# Lab 3: Containerizing an ASP.NET web application with Docker

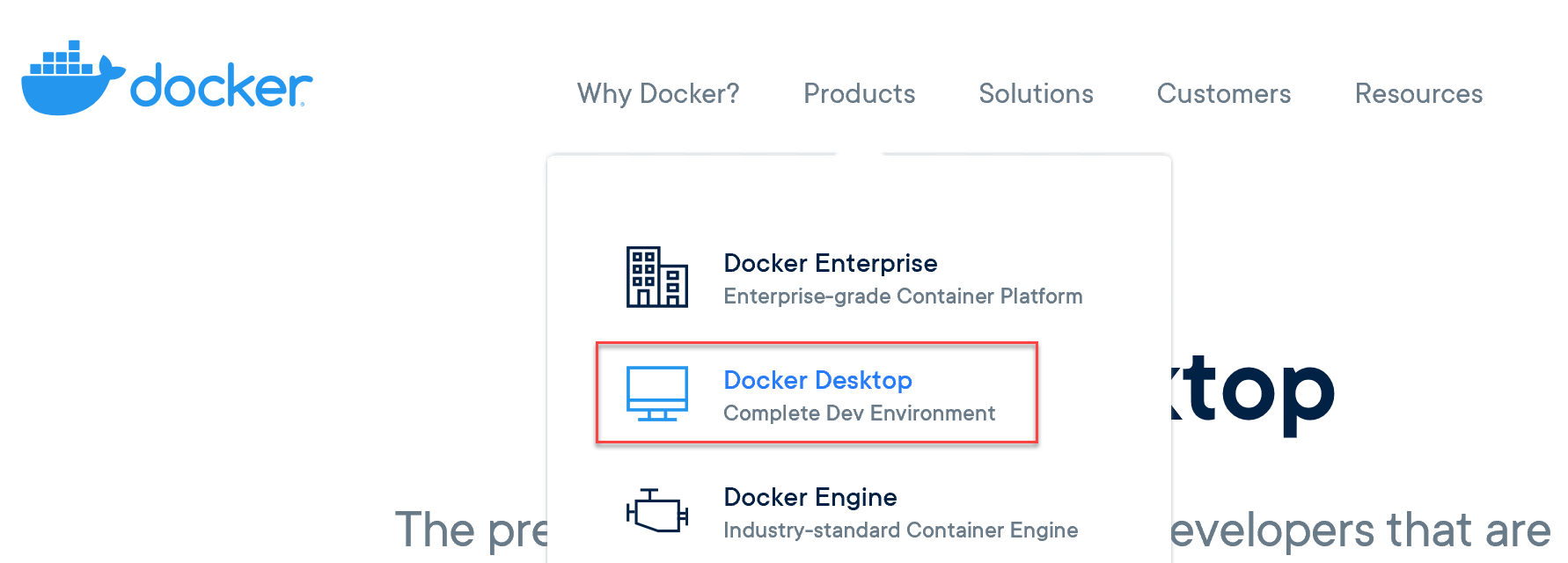
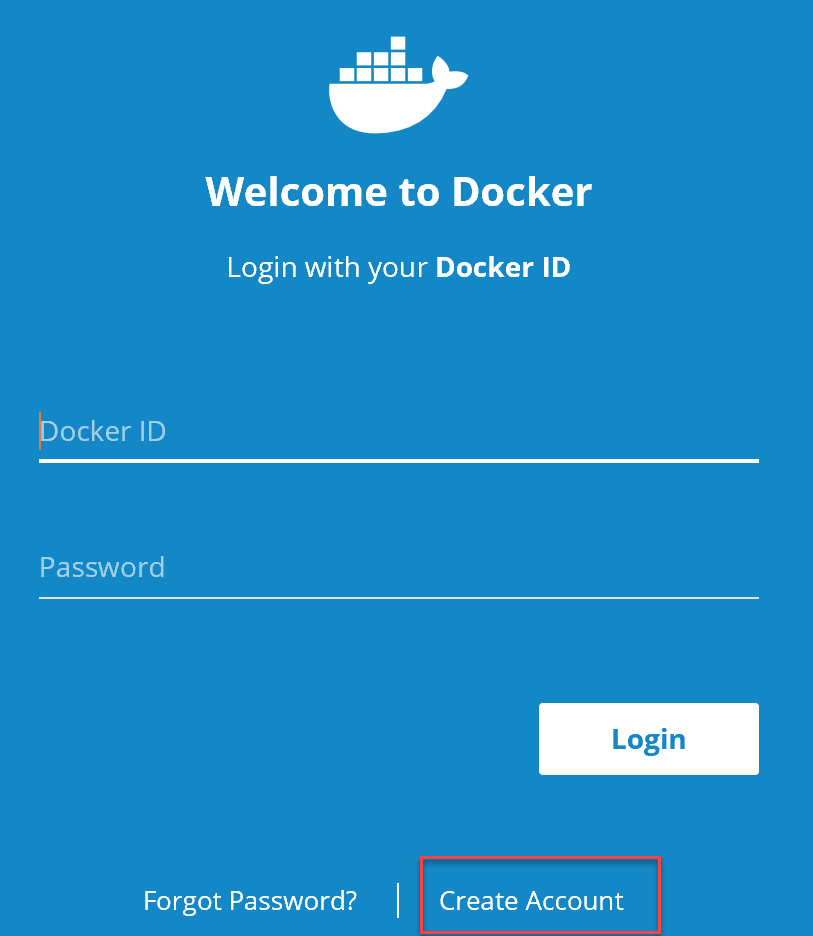
