

Table of Contents

Lab: Module 2 – Getting Started with Windows Containers	2
Duration: 80 minutes	2
Exercise 1: Working with Nano Server & Windows Server Core Containers	2
Exercise 2: Building and Running an IIS Server Windows Container Image.....	8 6
Exercise 3: Building and Running ASP.NET 4.5 Application in a Container Image	12 11
Exercise 4: Building and Running SQL Server 2016 in a Container Image	16
Exercise 5: Making ASP.NET 4.5 Container Connection to SQL Server Container	22 24
Exercise 6: Building an ASP.NET Core Application	26 29

Lab: Module 2 – Getting Started with Windows Containers

Duration: 80 minutes

Exercise 1: Working with Nano Server & Windows Server Core Containers

In this exercise, you will learn about the Windows Nano and Server images. Please read below for an overview of each image. Then you will complete the steps to build and run these containers.

Windows Server Core Overview

Microsoft starting with Windows Server 2016 has an option of Server Core installation. The Server Core option reduces the amount of space required on disk, the potential attack surface, and especially the servicing requirements, its recommend that you choose the Server Core installation unless you have a need for the additional user interface elements and graphical management tools that are included in the Server with Desktop Experience option. For an even more lightweight option, see the next section on Nano Server. Server Core allows you to install various Server roles that may not be available in Nano Server including following. For a comprehensive list of features on Server Core please visit:

<https://technet.microsoft.com/en-us/windows-server-docs/get-started/getting-started-with-server-core>

- Active Directory Certificate Services
- Active Directory Domain Services
- DHCP Server
- DNS Server
- File Services (including File Server Resource Manager)
- Active Directory Lightweight Directory Services (AD LDS)
- Hyper-V
- Print and Document Services
- Streaming Media Services
- Web Server (including a subset of ASP.NET)
- Windows Server Update Server
- Active Directory Rights Management Server
- Routing and Remote Access Server and the following sub-roles:
 - Remote Desktop Services Connection Broker
 - Licensing
 - Virtualization
 - Volume Activation Services

Windows Nano Server Overview

Windows Server 2016 offers a new type of server ie “Nano Server”. Nano Server is a remotely administered server operating system which takes up far less disk space, sets up significantly faster, and requires far fewer updates and restarts than Windows Server Core. And most importantly when it restarts, it restarts much faster. Due to these characteristics Nano Server is a great fit for container workloads. However, it is important to understand that because Nano Server is optimized as a lightweight operating system for running “cloud-native” applications based on containers and micro-services there are important differences in Nano Server versus Server Core or Server with Desktop Experience installations including: (for comprehensive list of capability differences visit <https://technet.microsoft.com/windows-server-docs/get-started/getting-started-with-nano-server>)

- Nano Server is "headless;" there is no local logon capability or graphical user interface.
- Only 64-bit applications, tools, and agents are supported.
- Nano Server cannot serve as an Active Directory domain controller.
- Group Policy is not supported. However, you can use Desired State Configuration to apply settings at scale.
- Nano Server cannot be configured to use a proxy server to access the internet.
- Best Practices Analyzer (BPA) cmdlets and BPA integration with Server Manager are not supported.
- Nano Server does not support virtual host bus adapters (HBAs).
- Nano Server does not need to be activated with a product key. When functioning as a Hyper-V host, Nano Server does not support Automatic Virtual Machine Activation (AVMA). Virtual machines running on a Nano Server host can be activated using Key Management Service (KMS) with a generic volume license key or using Active Directory-based activation.
- The version of Windows PowerShell provided with Nano Server has important differences.

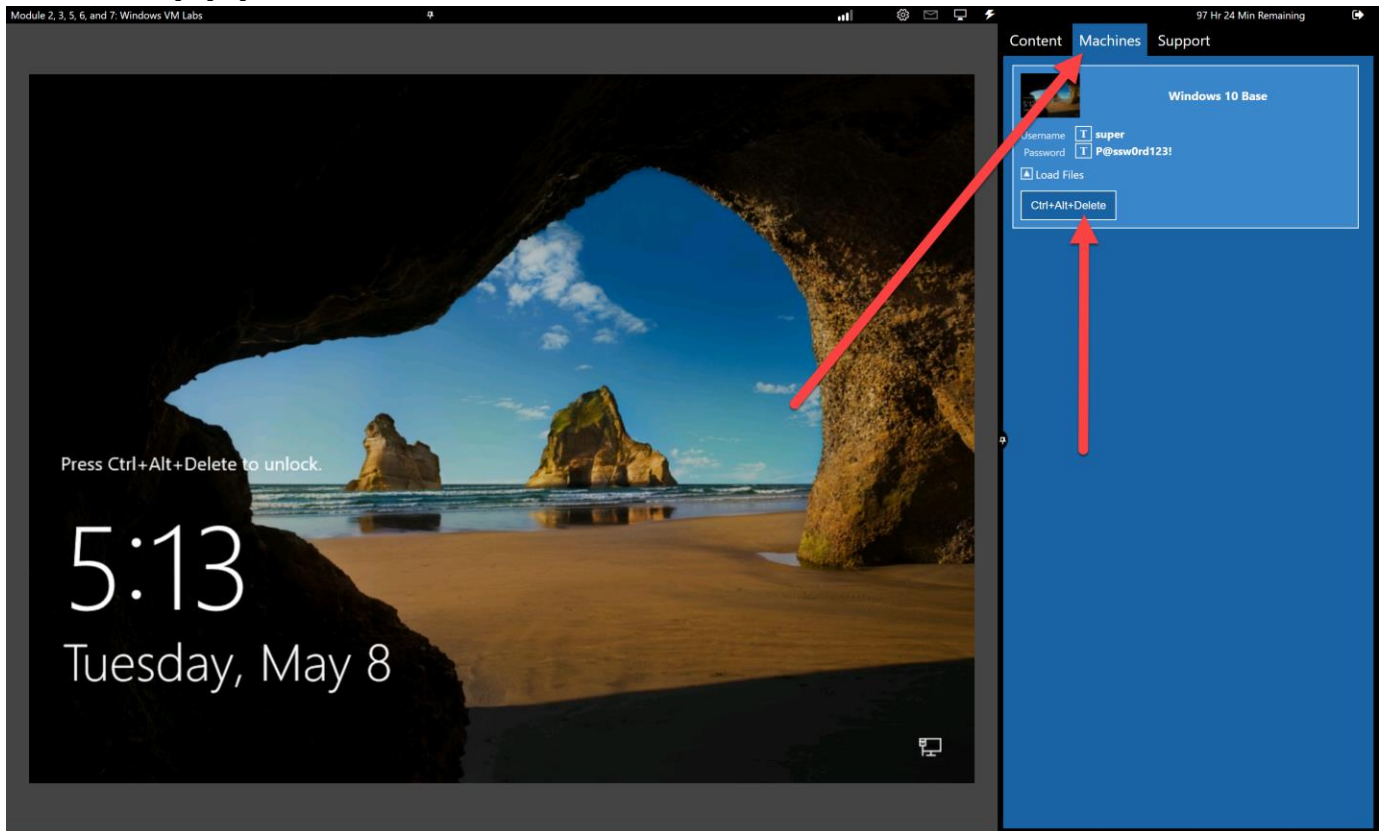
Microsoft offers both Nano Server and Server Core in the form of Docker images through the Docker Hub. Depending on the nature of application you like to build you may choose from among them. For example, SQL Server 2016 Express image will need Server Core as its base image, but a simple windows service may be able to run just fine on Nano server. In the following exercise, you will run “hello world” style container based on Nano and Server Core images.

