

Unit 9

HOSTING AND DEPLOYING ASP.NET CORE APPLICATION

HOSTING AND DEPLOYING ASP.NET CORE APPLICATION

- Once you successfully developed your web application, you may require to host the application to the server so that other people can access it. The process of deploying/installing an application into the server is called "Hosting".

Web Server

- A web server is a process for hosting web applications, which responds to HTTP requests and delivers contents and services. A web server allows an application to process messages that arrive through specific TCP ports (by default). Default port for HTTP traffic is 80, and the one for HTTPS is 443.
- When you visit a website in your browser, you don't typically specify the port number unless the web server is configured to receive traffic on ports other than the default.
- Some of the web servers that we can use to host ASP.NET Core are: Microsoft IIS, Apache, NGINX.

IIS web server

- The IIS web server comes from the Microsoft stable and runs only on the Microsoft Windows operating system. It is actually not free, since it comes as a part of the Windows operating system. You might feel comfortable with IIS if you have already used the Windows OS ecosystem. It also comes with the support of the .NET framework which was released by Microsoft and support services for IIS are provided directly by Microsoft.

Advantages of IIS

- Has the support of Microsoft.
- You can have access to the .NET framework along with ASPX scripts.
- Can be easily integrated with other Microsoft services like ASP, MS SQL etc.

Apache web server

- Apache is an open source web server which was developed and maintained Apache Software Foundation. It is a result of the collaborative efforts which was aimed at creating a robust and secure commercial grade web server which adhered to all the HTTP standards.
- It has been the market leader since it entered the web server market in 1995 and remains the web server of choice for its ability to function across multiple platforms.
- Apache is equally efficient on almost every operating system but finds can be found to be in maximum use when combined with Linux.

Advantages of Apache

- As it open source, so there are no licensing fees.
- It is flexible, meaning that you can choose the modules you want.
- Has a high level of security.
- Strong user community to provide backend support.
- Runs equally well on UNIX, Linux, MacOS, Windows.

NGINX

- NGINX is a robust web server which was developed by Russian developer Igor Sysoev. It is a free open-source HTTP server which can be used as a mail proxy, reverse proxy server when required. Most importantly, it can take care of a huge number of concurrent users with minimal resources in an efficient manner. NGINX, is particularly of great help when the situation of handling massive web traffic arises.
- NGINX has a lightweight architecture and is highly efficient. This is probably the only web server which can handle huge traffic with very limited hardware resources. NGINX acts as a sort of shock absorber which protects Apache servers when faced with security vulnerabilities and sudden traffic spikes.

Advantages of NGINX

- Open source.
- A high speed web server which can be used as a reverse-proxy server.
- Can be used better in a virtual private server environment.

HOSTING MODELS IN ASP.NET CORE

- There are 2 types of hosting models in ASP.NET Core:
 - Out-of-process Hosting
 - In-process Hosting
- Before ASP.Net Core 2.2 we have only one hosting model, which is Out-of-process but after due to the performance we have In Process Hosting Model in 2.2+ versions.

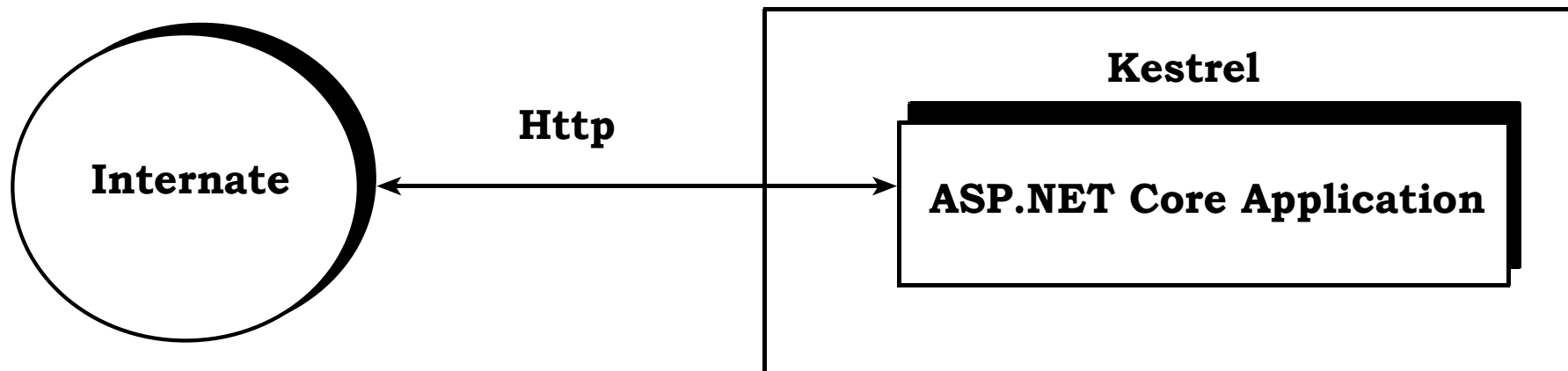
Out-of-process Hosting Model

- In Out-of-process hosting models, we can either use the Kestrel server directly as a user request facing server or we can deploy the app into IIS which will act as a proxy server and sends requests to the internal Kestrel server. In this type of hosting model, we have two options:
 - Using Kestrel
 - Using Proxy Server

