



# Azure Developer Series

Migrating a dotnetcore 2-tier application to Azure,  
using different architectures and DevOps best practices

Hands-On-Labs step-by-step guides

Prepared by:

Peter De Tender

CEO and Lead Technical Trainer  
PDTIT and 007FFFLearning.com

@pdtit @007FFFLearning

Version: Sept 2019 – 1.0

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/Usage/General.aspx> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

# Contents

Migrating a dotnetcore 2-tiered application to Azure using different architectures and DevOps best practices - Hands-On-Labs step-by-step .....	5
Abstract and Learning Objectives .....	6
Requirements .....	7
Naming Conventions:.....	7
Azure Subscription:.....	7
Other requirements: .....	7
Alternative Approach: .....	7
Final Remarks:.....	8
Abstract and Learning Objectives .....	12
Hands-On-Lab Scenario .....	13
Requirements .....	13
Naming Conventions:.....	13
Azure Subscription:.....	13
Other requirements: .....	13
Alternative Approach: .....	14
Final Remarks:.....	14
Lab 4: Containerizing an ASP.NET web application with Docker .....	15
What you will learn .....	15
Time Estimate .....	15
Task 1: Installing Docker for Windows on the lab-jumpVM.....	15
Task 2: Operating Docker from the command line .....	22
Task 3: Containerize your dotnet application using Visual Studio and Docker .....	27
Task 4: Deploying Containers to Azure Container Registry and Azure Container Instance .....	36
Task 5: Pushing a Docker image into an Azure Container Registry.....	42
Task 6: Running an Azure Container Instance from a Docker image in Azure Container Registry ...	44
Task 7: Deploy a Container using Azure Web Apps .....	48
Summary .....	53



# Migrating a dotnetcore 2-tiered application to Azure using different architectures and DevOps best practices

## - Hands-On-Labs step-by-step

You are part of an organization that is running a dotnetcore e-commerce platform application, using Windows Server infrastructure on-premises today, comprising a WebVM running Windows Server 2012 R2 with Internet Information Server (IIS) and a 2<sup>nd</sup> SQLVM running Windows Server 2012 R2 and SQL Server 2014.

The business has approved a migration of this business-critical workload to Azure, and you are nominated as the cloud solution architect for this project. No decision has been made yet on what the final architecture should or will look like. Your first task is building a Proof-of-Concept in your Azure environment, to test out the different architectures possible:

- Infrastructure as a Service (IAAS)
- Platform as a Service (PAAS)
- Containers as a Service (CaaS)

At the same time, your CIO wants to make use of this project to switch from a more traditional mode of operations, with barriers between IT sysadmin teams and Developer teams, to a DevOps way of working. Therefore, you are tasked to explore Azure DevOps and determine where CI/CD Pipelines can assist in optimizing the deployment and running operations of this e-commerce platform, especially when deploying updates to the application.

As you are new to the continuous changes in Azure, you want to make sure this process goes as smooth as possible, starting from the assessment to migration to day-to-day operations.

## Abstract and Learning Objectives

This workshop enables anyone to learn, understand and build a Proof of Concept, in performing a multi-tiered .Net Core web application using Microsoft SQL Server database, platform migration to Azure public cloud, leveraging on different Azure Infrastructure as a Service, Azure Platform as a Service (PaaS) and Azure Container offerings like Azure Container Instance (ACI) and Azure Kubernetes Services (AKS).

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 learn about the importance of performing proper assessments, and what tools Microsoft offers to help in this migration preparation phase. Once the application has been deployed on Azure Virtual Machines, students learn about Microsoft SQL database migration to SQL Azure PaaS, as well as deploying and migrating web applications to Azure Web Apps.

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

In the last part of the workshop, students get introduced to Azure DevOps, the new Microsoft Application Lifecycle environment, helping in building a CI/CD Pipeline to publish workloads using the DevOps principals and concepts, showing the integration with the rest of the already touched on Azure services like Azure Web Apps and Azure Kubernetes Services (AKS), closing the workshop with a module on overall Azure monitoring and operations and what tools Azure has available to assist your IT teams in this challenge.

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 2019;
- Publishing a .NET Core e-commerce application to an Azure Web Virtual Machine and SQL DB Virtual Machine;
- Performing a proper assessment of the as-is Web and SQL infrastructure using Microsoft Assessment Tools;
- Migrating a SQL 2014 database to Azure SQL PaaS (Lift & Shift);
- Migrating a .NET Core web application to Azure Web Apps (Lift & Shift);
- Containerizing a .NET Core web application using Docker, and pushing to Azure Container Registry (ACR);
- Running Azure Container Instance (ACI) and WebApp for Containers;
- Deploy and run Azure Kubernetes Services (AKS);

- Deploying Azure DevOps and building a CI/CD Pipeline for the subject e-commerce application;
- Managing and Monitoring Azure Kubernetes Services (AKS);

## 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) In one of the Azure Container Services tasks, you are required to create an Azure AD Service Principal, which 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.
- b) 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.
- c) 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); this could also be Visual Studio 2019 community edition - latest version
- Docker for Windows (updated to latest version)
- Azure CLI 2.0 (updated to latest version)
- Kubernetes CLI (updated to latest version)
- SimplCommerce Open Source e-commerce platform example  
(<http://www.simplcommerce.com>)

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.



# 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

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2018 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/Usage/General.aspx> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

# Contents

Abstract and Learning Objectives .....	12
Hands-On-Lab Scenario .....	13
Requirements .....	13
Naming Conventions:.....	13
Azure Subscription:.....	13
Other requirements: .....	13
Alternative Approach: .....	14
Final Remarks:.....	14
Lab 4: Containerizing an ASP.NET web application with Docker.....	15
What you will learn .....	15
Time Estimate .....	15
Task 1: Installing Docker for Windows on the lab-jumpVM.....	15
Task 2: Operating Docker from the command line .....	22
Task 3: Containerize your ASP.NET application using Visual Studio and Docker .....	27
Summary.....	35

# 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

- d) Azure Trial subscriptions won't work
- e) In one of the Azure Container Services tasks, you are required to create an Azure AD Service Principal, which 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.
- f) 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.
- g) 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 dotnetcore code that has been used in former lab, using Visual Studio.

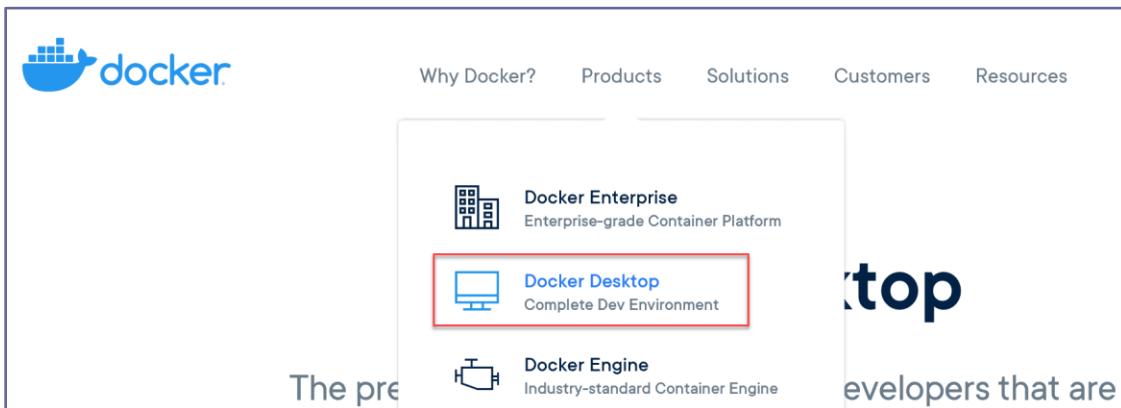
In the next task, you learn about Azure Container Registry (ACR) and how to publish your new Docker container in there, as well as using this as a source for Azure Container Instance (ACI) and running your web application. We will also touch on deploying an running Azure WebApps for Containers.

### Time Estimate

This lab duration is estimated 75 min.

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

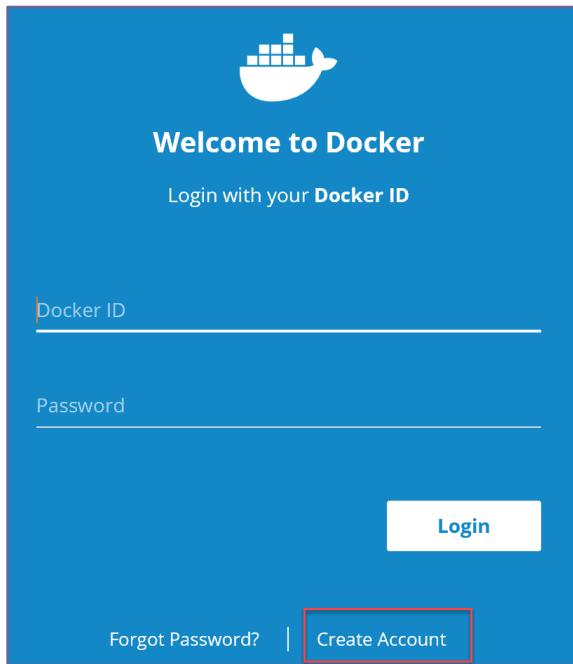
1. If not logged on anymore to the lab-jumpVM, open an RDP session to this Virtual Machine, using labadmin / [L@BadminPa55w.rd](http://L@BadminPa55w.rd) as credentials.
2. Connect to the Docker website <http://www.docker.com>; Here, Click Products, and select Docker Desktop.



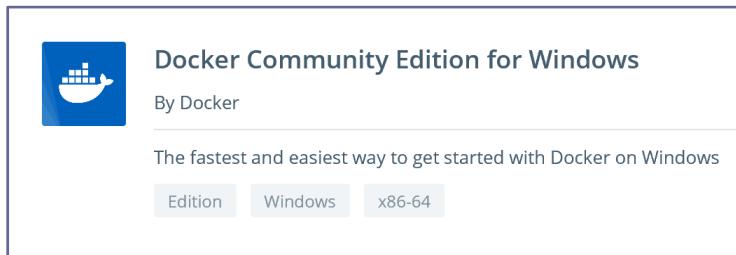
3. Click “Download for Windows”.
4. This redirects you to the Docker Store. (the direct link at the time of writing this lab guide is <https://store.docker.com/editions/community/docker-ce-desktop-windows>)
5. Before you can download the source install files, you need to create a **Docker Login**. Since you will use this account later on to connect to the Docker Hub, make sure you enter correct

details here.

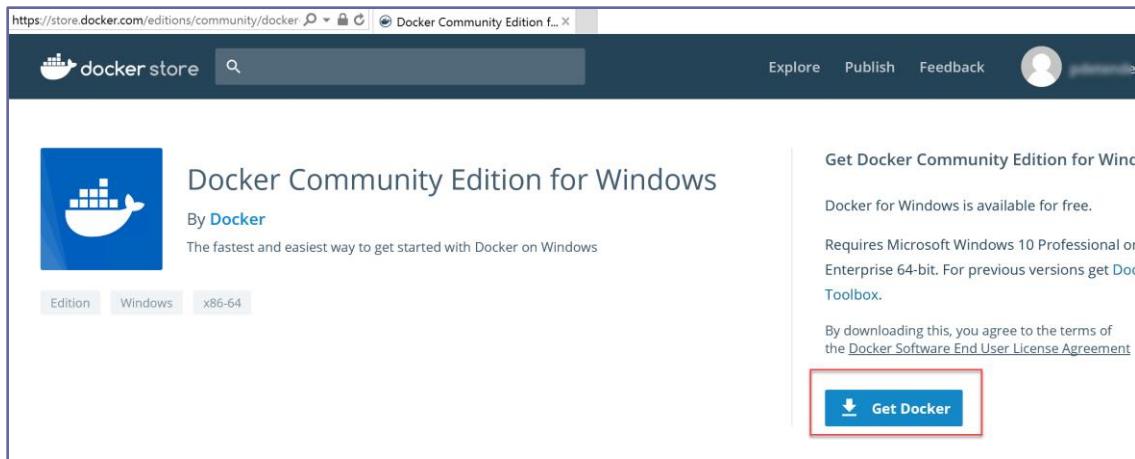
6. Click the **Login to Portal** button; in the login portal, choose Create Account.



