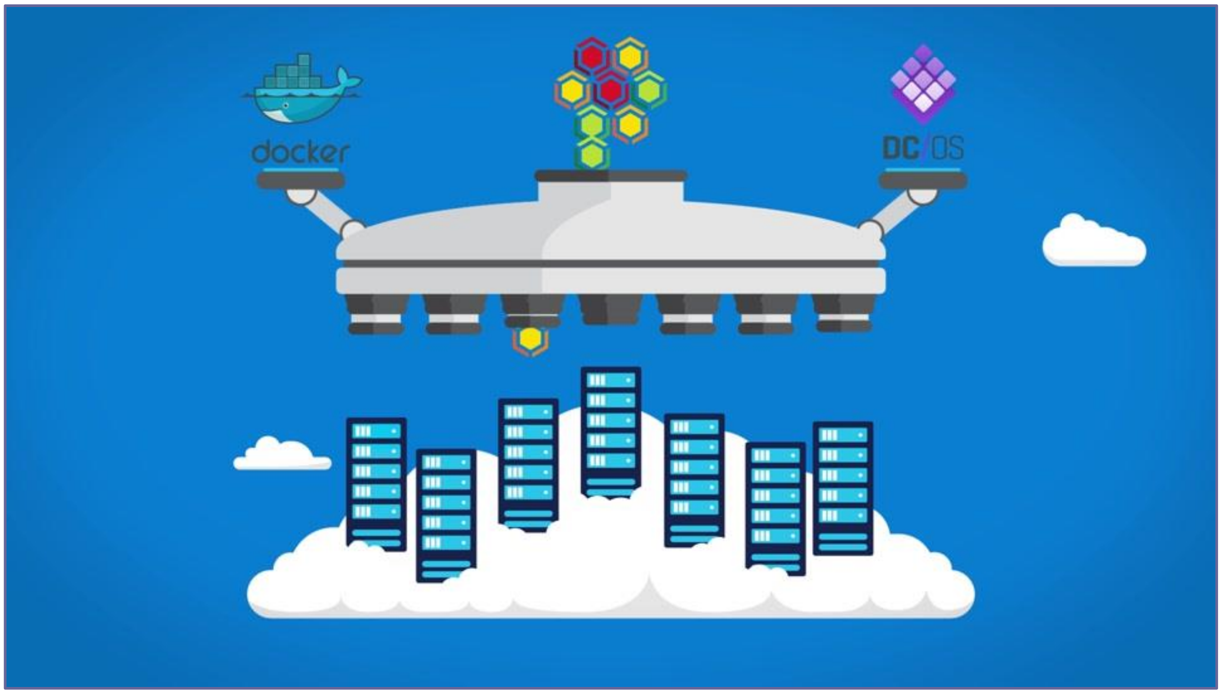


Azure Developer Series

Migrating a legacy ASP.NET 2-tier application to Azure using Container Services

Hands-On-Labs step-by-step guides

Prepared by:

Peter De Tender

CEO and Lead Technical Trainer
PDTIT and 007FFFLearning.com

@pdtit @007FFFLearning

Version: October 2018 – 2.0

Contents

Migrating a legacy ASP.NET 2-tiered application to Azure using Container Services - Hands-On-Labs step-by-step

Abstract and Learning Objectives

This workshop enables anyone to learn, understand and build a Proof of Concept, in performing a multi-tiered legacy ASP.NET web application using Microsoft SQL Server database, platform migration to Azure public cloud, leveraging on different Azure Platform Azure A Service (PaaS) and Azure Container Services.

After an introductory module on cloud app migration strategies and patterns, students get introduced to the basics of automating Azure resources deployments using Visual Studio and Azure Resource Manager (ARM) templates. Next, attendees will learn about Microsoft SQL database migration to SQL Azure PaaS, as well as deploying and migrating Azure Web Apps.

After these foundational platform components, the workshop will totally focus on the core concepts and advantages of using containers for running web apps, based on Docker, Azure Container Registry (ACR), Azure Container Instance (ACI), as well as how to enable container cloud-scale using Azure Container Services (ACS) with Kubernetes and Azure Kubernetes Service (AKS).

The focus of the workshop is having a Hands-On-Labs experience, by going through the following exercises and tasks:

- Deploying a 2-tier Azure Virtual Machine (Webserver and SQL database Server) using ARM-template automation with Visual Studio 2017;
- Migrating a legacy SQL 2012 database to Azure SQL PaaS (Lift & Shift);
- Migrating a legacy ASP.NET web application to Azure Web Apps (Lift & Shift);
- Containerizing a legacy ASP.NET web application using Docker;
- Running Azure Container Instance (ACI) from an Azure Container Registry (ACR) image;
- Deploy and run Azure Container Services (ACS) with Kubernetes;
- Deploy and run Azure Kubernetes Services (AKS);
- Managing and Monitoring Azure Container Services (ACS) and Azure Kubernetes Services (AKS);

Hands-On-Lab Scenario

The baseline of the hands-on-lab scenario is starting from an 8-year-old legacy ASP.NET application, developed around 2010, currently offering a product catalog browsing web page, pulling the product catalog list from a legacy Microsoft SQL Server 2012 database, running on dedicated Windows Server 2012 R2 Virtual Machines. (This could be seen as a subset of a larger application landscape within your organization, think of manufacturing database information, HR solutions, e-commerce platforms,… and alike). Imagine this application is running in our on-premises datacenter, where you want to perform an "application digital migration" to Azure Public cloud. You will use several Azure cloud-native services and solutions, ranging from Virtual Machines, Platform Services and different Container Solutions on Azure.

Requirements

Naming Conventions:

IMPORTANT: Most Azure resources require unique names. Throughout these steps you will see the word **"[SUFFIX]"** as part of resource names. You should replace this with your initials, guaranteeing those resources get uniquely named.

Azure Subscription:

Participants need a "pay-as-you-go", MSDN or other paid Azure subscription

a) Azure Trial subscriptions won't work
b) In one of the Azure Container Services tasks, you are required to create an Azure AD Service Principal, wich typically requires an Azure subscription owner to log in to create this object. If you don't have the owner right in your Azure subscription, you could ask another person to execute this step for you.
c) The Azure subscription must allow you to run enough cores, used by the baseline Virtual Machines, but also later on in the tasks when deploying the Azure Container Services, where ACS agent and master machines are getting set up. If you follow the instructions as written out in the lab guide, you need 12 cores.
d) If you run this lab setup in your personal or corporate Azure payable subscription, using the configuration as described in the lab guide, the estimated Azure consumption costs for running the setups during the 2 days of the workshop is $20.

Other requirements:

Participants need a local client machine, running a recent Operating System, allowing them to:

- browse to https://portal.azure.com from a most-recent browser;
- establish a secured Remote Desktop (RDP) session to a lab-jumpVM running Windows Server 2016;

Alternative Approach:

Where the lab scenario assumes all exercises will be performed from within the lab-jumpVM, (since several tools will be installed on the lab-jumpVM or are already installed by default), participants could also execute (most, if not all…) steps from their local client machine.

The following tools are being used throughout the lab exercises:

- Visual Studio 2017 community edition (updated to latest version)
- Docker for Windows (updated to latest version)
- Azure CLI 2.0 (updated to latest version)
- Kubernetes CLI (updated to latest version)

Make sure you have these tools installed prior to the workshop, if you are not using the lab-jumpVM. You should also have full administrator rights on your machine to execute certain steps within using these tools.

Final Remarks:

VERY IMPORTANT: You should be typing all of the commands as they appear in the guide, except where explicitly stated in this document. Do not try to copy and paste from Word to your command windows or other documents where you are instructed to enter information shown in this document. There can be issues with Copy and Paste from Word or PDF that result in errors, execution of instructions, or creation of file content.

IMPORTANT: Most Azure resources require unique names. Throughout these steps you will see the word “[SUFFIX]” as part of resource names. You should replace this with your initials, guaranteeing those resources get uniquely named.

Lab 4: Containerizing an ASP.NET web application with Docker

What you will learn

In this lab, we focus on Docker for Windows. Starting with installing the Docker for Windows application, you learn the basics of Docker commands using the Docker Command Line interface. Next, students learn how to 'Dockerize' the legacy ASP.NET code that has been used in former labs. Finally, you learn about Azure Container Registry (ACR) and how to publish your new Docker container in there, as well as using this as a source for Azure Container Instance (ACI) and running your web application.

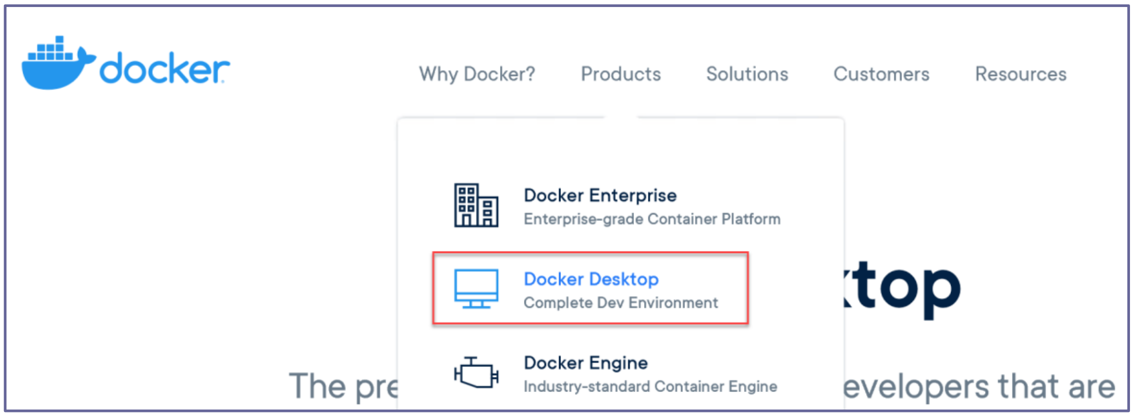
Time Estimate

This lab duration is estimated 60 min.

