Set up local development for Azure Static Web Apps

04/02/2021 • 4 minutes to read •  +2

In this article

When published to the cloud, an Azure Static Web Apps site has many services that work together as if they're the same application. These services include:

- The static web app
- Azure Functions API
- Authentication and authorization services
- Routing and configuration services

These services must communicate with each other, and Azure Static Web Apps handles this integration for you in the cloud.

Running locally, however, these services aren't automatically tied together.

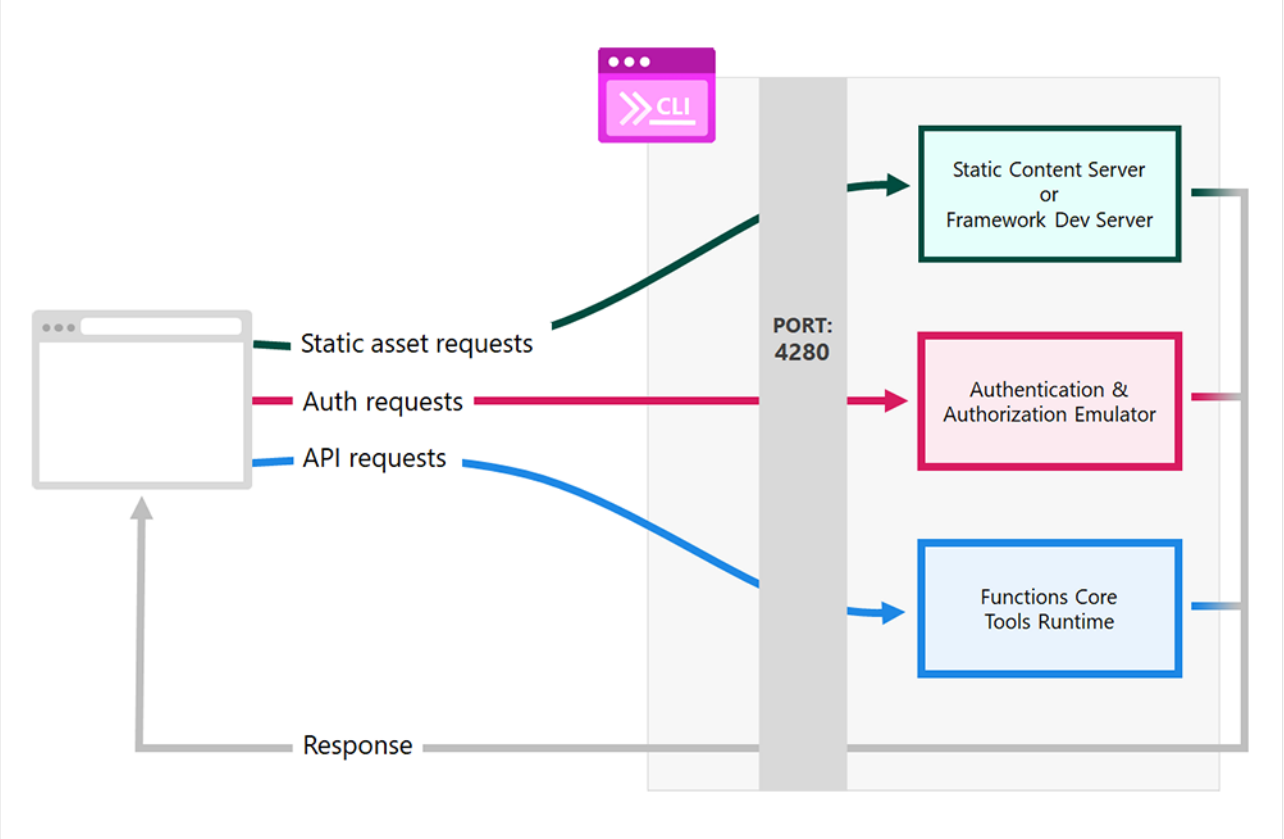
To provide a similar experience as to what you get in Azure, the Azure Static Web Apps CLI provides the following services:

- A local static site server
- A proxy to the front-end framework development server
- A proxy to your API endpoints - available through Azure Functions Core Tools
- A mock authentication and authorization server
- Local routes and configuration settings enforcement

> **Note**
>
> The Azure Static Web Apps CLI is currently a preview feature and doesn't read configuration defined in the *staticwebapp.config.json* file.

