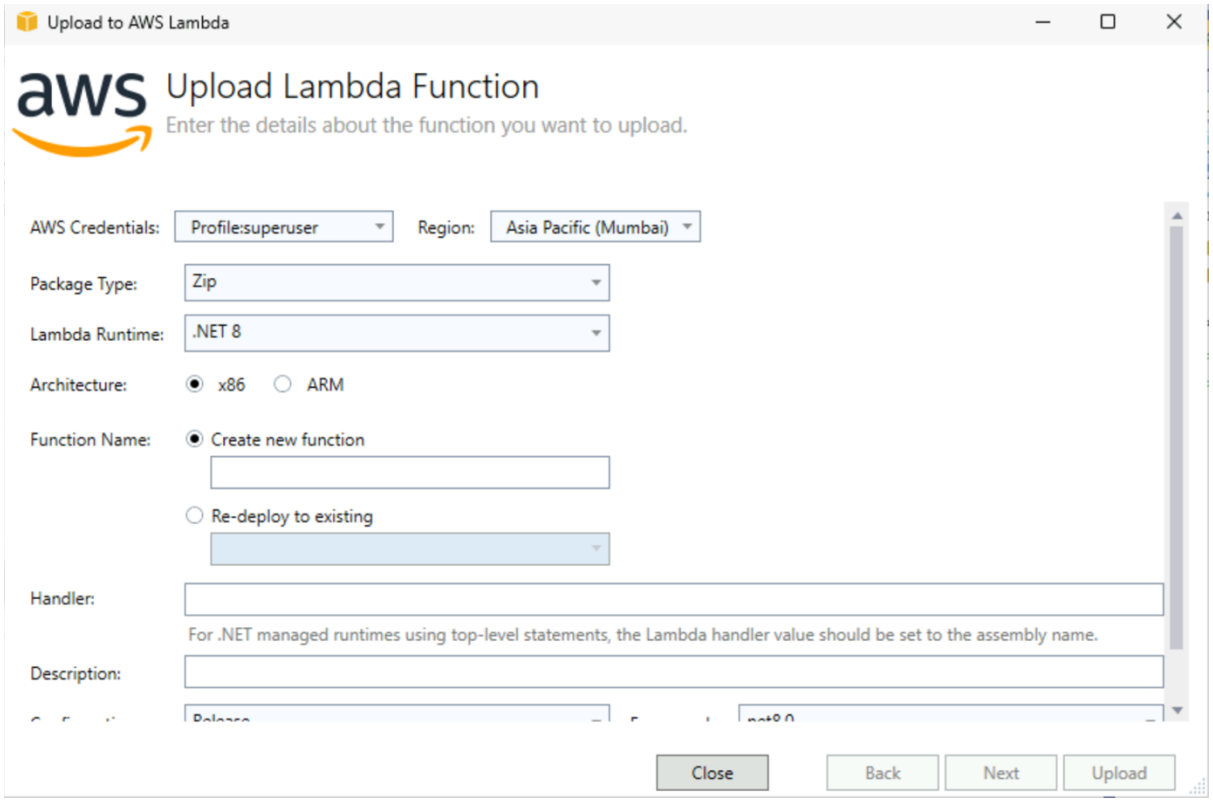
1. Sign in to AWS Management Console
2. Navigate to **Lambda Service**
3. Click **Create function**
4. Choose **Author from scratch**
5. Configure function:
    - Function name: `YourFunctionName`
    - Runtime: **.NET 8** (or your .NET version)
    - Architecture: x86_64
    - Execution role: Create new role with basic Lambda permissions

Step 5: Deploy Your Application

Option A: Deploy from Visual Studio (if you have AWS credentials configured)