Task 1: Installing Docker for Windows on the lab-jumpVM

1. If not logged on anymore to the lab-jumpVM, open an RDP session to this Virtual Machine, using labadmin / L@BadminPa55w.rd as credentials.

2. Connect to the Docker website http://www.docker.com; Here, Click Products, and select Docker Desktop.



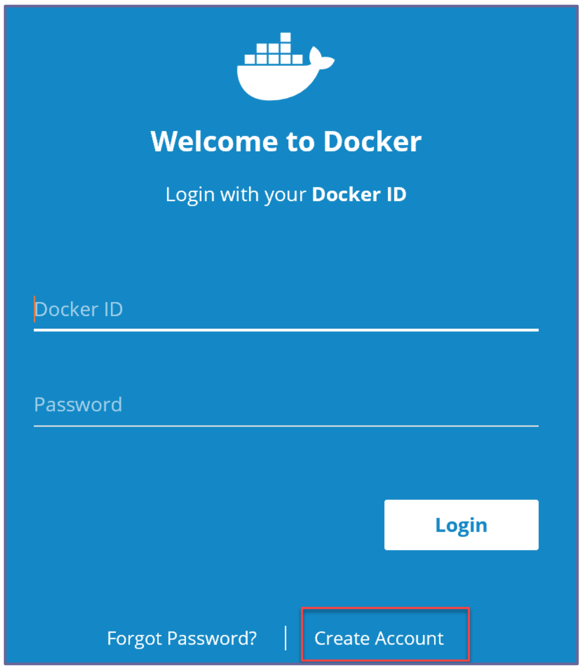
3. Click "Download for Windows".

4. This redirects you to the Docker Store. (the direct link at the time of writing this lab guide is https://store.docker.com/editions/community/docker-ce-desktop-windows)

5. Before you can download the source install files, you need to create a Docker Login. Since you will use this account later on to connect to the Docker Hub, make sure you enter correct

details here.

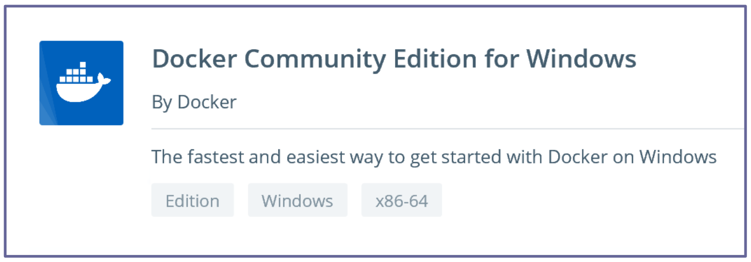
6. Click the Login to Portal button; in the login portal, choose Create Account.



7. Choose a Docker ID, use an active email address and choose a password. Complete the process for account creation (check your email for any activation confirmation email).

8. Once your account creation and activation is done, redirect to http://store.docker.com again.

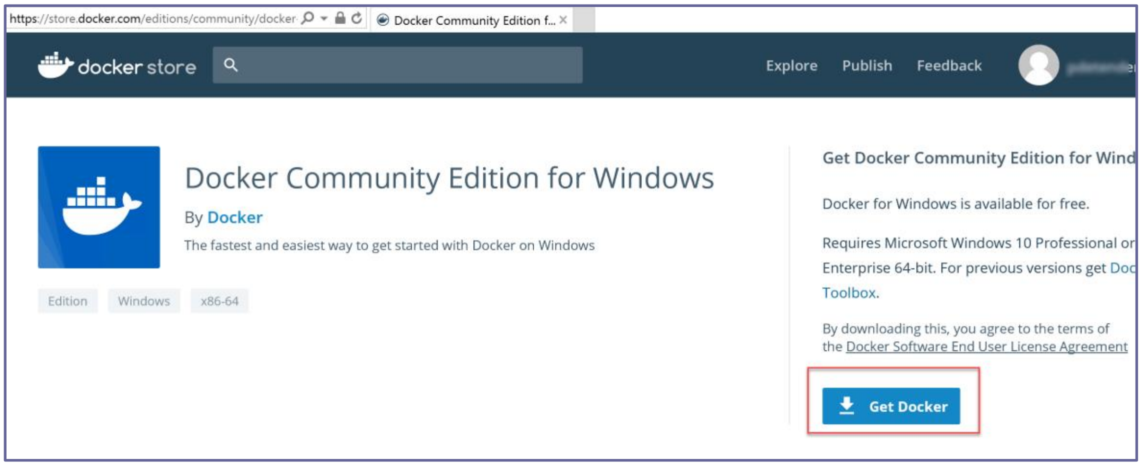
9. Here, Choose Docker CE; in the search result page, scroll down until you see Docker Community Edition for Windows.


Docker Community Edition for Windows

By Docker

The fastest and easiest way to get started with Docker on Windows

10. Click on the object title, which opens the download page for this solution. Click the Get Docker button.


https://store.docker.com/editions/community/docker

Docker Community Edition for Windows

By Docker

The fastest and easiest way to get started with Docker on Windows

Edition Windows x86-64

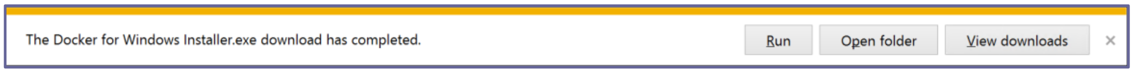
Get Docker Community Edition for Wind

Docker for Windows is available for free.

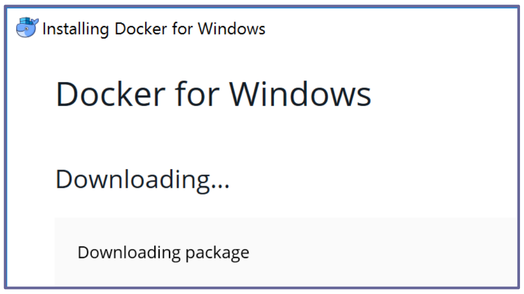
Requires Microsoft Windows 10 Professional or Enterprise 64-bit. For previous versions get Doc Toolbox.

By downloading this, you agree to the terms of the Docker Software End User License Agreement

11. **Save** the install files to the local machine's Downloads folder; wait for the download to complete, and run the actual install process.


The Docker for Windows Installer.exe download has completed.

12. The **Docker for Windows** installer kicks off

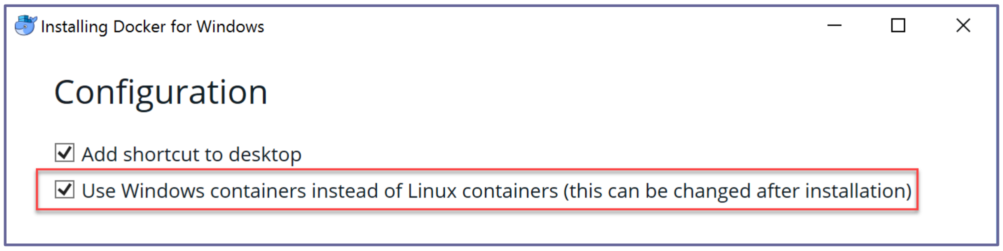

Installing Docker for Windows

Docker for Windows

Downloading...

Downloading package

13. Wait for the background download to complete, until you see the **Configuration** step of the install wizard. **Here, select "Use Windows containers instead of Linux containers"**. Press **OK** to continue.

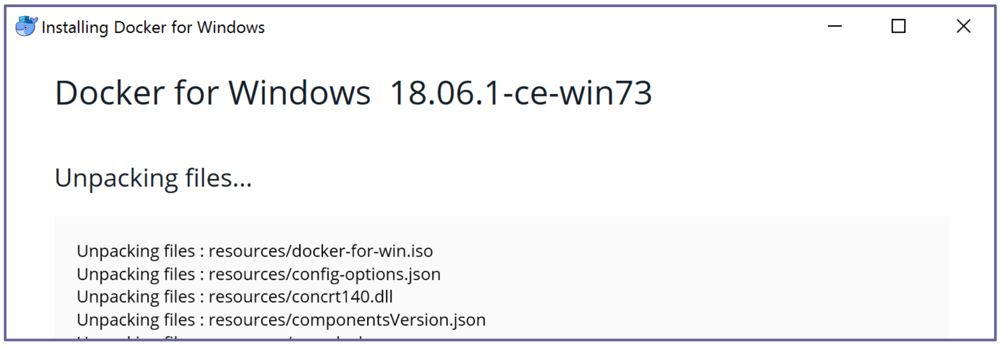

Installing Docker for Windows

Configuration

☑ Add shortcut to desktop

☑ Use Windows containers instead of Linux containers (this can be changed after installation)

14. The Docker install will update the install packages and run the actual tool installation. Wait for this step to complete.

