

Contents

Visual Studio Container Tools

[Overview](#)

[Quickstart: Docker in Visual Studio](#)

[Tutorial: Multi-container app with Docker Compose](#)

[Tutorial: Kubernetes in Visual Studio](#)

[Debugging apps in a local Docker container](#)

[Deploy an ASP.NET container to a container registry using Visual Studio](#)

[Configure a Docker Host with VirtualBox](#)

[Troubleshoot Visual Studio 2017 development with Docker](#)

Quickstart: Docker in Visual Studio

3/1/2019 • 7 minutes to read • [Edit Online](#)

With Visual Studio, you can easily build, debug, and run containerized ASP.NET Core apps and publish them to Azure Container Registry (ACR), Docker Hub, Azure App Service, or your own container registry. In this article, we'll publish to ACR.

Prerequisites

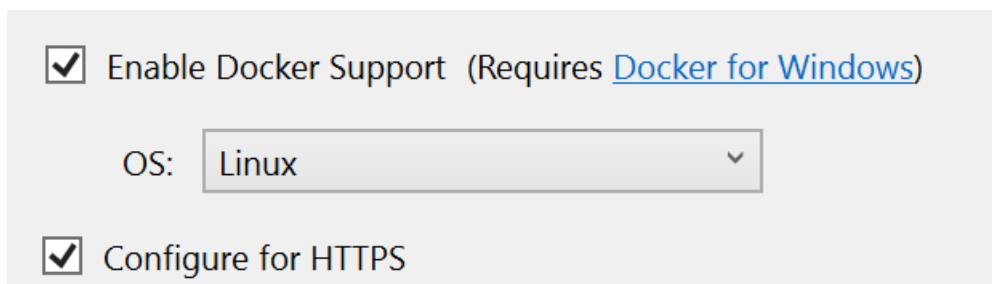
- [Docker Desktop](#)
- [Visual Studio 2017](#) with the **Web Development**, **Azure Tools** workload, and/or **.NET Core cross-platform development** workload installed
- To publish to Azure Container Registry, an Azure subscription. [Sign up for a free trial](#).

Installation and setup

For Docker installation, first review the information at [Docker Desktop for Windows: What to know before you install](#). Next, install [Docker Desktop](#).

Add a project to a Docker container

1. From the Visual Studio menu, select **File > New > Project**.
2. Under the **Templates** section of the **New Project** dialog box, select **Visual C# > Web**.
3. Select **ASP.NET Core Web Application**.
4. Give your new application a name (or take the default) and select **OK**.
5. Select **Web Application**.
6. Check the **Enable Docker Support** checkbox.



The screenshot shows a section of the Visual Studio 'New Project' dialog. It features a checkbox labeled 'Enable Docker Support' with the text '(Requires [Docker for Windows](#))' next to it. Below this is a dropdown menu labeled 'OS:' with 'Linux' selected. At the bottom of this section is another checkbox labeled 'Configure for HTTPS'.

7. Select the type of container you want (Windows or Linux) and click **OK**.

Dockerfile overview

A *Dockerfile*, the recipe for creating a final Docker image, is created in the project. Refer to [Dockerfile reference](#) for an understanding of the commands within it:

```

FROM microsoft/dotnet:2.1-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 59518
EXPOSE 44364

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY HelloDockerTools/HelloDockerTools.csproj HelloDockerTools/
RUN dotnet restore HelloDockerTools/HelloDockerTools.csproj
COPY . .
WORKDIR /src/HelloDockerTools
RUN dotnet build HelloDockerTools.csproj -c Release -o /app

FROM build AS publish
RUN dotnet publish HelloDockerTools.csproj -c Release -o /app

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

```

The preceding *Dockerfile* is based on the [microsoft/aspnetcore](#) image, and includes instructions for modifying the base image by building your project and adding it to the container.

When the new project dialog's **Configure for HTTPS** check box is checked, the *Dockerfile* exposes two ports. One port is used for HTTP traffic; the other port is used for HTTPS. If the check box isn't checked, a single port (80) is exposed for HTTP traffic.

Debug

Select **Docker** from the debug drop-down in the toolbar, and start debugging the app. You might see a message with a prompt about trusting a certificate; choose to trust the certificate to continue.

The **Output** window shows what actions are taking place.

Open the **Package Manager Console** (PMC) from the menu **Tools**> NuGet Package Manager, **Package Manager Console**.

The resulting Docker image of the app is tagged as *dev*. The image is based on the *2.1-aspnetcore-runtime* tag of the *microsoft/dotnet* base image. Run the `docker images` command in the **Package Manager Console** (PMC) window. The images on the machine are displayed:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hellodockertools	dev	d72ce0f1dfe7	30 seconds ago	255MB
microsoft/dotnet	2.1-aspnetcore-runtime	fcc3887985bb	6 days ago	255MB

NOTE

The **dev** image does not contain the app binaries and other content, as **Debug** configurations use volume mounting to provide the iterative edit and debug experience. To create a production image containing all contents, use the **Release** configuration.

Run the `docker ps` command in PMC. Notice the app is running using the container:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
baf9a678c88d	hellodockertools:dev	"C:\\remote_debugge..."	21 seconds ago	Up 19 seconds
0.0.0.0:37630->80/tcp	dockercompose4642749010770307127_hellodockertools_1			

Publish Docker images

Once the develop and debug cycle of the app is completed, you can create a production image of the app.

1. Change the configuration drop-down to **Release** and build the app.
2. Right-click your project in **Solution Explorer** and choose **Publish**.
3. On the publish target dialog, select the **Container Registry** tab.
4. Choose **Create New Azure Container Registry** and click **Publish**.
5. Fill in your desired values in the **Create a new Azure Container Registry**.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS Prefix	Globally unique name	Name that uniquely identifies your container registry.
Subscription	Choose your subscription	The Azure subscription to use.
Resource Group	myResourceGroup	Name of the resource group in which to create your container registry. Choose New to create a new resource group.
SKU	Standard	Service tier of the container registry
Registry Location	A location close to you	Choose a Location in a region near you or near other services that will use your container registry.

Create a new Azure Container Registry

DNS Prefix
MyWebApplication20180426102127

Subscription
Visual Studio Enterprise

Resource Group
myResourceGroup* [New...](#)

SKU
Standard

Registry Location
West US

[Export...](#)

Explore additional Azure services

- [Create App Service](#)
- [Create a new Azure Container Registry](#)
- [Create a SQL Database](#)
- [Create a storage account](#)

Clicking the Create button will create the following Azure resources

Azure Container Registry - MyWebApplication20180426102127

[Create](#) [Cancel](#)

6. Click **Create**

Next steps

You can now pull the container from the registry to any host capable of running Docker images, for example [Azure Container Instances](#).

Additional resources

- [Container development with Visual Studio](#)
- [Troubleshoot Visual Studio 2017 development with Docker](#)
- [Visual Studio Tools for Docker GitHub repository](#)

With Visual Studio, you can easily build, debug, and run containerized ASP.NET Core apps and publish them to Azure Container Registry (ACR), Docker Hub, Azure App Service, or your own container registry. In this article, we'll publish to ACR.

Prerequisites

- [Docker Desktop](#)
- [Visual Studio 2019](#) with the **Web Development, Azure Tools** workload, and/or **.NET Core cross-platform development** workload installed
- [.NET Core 2.2 Development Tools](#) for development with .NET Core 2.2
- To publish to Azure Container Registry, an Azure subscription. [Sign up for a free trial](#).

Installation and setup

For Docker installation, first review the information at [Docker Desktop for Windows: What to know before you install](#). Next, install [Docker Desktop](#).

Add a project to a Docker container

1. Create a new project using the **ASP.NET Core Web Application** template.
2. Select **Web Application**, and make sure the **Enable Docker Support** checkbox is selected.

The screenshot shows the 'Create a new ASP.NET Core Web Application' dialog. At the top, there are dropdowns for '.NET Core' and 'ASP.NET Core 2.2'. Below these, there are five project templates: 'Empty', 'API', 'Web Application' (highlighted with a red border), 'Web Application (Model-View-Controller)', and 'Razor Class Library'. To the right, there are sections for 'Authentication' (No Authentication), 'Advanced' (with checkboxes for 'Configure for HTTPS' and 'Enable Docker Support', the latter being checked and highlighted with a red border), and a dropdown for 'Linux' (also highlighted with a red border). At the bottom right, there are 'Back' and 'Create' buttons, with the 'Create' button highlighted with a red border.