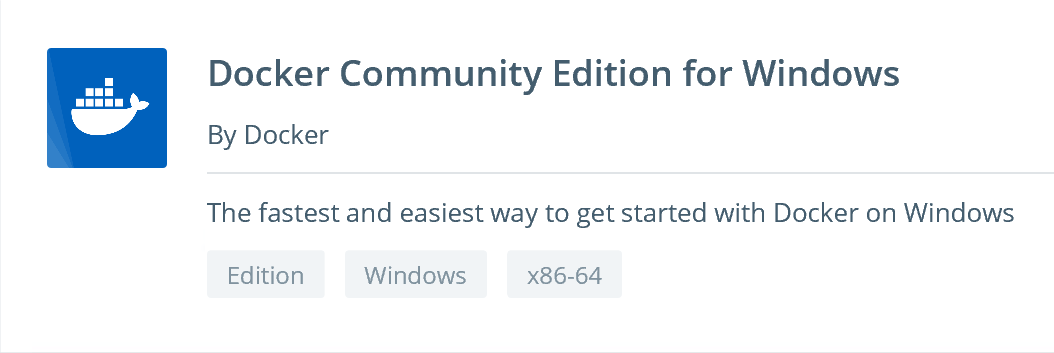
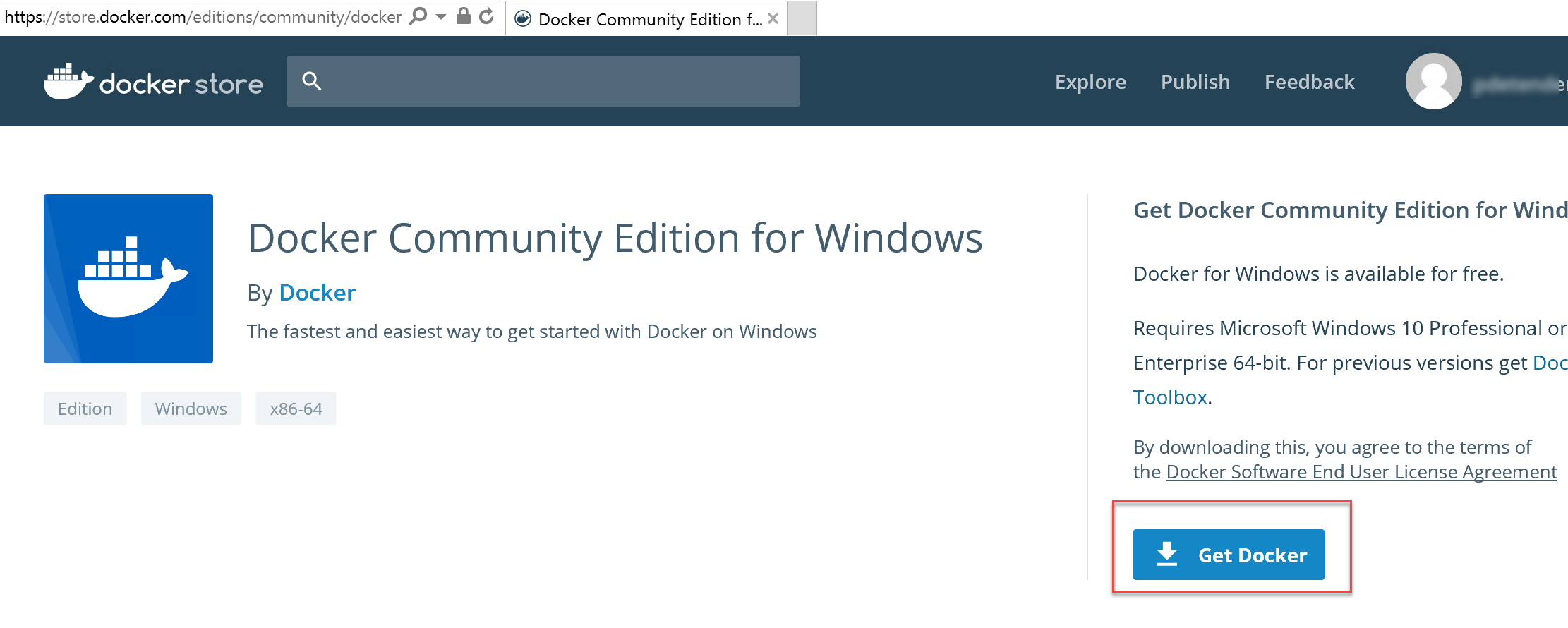
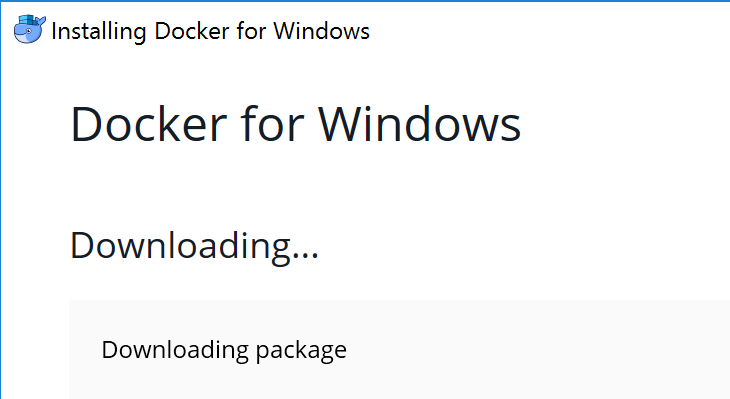