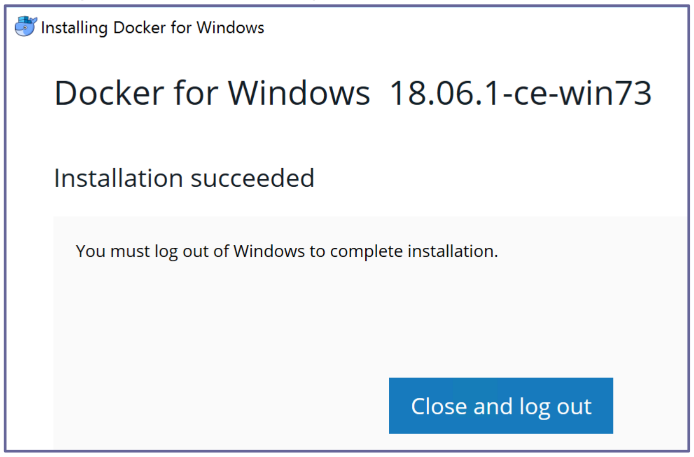

Installing Docker for Windows

Docker for Windows 18.06.1-ce-win73

Unpacking files...

```
Unpacking files : resources/docker-for-win.iso
Unpacking files : resources/config-options.json
Unpacking files : resources/concrt140.dll
Unpacking files : resources/componentsVersion.json
```

15. Wait for the installation to complete; once prompted, log out from the RDP session, by clicking the Close and Log out button.

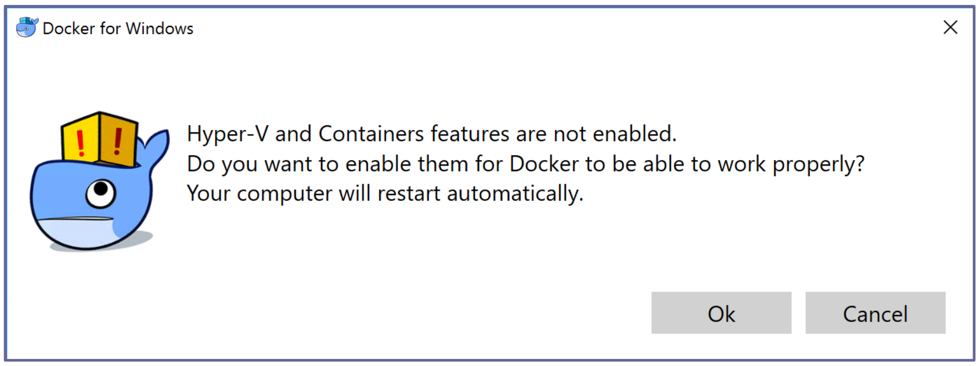

Installing Docker for Windows

Docker for Windows 18.06.1-ce-win73

Installation succeeded

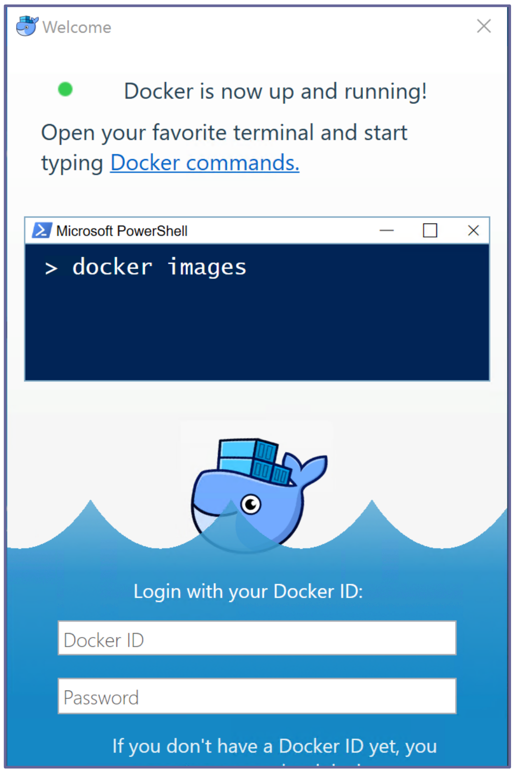
You must log out of Windows to complete installation.

16. Once your are disconnected from the RDP session, log back on to the lab-jumpVM. You are prompted by Docker it needs to enable the Hyper-V and Containers features.Click OK to get this configured.


Docker for Windows

Hyper-V and Containers features are not enabled.
Do you want to enable them for Docker to be able to work properly?
Your computer will restart automatically.

17. After a while, your lab-jumpVM server will reboot. Wait about a minute and open the RDP session to this virtual machine again. Start Docker for Windows by clicking its shortcut from the desktop.


Welcome
Docker is now up and running!
Open your favorite terminal and start typing Docker commands.
Microsoft PowerShell
Login with your Docker ID:
Docker ID
Password
If you don't have a Docker ID yet, you

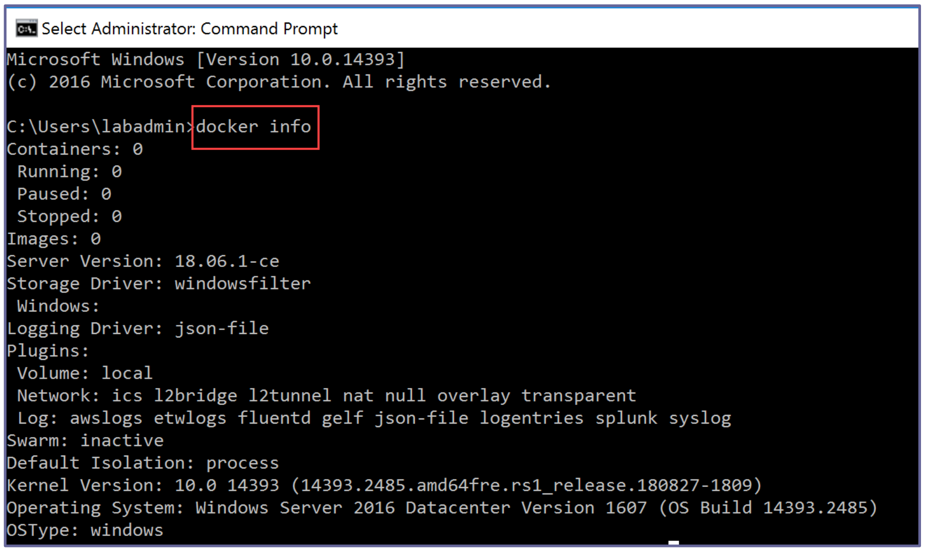
```
> docker images
```

18. Enter your Docker credentials.

19. Once you have successfully logged on to Docker CE, open a command prompt from the Start screen. (Note: you can also run this from within PowerShell if that gets your preference)

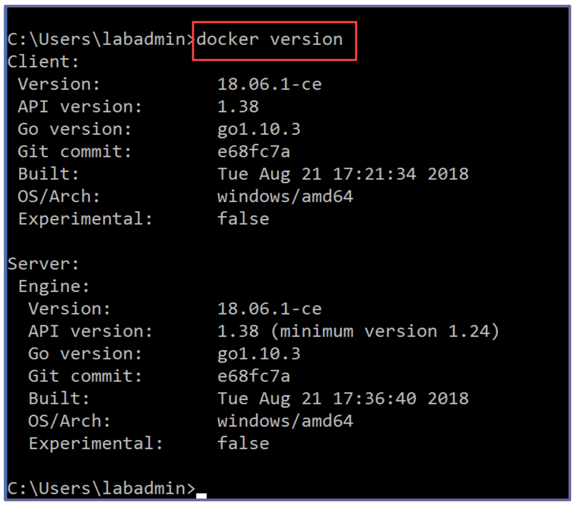
20. Execute the following command:

```
docker info
```



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

C:\Users\labadmin>docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 18.06.1-ce
Storage Driver: windowsfilter
 Windows:
Logging Driver: json-file
Plugins:
 Volume: local
 Network: ics l2bridge l2tunnel nat null overlay transparent
 Log: awslogs etwlogs fluentd gelf json-file logentries splunk syslog
Swarm: inactive
Default Isolation: process
Kernel Version: 10.0 14393 (14393.2485.amd64fre.rs1_release.180827-1809)
Operating System: Windows Server 2016 Datacenter Version 1607 (OS Build 14393.2485)
OSType: windows
```

21. Next, to validate the Docker version you are running, initiate `docker version` in the command prompt



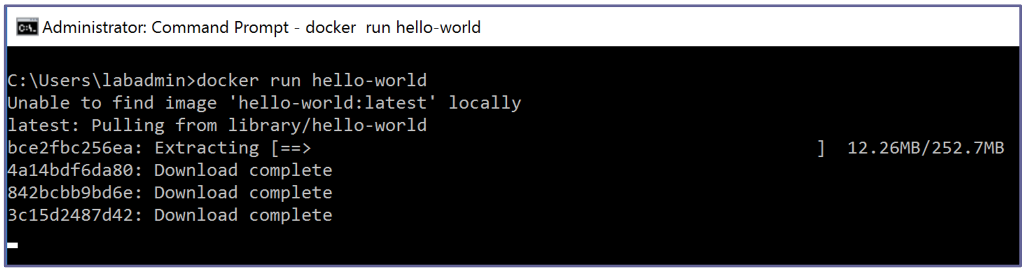
```
C:\Users\labadmin>docker version
Client:
 Version:           18.06.1-ce
 API version:       1.38
 Go version:        go1.10.3
 Git commit:        e68fc7a
 Built:             Tue Aug 21 17:21:34 2018
 OS/Arch:           windows/amd64
 Experimental:      false

Server:
 Engine:
  Version:          18.06.1-ce
  API version:      1.38 (minimum version 1.24)
  Go version:       go1.10.3
  Git commit:       e68fc7a
  Built:            Tue Aug 21 17:36:40 2018
  OS/Arch:          windows/amd64
  Experimental:     false

C:\Users\labadmin>
```

22. To validate we can actually "run" a Docker container, initiate the following command:

```
docker run hello-world
```

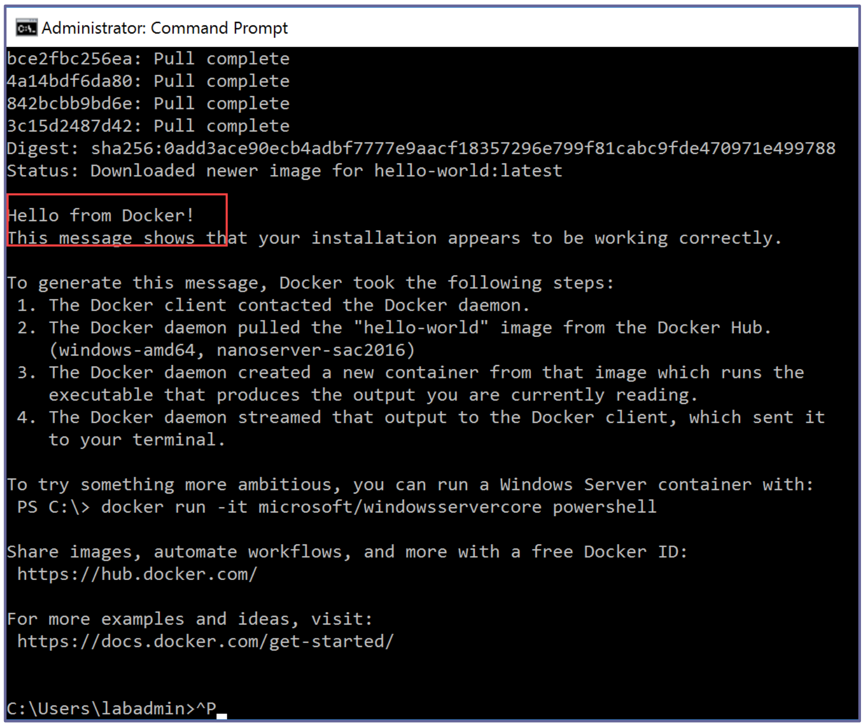
Some explanation what this command executes:

- <Docker>	"Hey Docker engine, I need you…"
- <run>	"… to do something…"
- <hello-world>	"run that container"



```
Administrator: Command Prompt - docker  run hello-world

C:\Users\labadmin>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
bce2fbc256ea: Extracting [==>                                                ]  12.26MB/252.7MB
4a14bdf6da80: Download complete
842bcbb9bd6e: Download complete
3c15d2487d42: Download complete
```

which initiates a download from the public Docker Hub repository (note you provided your Docker ID credentials for this…), and spins up the actual Docker Container. This is a small sample container echo-ing "hello-world" on screen as output.:

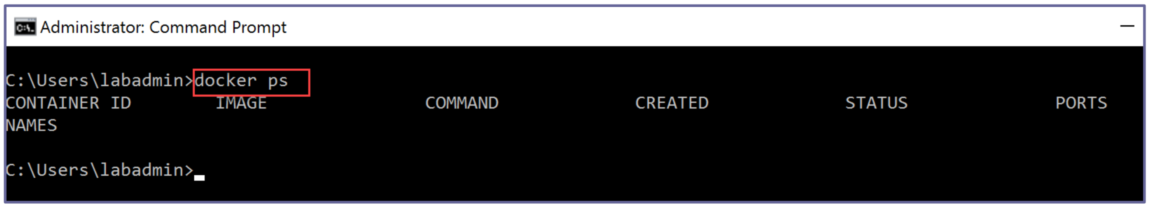


```
Administrator: Command Prompt
bce2fbc256ea: Pull complete
4a14bdf6da80: Pull complete
842bcbb9bd6e: Pull complete
3c15d2487d42: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (windows-amd64, nanoserver-sac2016)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run a Windows Server container with:
 PS C:\> docker run -it microsoft/windowsservercore powershell

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/


C:\Users\labadmin>^P
```

Task 2: Operating Docker from the command line

1. Another useful Docker command is to see what containers are running. Launch the following

```
docker ps
```

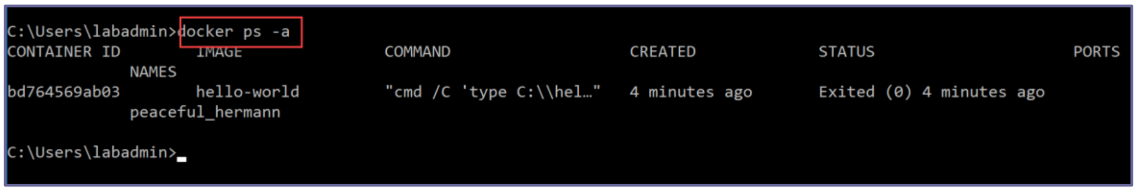


```
Administrator: Command Prompt

C:\Users\labadmin>docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES

C:\Users\labadmin>_
```

Note we don't have any Docker containers running; this was part of the hello-world container, which runs, and eventually automatically stops again.

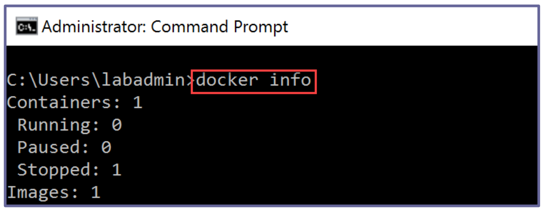
2. Like the previous command, you can add a "-a" parameter, showing you what containers were previously running

   `docker ps -a`



```
C:\Users\labadmin>docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS
            NAMES
bd764569ab03        hello-world         "cmd /C 'type C:\\hel…"   4 minutes ago       Exited (0) 4 minutes ago
            peaceful_hermann

C:\Users\labadmin>_
```

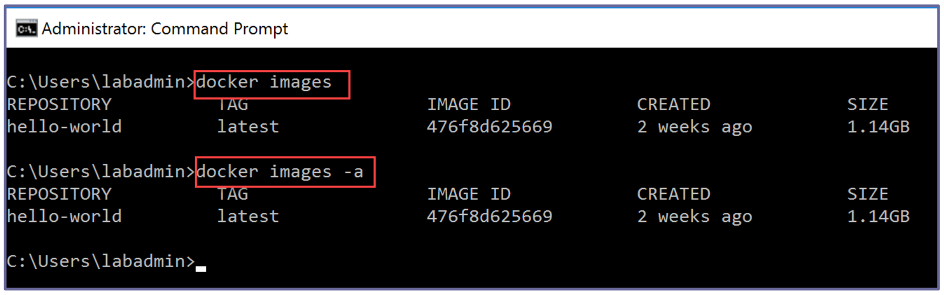
3. Execute the command `Docker info` again; it will now have additional information related to our containers and images



```
Administrator: Command Prompt

C:\Users\labadmin>docker info
Containers: 1
 Running: 0
 Paused: 0
 Stopped: 1
Images: 1
```

4. Checking what Docker images we have, is done using the following command:

   `docker images`

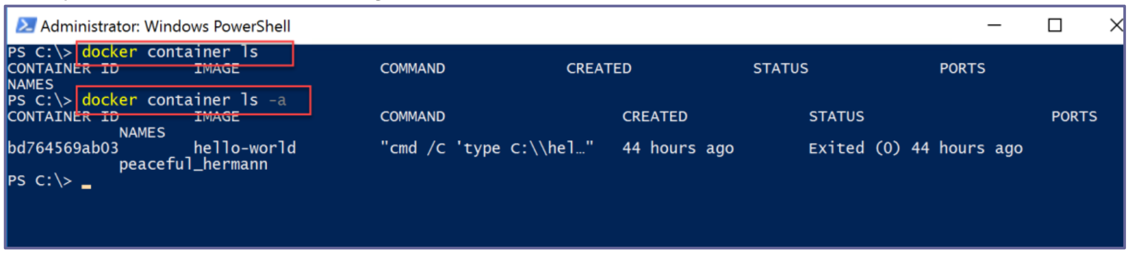


```
Administrator: Command Prompt

C:\Users\labadmin>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              476f8d625669        2 weeks ago         1.14GB

C:\Users\labadmin>docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              476f8d625669        2 weeks ago         1.14GB

C:\Users\labadmin>_
```

5. To get a view on the Docker Containers we have on your system, use the following command:

   `docker container ls` or `docker container ls -a`

   (Note I switched to PowerShell from here on, to show you both command line tools are transparent across Docker usage)

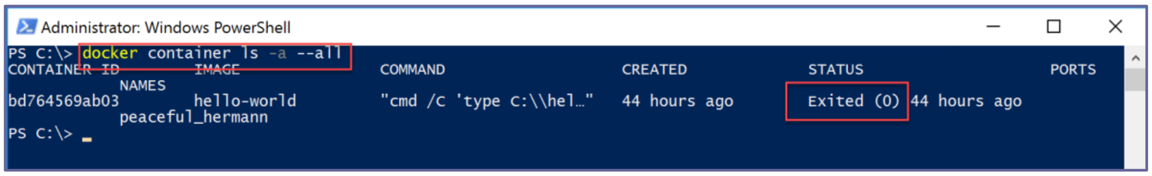