3. Select the type of container you want (Windows or Linux) and click **Create**.

Dockerfile overview

A *Dockerfile*, the recipe for creating a final Docker image, is created in the project. Refer to [Dockerfile reference](#) for an understanding of the commands within it:

```
FROM microsoft/dotnet:2.2-aspnetcore-runtime-stretch-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM microsoft/dotnet:2.2-sdk-stretch AS build
WORKDIR /src
COPY ["HelloDockerTools/HelloDockerTools.csproj", "HelloDockerTools/"]
RUN dotnet restore "HelloDockerTools/HelloDockerTools.csproj"
COPY . .
WORKDIR "/src/HelloDockerTools"
RUN dotnet build "HelloDockerTools.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "HelloDockerTools.csproj" -c Release -o /app

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

The preceding *Dockerfile* is based on the [microsoft/aspnetcore](#) image, and includes instructions for modifying the base image by building your project and adding it to the container.

When the new project dialog's **Configure for HTTPS** check box is checked, the *Dockerfile* exposes two ports. One port is used for HTTP traffic; the other port is used for HTTPS. If the check box isn't checked, a single port (80) is exposed for HTTP traffic.

Debug

Select **Docker** from the debug drop-down in the toolbar, and start debugging the app. You might see a message with a prompt about trusting a certificate; choose to trust the certificate to continue.

The **Container Tools** option in the **Output** window shows what actions are taking place.

Open the **Package Manager Console** (PMC) from the menu **Tools**> NuGet Package Manager, **Package Manager Console**.

The resulting Docker image of the app is tagged as *dev*. The image is based on the *2.2-aspnetcore-runtime* tag of the *microsoft/dotnet* base image. Run the `docker images` command in the **Package Manager Console** (PMC) window. The images on the machine are displayed:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hellodockertools	dev	d72ce0f1dfe7	30 seconds ago	255MB
microsoft/dotnet	2.2-aspnetcore-runtime	fcc3887985bb	6 days ago	255MB

NOTE

The **dev** image does not contain the app binaries and other content, as **Debug** configurations use volume mounting to provide the iterative edit and debug experience. To create a production image containing all contents, use the **Release** configuration.

Run the `docker ps` command in PMC. Notice the app is running using the container:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
cf5d2ef5f19a	hellodockertools:dev	"tail -f /dev/null"	2 minutes ago	Up 2 minutes	
0.0.0.0:52036->80/tcp, 0.0.0.0:44342->443/tcp		priceless_cartwright			


Publish Docker images

Once the develop and debug cycle of the app is completed, you can create a production image of the app.

1. Change the configuration drop-down to **Release** and build the app.
2. Right-click your project in **Solution Explorer** and choose **Publish**.
3. On the publish target dialog, select the **Container Registry** tab.
4. Choose **Create New Azure Container Registry** and click **Publish**.
5. Fill in your desired values in the **Create a new Azure Container Registry**.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS Prefix	Globally unique name	Name that uniquely identifies your container registry.
Subscription	Choose your subscription	The Azure subscription to use.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource Group	myResourceGroup	Name of the resource group in which to create your container registry. Choose New to create a new resource group.
SKU	Standard	Service tier of the container registry
Registry Location	A location close to you	Choose a Location in a region near you or near other services that will use your container registry.



Create a new Azure Container Registry

Create new

[New...](#)

[Export...](#)

[Create](#)
[Cancel](#)


Explore additional Azure services

Clicking the Create button will create the following Azure resources

Container Registry - HelloDockerTools20190201111619

6. Click **Create**





Publish

 **Azure successfully configured:** [How was your experience?](#)

[Publish](#)

[New](#)
[Rename](#)
[Delete](#)

Summary

Image Tag	latest 	Edit Image Tag
Repository	https://hellodockertools20190201111619.azurecr.io/ 	
Subscription	Microsoft Azure Internal Consumption 	
Configuration	Release 	

Next Steps

You can now pull the container from the registry to any host capable of running Docker images, for example [Azure Container Instances](#).

Additional resources

- [Container development with Visual Studio](#)
- [Troubleshoot Visual Studio 2017 development with Docker](#)
- [Visual Studio Tools for Docker GitHub repository](#)

Additional resources

- [Container development with Visual Studio](#)
- [Troubleshoot Visual Studio 2017 development with Docker](#)
- [Visual Studio Tools for Docker GitHub repository](#)

Tutorial: Create a multi-container app with Docker Compose

3/4/2019 • 3 minutes to read • [Edit Online](#)

In this tutorial, you'll learn how to manage more than one container and communicate between them when using Container Tools in Visual Studio. Managing multiple containers requires *container orchestration* and requires an orchestrator such as Docker Compose, Kubernetes, or Service Fabric. Here, we'll use Docker Compose. Docker Compose is great for local debugging and testing in the course of the development cycle.

Prerequisites

- [Docker Desktop](#)
- [Visual Studio 2017](#) with the **Web Development, Azure Tools** workload, or **.NET Core cross-platform development** workload installed
- [Docker Desktop](#)
- [Visual Studio 2019](#) with the **Web Development, Azure Tools** workload, and/or **.NET Core cross-platform development** workload installed
- [.NET Core 2.2 Development Tools](#) for development with .NET Core 2.2

Create a Web Application project

In Visual Studio, create an **ASP.NET Core Web Application** project, named `WebFrontEnd`. Select **Web Application** to create a web application with Razor pages.

Do not select **Enable Docker Support**. You'll add Docker support later.

New ASP.NET Core Web Application - WebFrontEnd

.NET Core ASP.NET Core 2.2 [Learn more](#)

Empty

API

Web Application

Web Application (Model-View-Controller)

Razor Class Library

Angular

React.js

React.js and Redux

A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content. [Learn more](#)

Author: Microsoft
Source: SDK 2.2.200-preview-009748

Authentication: **No Authentication**
[Change Authentication](#)

[Get additional project templates](#)

☐ Enable Docker Support (Requires [Docker for Windows](#))

OS: Linux

☒ Configure for HTTPS

OK Cancel

Configure your new project

ASP.NET Core Web Application C# Windows Linux macOS Web

Project name
WebFrontEnd

Location
C:\Source\Repos\

Solution name ⓘ
WebFrontEnd

☐ Place solution and project in the same directory


Back **Create**


Do not select **Enable Docker Support**. You'll add Docker support later.


Create a new ASP.NET Core Web Application


.NET Core


ASP.NET Core 2.2

**Empty**
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

**API**
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

**Web Application**
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content.

**Web Application (Model-View-Controller)**
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

**Razor Class Library**
A project template for creating a Razor class library.

[Get additional project templates](#)

Back

Create

Authentication
No Authentication
[Change](#)

Advanced
☒ Configure for HTTPS
☐ Enable Docker Support
(Requires [Docker for Windows](#))

Linux

Author: Microsoft
Source: SDK 2.2.200-preview-009804

Create a Web API project


Add a project to the same solution and call it *MyWebAPI*. Select **API** as the project type, and clear the checkbox for **Configure for HTTPS**. In this design, we're only using SSL for communication with the client, not for communication from between containers in the same web application. Only `WebFrontEnd` needs HTTPS.


New ASP.NET Core Web Application - MyWebAPI


.NET Core


ASP.NET Core 2.2


[Learn more](#)


**Empty**


**API**


**Web Application**

**Web Application (Model-View-Controller)**

**Razor Class Library**

**Angular**

**React.js**

**React.js and Redux**

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
[Learn more](#)

Author: Microsoft
Source: SDK 2.2.200-preview-009748

Authentication: **No Authentication**

Change Authentication

☐ Enable Docker Support (Requires [Docker for Windows](#))

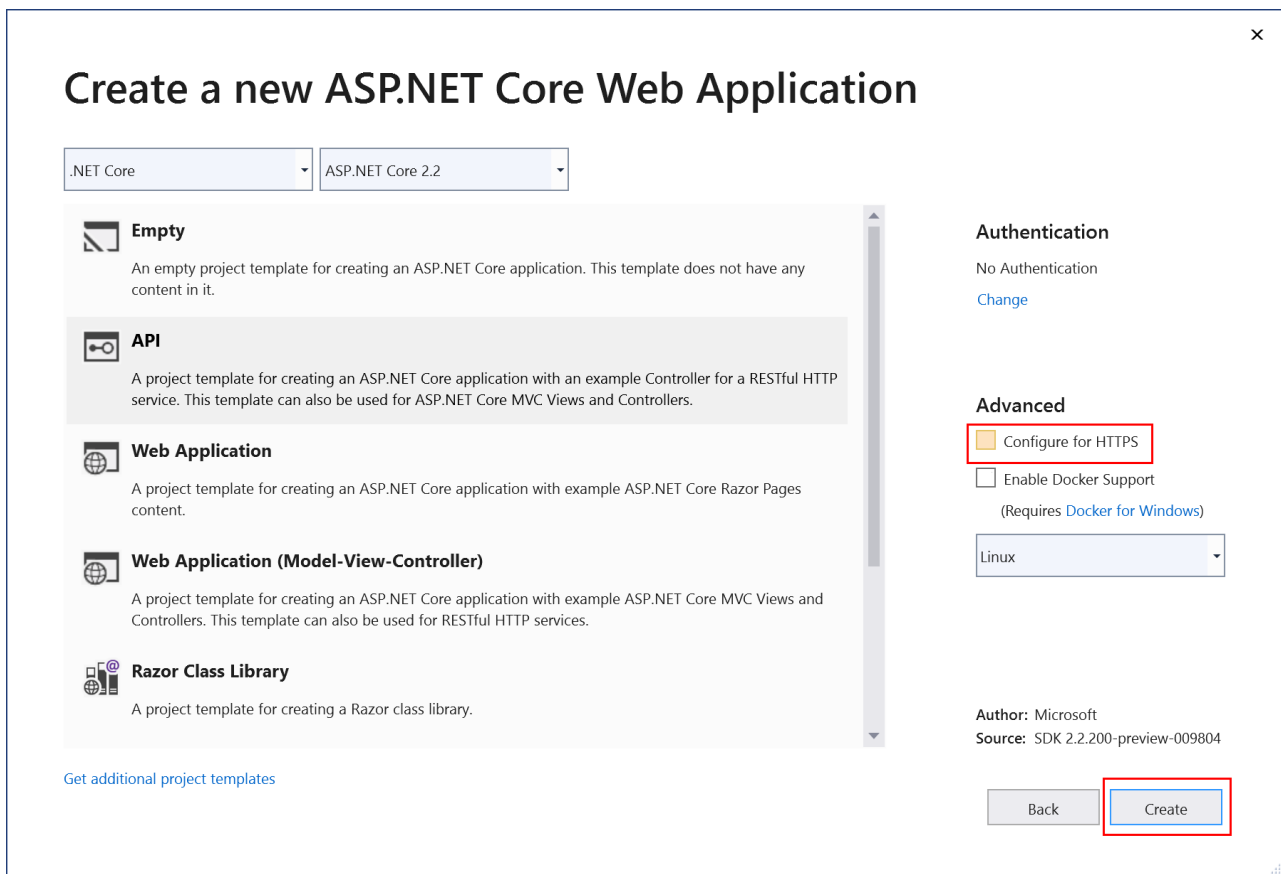
OS:

Linux

☐ Configure for HTTPS

OK

Cancel



Add code to call the Web API

1. In the `WebFrontEnd` project, open the `Index.cshtml.cs` file, and replace the `OnGet` method with the following code.

```
public async Task OnGet()
{
    ViewData["Message"] = "Hello from webfrontend";

    using (var client = new System.Net.Http.HttpClient())
    {
        // Call *mywebapi*, and display its response in the page
        var request = new System.Net.Http.HttpRequestMessage();
        request.RequestUri = new Uri("http://mywebapi/api/values/1");
        var response = await client.SendAsync(request);
        ViewData["Message"] += " and " + await response.Content.ReadAsStringAsync();
    }
}
```

2. In the `Index.cshtml` file, add a line to display `ViewData["Message"]` so that the file looks like the following code:

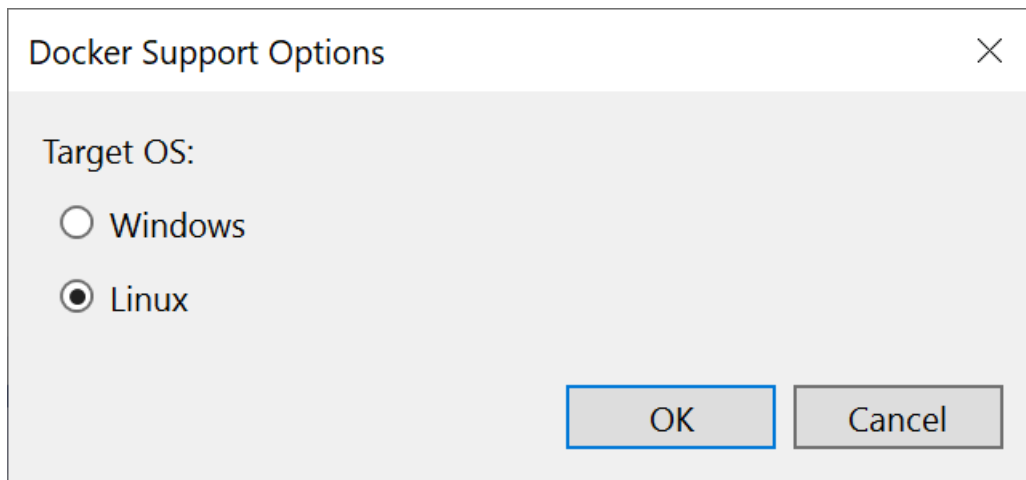
```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
    <p>@ViewData["Message"]</p>
</div>
```

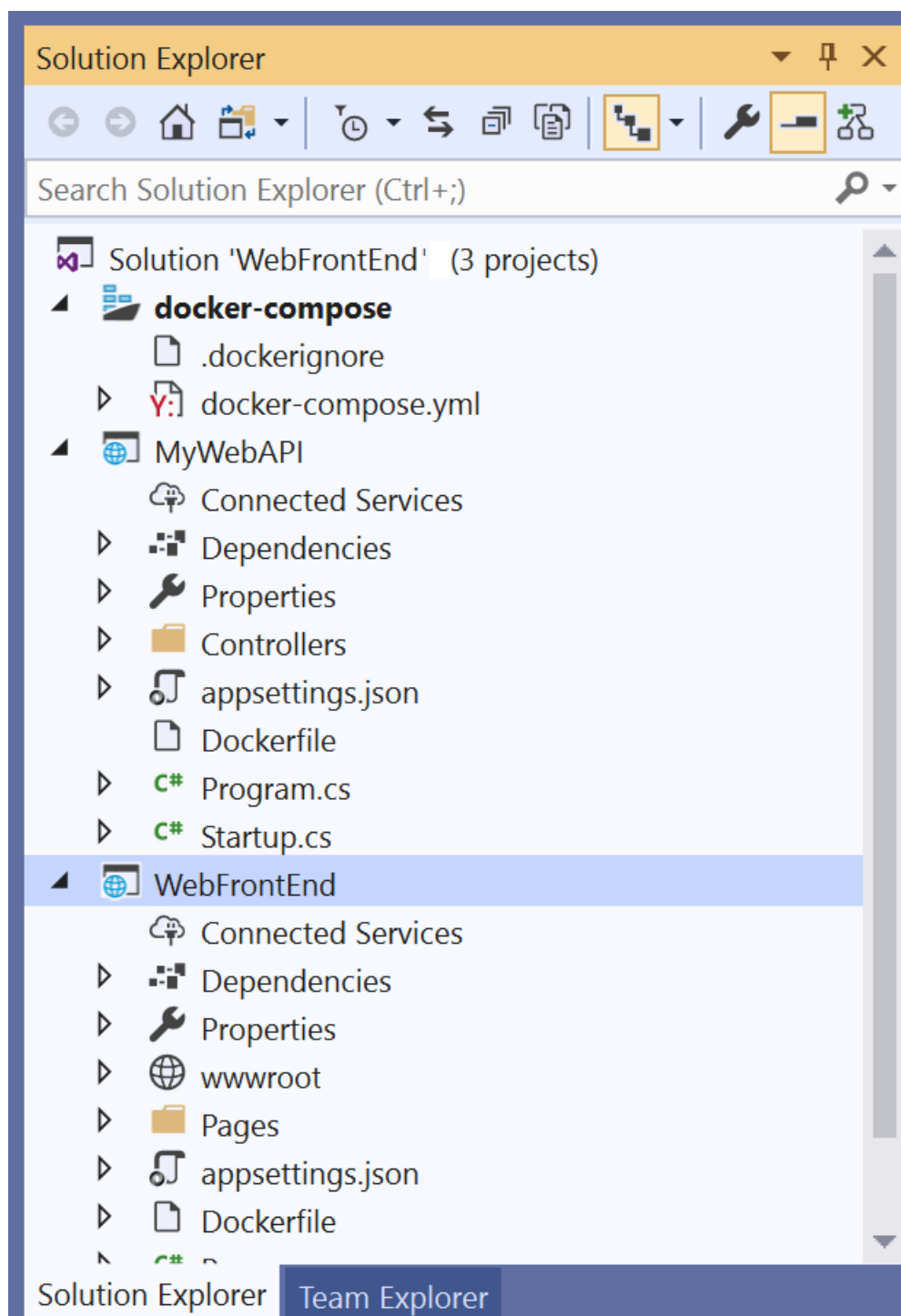
- Now in the Web API project, add code to the Values controller to customize the message returned by the API for the call you added from *webfrontend*.

