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

Contents

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 2nd 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 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, 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.
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:

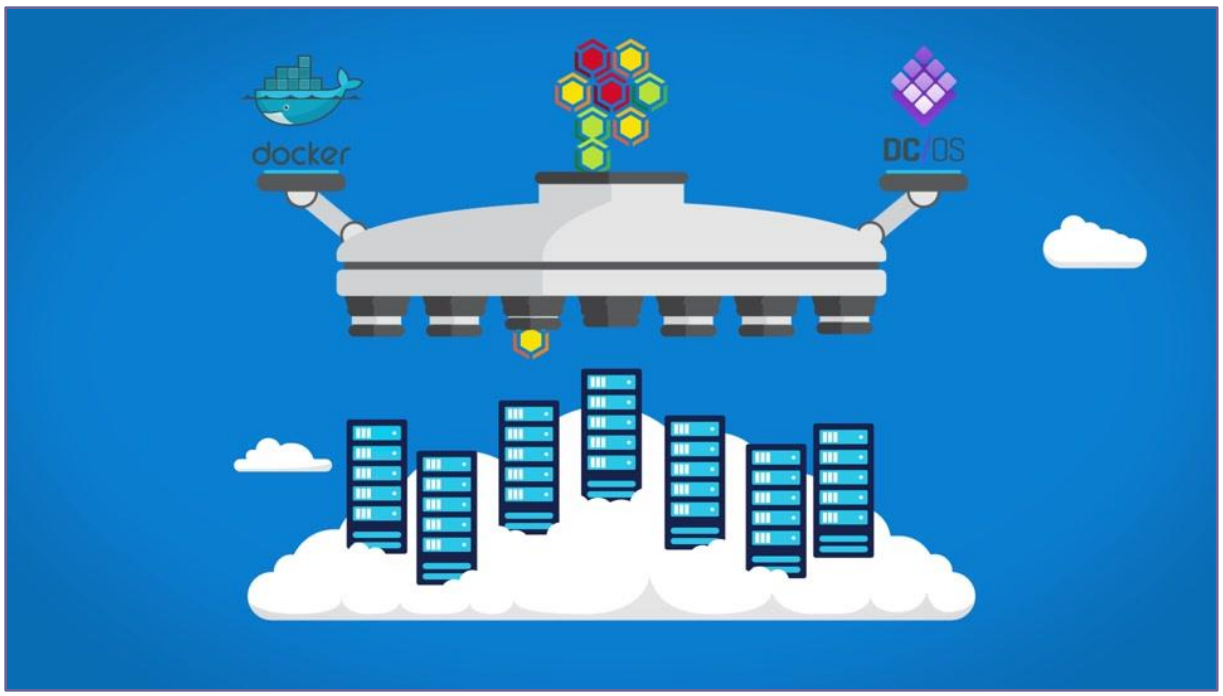
- Visual Studio 2017 community edition (updated to latest version); this could also be Visual Studio 2019 community edition - latest version
- Docker for Windows (updated to latest version)
- Azure CLI 2.0 (updated to latest version)
- Kubernetes CLI (updated to latest version)
- SimplCommerce Open Source 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

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

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, 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.
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.

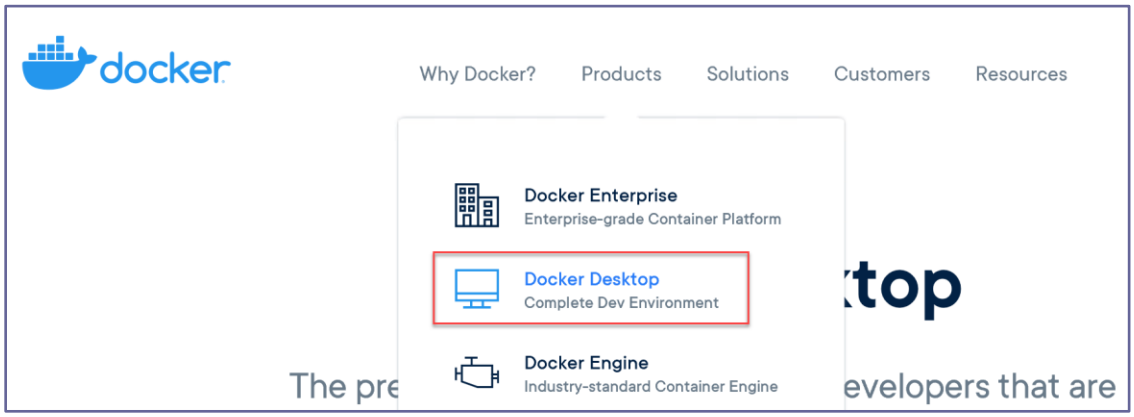
Time Estimate

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 as credentials.

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



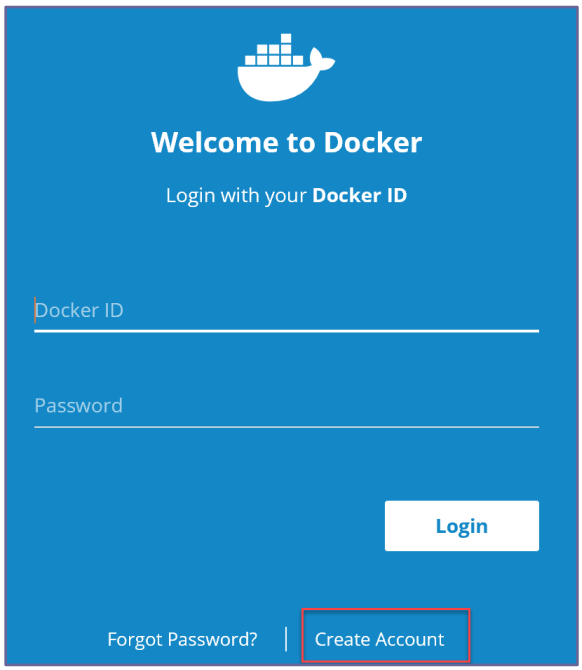
3. Click "Download for Windows".

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.

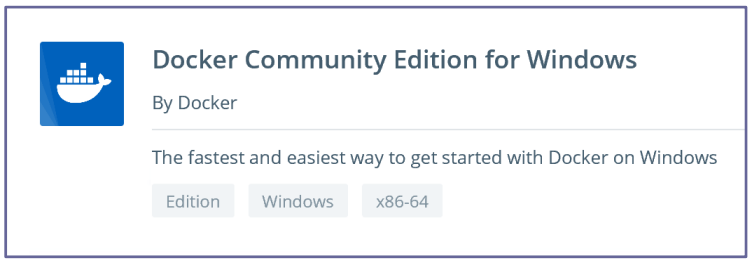
6. Click the Login to Portal button; in the login portal, choose Create Account.



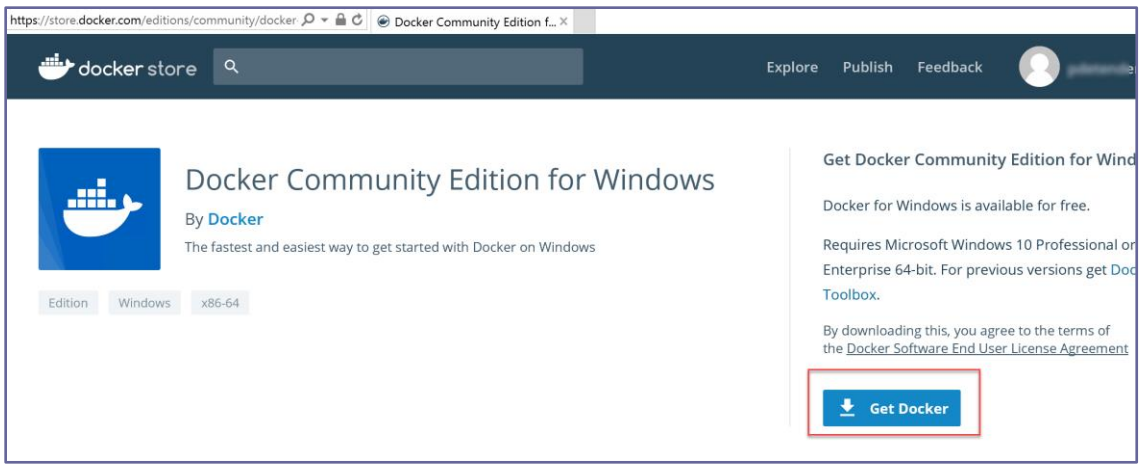
7. Choose a Docker ID, use an active email address and choose a password. Complete the process for account creation (check your email for any activation confirmation email).

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.

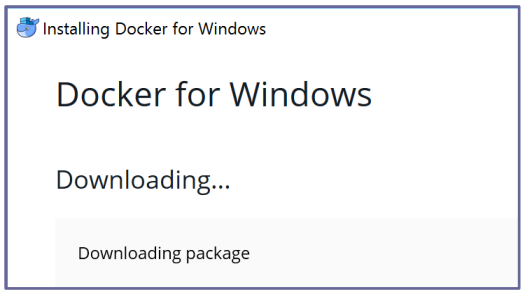


10. Click on the object title, which opens the download page for this solution. Click the Get Docker button.

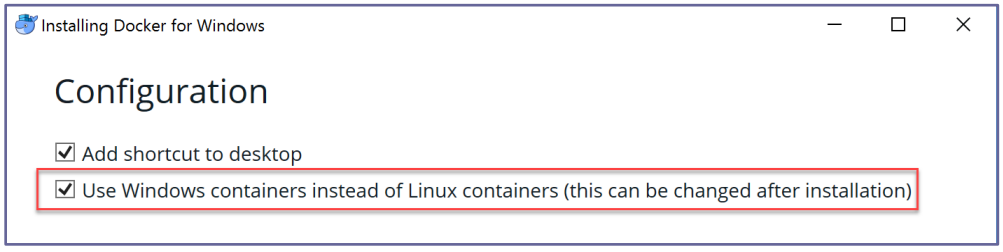


11. Save the install files to the local machine's Downloads folder; wait for the download to complete, and run the actual install process.

12. The Docker for Windows installer kicks off



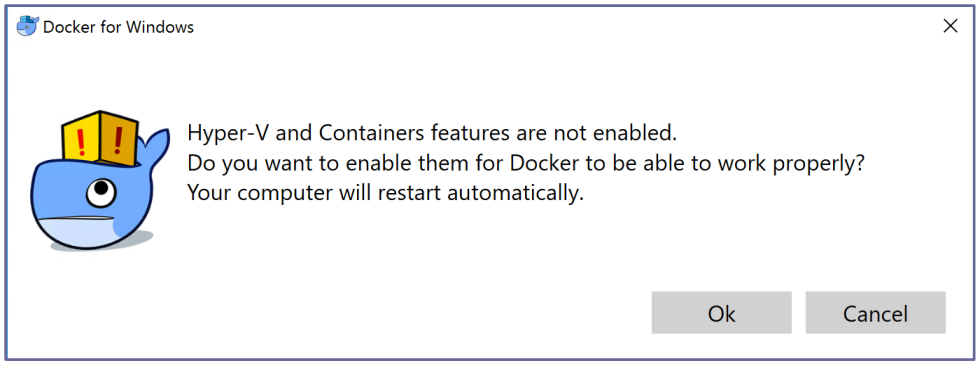
13. Wait for the background download to complete, until you see the Configuration step of the install wizard. Here, select "Use Windows containers instead of Linux containers". Press OK to continue.



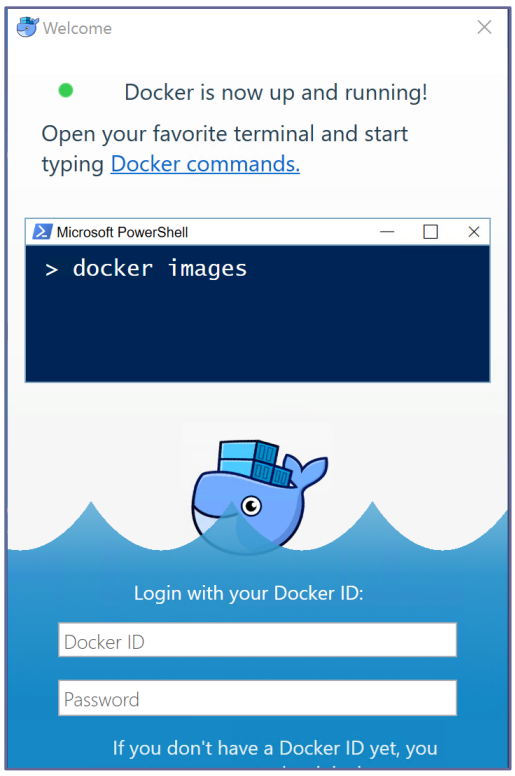
14. The Docker install will update the install packages and run the actual tool installation. Wait for this step to complete.

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 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.



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.