```
Administrator: Windows PowerShell
PS C:\> docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
PS C:\> docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                   PORTS
          NAMES
bd764569ab03        hello-world         "cmd /c 'type C:\\hel…"   44 hours ago        Exited (0) 44 hours ago
          peaceful_hermann
PS C:\> _
```

   Notice the difference between both commands though. Running the first, shows the active running containers (there are no running in our example – remember the Hello-World one automatically stopped after running). Where the second one reveals the containers we "had running" in the past. Where the hello-world example nicely shows up.

6. Now, execute the following command:

   docker container ls -a –all

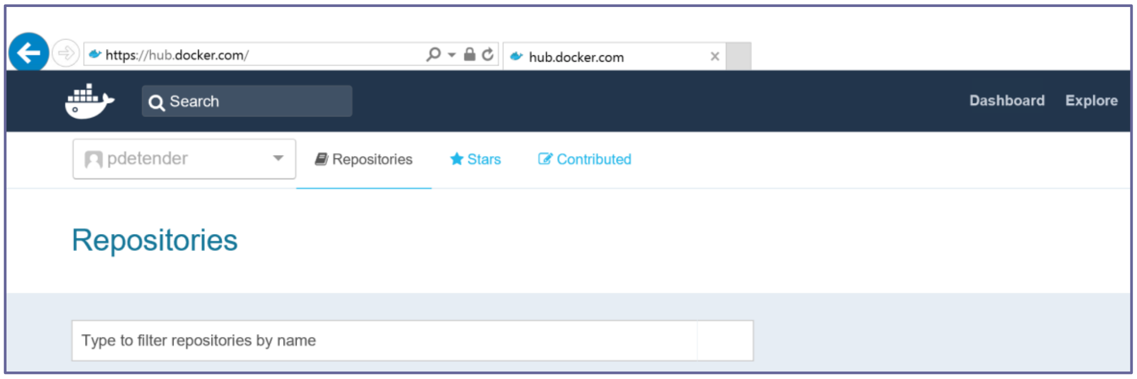


```
Administrator: Windows PowerShell
PS C:\> docker container ls -a --all
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                   PORTS
          NAMES
bd764569ab03        hello-world         "cmd /c 'type C:\\hel…"   44 hours ago        Exited (0) 44 hours ago
          peaceful_hermann
PS C:\> _
```

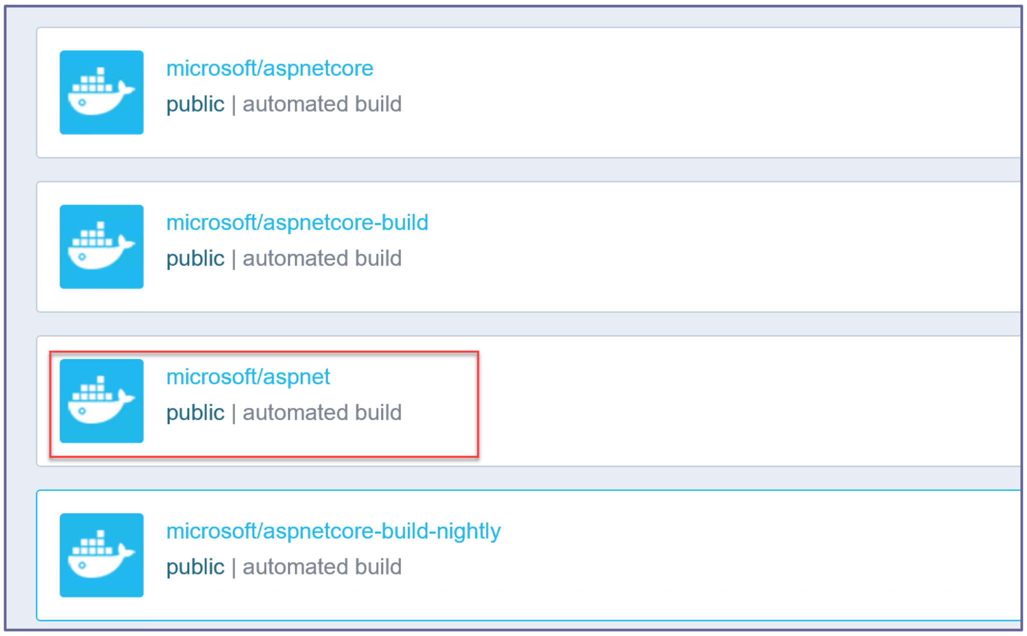
   Which shows you this container is in an exited (=stopped) state since 44 hours. (if you are wondering where the 44 hours comes from, no worries, lab guide authors have a life too 😉).

Similar to the hello-world container images that got pulled down before running, let's move on with exploring the different Windows OS-based container images in Docker Store, and pulling them to our local lab-jumpVM.

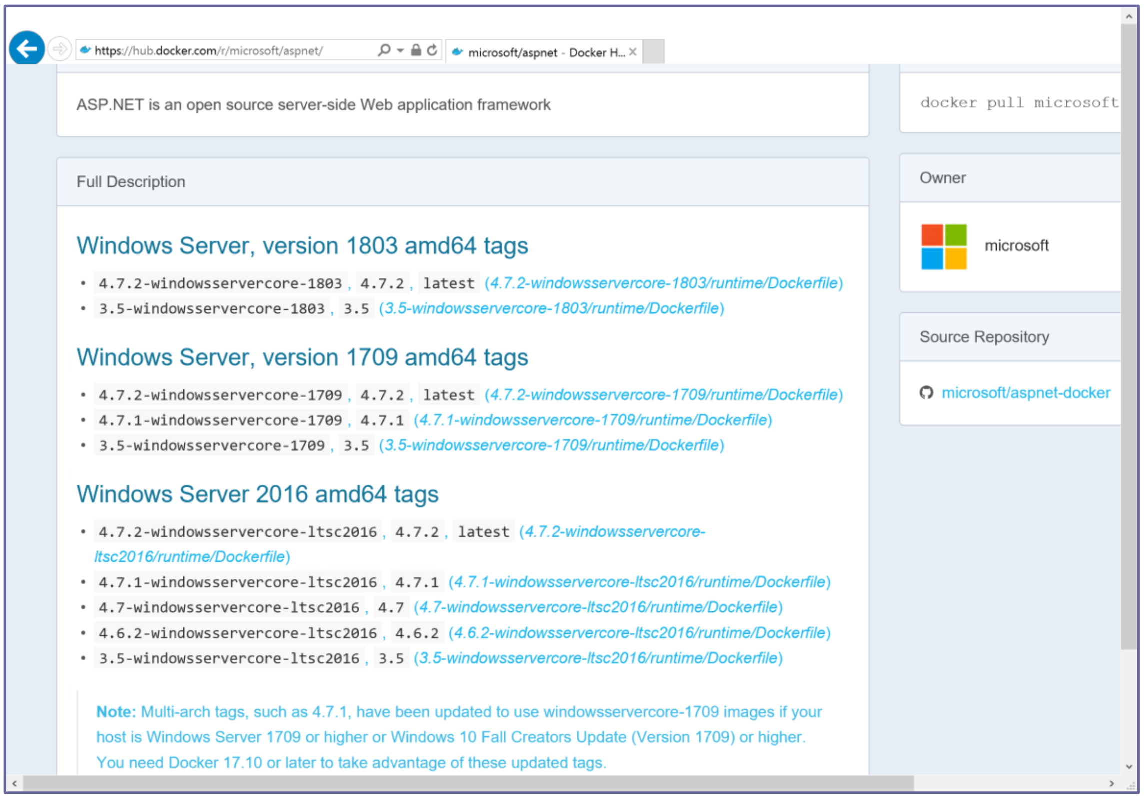
7. Connect to hub.docker.com from your internet browser, and log on with your previously created DockerID credentials.



8. In the upper **Search field, type "aspnet";** this results in a list of repositories.



9. **Select the "microsoft/aspnet"** repository. Which shows you the different ASPNET-based Docker images, provided by Microsoft. Important difference to note is the several Windows Server Operating System versions offered here.



https://hub.docker.com/r/microsoft/aspnet/

microsoft/aspnet - Docker H...

ASP.NET is an open source server-side Web application framework

docker pull microsoft

Full Description

Owner: microsoft

Source Repository: microsoft/aspnet-docker

Windows Server, version 1803 amd64 tags

- `4.7.2-windowsservercore-1803`, `4.7.2`, `latest` (4.7.2-windowsservercore-1803/runtime/Dockerfile)
- `3.5-windowsservercore-1803`, `3.5` (3.5-windowsservercore-1803/runtime/Dockerfile)

Windows Server, version 1709 amd64 tags

- `4.7.2-windowsservercore-1709`, `4.7.2`, `latest` (4.7.2-windowsservercore-1709/runtime/Dockerfile)
- `4.7.1-windowsservercore-1709`, `4.7.1` (4.7.1-windowsservercore-1709/runtime/Dockerfile)
- `3.5-windowsservercore-1709`, `3.5` (3.5-windowsservercore-1709/runtime/Dockerfile)

Windows Server 2016 amd64 tags

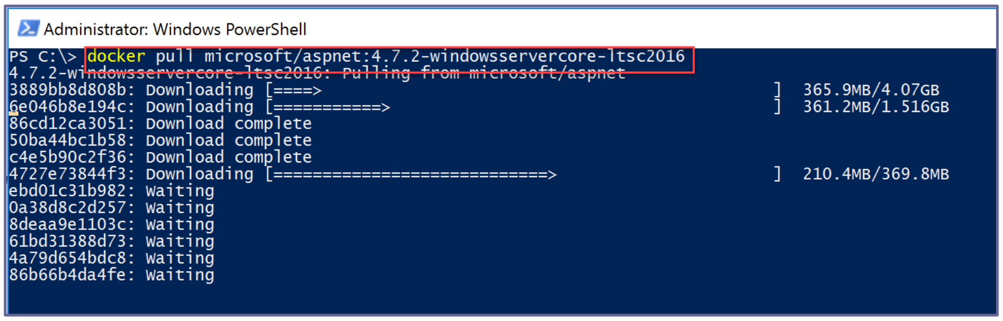
- `4.7.2-windowsservercore-ltsc2016`, `4.7.2`, `latest` (4.7.2-windowsservercore-ltsc2016/runtime/Dockerfile)
- `4.7.1-windowsservercore-ltsc2016`, `4.7.1` (4.7.1-windowsservercore-ltsc2016/runtime/Dockerfile)
- `4.7-windowsservercore-ltsc2016`, `4.7` (4.7-windowsservercore-ltsc2016/runtime/Dockerfile)
- `4.6.2-windowsservercore-ltsc2016`, `4.6.2` (4.6.2-windowsservercore-ltsc2016/runtime/Dockerfile)
- `3.5-windowsservercore-ltsc2016`, `3.5` (3.5-windowsservercore-ltsc2016/runtime/Dockerfile)

Note: Multi-arch tags, such as 4.7.1, have been updated to use windowsservercore-1709 images if your host is Windows Server 1709 or higher or Windows 10 Fall Creators Update (Version 1709) or higher. You need Docker 17.10 or later to take advantage of these updated tags.

As we will be using our ASP.NET application as a source for different Azure Container solutions (ACI, ACR, ACS,…), we have to use the **4.7.2-windowsservercore-ltsc2016** tagged version, which is one of the supported Windows-based containers at the time of writing.

10. **From the Command Prompt or PowerShell**, initiate the following Docker command to pull the public Docker image to our lab-JumpVM:

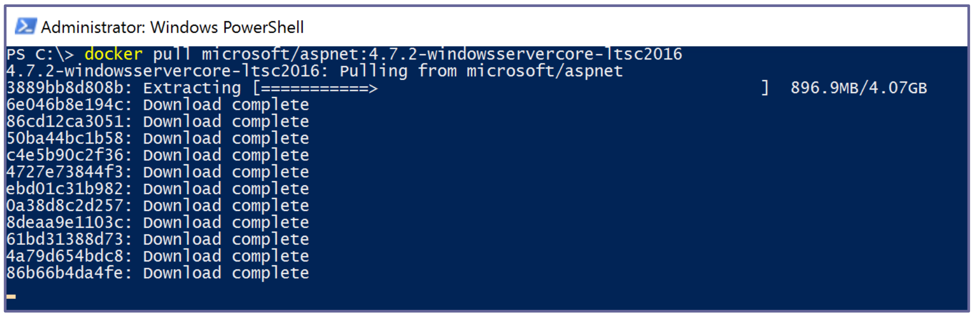
    docker pull microsoft/aspnet:4.7.2-windowsservercore-ltsc2016



Administrator: Windows PowerShell

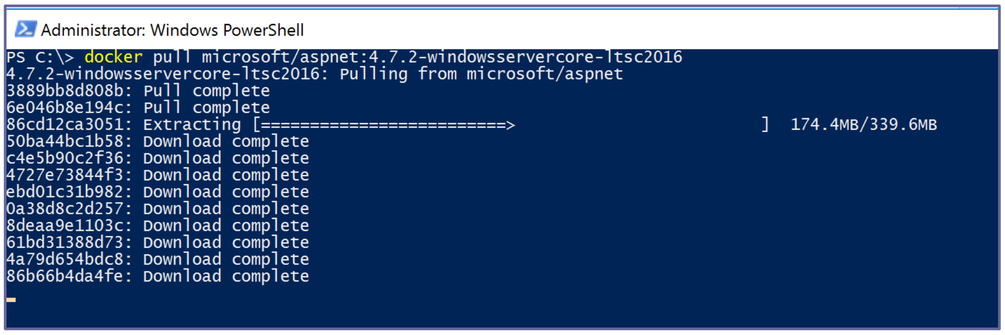
```
PS C:\> docker pull microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
4.7.2-windowsservercore-ltsc2016: Pulling from microsoft/aspnet
3889bb8d808b: Downloading [====>                                        ]  365.9MB/4.07GB
6e046b8e194c: Downloading [===========>                                 ]  361.2MB/1.516GB
86cd12ca3051: Download complete
50ba44bc1b58: Download complete
c4e5b90c2f36: Download complete
4727e73844f3: Downloading [============================>                ]  210.4MB/369.8MB
ebd01c31b982: Waiting
0a38d8c2d257: Waiting
8deaa9e1103c: Waiting
61bd31388d73: Waiting
4a79d654bdc8: Waiting
86b66b4da4fe: Waiting
```

Notice this image is based on different "layers" (the 3889bb…, 50ba44…, ebd01…)



Administrator: Windows PowerShell