Out-of-process Hosting Model

Using Kestrel

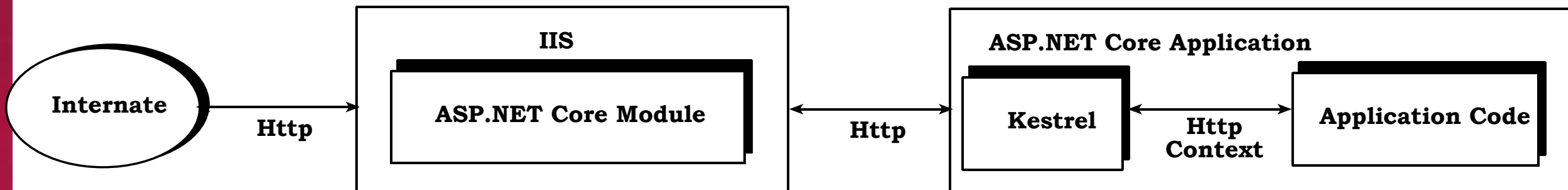
- Kestrel is a cross-platform web server for ASP.NET Core. Kestrel is the webserver that's included by default in ASP.NET Core project templates.
- Kestrel itself acts as edge server which directly server user requests. It means that we can only use the Kestrel server for our application.



Out-of-process Hosting Model

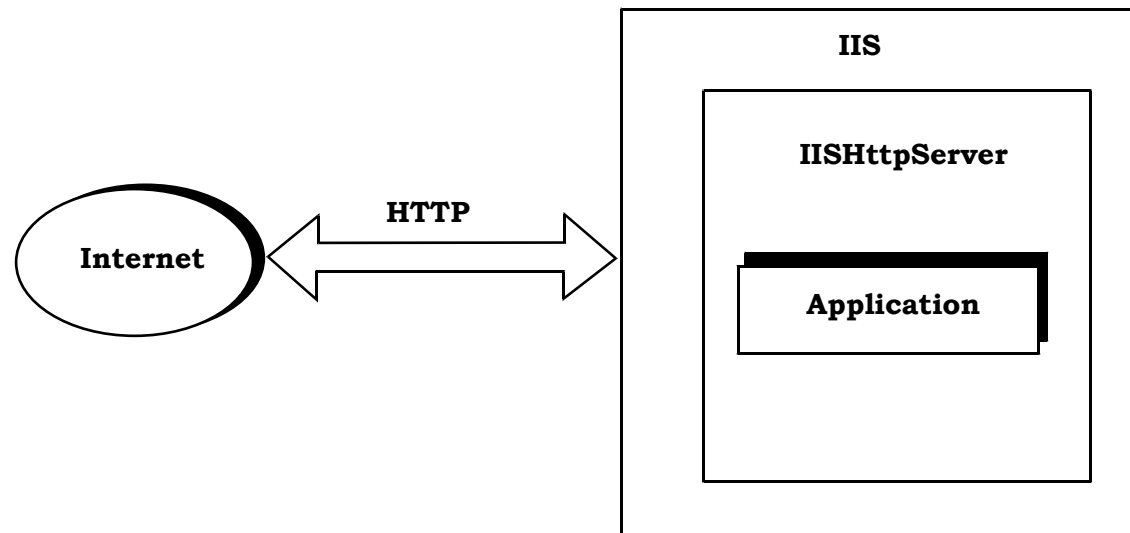
Using a Proxy Server

- Due to limitations of the Kestrel server, we cannot use this in all the apps. In such cases, we have to use powerful servers like IIS, NGINX or Apache. So, in that case, this server acts as a reverse proxy server which redirects every request to the internal Kestrel sever where our app is running. Here, two servers are running. One is IIS and another is Kestrel.
- This model is a default model for all the applications implemented before .NET Core 2.2. But there are some of the limitations of using this type such as performance slowness.