Tasks

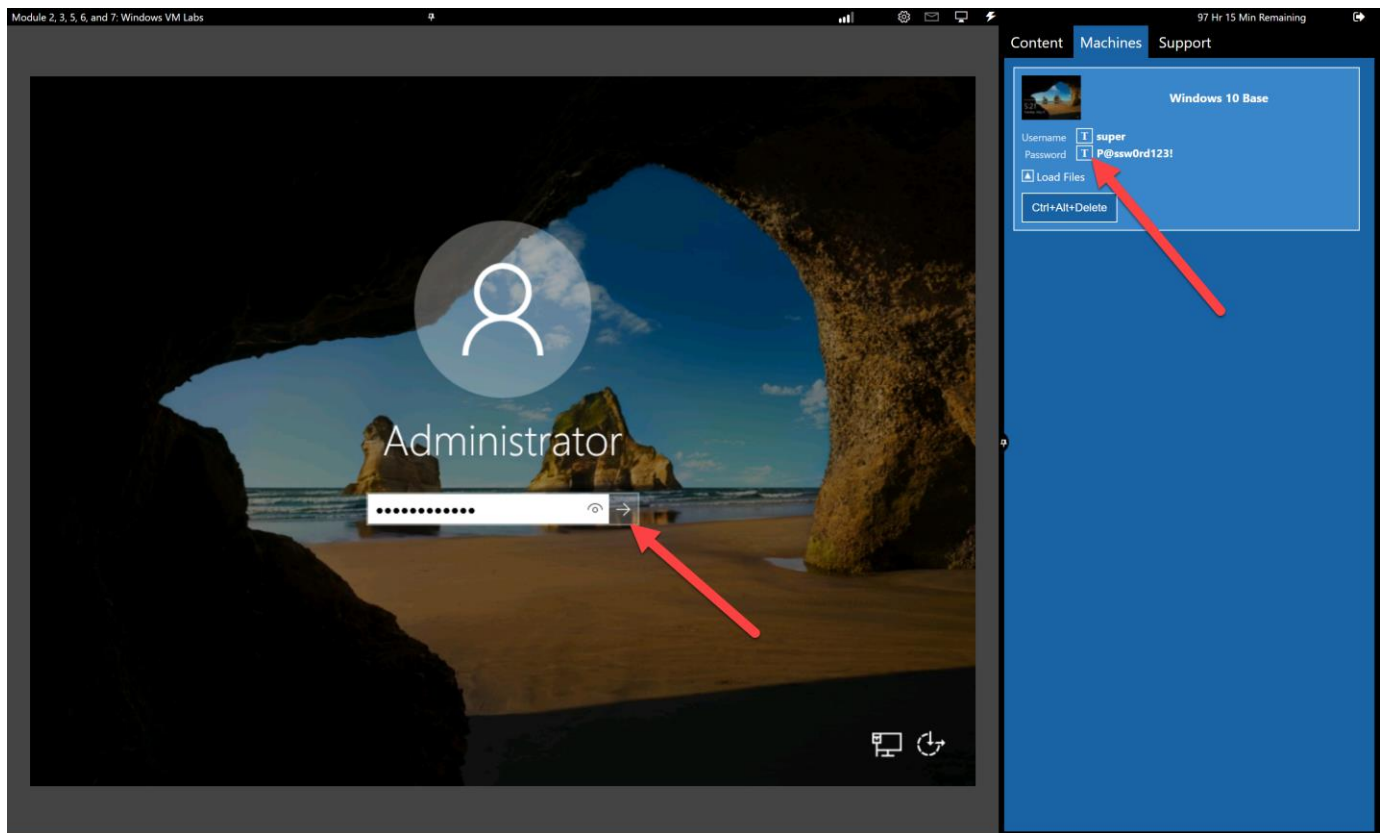
1. Run a Nano Server

1. Go into Labs on Demand (LOD) and start your Windows VM.

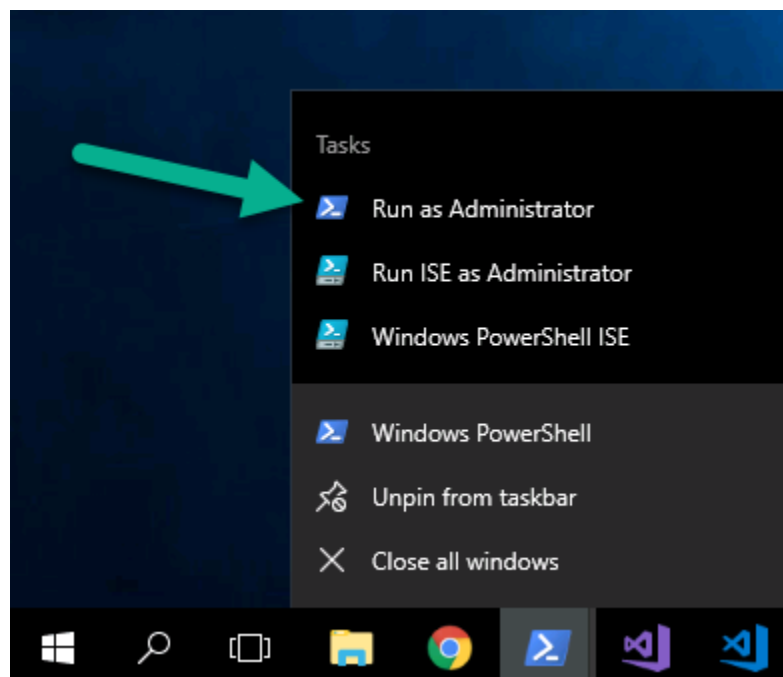
2. Switch lab windows to the Windows Server 2016 Lab Virtual Machine. On the LOD popup, click on the Machines tab, then click on the Ctrl+Alt+Delete button.



3. Click on the Password under the LOD Machine tab. Then click the arrow to login.



4. 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".



- 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.

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

Directory: C:\labs\module2

Mode                LastWriteTime         Length Name
----                -
d-----         4/20/2018   12:14 PM             aspnet4.5
d-----         4/29/2018    6:25 PM             aspnetcore
d-----         4/20/2018   12:14 PM              iis
d-----         4/20/2018   12:14 PM          sqlserver2016
-a-----         2/12/2018   12:02 PM           211 commands.ps1
-a-----         2/12/2018   12:02 PM       1670 Lab 2 Commands Sheet.txt
```

- 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 "microsoft/windowsservercore" and "microsoft/nanoserver" images available to you representing "Server Core" and "Nano Server" images.

```
PS C:\Users\Administrator> docker images
REPOSITORY              TAG                IMAGE ID            CREAT
microsoft/dotnet         nanoserver         0fc8122fafc6       8 days
microsoft/aspnetcore-build 2.0               eb21d939e0d8       2 weel
microsoft/aspnetcore     2.0               bb4964873e83       2 weel
microsoft/windowsservercore latest            ad6116672030       3 weel
microsoft/nanoserver     latest            353592ac9faa       3 weel
microsoft/mssql-server-windows-express latest            1986b8a8f950       3 mont
```

NOTE: It's important to understand that you can always download specific version of "microsoft/windowsservercore" and "microsoft/nanoserver" images by using an appropriate tag. For example, "docker pull microsoft/windowsservercore:10.0.14393.693" will pull the server core image that has a version number 10.0.14393.693. All the concepts you learn about docker (Linux) containers and images generally apply as-is to windows container too. The main deference is the fact that windows container requires windows operating system as a host while Linux container requires Linux operating system.

7. You will now run a container based on “Server Core” image (microsoft/windowsservercore). Before you do that run the command “hostname”. This will reveal the hostname of your virtual machine. Please note that your host machine name may be different.

```
PS C:\labs\module2> hostname  
WIN-EJI3ACSSVHL
```

8. Run the command `docker run -it microsoft/windowsservercore 'CMD'`. 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 ‘CMD’ is a parameter passed as an argument which basically gives you access to the CMD (command line) running inside the container. Technically, the “it” switch puts you inside a running container.

```
docker run -it microsoft/windowsservercore 'CMD'
```

9. 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 hostname 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 or checkout the filesystem.

```
Microsoft Windows [Version 10.0.14393]  
(c) 2016 Microsoft Corporation. All rights reserved.  
  
C:\>hostname  
6bbcefe609f6
```

10. Finally, exit the interactive session by typing “exit” and pressing Enter. This will take you back to the PowerShell console on the host.