```
PS C:\> docker pull microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
4.7.2-windowsservercore-ltsc2016: Pulling from microsoft/aspnet
3889bb8d808b: Extracting [===========>                                       ]  896.9MB/4.07GB
6e046b8e194c: Download complete
86cd12ca3051: Download complete
50ba44bc1b58: Download complete
c4e5b90c2f36: Download complete
4727e73844f3: Download complete
ebd01c31b982: Download complete
0a38d8c2d257: Download complete
8deaa9e1103c: Download complete
61bd31388d73: Download complete
4a79d654bdc8: Download complete
86b66b4da4fe: Download complete
```



Administrator: Windows PowerShell

```
PS C:\> docker pull microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
4.7.2-windowsservercore-ltsc2016: Pulling from microsoft/aspnet
3889bb8d808b: Pull complete
6e046b8e194c: Pull complete
86cd12ca3051: Extracting [=========================>                         ]  174.4MB/339.6MB
50ba44bc1b58: Download complete
c4e5b90c2f36: Download complete
4727e73844f3: Download complete
ebd01c31b982: Download complete
0a38d8c2d257: Download complete
8deaa9e1103c: Download complete
61bd31388d73: Download complete
4a79d654bdc8: Download complete
86b66b4da4fe: Download complete
```

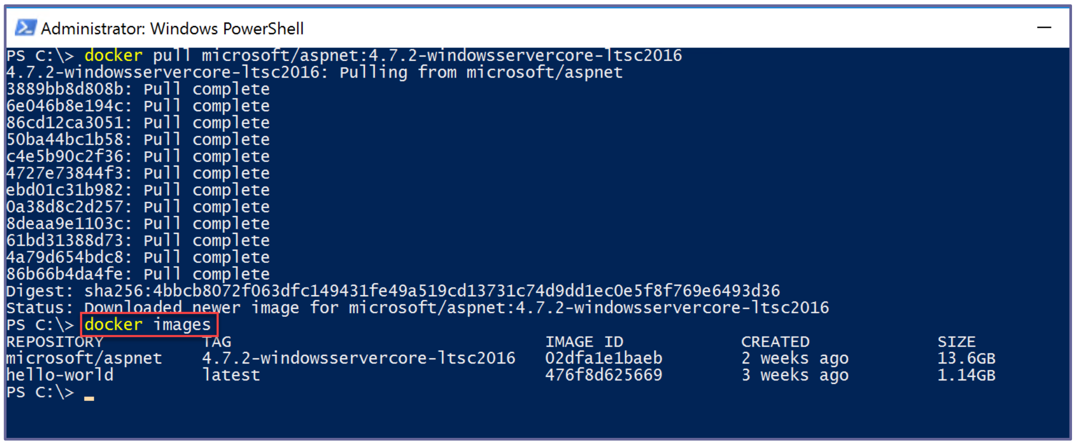
11. Wait for the image to finish downloading and extracting. Depending on your internet connection speed, this might take several minutes. From a container perspective, this is based on the Microsoft/aspnet, having a tag pointing to the latest edition (the 4.7.2-windowsservercore-ltsc2016 part in the name).

Task 3: Containerize your ASP.NET application using Visual Studio and Docker

1. From the Docker command line (Command Prompt or PowerShell), run the following Docker command:

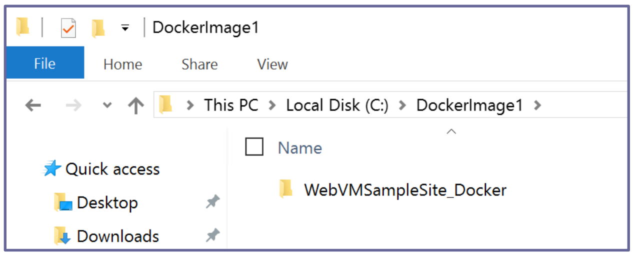
   ```
   docker images
   ```

   and validate the Docker image for microsoft/aspnet is showing up.



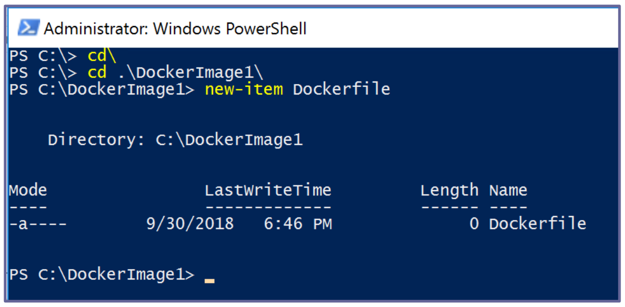
```
PS C:\> docker pull microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
4.7.2-windowsservercore-ltsc2016: Pulling from microsoft/aspnet
3889bb8d808b: Pull complete
6e046b8e194c: Pull complete
86cd12ca3051: Pull complete
50ba44bc1b58: Pull complete
c4e5b90c2f36: Pull complete
4727e73844f3: Pull complete
ebd01c31b982: Pull complete
0a38d8c2d257: Pull complete
8deaa9e1103c: Pull complete
61bd31388d73: Pull complete
4a79d654bdc8: Pull complete
86b66b4da4fe: Pull complete
Digest: sha256:4bbcb8072f063dfc149431fe49a519cd13731c74d9dd1ec0e5f8f769e6493d36
Status: Downloaded newer image for microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
PS C:\> docker images
REPOSITORY          TAG                                 IMAGE ID            CREATED             SIZE
microsoft/aspnet    4.7.2-windowsservercore-ltsc2016    02dfa1e1baeb        2 weeks ago         13.6GB
hello-world         latest                              476f8d625669        3 weeks ago         1.14GB
PS C:\> _
```

2. The next step involves creating our **Dockerfile**, which holds the different steps and actions to get our web application published into a Docker container. To do this in some sort of structured, but yet easy way, start with **creating a new folder in the C:\, called DockerImage1, and copy the folder c:\WebVMSampleSite** into it, renaming this folder after the copy is completed to **webvmsamplesite_docker**. (as such, you can detect the differences in source files when something would go wrong).



3. Next, initiate the creation of a new Dockerfile document, by running the following Powershell commands:

cd\	(this moves you to the root of C:\)
cd DockerImage1	(this moves you to the C:\DockerImage1 folder)
new-item Dockerfile	(this creates a new file called "Dockerfile", with no extension)



