Strategies for testing your code in Azure Functions

02/02/2017 • 12 minutes to read • Contributors all

In this article

This topic demonstrates the various ways to test functions, including using the following general approaches:

- HTTP-based tools, such as cURL, Postman, and even a web browser for web-based triggers
- Azure Storage Explorer, to test Azure Storage-based triggers
- Test tab in the Azure Functions portal
- Timer-triggered function
- Testing application or framework

All these testing methods use an HTTP trigger function that accepts input through either a query string parameter or the request body. You create this function in the first section.

Create a function for testing

For most of this tutorial, we use a slightly modified version of the HttpTrigger JavaScript function template that is available when you create a function. If you need help creating a function, review this tutorial. Choose the **HttpTrigger- JavaScript** template when creating the test function in the Azure portal.

The default function template is basically a "hello world" function that echoes back the name from the request body or query string parameter, `name=<your name>`. We'll update the code to also allow you to provide the name and an address as JSON content in the request body. Then the function echoes these back to the client when available.

Update the function with the following code, which we will use for testing:

JavaScript Copy

```
module.exports = function (context, req) {
    context.log("HTTP trigger function processed a request. RequestUri=%s", req.origina
    context.log("Request Headers = " + JSON.stringify(req.headers));
    var res;

    if (req.query.name || (req.body && req.body.name)) {
        if (typeof req.query.name != "undefined") {
            context.log("Name was provided as a query string param...");
            res = ProcessNewUserInformation(context, req.query.name);
        }
        else {
            context.log("Processing user info from request body...");
            res = ProcessNewUserInformation(context, req.body.name, req.body.address);
        }
    }
    else {
        res = {
            status: 400,
            body: "Please pass a name on the query string or in the request body"
        };
    }
    context.done(null, res);
};
function ProcessNewUserInformation(context, name, address) {
    context.log("Processing user information...");
    context.log("name = " + name);
    var echoString = "Hello " + name;
    var res;

    if (typeof address != "undefined") {
        echoString += "\n" + "The address you provided is " + address;
        context.log("address = " + address);
    }
    res = {
        // status: 200, /* Defaults to 200 */
        body: echoString
    };
    return res;
}
```

Test a function with tools

Outside the Azure portal, there are various tools that you can use to trigger your functions for testing. These include HTTP testing tools (both UI-based and command line), Azure Storage access tools, and even a simple web browser.

Test with a browser

The web browser is a simple way to trigger functions via HTTP. You can use a browser for GET requests that do not require a body payload, and that use only query string parameters. 2

To test the function we defined earlier, copy the **Function Url** from the portal. It has the following form:

Copy

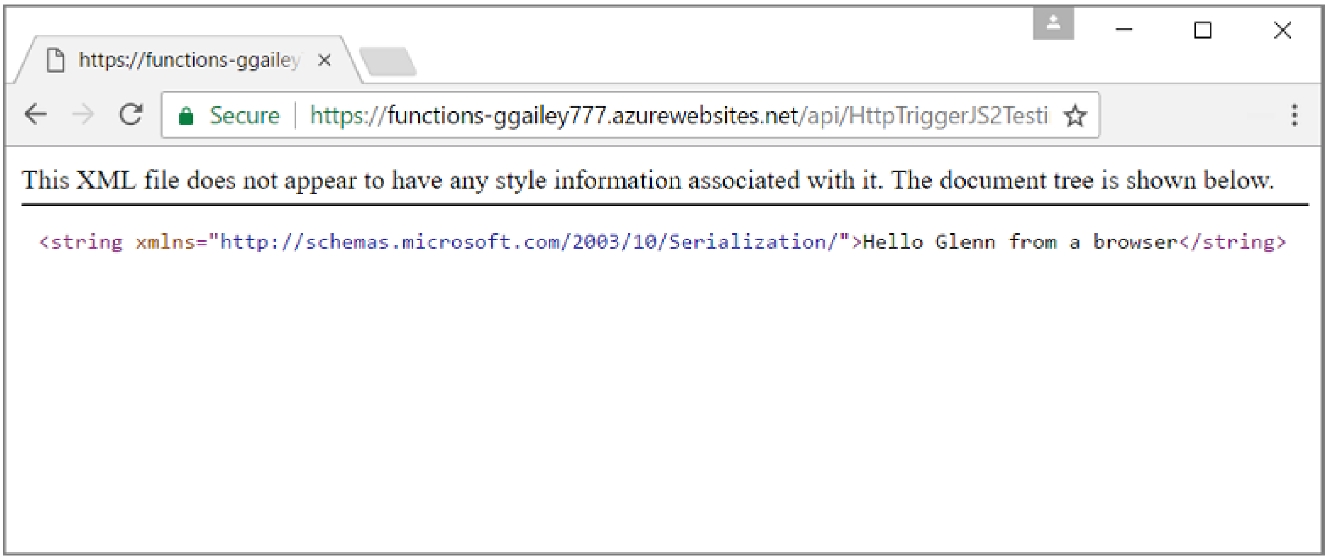
```
https://<Your Function App>.azurewebsites.net/api/<Your Function Name>?code=<your acces
```