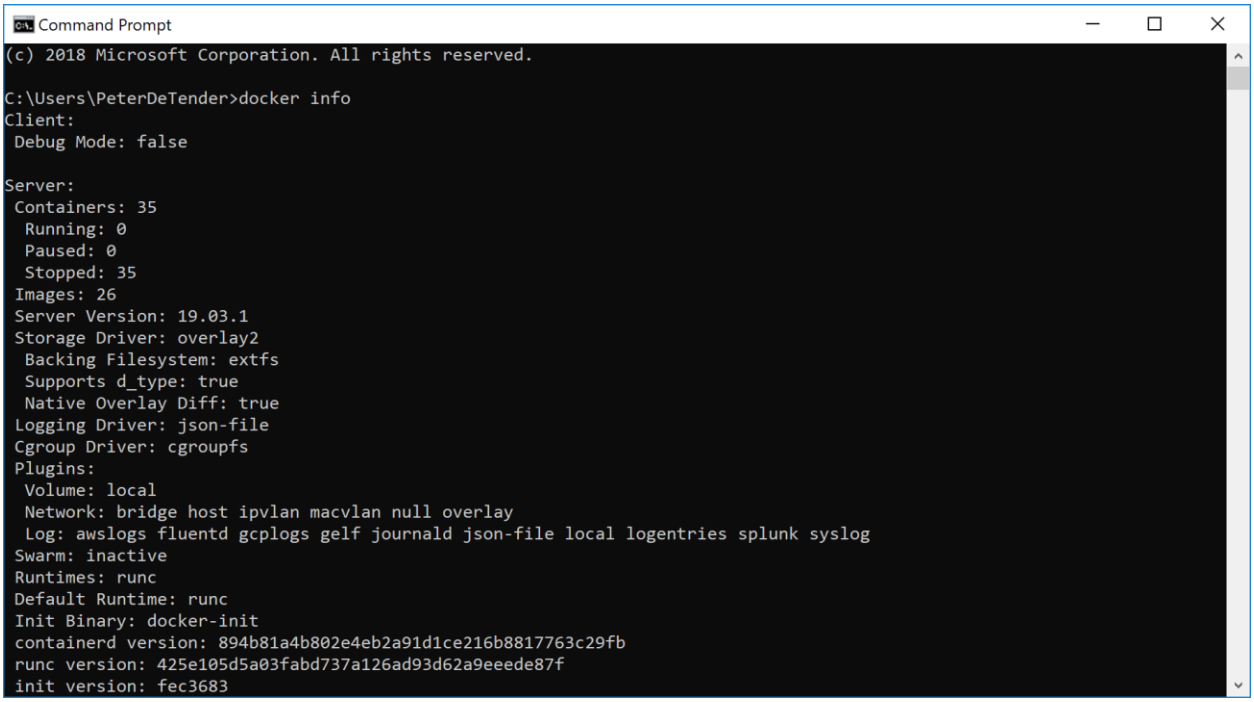


18. Enter your Docker credentials.

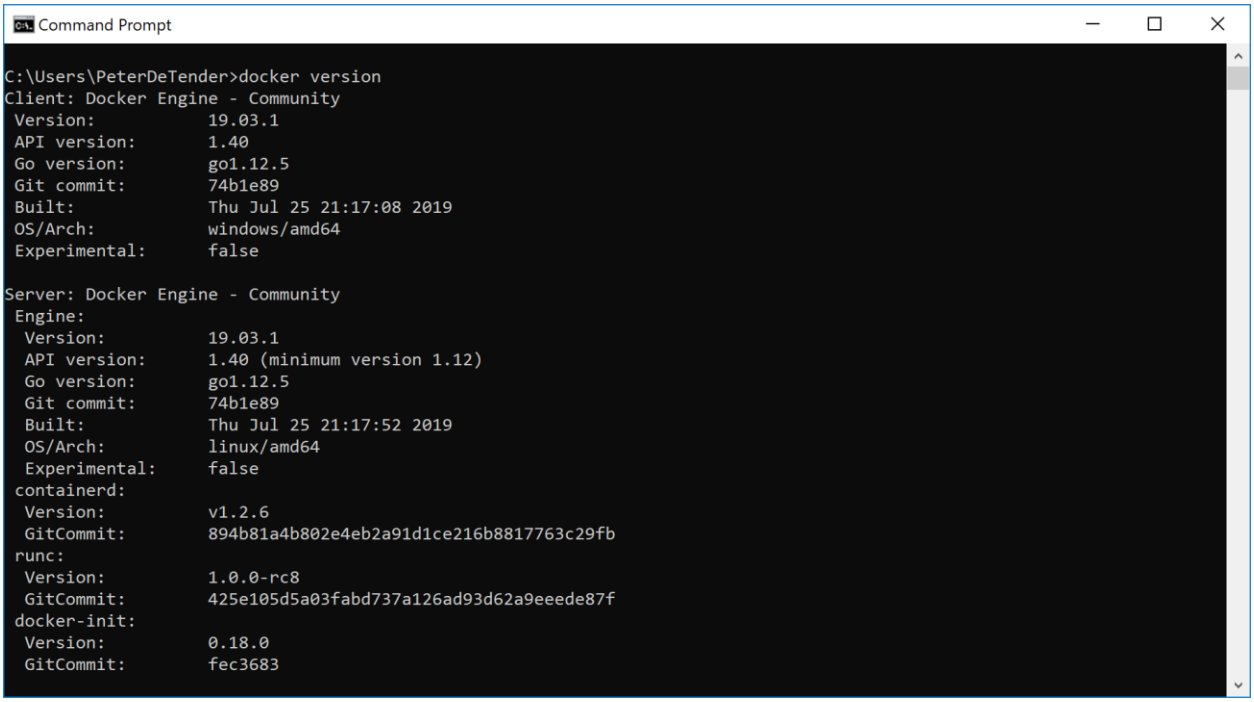
19. Once you have successfully logged on to Docker CE, open a command prompt from the Start screen. (Note: you can also run this from within PowerShell if that gets your preference)

20. Execute the following command:

```
docker info
```



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

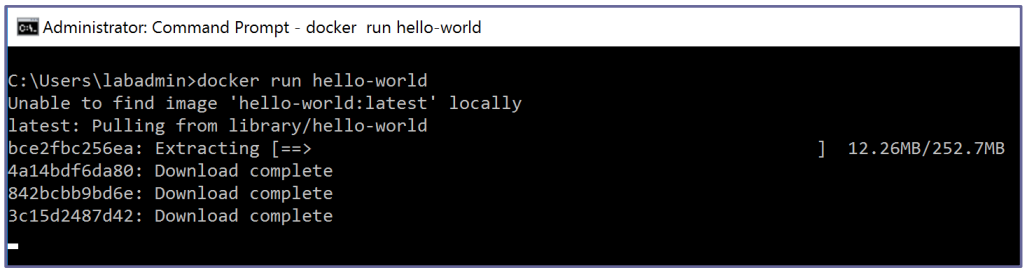


22. To validate we can actually "run" a Docker container, initiate the following command:

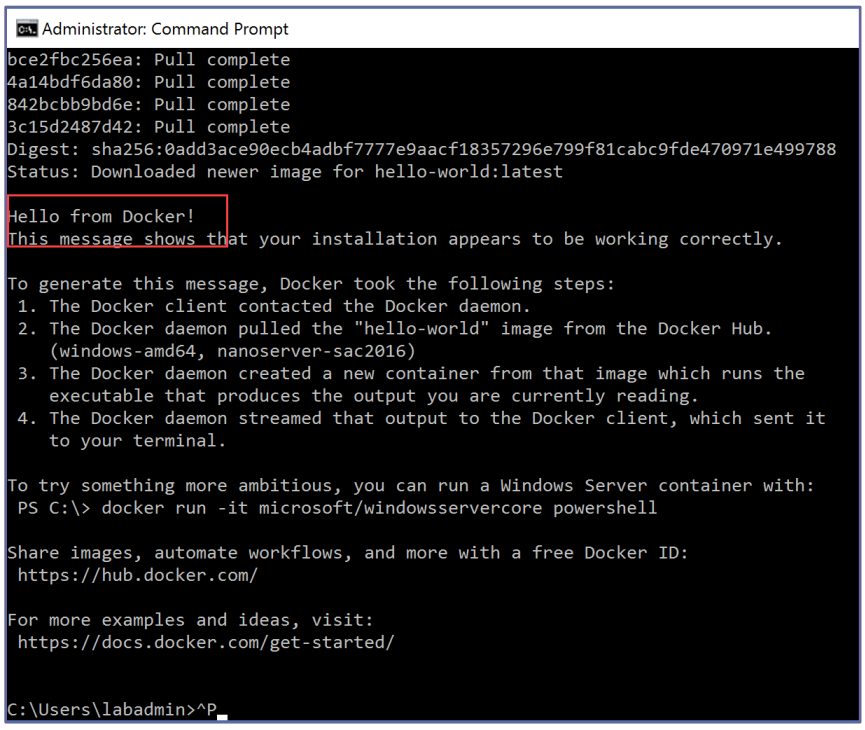
```
docker run hello-world
```

Some explanation what this command executes:

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



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.:



Task 2: Operating Docker from the command line

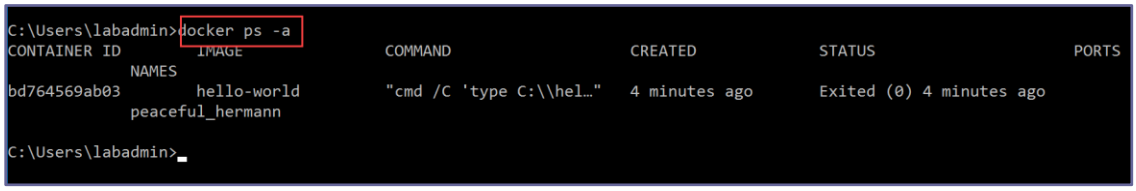
1. Another useful Docker command is to see what containers are running. Launch the following

   ```
   docker ps
   ```

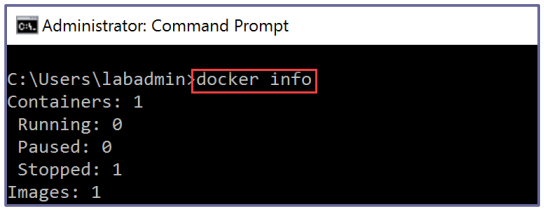
   Note we don't have any Docker containers running; this was part of the hello-world container, which runs, and eventually automatically stops again.

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

   ```
   docker ps -a
   ```

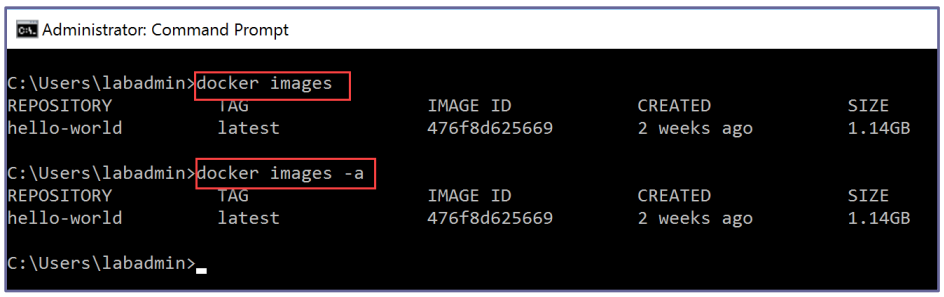


3. Execute the command `Docker info` again; it will now have additional information related to our containers and images



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

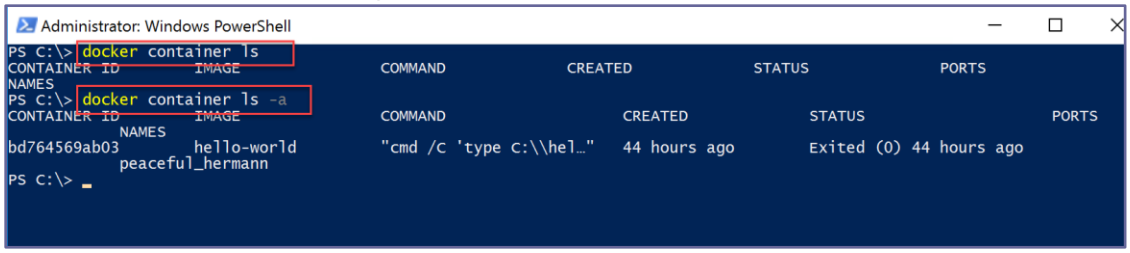
   ```
   docker images
   ```



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`

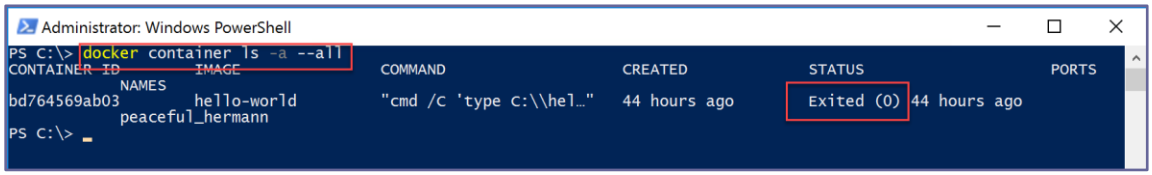
   (Note I switched to PowerShell from here on, to show you both command line tools are transparent across Docker usage)



   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

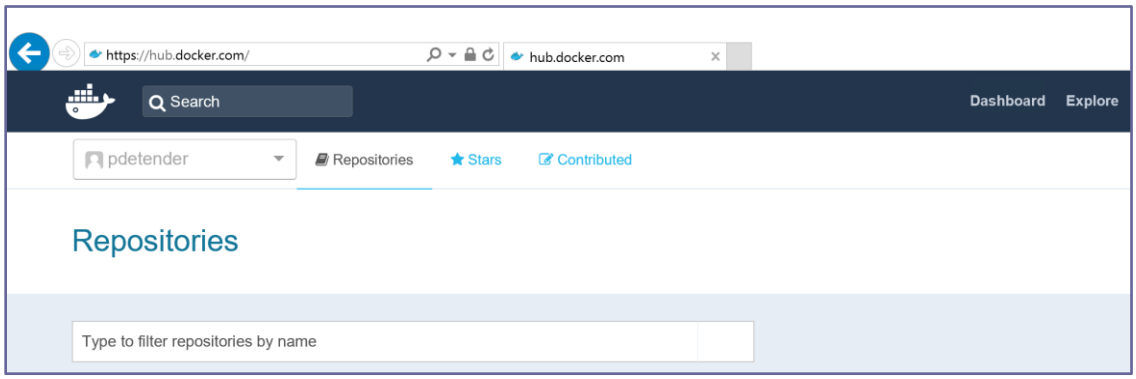


   Which shows you this container is in an exited (=stopped) state since 44 hours. (if you are wondering where the 44 hours comes from, no worries, lab guide authors have a life too 😊).

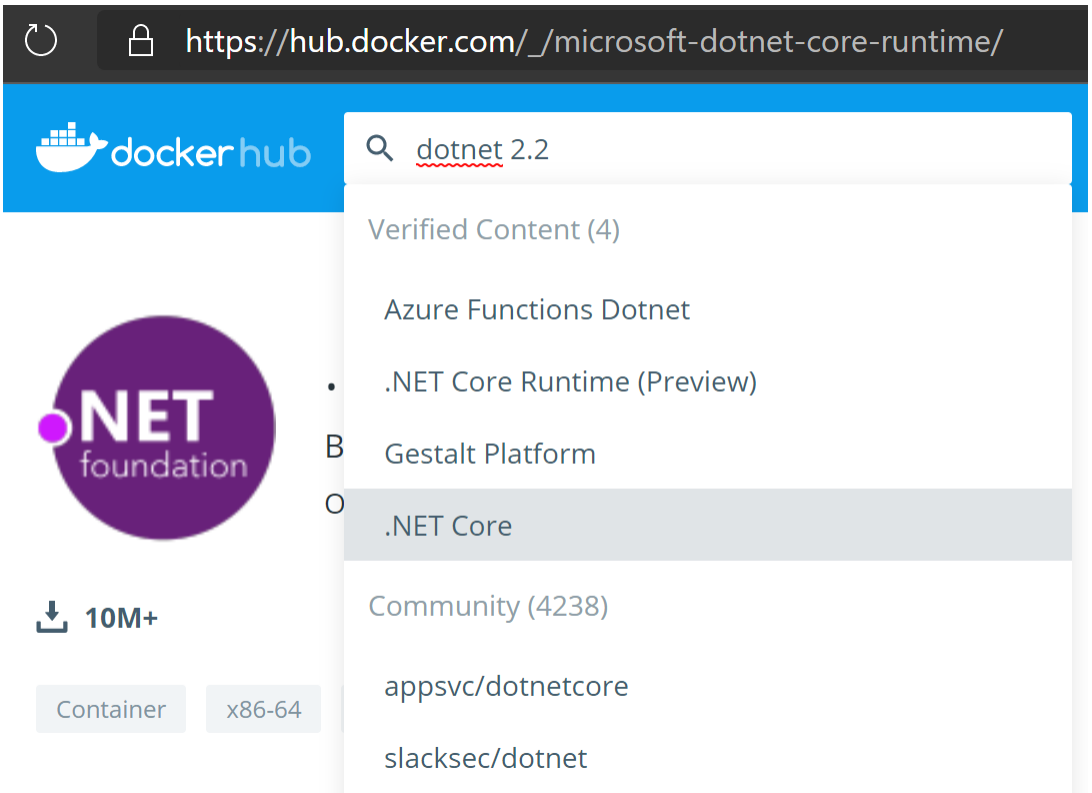
Similar to the hello-world container images that got pulled down before running, let's move on with exploring the different Windows OS-based container images in Docker Store, and pulling them to our

local lab-jumpVM.