```
// GET api/values/5
[HttpGet("{id}")]
public ActionResult<string> Get(int id)
{
    return "webapi (with value " + id + ")";
}
```

- In the `WebFrontEnd` project, choose **Add > Container Orchestrator Support**. The **Docker Support Options** dialog appears.
- Choose **Docker Compose**.
- Choose your Target OS, for example, Linux.



Visual Studio creates a *docker-compose.yml* file and a *.dockerignore* file in the **docker-compose** node in the solution, and that project shows in boldface font, which shows that it's the startup project.



The *docker-compose.yml* appears as follows:

```
version: '3.4'

services:
  webfrontend:
    image: ${DOCKER_REGISTRY-}webfrontend
    build:
      context: .
      dockerfile: WebFrontEnd/Dockerfile
```

The *.dockerignore* file contains file types and extensions that you don't want Docker to include in the container. These files are generally associated with the development environment and source control, not part of the app or service you're developing.

Look at the **Container Tools** section of the output pane for details of the commands being run. You can see the command-line tool docker-compose is used to configure and create the runtime containers.

7. In the Web API project, again right-click on the project node, and choose **Add > Container Orchestrator Support**. Choose **Docker Compose**, and then select the same target OS.

NOTE

In this step, Visual Studio will offer to create a Dockerfile. If you do this on a project that already has Docker support, you are prompted whether you want to overwrite the existing Dockerfile. If you've made changes in your Dockerfile that you want to keep, choose no.

Visual Studio makes some changes to your docker compose YML file. Now both services are included.

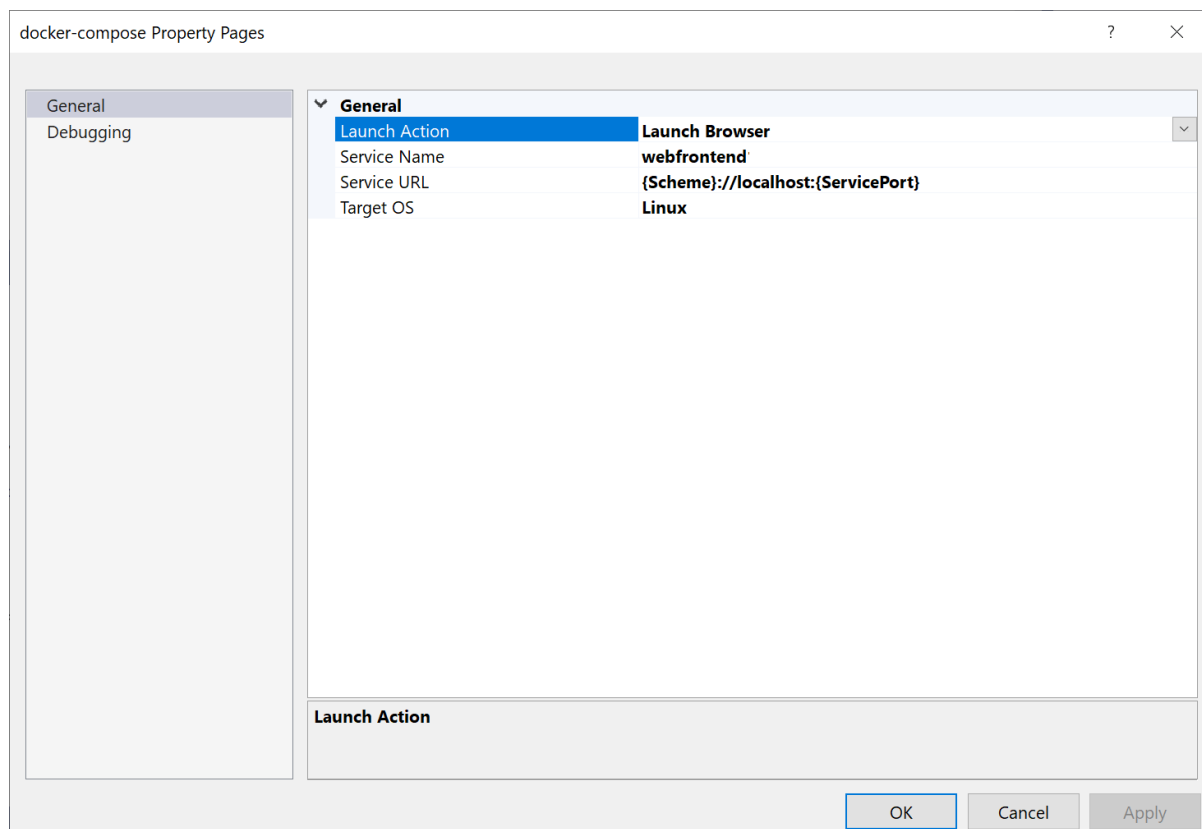
```
version: '3.4'

services:
  webfrontend:
    image: ${DOCKER_REGISTRY-}webfrontend
    build:
      context: .
      dockerfile: WebFrontEnd/Dockerfile