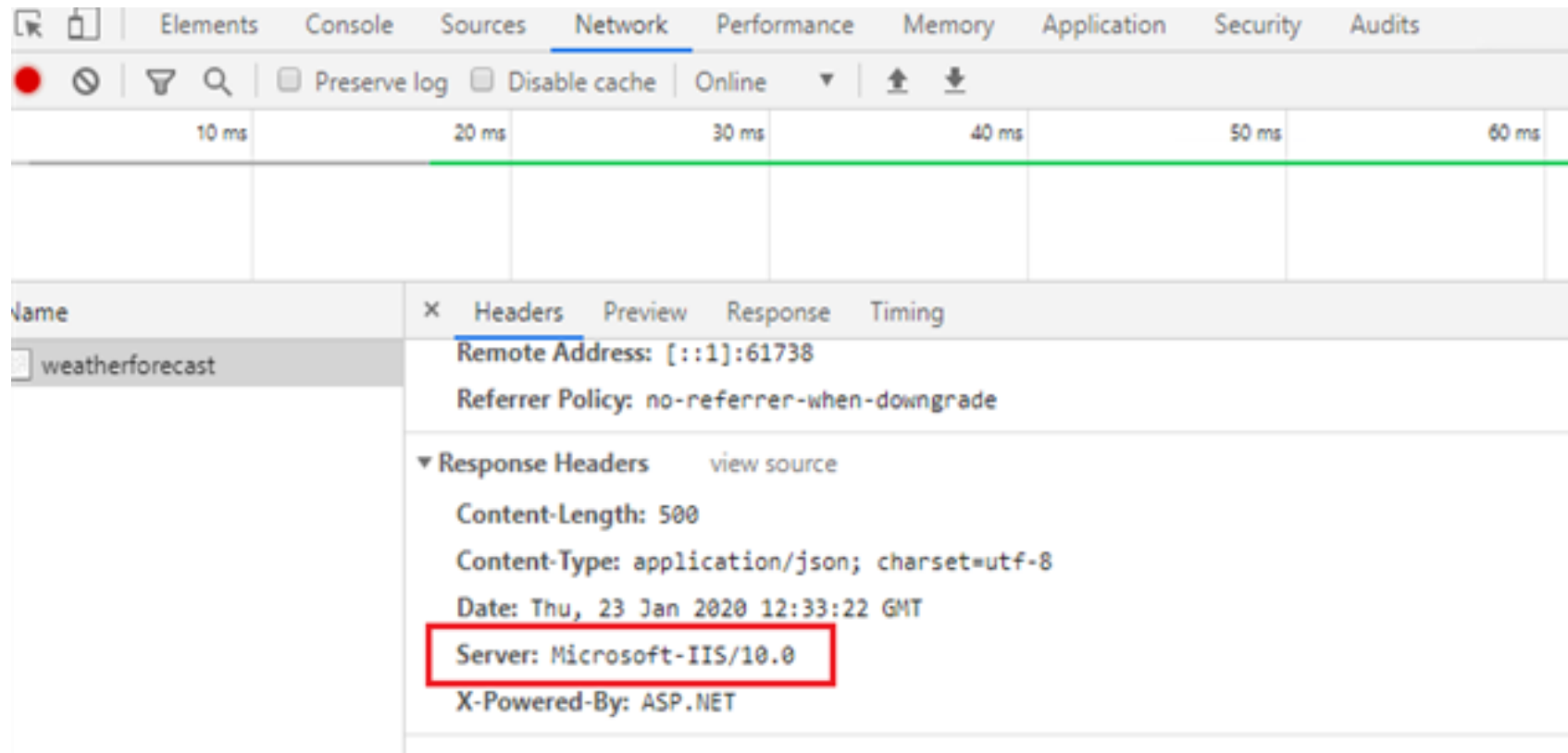
In-process Hosting Model

- After the release of .NET Core 2.2, it introduced a new type of hosting which is called In-process hosting. In this type, only one server is used for hosting like IIS, Nginx or Linux. It means that the App is directly hosted inside of IIS. No Kestrel server is being used. IIS HTTP Server (IISHttpServer) is used instead of the Kestrel server to host apps in IIS directly. ASP.NET Core 3.1 onwards In-process hosting model is used as a default model whenever you create a new application using an existing template.



Let's see different types of hosting models

- Now let's see how to check which hosting model is being used.
- Run the application on the IISExpress server, then open the browser's network tab and check for the first call. Under the server section, you will be able to see it showing Microsoft IIS.

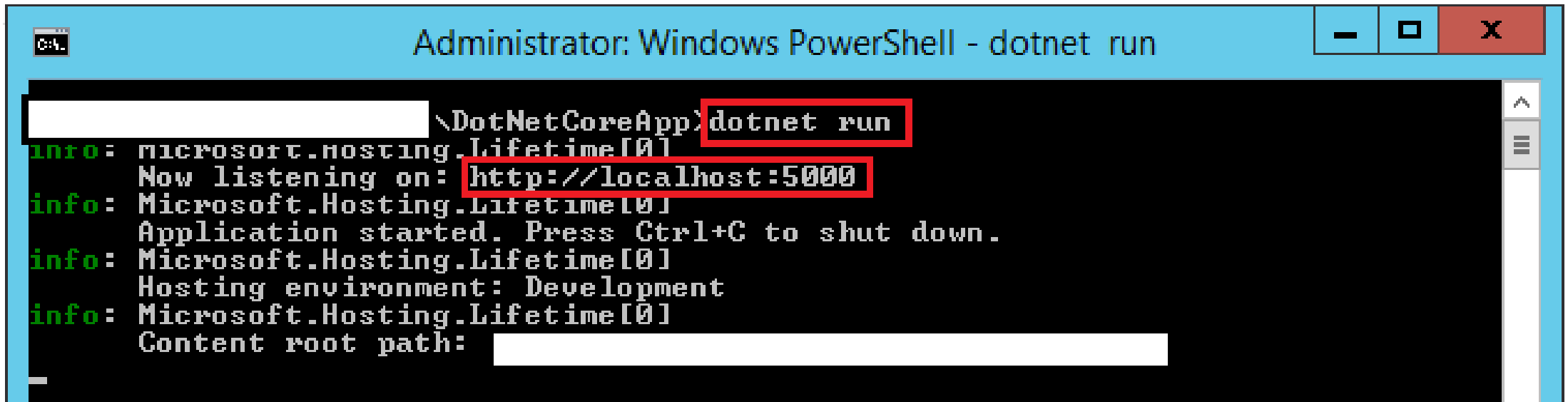


The screenshot shows the Chrome DevTools Network tab. The 'Headers' sub-tab is selected, displaying the response headers for a request named 'weatherforecast'. The 'Server' header is highlighted with a red box and contains the value 'Microsoft-IIS/10.0'. Other visible headers include 'Remote Address: [::1]:61738', 'Referrer Policy: no-referrer-when-downgrade', 'Content-Length: 500', 'Content-Type: application/json; charset=utf-8', and 'Date: Thu, 23 Jan 2020 12:33:22 GMT'. The 'X-Powered-By' header is also visible with the value 'ASP.NET'.