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

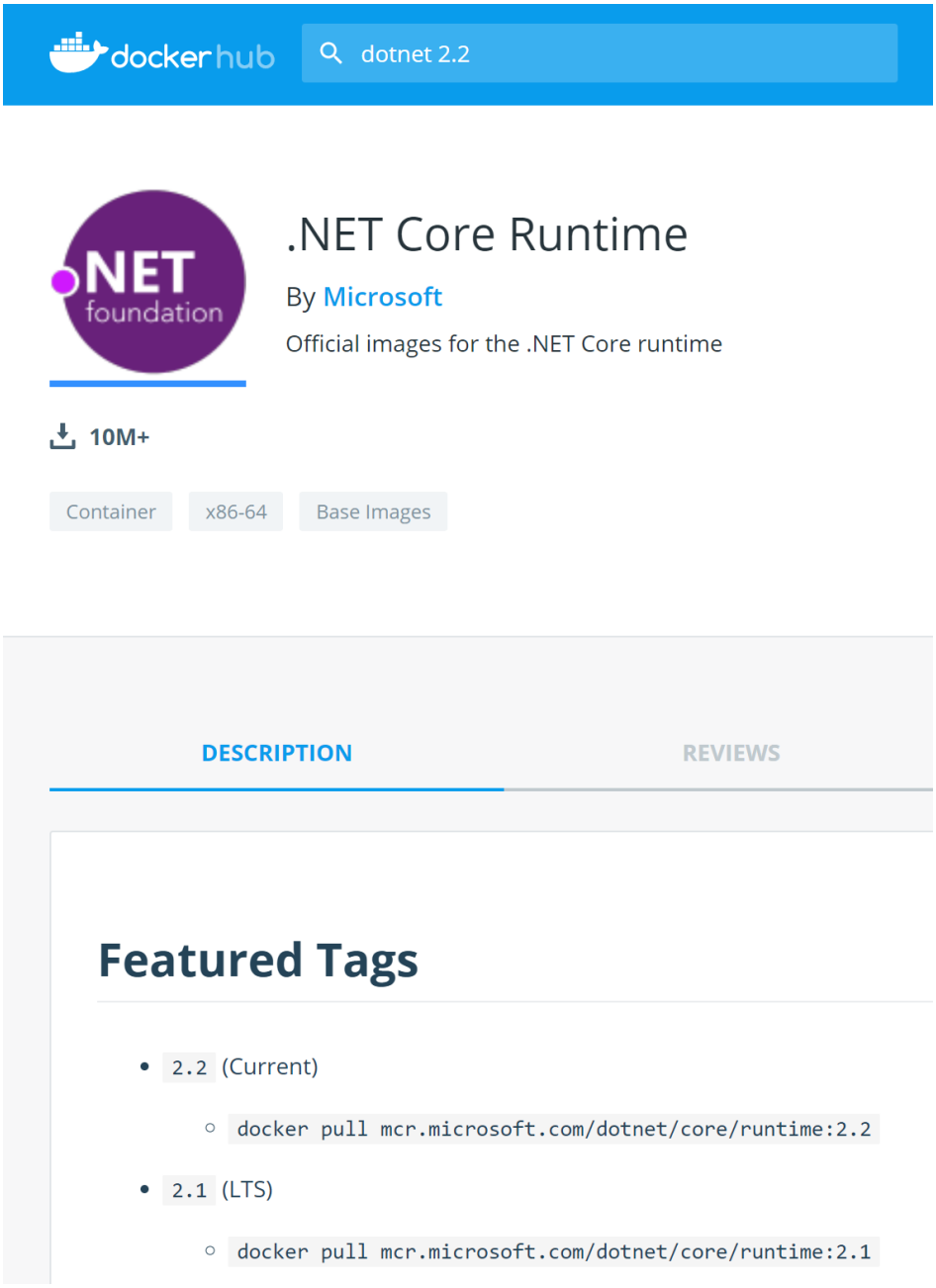


8. In the upper **Search field, type "dotnet 2.2";** this results in a list of repositories.



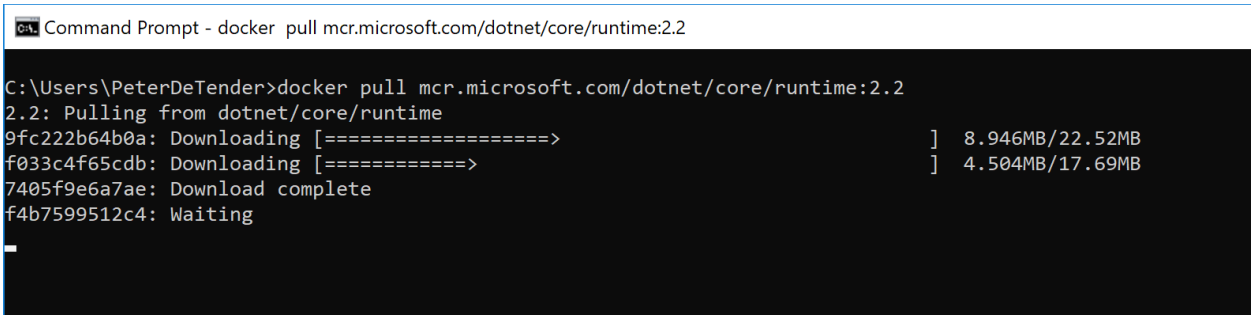
9. **Select the ".NET Core"** repository. Which shows you the different .NET Core-based Docker images, provided by Microsoft. Important difference to note is that the actual Container is NOT stored in the Docker Hub, but stored inside Microsoft's own Repository for containers

(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.



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

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



10. Notice this image is based on different "layers" (the 9fc222b64…, f033c4…,7405f9…)

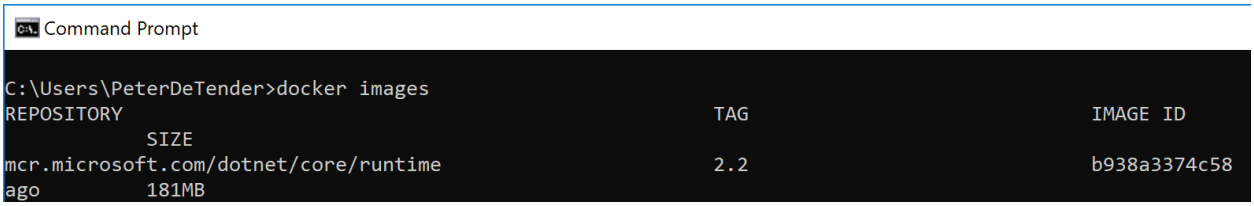
11. Wait for the image to finish downloading and extracting. Depending on your internet connection speed, this might take several minutes. From a container perspective, this is based on the Microsoft/dotnet core, having a tag pointing to the latest edition (the 2.2-part in the name). The advantage for 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:

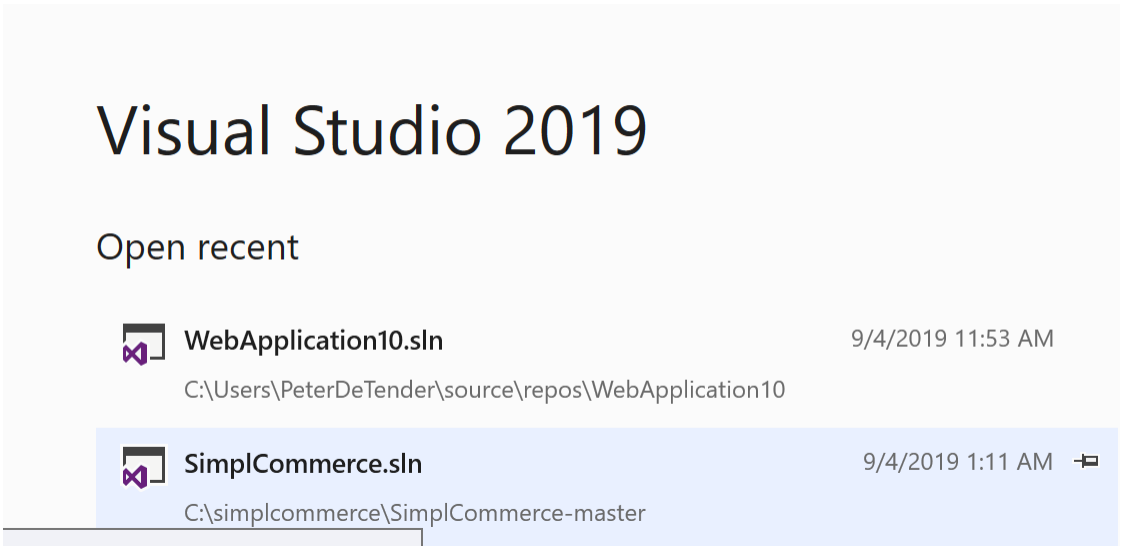
   ```
   docker images
   ```

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

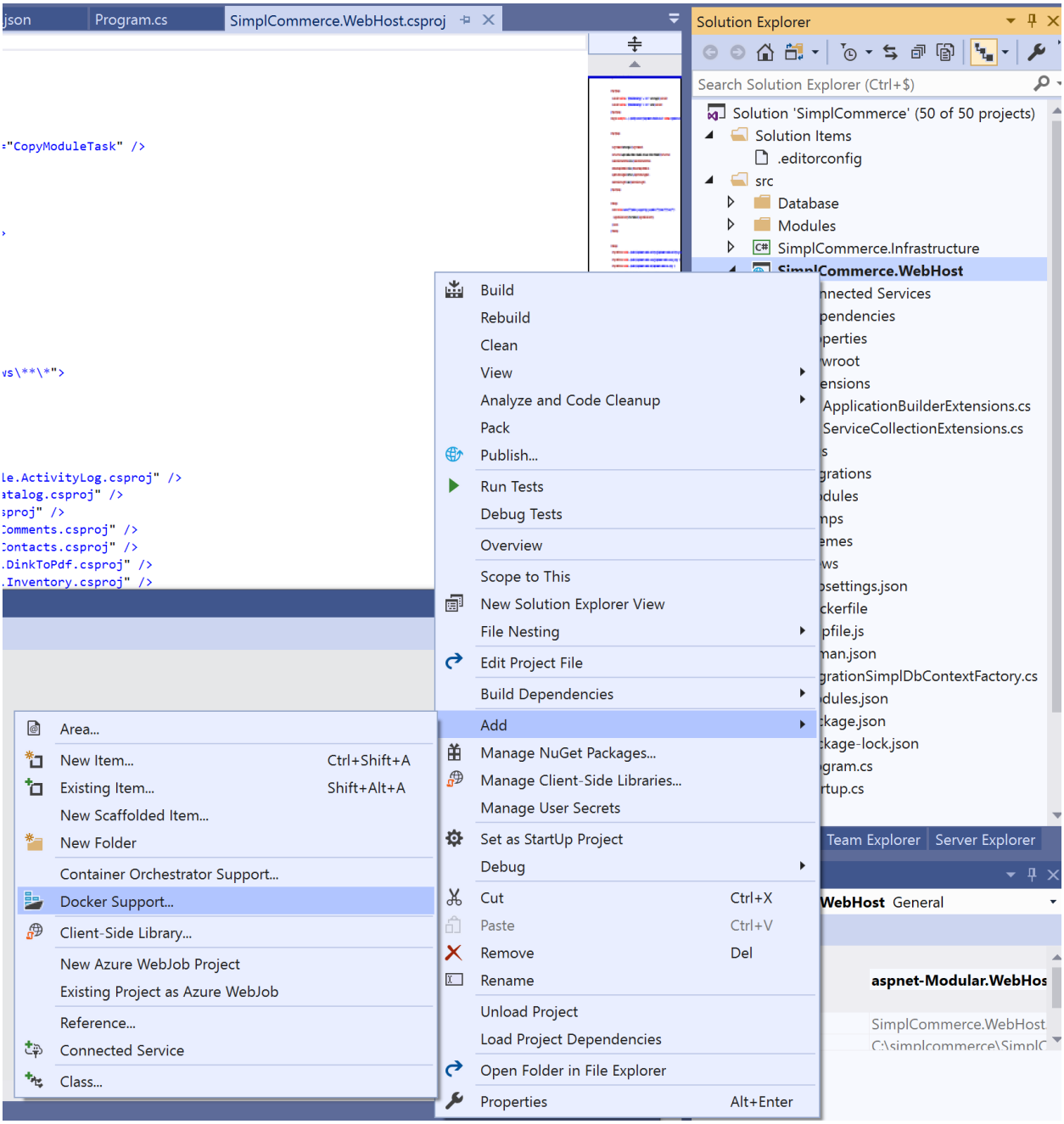


While you can configure a Docker Container in a full manual way, let me walk you through the Visual Studio-based approach, as yes, thanks to having installed Docker Desktop on the same host as Visual Studio, it's providing a really nice integration…

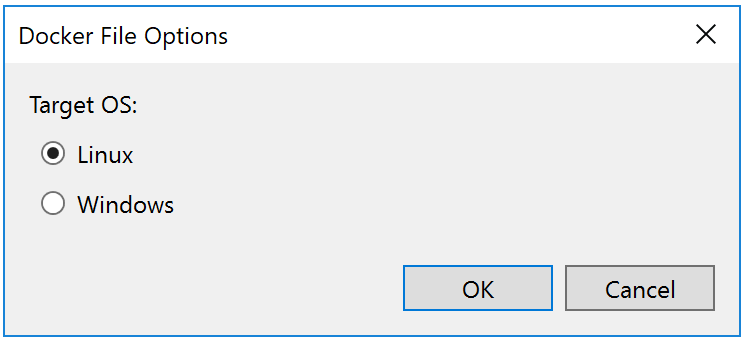
2. From Visual Studio, Open the recent SimplCommerce.cln project, pulling up the source code from the web application



3. Once loaded, select the SimplCommerce.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**.



5. 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)

