Add an API to Azure Static Web Apps with Azure Functions

05/14/2021 • 6 minutes to read • +9

In this article

You can add serverless APIs to Azure Static Web Apps that are powered by Azure Functions. This article demonstrates how to add and deploy an API to an Azure Static Web Apps site.

Note

The functions provided by default in Static Web Apps are pre-configured to provide secure API endpoints and only support HTTP-triggered functions. See **API support with Azure Functions** for information on how they differ from standalone Azure Functions apps.

Prerequisites

- Azure account with an active subscription.
  - If you don't have an account, you can create one for free .
- Visual Studio Code
- Azure Static Web Apps extension for Visual Studio Code
- Node.js to run the frontend app and API

Create the static web app

Before adding an API, create and deploy a frontend application to Azure Static Web Apps. Use an existing app that you have already deployed or create one by following the Building your first static site with Azure Static Web Apps quickstart.

In Visual Studio Code, open the root of your app's repository. The folder structure contains the source for your frontend app and the Static Web Apps GitHub workflow in *.github/workflows* folder.

Files

```
├── .github
│   └── workflows
│       └── azure-static-web-apps-<DEFAULT_HOSTNAME>.yml
│
└── (folders and files from your static web app)
```

Create the API

You create an Azure Functions project for your static web app's API. By default, the Static Web Apps Visual Studio Code extension creates the project in a folder named *api* at the root of your repository.

1. Press F1 to open the Command Palette.

2. Select **Azure Static Web Apps: Create HTTP Function...**. If you're prompted to install the Azure Functions extension, install it and re-run this command.

3. When prompted, enter the following values:

Prompt	Value
Select a language	**JavaScript**
Provide a function name	**message**

An Azure Functions project is generated with an HTTP triggered function. Your app now has a project structure similar to the following example.

Files Copy

```
├── .github
│   └── workflows
│       └── azure-static-web-apps-<DEFAULT_HOSTNAME>.yml
│
├── api
│   ├── message
│   │   ├── function.json
│   │   └── index.js
│   ├── host.json
│   ├── local.settings.json
│   └── package.json
│
└── (folders and files from your static web app)
```

4. Next, change the `message` function to return a message to the frontend. Update the function in *api/message/index.js* with the following code.

JavaScript Copy

```javascript
module.exports = async function (context, req) {
    context.res.json({
        text: "Hello from the API"
    });
};
```

Tip

You can add more API functions by running the **Azure Static Web Apps: Create HTTP Function...** command again.

Update the frontend app to call the API

Update your frontend app to call the API at `/api/message` and display the response message.

If you used the quickstarts to create the app, use the following instructions to apply the updates.

No Framework Angular React Vue

Update the content of the *index.html* file with the following code to fetch the text from the API function and display it on the screen.

HTML Copy

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>Vanilla JavaScript App</title>
</head>

<body>
    <main>
    <h1>Vanilla JavaScript App</h1>
    <p>Loading content from the API: <b id="name">...</b></p>
    </main>

    <script>
    (async function() {
        const { text } = await( await fetch(`/api/message`)).json();
        document.querySelector('#name').textContent = text;
    }())
    </script>
</body>

</html>
```

Run the frontend and API locally

To run your frontend app and API together locally, Azure Static Web Apps provides a CLI that emulates the cloud environment. The CLI leverages the Azure Functions Core Tools to run the API.

Install command line tools

Ensure you have the necessary command line tools installed.

1. Install Azure Static Web Apps CLI.

Bash Copy

```bash
npm install -g @azure/static-web-apps-cli
```

2. Install Azure Functions Core Tools V3.

Bash Copy

```bash
npm install -g azure-functions-core-tools@3
```

Build frontend app

If your app uses a framework, build the app to generate the output before running the Static Web Apps CLI.

No Framework | Angular | React | Vue

There is no need to build the app.

Start the CLI

Run the frontend app and API together by starting the app with the Static Web Apps CLI. Running the two parts of your application this way allows the CLI to serve your frontend's build output from a folder, and makes the API accessible to the running app.

1. In root of your repository, start the Static Web Apps CLI with the `start` command. Adjust the arguments if your app has a different folder structure.

   No Framework | Angular | React | Vue

   Pass the current folder (`.`) and the API folder (`api`) to the CLI.

   Bash

   ```bash
   swa start . --api api
   ```

2. When the CLI processes start, access your app at `http://localhost:4280/`. Notice how the page calls the API and displays its output, `Hello from the API`.

3. To stop the CLI, type `Ctrl-C`.

Add API location to workflow

Before you can deploy your app to Azure, update your repository's GitHub Actions workflow with the correct location of your API folder.

1. Open your workflow at *.github/workflows/azure-static-web-apps-<DEFAULT-HOSTNAME>.yml*.

2. Search for the property `api_location` and set the value to `api`.

3. Save the file.

Deploy changes

To publish changes to your static web app in Azure, commit and push your code to the remote GitHub repository.

1. Press `F1` to open the Command Palette.

2. Select the **Git: Commit All** command.

3. When prompted for a commit message, enter **add API** and commit all changes to your local git repository.

4. Press `F1` to open the Command Palette.

5. Select the **Git: push** command.

Your changes are pushed to the remote repository in GitHub, triggering the Static Web Apps GitHub Actions workflow to build and deploy your app.

6. Open your repository in GitHub to monitor the status of your workflow run.

7. When the workflow run completes, visit your static web app to view your changes.

Next steps

Configure app settings

Is this page helpful?

Yes No

Recommended content

Configure front-end frameworks with Azure Static Web Apps
Settings for popular front-end frameworks needed for Azure Static Web Apps

API support in Azure Static Web Apps with Azure Functions
Learn what API features Azure Static Web Apps supports

Configure application settings for Azure Static Web Apps
Learn to configure application settings for Azure Static Web Apps

Set up local development for Azure Static Web Apps
Learn to set you your local development environment for Azure Static Web Apps

Show more