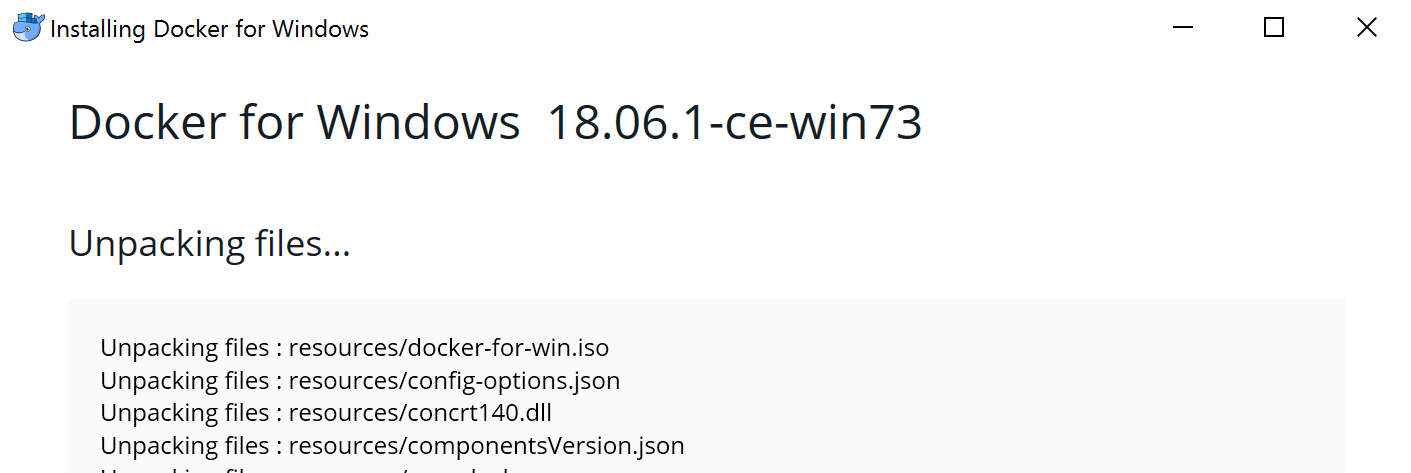
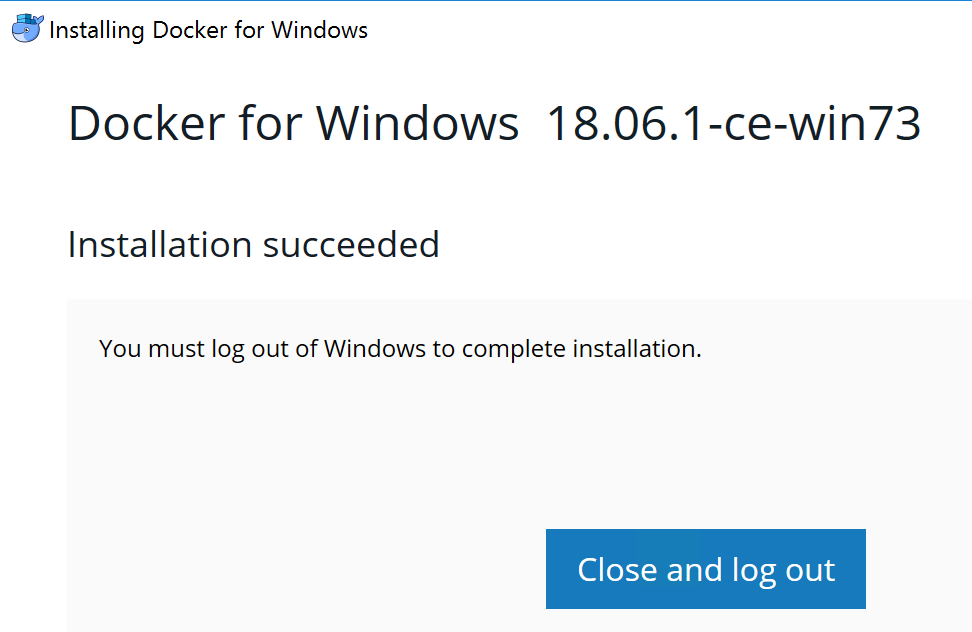
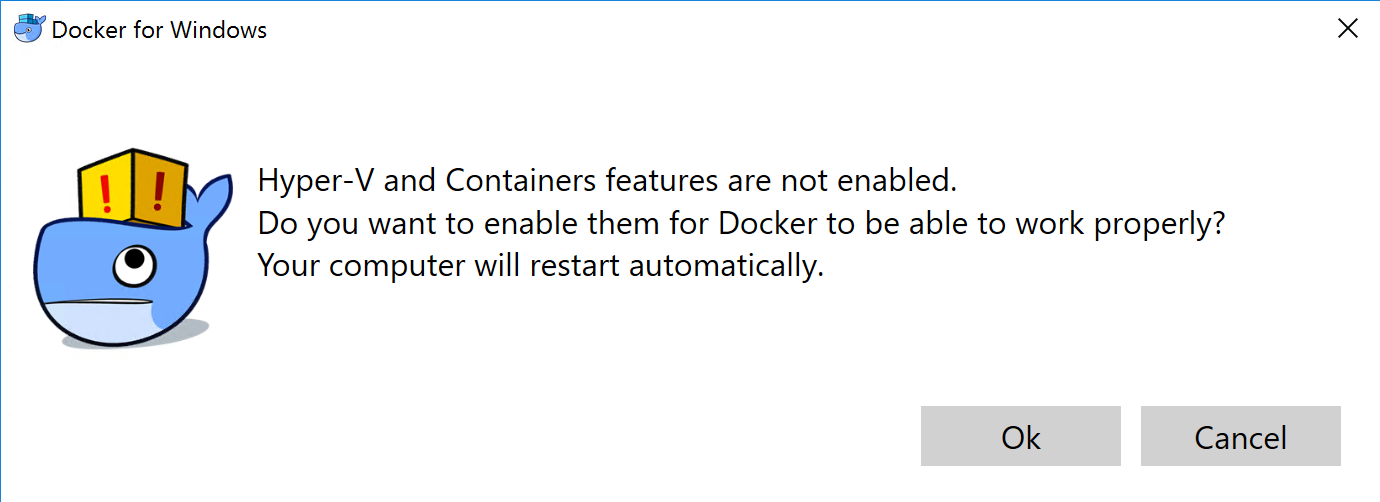