How it works

The following chart shows how requests are handled locally.



> **Important**
>
> Navigate to `http://localhost:4280` to access the application served by the CLI.

- **Requests** made to port `4280` are forwarded to the appropriate server depending on the type of request.
- **Static content** requests, such as HTML or CSS, are either handled by the internal CLI static content server, or by the front-end framework server for debugging.
- **Authentication and authorization** requests are handled by an emulator, which provides a fake identity profile to your app.
- **Functions Core Tools runtime** handles requests to the site's API.
- **Responses** from all services are returned to the browser as if they were all a single application.

Prerequisites

- **Existing Azure Static Web Apps site**: If you don't have one, begin with the vanilla-api starter app.
- Node.js **with npm**: Run the Node.js LTS version, which includes access to npm .
- Visual Studio Code : Used for debugging the API application, but not required for the CLI.

Get started

Open a terminal to the root folder of your existing Azure Static Web Apps site.

1. Install the CLI.

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

2. Build your app if required by your application.

Run `npm run build`, or the equivalent command for your project.

3. Change into the output directory for your app. Output folders are often named *build* or something similar.
4. Start the CLI.

   ```
   swa start
   ```

5. Navigate to `http://localhost:4280` to view the app in the browser.

Other ways to start the CLI

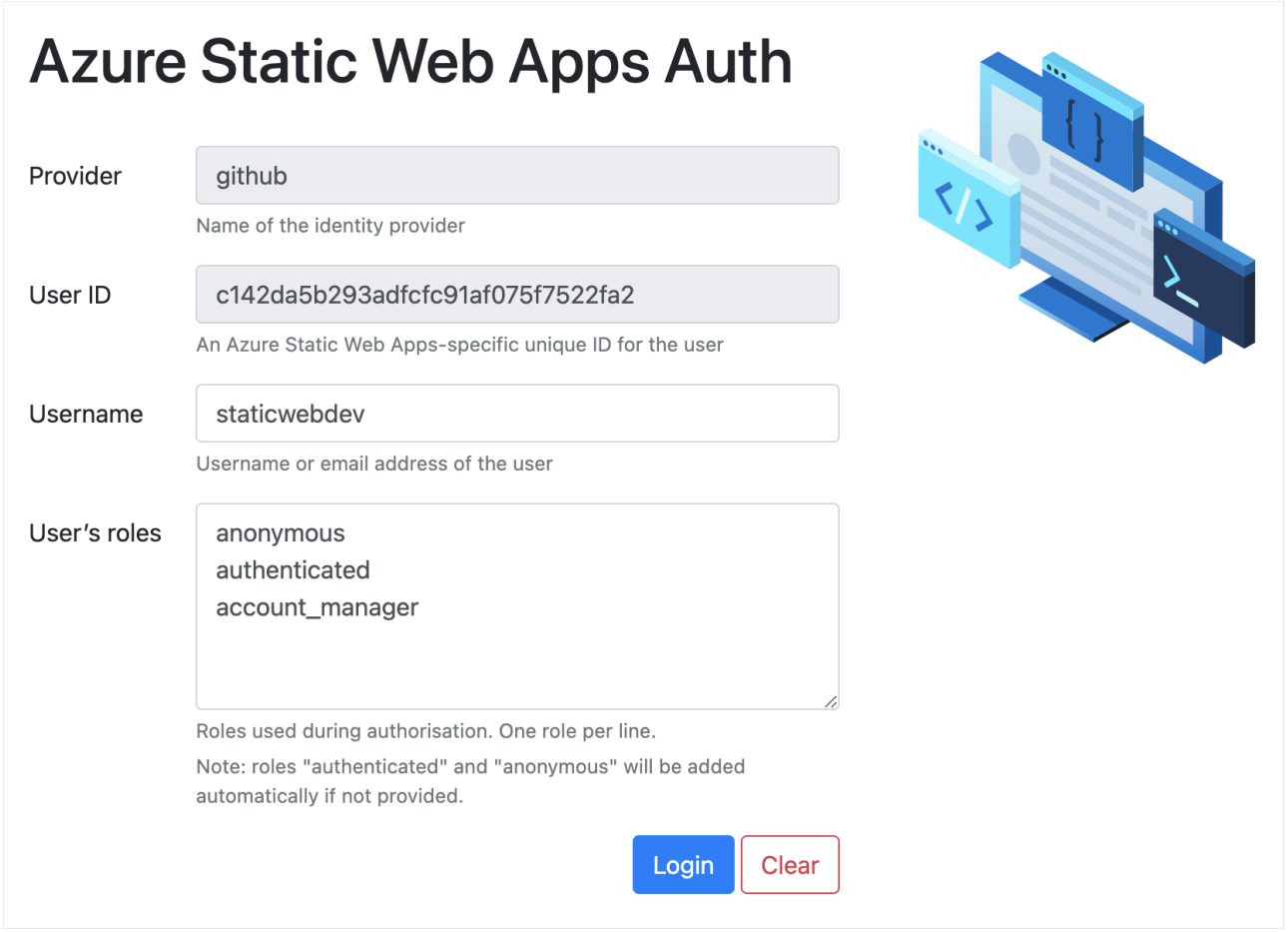
Description	Command
Serve a specific folder	`swa start ./output-folder`
Use a running framework development server	`swa start http://localhost:3000`
Start a Functions app in a folder	`swa start ./output-folder --api ./api`
Use a running Functions app	`swa start ./output-folder --api http://localhost:7071`