Name	Value
Remote Address	[::1]:61738
Referrer Policy	no-referrer-when-downgrade
Response Headers	<ul style="list-style-type: none">Content-Length: 500Content-Type: application/json; charset=utf-8Date: Thu, 23 Jan 2020 12:33:22 GMTServer: Microsoft-IIS/10.0X-Powered-By: ASP.NET

Let's see different types of hosting models

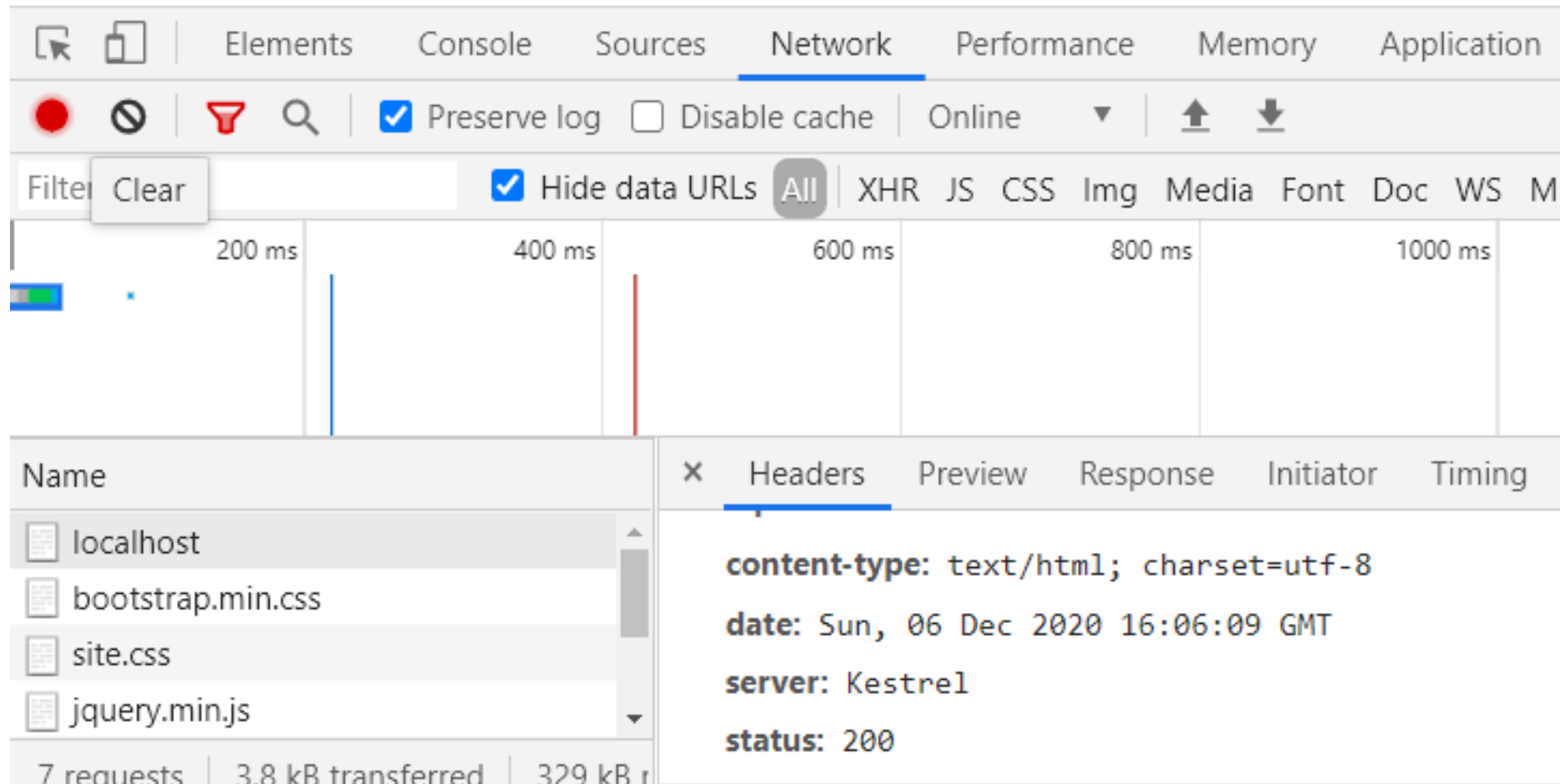
- Stop the app and open the command prompt and run the same application using dotnet CLI using the command dotnet run. Now it will host app on <http://localhost:5000>.



```
Administrator: Windows PowerShell - dotnet run
C:\> \DotNetCoreApp> dotnet run
info: microsoft.hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: 
```

Let's see different types of hosting models

- Browse the URL and open the network tab to see the server attribute as Kestrel.



The screenshot shows the Chrome DevTools Network tab. The top navigation bar includes 'Elements', 'Console', 'Sources', 'Network' (selected), 'Performance', 'Memory', and 'Application'. Below this, there are icons for a red dot, a no-interaction icon, a filter icon, and a search icon. A 'Preserve log' checkbox is checked, and 'Disable cache' is unchecked. The status is 'Online'. A filter bar shows 'Filter' and 'Clear' buttons, with 'Hide data URLs' checked and 'All' selected. Below the filter bar is a timeline with markers at 200 ms, 400 ms, 600 ms, 800 ms, and 1000 ms. The main area shows a list of requests: 'localhost', 'bootstrap.min.css', 'site.css', and 'jquery.min.js'. The 'localhost' request is selected, and its headers are displayed in the right pane. The headers are: 'content-type: text/html; charset=utf-8', 'date: Sun, 06 Dec 2020 16:06:09 GMT', 'server: Kestrel', and 'status: 200'. At the bottom, it shows '7 requests', '3.8 kB transferred', and '329 kB r'.