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.



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



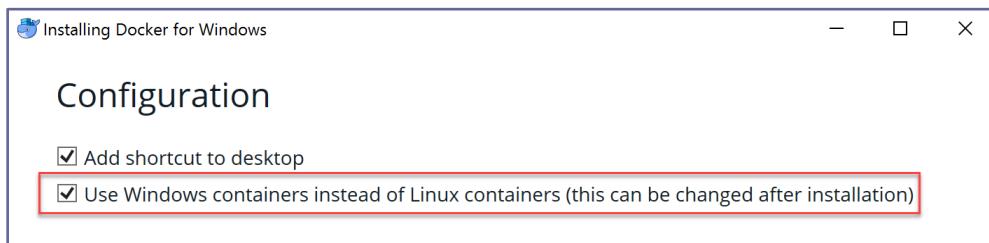
11. **Save** the install files to the local machine's Downloads folder; wait for the download to complete, and run the actual install process.



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

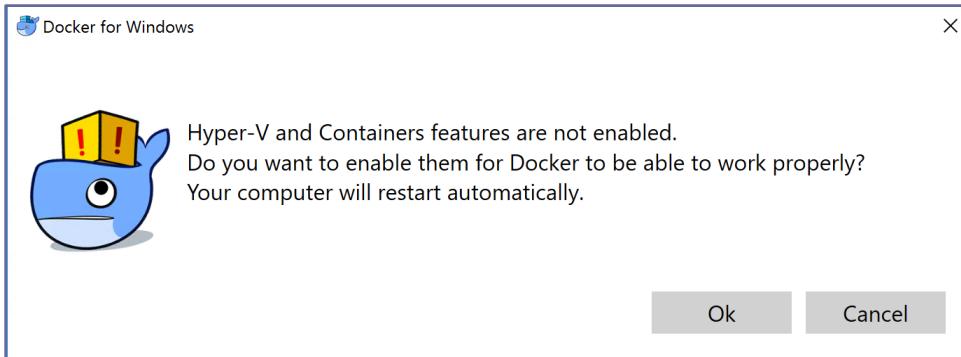


13. Wait for the background download to complete, until you see the **Configuration** step of the install wizard. **Here, select "Use Windows containers instead of Linux containers".** Press OK to continue.

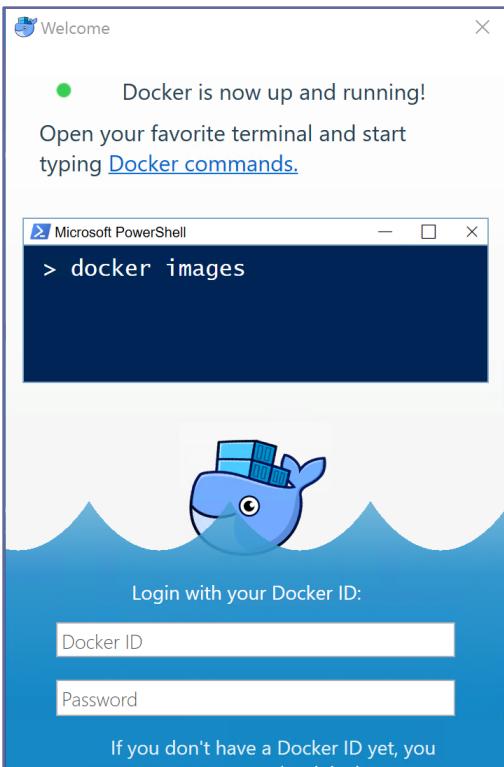


14. The Docker install will update the install packages and run the actual tool installation. Wait for this step to complete.

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



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.



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
```

```
Command Prompt
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\PeterDeTender>docker info
Client:
  Debug Mode: false

Server:
  Containers: 35
    Running: 0
    Paused: 0
    Stopped: 35
  Images: 26
  Server Version: 19.03.1
  Storage Driver: overlay2
    Backing Filesystem: extfs
    Supports d_type: true
    Native Overlay Diff: true
  Logging Driver: json-file
  Cgroup Driver: cgroupfs
  Plugins:
    Volume: local
    Network: bridge host ipvlan macvlan null overlay
    Log: awslogs fluentd gcplogs gelf journalctl json-file local logentries splunk syslog
  Swarm: inactive
  Runtimes: runc
  Default Runtime: runc
  Init Binary: docker-init
  containerd version: 894b81a4b802e4eb2a91d1ce216b8817763c29fb
  runc version: 425e105d5a03fabd737a126ad93d62a9eeede87f
  init version: fec3683
```

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

```
Command Prompt
C:\Users\PeterDeTender>docker version
Client: Docker Engine - Community
  Version:           19.03.1
  API version:        1.40
  Go version:         go1.12.5
  Git commit:        74b1e89
  Built:              Thu Jul 25 21:17:08 2019
  OS/Arch:            windows/amd64
  Experimental:       false

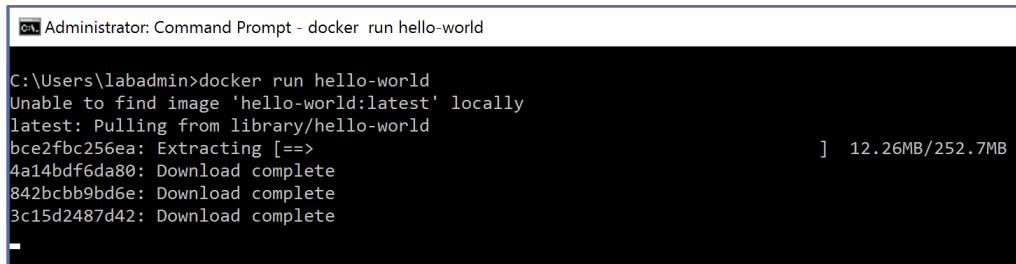
Server: Docker Engine - Community
  Engine:
    Version:           19.03.1
    API version:        1.40 (minimum version 1.12)
    Go version:         go1.12.5
    Git commit:        74b1e89
    Built:              Thu Jul 25 21:17:52 2019
    OS/Arch:            linux/amd64
    Experimental:       false
  containerd:
    Version:           v1.2.6
    GitCommit:          894b81a4b802e4eb2a91d1ce216b8817763c29fb
  runc:
    Version:           1.0.0-rc8
    GitCommit:          425e105d5a03fabd737a126ad93d62a9eeede87f
  docker-init:
    Version:           0.18.0
    GitCommit:          fec3683
```

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

```
docker run hello-world
```

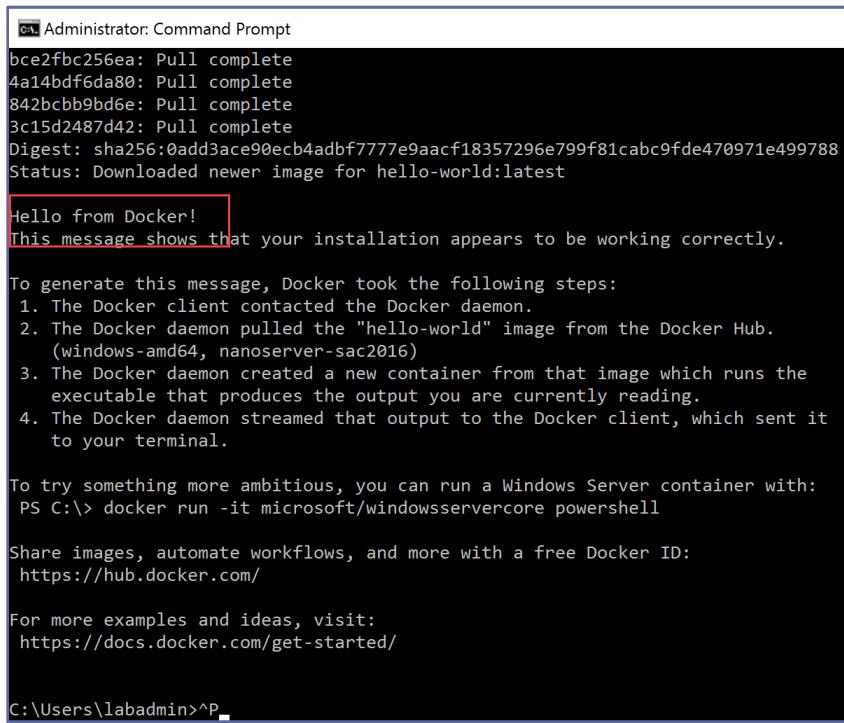
Some explanation what this command executes:

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



```
C:\> 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 [==>
4a14bdf6da80: Download complete
842bcb9bd6e: 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.:



```
C:\> Administrator: Command Prompt
bce2fbc256ea: Pull complete
4a14bdf6da80: Pull complete
842bcb9bd6e: Pull complete
3c15d2487d42: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabcfde470971e499788
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/
```

## 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
```

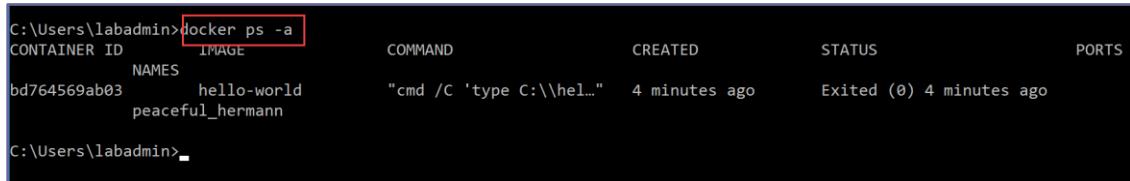


CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS

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

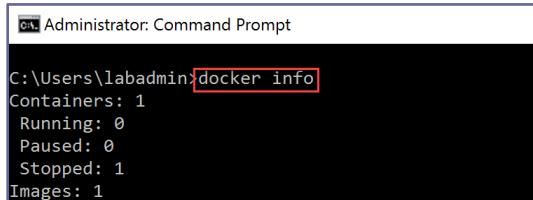
2. Like the previous command, you can add a "-a" parameter, showing you what containers were previously running

```
docker ps -a
```



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
bd764569ab03	hello-world	"cmd /C 'type C:\\hel..."	4 minutes ago	Exited (0) 4 minutes ago	
	peaceful_hermann				

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



```
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
```

```
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
```

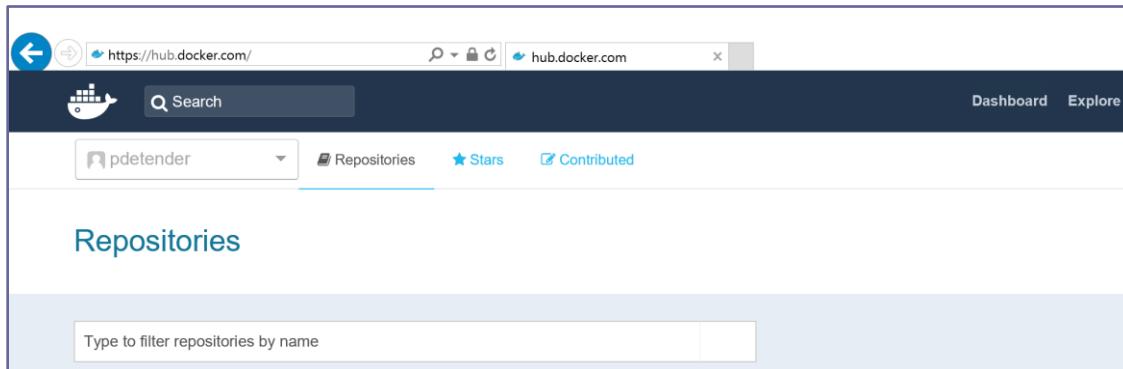
```
Administrator: Windows PowerShell
PS C:\> docker container ls -a --all
CONTAINER ID        NAMES              IMAGE               COMMAND             CREATED            STATUS              PORTS
bd764569ab03        hello-world       "cmd /c 'type C:\\\\hel..."   44 hours ago      Exited (0) 44 hours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