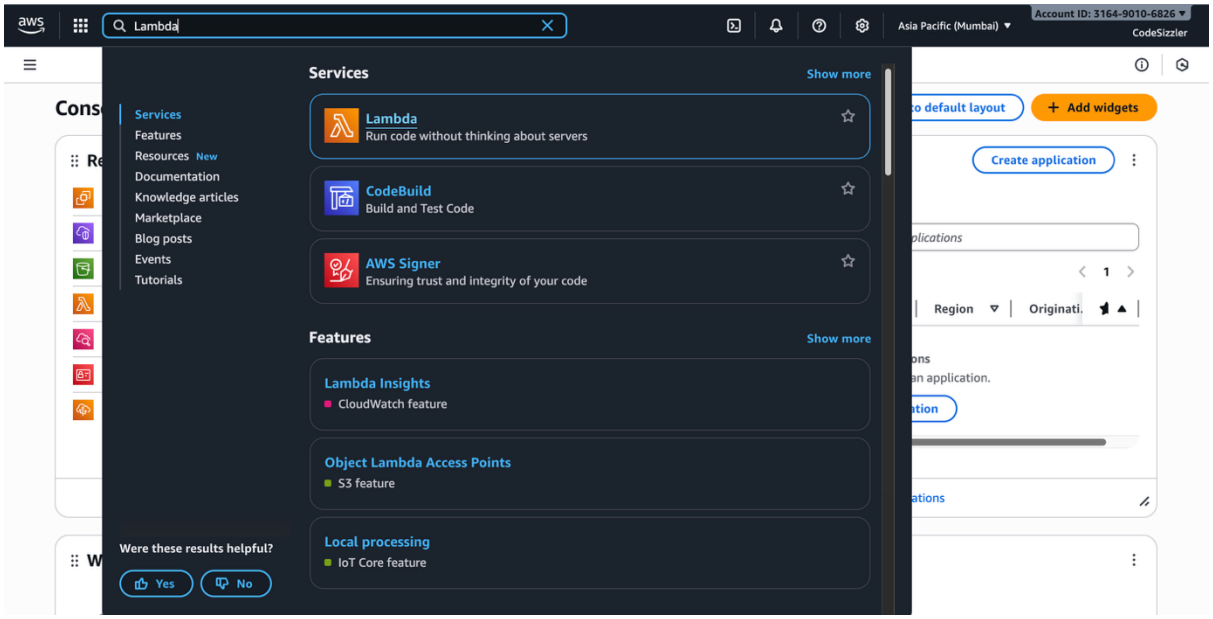
1. Right-click project in Solution Explorer
2. Select **Publish to AWS Lambda**
3. Follow the deployment wizard