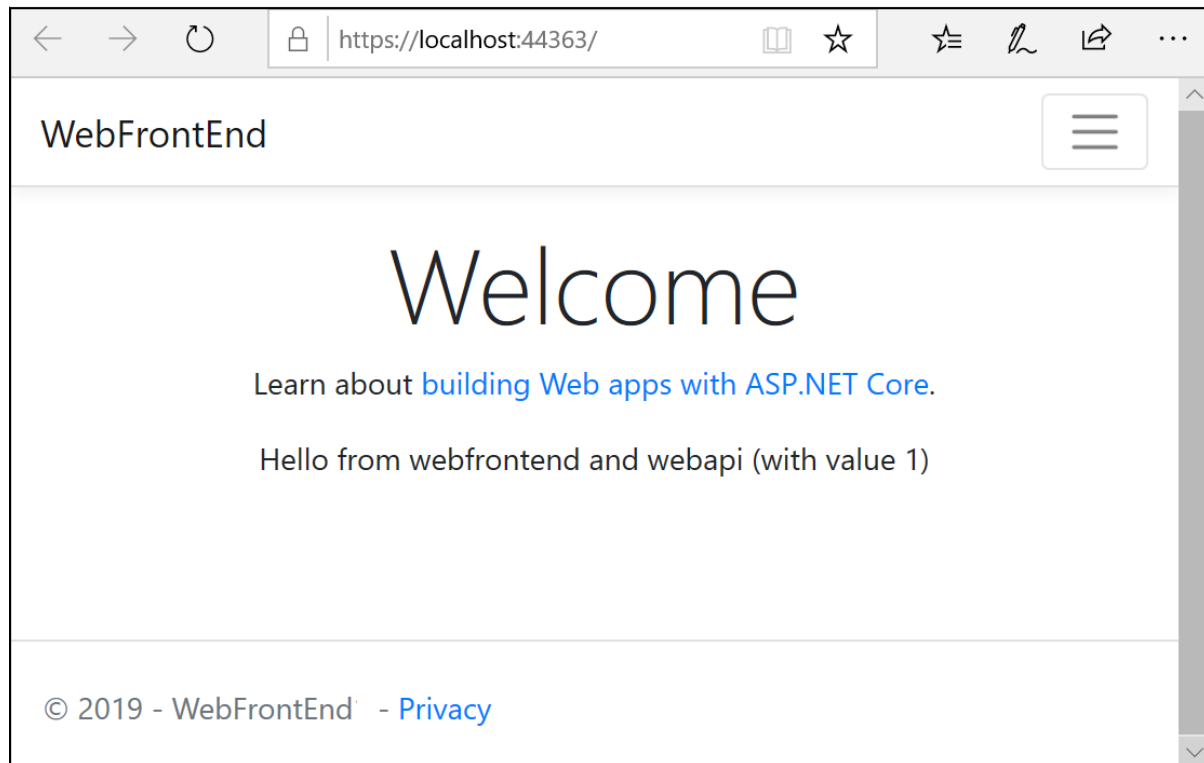
  mywebapi:
    image: ${DOCKER_REGISTRY-}mywebapi
    build:
      context: .
      dockerfile: MyWebAPI/Dockerfile
```

8. Run the site locally now (F5 or Ctrl+F5) to verify that it works as expected. If everything is configured correctly, you see the message "Hello from webfrontend and webapi (with value 1)."

The first project that you use when you add container orchestration is set up to be launched when you run or debug. You can configure the launch action in the **Project Properties** for the docker-compose project. On the docker-compose project node, right-click to open the context menu, and then choose **Properties**, or use Alt+Enter. The following screenshot shows the properties you would want for the solution used here. For example, you can change the page that is loaded by customizing the **Service URL** property.



Here's what you see when launched:



Next steps

Look at the options for deploying your [containers to Azure](#).

See also

[Docker Compose](#)

[Container Tools](#)

Get started with Visual Studio Kubernetes Tools

3/12/2019 • 4 minutes to read • [Edit Online](#)

The Visual Studio Kubernetes Tools help streamline the development of containerized applications targeting Kubernetes. Visual Studio can automatically create the configuration-as-code files needed to support Kubernetes deployment, such as Dockerfiles and Helm charts. You can debug your code in a live Azure Kubernetes Service (AKS) cluster using Azure Dev Spaces, or publish directly to an AKS cluster from inside Visual Studio.

This tutorial covers using Visual Studio to add Kubernetes support to a project and publish to AKS. If you are primarily interested in using [Azure Dev Spaces](#) to debug and test your project running in AKS, you can jump to the [Azure Dev Spaces tutorial](#) instead.

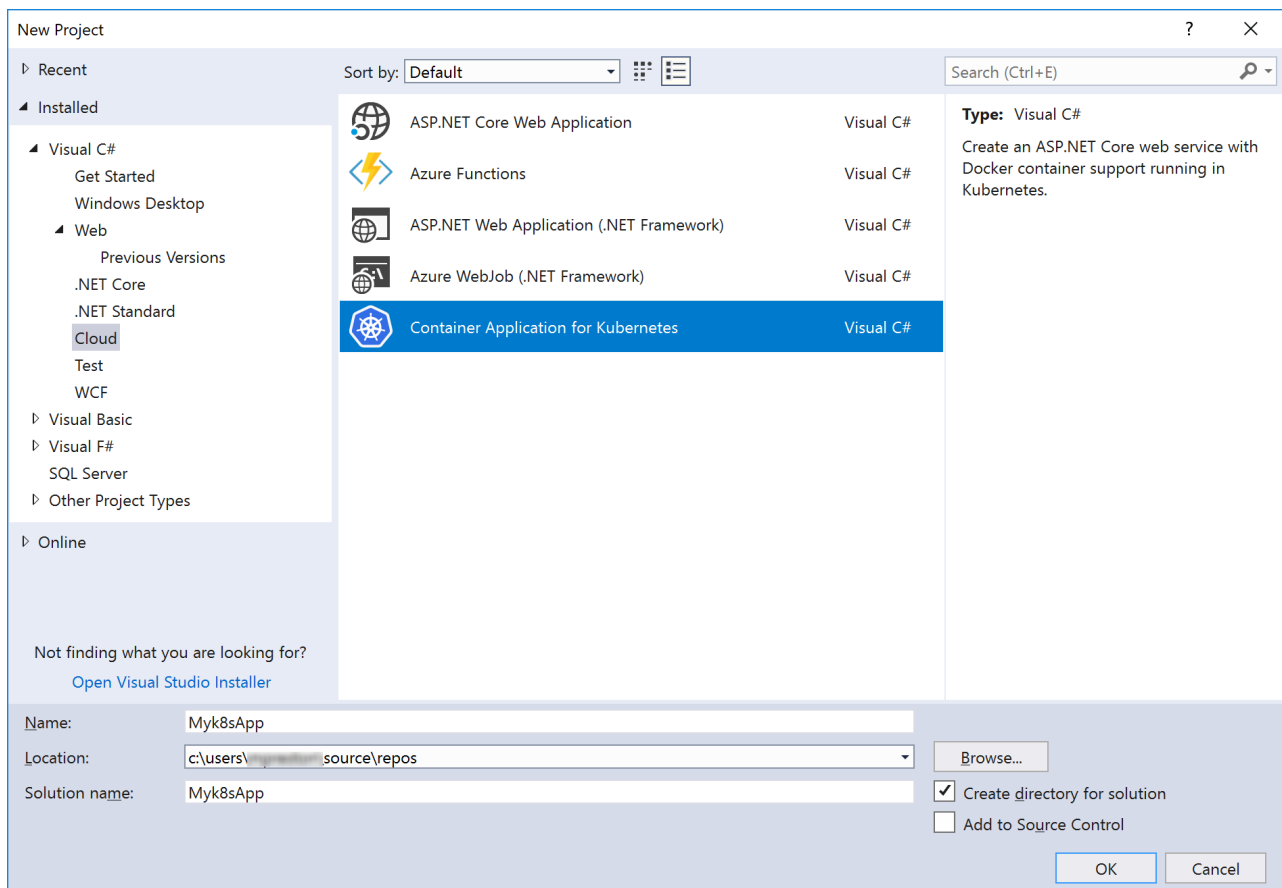
Prerequisites

To leverage this new functionality, you'll need:

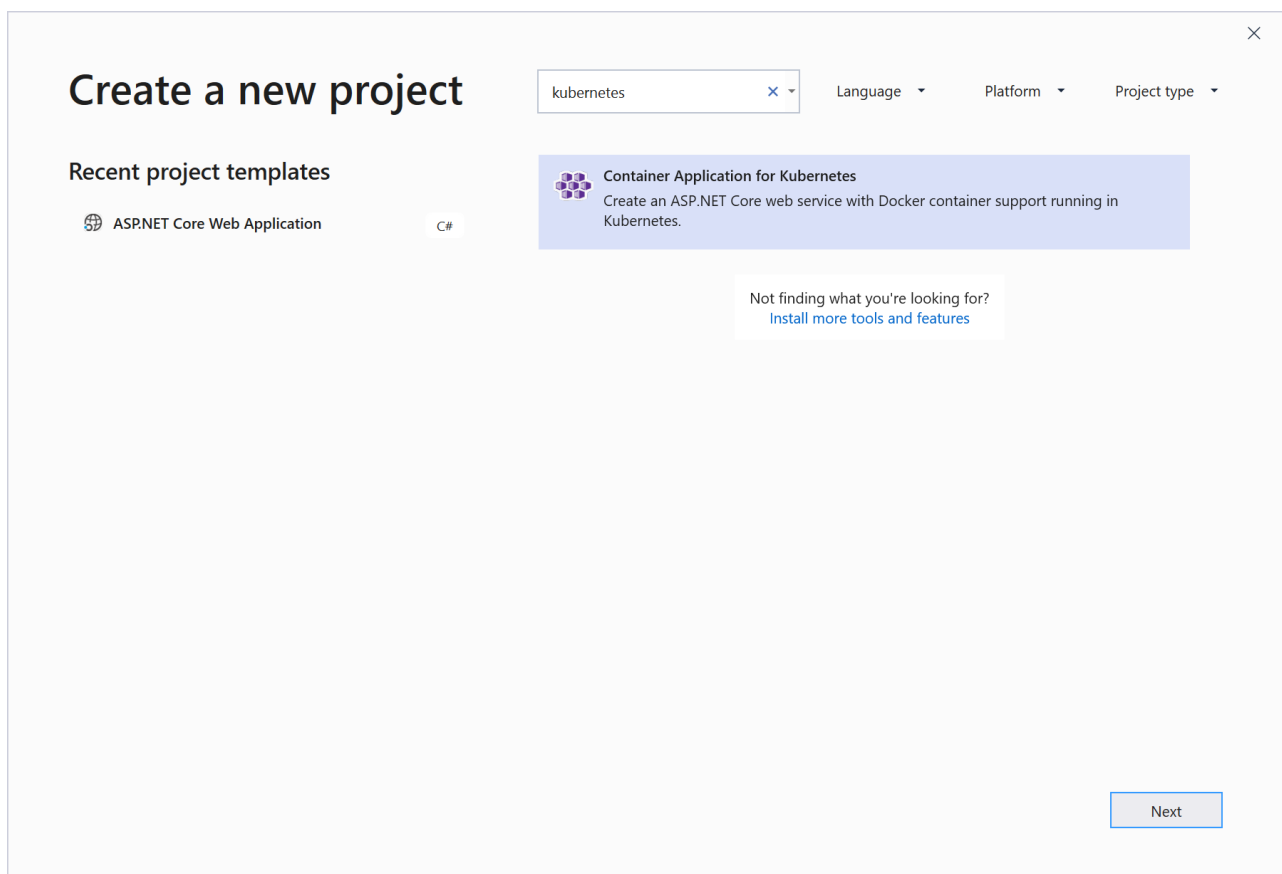
- The latest version of [Visual Studio 2017](#) with the *ASP.NET and web development* workload.
- The [Kubernetes tools for Visual Studio](#), available as a separate download.
- [Visual Studio 2019](#) with the *ASP.NET and web development* workload.
- [Docker Desktop](#) installed on your development workstation (that is, where you run Visual Studio), if you wish to build Docker images, debug Docker containers running locally, or publish to AKS. (Docker is *not* required for building and debugging Docker containers in AKS using Azure Dev Spaces.)
- If you wish to publish to AKS from Visual Studio (*not* required for debugging in AKS using Azure Dev Spaces):
 1. The [AKS publishing tools](#), available as a separate download.
 2. An Azure Kubernetes Service cluster. For more information, see [Creating an AKS cluster](#). Be sure to [connect to the cluster](#) from your development workstation.
 3. Helm CLI installed on your development workstation. For more information see [Installing Helm](#).
 4. Helm configured against your AKS cluster by using the `helm init` command. For more information on how to do this, see [How to configure Helm](#).

Create a new Kubernetes project

Once you have the appropriate tools installed, launch Visual Studio and create a new project. Under **Cloud**, choose the **Container Application for Kubernetes** project type. Select this project type and choose **OK**.



In the Visual Studio Start Window, search for *Kubernetes*, and choose the **Container Application for Kubernetes**.



Provide the project name.

×

Configure your new project

Container Application for Kubernetes

Project name

Kubernetes1

Location

C:\Source\Repos\

...

Solution name ⓘ

Kubernetes1

☐ Place solution and project in the same directory

BackCreate

You can then choose which type of ASP.NET Core web application to create. Choose **Web Application**. The usual **Enable Docker Support** option does not appear on this dialog. Docker support is enabled by default for a container application for Kubernetes.

New ASP.NET Core Web Application - Myk8sApp?×

Empty

API

Web Application

Web Application (Model-View-Controller)

Angular

React.js

React.js and Redux

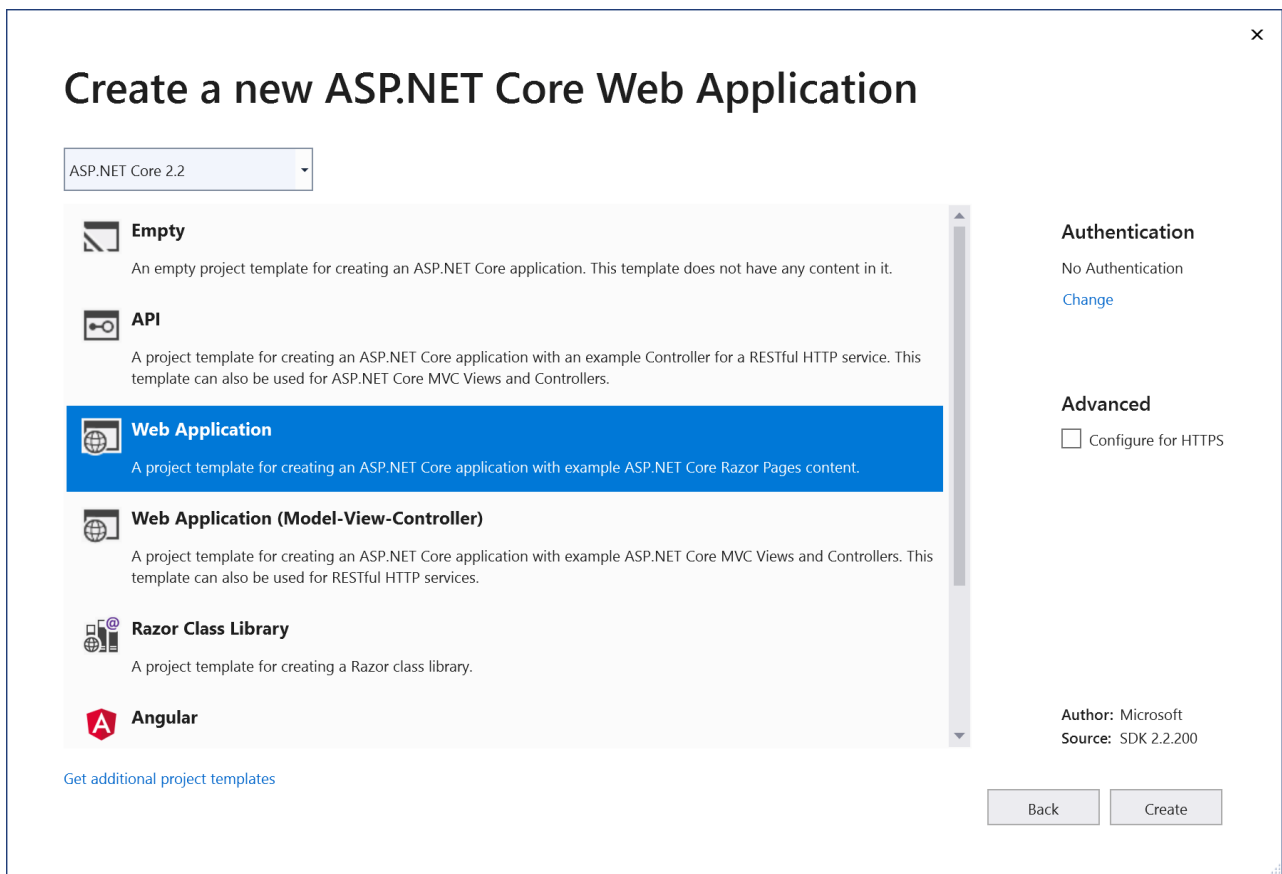
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content.

[Learn more](#)

Change Authentication

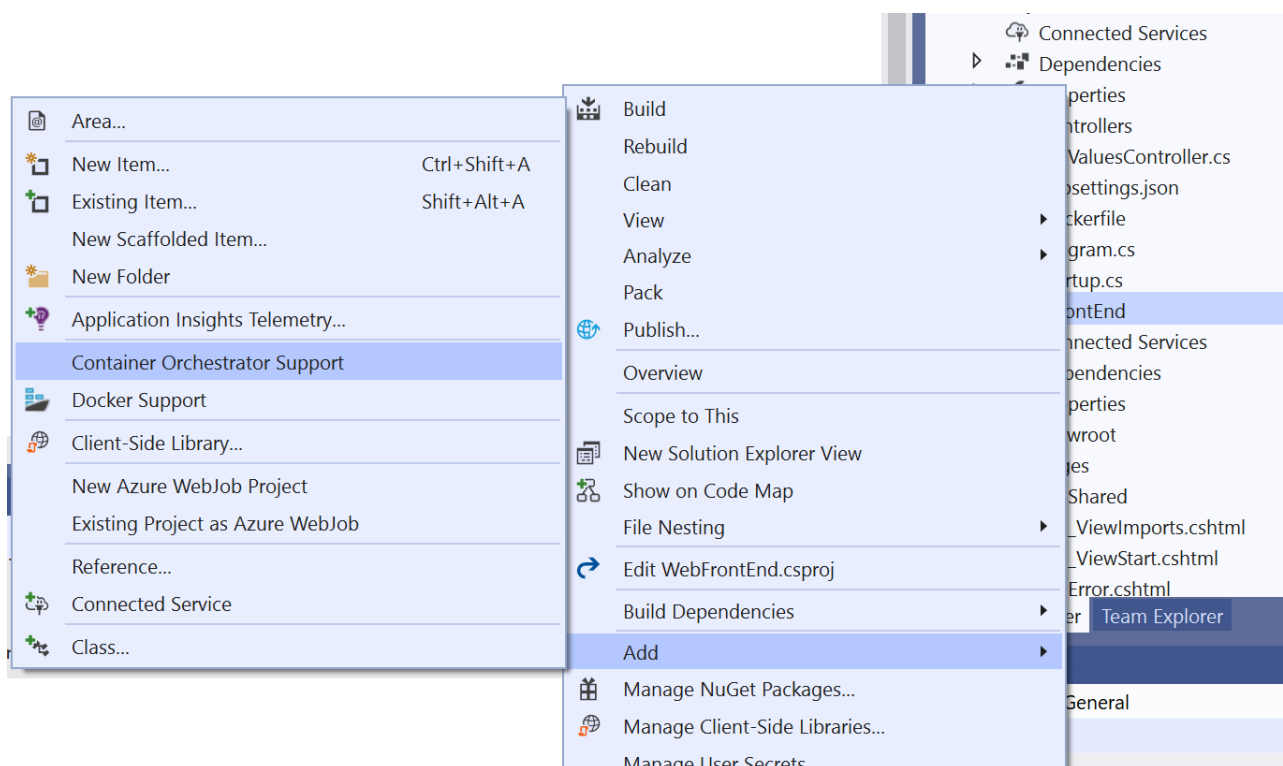
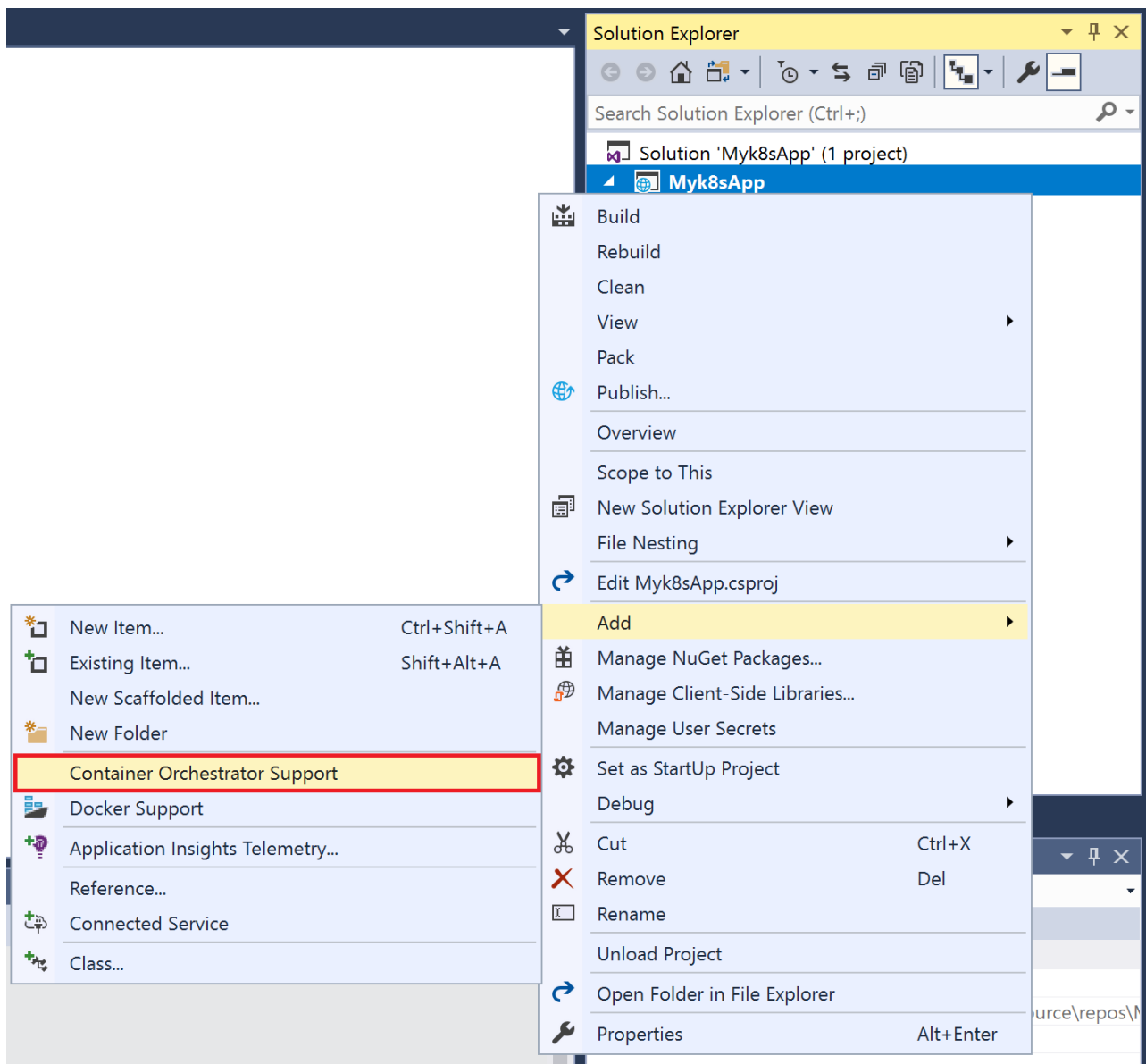
Authentication **No Authentication**

OKCancel

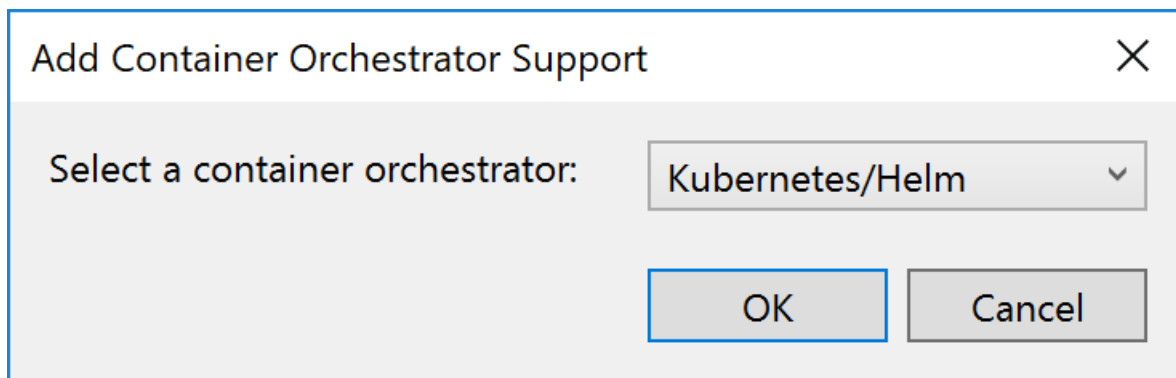


Add Kubernetes support to an existing project

Alternatively, you can add Kubernetes support to an existing ASP.NET Core web application project. To do this, right-click on the project, and choose **Add > Container Orchestrator Support**.

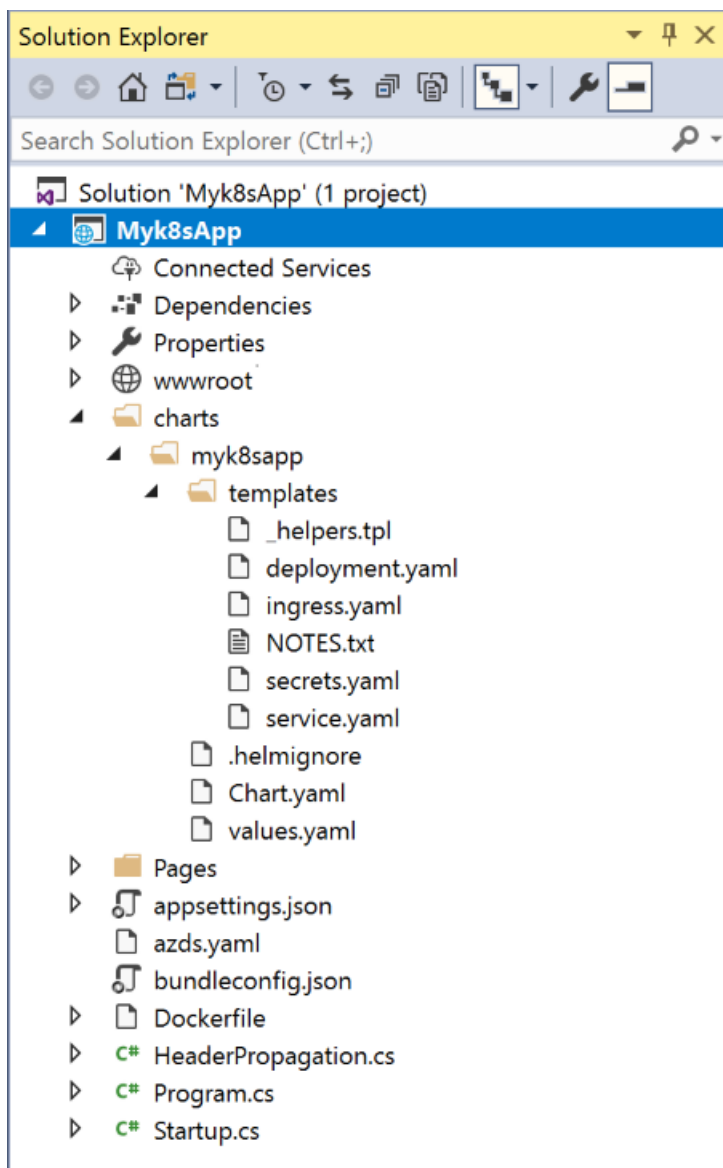


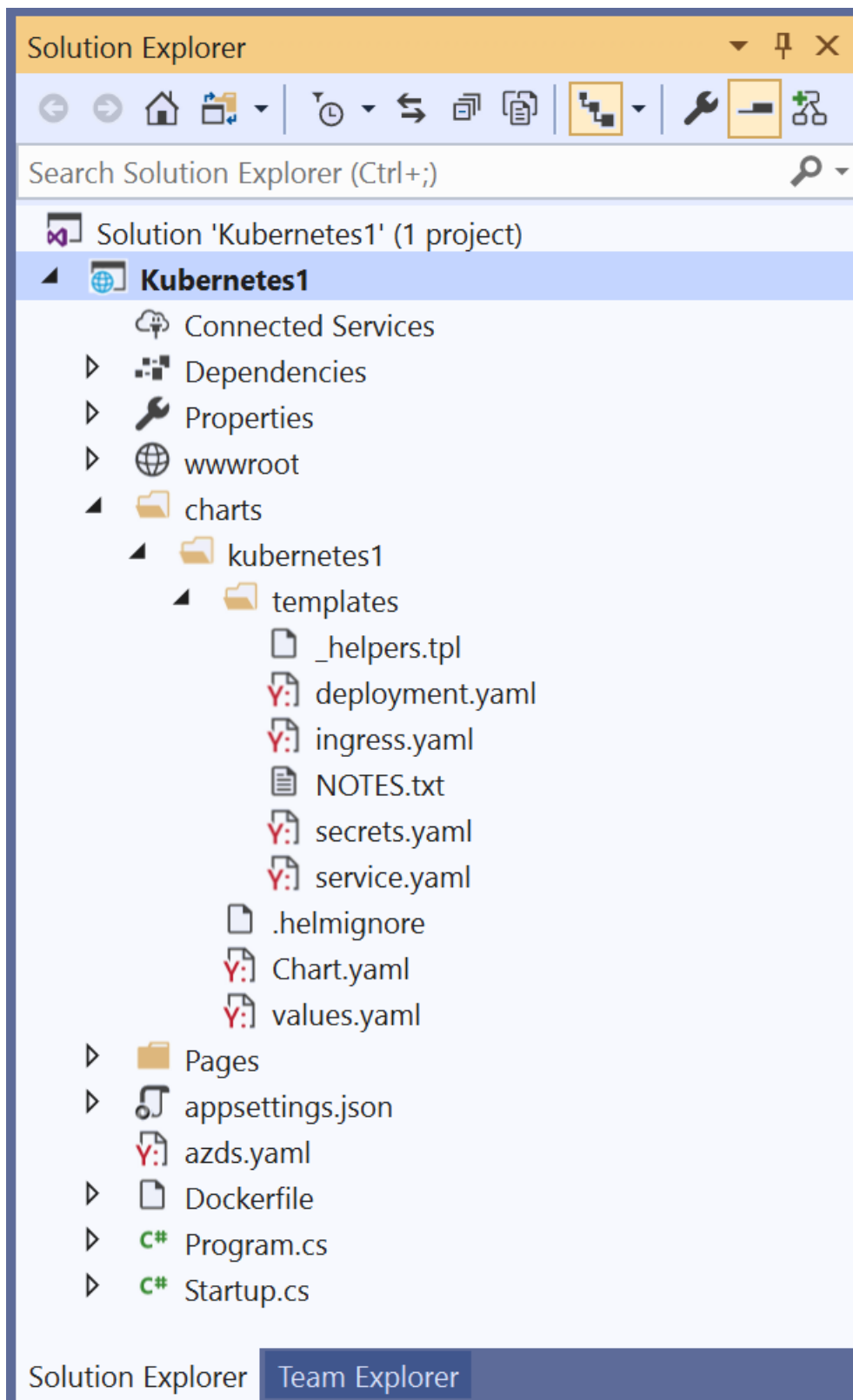
In the dialog box, select **Kubernetes/Helm** and choose **OK**.



What Visual Studio creates for you

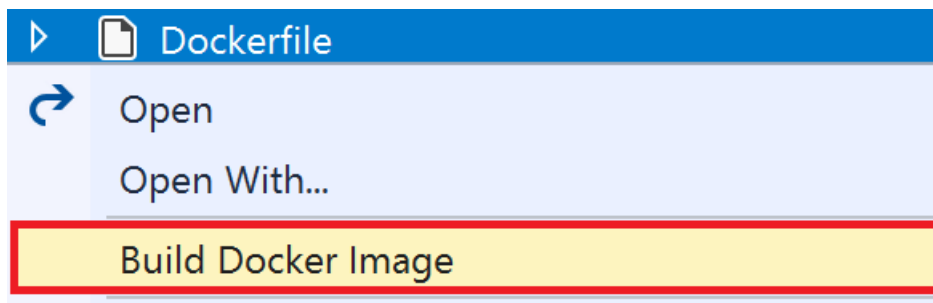
After creating a new **Container Application for Kubernetes** project or adding Kubernetes container orchestrator support to an existing project, you see some additional files in your project that facilitate deploying to Kubernetes.





The added files are:

- a Dockerfile, which allows you to generate a Docker container image hosting this web application. As you'll see, the Visual Studio tooling leverages this Dockerfile when debugging and deploying to Kubernetes. If you prefer to work directly with the Docker image, you can right-click on the Dockerfile and choose **Build Docker Image**.



- a Helm chart, and a *charts* folder. These yaml files make up the Helm chart for the application, which you can use to deploy it to Kubernetes. For more information on Helm, see <https://www.helm.sh>.