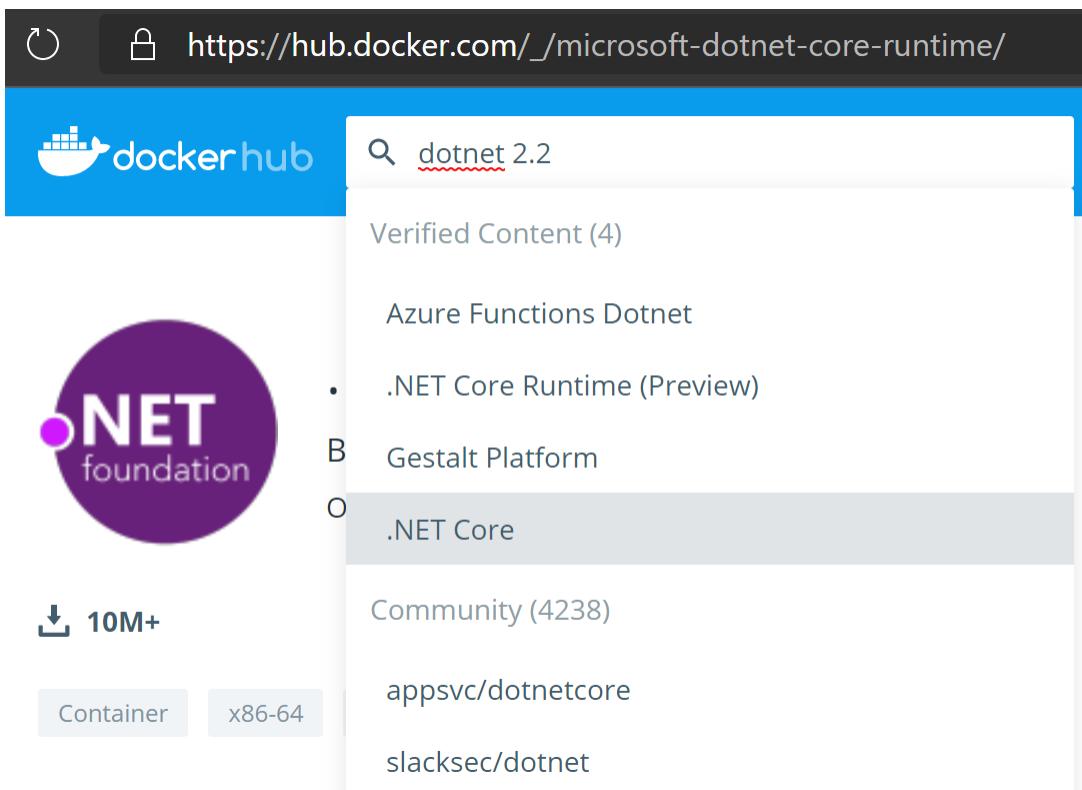
Similar to the hello-world container images that got pulled down before running, let's move on with exploring the different Windows OS-based container images in Docker Store, and pulling them to our

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 "dotnet 2.2"; this results in a list of repositories.



9. Select the ".NET Core" repository. Which shows you the different .NET Core-based Docker images, provided by Microsoft. Important difference to note is that the actual Container is NOT stored in the Docker Hub, but stored inside Microsoft's own Repository for containers

(mcr.microsoft.com) – this is useful to know that this container is coming from a legitimate source, and is not a customized images by somebody else, publishing it on Docker Hub.

The screenshot shows the Docker Hub search results for 'dotnet 2.2'. At the top, there's a blue header bar with the Docker Hub logo and a search bar containing 'dotnet 2.2'. Below the header, the search results are displayed. The first result is for '.NET Core Runtime' by Microsoft, which is described as 'Official images for the .NET Core runtime'. It has a purple circular icon with the .NET logo. Below the icon, there's a download count of '10M+'. Underneath the main title, there are three tabs: 'Container', 'x86-64', and 'Base Images'. Further down the page, there are sections for 'DESCRIPTION' and 'REVIEWS'. A large section titled 'Featured Tags' lists '2.2 (Current)' and '2.1 (LTS)'. For each tag, there's a link to pull the image: 'docker pull mcr.microsoft.com/dotnet/core/runtime:2.2' for the current tag and 'docker pull mcr.microsoft.com/dotnet/core/runtime:2.1' for the LTS tag.

Switch back to your Command Prompt, and run the following **Docker command**:

```
docker pull mcr.microsoft.com/dotnet/core/runtime:2.2
```

```
cmd Command Prompt - docker pull mcr.microsoft.com/dotnet/core/runtime:2.2

C:\Users\PeterDeTender>docker pull mcr.microsoft.com/dotnet/core/runtime:2.2
2.2: Pulling from dotnet/core/runtime
9fc222b64b0a: Downloading [=====>] 8.946MB/22.52MB
f033c4f65cdb: Downloading [=====>] 4.504MB/17.69MB
7405f9e6a7ae: Download complete
f4b7599512c4: Waiting
-
```

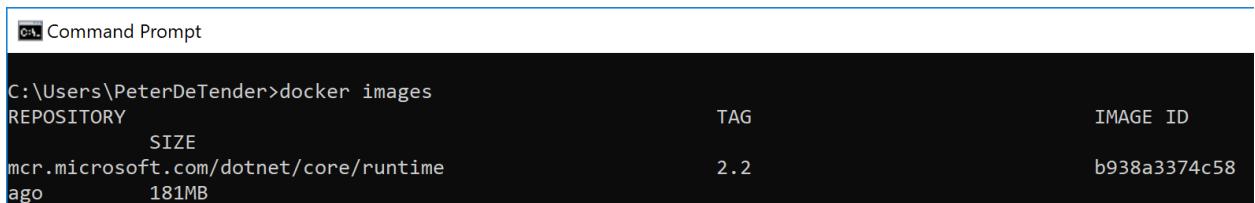
10. Notice this image is based on different "layers" (the 9fc222b64..., f033c4...,7405f9...)
11. **Wait for the image to finish downloading** and **extracting**. Depending on your internet connection speed, this might take several minutes. From a container perspective, this is based on the Microsoft/dotnet core, having a **tag** pointing to the latest edition (the 2.2-part in the name). The advantage for developers and the running applications, is that in this way, you can guarantee your application will always be based on the same dotnetcore version.

## Task 3: Containerize your dotnet application using Visual Studio and Docker

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

```
docker images
```

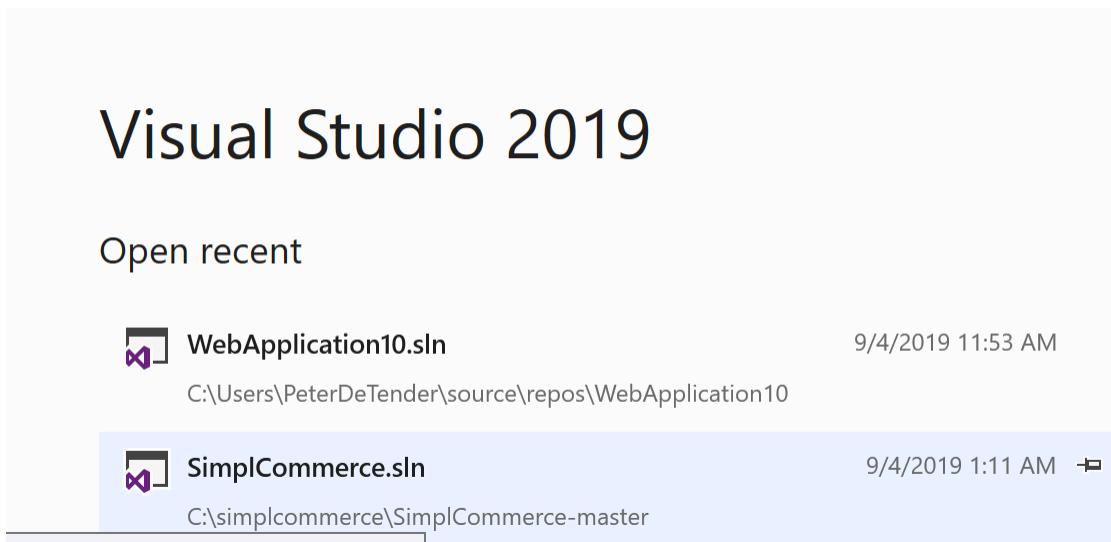
and validate the Docker image for `microsoft/dotnet/core/runtime` is showing up.



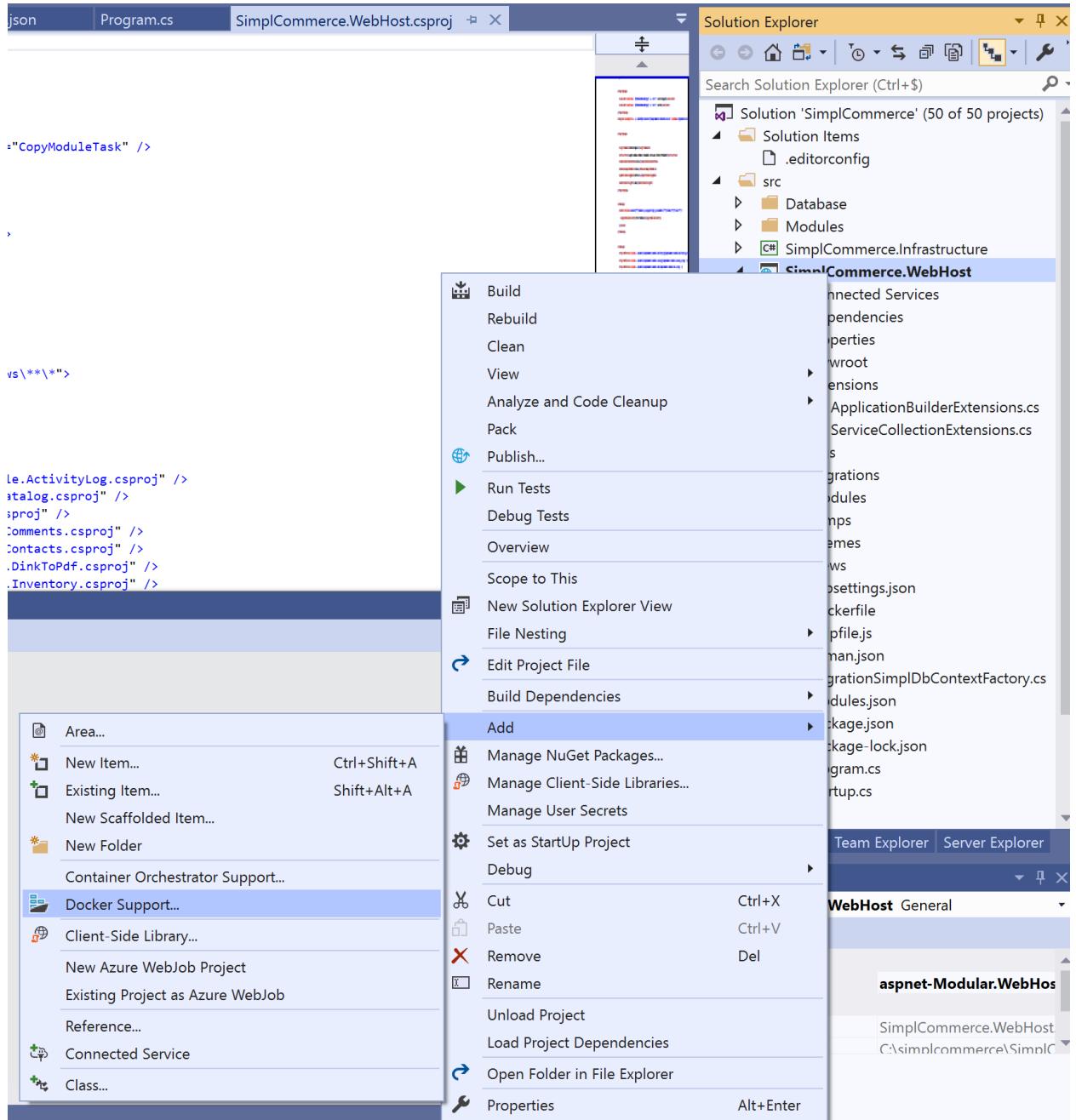
```
PS C:\Users\PeterDeTender>docker images
REPOSITORY          TAG      IMAGE ID
mcr.microsoft.com/dotnet/core/runtime   2.2      b938a3374c58
ago                181MB
```

While you can configure a Docker Container in a full manual way, let me walk you through the Visual Studio-based approach, as yes, thanks to having installed Docker Desktop on the same host as Visual Studio, it's providing a really nice integration...