Append the `name` parameter to the query string. Use an actual name for the `<Enter a name here>` placeholder.

Copy

```
https://<Your Function App>.azurewebsites.net/api/<Your Function Name>?code=<your acces
```

Paste the URL into your browser, and you should get a response similar to the following.


https://functions-ggailey
Secure
https://functions-ggailey777.azurewebsites.net/api/HttpTriggerJS2Testi

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Hello Glenn from a browser</string>
```

This example is the Chrome browser, which wraps the returned string in XML. Other browsers display just the string value.

In the portal **Logs** window, output similar to the following is logged in executing the function:

Copy

```
2016-03-23T07:34:59  Welcome, you are now connected to log-streaming service.
2016-03-23T07:35:09.195 Function started (Id=61a8c5a9-5e44-4da0-909d-91d293f20445)
2016-03-23T07:35:10.338 Node.js HTTP trigger function processed a request. RequestUri=h
2016-03-23T07:35:10.338 Request Headers = {"cache-control":"max-age=0","connection":"Ke
2016-03-23T07:35:10.338 Name was provided as a query string param.
2016-03-23T07:35:10.338 Processing User Information...
2016-03-23T07:35:10.369 Function completed (Success, Id=61a8c5a9-5e44-4da0-909d-91d293f
```

Test with Postman

The recommended tool to test most of your functions is Postman, which integrates with the Chrome browser. To install Postman, see Get Postman. Postman provides control over many more attributes of an HTTP request.

> Tip
>
> Use the HTTP testing tool that you are most comfortable with. Here are some alternatives to Postman:
>
> - Fiddler
> - Paw

To test the function with a request body in Postman:

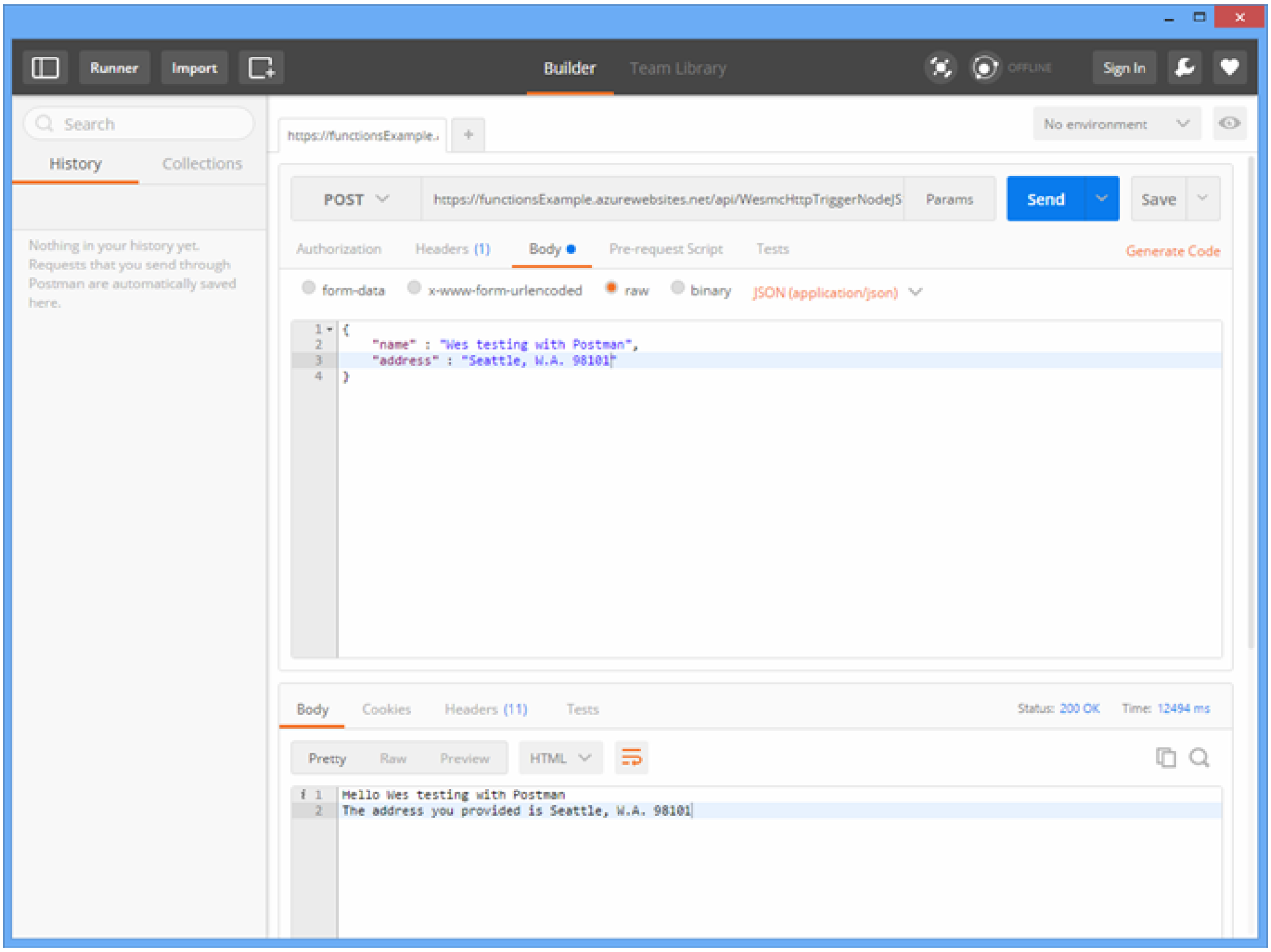
1. Start Postman from the **Apps** button in the upper-left corner of a Chrome browser window. 1
2. Copy your **Function Url**, and paste it into Postman. It includes the access code query string parameter.
3. Change the HTTP method to **POST**.