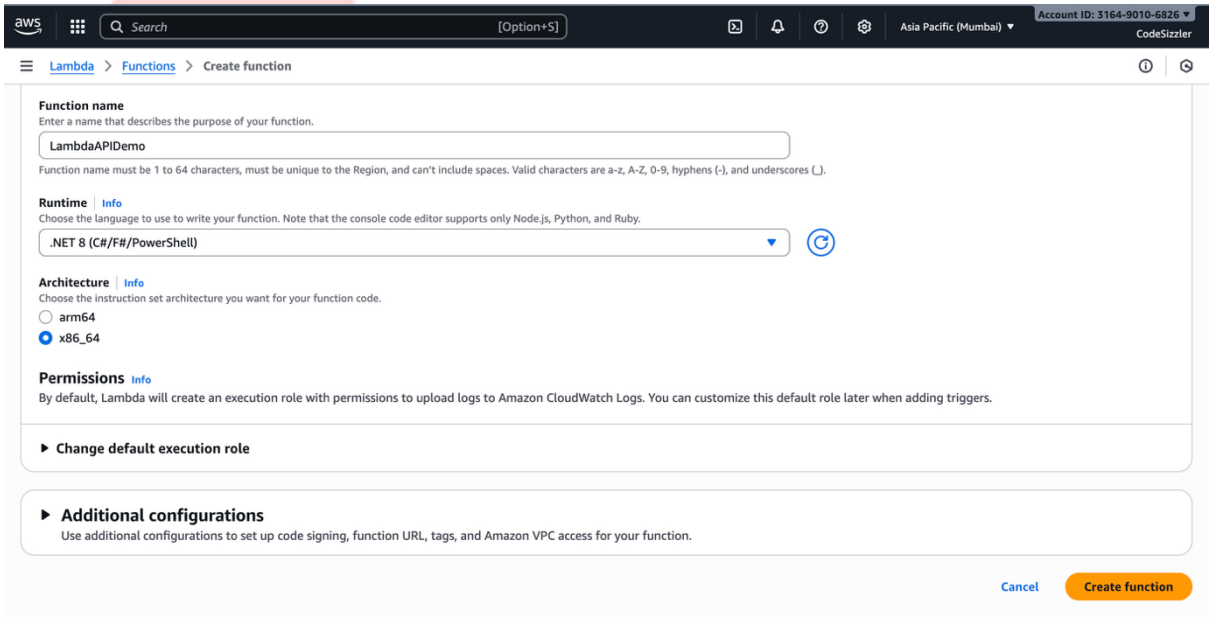


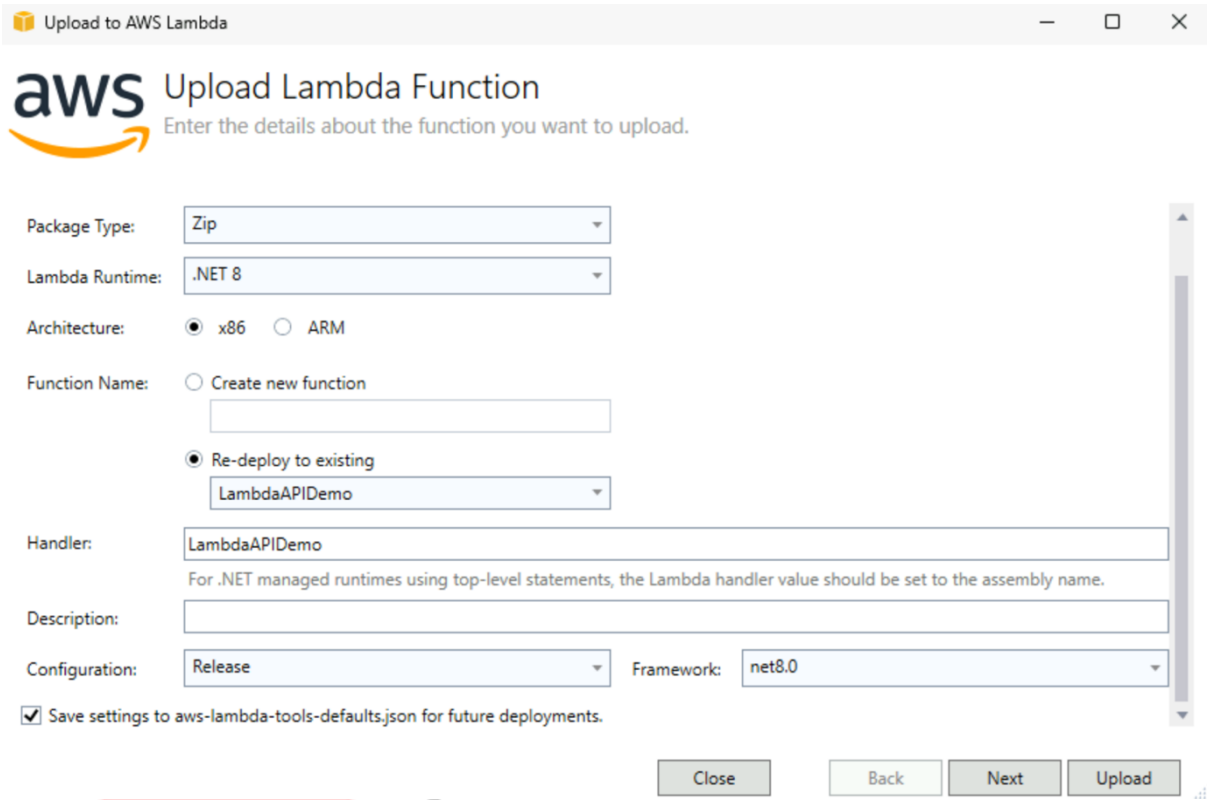
Option B: Manual Deployment via AWS Console

