Azure Functions Tools for Visual Studio

09/06/2017 • 6 minutes to read • Contributors

In this article

Azure Functions Tools for Visual Studio 2017 is an extension for Visual Studio that lets you develop, test, and deploy C# functions to Azure. If this is your first experience with Azure Functions, you can learn more at An introduction to Azure Functions.

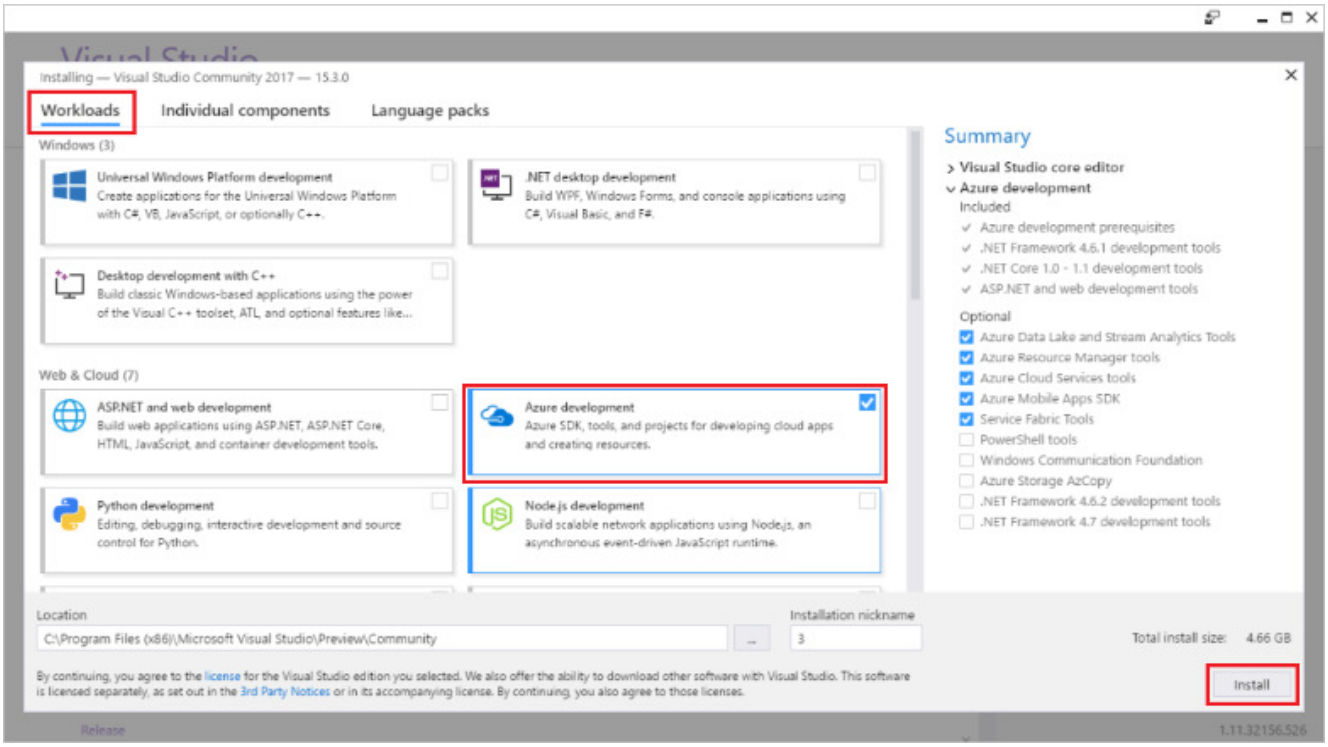
The Azure Functions Tools provides the following benefits:

- Edit, build, and run functions on your local development computer.
- Publish your Azure Functions project directly to Azure.
- Use WebJobs attributes to declare function bindings directly in the C# code instead of maintaining a separate function.json for binding definitions.
- Develop and deploy pre-compiled C# functions. Pre-complied functions provide a better cold-start performance than C# script-based functions.
- Code your functions in C# while having all of the benefits of Visual Studio development.

This topic shows you how to use the Azure Functions Tools for Visual Studio 2017 to develop your functions in C#. You also learn how to publish your project to Azure as a .NET assembly.