```
C:\>exit
```

NOTE: Typically, PowerShell on the host has a blue background but as you go back and forth from container to host PowerShell the black background color will persist. You can safely ignore it or if you wish you may launch a new PowerShell Console. If you do that, make sure to change current directory location where windows container labs are located.

11. Now let’s run another container based on “Nano Server” image (microsoft/nanoserver). To do that run the command “`docker run -it microsoft/nanoserver 'CMD'`” (Again note it might take a minute to load)

```
docker run -it microsoft/nanoserver 'CMD'
```

12. 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.

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\>hostname
e2be2f8cb1dd
```

13. Finally, exit the interactive session by typing "exit" and pressing Enter. This will take you back to the PowerShell console on the host.
14. 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 an IIS Server Windows Container Image

In the exercise you will learn how to install IIS Web Server (Web Server Role) on a Windows Server Core base core image. 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 both windows server core or nano server. In this task you will install the IIS Server on windows server core container image.

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\"

```
PS C:\labs\module2> cd iis
PS C:\labs\module2\iis> ls

Directory: C:\labs\module2\iis

Mode                LastWriteTime         Length Name
----                -
-a----           2/12/2018  12:02 PM             222 Dockerfile
-a----           2/12/2018  12:02 PM          135864 ServiceMonitor.exe
```


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 "Notepad .\Dockerfile" and press enter.

```
PS C:\labs\module2\iis> Notepad .\Dockerfile
```

3. Notice that Notepad is opened with the content of Dockerfile.

The FROM instruction points to the microsoft/windowsservercore 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 copy 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>). Finally, the ENTRYPOINT instruction make sure that monitoring of w3svc begins immediately as soon as container starts running.

```
FROM microsoft/windowsservercore
```

```
MAINTAINER Razi Rais
```

```
RUN powershell -Command Add-WindowsFeature Web-Server
```

```
ADD ServiceMonitor.exe /ServiceMonitor.exe
```

```
EXPOSE 80
```

```
ENTRYPOINT ["C:\\ServiceMonitor.exe", "w3svc"]
```

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 microsoft/windowsservercore
---> ad6116672030
Step 2/6 : MAINTAINER Razi Rais
---> Running in c3c5f8325ee8
---> a80874f08c81
Removing intermediate container c3c5f8325ee8
Step 3/6 : RUN powershell -Command Add-WindowsFeature Web-Server
---> Running in c82bf3c93800
```

- Now you can run a new container based on “**myiis:v1**” image by using command:
“`docker run -d -p 80:80 myiis:v1`”.

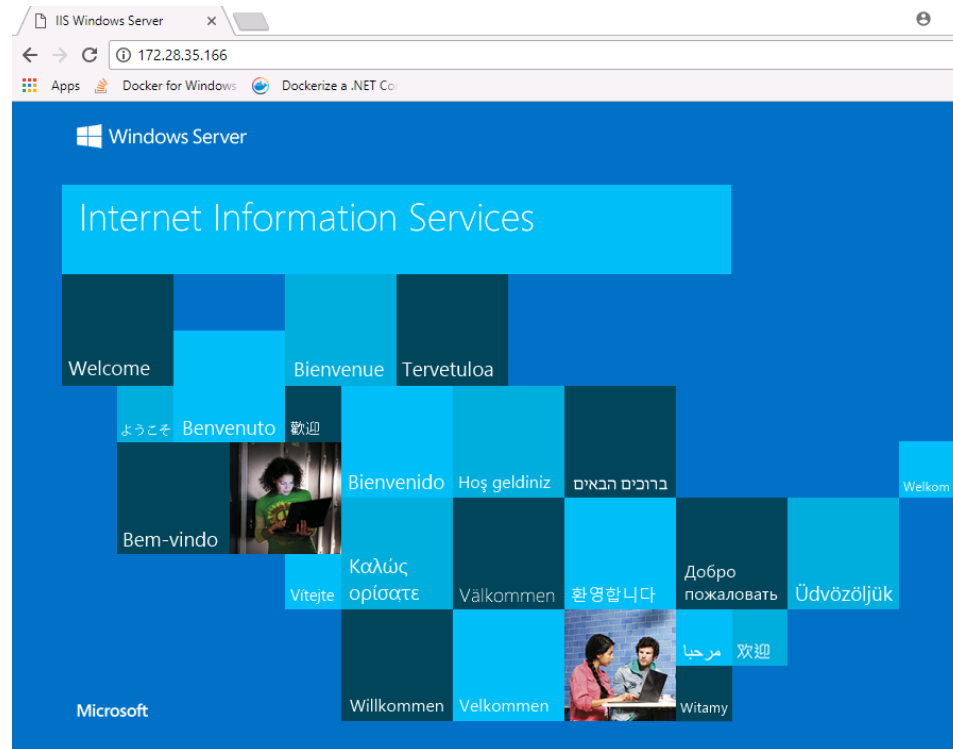
```
PS C:\Users\Administrator> docker run -d -p 80:80 myiis:v1
154edeacda86c12e57722a31d6c1c377b08cad68c378581484b971b7e415ab26
```

- 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 154 | FINDSTR "IPAddress"
"SecondaryIPAddresses": null,
"IPAddress": "",
"IPAddress": "172.28.35.166",
```

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



Note: In the Windows world the loopback side of this doesn't work right now, only the external request routing. If you browse to `http://localhost` on the container host, you won't see the site because of a limitation in the default NAT network stack. Container endpoints are only reachable from the Windows host using the container's IP and port. You can get the container's IP address with `docker inspect` command also:

`"docker inspect --format '{{ .NetworkSettings.Networks.nat.IPAddress }}' <<container-id>>"`

This concludes the exercise on creating a new image with IIS server on it. If you are looking to leverage IIS server beyond this lab, then you may want to use Microsoft official IIS server image (`microsoft/iis`) which is available at (<https://hub.docker.com/r/microsoft/iis/>). You will be using it later in this lab. The underlying process is pretty much same but main benefit of using official IIS image is that Microsoft release updated images on regular basis including patches and fixes.

Exercise 3: Building and Running ASP.NET 4.5 Application in a Container Image

In this task, you will learn how to package an existing ASP.NET 4.5 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 legacy Microsoft technologies, and Microsoft likes 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 4.5 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 "aspnet4.5" by running the command "cd" to get to "\labs\module2\aspnet4.5".

```
PS C:\labs\module2\iis> cd ..
PS C:\labs\module2> cd .\aspnet4.5\
PS C:\labs\module2\aspnet4.5>
```

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\aspnet4.5> docker stop (docker ps -aq) ; docker rm (docker ps -aq)
154edeacda86
b14847509244
154edeacda86
b14847509244
```

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

```
PS C:\labs\module2\aspnet4.5> Notepad .\Dockerfile
```

```

FROM microsoft/iis:latest
SHELL ["powershell"]

RUN Install-WindowsFeature NET-Framework-45-ASPNET ; \
    Install-WindowsFeature Web-Asp-Net45

COPY WebAppLegacy WebAppLegacy
RUN Remove-WebSite -Name 'Default Web Site'
RUN New-Website -Name 'guidgenerator' -Port 80 \
    -PhysicalPath 'c:\WebAppLegacy' -ApplicationPool '.NET v4.5'

EXPOSE 80

CMD ["ping", "-t", "localhost"]

```

You will notice that this Dockerfile is bit richer in instructions than the one we saw in the ~~ed~~ previous exercise. This is quite common particularly for “Brown Field Development” where you’re basically packaging an existing application into a container image. Remember, the best heuristic to follow when packaging legacy applications is to keep the container image follow “Single Responsibility Principle” which means single well-defined functionality per container image. You are following this principle as in this case we only going to run a single asp.net 4.5 web application running on the container and nothing else. Off course in other cases there may be exceptions to this and you may encounter legacy applications that are design in a certain fashion that means during first iteration to containerize them you have to package them as-is and then look into ways to re-design them for containers.

The first instruction in the Dockerfile asking docker to leverage the latest IIS base image that is released by Microsoft (microsoft/iis:latest) on Docker Hub. In this particular case, our web application does not have any specific dependencies on particular version of IIS but you should always point to specific version as needed.

Next is the SHELL instruction basically making PowerShell as default program when running instructions. The RUN commands installs .NET 4.5 and ASP.NET 4.5 which are available as Windows Feature. This is needed as the default IIS image does not have these two frameworks installed.

The COPY instruction simply copies the entire content of WebAppLegacy folder into the folder with the same name inside the image. A folder will be created inside the image as it won’t exist prior to this instruction.

The actual content that constitutes the ASP.NET Web Application resides inside the WebAppLegacy folder. You can examine them, but they are basically based on vanilla ASP.NET WebApplication template that ships with Visual Studio 2015.

The next RUN command removes the the default port 80 website on IIS using Remove-WebSite PowerShell Cmdlet. This serves two purposes: First we don't want anything extra running on the IIS to make sure all the resources allocated to our web application. Secondly, we also want to use default port 80 for the custom web application and not for any other web application.

The next instruction creates a new web application on IIS using New-WebSite PowerShell Cmdlet and then point to the folder where WebLegacyApp is copied by the earlier instruction.

The EXPOSE instruction exposes (or opens) the port 80 for incoming connections. You can expose multiple ports too if needed by listing them together in the same EXPOSE instruction and separate them using a space character e.g. EXPOSE 80 8080 5000.

Finally, the CMD instruction uses the Windows "ping" utility to probe the localhost. This instruction is needed at the time of this writing to keep the IIS process alive, but this may change in the future. We will keep it as-is for now.

4. Build a new image with web application packaged inside it by running the command "docker build -t aspnetapp:v4.5."

Notice that the tag "v4.5" indicates the version number of ASP.NET framework. This use of tag is optional but recommended.