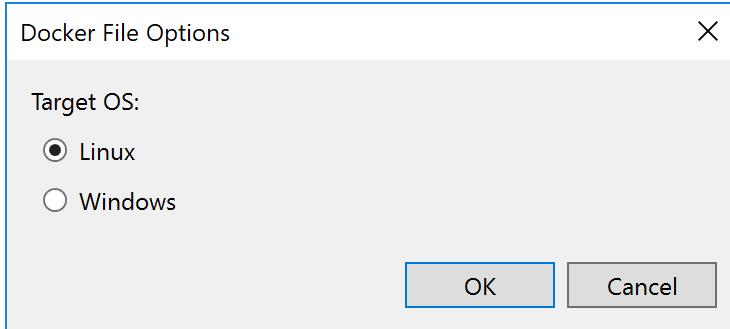
2. From Visual Studio, Open the recent `SimlCommerce.sln` project, pulling up the source code from the web application



3. Once loaded, select the `SimlCommerce.webhost` project, right-click, select Add / Docker Support



4. **Next**, you are prompted for what Container Operating System you want to use, Linux or Windows. For Dotnetcore, you could use either of them, but in this scenario, select **Linux**.



- Once you click **OK**, it will generate a new Dockerfile for you. A Dockerfile is like a configuration file, instructing Docker what to do for building a container.

(Note: if you get a prompt telling you there is already a Dockerfile available, DO NOT OVERWRITE the existing one. From the File Explorer, rename the Dockerfile to Dockerfile.org, and repeat the previous steps)

- Thanks to the Docker integration, it will “read out” the content of the source code, and based on that decided what container images are required. Notice it recommends using the `aspnet:2.2` container image here

```

1 FROM mcr.microsoft.com/dotnet/core/aspnet:2.2-stretch-slim AS base
2 WORKDIR /app
3 EXPOSE 80
4
5 FROM mcr.microsoft.com/dotnet/core/sdk:2.2-stretch AS build
6 WORKDIR /src
7 COPY ["src/SimplCommerce.WebHost/SimplCommerce.WebHost.csproj", "src/SimplCommerce.WebHost/"]
8 COPY ["src/Modules/SimplCommerce.Module.DinkToPdf/SimplCommerce.Module.DinkToPdf.csproj", "src/Modules/SimplCommerce.Module.DinkToPdf/"]
9 COPY ["src/SimplCommerce.Infrastructure/SimplCommerce.Infrastructure.csproj", "src/SimplCommerce.Infrastructure/"]
10 COPY ["src/Modules/SimplCommerce.Module.Core/SimplCommerce.Module.Core.csproj", "src/Modules/SimplCommerce.Module.Core/"]
11 COPY ["src/Modules/SimplCommerce.Module.Reviews/SimplCommerce.Module.Reviews.csproj", "src/Modules/SimplCommerce.Module.Reviews/"]
12 COPY ["src/Modules/SimplCommerce.Module.Orders/SimplCommerce.Module.Orders.csproj", "src/Modules/SimplCommerce.Module.Orders/"]
13 COPY ["src/Modules/SimplCommerce.Module.ShippingPrices/SimplCommerce.Module.ShippingPrices.csproj", "src/Modules/SimplCommerce.Module.ShippingPrices/"]
14 COPY ["src/Modules/SimplCommerce.Module.Catalog/SimplCommerce.Module.Catalog.csproj", "src/Modules/SimplCommerce.Module.Catalog/"]
15 COPY ["src/Modules/SimplCommerce.Module.Tax/SimplCommerce.Module.Tax.csproj", "src/Modules/SimplCommerce.Module.Tax/"]
16 COPY ["src/Modules/SimplCommerce.Module.Shipping/SimplCommerce.Module.Shipping.csproj", "src/Modules/SimplCommerce.Module.Shipping/"]
17 COPY ["src/Modules/SimplCommerce.Module.ShoppingCart/SimplCommerce.Module.ShoppingCart.csproj", "src/Modules/SimplCommerce.Module.ShoppingCart/"]
18 COPY ["src/Modules/SimplCommerce.Module.Pricing/SimplCommerce.Module.Pricing.csproj", "src/Modules/SimplCommerce.Module.Pricing/"]
19 COPY ["src/Modules/SimplCommerce.Module.ProductRecentlyViewed/SimplCommerce.Module.ProductRecentlyViewed.csproj", "src/Modules/SimplCommerce.Module.ProductRecentlyViewed/"]
20 COPY ["src/Modules/SimplCommerce.Module.Search/SimplCommerce.Module.Search.csproj", "src/Modules/SimplCommerce.Module.Search/"]
21 COPY ["src/Modules/SimplCommerce.Module.ShippingTableRate/SimplCommerce.Module.ShippingTableRate.csproj", "src/Modules/SimplCommerce.Module.ShippingTableRate/"]
22 COPY ["src/Modules/SimplCommerce.Module.Payments/SimplCommerce.Module.Payments.csproj", "src/Modules/SimplCommerce.Module.Payments/"]
23 COPY ["src/Modules/SimplCommerce.Module.Comments/SimplCommerce.Module.Comments.csproj", "src/Modules/SimplCommerce.Module.Comments/"]
24 COPY ["src/Modules/SimplCommerce.Module.ShippingFree/SimplCommerce.Module.ShippingFree.csproj", "src/Modules/SimplCommerce.Module.ShippingFree/"]
25 COPY ["src/Modules/SimplCommerce.Module.News/SimplCommerce.Module.News.csproj", "src/Modules/SimplCommerce.Module.News/"]
26 COPY ["src/Modules/SimplCommerce.Module.PaymentCashfree/SimplCommerce.Module.PaymentCashfree.csproj", "src/Modules/SimplCommerce.Module.PaymentCashfree/"]
27 COPY ["src/Modules/SimplCommerce.Module.PaymentNganLuong/SimplCommerce.Module.PaymentNganLuong.csproj", "src/Modules/SimplCommerce.Module.PaymentNganLuong/"]
28 COPY ["src/Modules/SimplCommerce.Module.EmailsenderSmtplib/SimplCommerce.Module.EmailSenderSmtplib.csproj", "src/Modules/SimplCommerce.Module.EmailSenderSmtplib/"]
29 COPY ["src/Modules/SimplCommerce.Module.SampleData/SimplCommerce.Module.SampleData.csproj", "src/Modules/SimplCommerce.Module.SampleData/"]
30 COPY ["src/Modules/SimplCommerce.Module.Wishlist/SimplCommerce.Module.Wishlist.csproj", "src/Modules/SimplCommerce.Module.Wishlist/"]
31 COPY ["src/Modules/SimplCommerce.Module.PaymentBraintree/SimplCommerce.Module.PaymentBraintree.csproj", "src/Modules/SimplCommerce.Module.PaymentBraintree/"]
32 COPY ["src/Modules/SimplCommerce.Module.Cms/SimplCommerce.Module.Cms.csproj", "src/Modules/SimplCommerce.Module.Cms/"]
33 COPY ["src/Modules/SimplCommerce.Module.PaymentStripe/SimplCommerce.Module.PaymentStripe.csproj", "src/Modules/SimplCommerce.Module.PaymentStripe/"]
34 COPY ["src/Modules/SimplCommerce.Module.ProductComparison/SimplCommerce.Module.ProductComparison.csproj", "src/Modules/SimplCommerce.Module.ProductComparison/"]
35 COPY ["src/Modules/SimplCommerce.Module.StorageLocal/SimplCommerce.Module.StorageLocal.csproj", "src/Modules/SimplCommerce.Module.StorageLocal/"]

```

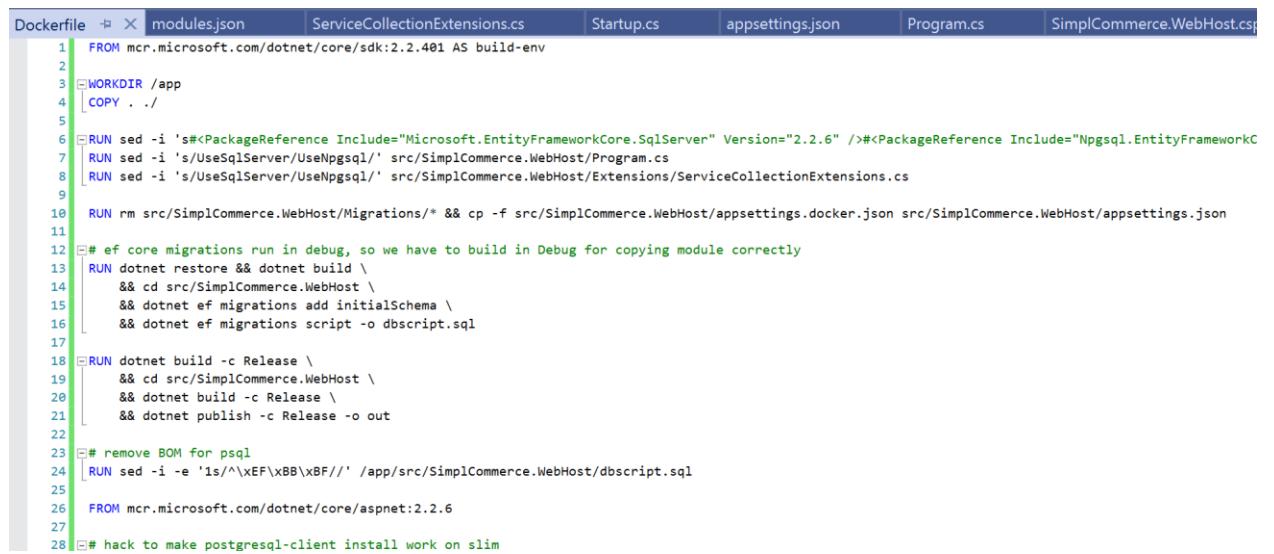
- Followed by a full list of copy statements of each and every VS Modle folder the application is using.
- Next, it will hand over this intermediate image to the next process in the Dockerfile, eventually resulting in a final containerized application image.

Note: while this process might work fine, I found out that the way how Visual Studio is building up this Dockerfile by itself, is different depending on Visual Studio versions (2017 or 2019, as well as Docker version installed).

Therefore, we will use a different approach, and reusing the Dockerfile we renamed earlier to Dockerfile.org.

9. Open the Dockerfile.org file (in Visual Studio or Notepad), select all, and replace the content of the Dockerfile that got automatically generated by Visual Studio.

the file should look like this:



```
Dockerfile  X  modules.json  ServiceCollectionExtensions.cs  Startup.cs  appsettings.json  Program.cs  SimplCommerce.WebHost.cs
1 FROM mcr.microsoft.com/dotnet/core/sdk:2.2.401 AS build-env
2
3 WORKDIR /app
4 COPY ./
5
6 RUN sed -i 's#<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.2.6" />#<PackageReference Include="Npgsql.EntityFrameworkCore
7 RUN sed -i 's/UseSqlServer/UseNpgsql/' src/SimplCommerce.WebHost/Program.cs
8 RUN sed -i 's/UseSqlServer/UseNpgsql/' src/SimplCommerce.WebHost/Extensions/ServiceCollectionExtensions.cs
9
10 RUN rm src/SimplCommerce.WebHost/Migrations/* && cp -f src/SimplCommerce.WebHost/appsettings.docker.json src/SimplCommerce.WebHost/appsettings.json
11
12 # ef core migrations run in debug, so we have to build in Debug for copying module correctly
13 RUN dotnet restore && dotnet build \
14     && cd src/SimplCommerce.WebHost \
15     && dotnet ef migrations add initialSchema \
16     && dotnet ef migrations script -o dbscript.sql
17
18 RUN dotnet build -c Release \
19     && cd src/SimplCommerce.WebHost \
20     && dotnet build -c Release \
21     && dotnet publish -c Release -o out
22
23 # remove BOM for psql
24 RUN sed -i -e '1s/^xEF\xBB\xBF//' /app/src/SimplCommerce.WebHost/dbscript.sql
25
26 FROM mcr.microsoft.com/dotnet/core/aspnet:2.2.6
27
28 # hack to make postgresql-client install work on slim
```

10. Next, start a debug process, by selecting Debug / Docker in the top menu



11. To follow what is happening in the back-end, check the Visual Studio Output window

The screenshot shows the Visual Studio Output window with the title 'Output' at the top. Below it, the text area displays the following Docker build logs:

```

Show output from: Build
1>e6cb98e32a52: Download complete
1>dcc190332fff6: Verifying Checksum
1>dcc190332fff6: Download complete
1>9cc2ad81d40d: Verifying Checksum
1>9cc2ad81d40d: Download complete
1>9cc2ad81d40d: Pull complete
1>e6cb98e32a52: Pull complete
1>ae1bbd879bad: Pull complete
1>42cfa3699b05: Download complete
1>42cfa3699b05: Pull complete
1>dcc190332fff6: Pull complete
1>b49ae448d777: Verifying Checksum
1>b49ae448d777: Download complete
1>b49ae448d777: Pull complete
1>67e832fd5d6a: Verifying Checksum
1>67e832fd5d6a: Download complete
1>67e832fd5d6a: Pull complete
1>Digest: sha256:e1b19d391986f21b177c1ba47f8a8519bf5cff66a9459d04f3aea23e1a7ec30f
1>Status: Downloaded newer image for mcr.microsoft.com/dotnet/core/sdk:2.2.401
1> ---> 08657316a4cd
1>Step 2/12 : WORKDIR /app
1> ---> Running in b3dc7a3da92d
1>Removing intermediate container b3dc7a3da92d
1>Step 3/12 : COPY . .
1> ---> 3360d22609ce

```

At the bottom of the window, there are tabs for 'Package Manager Console', 'Error List', 'Web Publish Activity', and 'Output'.