Prerequisites

Azure Functions Tools is included in the Azure development workload of Visual Studio 2017 version 15.3, or a later version. Make sure you include the **Azure development** workload in your Visual Studio 2017 version 15.3 installation:



> ⓘ Note
>
> After you install or upgrade to Visual Studio 2017 version 15.3, you must manually update the Visual Studio 2017 tools for Azure Functions. These additional steps are not required when you install or upgrade to version 15.4 or a later version.
> You can update the tools from the **Tools** menu under **Extensions and Updates...** > **Updates** > **Visual Studio Marketplace** > **Azure Functions and Web Jobs Tools** > **Update**.

To create and deploy functions, you also need:

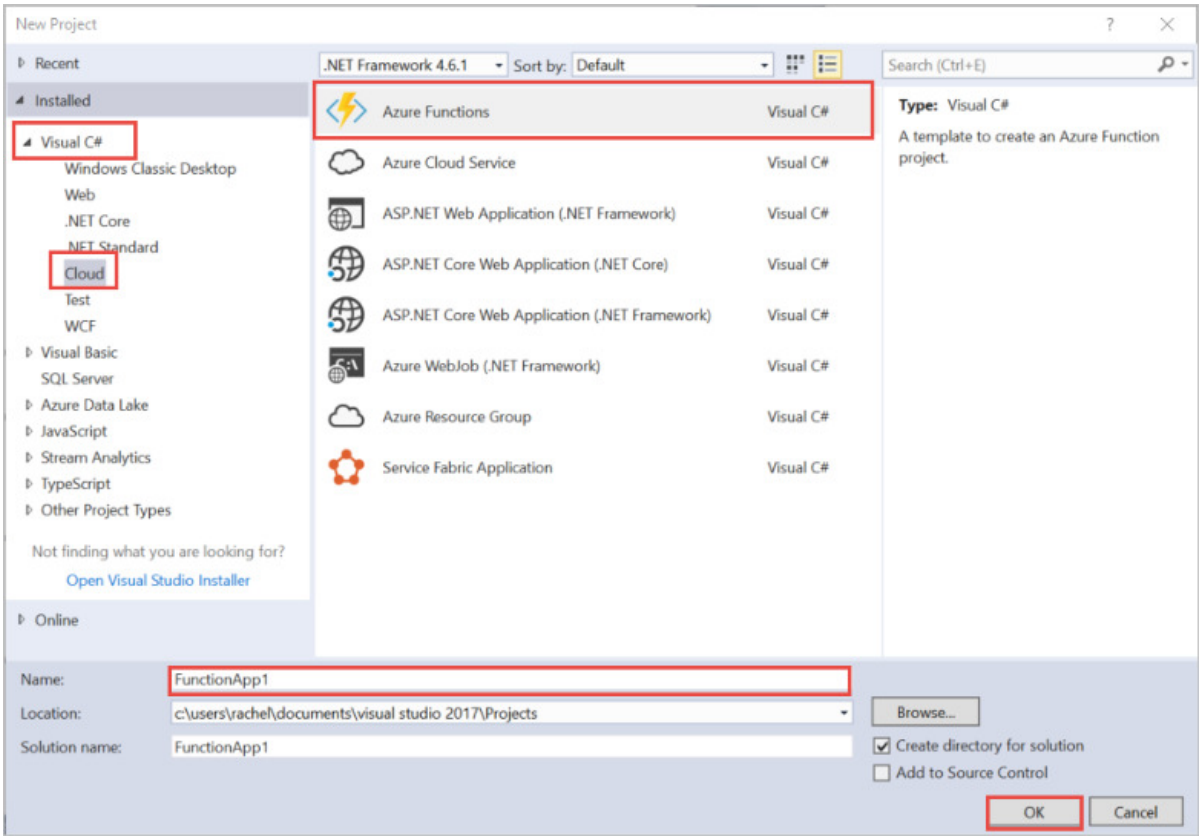
- An active Azure subscription. If you don't have an Azure subscription, free accounts are available.
- An Azure Storage account. To create a storage account, see Create a storage account.

Create an Azure Functions project

The Azure Functions project template in Visual Studio creates a project that can be published to a function app in Azure. A function app lets you group functions as a logic unit for easier management, deployment, and sharing of resources.

1. Right mouse click on the project node in **Solution Explorer**, then choose **Add** > **New Item**. Choose **Azure Function** from the dialog box.

