Originally from <https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-azure-functions>.

The demo has been modified from what you will find on MSDN. For the demo, you will be creating the Azure Function from within the Logic App

# Add and run custom code snippets in Azure Logic Apps with Azure Functions

## Prerequisites

To follow this article, you need these items:

* An Azure function app, which is a container for Azure functions, and your Azure function. If you don't have a function app, [create your function app first](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/azure-functions/functions-create-first-azure-function.md). You can then create your function either [separately outside your logic app](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/logic-apps-azure-functions.md#create-function-external), or [from inside your logic app](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/logic-apps-azure-functions.md#create-function-designer) in the Logic App Designer.

Both existing and new function apps and functions have the same requirements for working with logic apps:

* + Your function app must have the same Azure subscription as your logic app.
  + Your function uses an HTTP trigger, for example, the **HTTP trigger** function template for **JavaScript** or **C#**.

The HTTP trigger template can accept content that has application/json type from your logic app. When you add an Azure function to your logic app, the Logic App Designer shows custom functions created from this template within your Azure subscription.

* + Your function doesn't use custom routes unless you've defined an [OpenAPI definition](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/azure-functions/functions-openapi-definition.md), formerly known as a [Swagger file](http://swagger.io/).
  + If you've defined an OpenAPI definition for your function, the Logic Apps Designer gives you a richer experience for working with function parameters. Before your logic app can find and access functions that have OpenAPI definitions, [set up your function app by following these steps](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/logic-apps-azure-functions.md#function-swagger).
* The logic app where you want to add the function, including a [trigger](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/logic-apps-overview.md#logic-app-concepts) as the first step in your logic app

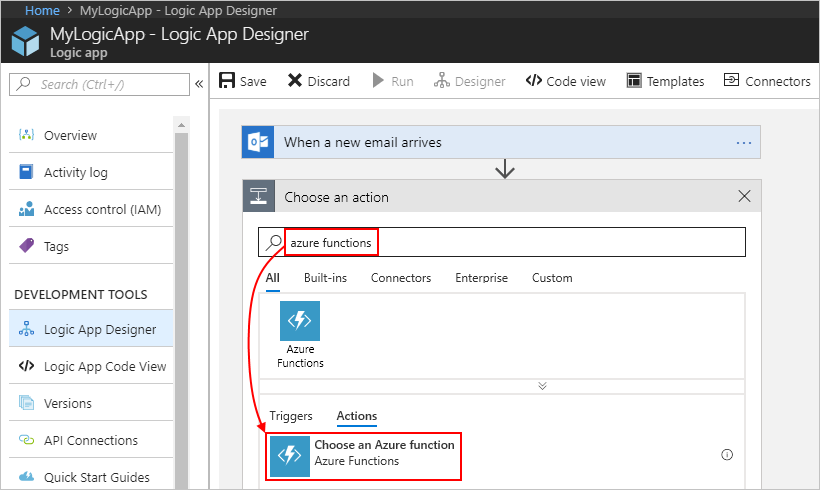
Before you can add actions that can run functions, your logic app must start with a trigger.

If you're new to logic apps, review [What is Azure Logic Apps](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/logic-apps-overview.md) and [Quickstart: Create your first logic app](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/quickstart-create-first-logic-app-workflow.md).

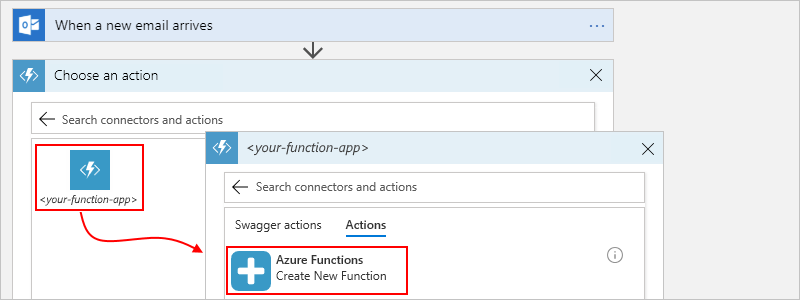
## Create functions inside logic apps

Before you can create an Azure function starting from inside your logic app in the Logic App Designer, you must first have an Azure function app, which is a container for your functions. If you don't have a function app, create that function app first. See [Create your first function in the Azure portal](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/azure-functions/functions-create-first-azure-function.md). Make sure this function app is set to version 1 (v1)!!