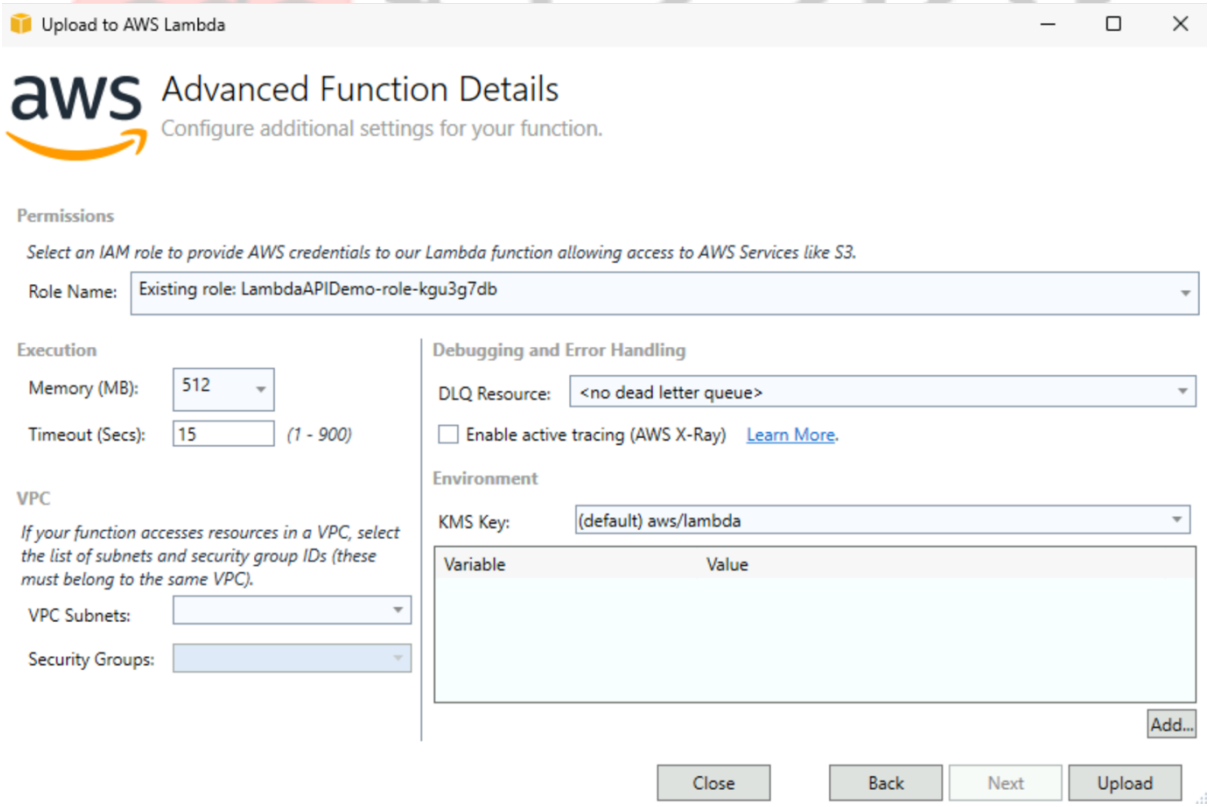
1. Build your project in **Release** mode
2. Locate the published files in `bin/Release/net6.0/publish` folder
3. Zip the contents (not the folder itself)
4. In AWS Lambda console:
    - Upload the zip file
    - Set handler: `YourAssemblyName::YourAssemblyName.LambdaEntryPoint::FunctionHandlerAsync`
    - Save and test

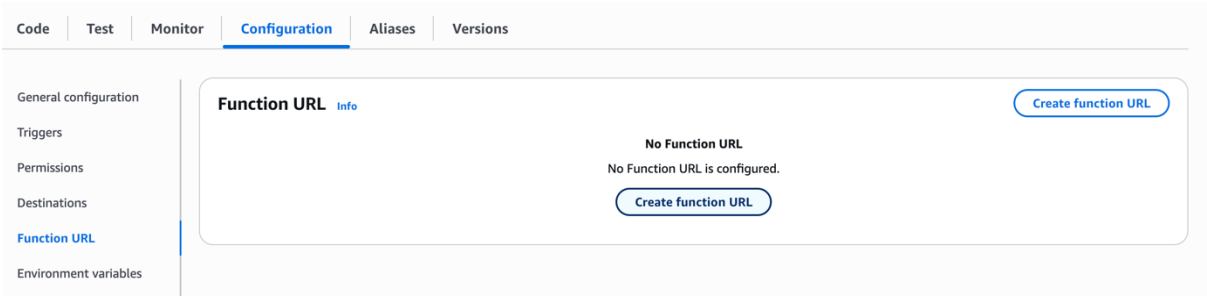


Add a Function name, select Runtime - .NET 8 and other configurations needed.