2. In the **New Project** dialog, expand **Visual C#** > **Cloud** node, select **Azure Functions**, type a **Name** for your project, and click **OK**. The function app name must be valid as a C# namespace, so don't use underscores, hyphens, or any other nonalphanumeric characters.



Configure the project for local development

When you create a new project using the Azure Functions template, you get an empty C# project that contains the following files:

- **host.json**: Lets you configure the Functions host. These settings apply both when running locally and in Azure. For more information, see host.json reference.
- **local.settings.json**: Maintains settings used when running functions locally. These settings are not used by Azure, they are used by the Azure Functions Core Tools. Use this file to specify settings, such as connection strings to other Azure services. Add a new key to the **Values** array for each connection required by functions in your project. For more information, see Local settings file in the Azure Functions Core Tools topic.

The Functions runtime uses an Azure Storage account internally. For all trigger types other than HTTP and webhooks, you must set the **Values.AzureWebJobsStorage** key to a valid Azure Storage account connection string.

> ⓘ Note
>
> Use of the Azure Storage Emulator is not supported by Azure Functions tools when developing locally.

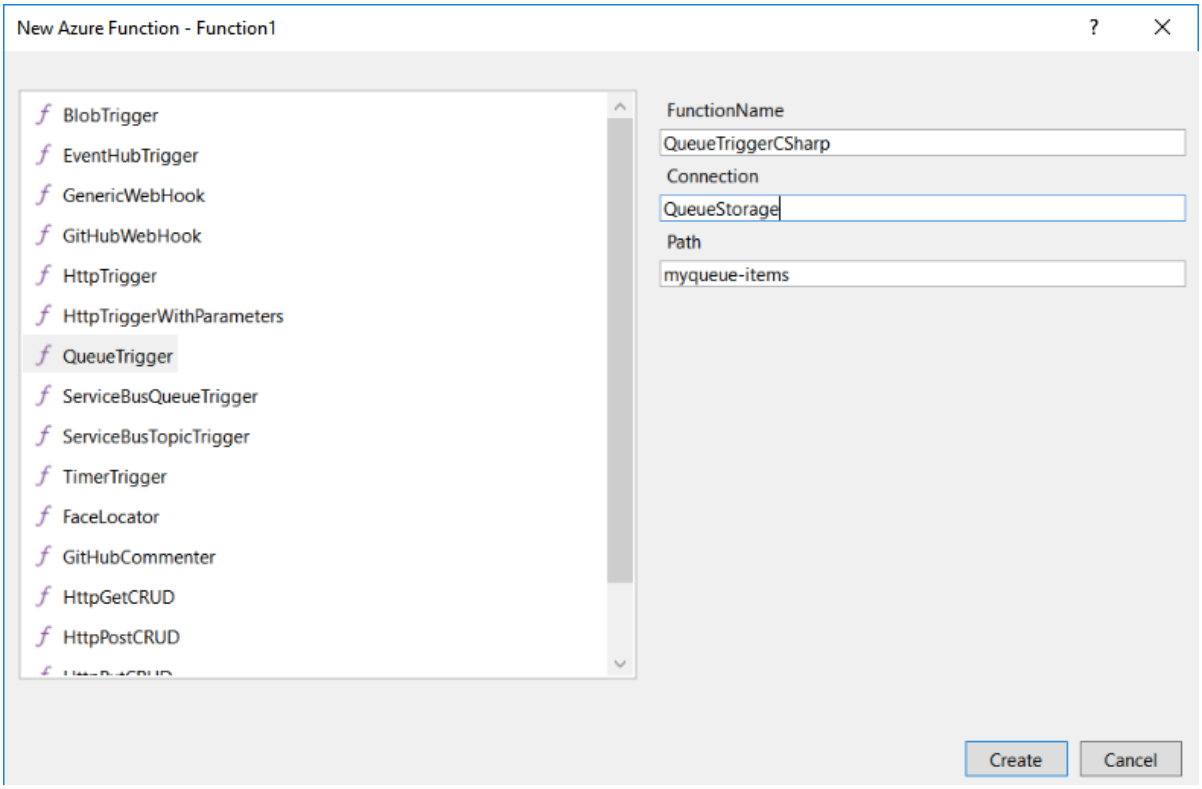
To set the storage account connection string:

1. In Visual Studio, open **Cloud Explorer**, expand **Storage Account** > **Your Storage Account**, then select **Properties** and copy the **Primary Connection String** value.
2. In your project, open the local.settings.json project file and set the value of the **AzureWebJobsStorage** key to the connection string you copied.
3. Repeat the previous step to add unique keys to the **Values** array for any other connections required by your functions.