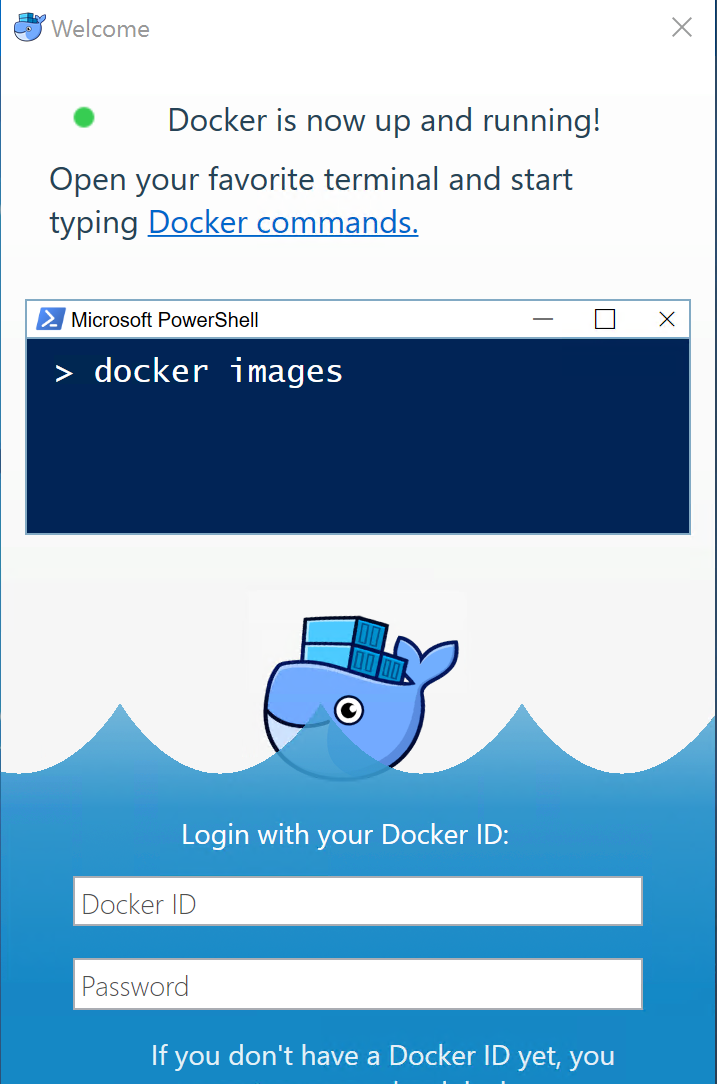
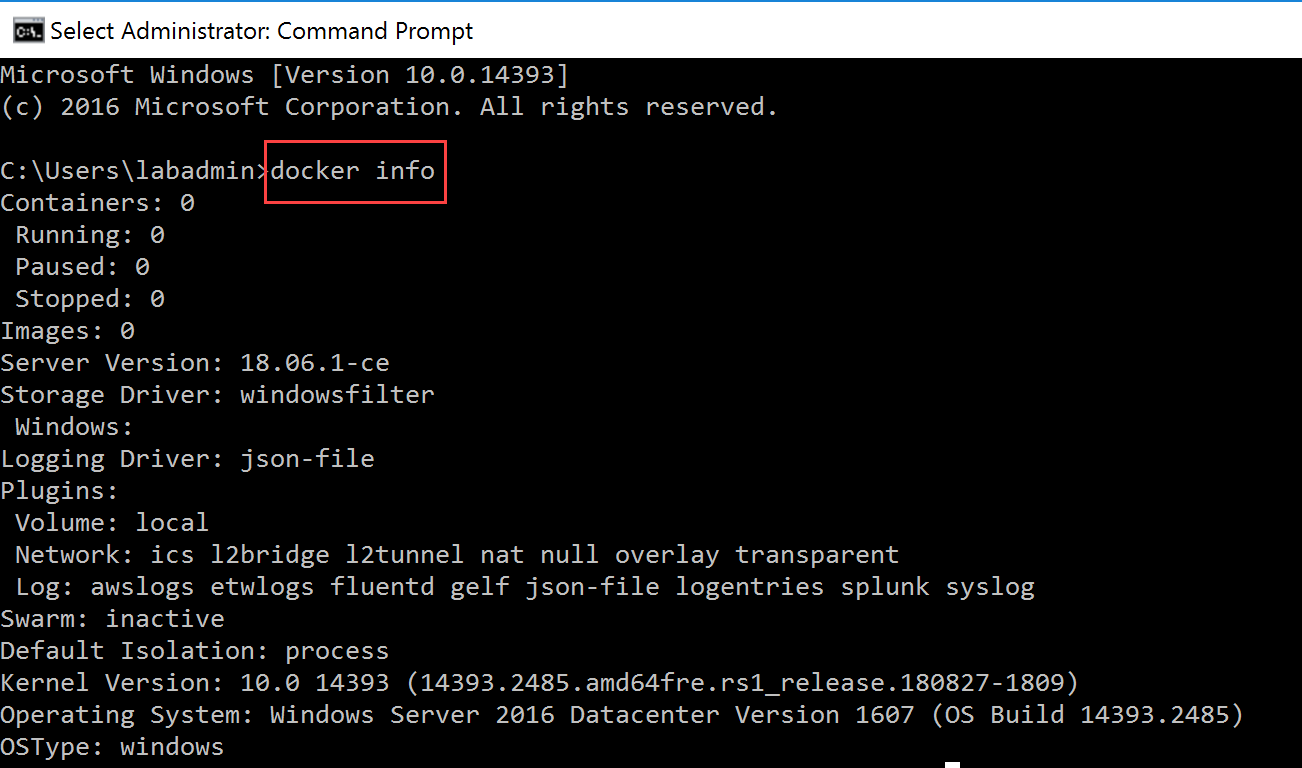
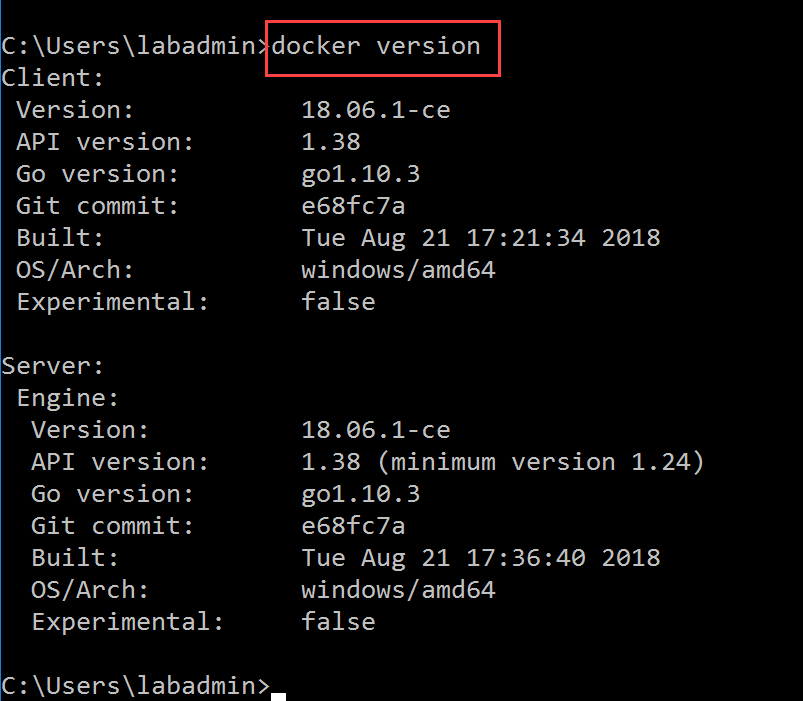
