Create a serverless API using Azure Functions

05/04/2017 • 7 minutes to read • Contributors all

In this article

Prerequisites
Customize your HTTP function
Proxies overview
Create your first proxy
Create a mock API
Next steps

In this tutorial, you will learn how Azure Functions allows you to build highly scalable APIs. Azure Functions comes with a collection of built-in HTTP triggers and bindings, which make it easy to author an endpoint in a variety of languages, including Node.JS, C#, and more. In this tutorial, you will customize an HTTP trigger to handle specific actions in your API design. You will also prepare for growing your API by integrating it with Azure Functions Proxies and setting up mock APIs. All of this is accomplished on top of the Functions serverless compute environment, so you don't have to worry about scaling resources - you can just focus on your API logic.

Prerequisites

This topic uses as its starting point the resources created in Create your first function from the Azure portal. If you haven't already done so, please complete these steps now to create your function app.

The resulting function will be used for the rest of this tutorial.

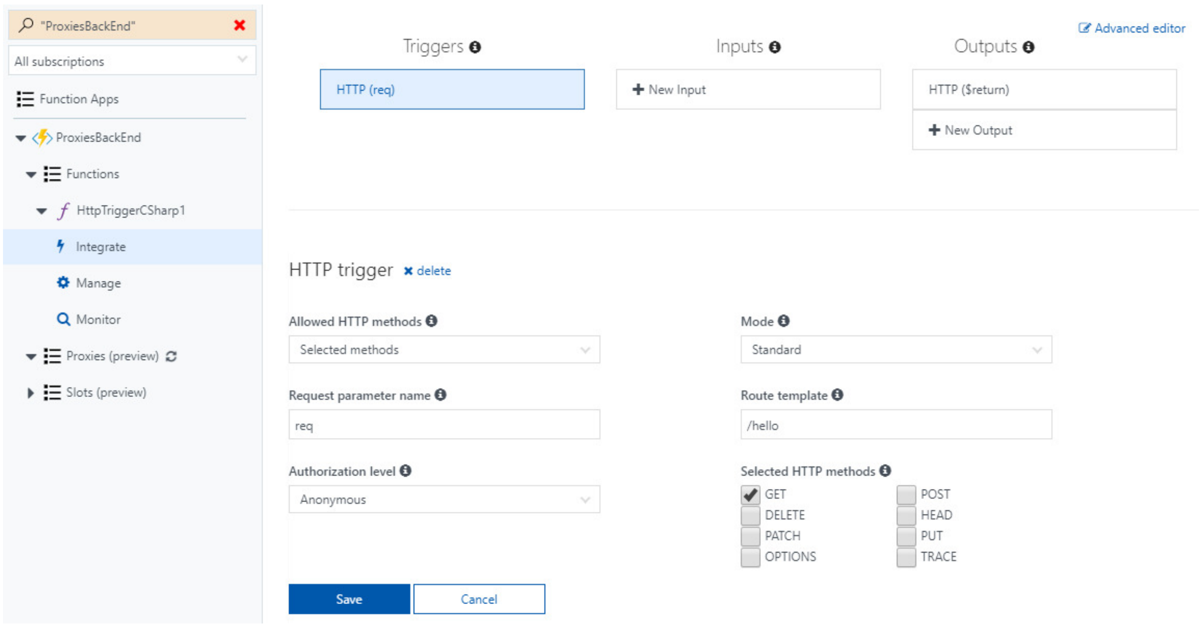
Sign in to Azure

Open the Azure portal. To do this, sign in to https://portal.azure.com with your Azure account.

Customize your HTTP function

By default, your HTTP-triggered function is configured to accept any HTTP method. There is also a default URL of the form `http://<yourapp>.azurewebsites.net/api/<funcname>?code=<functionkey>`. If you followed the quickstart, then `<funcname>` probably looks something like "HttpTriggerJS1". In this section, you will modify the function to respond only to GET requests against `/api/hello` route instead.

