Publish an add-in developed with Visual Studio Code

Article • 05/19/2025

This article describes how to publish an Office Add-in that you created using the Yeoman generator and developed with Visual Studio Code (VS Code) or any other editor.

> **Note**
>
> - For information about publishing an Office Add-in that you created using Visual Studio, see Publish your add-in using Visual Studio.
> - The process described in this article doesn't apply to add-ins that use the unified manifest for Microsoft 365. Add-ins created using Microsoft 365 Agents Toolkit use the unified manifest. For information about publishing an add-in that you created using Agents Toolkit, see Deploy Teams app to the cloud and Deploy your first Teams app. The latter article is about Teams tab apps, but it is applicable to Office Add-ins created with Agents Toolkit.

Publishing an add-in for other users to access

The simplest Office Add-in is made up of a manifest file and an HTML page. The manifest file describes the add-in's characteristics, such as its name, what Office applications it can run in, and the URL for the add-in's HTML page. The HTML page is contained in a web app that users interact with when they install and run your add-in within an Office application. You can host the web app of an Office Add-in on any web hosting platform, including Azure.

While you're developing, you can run the add-in on your local web server (`localhost`). When you're ready to publish it for other users to access, you'll need to deploy the web application and update the manifest to specify the URL of the deployed application.

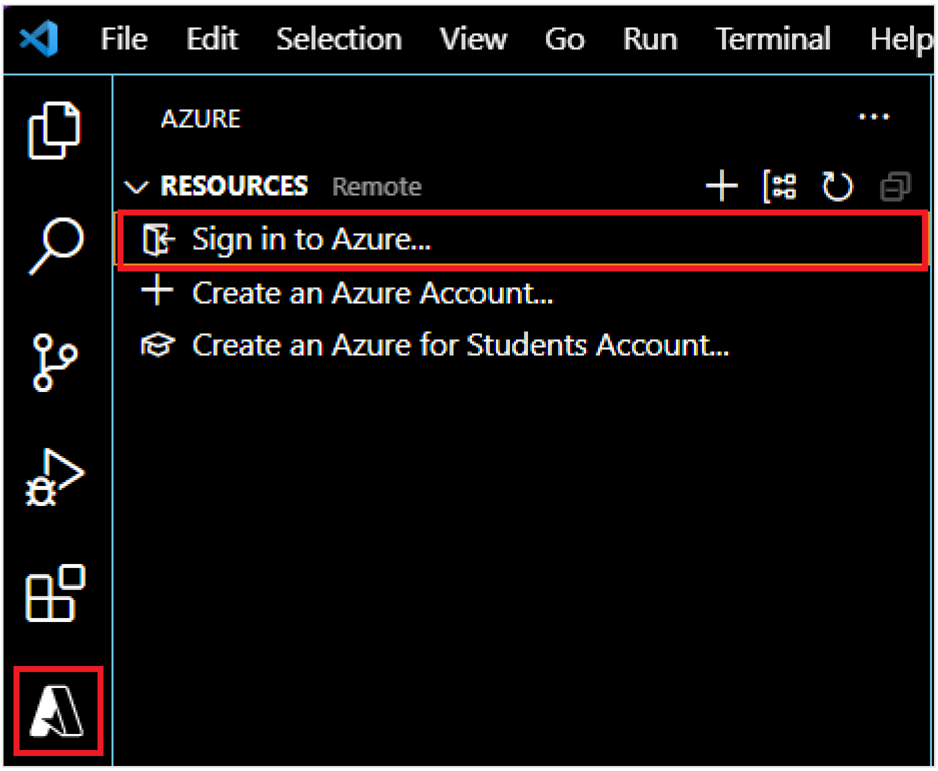
When your add-in is working as desired, you can publish it directly through Visual Studio Code using the Azure Storage extension.