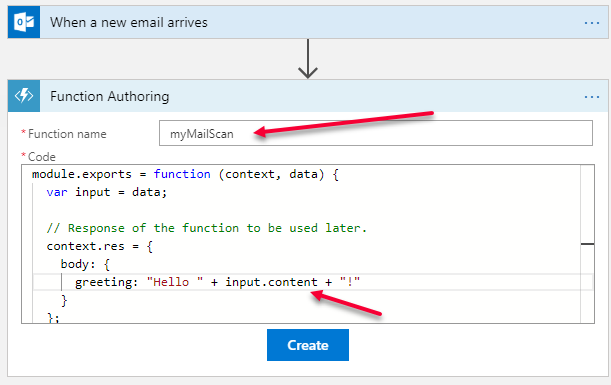
1. In the [Azure portal](https://portal.azure.com), open your logic app in the Logic App Designer. If you don’t already have a logic app, create one and just add the ‘When an email arrives’ action. Just start with a Blank Logic app. NOTE: For using your Microsoft email address, do not use your alias, use something like <firstname>.<lastname>@microsoft.com. For best results, use both combinations, ie Larry.Wall@microsoft.com;larrywa@microsoft.com.
2. To create and add your function, follow the step that applies to your scenario:
   * Under the last step in your logic app's workflow, choose **New step**.
   * Between existing steps in your logic app's workflow, move your mouse over the arrow, choose the plus (+) sign, and then select **Add an action**.
3. In the search box, enter "azure functions" as your filter. From the actions list, select this action: **Choose an Azure function - Azure Functions**

[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/find-azure-functions-action.png)

1. From the function apps list, select your function app. After the actions list opens, select this action: **Azure Functions - Create New Function**

[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/select-function-app-create-function.png)

1. In the function definition editor, define your function:
   * In the **Function name** box, provide a name for your function.
   * In the **Code** box, add your code to the function template, including the response and payload you want returned to your logic app after your function finishes running.



Notice that the code for the function above is a bit different than what will be added by the wizard by default. Make sure you put the ‘.context’ beside of ‘input’.

In the template's code, the *context* object refers to the message that your logic app sends through the **Request Body** field in a later step. To access the context object's properties from inside your function, use this syntax:

context.body.<property-name>

For example, to reference the content property inside the context object, use this syntax:

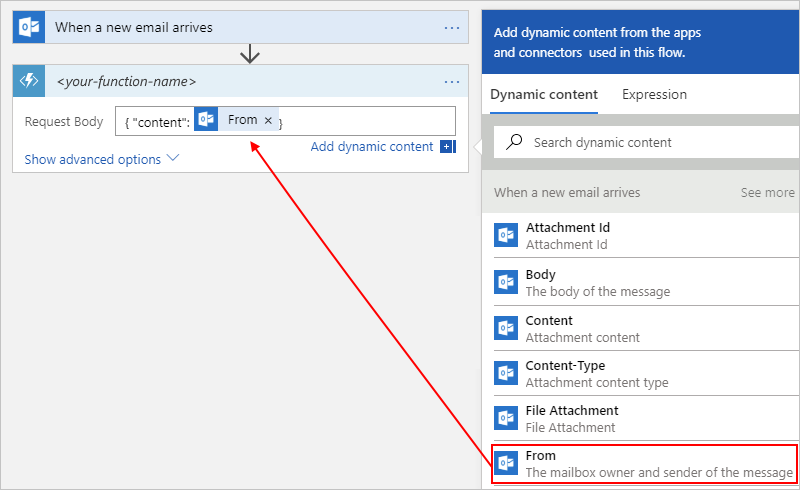
context.body.content

The template code also includes an input variable, which stores the value from the data parameter so your function can perform operations on that value. Inside JavaScript functions, the data variable is also a shortcut for context.body.

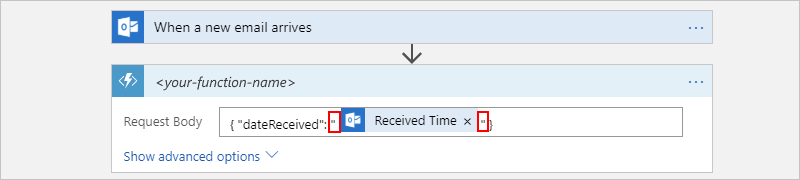
[!NOTE] The body property here applies to the context object and isn't the same as the **Body** token from an action's output, which you might also pass to your function.

1. When you're done, choose **Create**.
2. In the **Request Body** box, provide your function's input, which must be formatted as a JavaScript Object Notation (JSON) object.

