Table of Contents

Exercise Guide: Module 2 – Getting Started with Windows Containers

For all the demos involving Windows Containers, we recommend building all required images (even custom) before showing the exercise to have everything cached.

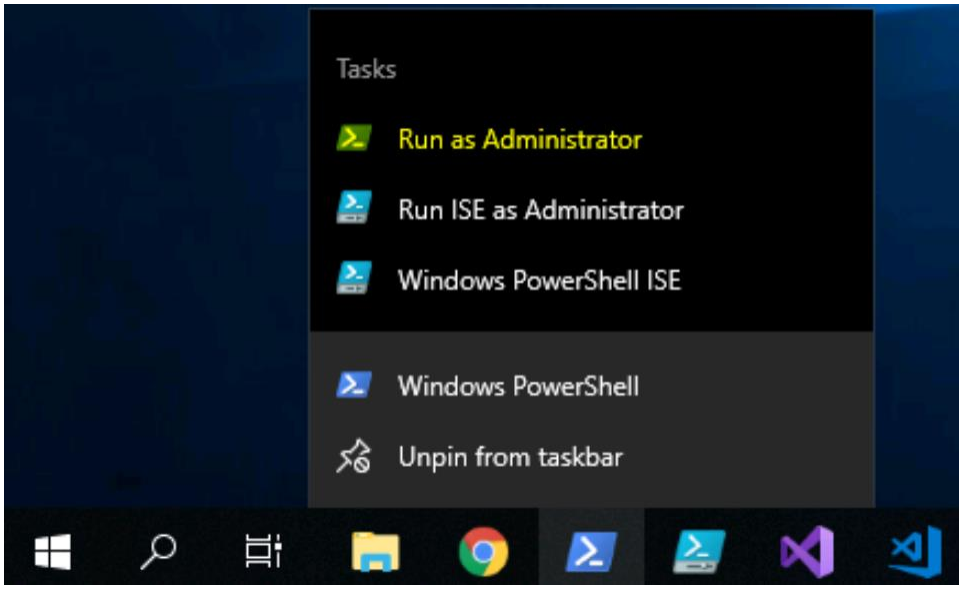
Exercise 1: Working with Nano Server & Server Core for Container Images

In this demo, you will provide walkthrough of Windows Nano and Server container images and running containers based on these images.

Tasks

1. **Run a Nano Server**

    1. You will need to run the commands in this section using the PowerShell console as an administrator. Right click the PowerShell icon on the taskbar and select "Run as Administrator".



    2. The PowerShell console is now available to you. Make sure you are inside the windows containers labs directory. You can do that by running the command "**`cd C:\labs\module2\`**". This will put you inside the windows containers lab folder where all the necessary files are located.

```
PS C:\Users\Administrator> cd C:\labs\module2\
PS C:\labs\module2> ls


    Directory: C:\labs\module2


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         5/10/2019   7:53 AM                aspnet4.7
d-----         5/10/2019   7:53 AM                aspnetcore
d-----         5/10/2019   7:53 AM                iis
-a----         5/10/2019   7:53 AM            196 commands.ps1
-a----         5/10/2019   7:53 AM            894 Lab 2 Commands Sheet.txt
```

3. First, let's get the list of all the container images available on this Docker host by running the command "**`docker images`**". Notice that you already have "windows/servercore" and "windows/nanoserver" images available to you representing "Server Core" and "Nano Server" images.

```
PS C:\labs\module2> docker images
REPOSITORY                                  TAG                                 IMAGE ID
mcr.microsoft.com/dotnet/core/sdk           2.2-nanoserver-1809                 6dd6c75ee1cb
mcr.microsoft.com/windows/servercore/iis    windowsservercore-ltsc2019          bb59d00aa26a
mcr.microsoft.com/dotnet/framework/sdk      4.7.2-windowsservercore-ltsc2019    6daad96eb1f2
mcr.microsoft.com/windows/nanoserver        1809                                e9bbec97e222
mcr.microsoft.com/windows/servercore        1809                                29a2c2cb7e4d
mcr.microsoft.com/dotnet/core/aspnet        2.2.3-nanoserver-1809               ef21b3f30c4b
mcr.microsoft.com/dotnet/framework/aspnet   4.7.2-windowsservercore-ltsc2019    4bc43f167268
```

NOTE: It's important to understand that you can always download specific version of "windows/servercore" and "windows/nanoserver" images by using an appropriate tag. It is also worth calling out that the images are pulled from Microsoft Container Registry although they are still discoverable from Docker Hub.