DEPLOY .NET CORE APPLICATION ON LINUX

- When Microsoft launched their .Net Core framework the key selling point was it is a cross-platform framework, which means that now we can host our .Net application not only on Windows but on Linux too.
- Let's see how we can deploy .Net core application on Linux.

Step 1 - Publish your .Net Core application

- First, create a .Net core application on VS; you can make an MVC project or Web API project and if you already have an existing project, then open it.
 1. Right Click on your project
 2. Click on publish
 3. Now create a new publish profile, and browse the folder where you want to publish your project dll
 4. Click on publish so it will create your dll in the folder

DEPLOY .NET CORE APPLICATION ON LINUX

Step 2 - Install required .Net Module on Linux

- Now we have our web application dll and now we need to host it on the Linux environment. First, we need to understand how the deployment works in Linux. .Net applications run on Kestrel servers and we run Apache or Nginx server in Linux environments, which acts as a proxy server and handles the traffic from outside the machine and redirects it to the Kestrel server so we will have Apache or Nginx server as the middle layer.
- Here we will use Apache as a proxy server.
- First, we need to install the .Net core module in our Linux environment. For that run the following commands

```
sudo apt-get update
```

```
sudo apt-get install apt-transport-https
```

```
sudo apt-get update
```

```
sudo apt-get install dotnet-sdk-3.1
```

```
sudo apt-get install dotnet-runtime-3.1
```

```
sudo apt-get install aspnetcore-runtime-3.1
```

DEPLOY .NET CORE APPLICATION ON LINUX

Step 3 - Install and configure Apache Server

- So now we have all the required .Net packages. I have installed an additional package so if you are running a different project it will help.
- Now install the Apache server,

```
sudo apt-get install apache2
```

```
sudo a2enmod proxy proxy_httpproxy_htmlproxy_wstunnel
```

```
sudo a2enmod rewrite
```
- Now we need to make a conf file to set up our proxy on Apache. Create the following file:

```
sudo nano /etc/apache2/conf-enabled/netcore.conf
```
- Now copy the following configuration in that file,

DEPLOY .NET CORE APPLICATION ON LINUX

Now copy the following configuration in that file,

```
<VirtualHost *:80>
```

```
    ServerName www.DOMAIN.COM
```

```
    ProxyPreserveHost On
```

```
    ProxyPass / http://127.0.0.1:5000/
```

```
    ProxyPassReverse / http://127.0.0.1:5000/
```

```
    RewriteEngine on
```

```
    RewriteCond %{HTTP:UPGRADE} ^WebSocket$ [NC]
```

```
    RewriteCond %{HTTP:CONNECTION} Upgrade$ [NC]
```

```
    RewriteRule /(.*) ws://127.0.0.1:5000/$1 [P]
```

```
    ErrorLog /var/log/apache2/netcore-error.log
```

```
    CustomLog /var/log/apache2/netcore-access.log common
```

```
</VirtualHost>
```


DEPLOY .NET CORE APPLICATION ON LINUX

<VirtualHost *:80>

This tag defines the IP and port it will bind Apache so we will access our application from outside our Linux environment through this Ip:Port.

Now restart the Apache server,

- `sudo service apache2 restart`
- `sudoapachectlconfigtest`

DEPLOY .NET CORE APPLICATION ON LINUX

Step 4 - Configure and Start Service

- Move your dll to the defined path with the below command.

```
"sudo cp -a ~/release/ /var/netcore/"
```
- Create a service file for our .Net application

```
"sudo nano /etc/systemd/system/ServiceFile.service"
```
- Copy the following configuration in that file and it will run our application,

DEPLOY .NET CORE APPLICATION ON LINUX

[Unit]

Description=ASP .NET Web Application

[Service]

WorkingDirectory=/var/netcore

ExecStart=/usr/bin/dotnet /var/netcore/Application.dll

Restart=always

RestartSec=10

SyslogIdentifier=netcore-demo

User=www-data

Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]

WantedBy=multi-user.target

DEPLOY .NET CORE APPLICATION ON LINUX

- `ExecStart=/usr/bin/dotnet /var/netcore/Application.dll` in this line replace `Application.dll` with your dll name that you want to run.
- Now start the service. Instead of the service name in the below commands use the name of the file made above,
 - `sudo systemctl enable {Service Name}`
 - `sudo systemctl start {Service Name}`
- Now your proxy server and kestrel server is running and you can access your application through any ip with port 80.
- To redeploy the code you need to replace the dll and stop and start your service again through the following commands
 - `sudo systemctl stop {Service Name}`
 - `sudo systemctl start {Service Name}`

ASP.NET CORE MODULE

- The ASP.NET Core Module is a native IIS module that plugs into the IIS pipeline to either:
 - Host an ASP.NET Core app inside of the IIS worker process (w3wp.exe), called the in-process hosting model.
 - Forward web requests to a backend ASP.NET Core app running the Kestrel server, called the out-of-process hosting model.
- **Supported Windows versions**
 - Windows 7 or later
 - Windows Server 2012 R2 or later
- When hosting in-process, the module uses an in-process server implementation for IIS, called IIS HTTP Server (IISHttpServer).
- When hosting out-of-process, the module only works with Kestrel. The module doesn't function with HTTP.sys.