4. Click **Body** > **raw**, and add a JSON request body similar to the following:

   JSON Copy

   ```json
   {
       "name" : "Wes testing with Postman",
       "address" : "Seattle, WA 98101"
   }
   ```

5. Click **Send**.

The following image shows testing the simple echo function example in this tutorial.



Nothing in your history yet. Requests that you send through Postman are automatically saved here.

POST https://functionsExample.azurewebsites.net/api/WesmcHttpTriggerNodeJS

● raw JSON (application/json)

```
{
    "name" : "Wes testing with Postman",
    "address" : "Seattle, W.A. 98101"
}
```

Status: 200 OK Time: 12494 ms

```
Hello Wes testing with Postman
The address you provided is Seattle, W.A. 98101
```

In the portal **Logs** window, output similar to the following is logged in executing the function:

```
2016-03-23T08:04:51  Welcome, you are now connected to log-streaming service.
2016-03-23T08:04:57.107 Function started (Id=dc5db8b1-6f1c-4117-b5c4-f6b602d538f7)
2016-03-23T08:04:57.763 HTTP trigger function processed a request. RequestUri=https://f
2016-03-23T08:04:57.763 Request Headers = {"cache-control":"no-cache","connection":"Kee
2016-03-23T08:04:57.763 Processing user info from request body...
2016-03-23T08:04:57.763 Processing User Information...
2016-03-23T08:04:57.763 name = Wes testing with Postman
2016-03-23T08:04:57.763 address = Seattle, W.A. 98101
2016-03-23T08:04:57.795 Function completed (Success, Id=dc5db8b1-6f1c-4117-b5c4-f6b602d
```

Test with cURL from the command line

Often when you're testing software, it's not necessary to look any further than the command line to help debug your application. This is no different with testing functions. Note that the cURL is available by default on Linux-based systems. On Windows, you must first download and install the cURL tool.

To test the function that we defined earlier, copy the **Function URL** from the portal. It has the following form:

Copy