12. **It starts** with downloading the different references images from the Microsoft repo, followed by copying the source code into the container image.
13. **In short**, it is going through the following steps:
  - validate Docker Desktop is running (we did that manually before)
  - verify the host's OS matches the target OS (remember Docker Desktop on Windows supports both Windows and Linux based containers)
  - Checking if the required Docker images are available on your machine
  - run the **Docker build** command, which is the actual **containerization part** of the process, pushing the source code into a container image.

The screenshot shows the Visual Studio Output window with the title 'Output' at the top. Below it, the text area displays the following Docker build logs with step notation:

```

Show output from: Build
1>Step 1/5 : FROM mcr.microsoft.com/dotnet/core/aspnet:2.2-stretch-slim AS base
1> ---> Using cache
1> ---> 647bf9fb6a4
1>Step 3/5 : EXPOSE 80
1> ---> Using cache
1> ---> 282fbabaa342
1>Step 4/5 : LABEL com.microsoft.created-by=visual-studio
1> ---> Using cache
1> ---> 9a7076005676
1>Step 5/5 : LABEL com.microsoft.visual-studio.project-name=SimplCommerce.WebHost
1> ---> Using cache
1> ---> 9776045e0cac

```

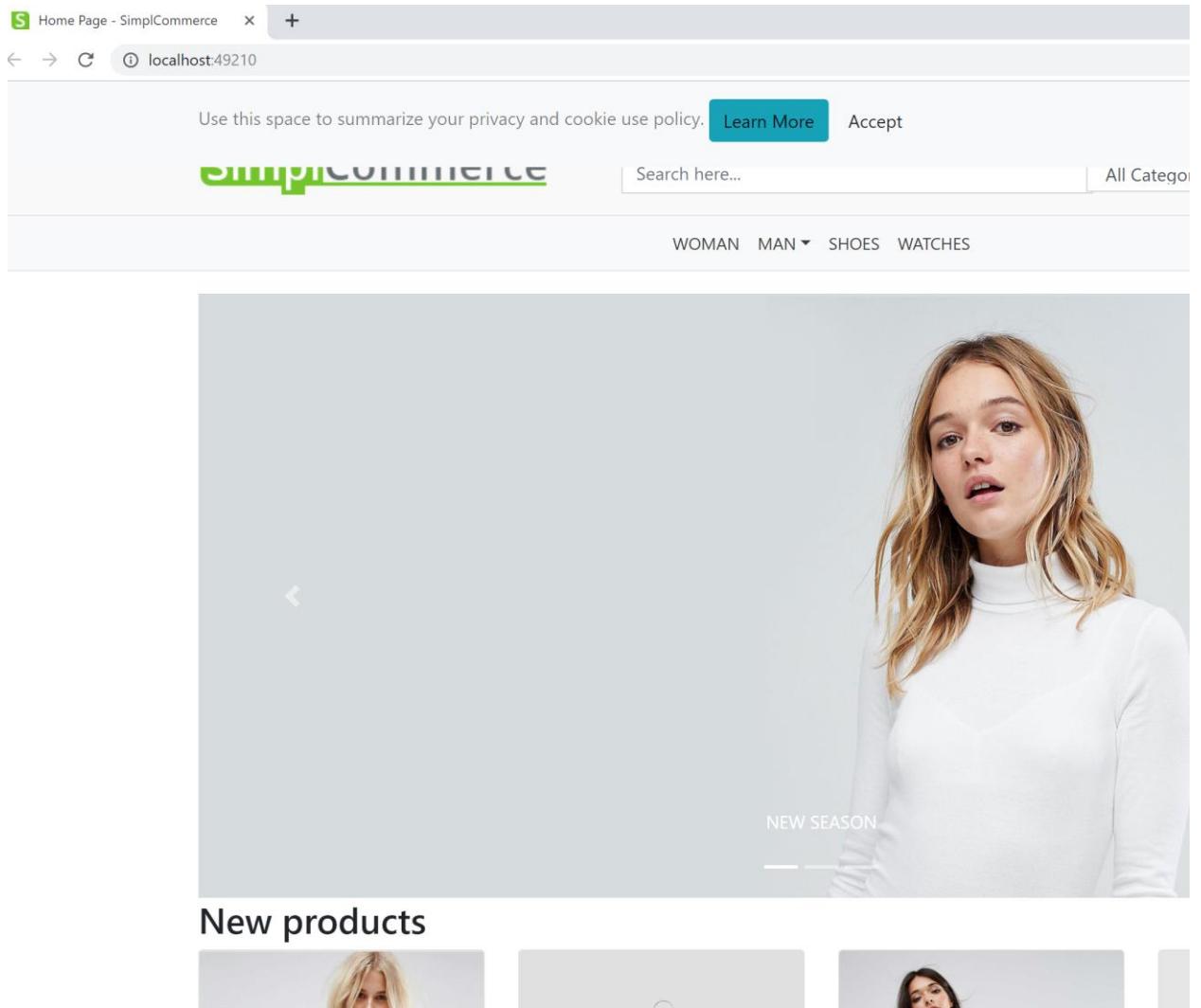
At the bottom of the window, there are tabs for 'Package Manager Console', 'Error List', 'Web Publish Activity', and 'Output'.

14. **Some other output** that is useful to know, is the notation Step X/5. Remember that the Dockerfile is providing the sequence on how the container gets build up. The different **FROM** statements in the Dockerfile reflect a step in the process.

**Note:** including the download of the different source image layers, this process might take about 10 minutes or longer, depending on your internet speed. Wait for the process to complete

successfully.

15. When the build process is completed, Visual Studio will automatically open your browser and run the application.



16. Notice that the application runs as "localhost", port 49210 (you might remember that the initial web application test in the browser in the previous lab, was using port 49209).
17. Validate the image is built successfully by running the following command in command prompt or PowerShell:

```
docker images
```

and notice the most recent container image is in the list, having a tag of "empty"

```
C:\Users\PeterDeTender>docker images
REPOSITORY          TAG      IMAGE ID
<none>              <none>   4d7985aaefca
s ago               2.3GB
```

18. Next, run the following command, to get details related to the actual running container:

```
docker container ls
```

```
C:\Users\PeterDeTender>docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
 NAMES
56b47396a1f1      simplcommercewebhost:dev "tail -f /dev/null"   11 minutes ago   Up 11 minutes   0.0.0.0:492
10->80/tcp        adoring_johnson

C:\Users\PeterDeTender>
```

19. You can read more details about your running Docker Container, useful for troubleshooting, by executing the following command:

```
docker inspect 56b473 (replace this number with the container ID in you environment)
```

```
C:\Users\PeterDeTender>docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
 NAMES
56b47396a1f1      simplcommercewebhost:dev "tail -f /dev/null"   11 minutes ago   Up 11 minutes   0.0.0.0:492
10->80/tcp        adoring_johnson

C:\Users\PeterDeTender>docker inspect 56b473
[{"Id": "56b47396a1f150f93ab9988a29b04fd441e1033a4459d98e58969324961dd03d", "Created": "2019-09-04T13:11:58.1484079Z", "Path": "tail", "Args": ["-f", "/dev/null"], "State": {"Status": "running", "Running": true, "Paused": false, "Restarting": false, "OOMKilled": false, "Dead": false, "Pid": 3591, "ExitCode": 0, "Error": "", "StartedAt": "2019-09-04T13:12:00.5090727Z", "FinishedAt": "2001-01-01T00:00:00Z"}}
```

20. The output of the last command, provides a JSON file with several interesting details:

for example: What is the state of my container

```
"State": {  
    "Status": "running",  
    "Running": true,  
    "Paused": false,  
    "Restarting": false,  
    "OOMKilled": false,  
    "Dead": false,  
    "Pid": 3591,  
    "ExitCode": 0,  
    "Error": "",  
    "StartedAt": "2019-09-04T13:12:00.5090727Z",  
    "FinishedAt": "0001-01-01T00:00:00Z"  
,
```

which tells me that it is running, it has not restarted yet, and I can also see the actual Start Time and Date.

what is the application port I'm using

```
"NetworkMode": "default",  
"PortBindings": {  
    "80/tcp": [  
        {  
            "HostIp": "",  
            "HostPort": "49210"  
        }  
    ]  
},
```

Showing us that we are using port 49210 on the host (your Docker host), where the internal application port is 80 – so from a container perspective, the app gets published over HTTP. Where port 80 gets mapped to 49210 on host level to avoid any other port conflicts.

what application configuration is being used?

```
PS Select Command Prompt
[{"Config": {
    "Hostname": "56b47396a1f1",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "ExposedPorts": {
        "80/tcp": {}
    },
    "Tty": true,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
        "ASPNETCORE_ENVIRONMENT=Development",
        "NUGET_PACKAGES=/root/.nuget/fallbackpackages2",
        "NUGET_FALLBACK_PACKAGES=/root/.nuget/fallbackpackages;/root/.nuget/fallbackpackages2",
        "DOTNET_USE_POLLING_FILE_WATCHER=1",
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "ASPNETCORE_URLS=http://+:80",
        "DOTNET_RUNNING_IN_CONTAINER=true",
        "ASPNETCORE_VERSION=2.2.6"
    ],
    "Cmd": [
        "-f",
        "/dev/null"
    ],
    "Image": "simplcommercewebhost:dev",
    "Volumes": null
}}
```

Telling me the hostname of the container, which is basically the **Container ID**, but also recognizing Environmental Variables, specifically for this container. (which means, you can push those whenever the container is starting up, like identifying this is a Dev environment, Production environment, Connection Strings,...)

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

Docker stop 56b4      (where 56b4 should be replaced with the ID of your container)

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

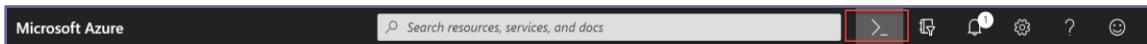
```
PS Command Prompt
C:\Users\PeterDeTender>docker container ls
CONTAINER ID        IMAGE               COMMAND       CREATED          STATUS          PORTS
NAMES

C:\Users\PeterDeTender>
```

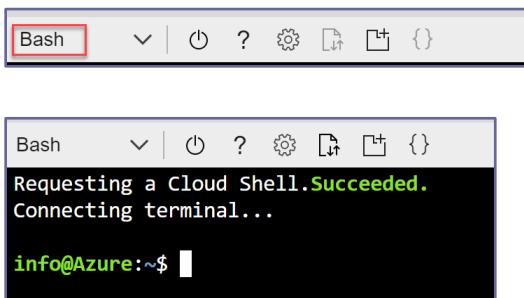
## Task 4: Deploying Containers to Azure Container Registry and Azure Container Instance

As we have a successfully built Docker container out of the previous task, we can move on with the next step in the process, and migrating this container to Azure. Starting from pushing it into Azure Container Registry (ACR), and running it as a Container Instance (ACI).