6. 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

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2-stretch-slim AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/core/sdk:2.2-stretch AS build
WORKDIR /src
COPY ["src/SimplCommerce.WebHost/SimplCommerce.WebHost.csproj", "src/SimplCommerce.WebHost/"]
COPY ["src/Modules/SimplCommerce.Module.DinkToPdf/SimplCommerce.Module.DinkToPdf.csproj", "src/Modules/SimplCommerce.Module.DinkToPdf/"]
COPY ["src/SimplCommerce.Infrastructure/SimplCommerce.Infrastructure.csproj", "src/SimplCommerce.Infrastructure/"]
COPY ["src/Modules/SimplCommerce.Module.Core/SimplCommerce.Module.Core.csproj", "src/Modules/SimplCommerce.Module.Core/"]
COPY ["src/Modules/SimplCommerce.Module.Reviews/SimplCommerce.Module.Reviews.csproj", "src/Modules/SimplCommerce.Module.Reviews/"]
COPY ["src/Modules/SimplCommerce.Module.Orders/SimplCommerce.Module.Orders.csproj", "src/Modules/SimplCommerce.Module.Orders/"]
COPY ["src/Modules/SimplCommerce.Module.ShippingPrices/SimplCommerce.Module.ShippingPrices.csproj", "src/Modules/SimplCommerce.Module.ShippingPrices/"]
COPY ["src/Modules/SimplCommerce.Module.Catalog/SimplCommerce.Module.Catalog.csproj", "src/Modules/SimplCommerce.Module.Catalog/"]
COPY ["src/Modules/SimplCommerce.Module.Tax/SimplCommerce.Module.Tax.csproj", "src/Modules/SimplCommerce.Module.Tax/"]
COPY ["src/Modules/SimplCommerce.Module.Shipping/SimplCommerce.Module.Shipping.csproj", "src/Modules/SimplCommerce.Module.Shipping/"]
COPY ["src/Modules/SimplCommerce.Module.ShoppingCart/SimplCommerce.Module.ShoppingCart.csproj", "src/Modules/SimplCommerce.Module.ShoppingCart/"]
COPY ["src/Modules/SimplCommerce.Module.Pricing/SimplCommerce.Module.Pricing.csproj", "src/Modules/SimplCommerce.Module.Pricing/"]
COPY ["src/Modules/SimplCommerce.Module.ProductRecentlyViewed/SimplCommerce.Module.ProductRecentlyViewed.csproj", "src/Modules/SimplCommerce.Module.ProductRecentlyViewed/"]
COPY ["src/Modules/SimplCommerce.Module.Search/SimplCommerce.Module.Search.csproj", "src/Modules/SimplCommerce.Module.Search/"]
COPY ["src/Modules/SimplCommerce.Module.ShippingTableRate/SimplCommerce.Module.ShippingTableRate.csproj", "src/Modules/SimplCommerce.Module.ShippingTableRate/"]
COPY ["src/Modules/SimplCommerce.Module.Payments/SimplCommerce.Module.Payments.csproj", "src/Modules/SimplCommerce.Module.Payments/"]
COPY ["src/Modules/SimplCommerce.Module.Comments/SimplCommerce.Module.Comments.csproj", "src/Modules/SimplCommerce.Module.Comments/"]
COPY ["src/Modules/SimplCommerce.Module.ShippingFree/SimplCommerce.Module.ShippingFree.csproj", "src/Modules/SimplCommerce.Module.ShippingFree/"]
COPY ["src/Modules/SimplCommerce.Module.News/SimplCommerce.Module.News.csproj", "src/Modules/SimplCommerce.Module.News/"]
COPY ["src/Modules/SimplCommerce.Module.PaymentCashfree/SimplCommerce.Module.PaymentCashfree.csproj", "src/Modules/SimplCommerce.Module.PaymentCashfree/"]
COPY ["src/Modules/SimplCommerce.Module.PaymentNganLuong/SimplCommerce.Module.PaymentNganLuong.csproj", "src/Modules/SimplCommerce.Module.PaymentNganLuong/"]
COPY ["src/Modules/SimplCommerce.Module.EmailSenderSmtp/SimplCommerce.Module.EmailSenderSmtp.csproj", "src/Modules/SimplCommerce.Module.EmailSenderSmtp/"]
COPY ["src/Modules/SimplCommerce.Module.SampleData/SimplCommerce.Module.SampleData.csproj", "src/Modules/SimplCommerce.Module.SampleData/"]
COPY ["src/Modules/SimplCommerce.Module.WishList/SimplCommerce.Module.WishList.csproj", "src/Modules/SimplCommerce.Module.WishList/"]
COPY ["src/Modules/SimplCommerce.Module.PaymentBraintree/SimplCommerce.Module.PaymentBraintree.csproj", "src/Modules/SimplCommerce.Module.PaymentBraintree/"]
COPY ["src/Modules/SimplCommerce.Module.Cms/SimplCommerce.Module.Cms.csproj", "src/Modules/SimplCommerce.Module.Cms/"]
COPY ["src/Modules/SimplCommerce.Module.PaymentStripe/SimplCommerce.Module.PaymentStripe.csproj", "src/Modules/SimplCommerce.Module.PaymentStripe/"]
COPY ["src/Modules/SimplCommerce.Module.ProductComparison/SimplCommerce.Module.ProductComparison.csproj", "src/Modules/SimplCommerce.Module.ProductComparison/"]
COPY ["src/Modules/SimplCommerce.Module.StorageLocal/SimplCommerce.Module.StorageLocal.csproj", "src/Modules/SimplCommerce.Module.StorageLocal/"]
```

7. Followed by a full list of copy statements of each and every VS Modle folder the application is using.

8. 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.

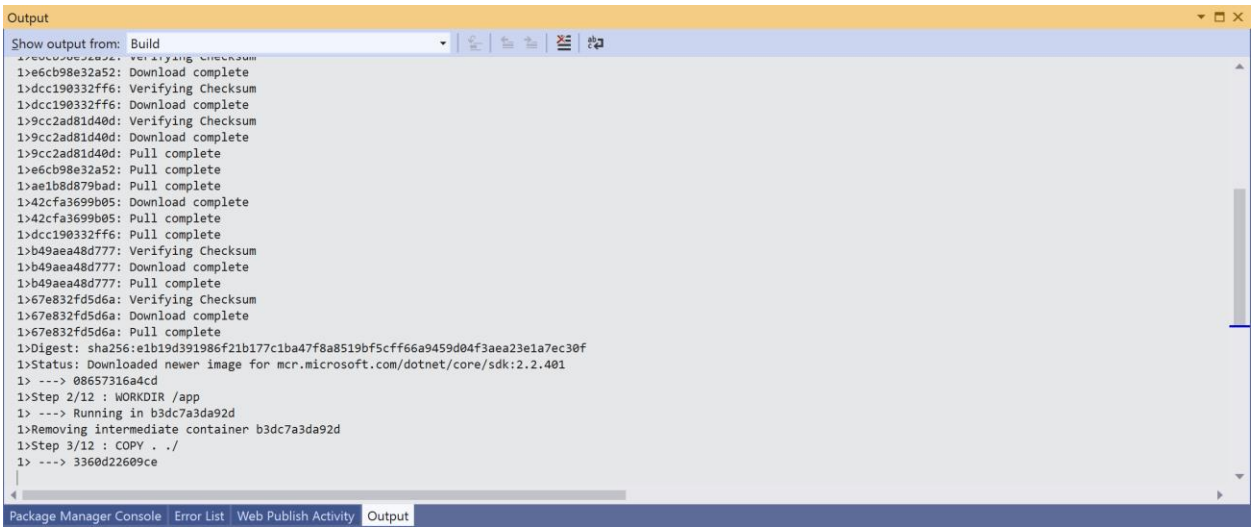
   the file should look like this:

Dockerfile | modules.json | ServiceCollectionExtensions.cs | Startup.cs | appsettings.json | Program.cs | SimplCommerce.WebHost.csp

```
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.EntityFrameworkC
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
```

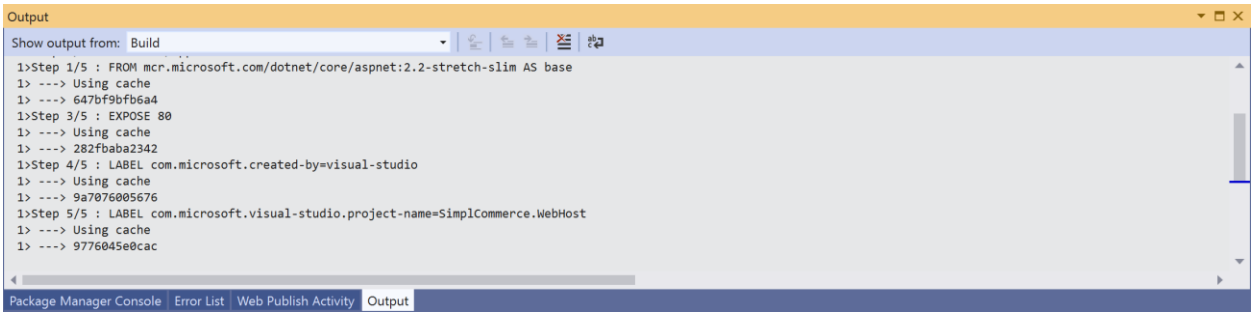
10. Next, start a debug process, by selecting Debug / Docker in the top menu

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



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.

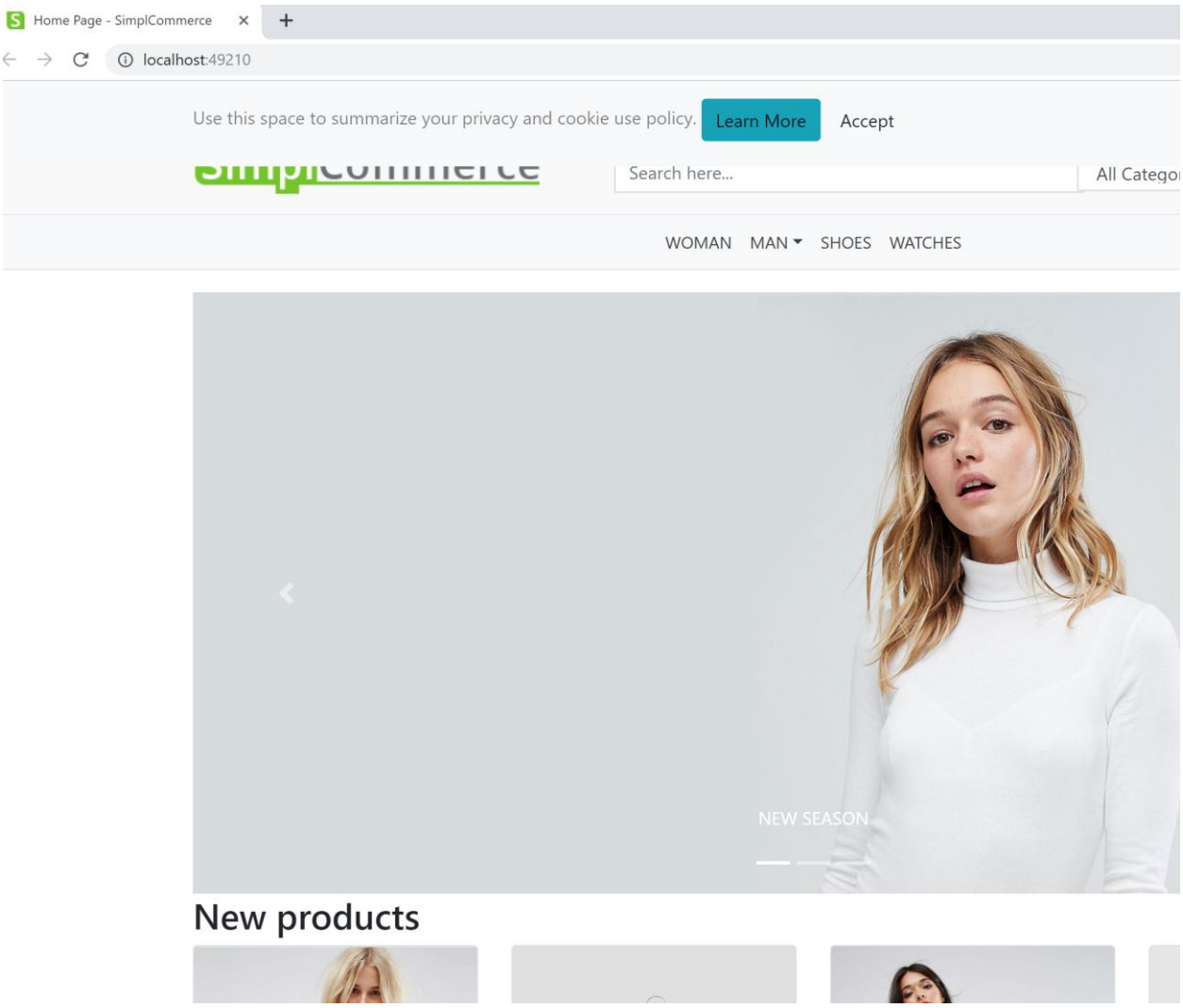


14. Some other output that is useful to know, is the notation Step X/5. Remember that the Dockerfile is providing the sequence on how the container gets build up. The different FROM statements in the Dockerfile reflect a step in the process.

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.

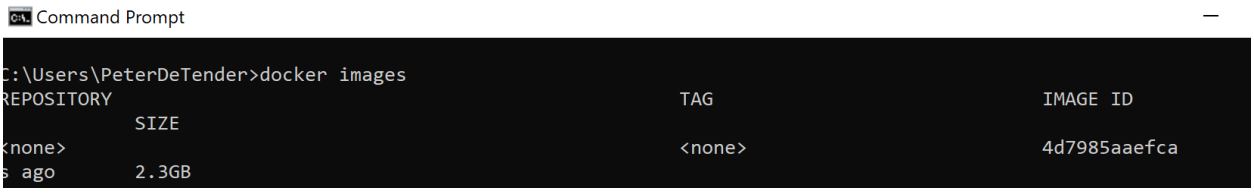


16. Notice that the application runs as "localhost", port 49210 (you might remember that the initial web application test in the browser in the previous lab, was using port 49209).

17. Validate the image is built successfully by running the following command in command prompt or PowerShell:

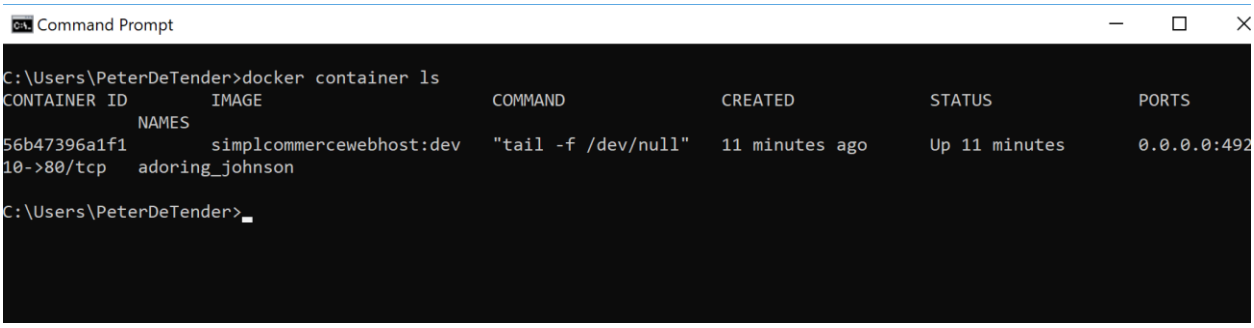
```
docker images
```

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



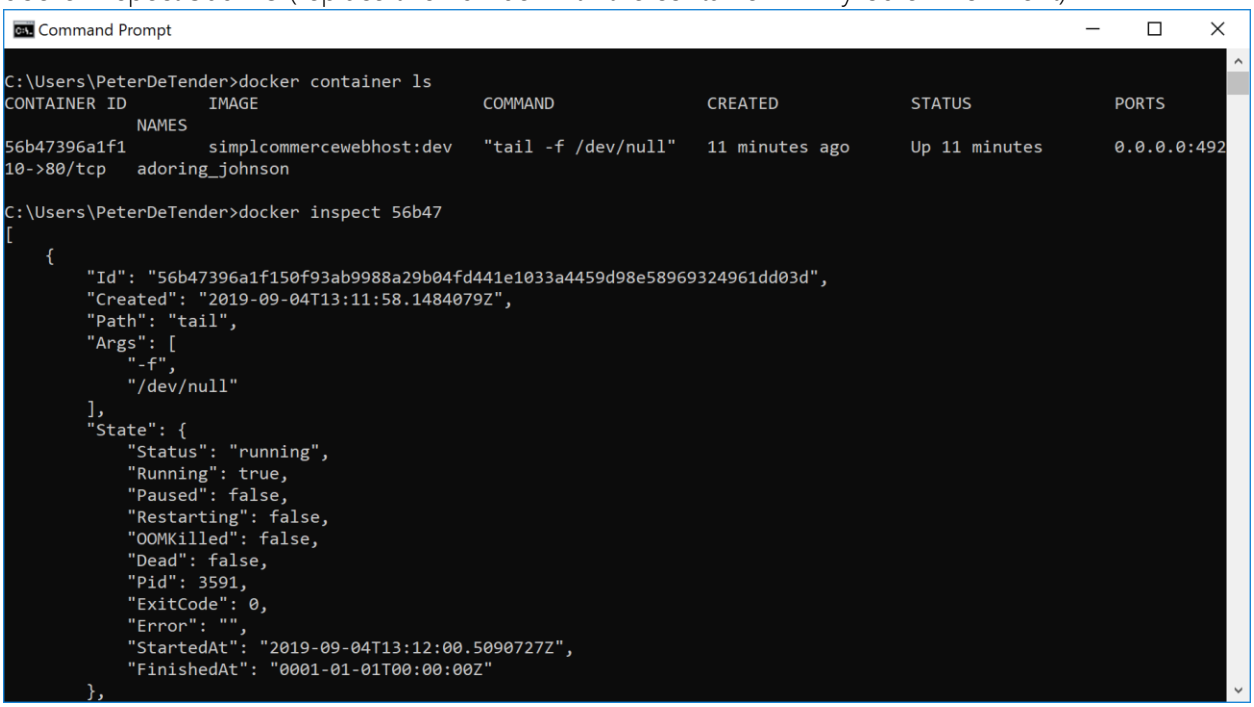
18. Next, run the following command, to get details related to the actual running container:

    docker container ls



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 yrou environment)



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"
```