Create a function

In pre-compiled functions, the bindings used by the function are defined by applying attributes in the code. When you use the Azure Functions Tools to create your functions from the provided templates, these attributes are applied for you.

1. In **Solution Explorer**, right-click on your project node and select **Add** > **New Item**. Select **Azure Function**, type a **Name** for the class, and click **Add**.
2. Choose your trigger, set the binding properties, and click **Create**. The following example shows the settings when creating a Queue storage triggered function.



A connection string key named **QueueStorage** is supplied, which is defined in the local.settings.json file.

3. Examine the newly added class. You see a static **Run** method, that is attributed with the **FunctionName** attribute. This attribute indicates that the method is the entry point for the function.

   For example, the following C# class represents a basic Queue storage triggered function:

C# Copy

```
using System;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Host;

namespace FunctionApp1
{
    public static class Function1
    {
        [FunctionName("QueueTriggerCSharp")]
        public static void Run([QueueTrigger("myqueue-items", Connection = "Que
        {
            log.Info($"C# Queue trigger function processed: {myQueueItem}");
        }
    }
}
```

A binding-specific attribute is applied to each binding parameter supplied to the entry point method. The attribute takes the binding information as parameters. In the previous example, The first parameter has a **QueueTrigger** attribute applied, indicating queue triggered function. The queue name and connection string setting name are passed as parameters.

Testing functions

Azure Functions Core Tools lets you run Azure Functions project on your local development computer. You are prompted to install these tools the first time you start a function from Visual Studio.

To test your function, press F5. If prompted, accept the request from Visual Studio to download and install Azure Functions Core (CLI) tools. You may also need to enable a firewall exception so that the tools can handle HTTP requests. 2

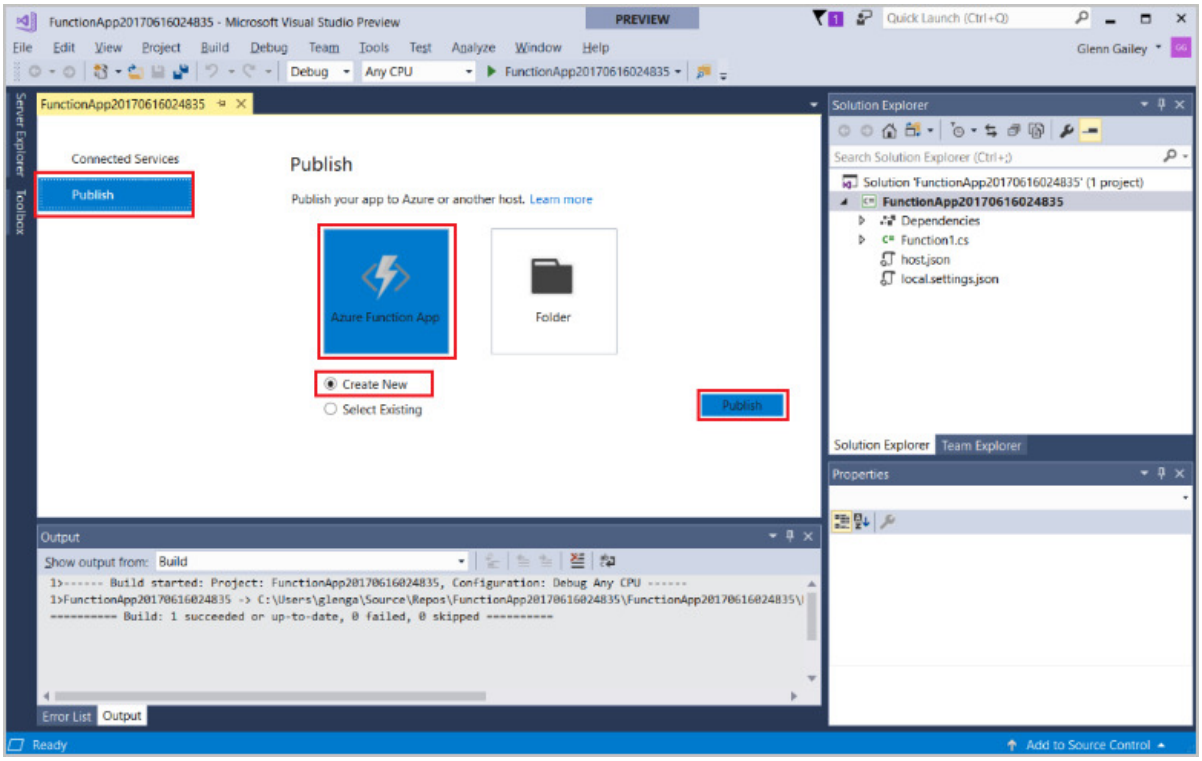
With the project running, you can test your code as you would test deployed function. For more information, see Strategies for testing your code in Azure Functions. When running in debug mode, breakpoints are hit in Visual Studio as expected.