1. Log on to the Azure Portal, <http://portal.azure.com>, with your Azure admin credentials. From here, Open Cloud Shell

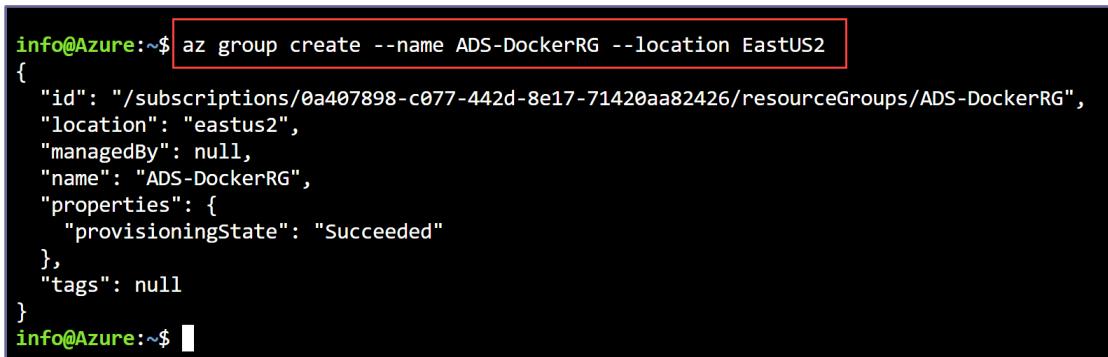


2. Follow the configuration steps if this is the first time you launched Cloud Shell. In the Environment, make sure you choose **Bash**.



3. Execute the following Azure CLI commands, to **create a new Azure Resource Group**:

```
az group create --name [SUFFIX]-dockerRG --location <Azure Region  
of choice>
```



4. Followed by another Azure CLI command to **create the Azure Container Registry**:

```
az acr create --resource-group [Suffix]-dockerRG --name
```

```
[SUFFIX]ACR --sku Basic --admin-enabled true
```

```
info@Azure:~$ az acr create --resource-group ADS-dockerRG --name ADSACR --sku Basic --admin-enabled true
{
  "adminUserEnabled": true,
  "creationDate": "2018-09-30T21:20:24.207509+00:00",
  "id": "/subscriptions/0a407898-c077-442d-8e17-71428aa82426/resourceGroups/ADS-dockerRG/providers/Microsoft.ContainerRegistry/registries/ADSACR",
  "location": "eastus2",
  "loginServer": "adsacr.azurecr.io",
  "name": "ADSACR",
  "provisioningState": "Succeeded",
  "resourceGroup": "ADS-dockerRG",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
}
```

5. The next involves connecting to the Azure Container Registry we just created, and pushing our Docker image into it. This relies on the following command:

```
az acr login -name [SUFFIX]ACR -resource-group [SUFFIX]-dockerRG
```

```
Bash   ▾ | ⌁ ? ⚙ ⌂ ⌃ {}  
info@Azure:~$ az acr login --name adsacr --resource-group ads-dockerRG  
This command requires running the docker daemon, which is not supported in Azure Cloud Shell.  
info@Azure:~$
```

6. This means, we have to execute the remaining commands from our local lab-jumpVM, instead of the Azure Cloud Shell. We should install the Azure CLI for Windows using the following link: <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest>
7. From the appearing web page, scroll down to the Download MSI installer button, and click it.

## Install Azure CLI on Windows

09/09/2018 • 2 minutes to read • Contributors 

For Windows the Azure CLI is installed via an MSI, which gives you access to the Windows Command Prompt (CMD) or PowerShell. When installing for Windows (WSL), packages are available for your Linux distribution. See the [main install](#) supported package managers or how to install manually under WSL.

### Install or update

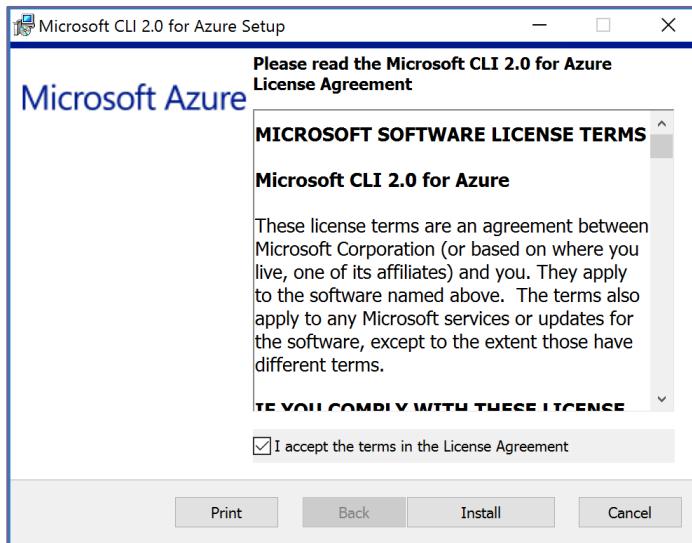
The MSI distributable is used for installing, updating, and uninstalling the `az` command-line interface on Windows.

[Download the MSI installer >](#)

- At the download prompt, choose **SAVE**; once the file is downloaded, select **RUN**



- The Microsoft CLI 2.0 for Azure Setup Installer launches



- Press the **Install** button to continue. Wait for the installation to **finish** successfully.
- To validate the Azure CLI is installed fine, **open a new PowerShell window**, and initiate the following command:

```
az
```

```
Administrator: Windows PowerShell
windows PowerShell
Copyright (C) 2016 Microsoft corporation. All rights reserved.

PS C:\users\labadmin> az
welcome to Azure CLI!
-----
Use `az -h` to see available commands or go to https://aka.ms/cli.

Telemetry
-----
The Azure CLI collects usage data in order to improve your experience.
The data is anonymous and does not include commandline argument values.
The data is collected by Microsoft.

You can change your telemetry settings with `az configure`.



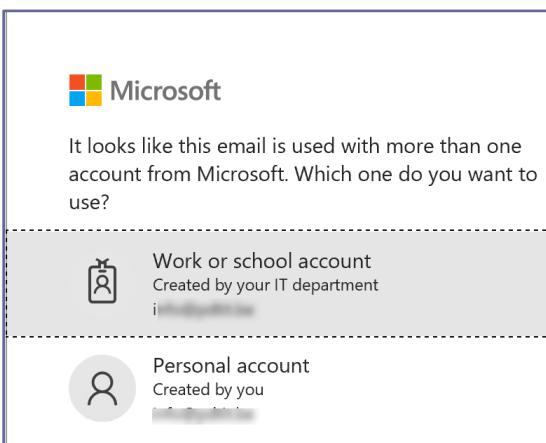
welcome to the cool new Azure CLI!
Use `az --version` to display the current version.
Here are the base commands:
```

12. This confirms Azure CLI 2.0 is running as expected. We can continue with our Azure Container Registry creation process. But first, we need to "authenticate" our session to Azure, by running the following command:

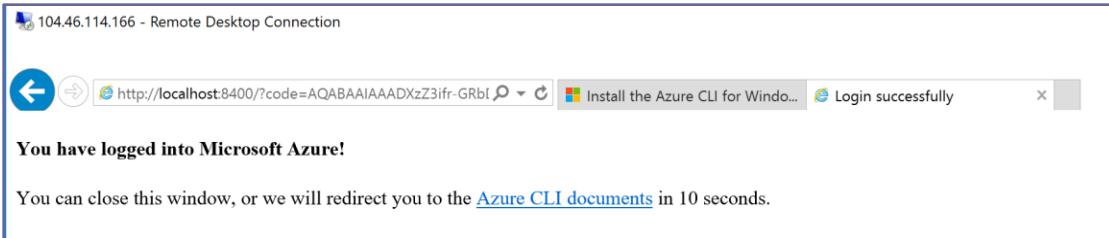
```
az login
```

```
Administrator: Windows PowerShell
PS C:\users\labadmin> az login
-
```

13. This opens your internet browsers, and prompts for your Azure admin credentials:



14. After successful login, the following information is displayed:



15. You can close the internet browser.
16. When you go back to the PowerShell window, it will show you the JSON output of your Azure subscription, related to this Azure admin user:

```
PS C:\Users\labadmin> az login
Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code"
you have logged in. Now let us find all the subscriptions to which you have access...
[ {
  "cloudName": "AzureCloud",
  "id": "0a407898-XXXX-XXXX-XXXX-XXXX26",
  "isDefault": true,
  "name": "Micr...p",
  "state": "Enabled",
  "tenantId": "7068...4bc",
  "user": {
    "name": "i...e",
    "type": "user"
  }
},
```

17. If you should have multiple Azure subscriptions linked to the same Azure admin credentials, run the following AZ CLI command to guarantee you are working in the correct subscription:

```
az account set --subscription "your subscription name here"
```

```
PS C:\Users\labadmin> az account set --subscription "Microsoft Azure ..."
PS C:\Users\labadmin>
```

18. Let's try to redo our Azure Container Registry process, by executing the following command:

```
az acr create --resource-group [SUFFIX]-DockerRG --name [SUFFIX]ACR --sku Basic --admin-enabled true
```

```

Administrator: Windows PowerShell
PS C:\Users\labadmin> az acr create --resource-group ADS-dockerrg --name ADSACR --sku Basic --admin-enabled true
{
  "adminUserEnabled": true,
  "creationDate": "2018-09-30T21:20:24.207509+00:00",
  "id": "/subscriptions/0a407898-c077-442d-8e17-71420aa82426/resourceGroups/ADS-dockerRG/providers/Microsoft.ContainerRe
gistry/registries/ADSAACR",
  "location": "eastus2",
  "loginServer": "adsacr.azurecr.io",
  "name": "ADSAACR",
  "provisioningState": "Succeeded",
  "resourceGroup": "ADS-dockerRG",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  },
  "status": null,
  "storageAccount": null,
  "tags": {},
  "type": "Microsoft.ContainerRegistry/registries"
}
PS C:\Users\labadmin>

```

19. While the JSON output is here, you can also validate from the Azure Portal

The screenshot shows the Azure Portal's Container registries page. At the top, there is a breadcrumb navigation: Home > Container registries. Below the header, there are buttons for Add, Edit columns, Refresh, and Assign tags. A callout box with the text 'Build, Run, Push and Patch containers in Azure with ACR Tasks' is overlaid on the page. The main area displays a table of container registries. The table has columns: NAME, TYPE, RESOURCE GROUP, LOCATION, and SUBSCRIPTION. One item is listed:

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
nopacr1	Container registry	noprg	Central US	007FFFLearning Labs

20. Next, we need to authenticate to the Azure Container Registry itself, using the following command:

```
az acr login --name <ACR-Name> --resource-group <RG Name>
```

```

PS C:\Users\PeterDeTender> az acr login --name nopacr1 -g noprg
Argument 'resource_group_name' has been deprecated and will be removed in a future release.
Login Succeeded
PS C:\Users\PeterDeTender>

```

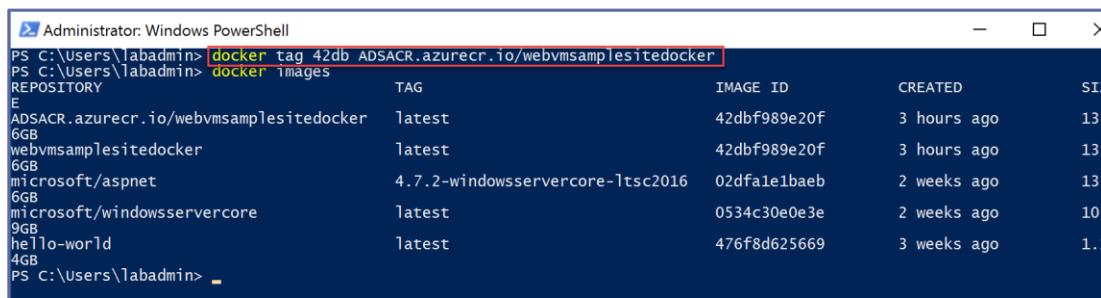
## Task 5: Pushing a Docker image into an Azure Container Registry