Authorization and authentication emulation

The Static Web Apps CLI emulates the security flow implemented in Azure. When a user logs in, you can define a fake identity profile returned to the app.

For instance, when you try to navigate to `/.auth/login/github`, a page is returned that allows you to define an identity profile.

> **Note**
>
> The emulator works with any security provider, not just GitHub.



The emulator provides a page allowing you to provide the following client principal values:

Value	Description
Username	The account name associated with the security provider. This value appears as the `userDetails` property in the client principal and is autogenerated if you don't provide a value.
User ID	Value autogenerated by the CLI.
Roles	A list of role names, where each name is on a new line.

Once logged in:

- You can use the `/.auth/me` endpoint, or a function endpoint to retrieve the user's client principal.
- Navigating to `/.auth/logout` clears the client principal and logs out the mock user.

Debugging

There are two debugging contexts in a static web app. The first is for the static content site, and the second is for API functions. Local debugging is possible by allowing the Static Web Apps CLI to use development servers for one or both of these contexts.

The following steps show you a common scenario that uses development servers for both debugging contexts.

1. Start the static site development server. This command is specific to the front-end framework you're using, but often comes in the form of commands like `npm run build`, `npm start`, or `npm run dev`.

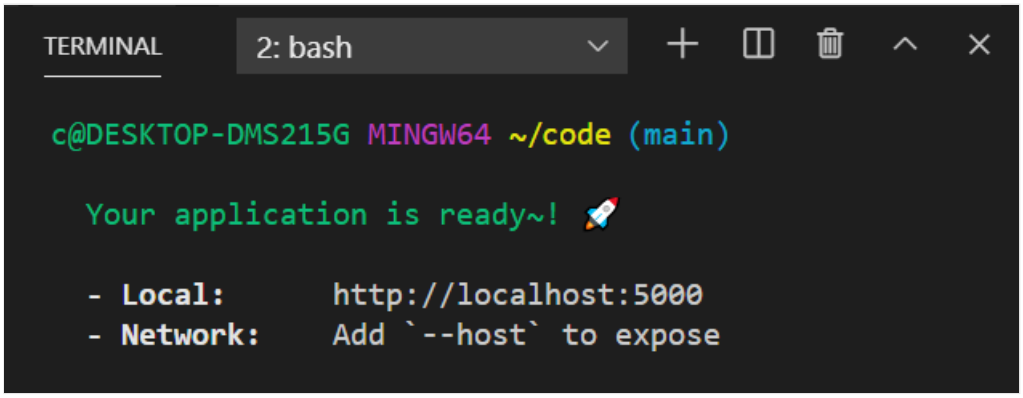
2. Open the API application folder in Visual Studio Code and start a debugging session.