ASP.NET CORE MODULE

In-process hosting model

- ASP.NET Core apps default to the in-process hosting model.
- The following characteristics apply when hosting in-process:
 1. IIS HTTP Server (IISHttpServer) is used instead of Kestrel server. For in-process, CreateDefaultBuilder calls UseIIS to:
 - Register the IISHttpServer.
 - Configure the port and base path the server should listen on when running behind the ASP.NET Core Module.
 - Configure the host to capture startup errors.

ASP.NET CORE MODULE

2. The `requestTimeout` attribute doesn't apply to in-process hosting.
3. Sharing an app pool among apps isn't supported. Use one app pool per app.
4. When using Web Deploy or manually placing an `app_offline.htm` file in the deployment, the app might not shut down immediately if there's an open connection. For example, a websocket connection may delay app shut down.
5. The architecture (bitness) of the app and installed runtime (x64 or x86) must match the architecture of the app pool.
6. Client disconnects are detected. The `HttpContext.RequestAborted` cancellation token is cancelled when the client disconnects.
7. In ASP.NET Core 2.2.1 or earlier, `GetCurrentDirectory` returns the worker directory of the process started by IIS rather than the app's directory (for example, `C:\Windows\System32\inetsrv` for `w3wp.exe`).

ASP.NET CORE MODULE

Out-of-process hosting model

- To configure an app for out-of-process hosting, set the value of the `<AspNetCoreHostingModel>` property to `OutOfProcess` in the project file (.csproj):

XMLCopy

```
<PropertyGroup>  
<AspNetCoreHostingModel>OutOfProcess</AspNetCoreHostingModel>  
</PropertyGroup>
```

- In-process hosting is set with `InProcess`, which is the default value.
- The value of `<AspNetCoreHostingModel>` is case insensitive, so `inprocess` and `outofprocess` are valid values.
- Kestrel server is used instead of IIS HTTP Server (`IISHttpServer`).
- For out-of-process, `CreateDefaultBuilder` calls `UseIISIntegration` to:
 - Configure the port and base path the server should listen on when running behind the ASP.NET Core Module.
 - Configure the host to capture startup errors.

DOCKER AND CONTAINERIZATION

- .NET Core can easily run in a Docker container. Containers provide a lightweight way to isolate your application from the rest of the host system, sharing just the kernel, and using resources given to your application. The Docker client has a CLI that you can use to manage images and containers.
- An image is an ordered collection of filesystem changes that form the basis of a container. The image doesn't have a state and is read-only. Much the time an image is based on another image, but with some customization. For example, when you create an new image for your application, you would base it on an existing image that already contains the .NET Core runtime.
- Because containers are created from images, images have a set of run parameters (such as a starting executable) that run when the container starts.
- Role requirements can also be expressed using the new Policy syntax, where a developer registers a policy at startup as part of the Authorization service configuration. This normally occurs in `ConfigureServices()` in your `Startup.cs` file.

DOCKER AND CONTAINERIZATION

Containers

- A container is a runnable instance of an image. As you build your image, you deploy your application and dependencies. Then, multiple containers can be instantiated, each isolated from one another. Each container instance has its own filesystem, memory, and network interface.

Registries

- Container registries are a collection of image repositories. You can base your images on a registry image. You can create containers directly from an image in a registry. The relationship between Docker containers, images, and registries is an important concept when architecting and building containerized applications or microservices. This approach greatly shortens the time between development and deployment.
- Docker has a public registry hosted at the Docker Hub that you can use. .NET Core related images are listed at the Docker Hub.