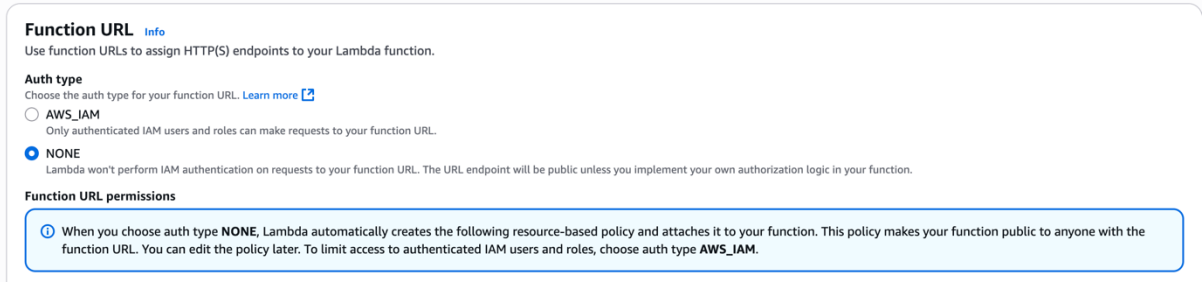


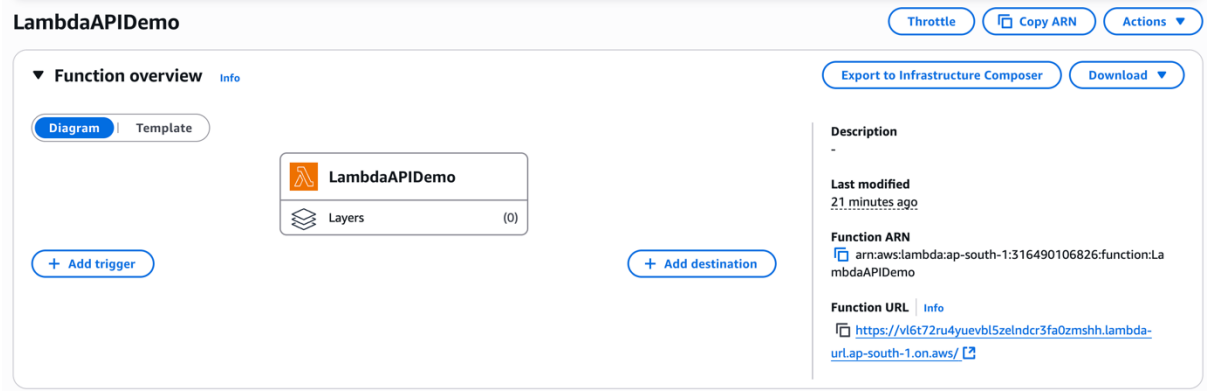

Upload to AWS Lambda
Upload Lambda Function
Enter the details about the function you want to upload.
Package Type:
Zip
Lambda Runtime:
.NET 8
Architecture:
x86
ARM
Function Name:
Create new function
Re-deploy to existing
LambdaAPIDemo
Handler:
LambdaAPIDemo
For .NET managed runtimes using top-level statements, the Lambda handler value should be set to the assembly name.
Description:
Configuration:
Release
Framework:
net8.0
Save settings to aws-lambda-tools-defaults.json for future deployments.
Close
Back
Next
Upload