For an example of how to test a queue triggered function, see the queue triggered function quickstart tutorial.

To learn more about using the Azure Functions Core Tools, see Code and test Azure functions locally.

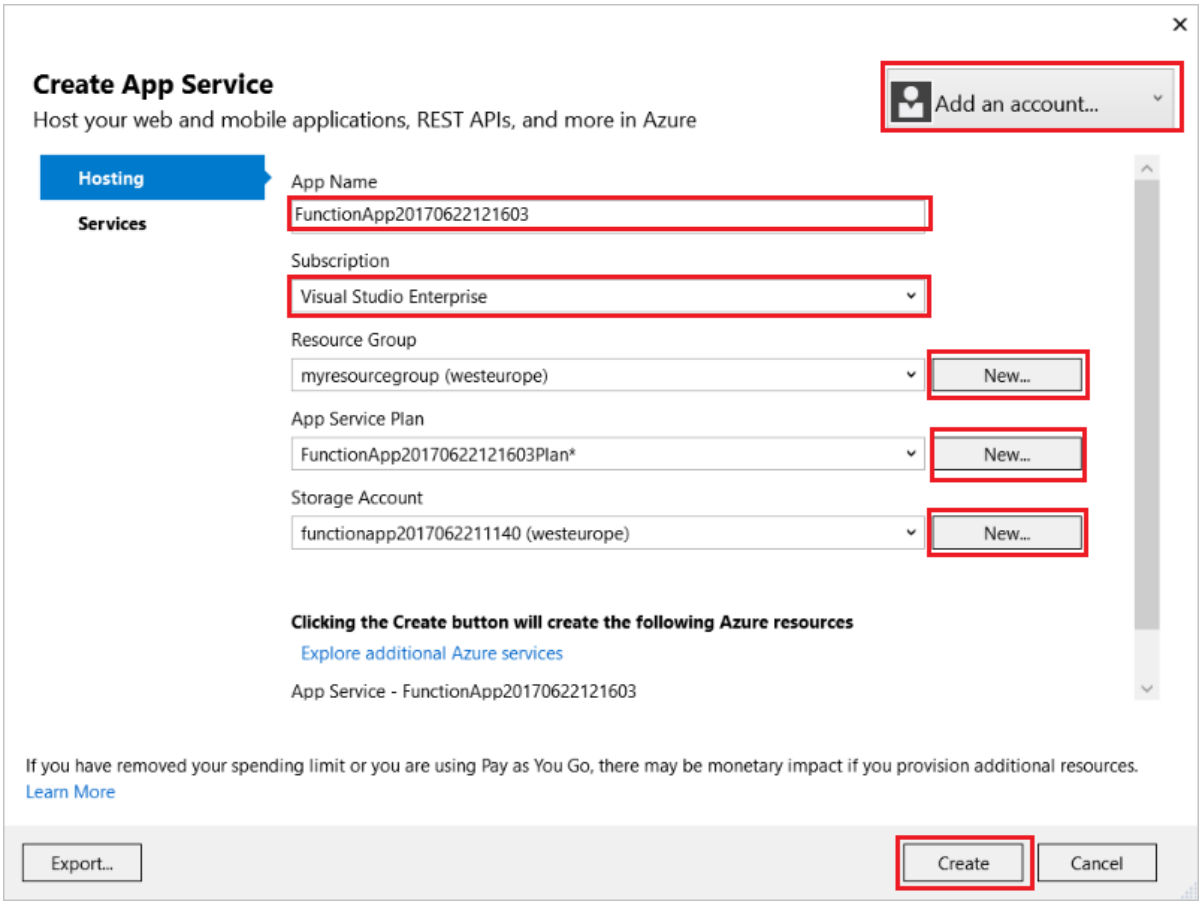
Publish to Azure

1. In **Solution Explorer**, right-click the project and select **Publish**. Choose **Create New** and then click **Publish**.



2. If you haven't already connected Visual Studio to your Azure account, click **Add an account...**.

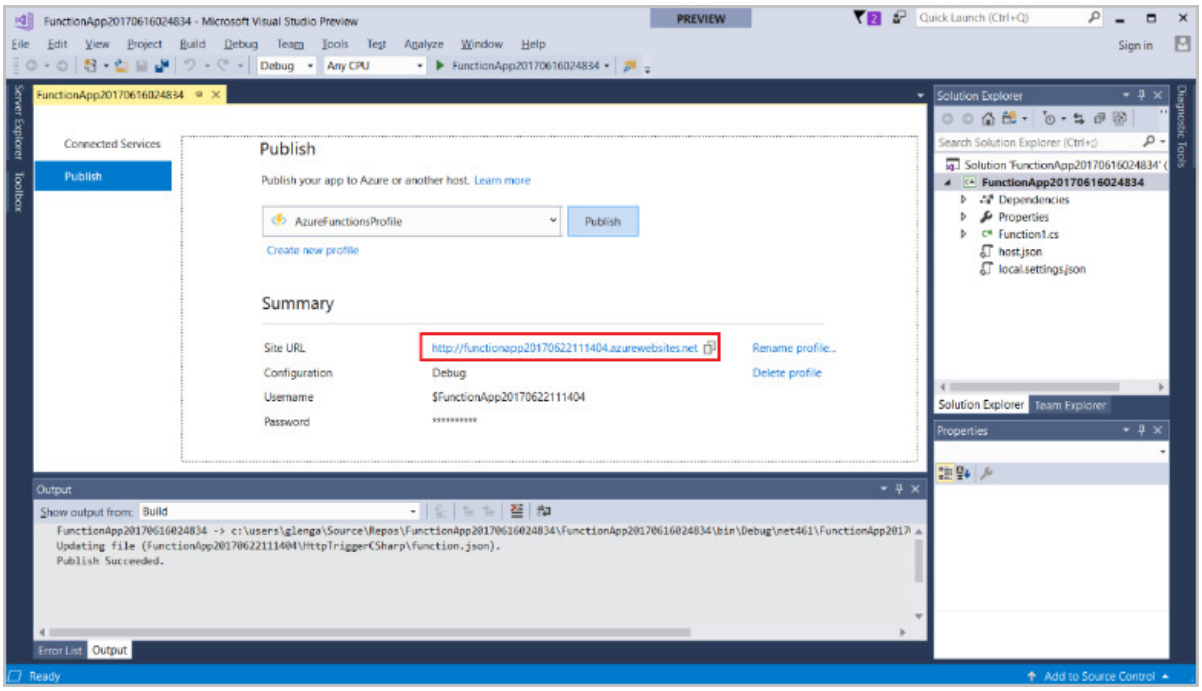
3. In the **Create App Service** dialog, use the **Hosting** settings as specified in the following table:



Setting	Suggested value	Description

Setting	Suggested value	Description
App Name	Globally unique name	Name that uniquely identifies your new function app.
Subscription	Choose your subscription	The Azure subscription to use.
Resource Group	myResourceGroup	Name of the resource group in which to create your function app.
App Service Plan	Consumption plan	Make sure to choose the **Consumption** under **Size** when you create a new plan.
Storage account	Globally unique name	Use an existing storage account or create a new one.

4. Click **Create** to create a function app in Azure with these settings. After the provisioning is complete, make a note of the **Site URL** value, which is the address of your function app in Azure.



ⓘ Note

Any settings you added in the local.settings.json must be also added to the function app in Azure. These settings are not added automatically. You can add required settings to your function app in one of these ways:

- Using the Azure portal.