1. Navigate to your function in the Azure portal. Select **Integrate** in the left navigation.



HTTP trigger ✖ delete

Allowed HTTP methods	Selected methods
Mode	Standard
Request parameter name	req
Route template	/hello
Authorization level	Anonymous

Selected HTTP methods

✓ GET	☐ POST
☐ DELETE	☐ HEAD
☐ PATCH	☐ PUT
☐ OPTIONS	☐ TRACE

2. Use the HTTP trigger settings as specified in the table.

Field	Sample value	Description
Allowed HTTP methods	Selected methods	Determines what HTTP methods may be used to invoke this function
Selected HTTP methods	GET	Allows only selected HTTP methods to be used to invoke this function
Route template	/hello	Determines what route is used to invoke this function
Authorization Level	Anonymous	Optional: Makes your function accessible without an API key

ⓘ Note

Note that you did not include the `/api` base path prefix in the route template, as this is handled by a global setting.

3. Click **Save**.

You can learn more about customizing HTTP functions in Azure Functions HTTP and webhook bindings.

Test your API

Next, test your function to see it working with the new API surface.

1. Navigate back to the development page by clicking on the function's name in the left navigation.
2. Click **Get function URL** and copy the URL. You should see that it uses the `/api/hello` route now.
3. Copy the URL into a new browser tab or your preferred REST client. Browsers will use GET by default.
4. Run the function and confirm that it is working. You may need to provide the "name" parameter as a query string to satisfy the quickstart code.
5. You can also try calling the endpoint with another HTTP method to confirm that the function is not executed. For this, you will need to use a REST client, such as cURL, Postman, or Fiddler.

Proxies overview

In the next section, you will surface your API through a proxy. Azure Functions Proxies is a preview feature that allows you to forward requests to other resources. You define an HTTP endpoint just like with HTTP trigger, but instead of writing code to execute when that endpoint is called, you provide a URL to a remote implementation. This allows you to compose multiple API sources into a single API surface which is easy for clients to consume. This is particularly useful if you wish to build your API as microservices.

A proxy can point to any HTTP resource, such as:

- Azure Functions
- API apps in Azure App Service
- Docker containers in App Service on Linux
- Any other hosted API

To learn more about proxies, see Working with Azure Functions Proxies (preview).

Create your first proxy

In this section, you will create a new proxy which serves as a frontend to your overall API.

Setting up the frontend environment

Repeat the steps to Create a function app to create a new function app in which you will create your proxy. This new app's URL will serve as the frontend for our API, and the function app you were previously editing will serve as a backend.

1. Navigate to your new frontend function app in the portal.
2. Select **Settings**. Then toggle **Enable Azure Functions Proxies (preview)** to "On".
3. Select **Platform Settings** and choose **Application Settings**.
4. Scroll down to **App settings** and create a new setting with key "HELLO_HOST". Set its value to the host of your backend function app, such as `<YourBackendApp>.azurewebsites.net`. This is part of the URL that you copied earlier when testing your HTTP function. You'll reference this setting in the configuration later.