Using Visual Studio Code to publish

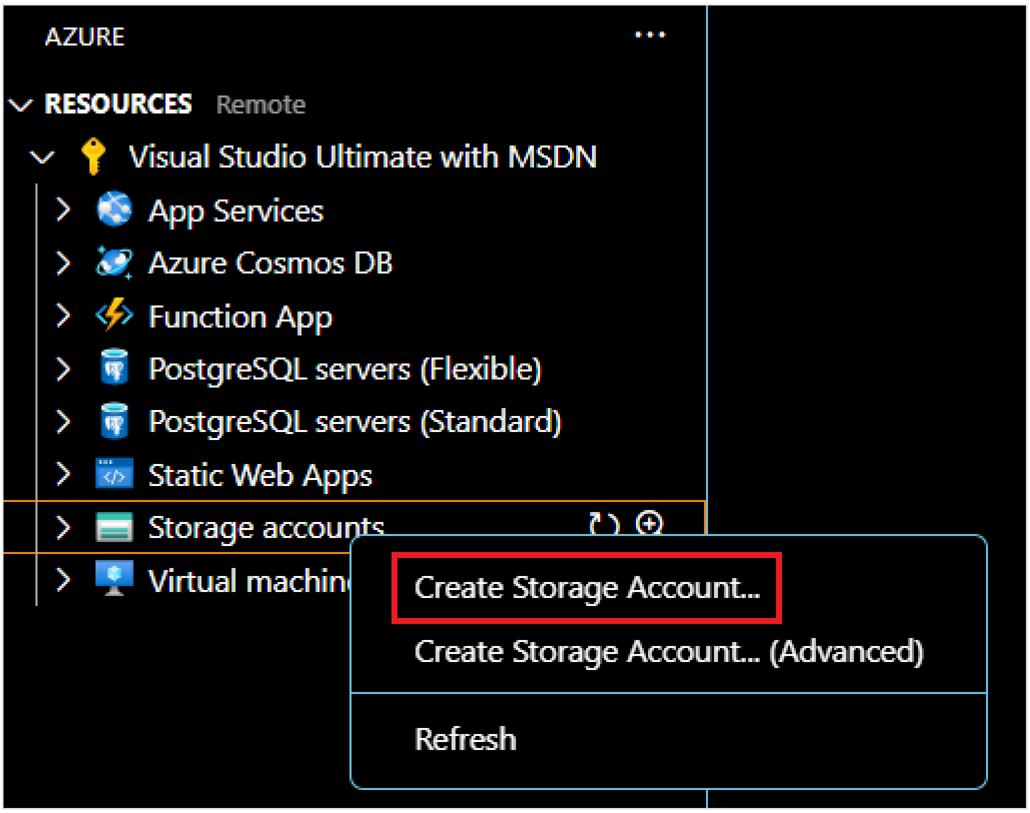
> **Note**

These steps only work for projects created with the Yeoman generator, and that use the add-in only manifest. They don't apply if you created the add-in using Agents Toolkit or created it with the Yeoman generator and it uses the unified manifest for Microsoft 365.

1. Open your project from its root folder in Visual Studio Code (VS Code).
2. Select **View** > **Extensions** (`Ctrl` + `Shift` + `X`) to open the Extensions view.
3. Search for the **Azure Storage** extension and install it.
4. Once installed, an Azure icon is added to the **Activity Bar**. Select it to access the extension. If the **Activity Bar** is hidden, open it by selecting **View** > **Appearance** > **Activity Bar**.
5. Select **Sign in to Azure** to sign in to your Azure account. If you don't already have an Azure account, create one by selecting **Create an Azure Account**. Follow the provided steps to set up your account.



6. Once you're signed in, you'll see your Azure storage accounts appear in the extension. If you don't already have a storage account, create one using the **Create Storage Account** option in the command palette. Name your storage account a globally unique name, using only 'a-z' and '0-9'. Note that by default, this creates a storage account and a resource group with the same name. It automatically puts the storage account in West US. This can be adjusted online through your Azure account.


AZURE
RESOURCES Remote
Visual Studio Ultimate with MSDN
App Services
Azure Cosmos DB
Function App
PostgreSQL servers (Flexible)
PostgreSQL servers (Standard)
Static Web Apps
Storage accounts
Virtual machin
Create Storage Account...
Create Storage Account... (Advanced)
Refresh

7. Right-click (or select and hold) your storage account and select **Configure Static Website**. You'll be asked to enter the index document name and the 404 document name. Change the index document name from the default `index.html` to **`taskpane.html`**. You may also change the 404 document name but aren't required to.