4. You will now run a container based on "Server Core" image (windows/servercore). Before you do that run the command "**`hostname`**". This will reveal the hostname of your virtual machine.

5. Run the command "**`docker run -it mcr.microsoft.com/windows/servercore:1809 powershell`**". Please be patient as it will take a minute or so for this command to work. The **-it** switch provides you with an interactive session. The **powershell** is a parameter passed as an argument which basically gives you access to Powershell (command line) running inside the container. Technically, the **-it** switch puts you inside a running container.

```
PS C:\labs\module2> docker run -it mcr.microsoft.com/windows/servercore:1809 powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```

6. Run the command "**`hostname`**". This time you are running it inside the running container. **Notice that the host name is different from the hostname you get in step 4.** The host name you see inside the container is host name of the container as is based on container id. You may want to run other commands as you wish.

7. Finally, exit the interactive session by typing "**`exit`**" and pressing Enter. This will take you back to the PowerShell console on the host.

8. Now let's run another container based on "Nano Server" image (windows/nanoserver). To do that run the command "**`docker run -it mcr.microsoft.com/windows/nanoserver:1903 CMD`**" (we use CMD because Powershell is not available in nanoserver).

9. Run the command "**`hostname`**". Notice that the host name is different from host name you get in step 4. The host name you see inside the container is the host name of the container which is based on container id. You can run other commands as you wish.

10. Finally, exit the interactive session by typing "**`exit`**" and pressing Enter. This will take you back to the PowerShell console on the host. In this task, you have created and run containers based on Windows Server Core & Nano Server container images that Microsoft provides.

Exercise 2: Building and Running IIS Server Container using Dockerfile

In the demo, you will show how to install IIS Web Server (Web Server Role) on a Windows Server Container and build a Container Image. Later, you will run a container with IIS running inside it. IIS Server is a popular Web Server released by Microsoft. Considering the strong footprint of IIS within the enterprises Microsoft supports IIS on Windows Container. IIS Server can be installed on windows server core.

Tasks

1. Build and Run IIS Server Image

1. Make sure you have a PowerShell Console open as an administrator (if you have followed previous task you should already be running a Console). Also, change the current directory to "**iis**" by running the command "cd" to get to "labs\module2\iis\"

2. The iis folder contains the Dockerfile with instructions to install IIS Server (Web Server Role) on the Windows Server Core base image. Open the Docker file by running the command "**`cat Dockerfile`**" and press enter.

3. Describe the structure of this file.

- The **FROM** instruction points to the **mcr.microsoft.com/windows/servercore** to be used as a base image for the new container image.
- The **RUN** instruction executes PowerShell to install Windows Feature "Web Server" (IIS Server).
- The next command is the **ADD** instruction which copies the **ServiceMonitor.exe** utility to the container image. The **ServiceMonitor.exe** is a utility that monitors **w3svc** service inside the container, if the service fails, the exe fails, so Docker knows the container is unhealthy. The **ServiceMonitor.exe** is developed and released by Microsoft: https://github.com/microsoft/iis-docker/tree/master/windowsservercore-ltsc2019
- The **EXPOSE** instructions opens the port 80 on the container. Without it, no traffic would be allowed by default.
- Finally, the **ENTRYPOINT** instruction makes sure that monitoring of **w3svc** begins immediately as soon as container starts running. This is what will keep the container in running state.

4. To build the new image with IIS installed on it run the command "**`docker build -t myiis:v1 .`**". This command builds a new container image with name "**myiis**" and tag "**v1**". That tag conveniently tells everyone about the information of version of the image. Please note that the STEP 3/6 of the build process performs the installation of Web-Server (IIS Server) and may take few minutes. Eventually you should see the results like following.