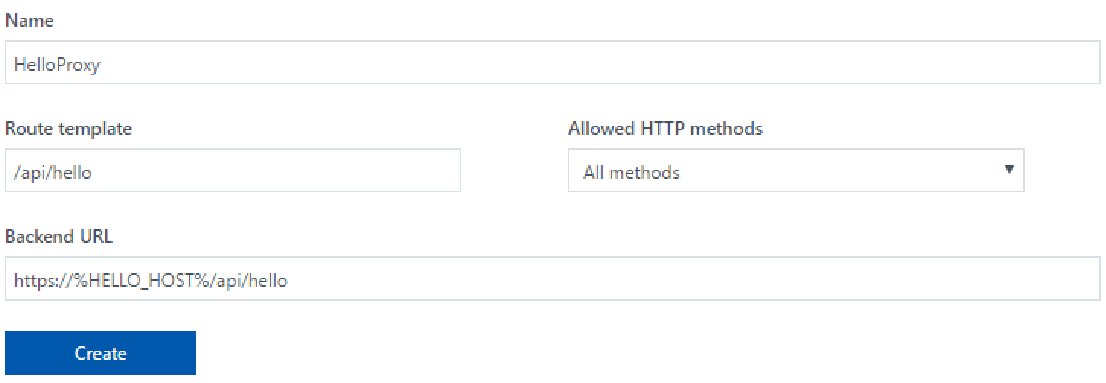
   ⓘ Note

   App settings are recommended for the host configuration to prevent a hard-coded environment dependency for the proxy. Using app settings means that you can move the proxy configuration between environments, and the environment-specific app settings will be applied.

5. Click **Save**.

Creating a proxy on the frontend

1. Navigate back to your frontend function app in the portal.
2. In the left-hand navigation, click the plus sign '+' next to "Proxies (preview)".



Name	HelloProxy
Route template	/api/hello
Allowed HTTP methods	All methods
Backend URL	https://%HELLO_HOST%/api/hello

3. Use proxy settings as specified in the table.

Field	Sample value	Description

Field	Sample value	Description
Name	HelloProxy	A friendly name used only for management
Route template	/api/hello	Determines what route is used to invoke this proxy
Backend URL	https://%HELLO_HOST%/api/hello	Specifies the endpoint to which the request should be proxied

4. Note that Proxies does not provide the `/api` base path prefix, and this must be included in the route template.
5. The `%HELLO_HOST%` syntax will reference the app setting you created earlier. The resolved URL will point to your original function.
6. Click **Create**.
7. You can try out your new proxy by copying the Proxy URL and testing it in the browser or with your favorite HTTP client.

    a. For an anonymous function use:

    i.
    ```
    https://YOURPROXYAPP.azurewebsites.net/api/hello?
    name="Proxies"
    ```

    b. For a function with authorization use:

    i.
    ```
    https://YOURPROXYAPP.azurewebsites.net/api/hello?
    code=YOURCODE&name="Proxies"
    ```

Create a mock API

Next, you will use a proxy to create a mock API for your solution. This allows client development to progress, without needing the backend fully implemented. Later in development, you could create a new function app which supports this logic and redirect your proxy to it.

To create this mock API, we will create a new proxy, this time using the App Service Editor. To get started, navigate to your function app in the portal. Select **Platform features** and find **App Service Editor**. Clicking this will open the App Service Editor in a new tab.

Select `proxies.json` in the left navigation. This is the file which stores the configuration for all of your proxies. If you use one of the Functions deployment methods, this is the file you will maintain in source control. To learn more about this file, see Proxies advanced configuration.

If you've followed along so far, your proxies.json should look like the following:

JSON Copy

```json
{
    "$schema": "http://json.schemastore.org/proxies",
    "proxies": {
        "HelloProxy": {
            "matchCondition": {
                "route": "/api/hello"
            },
            "backendUri": "https://%HELLO_HOST%/api/hello"
        }
    }
}
```

Next you'll add your mock API. Replace your proxies.json file with the following:

JSON Copy

```json
{
    "$schema": "http://json.schemastore.org/proxies",
    "proxies": {
        "HelloProxy": {
            "matchCondition": {
                "route": "/api/hello"
            },
            "backendUri": "https://%HELLO_HOST%/api/hello"
        },
        "GetUserByName" : {
            "matchCondition": {
                "methods": [ "GET" ],
                "route": "/api/users/{username}"
            },
            "responseOverrides": {
                "response.statusCode": "200",
                "response.headers.Content-Type" : "application/json",
                "response.body": {
                    "name": "{username}",
                    "description": "Awesome developer and master of serverless APIs",
                    "skills": [
                        "Serverless",
                        "APIs",
                        "Azure",
                        "Cloud"
                    ]
                }
            }
        }
    }
}
```