```
PS C:\> cd\
PS C:\> cd .\DockerImage1\
PS C:\DockerImage1> new-item Dockerfile


    Directory: C:\DockerImage1


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        9/30/2018   6:46 PM              0 Dockerfile


PS C:\DockerImage1> _
```

4. **Open** the Dockerfile file in Notepad for easy editing. From PowerShell, you can run "notepad Dockerfile", or browse to the file from File explorer and open it with Notepad.

5. **Type in** the following lines **into the Notepad window** (copy/paste might work, but sometimes injects garbage code, which breaks the file – therefore, typing is the safest option).

   ```
   FROM microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
   ADD WebVMSampleSite_Docker /inetpub/wwwroot/
   EXPOSE 80
   ```

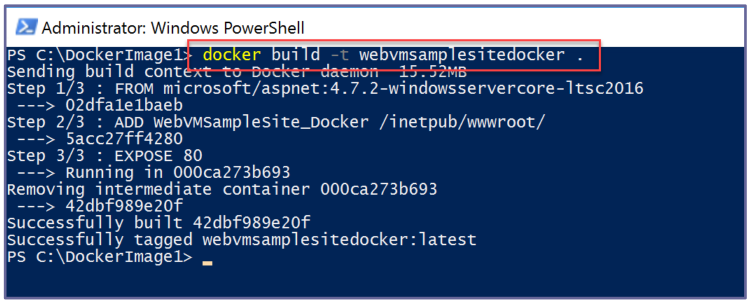
   and **save the file**.

6. Some words of explanation what this Dockerfile does:
   - the **FROM** statement defines what Docker image we will use as a starting point, meaning, this Container should be based on the Windows Server Cored LTSC2016 version
   - the **ADD** statement tells the Docker image to copy all contents from the subfolder "WebVMSampleSite_Docker to the inetpub/wwwroot folder, which is running inside the container (=the default IIS folder on a regular Windows machine)
   - the **EXPOSE 80** statement allows running the web site on port 80, allowing us to connect to the IP-address of the container, and browsing to it on port 80, validating the web site is running as expected.

7. **Back in the PowerShell window,** initiate the following command to get the Docker image build:

   ```
   docker build -t webvmsamplesitedocker .
   ```

   (Note the dot at the end – what this points at is creating a Docker image from all content in the current folder, which technically is everything in the \WebVMSampleSite_Docker folder)

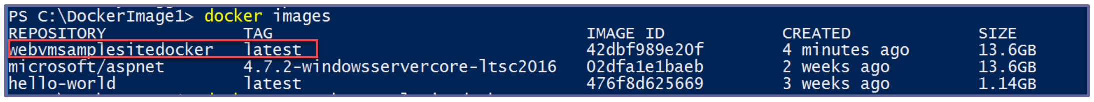


```
Administrator: Windows PowerShell
PS C:\DockerImage1> docker build -t webvmsamplesitedocker .
Sending build context to Docker daemon  15.52MB
Step 1/3 : FROM microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
 ---> 02dfa1e1baeb
Step 2/3 : ADD WebVMSampleSite_Docker /inetpub/wwwroot/
 ---> 5acc27ff4280
Step 3/3 : EXPOSE 80
 ---> Running in 000ca273b693
Removing intermediate container 000ca273b693
 ---> 42dbf989e20f
Successfully built 42dbf989e20f
Successfully tagged webvmsamplesitedocker:latest
PS C:\DockerImage1> _
```

8. **Validate** the image is built successfully by running the following command:

   ```
   docker images
   ```

and notice the webvmsamplesitedocker is in the list, having a tag of "latest"



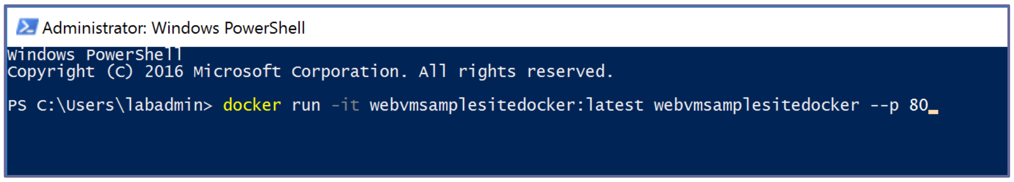
```
PS C:\DockerImage1> docker images
REPOSITORY             TAG                                  IMAGE ID       CREATED          SIZE
webvmsamplesitedocker  latest                               42dbf989e20f   4 minutes ago    13.6GB
microsoft/aspnet       4.7.2-windowsservercore-ltsc2016     02dfa1e1baeb   2 weeks ago      13.6GB
hello-world            latest                               476f8d625669   3 weeks ago      1.14GB
```

Now our image is available, let us **spin up a new container** using this image, by executing the following command:

```
docker run -it -d --publish 80 --name webvmsamplesitedocker
webvmsamplesitedocker:latest
```

Where:

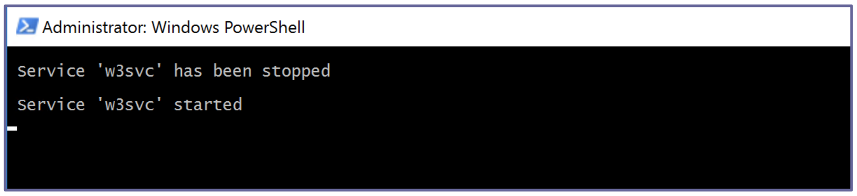
- "docker run"	start the container run process
- "-it"	defines the container needs to run in interactive mode (output)
- "webvm…:latest"	the name of the image to use, note the tag
- "webvm…"	the name for the container
-"—p 80"	run the container on port 80



Administrator: Windows PowerShell

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

PS C:\Users\labadmin> docker run -it webvmsamplesitedocker:latest webvmsamplesitedocker --p 80_
```

Where the PowerShell script executes the container, and shows us the "interactive" mode output on screen. Since we are running a web site, it is interesting to see it started the "w3svc" service, pointing to the World Wide Web Service within a Windows Server Operating System.

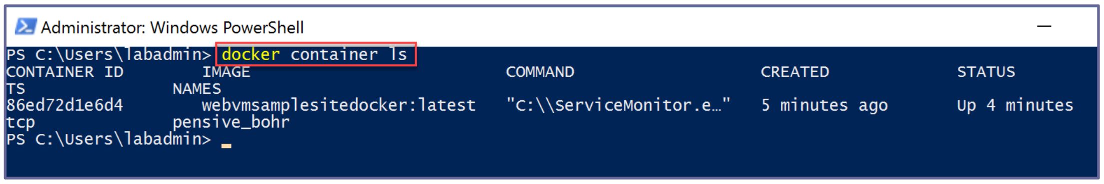


Administrator: Windows PowerShell

```
Service 'w3svc' has been stopped

Service 'w3svc' started
```

9. **From a new PowerShell window**, let us check on the details of this running container, by executing the following command:

```
docker container ls
```



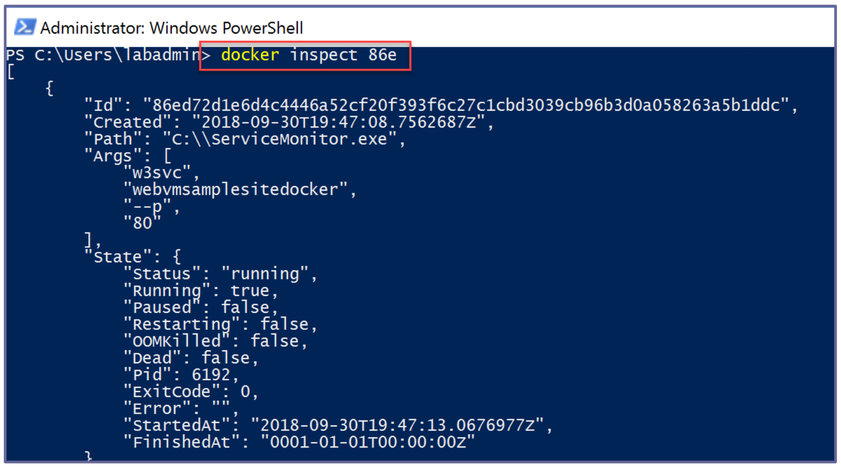
```
Administrator: Windows PowerShell
PS C:\Users\labadmin> docker container ls
CONTAINER ID        IMAGE                              COMMAND                  CREATED             STATUS
TS                  NAMES
86ed72d1e6d4        webvmsamplesitedocker:latest       "C:\\ServiceMonitor.e…"   5 minutes ago       Up 4 minutes
tcp                 pensive_bohr
PS C:\Users\labadmin> _
```

This shows our running container.