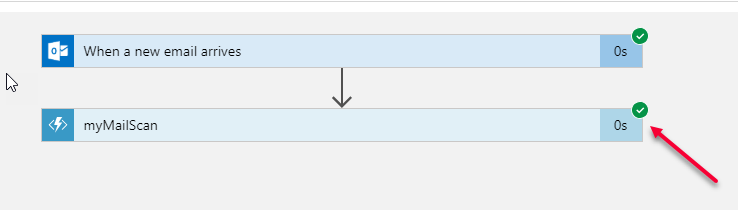
This input is the context object or message that your logic app sends to your function. When you click in the **Request Body** field, the dynamic content list appears so you can select tokens for outputs from previous steps. This example specifies that the context payload contains a property named content that has the **From** token's value from the email trigger:

[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/function-request-body-example.png)

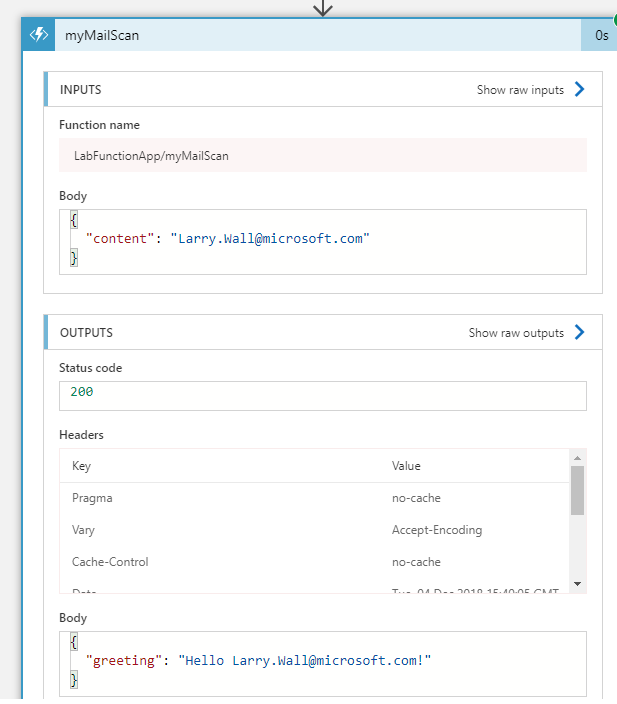
Here, the context object isn't cast as a string, so the object's content gets added directly to the JSON payload. However, when the context object isn't a JSON token that passes a string, a JSON object, or a JSON array, you get an error. So, if this example used the **Received Time** token instead, you can cast the context object as a string by adding double-quotation marks:

[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/function-request-body-string-cast-example.png)

1. To specify other details such as the method to use, request headers, or query parameters, choose **Show advanced options**.
2. Make sure you save the logic app and then click on the breadcrumb trail that takes you back to the previous screen in the portal (the logic app blade).
3. Click Run Trigger and then send yourself an email.
4. When you see the succeeded status appear in the Summary area, click on the Succeeded link. You will be able to see that the workflow ran to completion via the green checkmarks on each action.



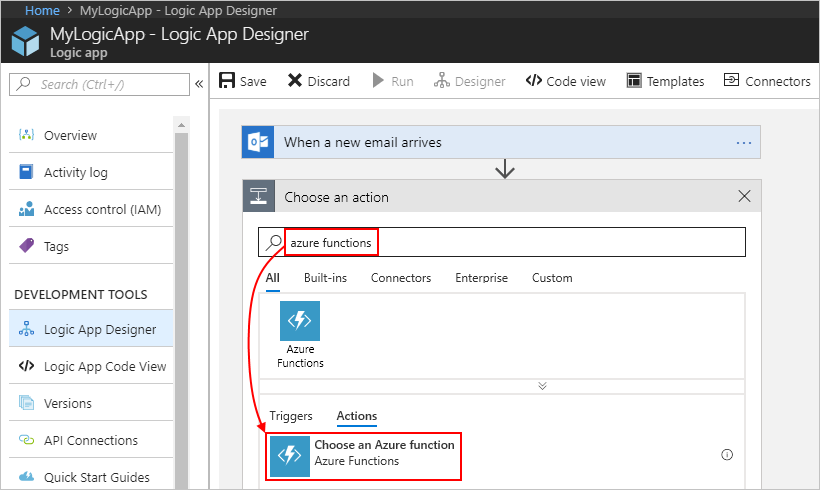
1. Click on your function action and it will expand to show both your input and output to the function.



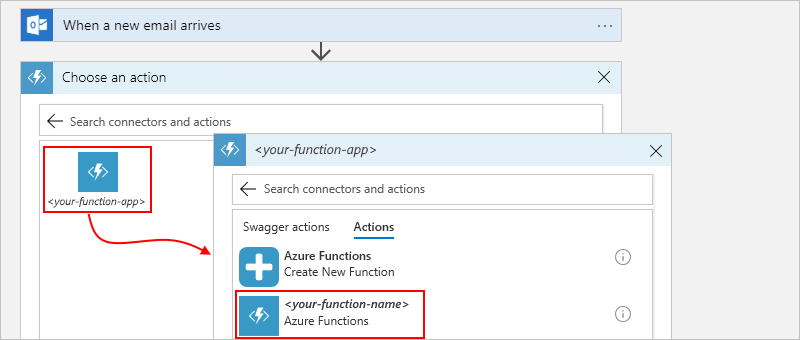
## Add existing functions to logic apps

(Optional if you already have another function, or you can show the steps and you’ll see the function above in the list). To call existing Azure functions from your logic apps, you can add Azure functions like any other action in the Logic App Designer.