## What you will learn

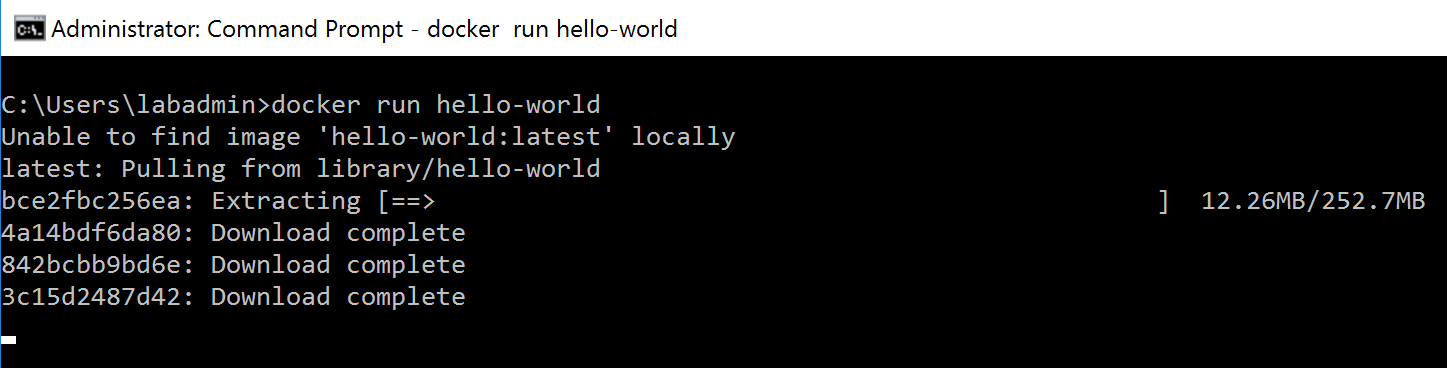
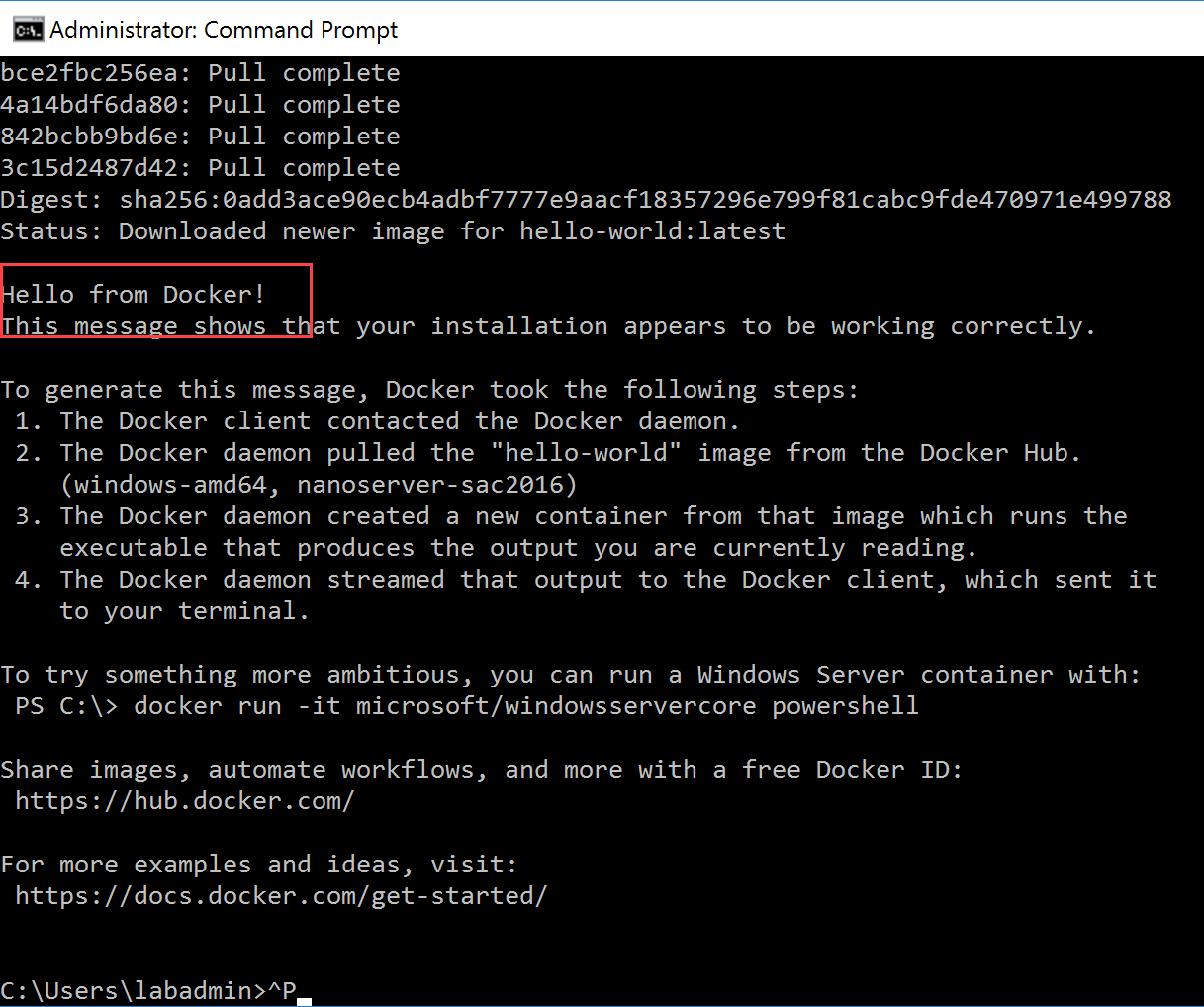
In this lab, we focus on