```
PS C:\labs\module2\aspnet4.5> docker build -t aspnetapp:v4.5 .
Sending build context to Docker daemon 25.24MB
Step 1/8 : FROM microsoft/iis:latest
latest: Pulling from microsoft/iis
3889bb8d808b: Already exists
8f3369c32b2e: Already exists
894a0ffc7bb1: Downloading 5.848MB/131.6MB
f4ab7a8eb9ff: Download complete
d6641fe0169f: Download complete
```

Note: 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.

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:v4.5"

```
PS C:\labs\module2\aspnet4.5> docker run -d -p 80:80 aspnetapp:v4.5  
f0f43302b2714e5b02ba6096bbfdb53ac59797309a6c5623759faddf5ba0aa45
```

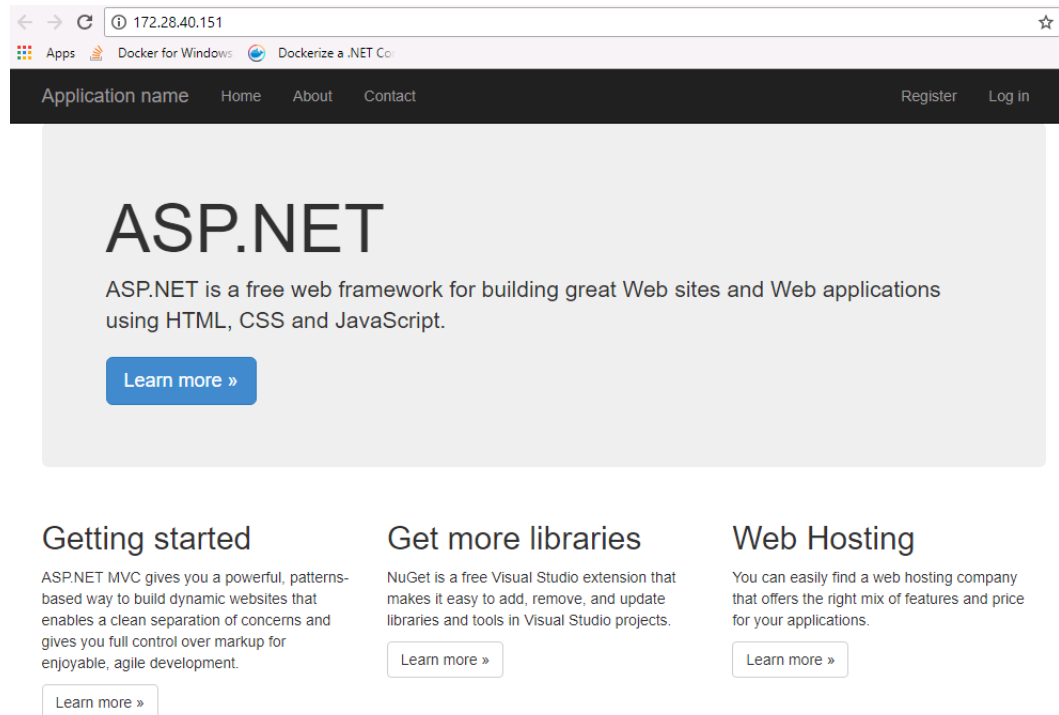
6. Run the following to get the IP address (remember the container ID comes from the previous run step):

docker inspect <container id> | FINDSTR "IPAddress"

```
PS C:\labs\module2\aspnet4.5> docker inspect f0f | FINDSTR "IPAddress"  
"SecondaryIPAddresses": null,  
"IPAddress": "",  
"IPAddress": "172.28.40.151",
```

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 4.5 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.

NOTE: The Register and Login links are not implemented as they require SQL Server Database on the backend. Working with SQL Server inside Containers is covered in the next task. You will also connect this web application to the backend in the same task.



Exercise 4: Building and Running SQL Server 2016 in a Container Image

Microsoft SQL Server is one of the most commonly use database server in the market today. Microsoft has made an investment to ensure that customers moving towards containers have an ability to leverage SQL Server through a container image. Currently Microsoft has released container images for Microsoft SQL Server 2016 for both Linux (<https://hub.docker.com/r/microsoft/mssql-server-linux>) and for Windows including Microsoft SQL Server Express Edition (<https://hub.docker.com/r/microsoft/mssql-server-windows-express>) and Microsoft SQL Server (180 days public preview of Enterprise Edition : <https://hub.docker.com/r/microsoft/mssql-server-windows>)

NOTE: Microsoft SQL Server 2016 as a product ships in various editions. For full comparison between several editions please visit: <https://docs.microsoft.com/en-us/sql/sql-server/editions-and-supported-features-for-sql-server-2016>

In this lab, you will work with Microsoft SQL Server 2016 Express container image to run a custom database that can store user related information. You will first learn how to associate relevant SQL Server database files to SQL Server container image. Then you will connect the Web Application that you package and run as a container in previous task to the database running inside the SQL Server container. Basically, you will end up with web application running in a container talking to a database in another container. This is very common scenario so understanding how it works is important. Let's start by running a SQL Server Express container with the custom database.

Tasks

1. SQL Server 2016 Container 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 PowerShell Console). Also, change the current directory to "labs\module2\sqlserver2016" by using the "cd" command.

```
PS C:\labs\module2\aspnet4.5> cd ..
PS C:\labs\module2> cd .\sqlserver2016\
PS C:\labs\module2\sqlserver2016>
```

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

```
PS C:\labs\module2\sqlserver2016> docker stop (docker ps -aq) ; docker rm (docker ps -aq)
f0f43302b271
f0f43302b271
```


- Let's examine the docker command needed to run the sql server express edition with the custom database. Open the notepad by running the command "Notepad .\command.txt".

