Deploying ASP.NET Core Web API to Azure App Service

A Step-by-Step Guide for Students

Introduction

This guide will walk you through deploying an existing ASP.NET Core Web API project to Azure App Service. You'll learn two approaches: using Visual Studio's built-in tools and using the Azure CLI for command-line deployment.

Prerequisites

- Visual Studio 2022 (Community, Professional, or Enterprise)
- Azure subscription (You can use a student account or free trial)
- Azure CLI installed on your development machine
- An existing ASP.NET Core Web API project that you want to deploy

Method 1: Using Visual Studio 2022

Step 1: Prepare Your Project

1. Open your ASP.NET Core Web API project in Visual Studio 2022.
2. Ensure your project builds without errors by selecting **Build > Build Solution**.
3. Verify your API works locally by pressing F5 to run it in debug mode.

Step 2: Publish to Azure

1. Right-click your project in Solution Explorer.

2. Select **Publish**.

3. In the publish dialog, select **Azure** as your target, then click **Next**.

4. Select **Azure App Service (Windows)** or **Azure App Service (Linux)** depending on your preference, then click **Next**.

5. Sign in to your Azure account if prompted.

6. In the App Service dialog:

   - To create a new App Service, click the **+** button.
   - Or, to use an existing App Service, select it from the list.

7. If creating a new App Service:

   - Enter a name for your app (this will form part of the URL, e.g., `yourappname.azurewebsites.net`)
   - Select your subscription

- Create or select a Resource Group
- Create or select an App Service Plan/Location
- Click **Create**

8. Click **Finish** to return to the publish profile screen.

9. Click **Publish** to deploy your application to Azure.

10. Visual Studio will show a progress bar and then open your deployed API in a browser.

Step 3: Verify Deployment

1. Once deployment is complete, Visual Studio will open your browser pointing to your deployed API.
2. Test your API endpoints using a tool like Postman, Swagger (if configured), or a web browser.

Method 2: Using Azure CLI

Step 1: Publish Your Application Locally

1. Open a command prompt in your project directory.

2. Run the following command to build and publish your application:

```
dotnet publish -c Release
```

3. This creates a publish-ready version of your app in the `bin/Release/net8.0/publish` directory (adjust for your .NET version).

Step 2: Log in to Azure

1. Open a command prompt or PowerShell window.

2. Run the following command and follow the instructions to log in:

```
az login
```

Step 3: Create a Resource Group (if needed)

1. If you don't already have a resource group, create one:

```
az group create --name YourResourceGroupName --location eastus
```

You can change `eastus` to a location closer to you or your users.

Step 4: Create an App Service Plan

1. Create an App Service Plan (the hosting plan for your API):

```
az appservice plan create --name YourPlanName --resource-group
YourResourceGroupName --sku B1
```

Note: B1 is a Basic tier suitable for testing. For production, consider S1 or higher.

Step 5: Create a Web App

1. Create a Web App in Azure:

```
az webapp create --resource-group YourResourceGroupName --plan YourPlanName
--name YourUniqueAppName --runtime "DOTNET:8.0"
```

Replace `YourUniqueAppName` with a globally unique name for your app.

Step 6: Create a ZIP File of Your Published App

1. Navigate to your publish directory:

```
cd bin/Release/net8.0/publish
```

2. Create a ZIP file of the published content:

```
# In PowerShell
Compress-Archive -Path * -DestinationPath ../../publish.zip -Force
```

OR

```
# In Command Prompt (if you have 7zip installed)
7z a -r ../../publish.zip *
```

3. Navigate back to your project root:

```
cd ../../../..
```

Step 7: Deploy the ZIP File to Azure

1. Deploy your application using the ZIP deployment method:

```
az webapp deployment source config-zip --resource-group
YourResourceGroupName --name YourUniqueAppName --src
bin/Release/net8.0/publish.zip
```

Alternatively, you can use the newer command (if your Azure CLI version supports it):

```
az webapp deploy --resource-group YourResourceGroupName --name
YourUniqueAppName --src-path bin/Release/net8.0/publish.zip --type zip
```

Step 8: Verify Your Deployment

1. Open your browser and navigate to:

```
https://YourUniqueAppName.azurewebsites.net
```

2. Test your API endpoints using Postman, Swagger, or a web browser.

Troubleshooting

Path Issues with Azure CLI

If you encounter path-related errors when using `az webapp deploy`, try:

1. Using the full absolute path:

```
az webapp deploy --resource-group YourResourceGroupName --name
YourUniqueAppName --src-path C:\Full\Path\To\publish.zip --type zip
```

2. Creating the ZIP file in a simpler location:

```
Compress-Archive -Path .\bin\Release\net8.0\publish\* -DestinationPath
C:\Users\$env:USERNAME\publish.zip -Force
az webapp deploy --resource-group YourResourceGroupName --name
YourUniqueAppName --src-path C:\Users\$env:USERNAME\publish.zip --type zip
```

3. Using the older command which might have better path handling:

```
az webapp deployment source config-zip --resource-group
YourResourceGroupName --name YourUniqueAppName --src
C:\Users\$env:USERNAME\publish.zip
```

Common Deployment Issues

1. **Deployment appears successful but API doesn't work:**
   - Check logs in Azure Portal: App Service > YourApp > Diagnose and solve problems
   - Check Application Insights if configured
2. **CORS issues:**
   - Ensure your CORS settings are configured properly in Azure
   - In Azure Portal, go to App Service > CORS
3. **Connection string issues:**
   - Check if your app uses connection strings that need to be updated
   - Configure connection strings in Azure Portal: App Service > Configuration > Connection strings

Final Tips

1. **Consider using GitHub Actions or Azure DevOps for CI/CD:**
   - Set up automatic deployments whenever you push to your repository
   - Visual Studio can help you set this up during the publish process
2. **Set up proper logging:**
   - Configure Application Insights for comprehensive monitoring
   - Check logs regularly to identify issues
3. **Always test thoroughly after deployment:**
   - Test all endpoints in your production environment
   - Verify that your app can connect to all required services

Conclusion

You've successfully deployed your ASP.NET Core Web API to Azure App Service! Your API is now accessible worldwide via the URL `https://YourUniqueAppName.azurewebsites.net`.

For more advanced scenarios, consider exploring:

- Azure Key Vault for secure secrets management
- Azure API Management for API gateways and management
- Azure Front Door for global load balancing and security