```
https://<Your Function App>.azurewebsites.net/api/<Your Function Name>?code=<your acces
```

This is the URL for triggering your function. Test this by using the cURL command on the command line to make a GET (`-G` or `--get`) request against the function:

Copy

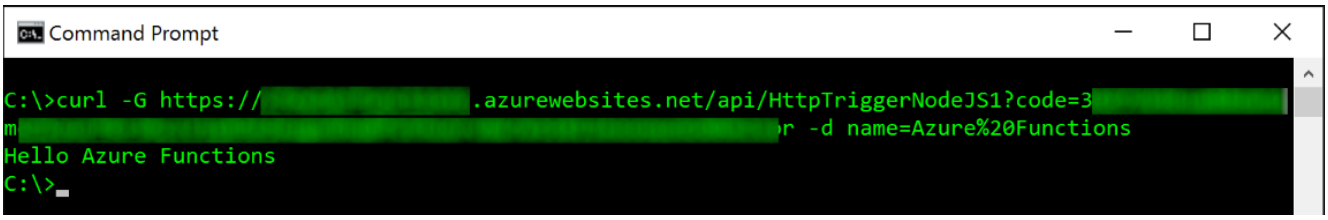
```
curl -G https://<Your Function App>.azurewebsites.net/api/<Your Function Name>?code=<yo
```

This particular example requires a query string parameter, which can be passed as Data (`-d`) in the cURL command:

Copy

```
curl -G https://<Your Function App>.azurewebsites.net/api/<Your Function Name>?code=<yo
```

Run the command, and you see the following output of the function on the command line:



```
C:\>curl -G https://[redacted].azurewebsites.net/api/HttpTriggerNodeJS1?code=3[redacted]
m[redacted]r -d name=Azure%20Functions
Hello Azure Functions
C:\>
```

In the portal **Logs** window, output similar to the following is logged in executing the function:

Copy