```
PS C:\labs\module2\sqlserver2016> Notepad .\command.txt
```

- Notice the docker run command has various parameters. Following table provide description for parameters specific to SQL server.

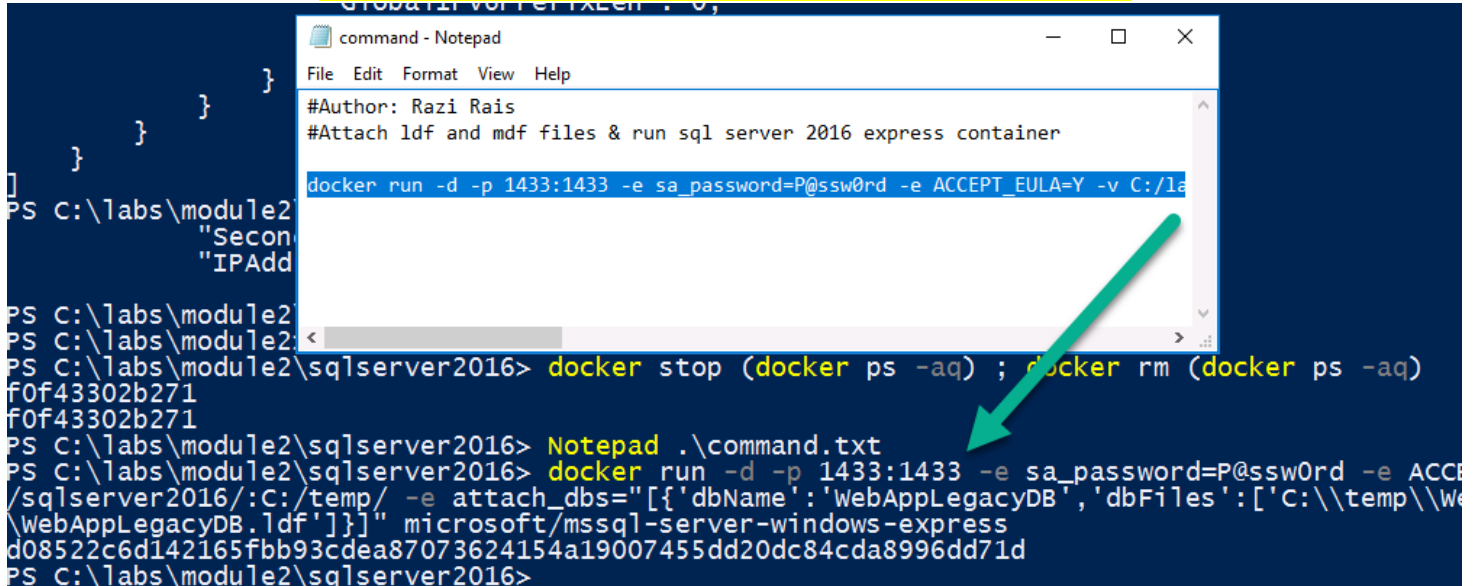
```
#Author: Razi Rais
#Attach ldf and mdf files & run sql server 2016 express container

docker run -d -p 1433:1433 -e sa_password=P@ssw0rd -e ACCEPT_EULA=Y -v
C:/labs/windows-containers/sqlserver2016/:C:/temp/ -e
attach_dbs="[{'dbName': 'WebAppLegacyDB', 'dbFiles': ['C:\\temp\\
WebAppLegacyDB.mdf', 'C:\\temp\\WebAppLegacyDB.ldf']}]" microsoft/mssql-server-
windows-express
```

Description

Parameter	Description
sa_password	The system administrator (userid = 'sa') password used to connect to SQL Server once the container is running. The password in this case is provided in plain text for brevity. However, best practice is to use secrets in Docker: https://docs.docker.com/engine/reference/commandline/secret
ACCEPT_EULA	Confirms acceptance of the end user licensing agreement found here .
attach_dbs	The configuration for attaching custom DBs (.mdf, .ldf files) in a JSON format. In this particular case the files are provided as part of lab and named "WebAppLegacyDB.ldf" and "WebAppLegacyDB.mdf"
-e	Flag that is used to pass environment variables to the container. In this particular case password, license eula and sql database files location are passed as environment variable.
-v	Flag that is used to share volumes across container(s). The basic idea of a volume is to keep files on the host system and make them available/share them inside the container. This is particularly helpful in this case as database files may become large over time so rather than bulking container with them you map them using volume and they will be available inside the container as if they are located locally inside a container. Another benefit of using volumes is that files on the host persist beyond the container lifecycle making them available for later usage. NOTE: Volumes are covered in depth in the Docker advance concepts lab. You can read more about volumes at: https://docs.docker.com/engine/reference/commandline/volume

- Execute the following command. Make sure to copy the command from the Notepad on your VM to avoid typing as that may introduce typos.



```

command - Notepad
File Edit Format View Help
#Author: Razi Rais
#Attach ldf and mdf files & run sql server 2016 express container

docker run -d -p 1433:1433 -e sa_password=P@ssw0rd -e ACCEPT_EULA=Y -v C:/1a

PS C:\labs\module2> docker stop (docker ps -aq) ; docker rm (docker ps -aq)
f0f43302b271
f0f43302b271
PS C:\labs\module2\sqlserver2016> Notepad .\command.txt
PS C:\labs\module2\sqlserver2016> docker run -d -p 1433:1433 -e sa_password=P@ssw0rd -e ACC
/sqlserver2016/:C:/temp/ -e attach_dbs="{ 'dbName': 'WebAppLegacyDB', 'dbFiles': [ 'C:\\temp\\We
\WebAppLegacyDB.ldf' ] }" microsoft/mssql-server-windows-express
d08522c6d142165fbb93cdea87073624154a19007455dd20dc84cda8996dd71d
PS C:\labs\module2\sqlserver2016>

```

- Let's get inside the running SQL Server container and examine the database. To do that you will need CONTAINER ID. Run the command "docker ps" and capture the CONTAINER ID.

```

PS C:\labs\module2\sqlserver2016> docker ps

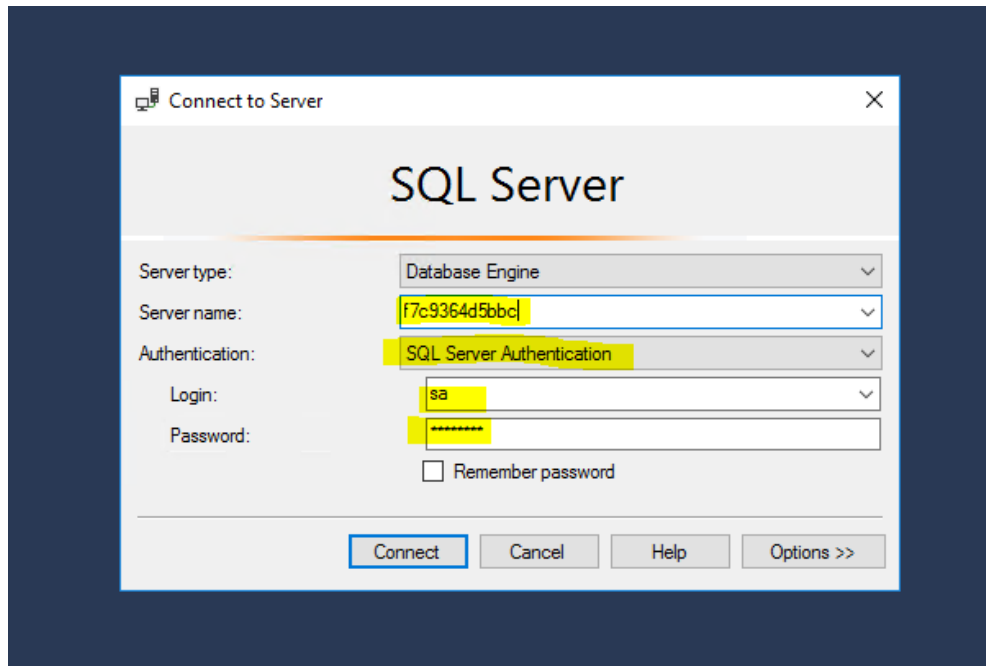
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATE
d08522c6d142	microsoft/mssql-server-windows-express		"powershell -Comma..."	2 minu