```
PS C:\labs\module2\iis> docker build -t myiis:v1 .
Sending build context to Docker daemon  138.8kB
Step 1/6 : FROM mcr.microsoft.com/windows/servercore:1809
 ---> 29a2c2cb7e4d
Step 2/6 : LABEL owner="cloudarchitectureteam@contoso.com"
 ---> Running in d61cf37b0259
Removing intermediate container d61cf37b0259
 ---> 43639b144ca5
Step 3/6 : RUN powershell -Command Add-WindowsFeature Web-Server
 ---> Running in 6a9a67b56d41

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {Common HTTP Features, Default Documen...


Removing intermediate container 6a9a67b56d41
 ---> 135a6bdc4c43
Step 4/6 : ADD ServiceMonitor.exe /ServiceMonitor.exe
 ---> 52269fd95364
Step 5/6 : EXPOSE 80
 ---> Running in 1ee5693b6f25
Removing intermediate container 1ee5693b6f25
 ---> db56ed82165f
Step 6/6 : ENTRYPOINT ["C:\\ServiceMonitor.exe", "w3svc"]
 ---> Running in b79521456a76
Removing intermediate container b79521456a76
 ---> 348962ba6bba
Successfully built 348962ba6bba
Successfully tagged myiis:v1
```

5. Now you can a run a new container based on "**myiis:v1"** image by using command (please type this one in instead of copy and pasting it, as it tends to have issues with copy pasting)
   docker run -d -p 80:80 myiis:v1

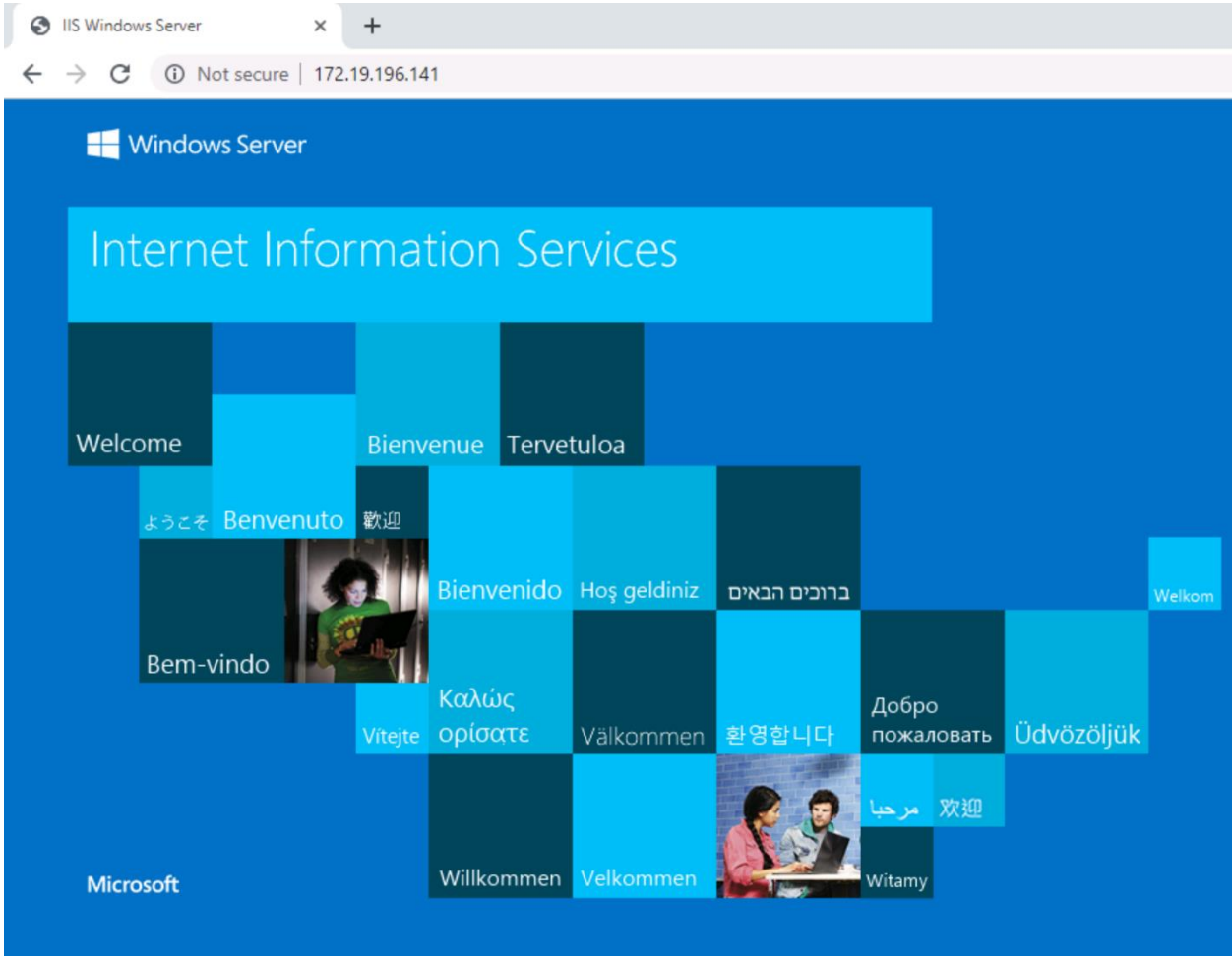
```
PS C:\labs\module2\iis> docker run -d -p 80:80 myiis:v1
d83e6af280f7ca66188e49facd9ec267c8b8e2a1e693768aa7f205dd7069490f
```

6. The container ID is shown after the run command ("154" in the above screenshot), or by using "docker ps". To get the IP address of the container, run the following command:

   docker inspect <container id> | FINDSTR "IPAddress"