8. Right-click (or select and hold) your storage account again and this time select **Browse Static Website**. From the browser window that opens, copy the website URL.

9. Open your project's manifest file and change all references to your localhost URL (such as `https://localhost:3000`) to the URL you've copied. This endpoint is the static website URL for your newly created storage account. Save the changes to your manifest file.

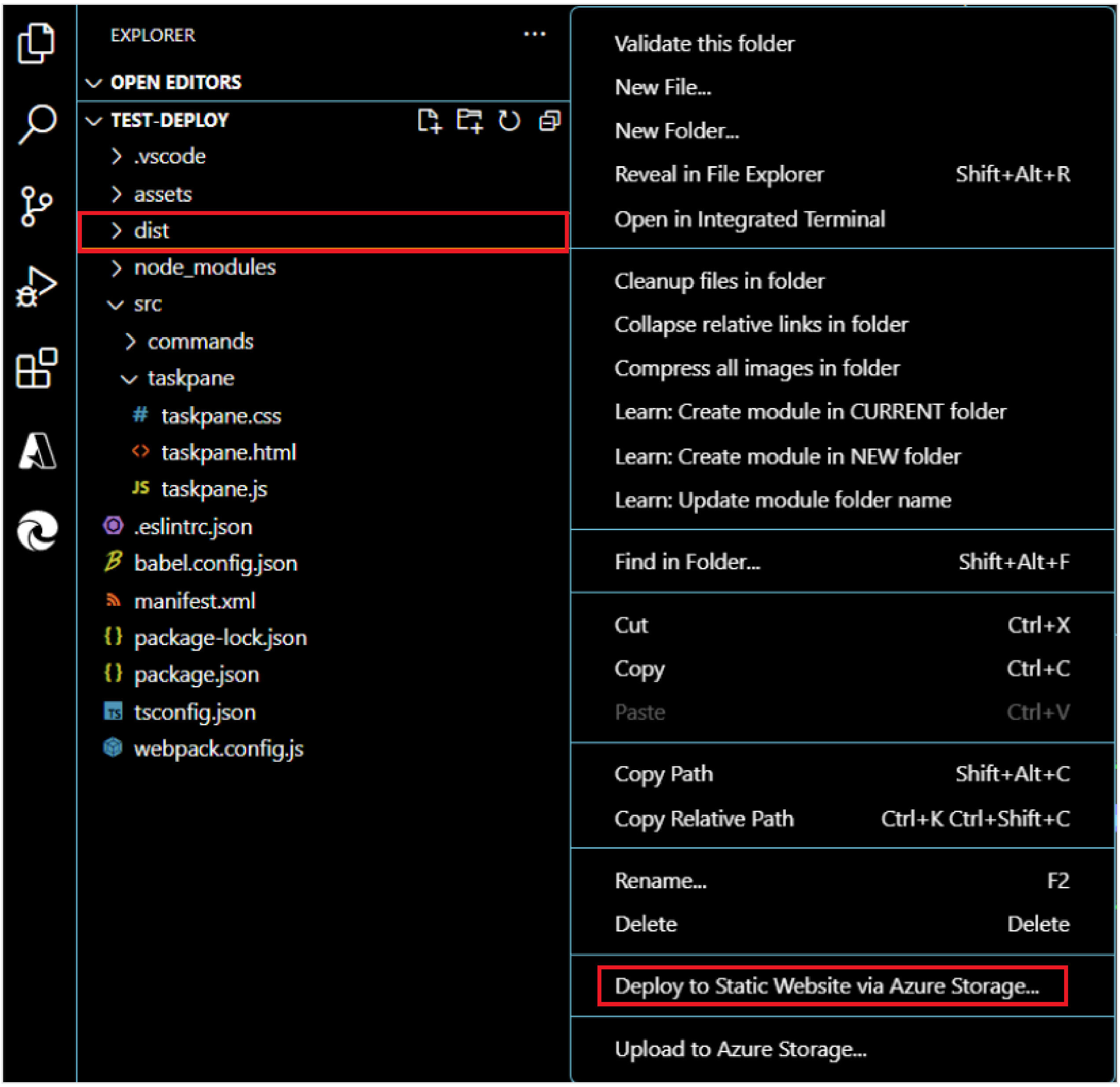
10. Open a command line prompt or terminal window and go to the root directory of your add-in project. Run the following command to prepare all files for production deployment.