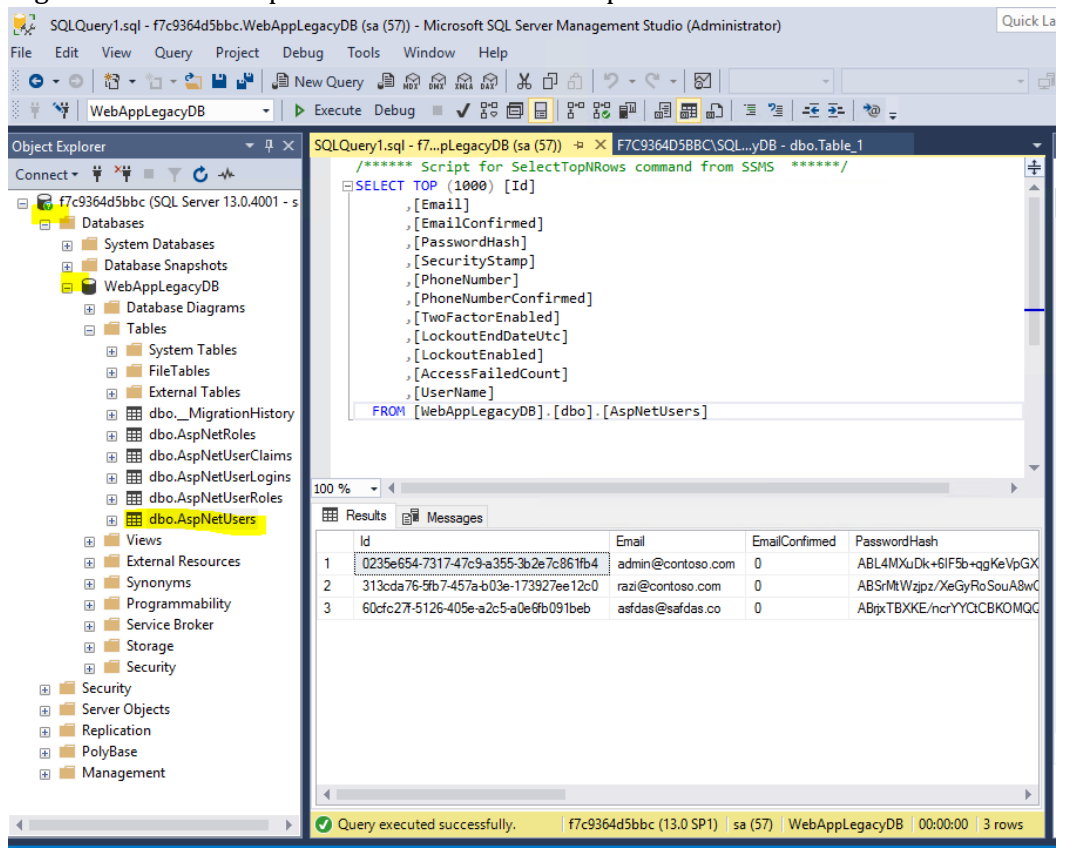
- Let's first look at the SQL Server using SQL Server Management Studio 2017 which has been pre-installed on your VM. It will be pinned to your taskbar.



- Make sure you keep Server type as Database Engine. Server name will be the container ID you got from running the "docker ps" command. Change Authentication to SQL Server Authentication. Login username will be "sa" and the password is "P@ssw0rd". See the following screenshot to verify your settings are the same.



9. Hit the plus sign next to Databases.
10. Hit the plus sign next to WebAppLegacyDB.
11. Hit the plus sign next to Tables.
12. Right click on dbo.AspNetUsers and hit Select top 1000 Rows.



13. Feel free to view any other tables.
14. Secondly, let's look at the SQL Server via command line interactively. Now run the command: " to launch an interactive session with container. Replace the CONTAINER ID with actual container id of running SQL server container. The sqlcmd is a basic command-line utility provided by Microsoft (<https://docs.microsoft.com/en-us/sql/relational-databases/scripting/sqlcmd-use-the-utility>) for ad hoc, interactive execution of Transact-SQL statements and scripts.

```
docker exec -it d08 sqlcmd -U sa
```

15. You will need to provide the password (same password you used in step 5: "P@ssw0rd") to connect to the SQL server. Once you enter the password you will have a sqlcmd utility prompt available to you for running Transact SQL statements.

```
Password:  
1> 
```

16. Let's begin by listing down all the databases available by running the command

```
SELECT name FROM master.dbo.sysdatabases  
GO
```

```
1> SELECT name FROM master.dbo.sysdatabases  
2> go  
name  
-----  
master  
tempdb  
model  
msdb  
WebAppLegacyDB
```

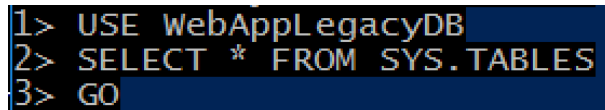
Notice that **“WebAppLegacyDB”** is listed at the very bottom. Please, ignore the background color which may get blended with PowerShell Console default blue background color.

17. Now, run the following commands one line at a time to display the list all the tables available inside the **“WebAppLegacyDB”** database.

```
USE WebAppLegacyDB
```

```
SELECT * FROM SYS.TABLES
```

```
GO
```



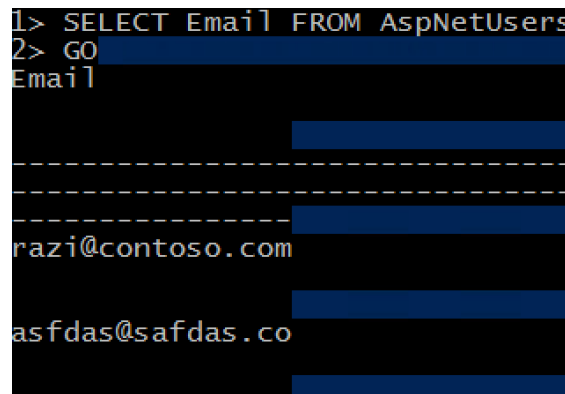
```
1> USE WebAppLegacyDB
2> SELECT * FROM SYS.TABLES
3> GO
```

18. Finally, list the Email of all the users inside the table **AspNetUsers** table by running the command:

```
SELECT Email FROM AspNetUsers
```

```
GO
```

..and then hit enter. Notice that there are a few entries in the table that already exist.



```
1> SELECT Email FROM AspNetUsers
2> GO
Email
-----
razi@contoso.com
asfdas@safdas.co
```

19. You can now exit the sqlcmd utility by running the command **“exit”**. This will end the interactive session with the container and you will be placed back inside the PowerShell Console on the host.

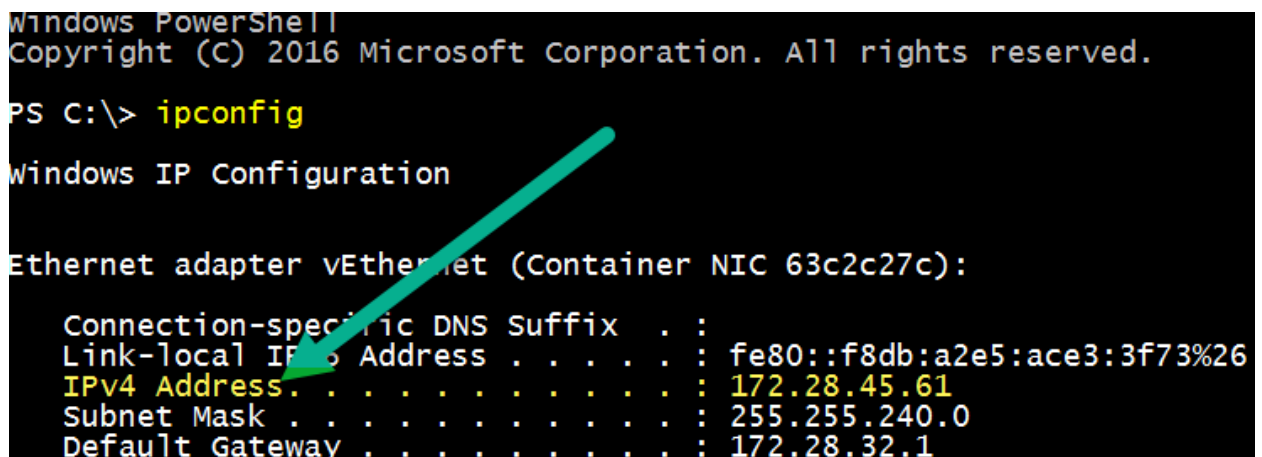
Exercise 5: Creating an ASP.NET 4.5 Container Connection to SQL Server Container

In this exercise, we will connect our ASP.NET project to our SQL Server and display that they can communicate to one another from separate containers.