10. **Next,** let's check on **some additional and interesting technical details** for this specific container, by running the following command:

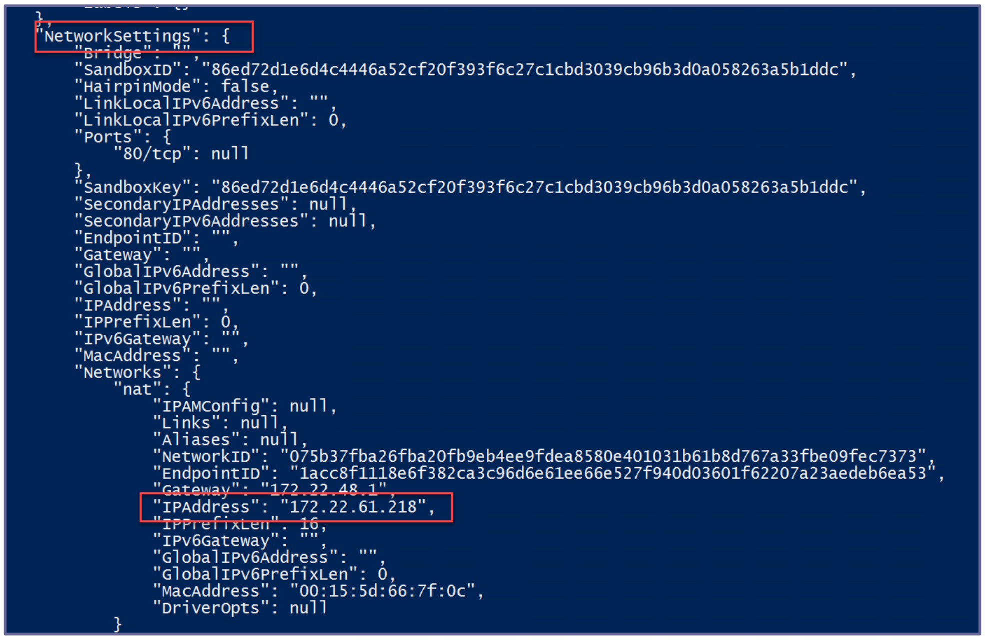
    Note: each container has an ID (the 86ed721e6d4 in my example), where we can reuse this in the docker commands, referring to that object. Easier than using the long names 😉

    `docker inspect 86ed`   (where 86ed should be replaced with the ID of your container)



```
Administrator: Windows PowerShell
PS C:\Users\labadmin> docker inspect 86e
[
    {
        "Id": "86ed72d1e6d4c4446a52cf20f393f6c27c1cbd3039cb96b3d0a058263a5b1ddc",
        "Created": "2018-09-30T19:47:08.7562687Z",
        "Path": "C:\\ServiceMonitor.exe",
        "Args": [
            "w3svc",
            "webvmsamplesitedocker",
            "--p",
            "80"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 6192,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2018-09-30T19:47:13.0676977Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        }
```

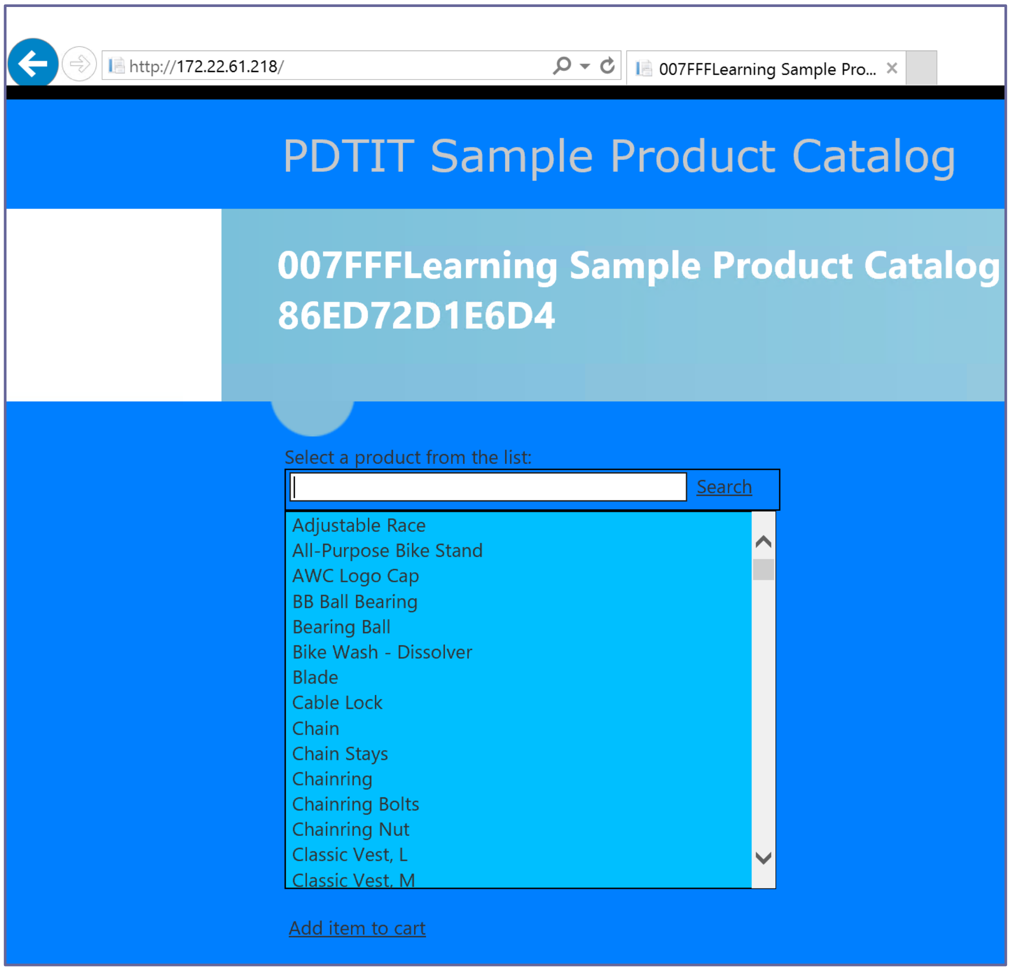
11. In this JSON output file, scroll down and search for the **Networking** section:



```
    },
    "NetworkSettings": {
        "Bridge": "",
        "SandboxID": "86ed72d1e6d4c4446a52cf20f393f6c27c1cbd3039cb96b3d0a058263a5b1ddc",
        "HairpinMode": false,
        "LinkLocalIPv6Address": "",
        "LinkLocalIPv6PrefixLen": 0,
        "Ports": {
            "80/tcp": null
        },
        "SandboxKey": "86ed72d1e6d4c4446a52cf20f393f6c27c1cbd3039cb96b3d0a058263a5b1ddc",
        "SecondaryIPAddresses": null,
        "SecondaryIPv6Addresses": null,
        "EndpointID": "",
        "Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "MacAddress": "",
        "Networks": {
            "nat": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": null,
                "NetworkID": "075b37fba26fba20fb9eb4ee9fdea8580e401031b61b8d767a33fbe09fec7373",
                "EndpointID": "1acc8f1118e6f382ca3c96d6e61ee66e527f940d03601f62207a23aedeb6ea53",
                "Gateway": "172.22.48.1",
                "IPAddress": "172.22.61.218",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "00:15:5d:66:7f:0c",
                "DriverOpts": null
            }
```

12. Here, check for the IPAddress under the Networks / nat section specifically. This reveals the IP-address that is used by our running container.

13. From your internet browser, connect to this IP-address, which shows you the running web application. Yeah! (Notice it also refers to 86ED… as hostname, which is the technical ID of our container)



PDTIT Sample Product Catalog

007FFFLearning Sample Product Catalog 86ED72D1E6D4

Select a product from the list:

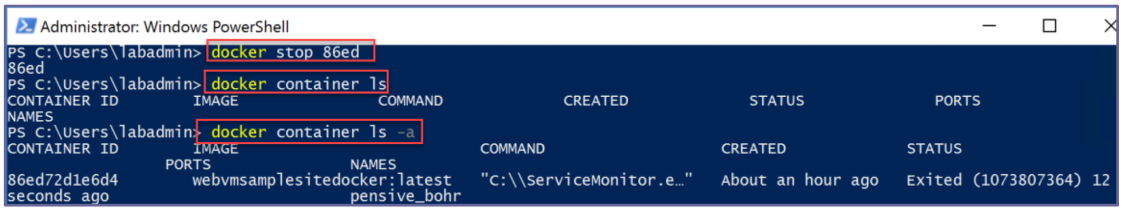
Adjustable Race
All-Purpose Bike Stand
AWC Logo Cap
BB Ball Bearing
Bearing Ball
Bike Wash - Dissolver
Blade
Cable Lock
Chain
Chain Stays
Chainring
Chainring Bolts
Chainring Nut
Classic Vest, L
Classic Vest, M

14. To **stop** our running Docker container, **execute** the following command, **one after the other**:

`Docker stop 86ed` (where 86ed should be replaced with the ID of your container)

`Docker container ls` (showing no information, meaning no running containers)

`Docker container ls -a` (showing our 86ed… container, with a status Exited)



```
PS C:\Users\labadmin> docker stop 86ed
86ed
PS C:\Users\labadmin> docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
PS C:\Users\labadmin> docker container ls -a
CONTAINER ID        IMAGE                              COMMAND                    CREATED             STATUS                         
                    PORTS               NAMES
86ed72d1e6d4        webvmsamplesitedocker:latest       "C:\\ServiceMonitor.e…"    About an hour ago   Exited (1073807364) 12
seconds ago                             pensive_bohr
```

Summary

In this lab, you learned about installing Docker for Windows. Next, you learned the basics of running a sample Hello-world Docker image and container, followed by executing several Docker commands that are common when operating Docker images and containers. The next task involved ‘containerizing’ your Visual Studio source web site, and running this on your local Docker machine. Lastly, you stopped the running Docker container.

Migrating a legacy ASP.NET 2-tier application to Azure using Container Services

Hands-On-Labs step-by-step guides

Prepared by:

Peter De Tender

CEO and Lead Technical Trainer
PDTIT and 007FFFLearning.com

@pdtit @007FFFLearning

Version: October 2018 – 2.0