```
PS C:\labs\module2\iis> docker inspect d83 | FINDSTR "IPAddress"
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.19.196.141",
```

7. Open any web browser of your choice and browse to the IP address from the previous step.



Note: ***In the Windows world, the loopback address might not work depending on the kernel version (due to a limitation in the default NAT network stack), only the external request routing would work.***

Exercise 3: Building and Running ASP.NET 4.7 Application in a Container Image

In this demo, you will provide walkthrough of how to package an existing ASP.NET 4.7 web application into a container. It's important to understand that Microsoft supports both the latest .NET frameworks like .NET Core, ASP.NET Core etc. and more legacy .NET Frameworks like .NET 3.5, .NET 4.5 and ASP.NET 4.5 on Windows Containers. Most customers today have heavy investments on some of the legacy Microsoft technologies, and Microsoft wants to

make sure that they can still move towards application containerization, not only for new applications but also for legacy applications.

NOTE: You can find more comprehensive list of application frameworks supported by Microsoft on Windows Containers at:
https://docs.microsoft.com/en-us/virtualization/windowscontainers/samples#Application-Frameworks

Tasks

1. Build and Run ASP.NET 4.7 MVC Application

1. Make sure you have a PowerShell Console open as an administrator (if you have followed previous task you should already be running a Console). Also, change the current directory to "aspnet4.7" by running the command "`cd`" to get to "`\labs\module2\aspnet4.7\`".

```
PS C:\labs\module2\aspnet4.7> ls


    Directory: C:\labs\module2\aspnet4.7


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         5/6/2019   3:15 PM                WebAppLegacy
-a----         5/6/2019   3:14 PM            434 Dockerfile
```

2. Before proceeding further let's stop and remove all the running containers from previous task. Run the command "**`docker rm -f (docker ps -aq)`**"

```
PS C:\labs\module2\aspnet4.7> docker rm -f (docker ps -aq)
d83e6af280f7
eb618b5b542a
ae998ed4771b
d8e8e689d89f
4bdbbde8a36c
```

3. Let's examine the Dockerfile. Open it in Notepad by running the command "**`cat Dockerfile`**". See the text below for an explanation of the commands in the DockerFile image.