- As we now have connectivity towards the ACR, we can push our Docker image to it. There is however a dependency that the name of our Docker image needs to have the name of the Azure Container Registry in it. So we first need to update the Docker image tag for our Docker image, by executing the following command:

```
docker images (To get the image ID number)
```

```
docker tag 42db [SUFFIX]ACR.azurecr.io/<nameyouwanttогive>
```

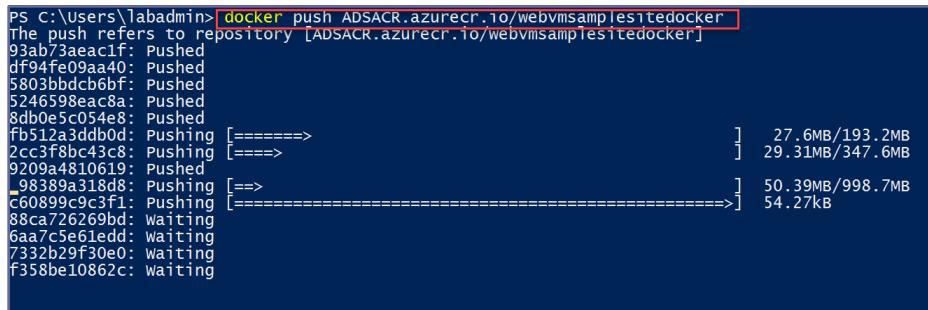
```
docker images (To validate the "new" image)
```



```
Administrator: Windows PowerShell
PS C:\Users\labadmin> docker tag 42db ADSACR.azurecr.io/webvmsamplesitedocker
PS C:\Users\labadmin> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
ADSACR.azurecr.io/webvmsamplesitedocker latest 42dbf989e20f 3 hours ago 13.6GB
webvmsamplesitedocker latest 42dbf989e20f 3 hours ago 13.6GB
microsoft/aspenet 4.7.2-windowsservercore-ltsc2016 02dfa1e1baeb 2 weeks ago 13.6GB
microsoft/windowsservercore 9.0.17763.1000 0534c30e0e3e 2 weeks ago 10.9GB
hello-world 4.0.0 476f8d625669 3 weeks ago 1.14GB
PS C:\Users\labadmin>
```

- Execute the following command to upload this image to the Azure Container Registry:

```
docker push [SUFFIX]ACR.azurecr.io//<nameyouwanttогive>
```



```
PS C:\Users\labadmin> docker push ADSACR.azurecr.io/webvmsamplesitedocker
The push refers to repository [ADSACR.azurecr.io/webvmsamplesitedocker]
93ab73aeac1f: Pushed
df94fe09aa40: Pushed
5803bbdcbb6f: Pushed
5246598eac8a: Pushed
8db0e5c054e8: Pushed
fb512a3ddb0d: Pushing [=====] 27.6MB/193.2MB
2cc3f8bc43c8: Pushing [====] 29.31MB/347.6MB
9209a4810619: Pushed
98389a318d8: Pushing [==>] 50.39MB/998.7MB
c60899c9c3f1: Pushing [=====] 54.27KB
88ca726269bd: Waiting
6aa7c5e61edd: Waiting
7332b29f30e0: Waiting
f358be10862c: Waiting
```

- Wait for this process to complete successfully; depending on internet connection speed, this might take some time.
- From the Azure Portal | All Services | Azure Container Registries | select the ACR you created earlier.

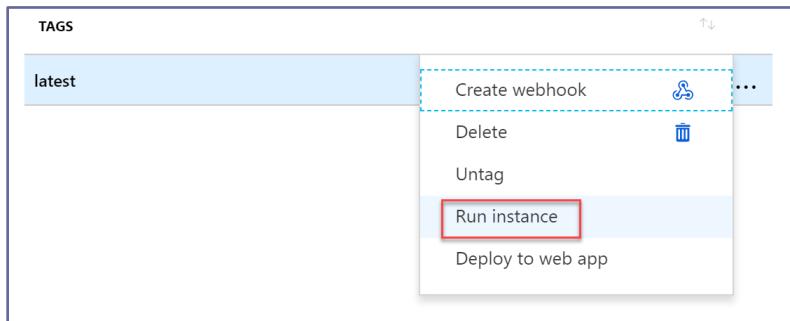
- In the blade menu to the left under the **Services** section, click **Repositories**. Notice the Docker image "webvmsamplesitedocker" is stored in here.

- Click the <yourcontainername> repository, which opens the specific details for this image, exposing its version:

The screenshot shows the Azure Container Registry interface. On the left, there's a search bar and a list of repositories: 'nopcommerce\_420\_source\_nopcom...', 'nopsql', 'simldockerv1', and 'simplpdvtv1'. The 'simplpdvtv1' repository is selected and highlighted in blue. On the right, the repository details are shown: name 'simplpdvtv1', tag count '1', last updated date '8/13/2019, 11:24 PM GMT+3', manifest count '1', and a tag list with 'latest'.

## Task 6: Running an Azure Container Instance from a Docker image in Azure Container Registry

1. Click the ... next to latest, and choose Run Instance



2. This opens the Create Container Instance blade. Complete the parameter fields using the following information:

- Container Name	samplesitecontainer
- OS-type	Windows
- Subscription	your Azure Subscription
- Resource Group	select [Suffix]-DockerRG as Resource Group
- Location	should be picked up from the Resource Group

Leave all other settings unchanged (1 core, 1,5GB memory, Public IP address YES and Port 80)

## Create container instance

\* Container name

Container image

OS type

Linux  Windows

\* Subscription

\* Resource group

[Create new](#)

\* Location

Number of cores

\* Memory (GB)

Public IP address

Yes  No

\* Port

3. Press **OK** to have the Container Instance created.
4. Wait for the deployment process to complete successfully.
5. Once the deployment is finished, **open the Azure Container Instance** in the portal (All Services | Container Instances), and **browse to the ACI "samplesitecontainer"** that just got created.

**simplpdvtv1aci**  
Container instances

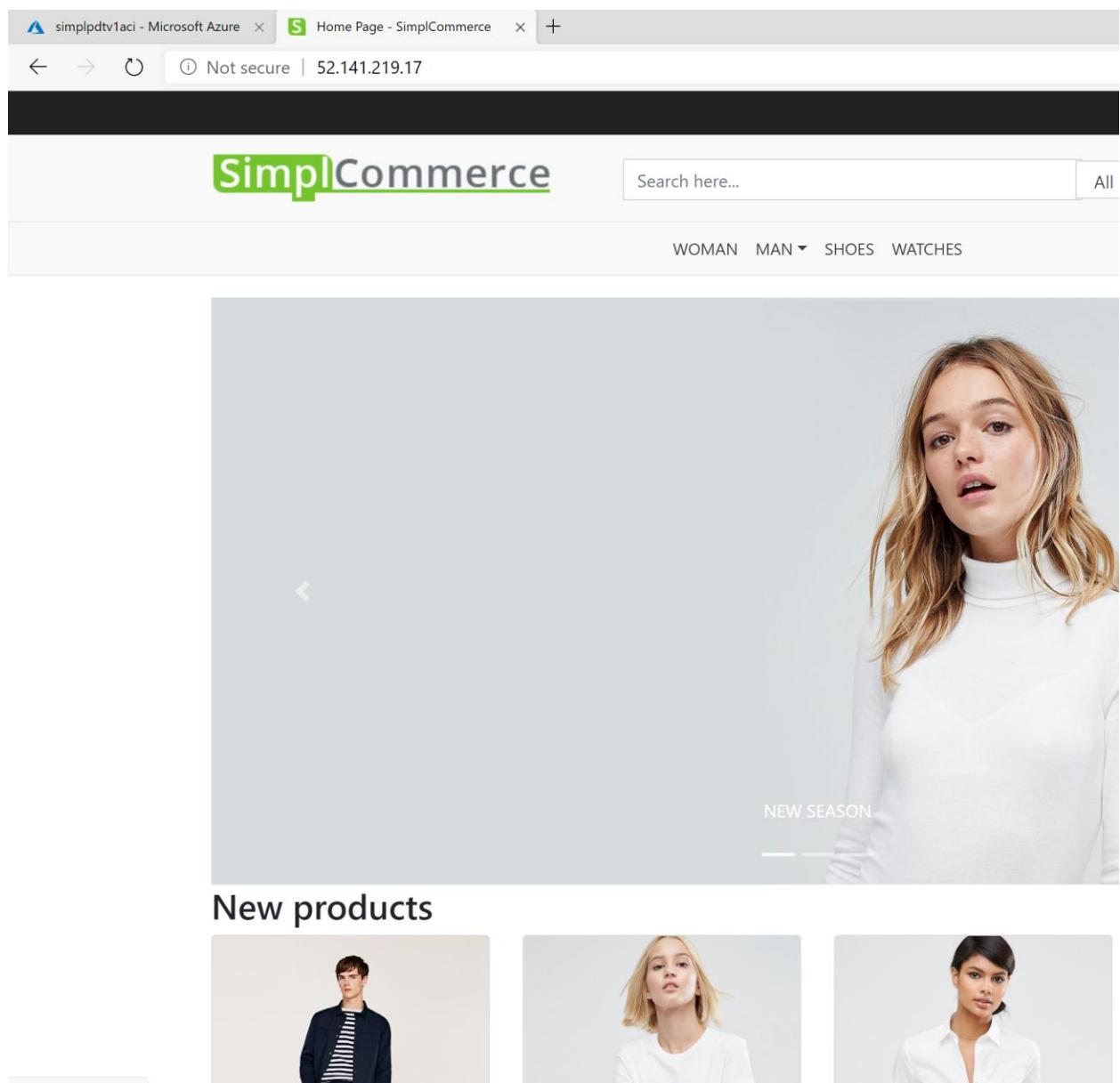
Search (Ctrl+ /)

Overview     Activity log     Access control (IAM)     Tags     Settings     Containers

Resource group (change) : simplwebappRG     OS type : Linux  
Status : Running     IP address : 52.141.219.17  
Location : Central US     FQDN : ---  
Subscription (change) : 007FFFLearning Labs     Container count : 1  
Subscription ID : 0a407898-c077-442d-8e17-71420aa82426  
Tags (change) : Click here to add tags

Refresh

6. Copy the IP address for this Azure Container Instance, or directly browse to it from your internet browser, which should load your application successfully.



7. The sample website starts successfully, and again has connectivity to the underlying SQL Azure database. Notice the name of the Azure Container Instance is visible too.

## Task 7: Deploy a Container using Azure Web Apps

As you heard, it is also possible to run your Docker Containers in an Azure Web App, as an alternative to Azure Container Instance. That's what you will build in this task.

1. From the Azure Portal | Create New Resource | Web App.

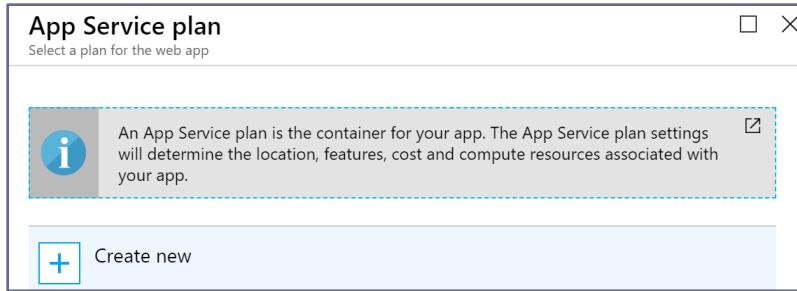
The screenshot shows the Azure Marketplace search results for 'web app'. The search bar at the top contains 'web app'. Below the search bar, there is a 'Filter' button and a 'Results' section. The results table has columns for NAME, PUBLISHER, and CATEGORY. One result is listed: 'Web App' by Microsoft, categorized under 'Web'.