Tasks

1. Connect the two containers.

1. First, you will need the IP address of the container running the SQL Server. To do that you will need to establish the interactive session with the container. Run the command “`docker exec CONTAINER-ID ipconfig`” which starts the interactive session with the container and give you access to the PowerShell console inside the container. You can refer to previous task which explains how to capture the CONTAINER ID. To gather the IP address, note down the **IPV4 address** (e.g. 172.20.166.217) as you will need it later.



```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

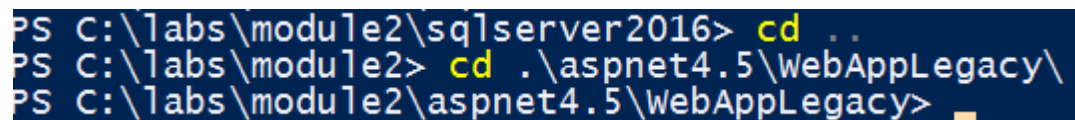
PS C:\> ipconfig

Windows IP Configuration

Ethernet adapter vEthernet (Container NIC 63c2c27c):

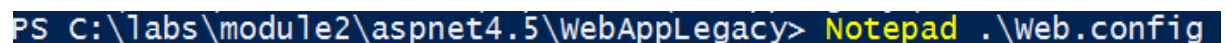
    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::f8db:a2e5:ace3:3f73%26
    IPv4 Address. . . . . : 172.28.45.61
    Subnet Mask . . . . . : 255.255.240.0
    Default Gateway . . . . . : 172.28.32.1
```

2. You will now update the connection string inside the **web.config** file for the **WebAppLegacy** web application. This is the same web application you worked with in previous task. To do that first change the current folder location to “**WebAppLegacy**” folder by running the command “`cd C:\labs\module2\aspnet4.5\WebAppLegacy\`”.



```
PS C:\labs\module2\sqlserver2016> cd ..
PS C:\labs\module2> cd .\aspnet4.5\WebAppLegacy\
PS C:\labs\module2\aspnet4.5\WebAppLegacy> _
```

3. Open the **web.config** file to edit the connection string in it by typing the command “`Notepad .\web.config`”



```
PS C:\labs\module2\aspnet4.5\WebAppLegacy> Notepad .\web.config
```

4. Locate the **<connectionStrings>** section inside the **web.config**. Then, change the value of **"Data Source=0.0.0.0"** to **"Data Source = CONTAINER IP ADDRESS"**. Where CONTAINER IP ADDRESS is the IP Address you capture in the previous task (e.g. 172.28.45.61)

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Password=P@ssw0rd;Persist Security
Info=True;User ID=sa;Initial Catalog=WebAppLegacyDB;Data Source=172.28.45.61"
providerName="System.Data.SqlClient" />
</connectionStrings>
<appSettings>
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
</appSettings>
```

Make sure you save the **web.config** with the updated Data Source value. Finally, close the notepad.

NOTE: You may be wondering the way we updated the values inside web.config file may not be the ideal way as it seems error prone and hard to scale. There is a better way and that is to use docker compose. Basically, docker compose allows you to declare dependencies between the containers which exactly what we need in this case. The web application is dependent on the database and specially needs data source values which can be server name of the database container. You will learn and explore docker compose in the Docker advanced concepts lab. Also, sometimes for ad-hoc testing it may make sense to use the approach used in this section.

5. You are now ready to create a new container image for the WebLegacyApp web application with updated web.config file. First, let's change the current directory to **"aspnet45"** where the Dockerfile is located by running the command **"cd .."**

```
PS C:\labs\module2\aspnet4.5\WebAppLegacy> cd ..
PS C:\labs\module2\aspnet4.5>
```

6. Now, build the container image for the WebLegacyApp by running the command **"docker build -t aspnetapp:v4.5-with-db ."**

```

PS C:\labs\module2\aspnet4.5> docker build -t aspnetapp:v4.5-with-db .
Sending build context to Docker daemon 25.24MB
Step 1/8 : FROM microsoft/iis:latest
--> b8f924611ebb
Step 2/8 : SHELL powershell
--> Using cache
--> dcadf2b4e2e2
Step 3/8 : RUN Install-WindowsFeature NET-Framework-45-ASPNET ; Ins
--> Using cache
--> b202838d09fb
Step 4/8 : COPY WebAppLegacy WebAppLegacy
--> da215efba9c2
Removing intermediate container 39e95b1e5ed9
Step 5/8 : RUN Remove-WebSite -Name 'Default Web Site'
--> Running in 1feeb05a4fa9
--> 66522b9b316c
Removing intermediate container 1feeb05a4fa9
Step 6/8 : RUN New-Website -Name 'guidgenerator' -Port 80 -Physical
5'
--> Running in 382855e146f0

Name                ID          State      Physical Path      Binding
----                --          -
guidgenerator       1450222056 Started      c:\WebAppLegacy    http *:
--> 36124030836b
Removing intermediate container 382855e146f0
Step 7/8 : EXPOSE 80
--> Running in 4f30473fe426
--> d0a73430b992

```

Notice that this time the build process is much faster compare to the last time when you build the container image for the same web application for the first time. This is because Docker already has most of the layers that goes inside the container image present in the cache (labelled as --> using cache in the output). Layers are covered in depth in the Docker advanced concepts lab.

7. You are now ready to run the container with the web application. Run the command "docker run -d -p 80:80 aspnetapp:v4.5-with-db"

```

PS C:\labs\module2\aspnet4.5> docker run -d -p 80:80 aspnetapp:v4.5-with-db
89f20498f7981c4b80372c5a3471dd7afdbce9f097246220e23b86881a907601

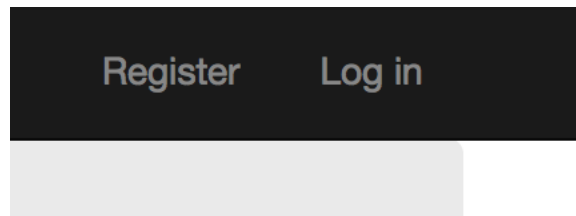
```

8. Get the IP Address of the container you just ran:
docker inspect <container id> | FINDSTR "IPAddress"


```
PS C:\labs\module2\aspnet4.5> docker inspect 89f | FINDSTR "IPAddress"
"SecondaryIPAddresses": null,
"IPAddress": "",
"IPAddress": "172.28.43.34",
```

9. To test that connection to database open web browser of your choice and browse to then IP address from the previous step. This will take you to the Home page the 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.

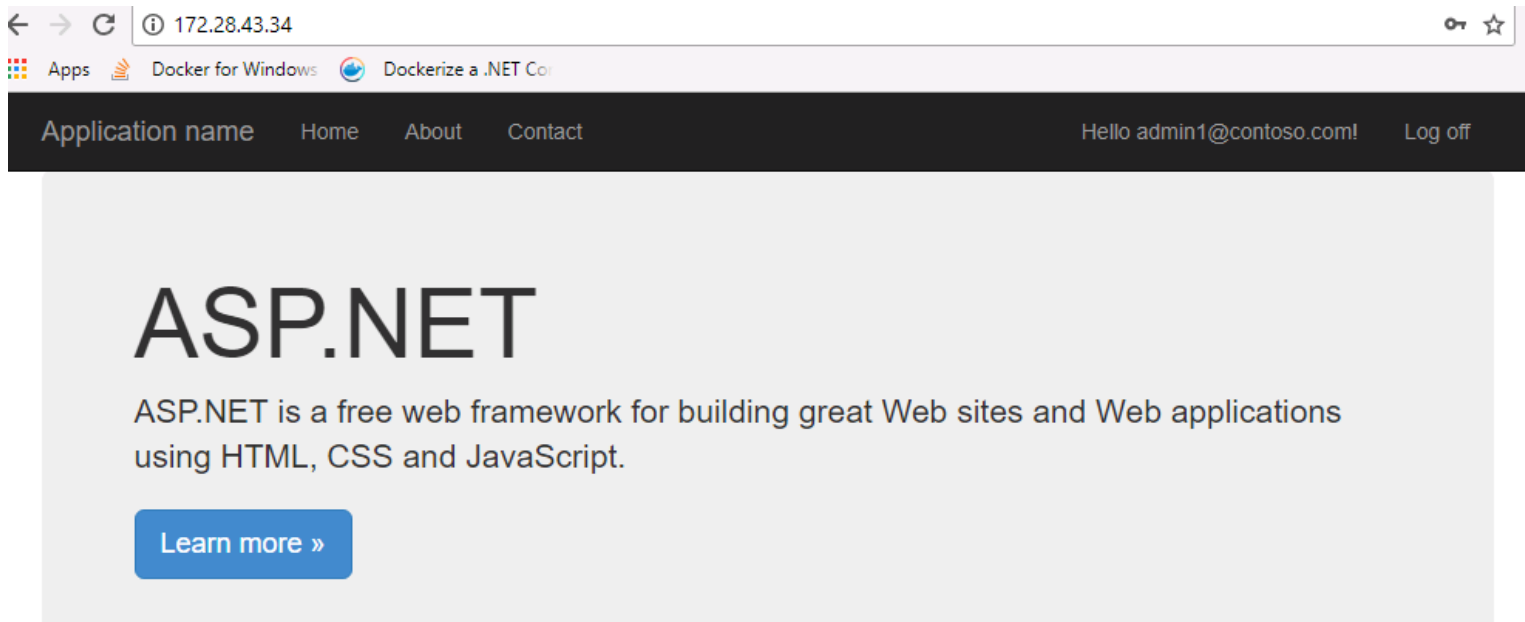
On the home page click on the Register link located of the top right.



On the registration page enter new user Email (e.g. admin1@contoso.com) and strong password with a capital letter, number, and a special character. Once done click on the Register button.

Email	<input type="text" value="admin1@contoso.com"/>
Password	<input type="password" value="....."/>
Confirm password	<input type="password" value="....."/>
	<input type="button" value="Register"/>

This will trigger the connection to database to create a new user. You will also be Logged in with the new user as shown below. Notice the welcome admin1@contoso.com message on the top right instead of a Register link.



As an additional test you can establish the interactive session with the container (using instructions from previous task) and run sqlcmd utility to query the Email of all users inside the **AspNetUsers** table. Notice the new user with Email admin1@contoso.com is now available inside the database.