DOCKER AND CONTAINERIZATION

Dockerfile

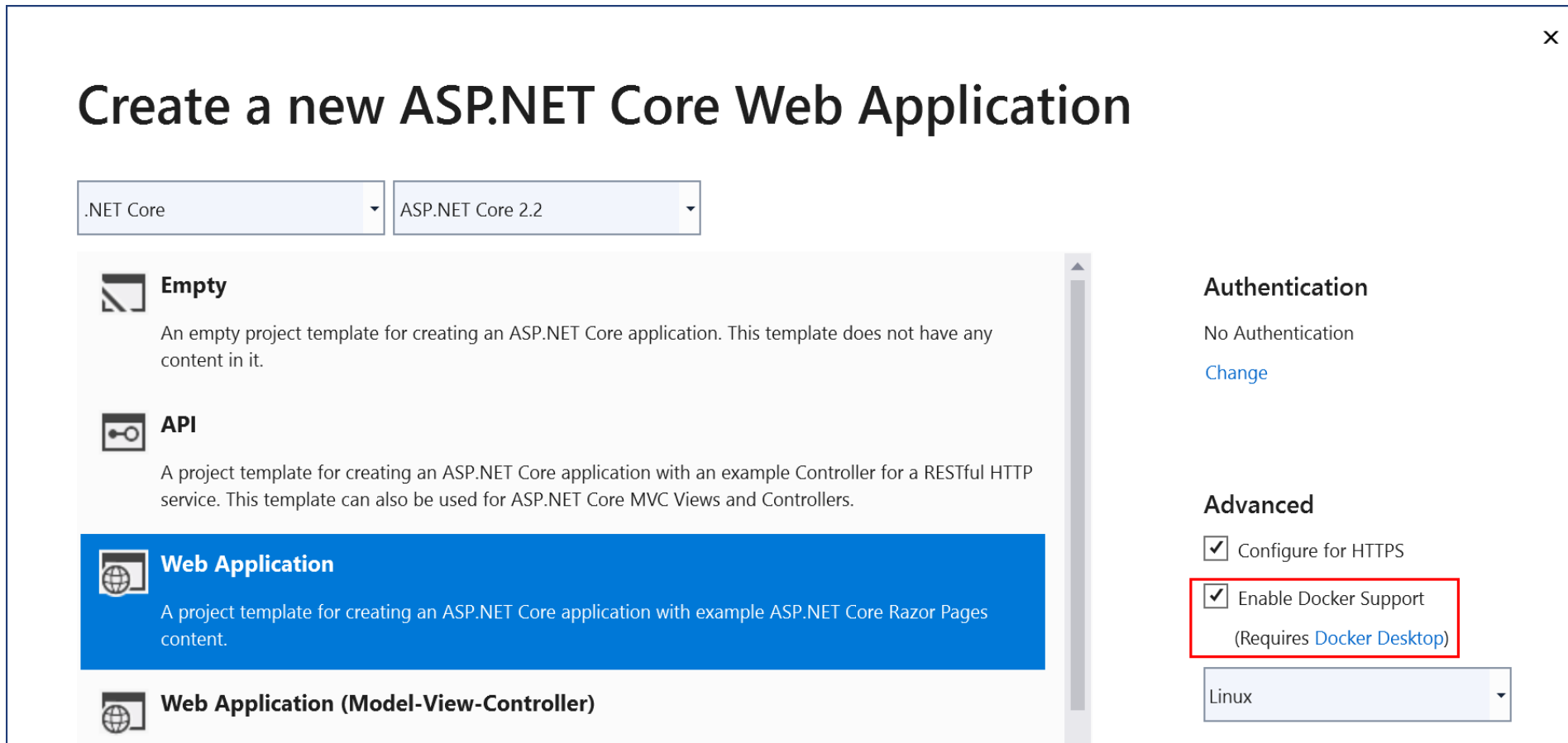
- A Dockerfile is a file that defines a set of instructions that creates an image. Each instruction in the Dockerfile creates a layer in the image. For the most part, when you rebuild the image, only the layers that have changed are rebuilt. The Dockerfile can be distributed to others and allows them to recreate a new image in the same manner you created it. While this allows you to distribute the instructions on how to create the image, the main way to distribute your image is to publish it to a registry.

Docker support in Visual Studio

- Docker support is available for ASP.NET projects, ASP.NET Core projects, and .NET Core and .NET Framework console projects.

Adding Docker support

- You can enable Docker support during project creation by selecting Enable Docker Support when creating a new project, as shown.



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)

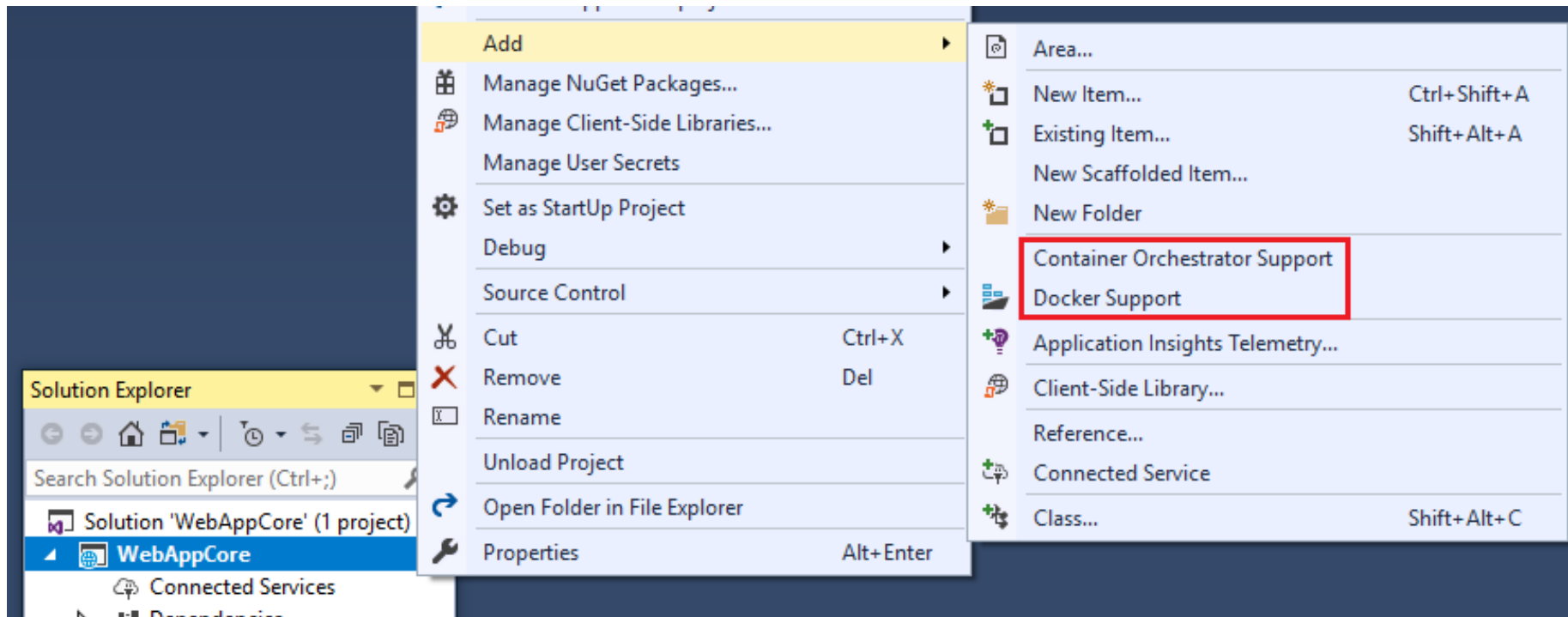
Authentication
No Authentication
[Change](#)