```
2016-04-05T21:55:09  Welcome, you are now connected to log-streaming service.
2016-04-05T21:55:30.738 Function started (Id=ae6955da-29db-401a-b706-482fcd1b8f7a)
2016-04-05T21:55:30.738 Node.js HTTP trigger function processed a request. RequestUri=h
2016-04-05T21:55:30.738 Function completed (Success, Id=ae6955da-29db-401a-b706-482fcd1
```

Test a blob trigger by using Storage Explorer

You can test a blob trigger function by using Azure Storage Explorer.

1. In the Azure portal for your function app, create a C#, F# or JavaScript blob trigger function. Set the path to monitor to the name of your blob container. For example:

   Copy

   ```
   files
   ```

2. Click the **+** button to select or create the storage account you want to use. Then click **Create**.

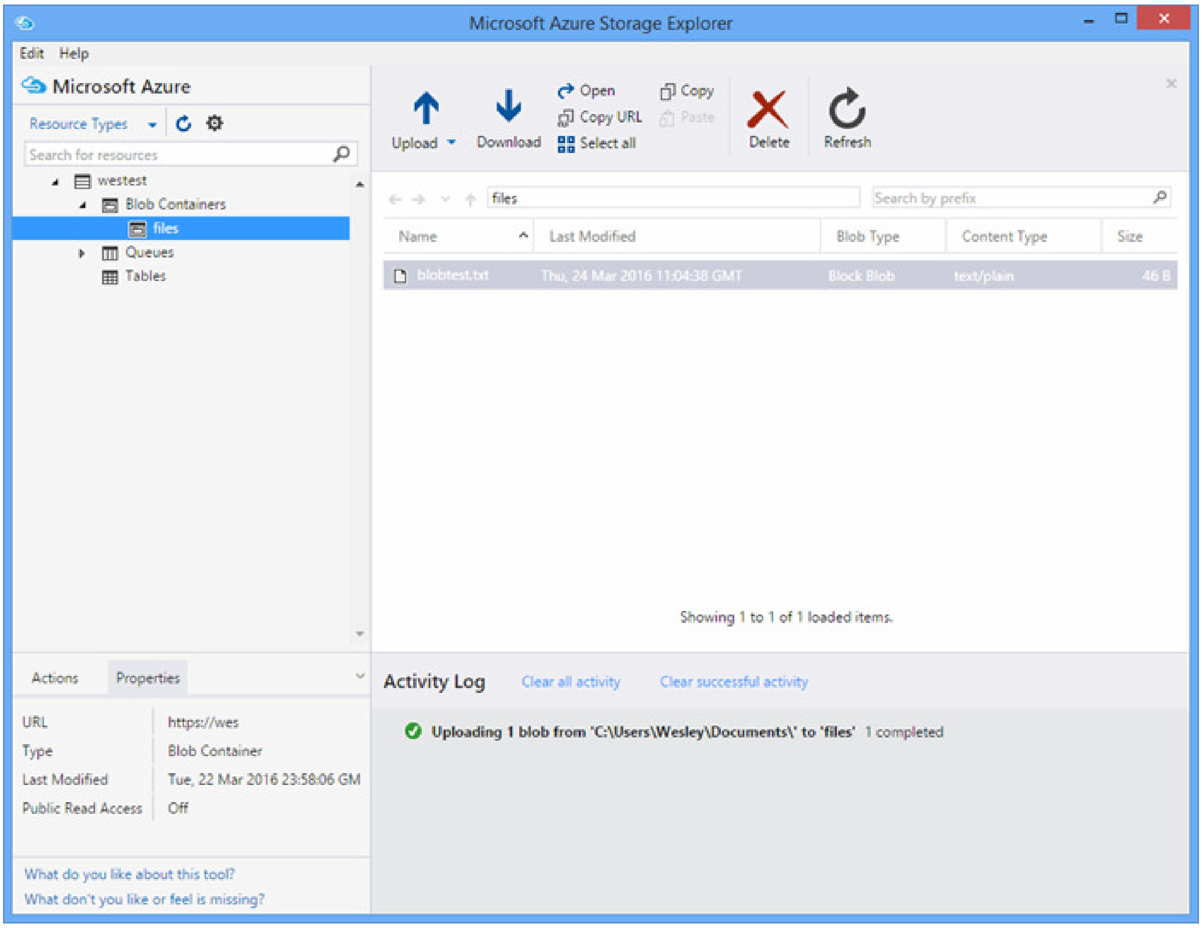
3. Create a text file with the following text, and save it:

   Copy

   ```
   A text file for blob trigger function testing.
   ```

4. Run Azure Storage Explorer, and connect to the blob container in the storage account being monitored.

5. Click **Upload** to upload the text file.

Name	Last Modified	Blob Type	Content Type	Size
blobtest.txt	Thu, 24 Mar 2016 11:04:38 GMT	Block Blob	text/plain	46 B

   Showing 1 to 1 of 1 loaded items.

URL	https://wes
Type	Blob Container
Last Modified	Tue, 22 Mar 2016 23:58:06 GM
Public Read Access	Off

   ```
   Uploading 1 blob from 'C:\Users\Wesley\Documents\' to 'files'  1 completed
   ```

The default blob trigger function code reports the processing of the blob in the logs:

Copy

```
2016-03-24T11:30:10  Welcome, you are now connected to log-streaming service.
2016-03-24T11:30:34.472 Function started (Id=739ebc07-ff9e-4ec4-a444-e479cec2e460)
2016-03-24T11:30:34.472 C# Blob trigger function processed: A text file for blob trigge
2016-03-24T11:30:34.472 Function completed (Success, Id=739ebc07-ff9e-4ec4-a444-e479cec
```

Test a function within functions

The Azure Functions portal is designed to let you test HTTP and timer triggered functions. You can also create functions to trigger other functions that you are testing.

Test with the Functions portal Run button

The portal provides a **Run** button that you can use to do some limited testing. You can provide a request body by using the button, but you can't provide query string parameters or update request headers. 1

Test the HTTP trigger function we created earlier by adding a JSON string similar to the following in the **Request body** field. Then click the **Run** button.

JSON Copy

```
{
    "name" : "Wes testing Run button",
    "address" : "USA"
}
```

In the portal **Logs** window, output similar to the following is logged in executing the function:

Copy