```
PS C:\labs\module2\aspnet4.7> cat Dockerfile
FROM mcr.microsoft.com/dotnet/framework/sdk:4.7.2-windowsservercore-ltsc2019 AS build
WORKDIR /app

# copy csproj and restore as distinct layers
COPY *.sln .
COPY WebAppLegacy/*.csproj ./WebAppLegacy/
COPY WebAppLegacy/*.config ./WebAppLegacy/
RUN nuget restore

# copy everything else and build app
COPY WebAppLegacy/. ./WebAppLegacy/
WORKDIR /app/WebAppLegacy
RUN msbuild /p:Configuration=Release


FROM mcr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-ltsc2019 AS runtime
WORKDIR /inetpub/wwwroot
COPY --from=build /app/WebAppLegacy/. ./
```

The first noticeable statements are the two **FROM** which are used in what is called, a **multi-staged build** process. It allows us to create two Docker images with a single **`docker build`** command. This is a good time to introduce multi-staged build. Explain each part of the Dockerfile: https://docs.docker.com/develop/develop-images/multistage-build/). For more information about multi-staged build: https://www.youtube.com/watch?v=gdoXtFpXvik

- The first image is built on top of the .NET 4.7 SDK base image containing all utilities necessary to build your application: **mcr.microsoft.com/dotnet/framework/sdk:4.7.2-windowsservercore-ltsc2019**. This resulting image will be much bigger than what is required to simply run your application. This build stage will produce all application artifacts that will be picked up when building the second image. They will be copied in the folder **/app/WebAppLegacy** of the first image. Also note that this first image is identified as **build**
- The second image is built on top the .NET 4.7 runtime base image and only contains what is needed to run the application: **cr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-ltsc2019**. It uses the output from the first image to build its own runtime image **COPY --from=build /app/WebAppLegacy/. ./**. Having an image as small as possible is beneficial to reduce the attack surface (less tools equals less opportunities to exploit in an attack) and they will be much faster to download at deployment time.

4. Build a new image with web application packaged inside it by running the command **`docker build -t aspnetapp:v4.7 .`**

The build process to create a new container image will take few minutes. You will see the progress of all the steps as they are happening, so you will know the overall progress.

*Notice the tag ****v4.7**** that indicates the version number of ASP.NET framework. This use of tag is optional but recommended.*
*Note: At the end of the build process, feel free to look at the produced images with `**docker images**`. You will see the two images that have been built. the SDK image (that is not named) and the image that is actually going to be hosting our application: ****aspnetapp:v4.7****. Note that we can automatically remove the SDK image once the runtime image is built. For that, use the ****--rm**** parameter in the `docker build` command*

5. To run a container with the ASP.NET 4.5 web application based on new container image run the command: **docker run -d -p 80:80 aspnetapp:v4.7**

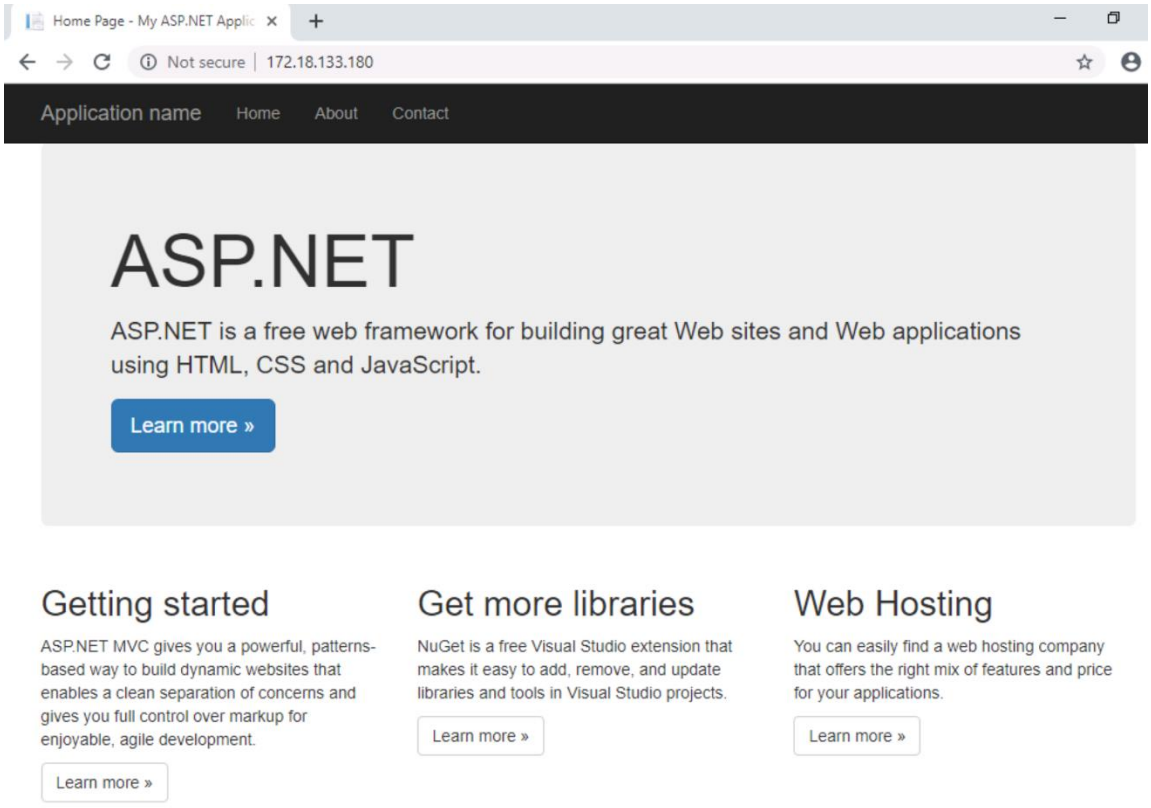
```
PS C:\labs\module2\aspnet4.7> docker run -d -p 80:80 aspnetapp:v4.7
5ddb0f0f9e9660604bce01cfa0a003b1ff2587417ac16f1ec8a67abedc5b37d4
```

6. Run the following to get the IP address (remember the container ID comes from the previous run step):