Advanced
 Configure for HTTPS
 Enable Docker Support
(Requires [Docker Desktop](#))

Linux

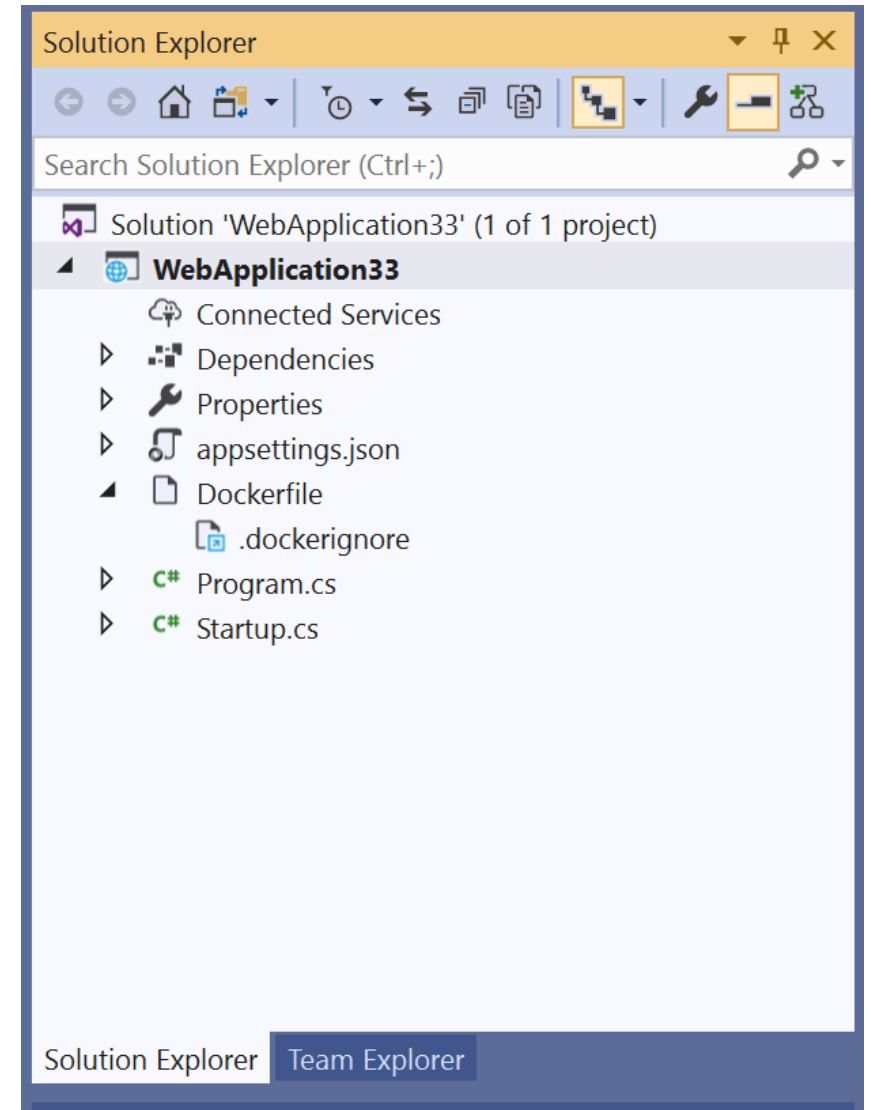
Adding Docker support

- You can add Docker support to an existing project by selecting Add > Docker Support in Solution Explorer. The Add > Docker Support and Add > Container Orchestrator Support commands are located on the right-click menu (or context menu) of the project node for an ASP.NET Core project in Solution Explorer, as shown.



Adding Docker support

- When you add or enable Docker support, Visual Studio adds the following to the project:
 - a Dockerfile file
 - a .dockerignore file
 - a NuGet package reference to the `Microsoft.VisualStudio.Azure.Containers.Tools.Targets`
- The solution looks like this once you add Docker support:



DEPLOY YOUR ASP.NET CORE APP TO AZURE

Publish to Azure App Service

- <https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-aspnet-core-ef-step-05?view=vs-2019>

Discussion Exercise

1. What is a web server? List down the of Web Server you can find to host ASP.NET Core Application.
2. Explain about the Hosting Models in ASP.NET Core
3. What is IIS? How can you deploy your ASP.NET Core Application on IIS Server?
4. Write down the details steps to host ASP.NET Application on Linux.
5. What is ASP.NET Core Module? Explain.
6. What is Docker and Containers.
7. How do you deploy ASP.NET Core app to Azure?