command line

```
npm run build
```

When the build completes, the **dist** folder in the root directory of your add-in project will contain the files that you'll deploy in subsequent steps.

11. In VS Code, go to the Explorer and right-click (or select and hold) the **dist** folder, and select **Deploy to Static Website via Azure Storage**. When prompted, select the storage account you created previously.


EXPLORER
OPEN EDITORS
TEST-DEPLOY
.vscode
assets
dist
node_modules
src
commands
taskpane
taskpane.css
taskpane.html
taskpane.js
.eslintrc.json
babel.config.json
manifest.xml
package-lock.json
package.json
tsconfig.json
webpack.config.js
Validate this folder
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Cleanup files in folder
Collapse relative links in folder
Compress all images in folder
Learn: Create module in CURRENT folder
Learn: Create module in NEW folder
Learn: Update module folder name
Find in Folder... Shift+Alt+F
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
Rename... F2
Delete Delete
Deploy to Static Website via Azure Storage...
Upload to Azure Storage...

12. When deployment is complete, right-click (or select and hold) the storage account that you created previously and select **Browse Static Website**. This opens the static web site and displays the task pane.

13. Finally, sideload the manifest file and the add-in will load from the static web site you just deployed.

Deploy custom functions for Excel

If your add-in has custom functions, there are a few more steps to enable them on the Azure Storage account. First, enable CORS so that Office can access the functions.json file.

1. Right-click (or select and hold) the Azure storage account and select **Open in Portal**.

2. In the Settings group, select **Resource sharing (CORS)**. You can also use the search box to find this.

3. Create a new CORS rule for the **Blob service** with the following settings.

Expand table

Property	Value
Allowed origins	*
Allowed methods	GET
Allowed headers	*
Exposed headers	Access-Control-Allow-Origin
Max age	200

4. Select **Save**.

Caution

This CORS configuration assumes all files on your server are publicly available to all domains.

Next, add a MIME type for JSON files.

1. Create a new file in the /src folder named **web.config**.

2. Insert the following XML and save the file.

XML

```xml
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <staticContent>
      <mimeMap fileExtension=".json" mimeType="application/json" />
    </staticContent>
  </system.webServer>
</configuration>
```

3. Open the **webpack.config.js** file.

4. Add the following code in the list of `plugins` to copy the web.config into the bundle when the build runs.

JavaScript

```
new CopyWebpackPlugin({
  patterns: [
  {
    from: "src/web.config",
    to: "src/web.config",
  },
 ],
}),
```

5. Open a command line prompt and go to the root directory of your add-in project. Then, run the following command to prepare all files for deployment.

   command line

   ```
   npm run build
   ```

   When the build completes, the **dist** folder in the root directory of your add-in project will contain the files that you'll deploy.

6. To deploy, in the VS Code **Explorer**, right-click (or select and hold) the **dist** folder and select **Deploy to Static Website via Azure Storage**. When prompted, select the storage account you created previously. If you already deployed the **dist** folder, you'll be prompted if you want to overwrite the files in the Azure storage with the latest changes.

Deploy updates

You'll deploy updates to your web application in the same manner as described previously. Changes to the manifest require redistributing your manifest to users. The process to do so depends on your publishing method. For more information on updating your add-in, see Maintain your Office Add-in.

See also

- Develop Office Add-ins with Visual Studio Code
- Deploy and publish your Office Add-in
- Cross-Origin Resource Sharing (CORS) support for Azure Storage