3. Pass the addresses for the static server and API server to the `swa start` command by listing them in order.

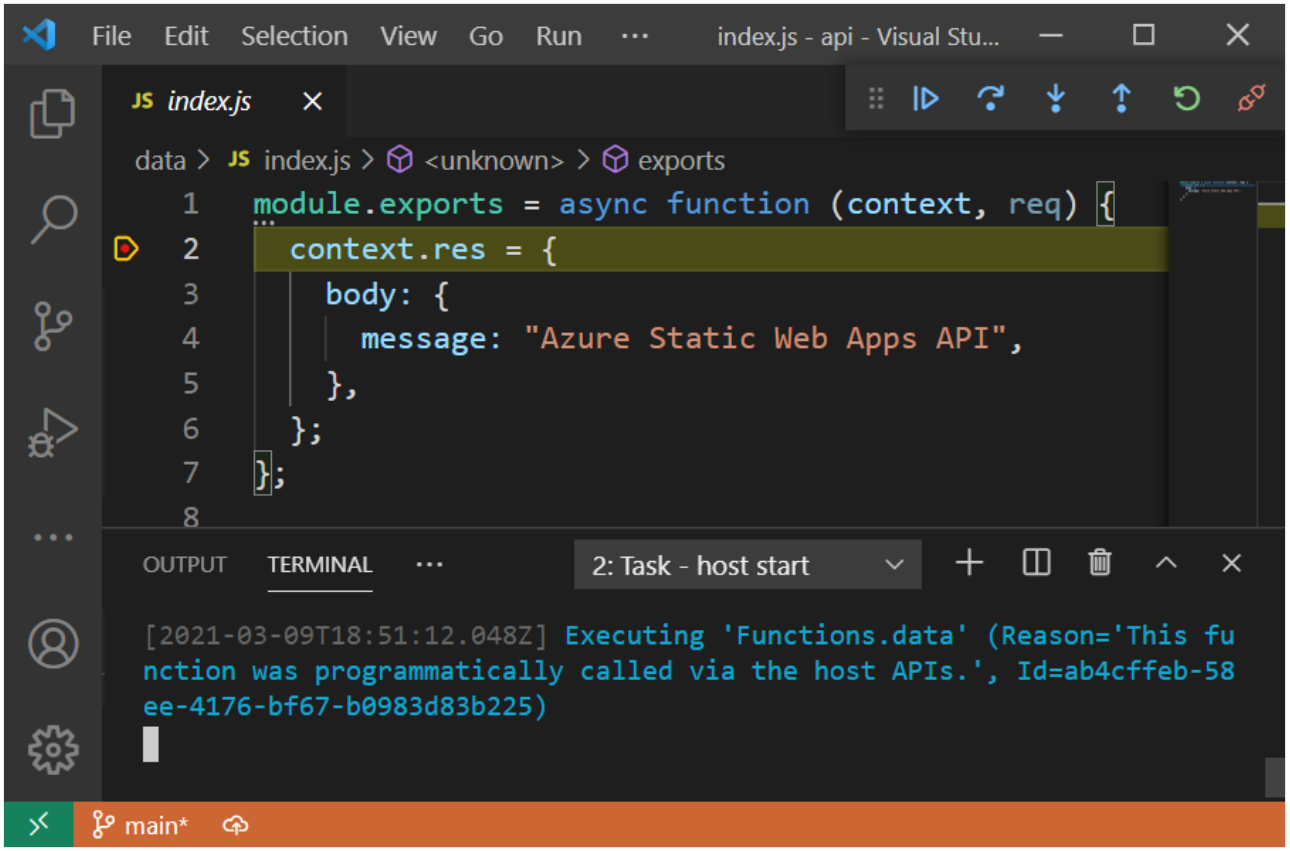
```
swa start http://localhost:<DEV-SERVER-PORT-NUMBER> --api=http://localhost:7071
```

