

Tutorial - Azure functions in .NET

Introduction

Azure Functions is a serverless computing service provided by Microsoft Azure, allowing developers to build and deploy event-driven functions without managing infrastructure. This tutorial will guide you through the process of creating and deploying Azure Functions using the .NET runtime.

Prerequisites

Before you begin, ensure you have the following:

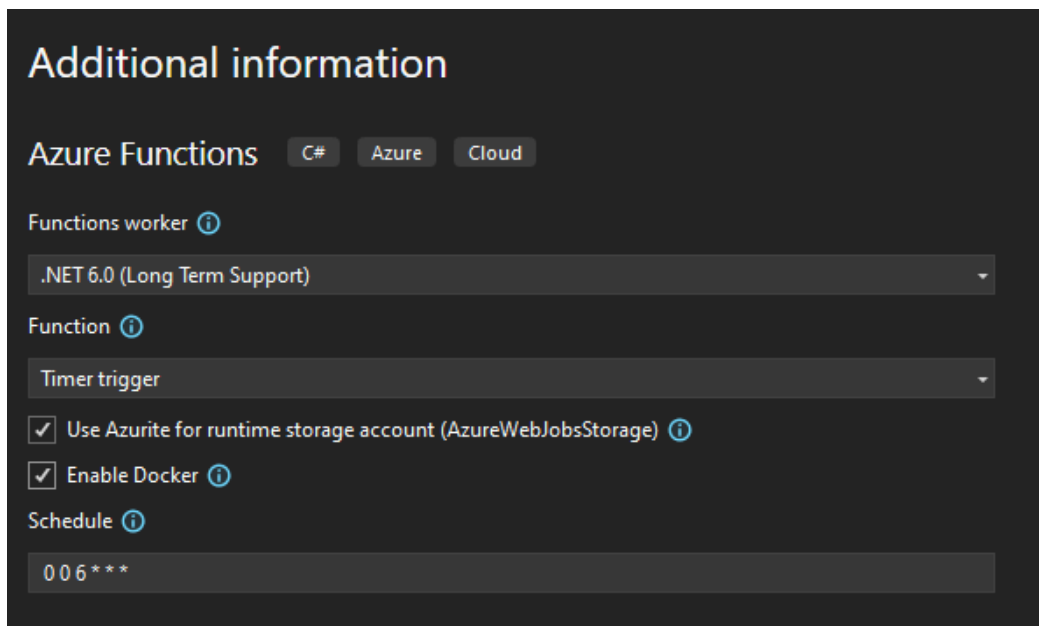
- An Azure account
- Visual Studio

Create a New Azure Functions Project

1. Open Visual Studio and select "Create a new project."
2. In the project creation wizard, search for "Azure Functions" and choose the "Azure Functions" template.
3. Choose the appropriate options for your project, such as the template.

For this example we will be creating a Timer trigger function. This will be triggered according to the cron expression schedule. In this example the function will be triggered every day at 6:00.

Pay attention to the schedule expression format: Azure functions don't use the usual five-part format. Instead they need a six-part NCrontab expression. You can test different values for NCrontab [here](#).



The screenshot shows the 'Additional information' dialog in Visual Studio. It has a title bar 'Additional information' and a subtitle 'Azure Functions'. Below the subtitle are three tabs: 'C#', 'Azure', and 'Cloud'. The 'Functions worker' section has a dropdown menu set to '.NET 6.0 (Long Term Support)'. The 'Function' section has a dropdown menu set to 'Timer trigger'. There are two checked checkboxes: 'Use Azurite for runtime storage account (AzureWebJobsStorage)' and 'Enable Docker'. The 'Schedule' section has a text input field containing the cron expression '0 0 6 * * *'.

4. Click "Create" to generate the project.

Writing Your First Azure Function

Open the generated Azure Functions project in Visual Studio.

There will be a function already generated, which you can rename to something more meaningful. In this example our function will generate a random 5-letter word, so we can name it “WordGenerator”.

```
public class WordGenerator
{
    private readonly ILogger _logger;

    0 references | Alexandra.Birle, 14 hours ago | 1 author, 1 change
    public WordGenerator(ILoggerFactory loggerFactory)
    {
        _logger = loggerFactory.CreateLogger<WordGenerator>();
    }

    [FunctionName(name: "WordGenerator")]
    0 references | Alexandra.Birle, 14 hours ago | 1 author, 1 change
    public void Run([TimerTrigger(scheduleExpression: "0 0 6 * * *")
        #if DEBUG
        , RunOnStartup=true
        #endif
        ][TimerInfo myTimer)
    {
        _logger.LogInformation(message: $"C# Timer trigger function executed at: {DateTime.Now}");

        if (myTimer.ScheduleStatus is not null)
        {
            _logger.LogInformation(message: $"Next timer schedule at: {myTimer.ScheduleStatus.Next}");
        }

        var Random random = new Random();
        var List<string> words = new List<string>() { "speak", "stick", "paste", "tears", "glove" };
        int index = random.Next(words.Count);
        var string word = words[index];

        _logger.LogInformation(message: $"Today's word is: {word}");
    }
}
```

We have a logger that logs information about our function every time it is triggered.

We can also see our chosen schedule in the TimerTrigger attribute. We also added some simple code that chooses a random word from a list and logs it.

For local testing, you could change the schedule so that the function runs more often (every minute for instance: 0 * * * * *). You can also just set the RunOnStartup attribute to “true” when in debug mode, as shown in the example. This would trigger the function once when tested locally.

Debugging and Testing

Build the project to ensure there are no errors.

Press F5 to run the project in debug mode. If you set the RunOnStartup attribute, the function should trigger and display the logs once.

Deploying Your Azure Function

Right-click on the project and select "Publish."

Choose the target Azure subscription and follow the prompts to deploy your function to Azure.

Once deployed, your function will be accessible in Azure. You can view the triggers in the Monitor section of your function.

WordGenerator | Monitor

Function

Search

Overview

Developer

Code + Test

Integration

Monitor

Function Keys

Invocations

Logs

Success Count

9

Last 30 Days

Error Count

0

Last 30 Days

Invocation Traces

The twenty most recent function invocation traces. For more advanced analysis, run the query in Application Insights.

Run query in Application Insights

Refresh

Filter invocations

Date (UTC)	Success	Result Code	Duration (ms)	Operation Id
2024-02-01 09:29:00.004	Success	0	1	9d41e84fae439685d79355b8562de6fe
2024-02-01 09:28:00.017	Success	0	1	98fcf4708b0c394dea3a8d72897034f7
2024-02-01 09:27:00.006	Success	0	1	4f4eb5882c22ca1e155d5adfa181e917
2024-02-01 09:25:59.989	Success	0	1	22ac7578a1e1f0dfe52564d8758a6f

You can also trigger it yourself from the Code + Test section. Click on “Test/Run”, make sure you have the “master (Host key)” selected from the dropdown and an empty Body, then click “Run”. You should be able to see your logs if you expand the Logs section.

Input

Output

Provide parameters to test your function in the portal. Results can be found in the Output tab.

Key

master (Host key)

Body

1

Run

Close

You can find the code from the example [here](#).