2. Press the Create button to open the Web App blade. Complete the required parameters as follows:
  - App Name: [SUFFIX]dockerwebapp.azurewebsites.net
  - Resource Group: Create New | [SUFFIX]dockerwebappRG
  - OS: Windows
  - Publish: Docker Image

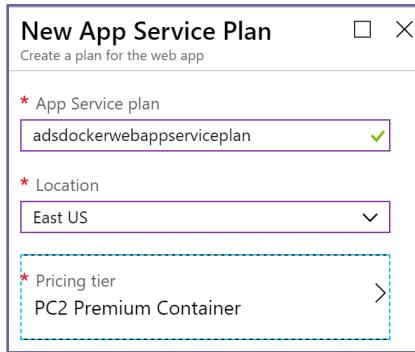
The screenshot shows the 'Web App' creation blade. The 'Create' tab is selected. The form fields are:

- \* App name: adsdockerwebapp.azurewebsites.net
- \* Subscription: Microsoft Azure Sponsorship
- \* Resource Group: Create new (radio button selected) adsdockerwebappRG
- \* OS: Windows (radio button selected)
- \* Publish: Docker Image (radio button selected)

3. For the Service Plan parameter, click Create New



4. Complete the required parameters for the App Service Plan as follows:
  - App Service Plan: [SUFFIX]dockerwebappserviceplan
  - Location: East US
  - Pricing Tier: Select the PC2 Premium Container plan



And confirm the plan with **OK**.

5. Completing the "Configure Container" parameter opens the detailed blade, where you make the following selections. Notice we have a few options here, using Azure Container Registry, the public Docker Hub repo or Kubernetes integration.

The SimplCommerce application we have been using, is also available as a public Docker Hub container image, so why not testing with that one:

- Single Container
- Image Source Docker Hub
- Registry [SUFFIX]ACR
- image simplcommerce/ci-build
- Tag latest

Home > App Services > simplcontwebapp - Container settings

## simplcontwebapp - Container settings

App Service

Search (Ctrl+ /)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Security

### Deployment

- Quickstart
- Deployment credentials
- Deployment slots
- Deployment Center

## Single Container

Image source

Azure Container Registry Docker Hub Private Registry

Repository Access

Public Private

Image and optional tag (eg 'image:tag')

simplcommerce/ci-build

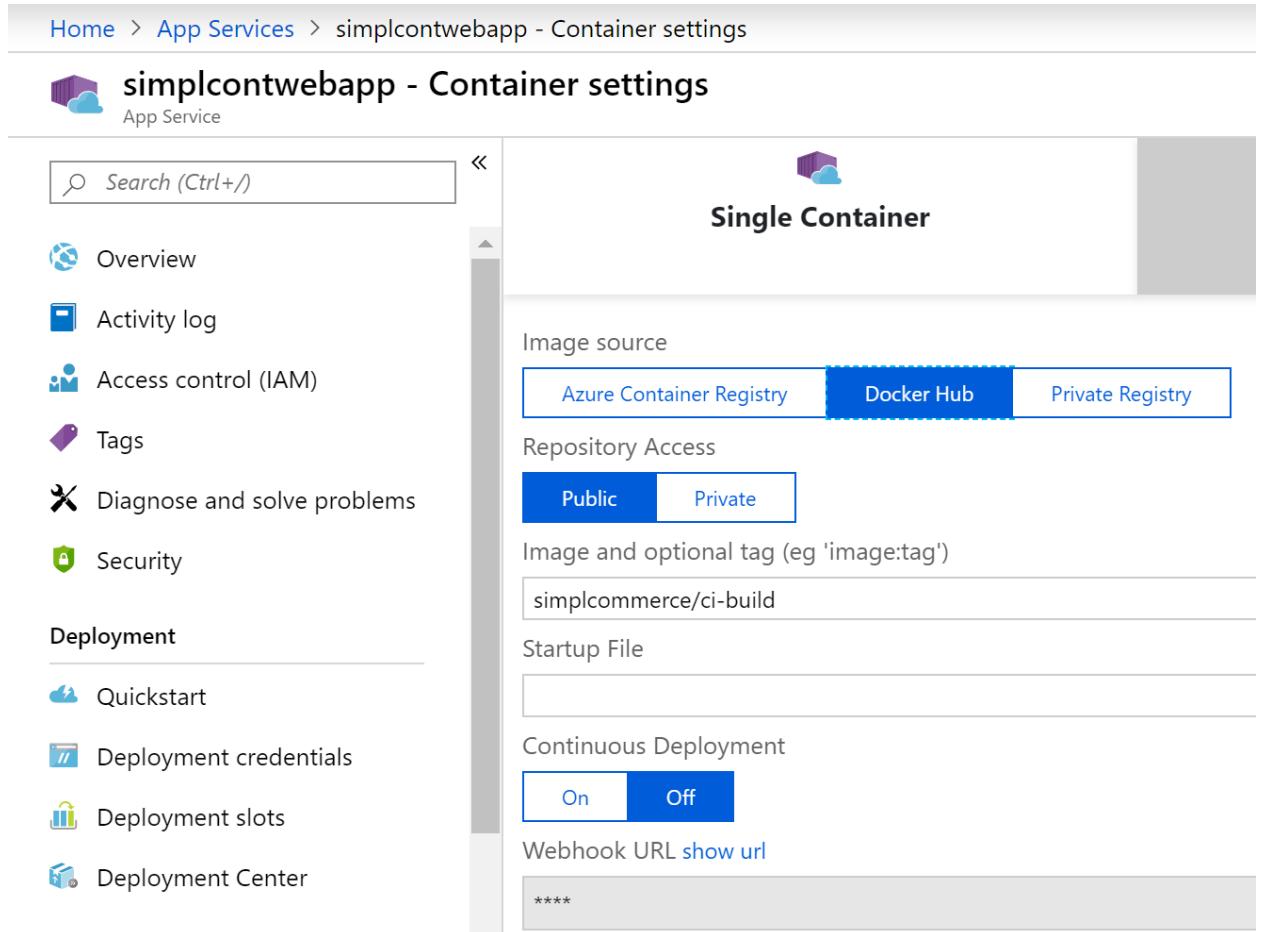
Startup File

Continuous Deployment

On Off

Webhook URL [show url](#)

\*\*\*\*



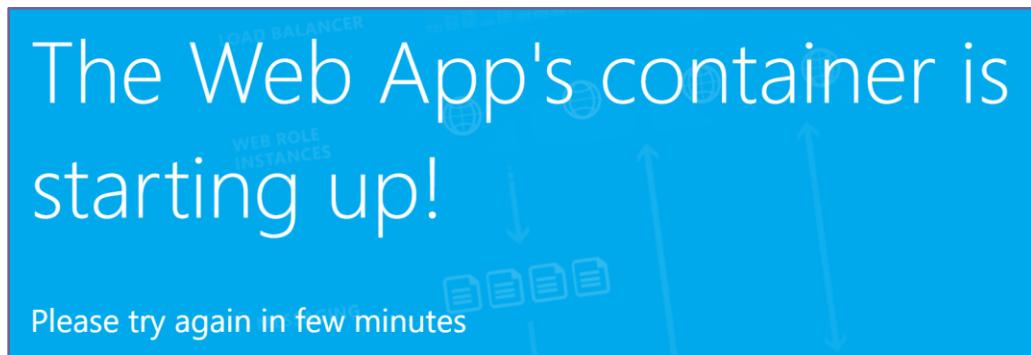
6. Confirm the creation by pressing the **Apply** button.
7. Press the **Create** button to start the deployment of the Azure Web App for Containers.
8. Follow-up on the deployment from the notification area.
9. Once deployed, browse to the [SUFFIX]dockerwebapp Azure Resource, which opens the detailed blade:

The screenshot shows the Azure Portal's 'App Services' section. On the left, a sidebar lists options like 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Deployment', 'Settings', 'Configuration', and 'Container settings'. The 'Overview' tab is selected. The main pane displays details for the app service 'simplcontwebapp' under 'Resource group (change) : simplwebappRG'. Key information includes:

- Status: Running
- Location: Central US
- Subscription (change): 007FFF Learning Labs
- Subscription ID: 0a407898-c077-442d-8e17-71420aa82426
- Tags (change): Click here to add tags
- URL: https://simplcontwebapp.azurewebsites.net
- App Service Plan: noplans (S1: 1)
- FTP/Deployment user name: simplcontwebapp\ftppdt
- FTP hostname: ftp://waws-prod-dm1-123.ftp.azurewebsites.win...
- FTPS hostname: https://waws-prod-dm1-123.ftp.azurewebsites.wi...

Below the main details, there are two cards: 'Diagnose and solve problems' and 'App Service Advisor'. At the bottom, three performance metrics are shown: 'Http 5xx' (100), 'Data In' (800B), and 'Data Out' (10kB).

- Copy the URL and paste it in your internet browser. Note the message about the container starting up:



- Go back to the Azure Portal, which still has your Azure Web App for Containers open; here, browse to Settings | Container Settings and look at the LOGs section. This shows the different steps undergoing to get the container running.

The screenshot shows the 'Deployment Center (Preview)' interface. The left sidebar has tabs for 'Deployment Center (Preview)', 'Settings', 'Application settings', 'Container settings' (which is selected), 'Authentication / Authorizati...', 'Application Insights', and 'Managed service identity'. The right pane is titled 'Logs' and displays the following log entries:

```

[01/10/2018 03:08:56.50 / INFO - Site: adsdockerwebapp - Image: adsacr.azurecr.io/webvmsamplesitedockerlatest
Custom Registry: https://adsacr.azurecr.io
Status: Digest: sha256:84569ec016f4f6bf381cf904e051d01b92506c8687bd214a679cab94e6aeef4c
01/10/2018 03:08:56.513 INFO - Site: adsdockerwebapp - Image: adsacr.azurecr.io/webvmsamplesitedockerlatest
Custom Registry: https://adsacr.azurecr.io
Status: Status: Downloaded newer image for adsacr.azurecr.io/webvmsamplesitedocker:latest
01/10/2018 03:08:56.517 INFO - Site: adsdockerwebapp - Creating container for image: adsacr.azurecr.io/webvmsamplesitedocker:latest.
01/10/2018 03:08:56.742 INFO - Site: adsdockerwebapp - Create container for image: adsacr.azurecr.io/webvmsamplesitedocker:latest succeeded.
Container Id: 611926af5e92281c2bd31c8289c030f3c0db6e71cc04ee89e7e3f2f3e4b57cd7

```

- Wait for the Logs output mentioning the container is started and configuration completed successfully.

Deployment Center (Preview)

**Logs**

```

status: Status: Downloaded newer image for adsacr.azurecr.io/webvmsamplesitedocker:latest
01/10/2018 03:08:56.517 INFO - Site: adsdockerwebapp - Creating container for image: adsacr.azurecr.io/webvmsamplesitedocker:latest.
01/10/2018 03:08:56.742 INFO - Site: adsdockerwebapp - Create container for image: adsacr.azurecr.io/webvmsamplesitedocker:latest succeeded.
Container Id 611926af5e92281c2bd31c8289c030f3c0db6e71cc04ee89e7e3f2f3e4b57cd7
01/10/2018 03:10:08.438 INFO - Site: adsdockerwebapp - Start container succeeded. Container: 611926af5e92281c2bd31c8289c030f3c0db6e71cc04ee89e7e3f2f3e4b57cd7
01/10/2018 03:10:13.201 INFO - Site: adsdockerwebapp - Container ready
01/10/2018 03:10:13.201 INFO - Site: adsdockerwebapp - Configuring container
01/10/2018 03:10:51.379 INFO - Site: adsdockerwebapp - Container ready
01/10/2018 03:10:51.381 INFO - Site: adsdockerwebapp - Container start-up and configuration completed successfully

```

- If you go back to your browser window with the “container starting message” and refresh it, it opens up the containerized web application as expected.

https://simplcontwebapp.azurewebsites.net

**SimplCommerce**

Search here... All Categories

WOMAN MAN ▾ SHOES WATCHES

New arrivals

New products

<b>Lightweight Jacket</b> \$58.79 <small>-34%</small> \$69.00	<b>Esprit Ruffle Shirt</b> \$16.64	<b>Herschel supply</b> \$35.31	<b>Only Check Trouser</b> \$25.50
☆☆☆☆☆	☆☆☆☆☆	☆☆☆☆☆	☆☆☆☆☆

- This completes this task.

## 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.

In the following tasks, you migrated the Docker container to Azure Container Registry, and deployed a Container Instance running the image. You also learned how to deploy Azure WebApp for containers and validating each process was working fine and offering a running e-commerce platform.

# 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