```
2016-03-23T08:03:12  Welcome, you are now connected to log-streaming service.
2016-03-23T08:03:17.357 Function started (Id=753a01b0-45a8-4125-a030-3ad543a89409)
2016-03-23T08:03:18.697 HTTP trigger function processed a request. RequestUri=https://f
2016-03-23T08:03:18.697 Request Headers = {"connection":"Keep-Alive","accept":"*/*","ac
2016-03-23T08:03:18.697 Processing user info from request body...
2016-03-23T08:03:18.697 Processing User Information...
2016-03-23T08:03:18.697 name = Wes testing Run button
2016-03-23T08:03:18.697 address = USA
2016-03-23T08:03:18.744 Function completed (Success, Id=753a01b0-45a8-4125-a030-3ad543a
```

Test with a timer trigger

Some functions can't be adequately tested with the tools mentioned previously. For example, consider a queue trigger function that runs when a message is dropped into Azure Queue storage. You can always write code to drop a message into your queue, and an example of this in a console project is provided later in this article. However, there is another approach you can use that tests functions directly.

You can use a timer trigger configured with a queue output binding. That timer trigger code can then write the test messages to the queue. This section walks through an example.

For more in-depth information on using bindings with Azure Functions, see the Azure Functions developer reference.

Create a queue trigger for testing

To demonstrate this approach, we first create a queue trigger function that we want to test for a queue named `queue-newusers`. This function processes name and address information dropped into Queue storage for a new user.

> ⓘ Note
>
> If you use a different queue name, make sure the name you use conforms to the Naming Queues and MetaData rules. Otherwise, you get an error.

1. In the Azure portal for your function app, click **New Function** > **QueueTrigger - C#**.

2. Enter the queue name to be monitored by the queue function:

   Copy

   ```
   queue-newusers
   ```

3. Click the **+** button to select or create the storage account you want to use. Then click **Create**.
4. Leave this portal browser window open, so you can monitor the log entries for the default queue function template code.

Create a timer trigger to drop a message in the queue

1. Open the Azure portal in a new browser window, and navigate to your function app.

2. Click **New Function** > **TimerTrigger - C#**. Enter a cron expression to set how often the timer code tests your queue function. Then click **Create**. If you want the test to run every 30 seconds, you can use the following CRON expression:

Copy

```
*/30 * * * * *
```

3. Click the **Integrate** tab for your new timer trigger.
4. Under **Output**, click **+ New Output**. Then click **queue** and **Select**.
5. Note the name you use for the **queue message object**. You use this in the timer function code.

Copy

```
myQueue
```

6. Enter the queue name where the message is sent:

Copy

```
queue-newusers
```

7. Click the **+** button to select the storage account you used previously with the queue trigger. Then click **Save**.
8. Click the **Develop** tab for your timer trigger.
9. You can use the following code for the C# timer function, as long as you used the same queue message object name shown earlier. Then click **Save**.

C# Copy

```
using System;

public static void Run(TimerInfo myTimer, out String myQueue, TraceWriter log)
{
    String newUser =
    "{\"name\":\"User testing from C# timer function\",\"address\":\"XYZ\"}";

    log.Verbose($"C# Timer trigger function executed at: {DateTime.Now}");
    log.Verbose($"{newUser}");

    myQueue = newUser;
}
```

At this point, the C# timer function executes every 30 seconds if you used the example cron expression. The logs for the timer function report each execution:

Copy

```
2016-03-24T10:27:02  Welcome, you are now connected to log-streaming service.
2016-03-24T10:27:30.004 Function started (Id=04061790-974f-4043-b851-48bd4ac424d1)
```

```
2016-03-24T10:27:30.004 C# Timer trigger function executed at: 3/24/2016 10:27:30 AM
2016-03-24T10:27:30.004 {"name":"User testing from C# timer function","address":"XYZ"}
2016-03-24T10:27:30.004 Function completed (Success, Id=04061790-974f-4043-b851-48bd4ac
```

In the browser window for the queue function, you can see each message being processed:

Copy