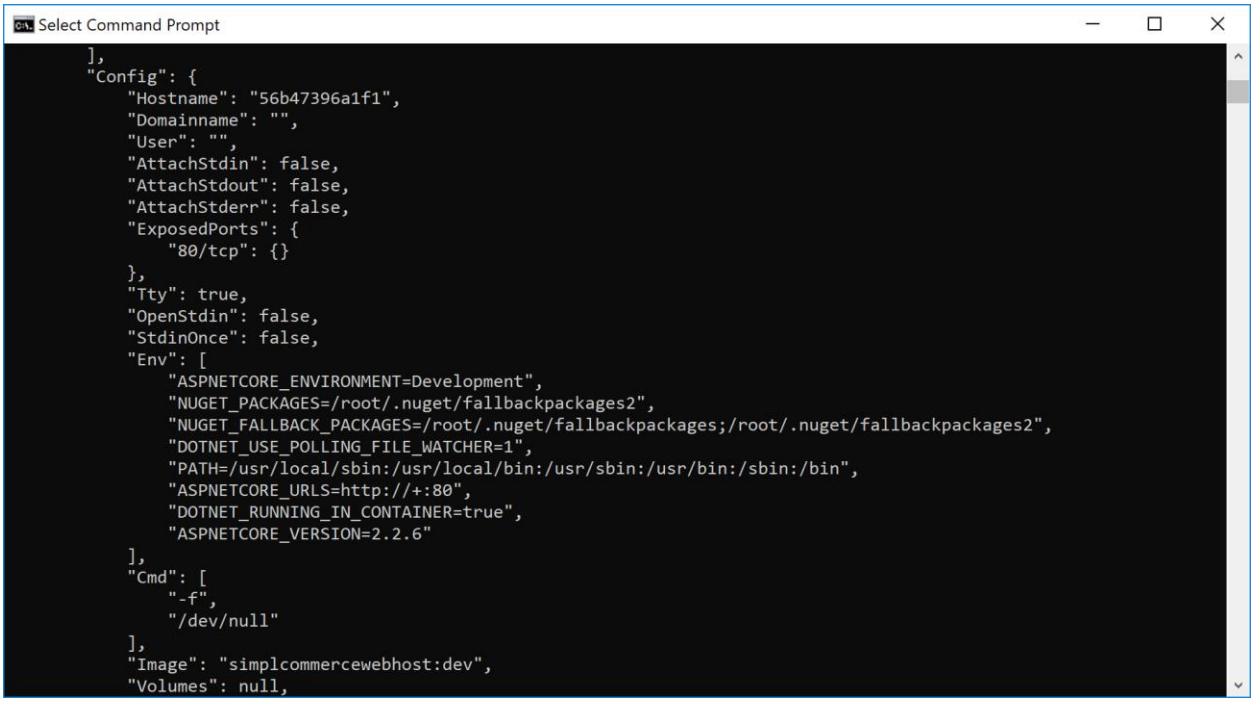

    which tells me that it is running, it has not restarted yet, and I can also see the actual Start Time and Date.

    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?

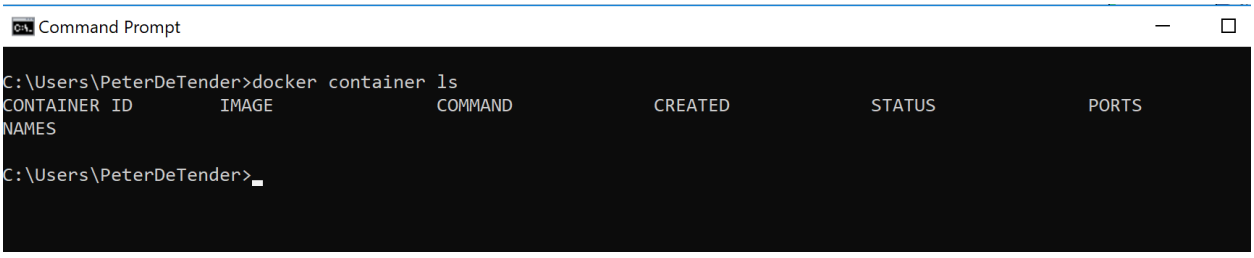


Telling me the hostname of the container, which is basically the Container ID, but also recognizing Environmental Variables, specifically for this container. (which means, you can push those whenever the container is starting up, like identifying this is a Dev environment, Production environment, Connection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

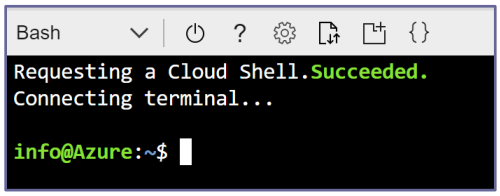


Task 4: Deploying Containers to Azure Container Registry and Azure Container Instance

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).

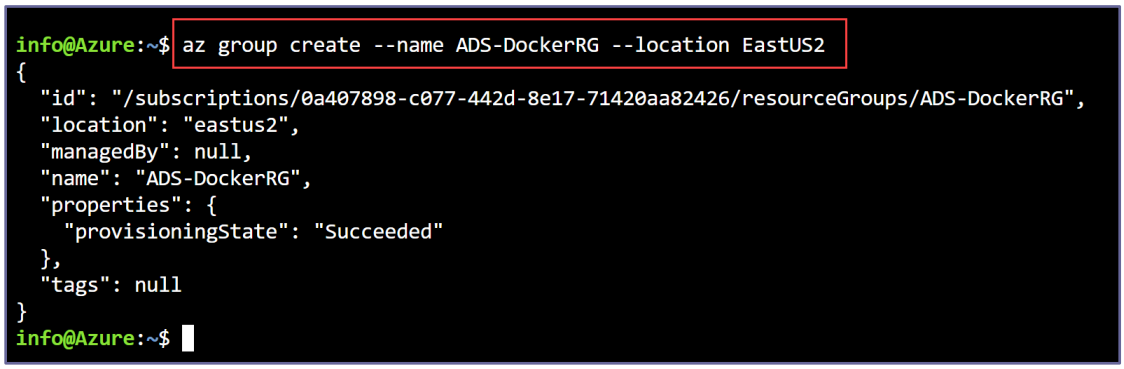
1. Log on to the Azure Portal, http://portal.azure.com, with your Azure admin credentials. From here, Open Cloud Shell

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:

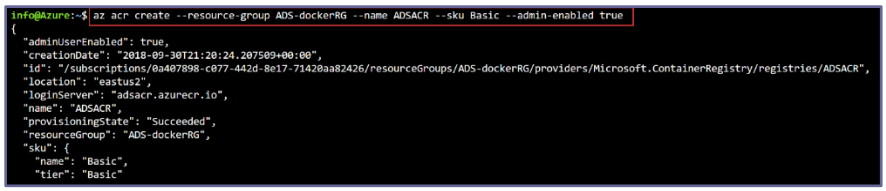
```
az group create --name [SUFFIX]-dockerRG --location <Azure Region
of choice>
```



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

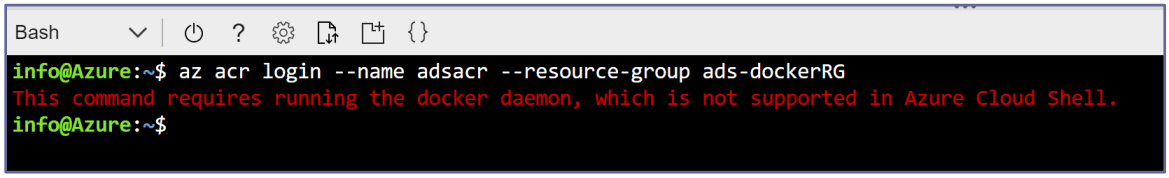
```
az acr create --resource-group [Suffix]-dockerRG --name
```

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



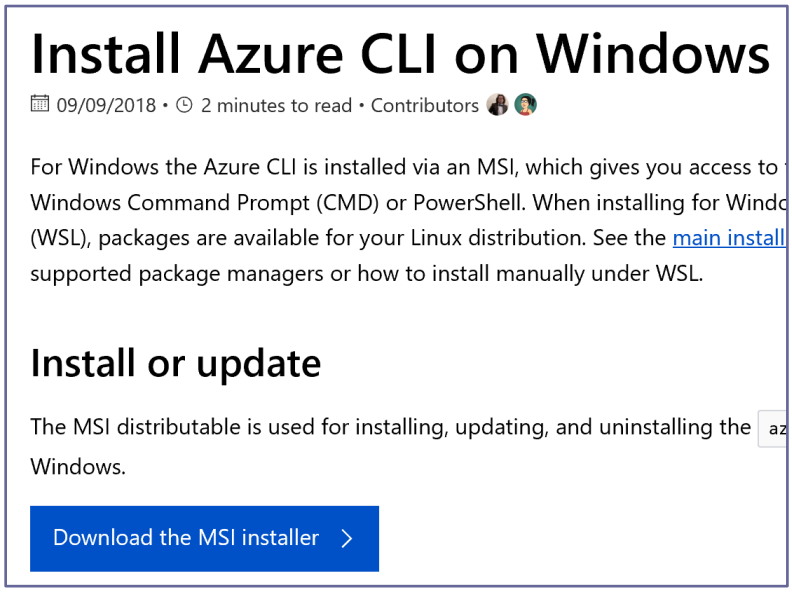
5. The next involves connecting to the Azure Container Registry we just created, and pushing our Docker image into it. This relies on the following command:

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



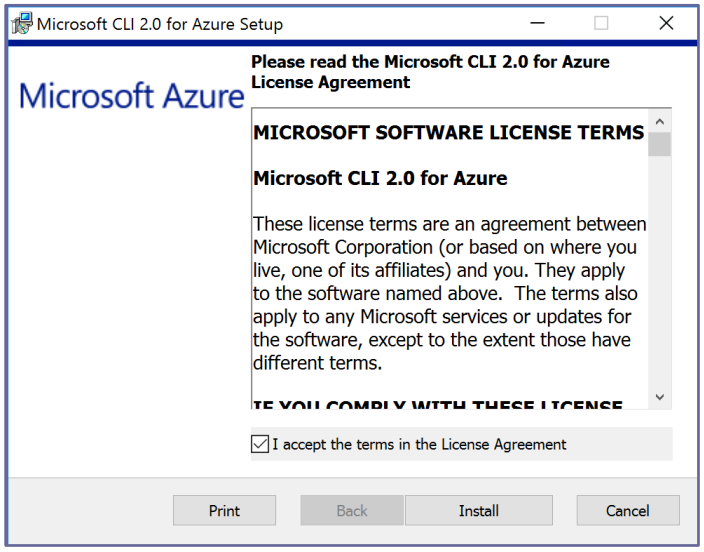
6. This means, we have to execute the remaining commands from our local lab-jumpVM, instead of the Azure Cloud Shell. We should install the Azure CLI for Windows using the following link: https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest

7. From the appearing web page, scroll down to the Download MSI installer button, and click it.



8. At the download prompt, choose SAVE; once the file is downloaded, select RUN

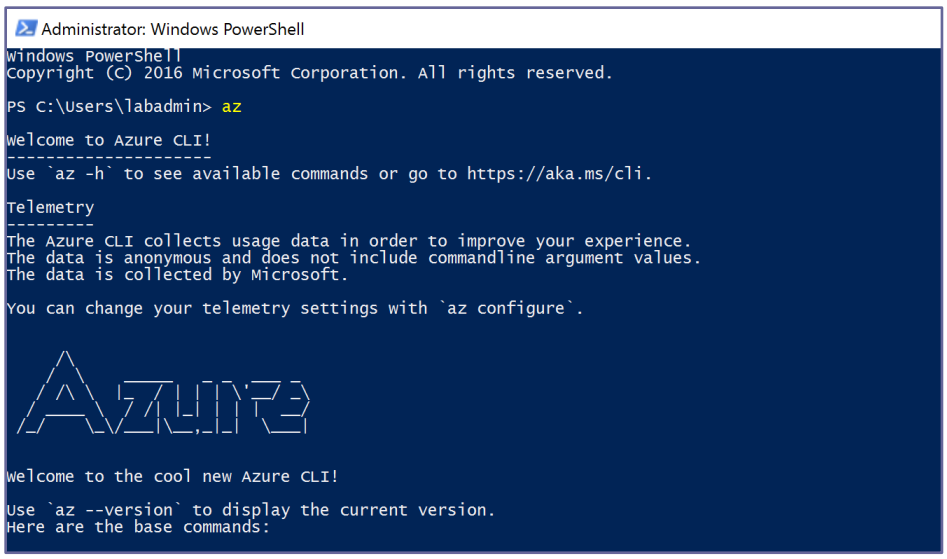
9. The Microsoft CLI 2.0 for Azure Setup Installer launches



10. Press the Install button to continue. Wait for the installation to finish successfully.

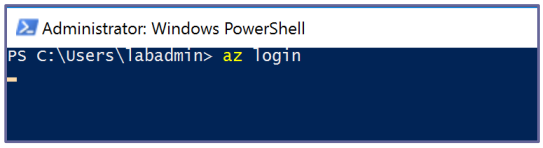
11. To validate the Azure CLI is installed fine, open a new PowerShell window, and initiate the following command:

    az

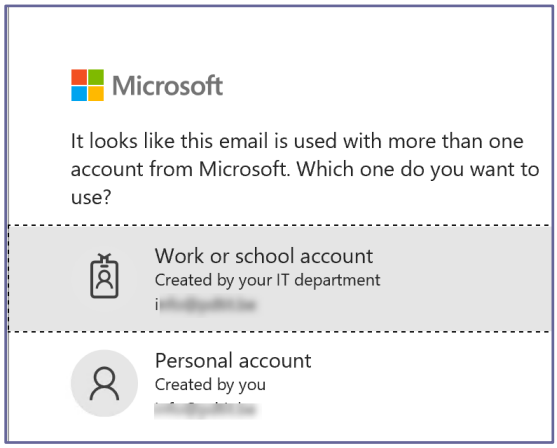


12. This confirms Azure CLI 2.0 is running as expected. We can continue with our Azure Container Registry creation process. But first, we need to "authenticate" our session to Azure, by running the following command:

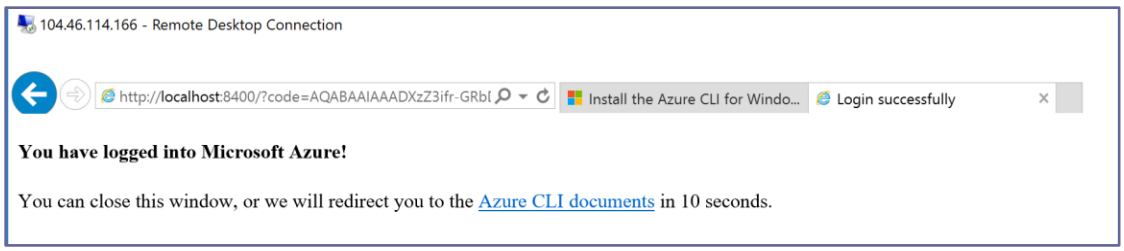
    az login



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

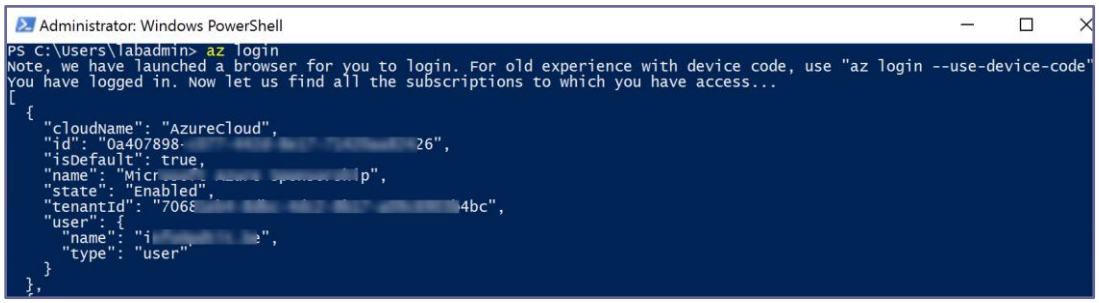


14. After successful login, the following information is displayed:



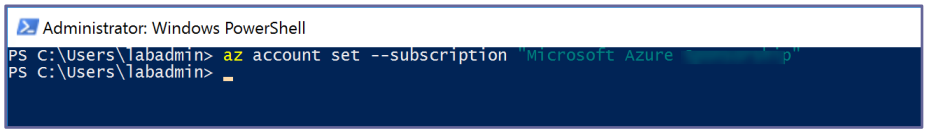
15. You can close the internet browser.

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:



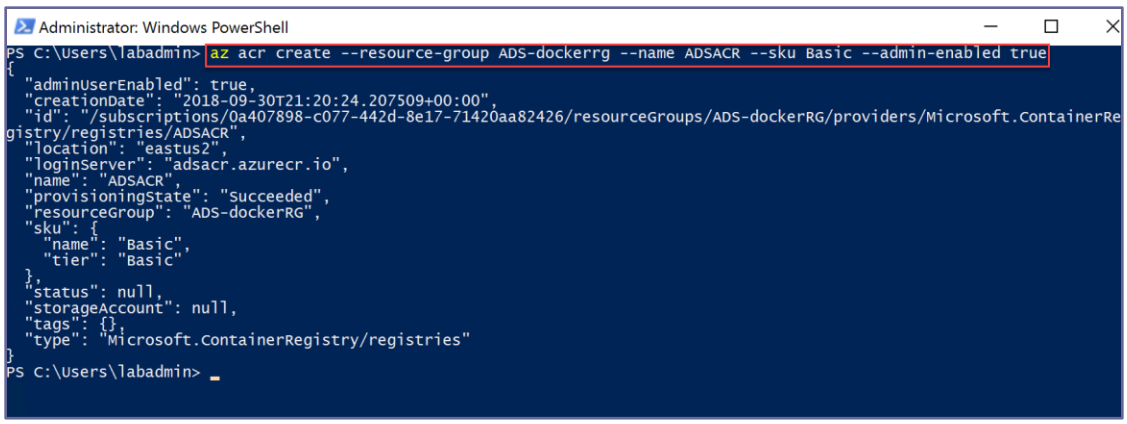
17. If you should have multiple Azure subscriptions linked to the same Azure admin credentials, run the following AZ CLI command to guarantee you are working in the correct subscription:

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

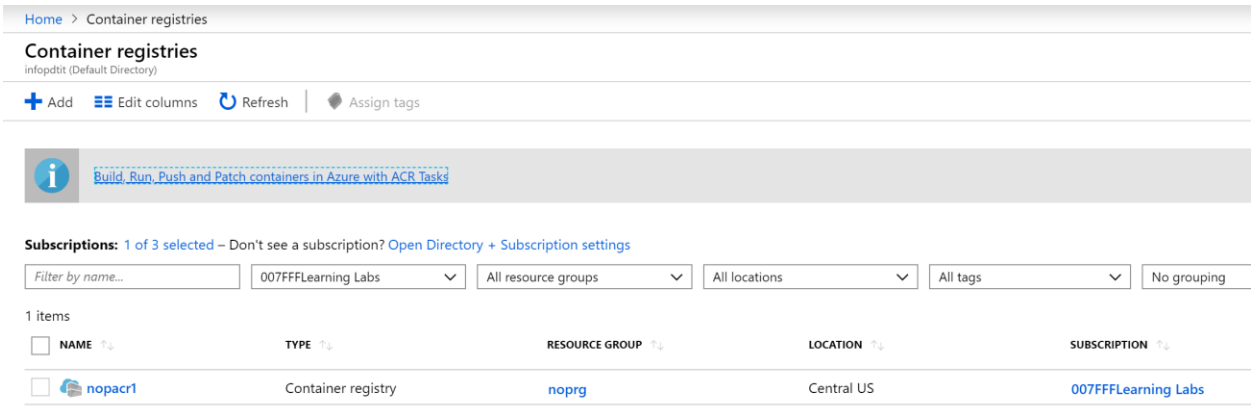


18. Let's try to redo our Azure Container Registry process, by executing the following command:

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

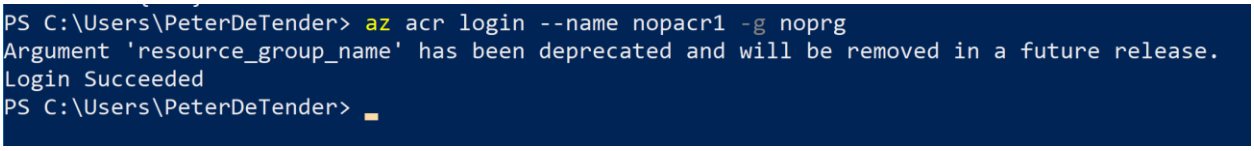


19. While the JSON output is here, you can also validate from the Azure Portal



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>
```



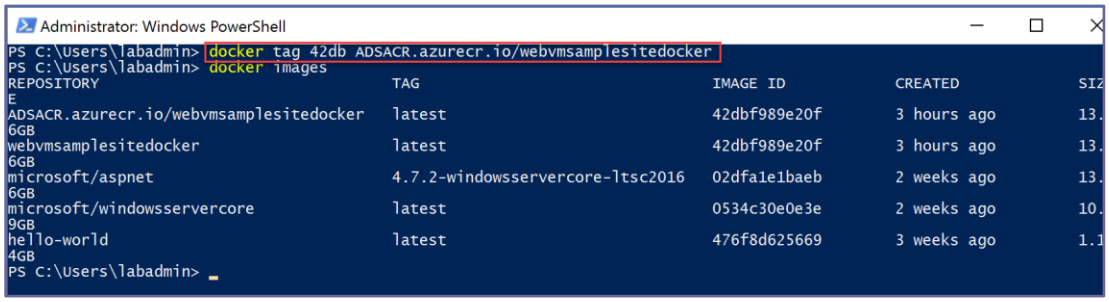
Task 5: Pushing a Docker image into an Azure Container Registry

1. 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)

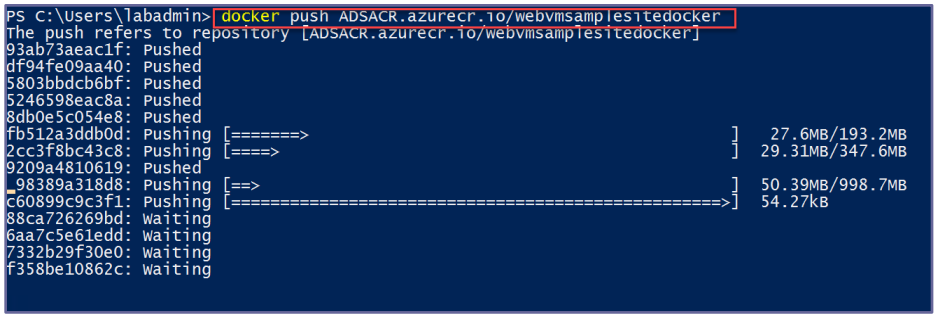
   `docker tag 42db [SUFFIX]ACR.azurecr.io/<nameyouwanttogive>`

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



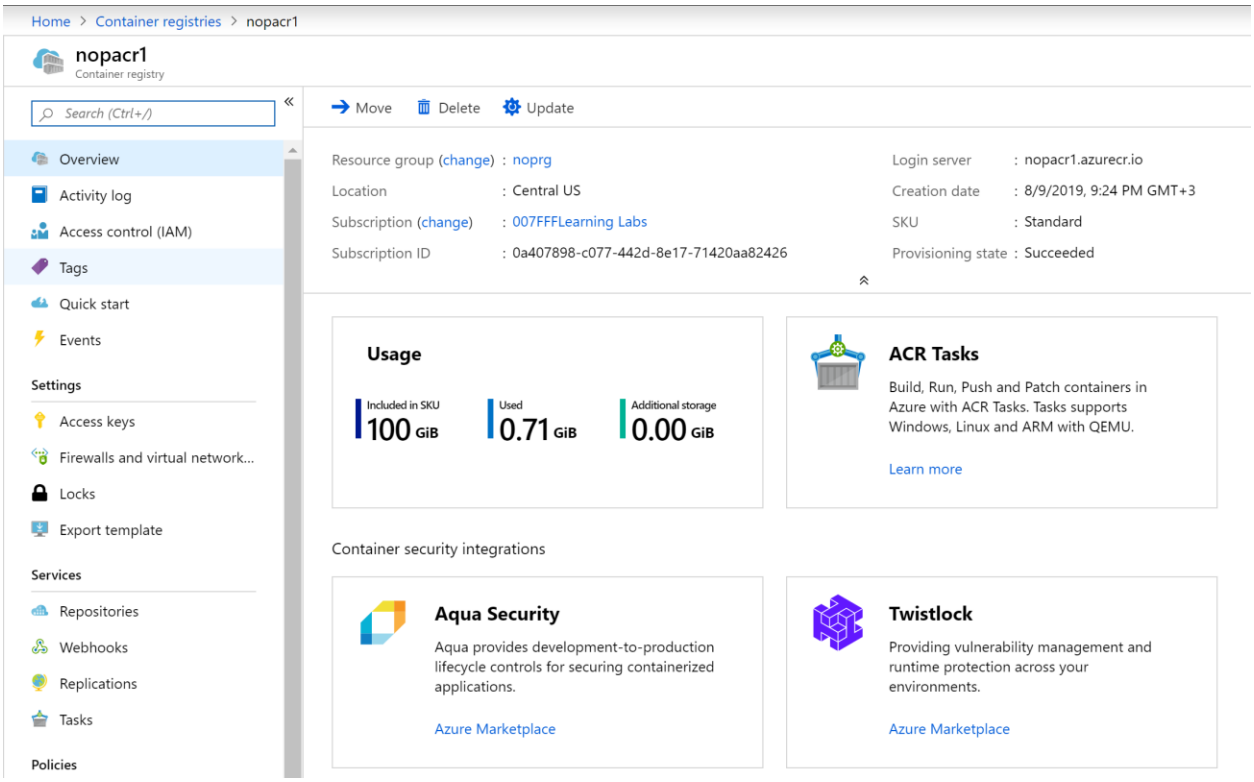
2. Execute the following command to upload this image to the Azure Container Registry:

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

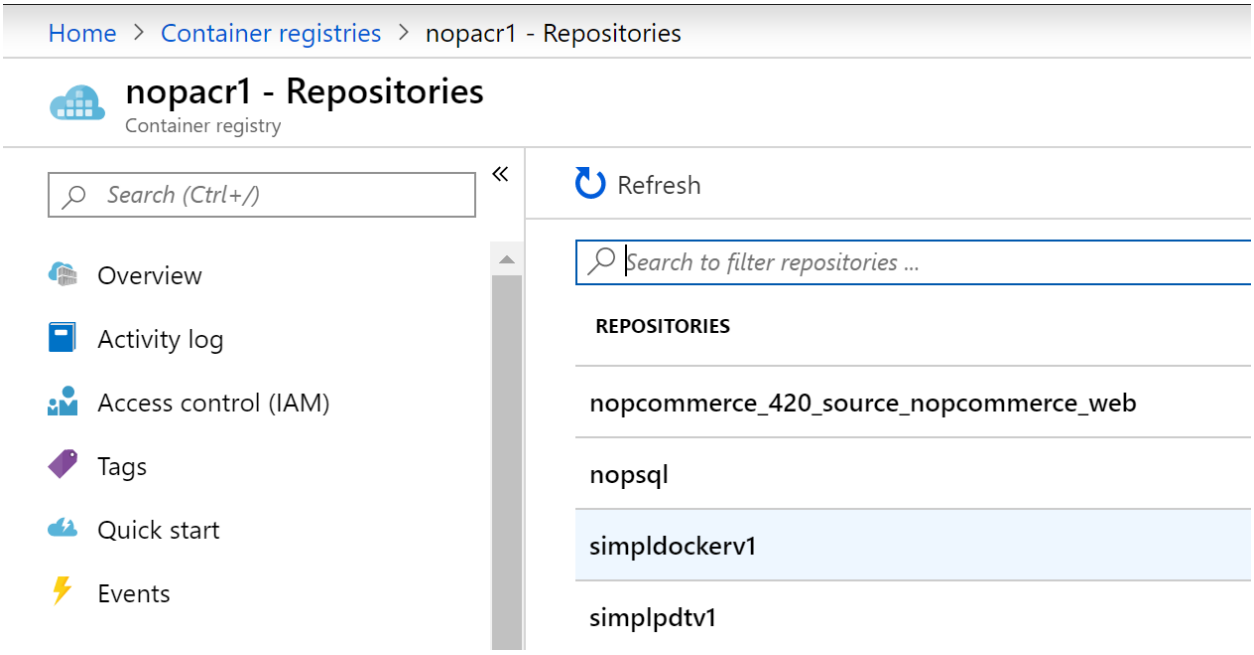


3. Wait for this process to complete successfully; depending on internet connection speed, this might take some time.

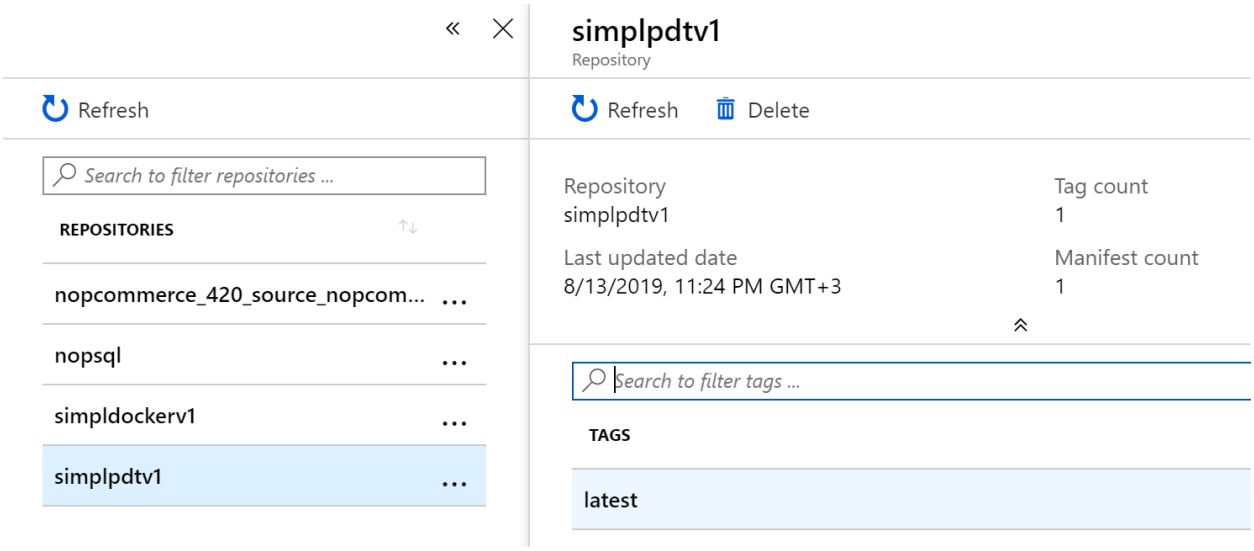
4. From the Azure Portal | All Services | Azure Container Registries | select the ACR you created earlier.



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

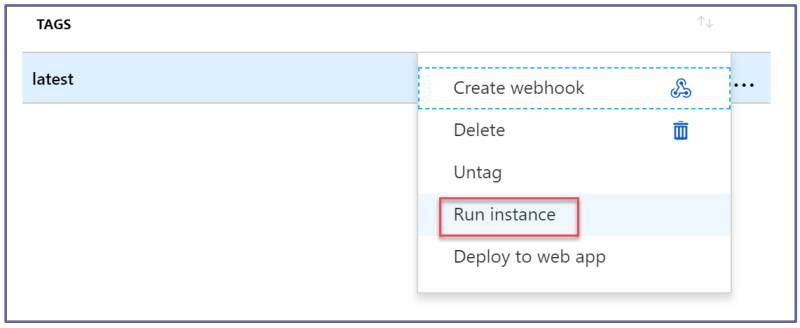


6. Click the <yourcontainername> repository, which opens the specific details for this image, exposing its version:



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

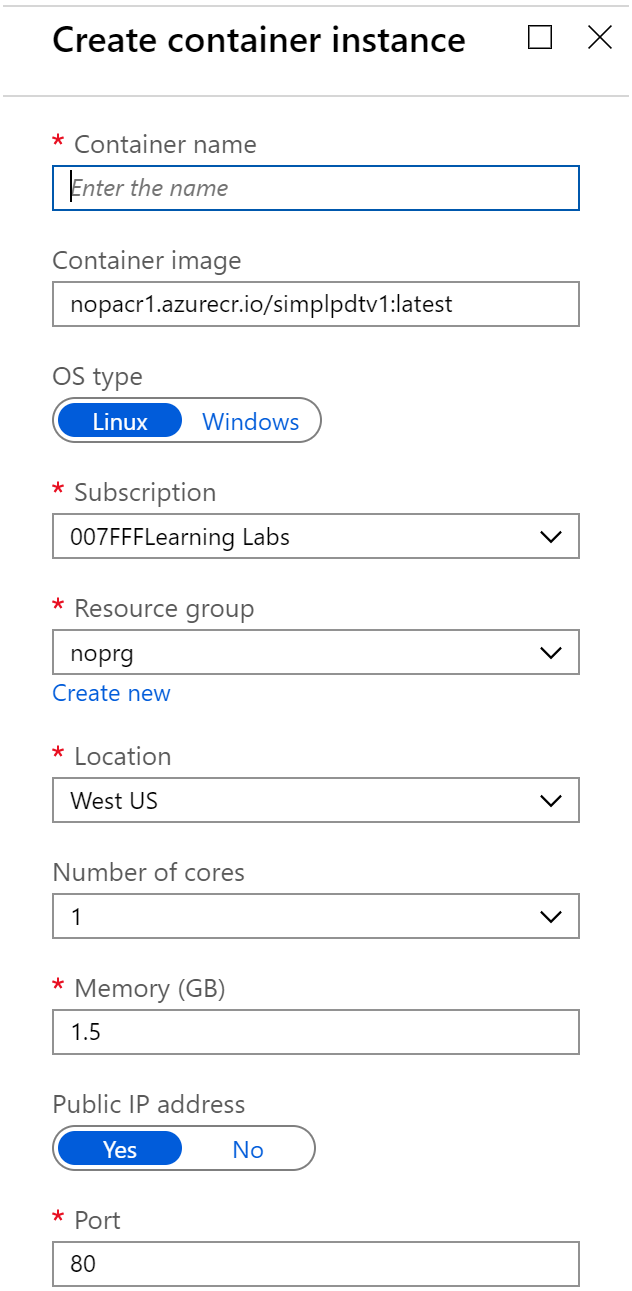
1. Click the ... next to latest, and choose Run Instance



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

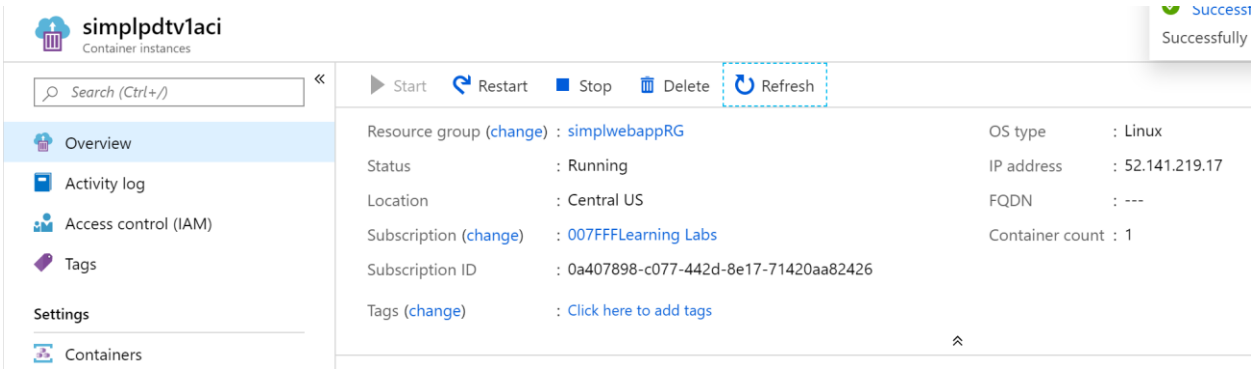
Leave all other settings unchanged (1 core, 1,5GB memory, Public IP address YES and Port 80)



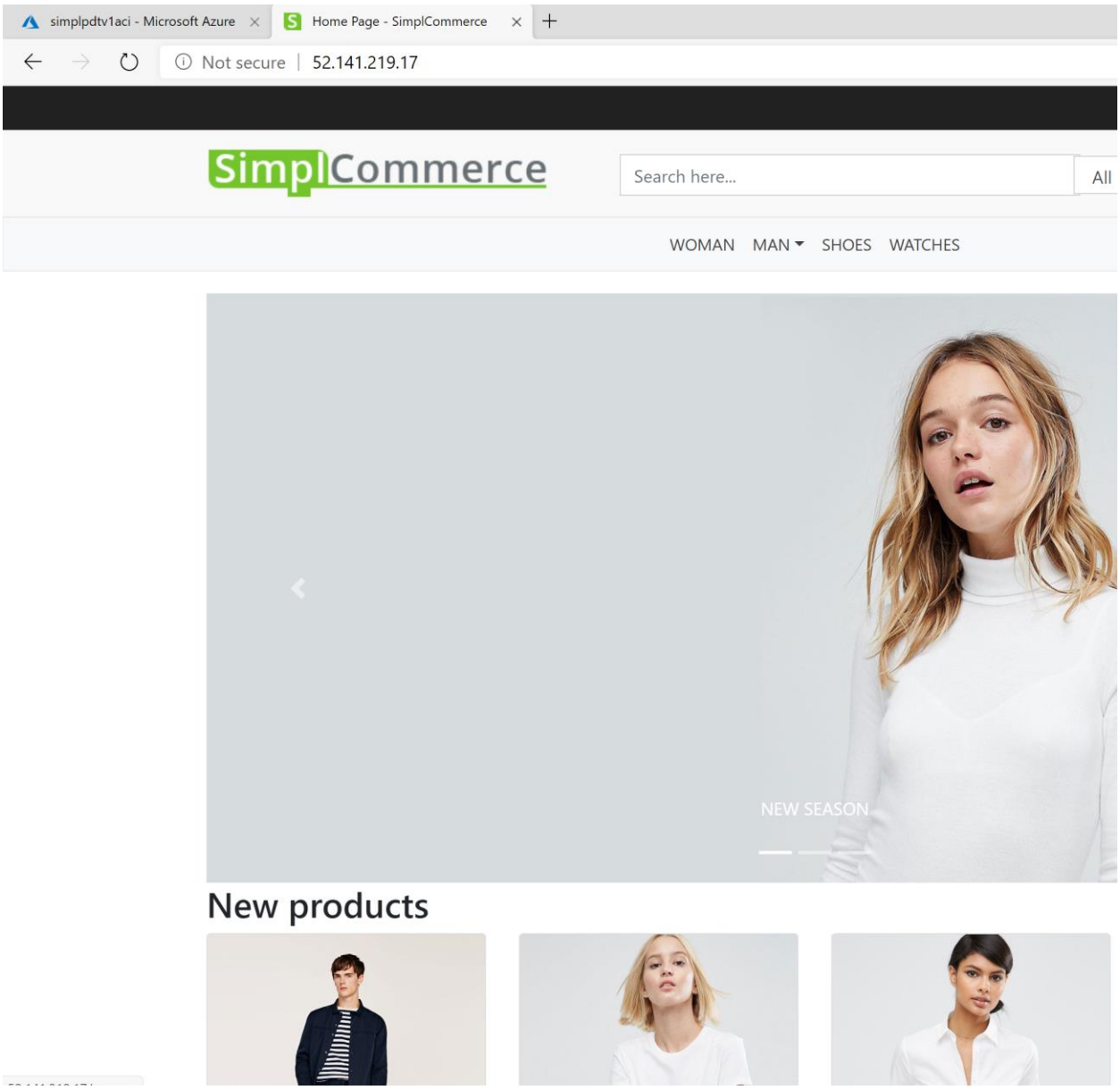