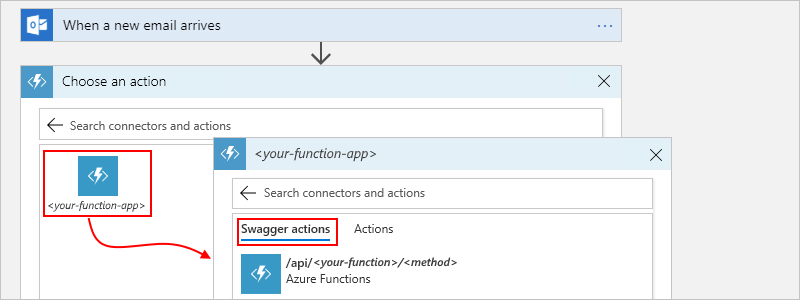
1. In the [Azure portal](https://portal.azure.com), open your logic app in the Logic App Designer.
2. Under the step where you want to add the function, choose **New step** > **Add an action**.
3. In the search box, enter "azure functions" as your filter. From the actions list, select this action: **Choose an Azure function - Azure Functions**

[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/find-azure-functions-action.png)

1. From the function apps list, select your function app. After the functions list appears, select your function.

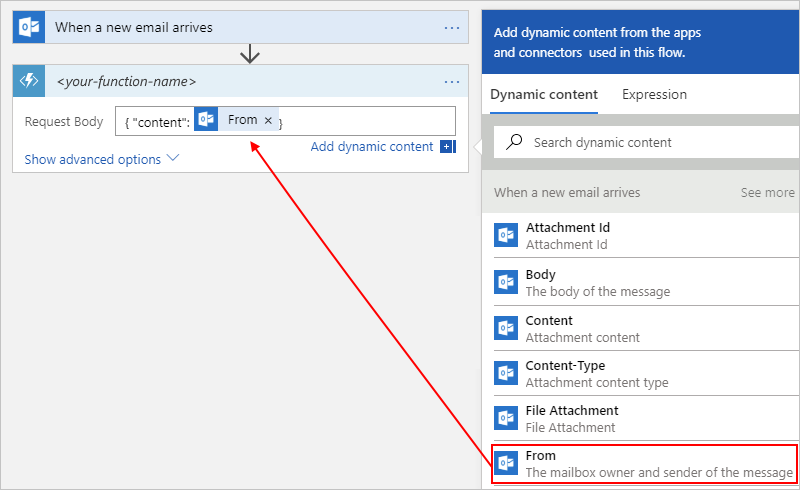
[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/select-function-app-existing-function.png)

For functions that have API definitions (Swagger descriptions) and are [set up so your logic app can find and access those functions](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/logic-apps-azure-functions.md#function-swagger), you can select **Swagger actions**:

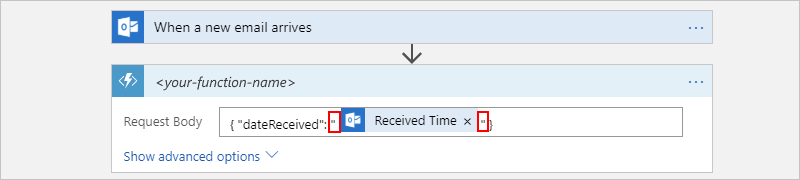
[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/select-function-app-existing-function-swagger.png)

1. In the **Request Body** box, provide your function's input, which must be formatted as a JavaScript Object Notation (JSON) object.

This input is the context object or message that your logic app sends to your function. When you click in the **Request Body** field, the dynamic content list appears so you can select tokens for outputs from previous steps. This example specifies that the context payload contains a property named content that has the **From** token's value from the email trigger:

[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/function-request-body-example.png)

Here, the context object isn't cast as a string, so the object's content gets added directly to the JSON payload. However, when the context object isn't a JSON token that passes a string, a JSON object, or a JSON array, you get an error. So, if this example used the **Received Time** token instead, you can cast the context object as a string by adding double-quotation marks:

[](https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/logic-apps/media/logic-apps-azure-functions/function-request-body-string-cast-example.png)

1. To specify other details such as the method to use, request headers, or query parameters, choose **Show advanced options**.