```
2016-03-24T10:27:06  Welcome, you are now connected to log-streaming service.
2016-03-24T10:27:30.607 Function started (Id=e304450c-ff48-44dc-ba2e-1df7209a9d22)
2016-03-24T10:27:30.607 C# Queue trigger function processed: {"name":"User testing from
2016-03-24T10:27:30.607 Function completed (Success, Id=e304450c-ff48-44dc-ba2e-1df7209
```

Test a function with code

You may need to create an external application or framework to test your functions.

Test an HTTP trigger function with code: Node.js

You can use a Node.js app to execute an HTTP request to test your function. Make sure to set:

- The `host` in the request options to your function app host.
- Your function name in the `path`.
- Your access code (`<your code>`) in the `path`.

Code example:

JavaScript

Copy

```javascript
var http = require("http");

var nameQueryString = "name=Wes%20Query%20String%20Test%20From%20Node.js";

var nameBodyJSON = {
    name : "Wes testing with Node.JS code",
    address : "Dallas, T.X. 75201"
};

var bodyString = JSON.stringify(nameBodyJSON);

var options = {
  host: "functions841def78.azurewebsites.net",
  //path: "/api/HttpTriggerNodeJS2?code=sc1wt62opn7k9buhrm8jpds4ikxvvj42m5ojdt0p91lz5jn
  path: "/api/HttpTriggerNodeJS2?code=sc1wt62opn7k9buhrm8jpds4ikxvvj42m5ojdt0p91lz5jnhf
  method: "POST",
  headers : {
      "Content-Type":"application/json",
      "Content-Length": Buffer.byteLength(bodyString)
    }
};

callback = function(response) {
  var str = ""
  response.on("data", function (chunk) {
    str += chunk;
  });

  response.on("end", function () {
    console.log(str);
  });
}

var req = http.request(options, callback);
console.log("*** Sending name and address in body ***");
console.log(bodyString);
req.end(bodyString);
```

Output:

Copy

```
C:\Users\Wesley\testing\Node.js>node testHttpTriggerExample.js
*** Sending name and address in body ***
{"name" : "Wes testing with Node.JS code","address" : "Dallas, T.X. 75201"}
Hello Wes testing with Node.JS code
The address you provided is Dallas, T.X. 75201
```

In the portal **Logs** window, output similar to the following is logged in executing the function: 1

Copy

```
2016-03-23T08:08:55  Welcome, you are now connected to log-streaming service.
2016-03-23T08:08:59.736 Function started (Id=607b891c-08a1-427f-910c-af64ae4f7f9c)
2016-03-23T08:09:01.153 HTTP trigger function processed a request. RequestUri=http://fu
2016-03-23T08:09:01.153 Request Headers = {"connection":"Keep-Alive","host":"functionsE
2016-03-23T08:09:01.153 Name not provided as query string param. Checking body...
2016-03-23T08:09:01.153 Request Body Type = object
2016-03-23T08:09:01.153 Request Body = [object Object]
2016-03-23T08:09:01.153 Processing User Information...
2016-03-23T08:09:01.215 Function completed (Success, Id=607b891c-08a1-427f-910c-af64ae4
```

Test a queue trigger function with code: C#

We mentioned earlier that you can test a queue trigger by using code to drop a message in your queue. The following example code is based on the C# code presented in the Getting started with Azure Queue storage tutorial. Code for other languages is also available from that link.

To test this code in a console app, you must:

- Configure your storage connection string in the app.config file.
- Pass a `name` and `address` as parameters to the app. For example, `C:\myQueueConsoleApp\test.exe "Wes testing queues" "in a console app"`. (This code accepts the name and address for a new user as command-line arguments during runtime.)

Example C# code:

C#

Copy

```
static void Main(string[] args)
{
    string name = null;
    string address = null;
    string queueName = "queue-newusers";
    string JSON = null;

    if (args.Length > 0)
    {
        name = args[0];
    }
    if (args.Length > 1)
    {
        address = args[1];
    }

    // Retrieve storage account from connection string
    CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConfigurationManager

    // Create the queue client
    CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

    // Retrieve a reference to a queue
    CloudQueue queue = queueClient.GetQueueReference(queueName);

    // Create the queue if it doesn't already exist
    queue.CreateIfNotExists();

    // Create a message and add it to the queue.
    if (name != null)
    {
        if (address != null)
            JSON = String.Format("{{\"name\":\"{0}\",\"address\":\"{1}\"}}", name, addr
        else
```