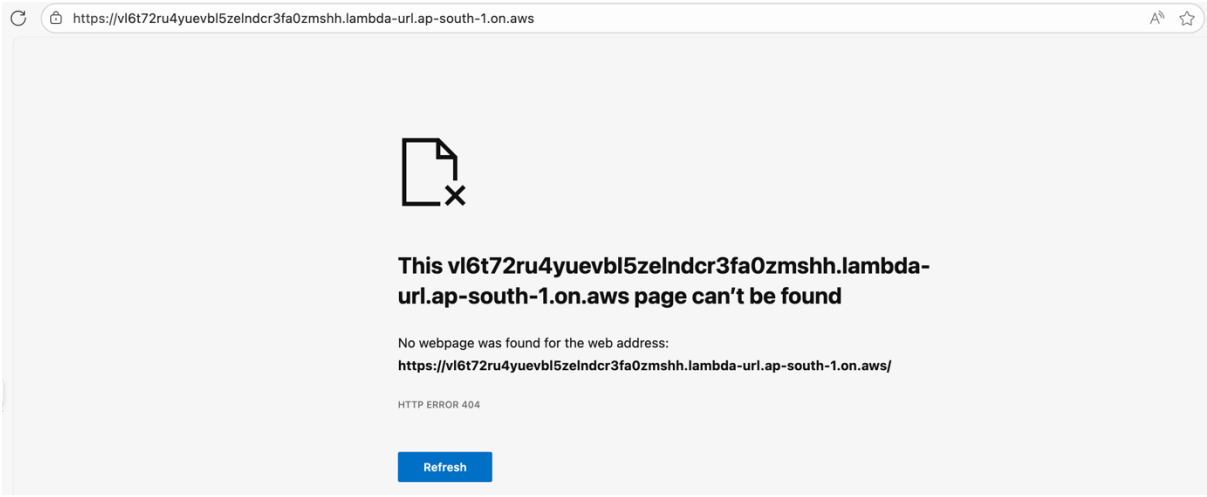

Upload to AWS Lambda
Advanced Function Details
Configure additional settings for your function.
Permissions
Select an IAM role to provide AWS credentials to our Lambda function allowing access to AWS Services like S3.
Role Name:
Existing role: LambdaAPIDemo-role-kgu3g7db
Execution
Memory (MB):
512
Timeout (Secs):
15
(1 - 900)
VPC
If your function accesses resources in a VPC, select the list of subnets and security group IDs (these must belong to the same VPC).
VPC Subnets:
Security Groups:
Debugging and Error Handling
DLQ Resource:
<no dead letter queue>
Enable active tracing (AWS X-Ray)
Learn More.
Environment
KMS Key:
(default) aws/lambda
Variable
Value
Add...
Close
Back
Next
Upload