- *azds.yaml*. This contains settings for Azure Dev Spaces, which provides a rapid, iterative debugging experience in Azure Kubernetes Service. For more information, see [the Azure Dev Spaces documentation](#).

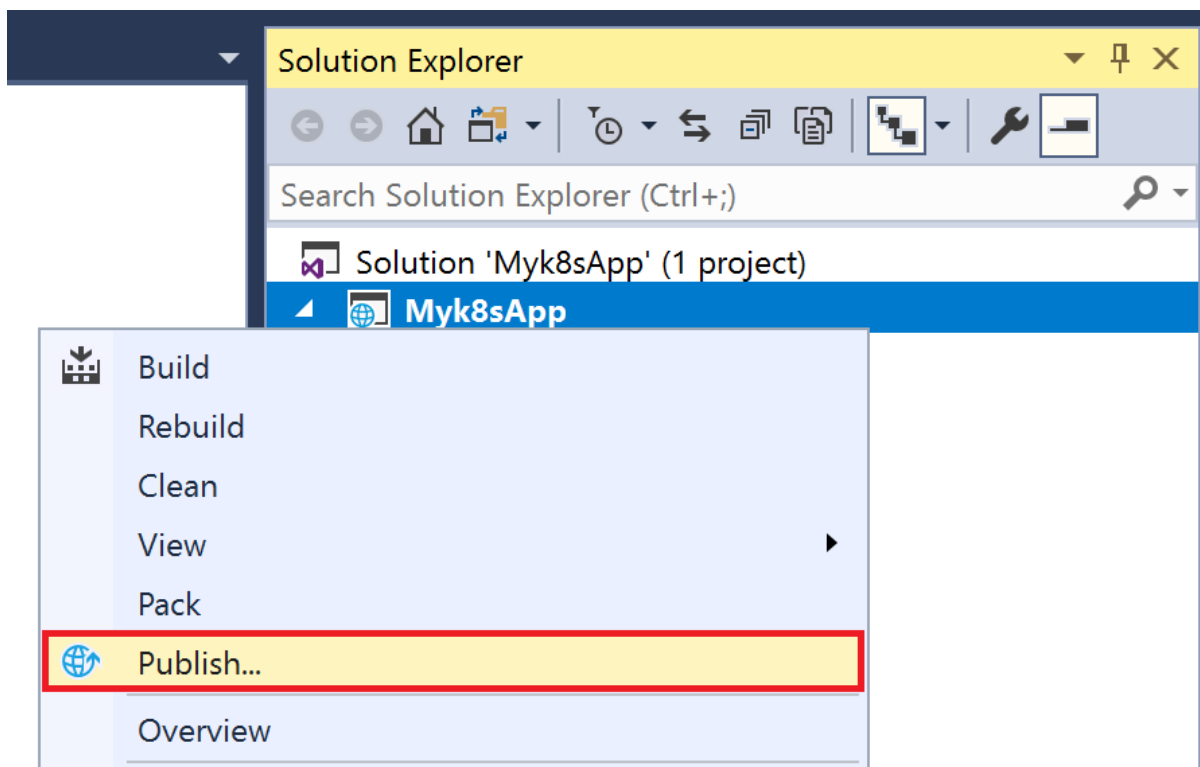
Publish to Azure Kubernetes Service (AKS)

With all these files in place, you can use the Visual Studio IDE to write and debug your application code, just as you always have. You can also use [Azure Dev Spaces](#) to quickly run and debug your code running live in an AKS cluster. For more information, please reference the [Azure Dev Spaces tutorial](#)

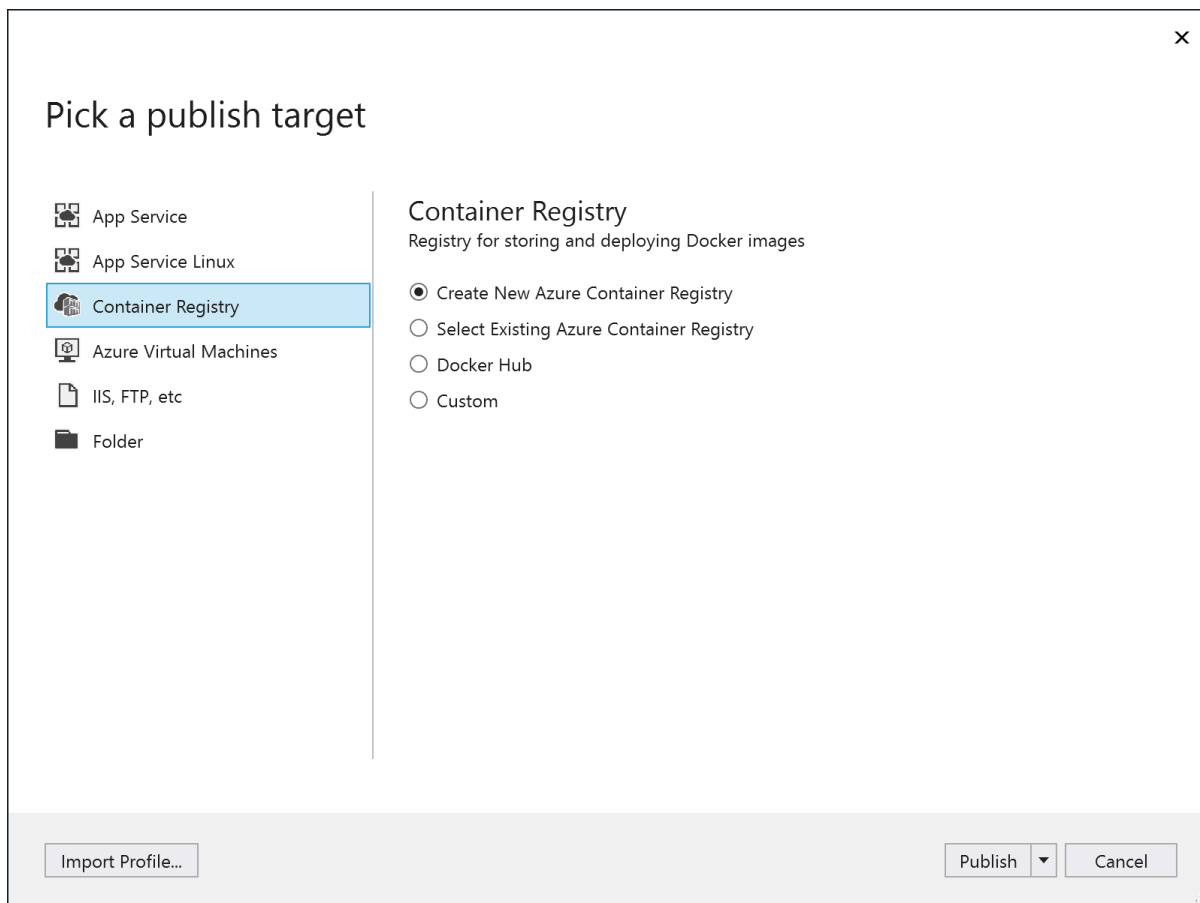
Once you have your code running the way you want, you can publish directly from Visual Studio to an AKS cluster.

To do this, you first need to double-check that you've installed everything as described in the [Prerequisites](#) section under the item for publishing to AKS, and run through all the command line steps given in the links. Then, set up a publish profile that publishes your container image to Azure Container Registry (ACR). Then AKS can pull your container image from ACR and deploy it into the cluster.

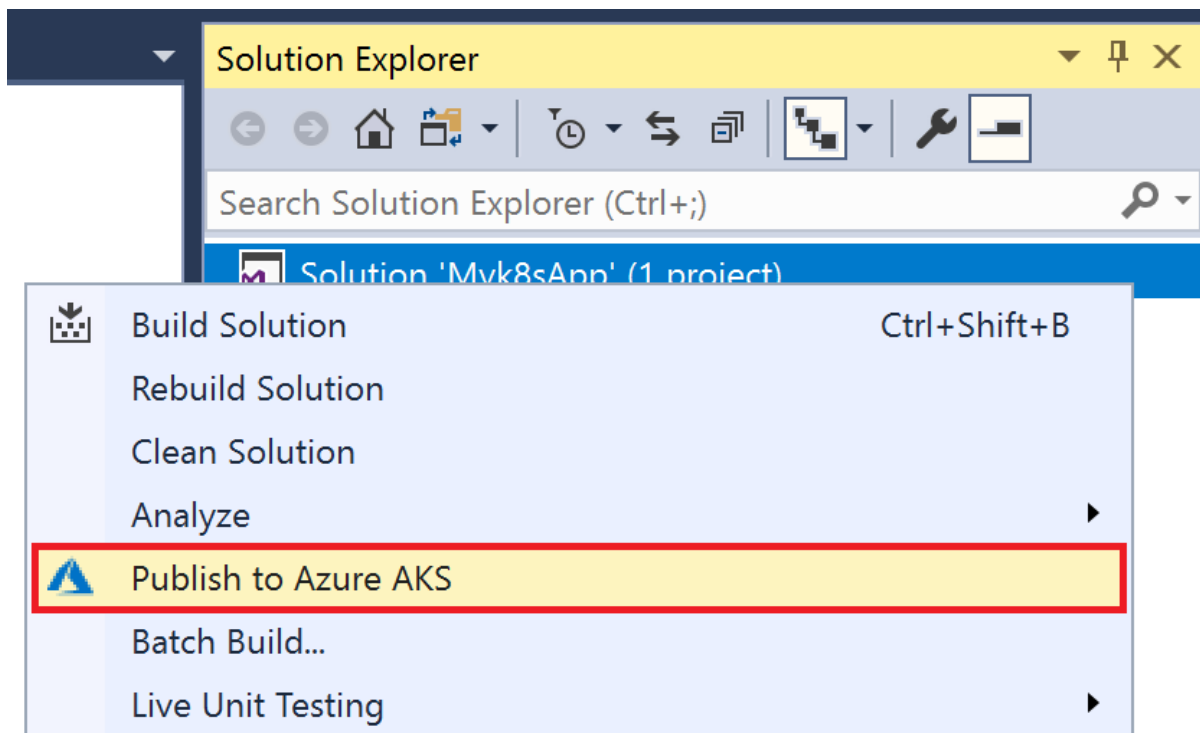
1. In **Solution Explorer**, right-click on your *project* and choose **Publish**.



2. In the **Publish** screen, choose **Container Registry** as the publish target, and follow the prompts to select your container registry. If you don't already have a container registry, choose **Create New Azure Container Registry** to create one from Visual Studio. For more information, see [Publish your container to Azure Container Registry](#).



3. Back in Solution Explorer, right click on your *solution* and click **Publish to Azure AKS**.



4. Choose your subscription and your AKS cluster, along with the ACR publish profile that you just created. Then click **OK**.

Microsoft account

mia_preston1@outlook.com

Publish to AKS

Publish your application using Helm charts

Subscription:

Azure

AKS Cluster:

MyNewAksCluster

[How to create an AKS cluster](#)

Container Registry:

vspublishtest

OK

Cancel

This takes you to the **Publish to Azure AKS** screen.

5. Choose the **Configure Helm** link to update the command line used to install the Helm charts on the server.

Server Explorer

Toolbox

AKS Cluster Management (Myk8sApp)

Dockerfile

Publish to Azure AKS

MyNewAksCluster

Publish

[New Profile...](#)

Publish Method:

Helm Install

[Configure Helm...](#)

AKS Cluster:

MyNewAksCluster

Container Registry:

vspublishtest

kubectl Context:

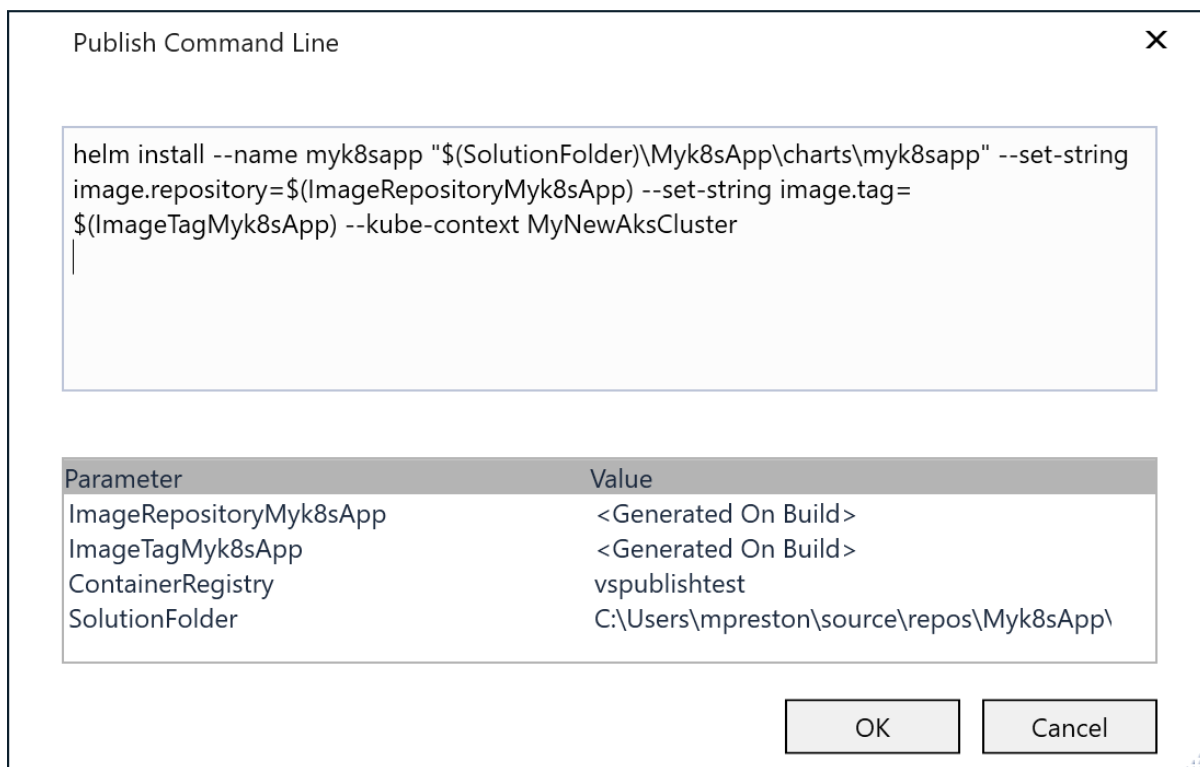
MyNewAksCluster

Projects to Publish:

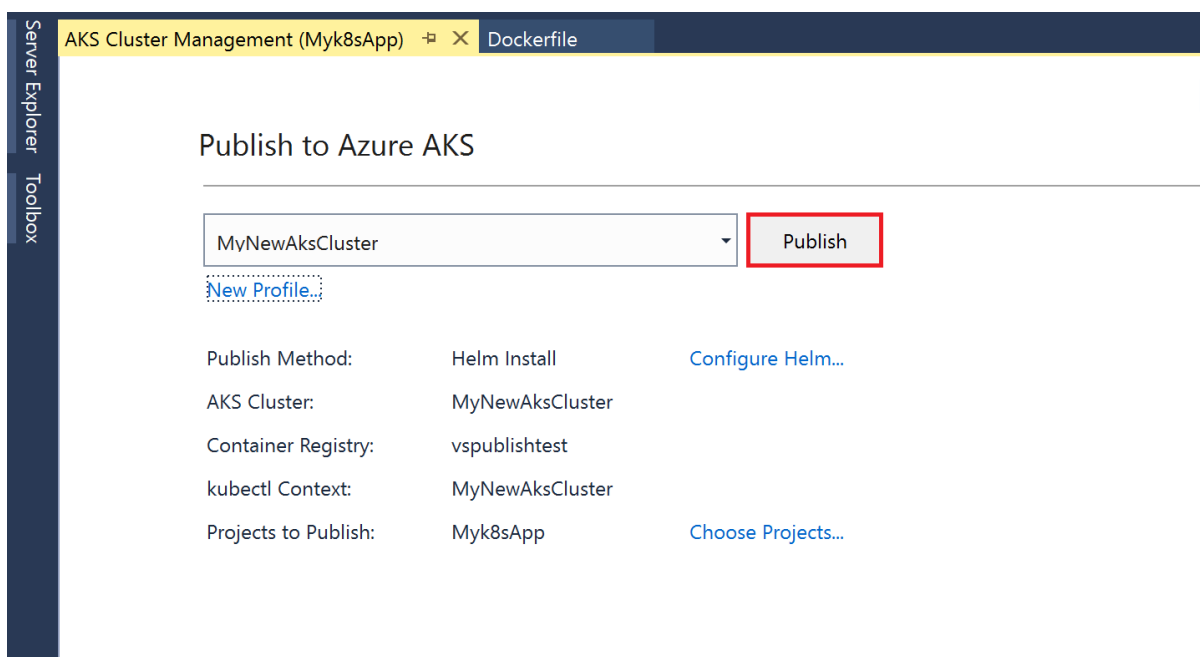
Myk8sApp

[Choose Projects...](#)

Updating the command line is useful if there are custom command line arguments that you wish to specify, such as a different Kubernetes context or chart name.



6. When you are ready to deploy, click the **Publish** button to publish your application to AKS.



Congratulations! You can now use the full power of Visual Studio for all your Kubernetes app development.

Next steps

Learn more about Kubernetes development on Azure by reading the [AKS documentation](#).

Learn more about Azure Dev Spaces by reading the [Azure Dev Spaces documentation](#)

Debugging apps in a local Docker container

3/6/2019 • 3 minutes to read • [Edit Online](#)

Overview

Visual Studio provides a consistent way to develop in a Docker container and validate your application locally. You don't have to restart the container each time you make a code change. This article illustrates how to use the "Edit and Refresh" feature to start an ASP.NET Core Web app in a local Docker container, make any necessary changes, and then refresh the browser to see those changes. This article also shows you how to set breakpoints for debugging.

Prerequisites

The following tools must be installed.

- [Visual Studio 2017](#) with the Web Development workload installed.
- [Visual Studio 2019](#) with the Web Development workload installed.

To run Docker containers locally, you'll need a local docker client. You can use the [Docker Toolbox](#), which requires Hyper-V to be disabled, or you can use [Docker for Windows](#), which uses Hyper-V, and requires Windows 10.

If using Docker Toolbox, you'll need to [configure the Docker client](#)

1. Create a web app

1. From the Visual Studio menu, select **File > New > Project**.
 2. Under the **Templates** section of the **New Project** dialog box, select **Visual C# > Web**.
 3. Select **ASP.NET Core Web Application**.
 4. Give your new application a name (or take the default) and select **OK**.
 5. Select **Web Application**.
 6. Check the **Enable Docker Support** checkbox.
 7. Select the type of container you want (Windows or Linux) and click **OK**.
-
1. From the Visual Studio Start Window, choose **Create a New Project**.
 2. Choose **ASP.NET Core Web Application**, and choose **Next**.
 3. Give your new application a name (or take the default) and choose **Create**.
 4. Choose **Web Application**.
 5. Choose whether or not you want SSL support by using the **Configure for HTTPS** checkbox.
 6. Check the **Enable Docker Support** checkbox.
 7. Select the type of container you want (Windows or Linux) and click **Create**.

2. Edit your code and refresh

To quickly iterate changes, you can start your application within a container, and continue to make changes, viewing them as you would with IIS Express.

1. Set the Solution Configuration to Debug and press Ctrl+F5 to build your docker image and run it locally.

Once the container image has been built and is running in a Docker container, Visual Studio launches the web app in your default browser.

2. Go to the Index page, which is where we're going to make our changes.
3. Return to Visual Studio and open `Index.cshtml`.
4. Add the following HTML content to the end of the file and save the changes.