3. Press **OK** to have the Container Instance created.

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.



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.

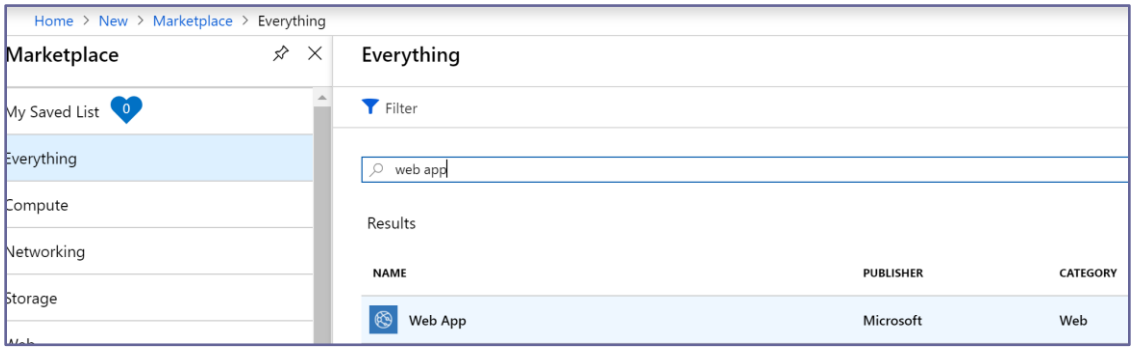


7. The sample website starts successfully, and again has connectivity to the underlying SQL Azure database. Notice the name of the Azure Container Instance is visible too.

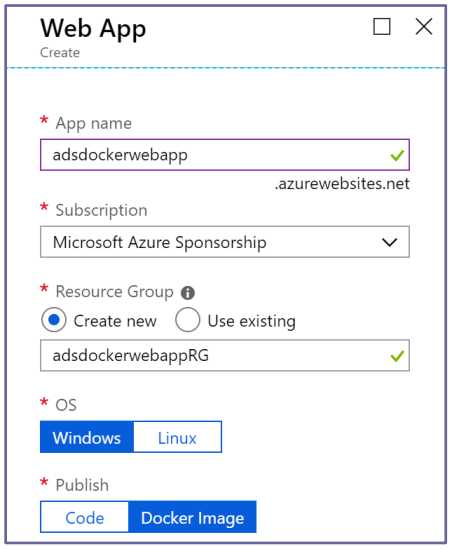
Task 7: Deploy a Container using Azure Web Apps

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.

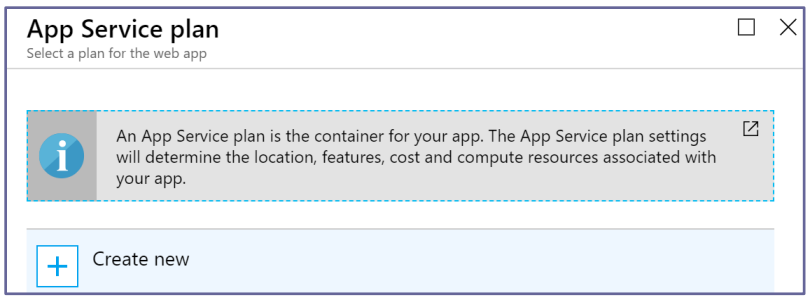
1. From the Azure Portal | Create New Resource | Web App.



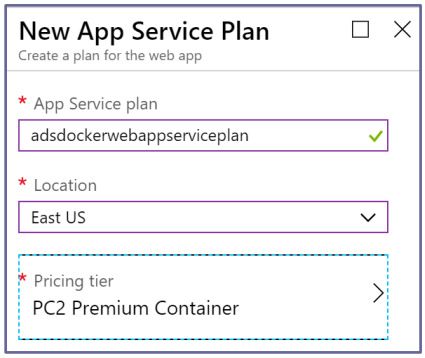
2. Press the Create button to open the Web App blade. Complete the required parameters as follows:
   - App Name: [SUFFIX]dockerwebapp.azurewebsites.net
   - Resource Group: Create New | [SUFFIX]dockerwebappRG
   - OS: Windows
   - Publish: Docker Image



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

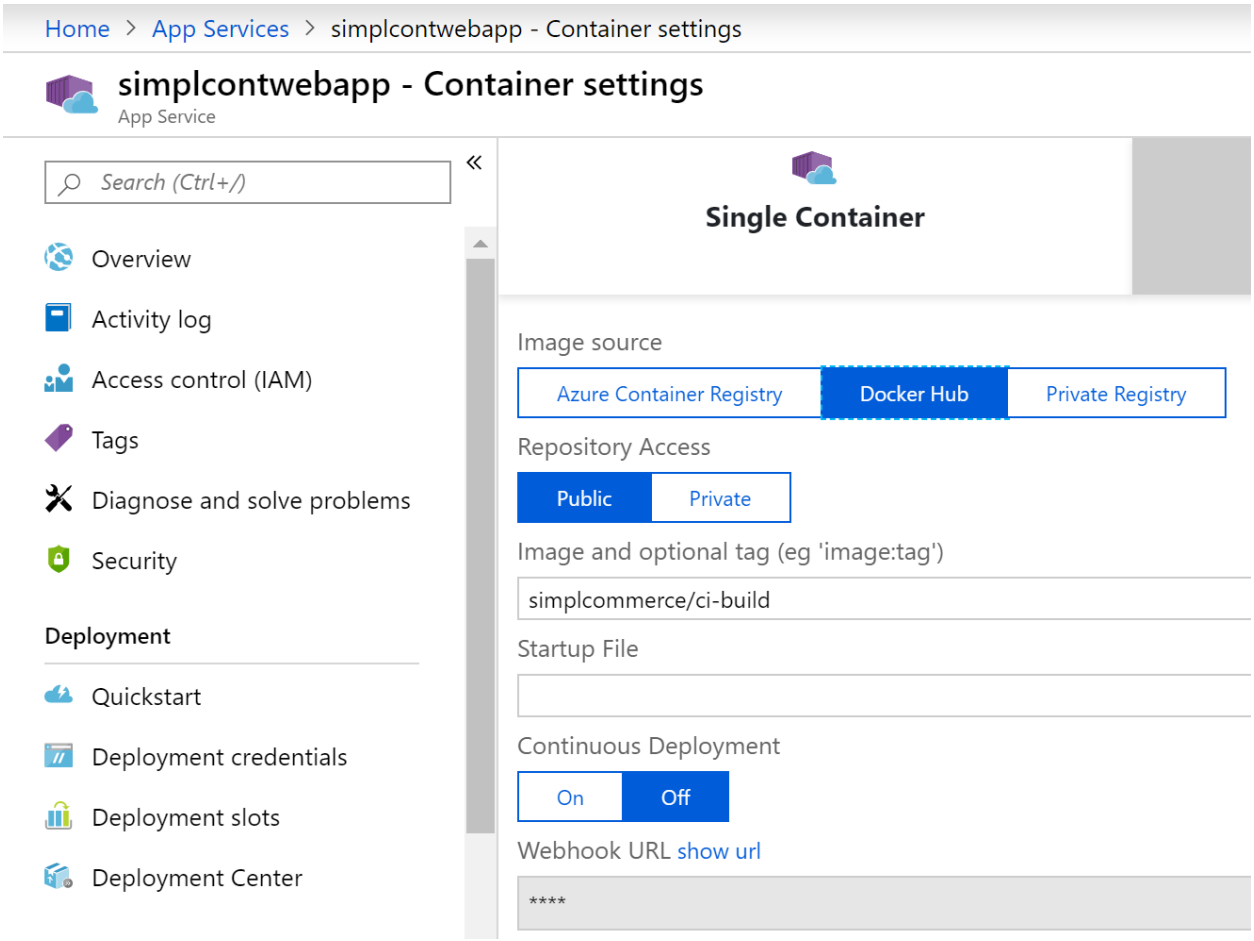


   And confirm the plan with OK.

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

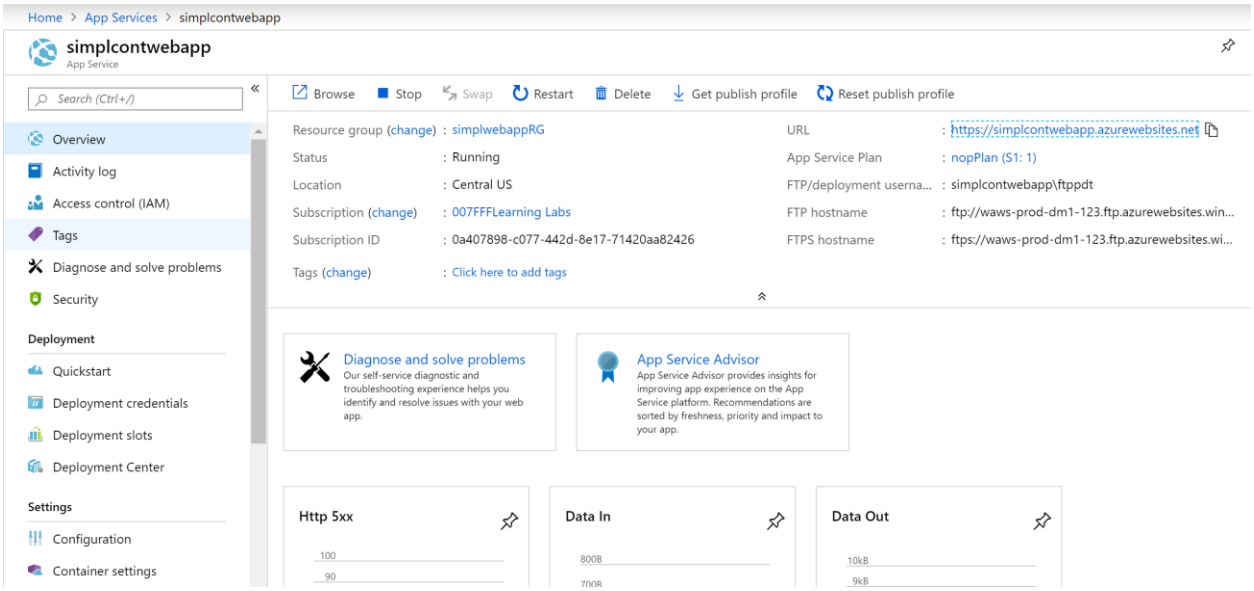


6. Confirm the creation by pressing the Apply button.

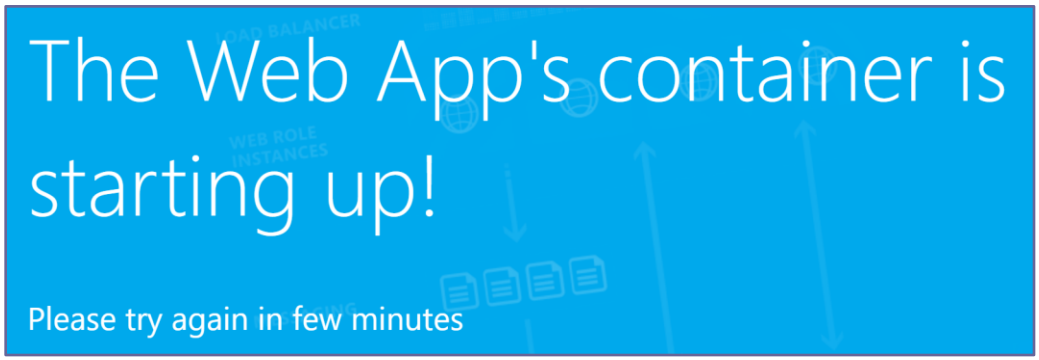
7. Press the Create button to start the deployment of the Azure Web App for Containers.

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:



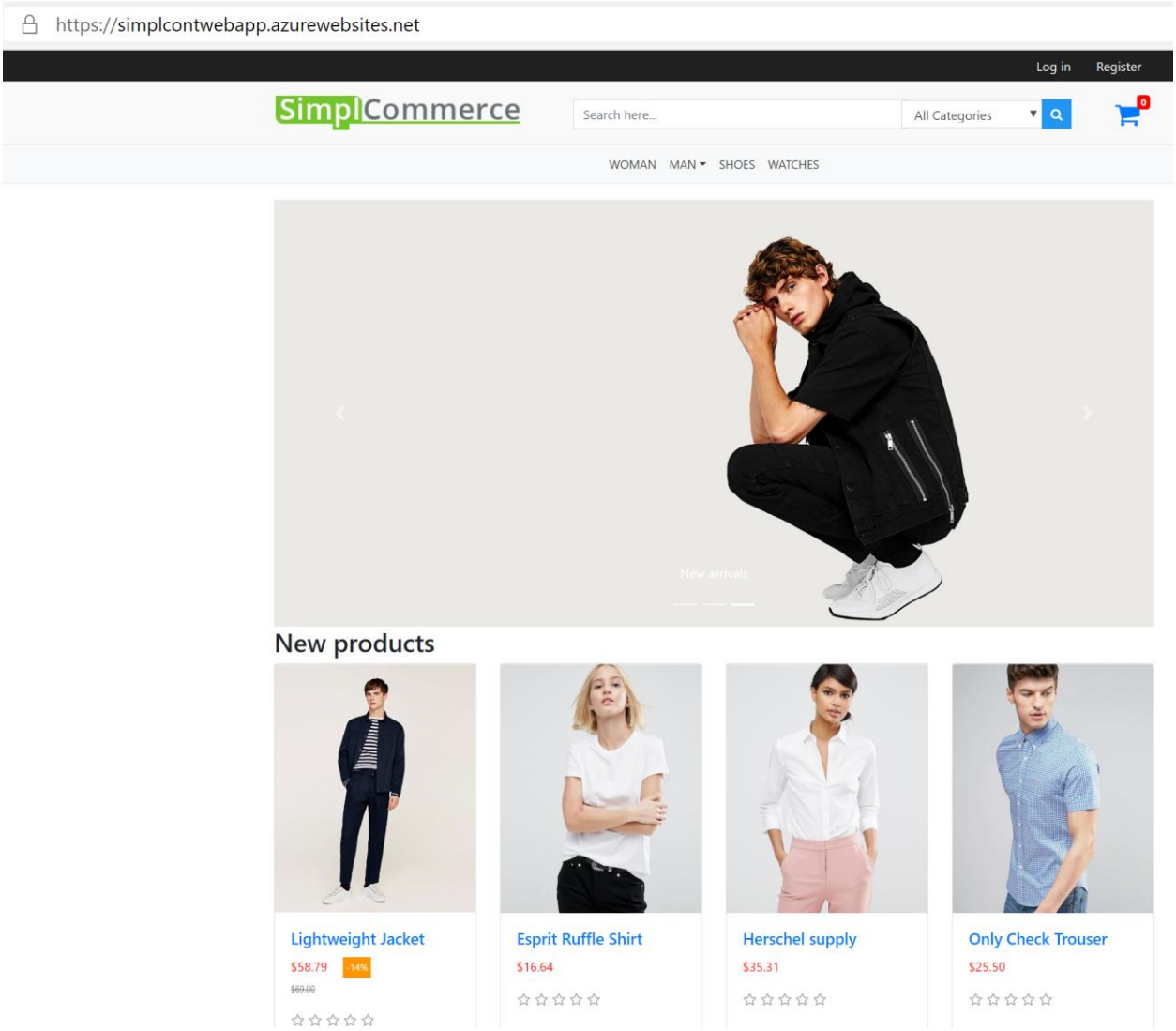
10. **Copy** the URL and paste it in your internet browser. Note the message about the container starting up:



11. **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.

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

13. If you **go back to your browser window** with the "container starting message" and **refresh it, it opens up the c**ontainerized web application as expected.



14. 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