```
docker inspect <container id> | FINDSTR "IPAddress"
```

```
PS C:\labs\module2\aspnet4.7> docker inspect 5d | FINDSTR "IPAddress"
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.18.133.180",
```

7. Open web browser of your choice and browse to the IP address from the previous step. This will take you to the Home page of the ASP.NET 4.7 Web Application. It may take few seconds for the home page to load for the first time since IIS Application Pool takes a hit for serving the first page. Subsequent navigation to other pages will be much faster.



Exercise 4: Package ASP.NET Core Web Application as Container

In the previous exercise you built container images using some of the more mature technologies and products released by Microsoft. In this task you will build container that will run ASP.NET Core Web Application. ASP.NET Core is a significant step forward for Microsoft to allow ASP.NET to run cross platform including MacOS, Linux and Windows. ASP.NET sits on top of .NET Core so this cross-platform support will of course include .NET Framework.

NOTE: To understand when to use .NET Core and when to use .NET Framework please read article: https://docs.microsoft.com/en-us/dotnet/articles/standard/choosing-core-framework-server

In this task, you will package a simple ASP.NET Core MVC application into a container image using Dockerfile. Finally, you will run container hosting the ASP.NET Core application using the `docker run` command.

Tasks

1. **Build an ASP.NET Core Application**

    1. Change to the relevant directory by using “cd” to go to the path: module2\aspnetcore.

2. You are provided with a Dockerfile. View the content of Dockerfile by running a command "**`cat .\Dockerfile`**". The Dockerfile should look like below Unlike in module 1, the dotnet build/publish is within the Dockerfile itself:

```
PS C:\labs\module2\aspnetcore> cat .\Dockerfile
FROM mcr.microsoft.com/dotnet/core/sdk:2.2-nanoserver-1809 AS build-env
WORKDIR /app

# Copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2.3-nanoserver-1809
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "mywebapp.dll"]
```

3. You are all set to build ASP.NET Core web application container image. Run the command
   `docker build -t aspnetcoreapp:2.2 .`

4. Finally, to run the container by executing command **`docker run -d -p 8080:80 aspnetcoreapp:2.2`**
5. You are now running ASP.NET Core application inside the container listening at port 80 which is mapped to port 8080 on the host. Get the IP Address of the container you just ran:

   "**`docker inspect <container id> | FINDSTR "IPAddress"`**"

6. Navigate to the IP address from the previous step: