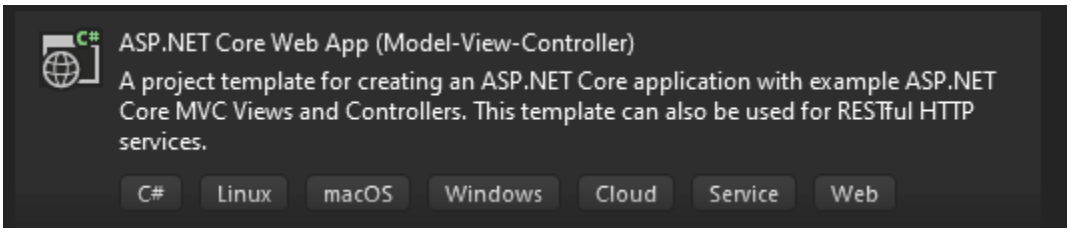
Creating a Microservice in Visual Studio

Caden Lafollette

Some notes:

Each service should be its own project (meaning it should have its own *.sln file).

When making a new project, use the ASP .NET Core Web App (MVC) template.

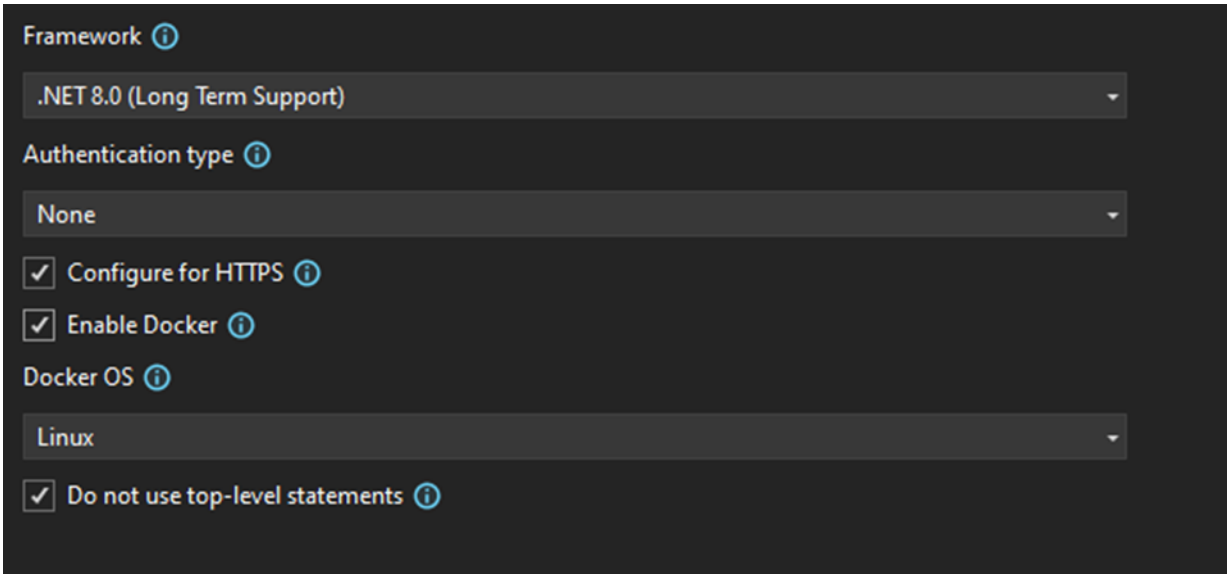


If you aren't familiar with Microsoft's version of MVC, you can find some information here:

https://dotnet.microsoft.com/en-us/apps/aspnet/mvc

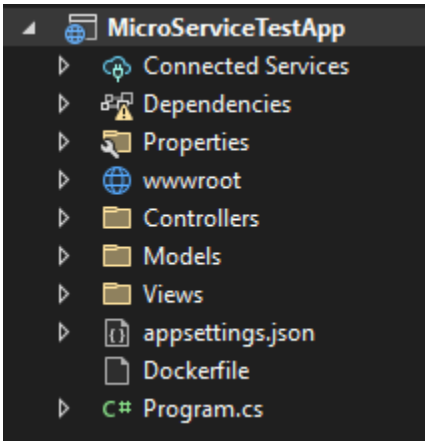
1. Creating a new project for the service

Starting with the MVC template, you can use these settings:



These are the settings I used when making a test service. Top-level statements are optional, but I don't like them, so I disable them.

This is the file structure you should have by default.



This dockerfile is also included by default, which will let you deploy the project in a docker container.

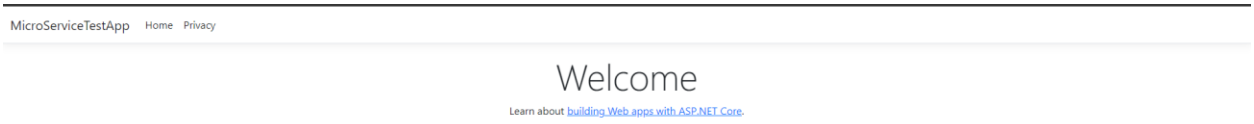
```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["MicroServiceTestApp/MicroServiceTestApp.csproj", "MicroServiceTestApp/"]
RUN dotnet restore "./MicroServiceTestApp/MicroServiceTestApp.csproj"
COPY . .
WORKDIR "/src/MicroServiceTestApp"
RUN dotnet build "./MicroServiceTestApp.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./MicroServiceTestApp.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MicroServiceTestApp.dll"]
```

You will also be given some code inside of Program.cs that will let you run the application, although it won't do much yet.

If you don't already have it, install docker desktop.

https://www.docker.com/products/docker-desktop/

You should be able to run the project, and it will build successfully, and you should see this running in your local browser.



You now have a basic service running that can be deployed in an AWS instance (or something else that supports docker containers.)

2. Adding Functionality

A microservice isn't very useful without communicating to other apps, so let's look at the workflow for doing that.

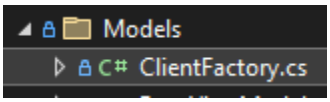
First, add a new Model to the "Models" folder. I'm naming mine "HttpModel".

In this Model, add the following code, sourced and altered from https://learn.microsoft.com/en-us/aspnet/core/fundamentals/http-requests?view=aspnetcore-8.0

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Net.Http.Headers;
using System.Text.Json;

namespace MicroServiceTestApp.Models
{
    2 references | 0 changes | 0 authors, 0 changes
    public class HttpModel : PageModel
    {
        private readonly IHttpClientFactory _httpClientFactory;

        public HttpResponseMessage ResponseMessage;

        1 reference | 0 changes | 0 authors, 0 changes
        public HttpModel(IHttpClientFactory httpClientFactory) =>
            _httpClientFactory = httpClientFactory;

        1 reference | 0 changes | 0 authors, 0 changes
        public async Task OnGet()
        {
            var httpClient = _httpClientFactory.CreateClient("GitHub");
            var httpRequestMessage = new HttpRequestMessage(
                HttpMethod.Get,
                "https://api.github.com/repos/dotnet/AspNetCore.Docs/branches")
            {
            };

            var httpResponseMessage = await httpClient.SendAsync(httpRequestMessage);

            ResponseMessage = httpResponseMessage;
        }
    }
}

```

We aren't worried about parsing the data, just confirming that we can do an HTTP request, so we will only track the status code.

Next, add a class to implement the “IHttpClientFactory” interface. I added this to the “models” folder, though there may be a better place.

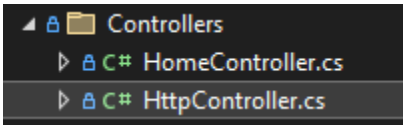


Add the following code:

```
1
2   namespace MicroServiceTestApp.Models
3   {
        1 reference | 0 changes | 0 authors, 0 changes
4       public class ClientFactory : IHttpClientFactory
5       {
            0 references | 0 changes | 0 authors, 0 changes
6           public HttpClient CreateClient(string name)
7           {
8               return new HttpClient();
9           }
10      }
11  }
12
```

CreateClient() is the only method we need to implement, and it takes a string. You could use this to set configurations for the client, but we don’t need to do that here, so the default is fine.

Next, add a Controller for testing the Http request.



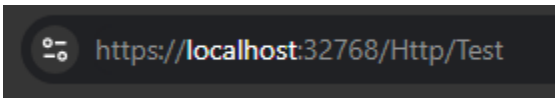
Add the following code to this controller:

```
using MicroServiceTestApp.Models;
using Microsoft.AspNetCore.Mvc;
using System.Dynamic;

namespace MicroServiceTestApp.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class HttpController : Controller
    {
        0 references | 0 changes | 0 authors, 0 changes
        public IActionResult Index()
        {
            return View();
        }

        0 references | 0 changes | 0 authors, 0 changes
        public async Task<IActionResult> Test()
        {
            var client = new HttpModel(new ClientFactory());
            await client.OnGet();

            if(client.ResponseMessage.IsSuccessStatusCode)
            {
                return Content("Get request was a success!");
            }
            else
            {
                return Content("Get request was not a success.");
            }
        }
    }
}
```

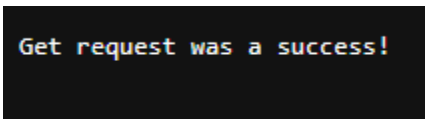
The Test() action creates a client and uses the OnGet() method to perform a get request. We check the response message and see if it was successful.

You can make a new View for Test() if you'd like, but it isn't necessary to test the request.

Finally, to test the request, build the project. Once it is running, append Http/Test to your local browser, like this:



If all goes well, you should see the following message in the browser:



If not, ensure that you've enabled the headers when making the client that GitHub needs. You can also try a few times to see if it was a fluke.