```
1> USE WebAppLegacyDB
2> go
Changed database context to 'WebAppLegacyDB'.
1> SELECT EMAIL FROM ASPNETUSERS
2> GO
EMAIL
-----
-----
-----
```

Exercise 6: Building an ASP.NET Core Application

In the previous task 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. If you completed the Module 1 lab, then this is very similar however, now we will now build the Core application on Windows instead of Linux.

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 is of course included in the .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 first install the .NET Core Framework on the Windows Server 2016 for Container virtual machine. You will then package a simple ASP.NET Core MVC application into container image using Dockerfile. Finally, you will run container hosting the ASP.NET Core application using docker run command.

In this exercise, you will learn to add an ASP.NET Core project into a Container on Windows.

Tasks

1. Building and Running ASP.NET Core 2.x Application Inside Container

1. In this exercise, you will build ASP.NET Core 2.x application and then package and run it as a container.
2. Change to the relevant directory by using “cd” to go to the path: module2\aspnetcore.

```
PS C:\labs\module2\aspnet4.5> cd ..
PS C:\labs\module2> cd .\aspnetcore\
PS C:\labs\module2\aspnetcore>
```

3. You are provided with a Dockerfile. View the content of Dockerfile by running a command “Notepad .\Dockerfile”. The Dockerfile should look like below (note this is a multi-stage Dockerfile and will be explained in further detail in the next module’s presentation). The reason the .csproj is copied first is that if there are no changes to the project dependencies, it is easy to use the same cached layers during the Docker build step.

```
FROM microsoft/aspnetcore-build:2.0 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 microsoft/aspnetcore:2.0
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "mywebapp.dll"]
```

4. To create the container image run the command “docker build -t aspnetcoreapp:2.0 .”
Notice the use of tag “2.0” that signifies the use of framework dotnet core.

```
PS C:\labs\module2\aspnetcore> docker build -t aspnetcoreapp:2.0 .
Sending build context to Docker daemon 8.646MB
Step 1/10 : FROM microsoft/aspnetcore-build:2.0 AS build-env
--> eb21d939e0d8
Step 2/10 : WORKDIR /app
--> 42dec1e50eb1
Removing intermediate container 0b5fc9290e61
Step 3/10 : COPY *.csproj ./
--> 758af69b7232
Removing intermediate container 4055f48080cb
Step 4/10 : RUN dotnet restore
--> Running in 5bbc77a13762
Restoring packages for C:\app\mywebapp.csproj...
Installing Microsoft.AspNetCore.All 2.0.6.
Generating MSBuild file C:\app\obj\mywebapp.csproj.nuget.g.props.
```

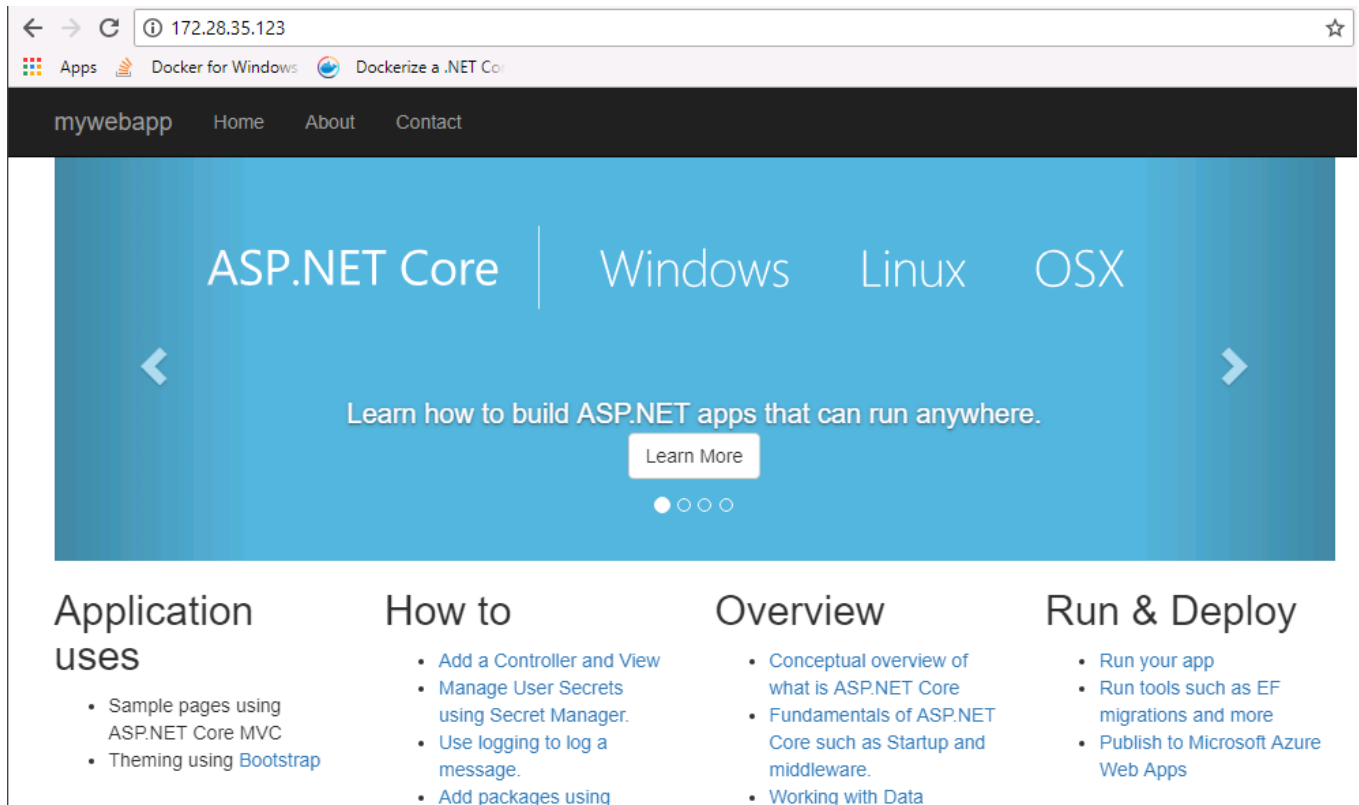
5. Launch the container running the app inside it by running the command
“docker run -d -p 8080:80 aspnetcoreapp:2.0”.

```
PS C:\labs\module2\aspnetcore> docker run -d -p 8080:80 aspnetcoreapp:2.0
aa6f82318f613c6539e0bb901b3d9c70f75d0eaae16fa6691404495d6a8b0a78
```

6. 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"
PS C:\labs\module2\aspnetcore> docker inspect aa6 | FINDSTR "IPAddress"
"SecondaryIPAddresses": null,
"IPAddress": "",
"IPAddress": "172.28.35.123",
```

7. To see the ASP.NET Core web application in action open any web browser of your choice and browse to your IP address. This will take you to the Home page the Web Application.



8. This concludes the task on building and running ASP.NET Core web application inside a container.
9. Run the following command to stop and remove all running containers.
"docker stop (docker ps -aq) ; docker rm (docker ps -aq)"