```
<h1>Hello from a Docker Container!</h1>
```

5. Viewing the output window, when the .NET build is completed and you see these lines, switch back to your browser and refresh the page.

```
Now listening on: http://*:80  
Application started. Press Ctrl+C to shut down
```

6. Your changes have been applied!

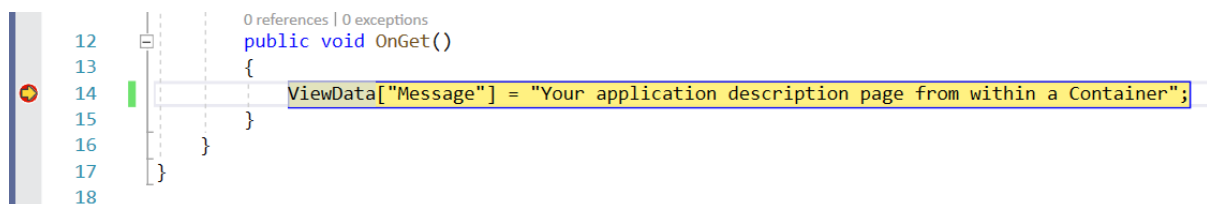
3. Debug with breakpoints

Often, changes will need further inspection, leveraging the debugging features of Visual Studio.

1. Return to Visual Studio and open `Index.cshtml.cs`.
2. Replace the contents of the `OnGet` method with the following:

```
ViewData["Message"] = "Your application description page from within a Container";
```

3. Set a breakpoint to the left of the code line.
4. Hit F5 to start debugging and hit the breakpoint.
5. Switch to Visual Studio to view the breakpoint, inspect values, and so on.



Summary

With Docker support in Visual Studio, you can get the productivity of working locally, with the production realism of developing within a Docker container.

Troubleshooting

[Troubleshooting Visual Studio Docker Development](#)

More about Docker with Visual Studio, Windows, and Azure

- [Container development with Visual Studio](#) - a container development landing page
- [Docker Integration for Azure Pipelines](#) - Build and Deploy docker containers
- [Windows Container Information](#) - Windows Server and Nano Server information
- [Azure Kubernetes Service - Azure Kubernetes Service Documentation](#)

Deploy an ASP.NET container to a container registry using Visual Studio

2/4/2019 • 2 minutes to read • [Edit Online](#)

Overview

Docker is a lightweight container engine, similar in some ways to a virtual machine, which you can use to host applications and services. This tutorial walks you through using Visual Studio to publish your containerized application to an [Azure Container Registry](#).

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

To complete this tutorial:

- Install the latest version of [Visual Studio 2017](#) with the "ASP.NET and web development" workload
- Install [Docker for Windows](#)

1. Create an ASP.NET Core web app

The following steps guide you through creating a basic ASP.NET Core app that will be used in this tutorial.

1. From the Visual Studio menu, select **File > New > Project**.
2. Under the **Templates** section of the **New Project** dialog box, select **Visual C# > Web**.
3. Select **ASP.NET Core Web Application**.
4. Give your new application a name (or take the default) and select **OK**.
5. Select **Web Application**.
6. Check the **Enable Docker Support** checkbox.
7. Select the type of container you want (Windows or Linux) and click **OK**.


2. Publish your container to Azure Container Registry

1. Right-click your project in **Solution Explorer** and choose **Publish**.
2. On the publish target dialog, select the **Container Registry** tab.
3. Choose **New Azure Container Registry** and click **Publish**.
4. Fill in your desired values in the **Create a new Azure Container Registry**.

SETTING	SUGGESTED VALUE	DESCRIPTION
DNS Prefix	Globally unique name	Name that uniquely identifies your container registry.
Subscription	Choose your subscription	The Azure subscription to use.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource Group	myResourceGroup	Name of the resource group in which to create your container registry. Choose New to create a new resource group.
SKU	Standard	Service tier of the container registry
Registry Location	A location close to you	Choose a Location in a region near you or near other services that will use your container registry.

Create a new Azure Container Registry



DNS Prefix

MyWebApplication20180426102127

Subscription

Visual Studio Enterprise

Resource Group

myResourceGroup*

New...

SKU

Standard

Registry Location

West US

Export...

Create

Cancel

Explore additional Azure services

Create App Service

Create a new Azure Container Registry

Create a SQL Database

Create a storage account

Clicking the Create button will create the following Azure resources

Azure Container Registry - MyWebApplication20180426102127

5. Click **Create**

You can now pull the container from the registry to any host capable of running Docker images, for example [Azure Container Instances](#).

Configure a Docker Host with VirtualBox

2/4/2019 • 2 minutes to read • [Edit Online](#)

Overview

This article guides you through configuring a default Docker instance using Docker Machine and VirtualBox. If you're using the [Docker for Windows](#), this configuration is not necessary.

Prerequisites

The following tools need to be installed.

- [Docker Toolbox](#)

Configuring the Docker client with Windows PowerShell

To configure a Docker client, simply open Windows PowerShell, and perform the following steps:

1. Create a default docker host instance.

```
docker-machine create --driver virtualbox default
```

2. Verify the default instance is configured and running. (You should see an instance named 'default' running.

```
docker-machine ls
```



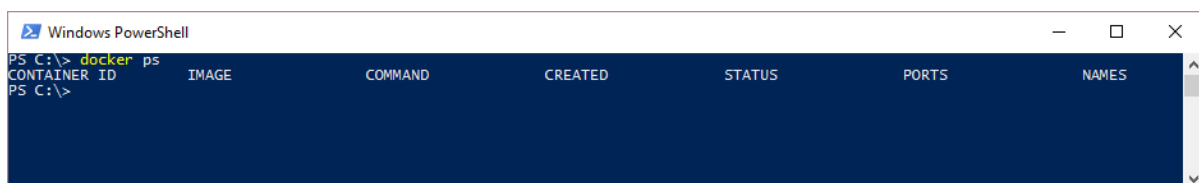
NAME	ACTIVE	DRIVER	STATE	URL	SWARM	DOCKER	ERRORS
default	*	virtualbox	Running	tcp://192.168.99.100:2376		v1.10.2	

3. Set default as the current host, and configure your shell.

```
docker-machine env default | Invoke-Expression
```

4. Display the active Docker containers. The list should be empty.

```
docker ps
```



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

NOTE

Each time you reboot your development machine, you'll need to restart your local docker host. To do this, issue the following command at a command prompt: `docker-machine start default`.

Troubleshoot Visual Studio 2017 development with Docker

2/4/2019 • 2 minutes to read • [Edit Online](#)

When you're working with Visual Studio Tools for Docker, you may encounter issues while building or debugging your application. Below are some common troubleshooting steps.

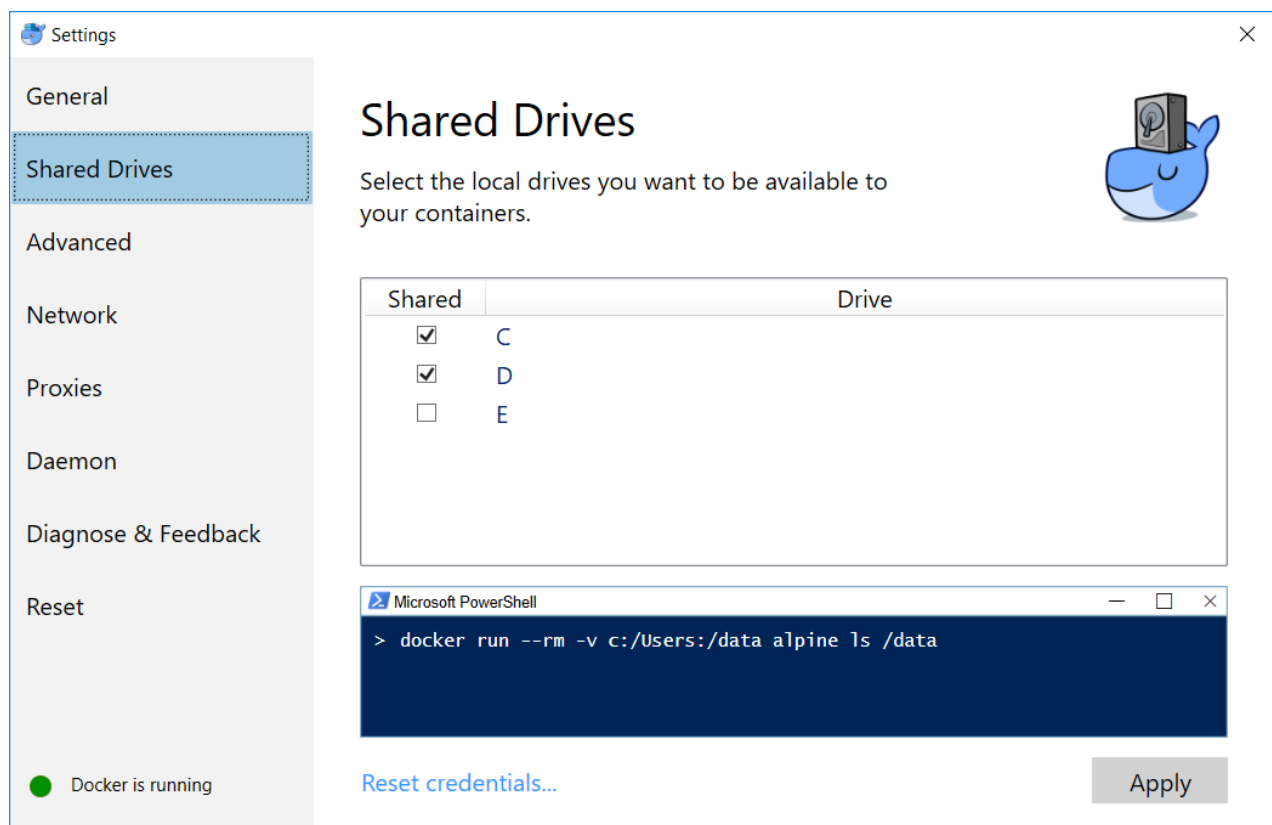
Volume sharing is not enabled. Enable volume sharing in the Docker CE for Windows settings (Linux containers only)

To resolve this issue:

1. Right-click **Docker for Windows** in the notification area, and then select **Settings**.
2. Select **Shared Drives** and share the system drive along with the drive where the project resides.

NOTE

If files appear shared, you may still need to click the "Reset credentials..." link at the bottom of the dialog in order to re-enable volume sharing. To continue after you reset credentials, you might have to restart Visual Studio.



TIP

Visual Studio 2017 versions 15.6 and later prompt when **Shared Drives** aren't configured.

Container type

When adding Docker support to a project, you choose either a Windows or a Linux container. The Docker host

must be running the same container type. To change the container type in the running Docker instance, right-click the System Tray's Docker icon and choose **Switch to Windows containers...** or **Switch to Linux containers...**

Unable to start debugging

One reason could be related to having stale debugging components in your user profile folder. Execute the following commands to remove these folders so that the latest debugging components are downloaded on the next debug session.

- `del %userprofile%\vsdbg`
- `del %userprofile%\onecoremsvsmon`

Errors specific to networking when debugging your application

Try executing the script downloadable from [Cleanup Container Host Networking](#), which will refresh the network-related components on your host machine.

Mounts denied

When using Docker for macOS, you might encounter an error referencing the folder `/usr/local/share/dotnet/sdk/NuGetFallbackFolder`. Add the folder to the File Sharing tab in Docker

Microsoft/DockerTools GitHub repo

For any other issues you encounter, see [Microsoft/DockerTools](#) issues.