Develop Azure Logic Apps workflows locally

Azure Logic Apps Extensions for the Azure Functions runtime along with the new Visual Studio Code extension for Azure Logic Apps were created to provide a great developer experience for building logic app workflows. You can build these workflows in your development environment and deploy them to multiple hosting environments, such as Azure App Service, Azure Function App, or as a Docker container anywhere you want.

**Important:** This early preview release provides an early look at functionality so that participants can give feedback. Bugs and issues are expected.

These extensions for Azure Logic Apps bring most of the capabilities from Azure Logic Apps in the cloud to your local development experience. The extension also provides many new capabilities, for example:

* Managed API connectors
  + Logic Apps offers 300+ managed connectors for connecting to Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS) apps and services.

**Important:** Currently, access to cloud-based connectors requires enabling them at the subscription level. This action will initially be done on a case by case basis for select preview participants.

* + You can still create connections that store credentials in the cloud for connectors, for example, OAuth access tokens for connecting to Outlook.
  + Logic Apps generates a Shared Access Signature (SAS) connection string that workflows running anywhere can use to send requests to the cloud connection runtime endpoint. This connection string is saved with other application settings so that you can easily store them in Azure Key Vault when deployed to Azure.
* Stateless workflows
  + You can author stateless workflows like any other workflow by using the Logic Apps Designer. However, unlike regular workflows, stateless workflows don't persist between actions, and don't store run histories by default.
  + You can enable run histories, if necessary, for better debuggability.
  + Stateless workflows provide faster response times and high throughput. Due to non-persistence, these workflows are also less costly to run.
* Additional connectors, specifically, Event Hubs, Service Bus, and SQL, run in-process like built-in native connectors, and provide faster response time, high throughput, and no throttling, unlike the cloud-based connectors.
* You can invoke an Azure function natively and directly from your workflow, which runs on the Azure Functions runtime.

# Set up your developer environment

* In Visual Studio Code, download and install the preview version of the [Azure Functions extension](https://microsoft.sharepoint.com/:u:/t/LogicApps/EQiZLIFiLwlBl0CTOgNKsN0BTP0NQ1z-11LMmBjfg5nfvQ?e=Rg4O7k), which you can install as a VSIX file. On the extensions list toolbar, select the ellipses (…), and select **Install from VSIX**. **Note**: This version will replace any existing Azure Functions extension but preserves the capability to author Azure Functions.

|  |
| --- |
|  |

* If you haven't already done so, install the [C# extension for Visual Studio Code](https://marketplace.visualstudio.com/items?itemName=ms-vscode.csharp). This extension enables F5 functionality to run the workflow.
* Reload Visual Studio Code so that the extensions are correctly installed. Also, you can close and reopen Visual Studio Code.
* Change the Visual Studio Code settings to use the Azure Functions project runtime Version 3. To open the settings, press **Ctrl +**.

|  |
| --- |
|  |

* Install the latest Azure Functions core tools from these locations: (TODO: we should move these to our CDN if the official packages are not available at the time of shipping)

**Note**: If you previously installed the Azure Functions core tools, please uninstall them first or make sure that the PATH environment variable points to the specific version below.

win-x64 (MSI): [https://functionscdn.azureedge.net/public/3.0.2569/func-cli-3.0.2569-x64.msi](https://nam06.safelinks.protection.outlook.com/?url=https%3A%2F%2Ffunctionscdn.azureedge.net%2Fpublic%2F3.0.2569%2Ffunc-cli-3.0.2569-x64.msi&data=02%7C01%7Crohithah%40microsoft.com%7C81c2a988953143497c5b08d8034f9299%7C72f988bf86f141af91ab2d7cd011db47%7C1%7C0%7C637262987581935700&sdata=h5r4IGeVjn1ZIHNe9ZF71xYHeh0Ahu9AKyKXa4Uinlc%3D&reserved=0)

win-x86 (MSI): [https://functionscdn.azureedge.net/public/3.0.2569/func-cli-3.0.2569-x86.msi](https://nam06.safelinks.protection.outlook.com/?url=https%3A%2F%2Ffunctionscdn.azureedge.net%2Fpublic%2F3.0.2569%2Ffunc-cli-3.0.2569-x86.msi&data=02%7C01%7Crohithah%40microsoft.com%7C81c2a988953143497c5b08d8034f9299%7C72f988bf86f141af91ab2d7cd011db47%7C1%7C0%7C637262987581925704&sdata=g2nqLBSPgLnX8WbSSswZ8YqSd0VmJnNXnq8e%2BOdWy8o%3D&reserved=0)

* If you don't have the [standalone Azure Storage Emulator](https://go.microsoft.com/fwlink/?linkid=717179&clcid=0x409), download and install the emulator first. If you haven't previously used the emulator, you must [initialize the emulator](https://docs.microsoft.com/azure/storage/common/storage-use-emulator#start-and-initialize-the-storage-emulator) before you can start.
* If you have a previous version of the extension bundle, clean up any existing bundle binaries in the following location:

%TEMP%\Functions\ExtensionBundles\Microsoft.Azure.Functions.ExtensionBundle.Workflows\\*

# Develop, test, and deploy a workflow app

See the demo video in the preview folder in GitHub for help navigating the user experience.

* How to create workflow app project
* How to choose stateless/stateful
* How to enable azure connectors
* How to use built-in triggers and actions
* How to invoke a function
* How to deploy to a function app

# Deploy to a Docker container

1. Build your project by running this command:

dotnet build -c release

1. Publish your build by running this command:

dotnet publish -c release

1. Build a Docker container by using a workflow. For example, here's a sample Docker file for a .NET workflow:

FROM mcr.microsoft.com/azure-functions/dotnet:3.0.13614-appservice

ENV AzureWebJobsStorage <STORAGE\_CONNECTION\_STRING>

ENV AzureWebJobsScriptRoot=/home/site/wwwroot \ AzureFunctionsJobHost\_\_Logging\_\_Console\_\_IsEnabled=true

COPY ./bin/Release/netcoreapp3.1/publish/ /home/site/wwwroot

**Note:** Replace the <STORAGE\_CONNECTION\_STRING> value with the connection string to Azure Storage.

docker build --tag local/workflowcontainer .

1. Run the container locally:

docker run -p 8080:80 local/workflowcontainer

# Known issues (For an updated list, please see GitHub)

* The Workflow Action Trigger Function template is supported only for C#.
* To get the callback URL for the Request trigger in a workflow, sign in to the [Azure portal](https://portal.azure.com), and go to the logic app's **Overview** pane.
* The Python language is not supported.
* Retries are not supported for stateless workflow.
* Requires the Azure Storage Emulator for the local developer experience. You can use Azure Storage when running workflows on containers or Linux.
* An Azure function exists within a function app, so there's no impact on existing workflows. When you use a C# project, Visual Studio Code tries to pick up the bundle from a custom location, which doesn't work.
* To enable the run history for stateless workflows, in the logic app's JSON definition, set the **operationOptions** property inside the **runtimeConfiguration** object to "WithStatelessRunHistory", for example:

"runtimeConfiguration": {

"operationOptions": "WithStatelessRunHistory"

}

* When you use F5 for debugging JavaScript and TypeScript apps, a pop-up window appears and prompts you to open the Chrome dev tools. You can safely cancel this pop-up window.
* The connection string used for cloud-based connectors is valid for only 30 days.
* Although the **For each** action isn't yet supported in stateless workflows, the **For each** action is automatically added inside a stateless workflow if the output parameter is an array.
* Cloud-based connectors are supported only in the West Central US region and require subscription registration for access.
* The SQL connector only works with Azure Functions Version 2. If you use Version 3, the SQL connector only works with a C# function app.
* If you have a running workflow and want to make any changes, stop the workflow first. Or, you can restart the function after you change any configurations or workflows in the project.
* Dynamic scaling only works with Premium SKU functions when Runtime scale monitoring is enabled.
* Requires allowed origin of function apps CORS setting to be set to “\*” for run histories of workflows deployed to a function app to work.