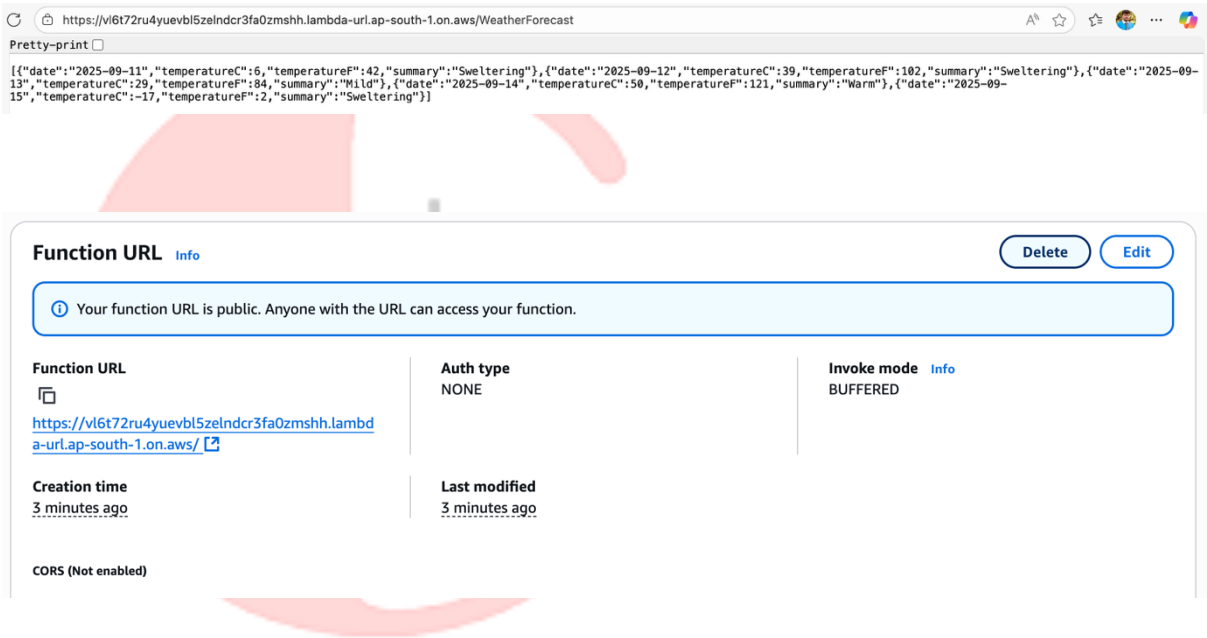


Configure Function URL



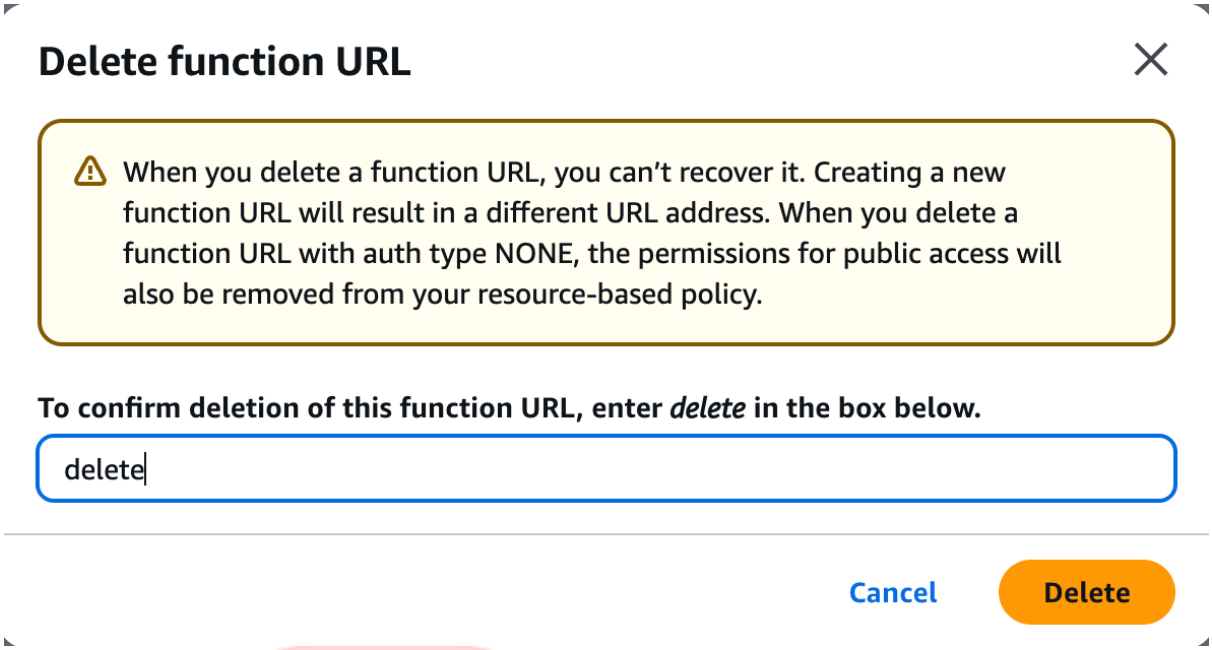






Important Notes

- Ensure your IAM role has proper permissions for Lambda execution
- The `AWSProjectType` tag helps Visual Studio recognize it as a Lambda project
- Handler name must match your assembly name
- For production, set up proper logging and monitoring using AWS CloudWatch



Conclusion

This lab successfully demonstrated the process of deploying an ASP.NET Core Web API application to AWS Lambda, showcasing the powerful combination of modern .NET development with serverless cloud architecture. Through this implementation, we achieved a fully functional, scalable web API that operates without traditional server management overhead.

The integration of ASP.NET Core with AWS Lambda proves to be an effective solution for building cost-efficient, highly available APIs that automatically scale with demand. By leveraging the `Amazon.Lambda.AspNetCoreServer.Hosting` package and proper configuration, we maintained the familiar ASP.NET Core development experience while gaining the benefits of serverless execution.

Key successes include:

- Seamless transition from traditional hosting to serverless architecture
- Proper configuration of Lambda-specific project settings
- Successful deployment and testing of API endpoints
- Demonstration of AWS's pay-per-use pricing model for API solutions

This approach is particularly valuable for applications with variable traffic patterns, microservices architectures, and projects requiring rapid deployment with minimal operational overhead. The lab reinforces that AWS Lambda provides an excellent platform for running .NET applications while abstracting infrastructure management

concerns, allowing developers to focus on business logic rather than server maintenance.

The skills developed in this lab are directly applicable to real-world scenarios where organizations are migrating to serverless architectures to improve scalability, reduce costs, and increase deployment agility.