The following screenshots show the terminals for a typical debugging scenario:

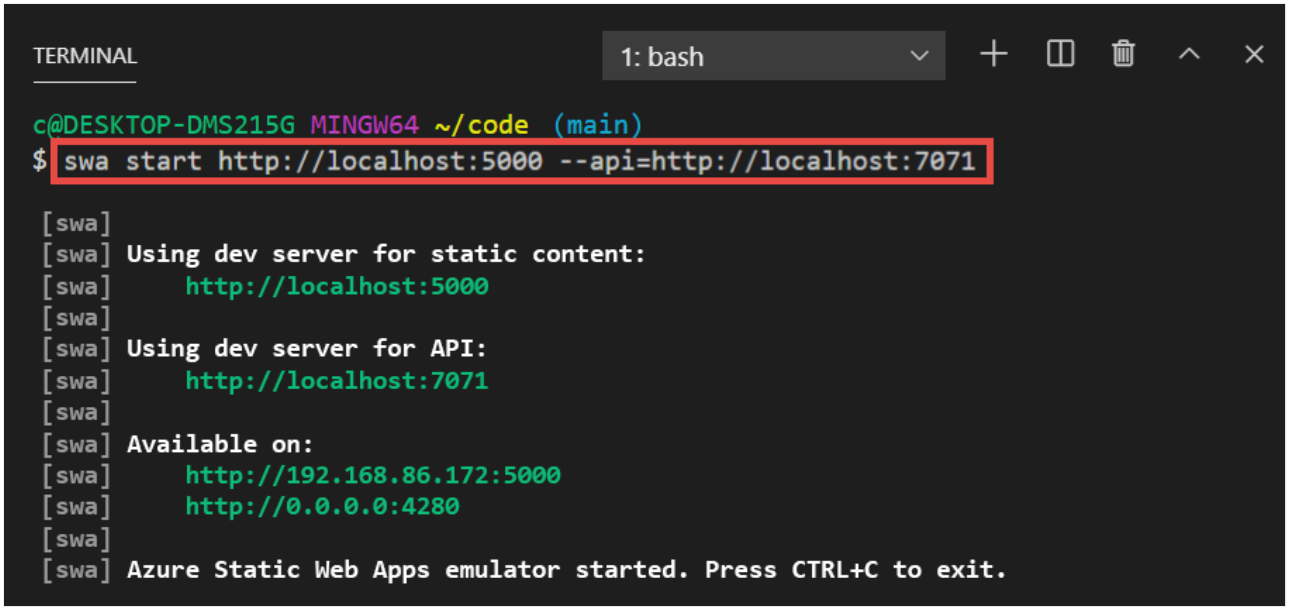
The static content site is running via `npm run dev`.



The Azure Functions API application is running a debug session in Visual Studio Code.



The Static Web Apps CLI is launched using both development servers.



Now requests that go through port `4280` are routed to either the static content development server, or the API debugging session.

For more information on different debugging scenarios, with guidance on how to customize ports and server addresses, see the Azure Static Web Apps CLI repository .

Next steps

Configure your application

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

Add an API to Azure Static Web Apps with Azure Functions

Get started with Azure Static Web Apps by adding a Serverless API to your static web app using Azure Functions.

GitHub Actions workflows for Azure Static Web Apps

Learn how to use GitHub repositories to set up continuous deployment to Azure Static Web Apps.

Show more