Docker for Windows. Starting with installing the Docker for Windows application, you learn the basics of Docker commands using the Docker Command Line interface. Next, students learn how to ‘Dockerize’ the legacy ASP.NET code that has been used in former labs. Finally, you learn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

## Time Estimate

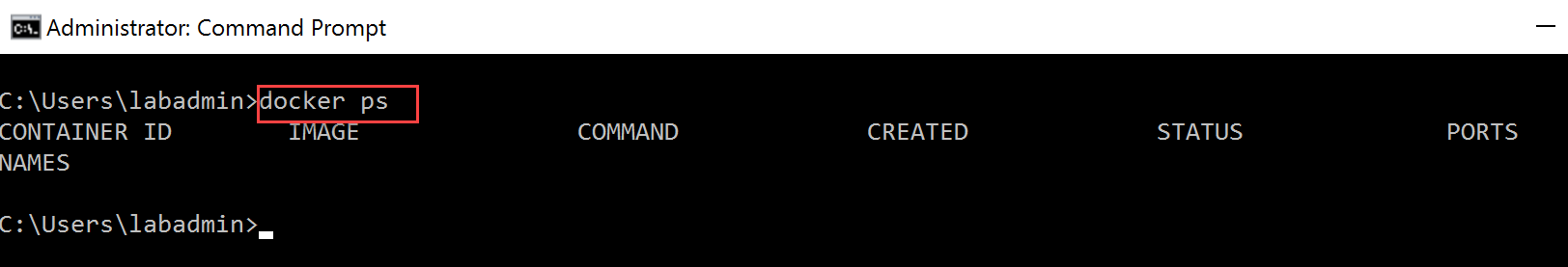
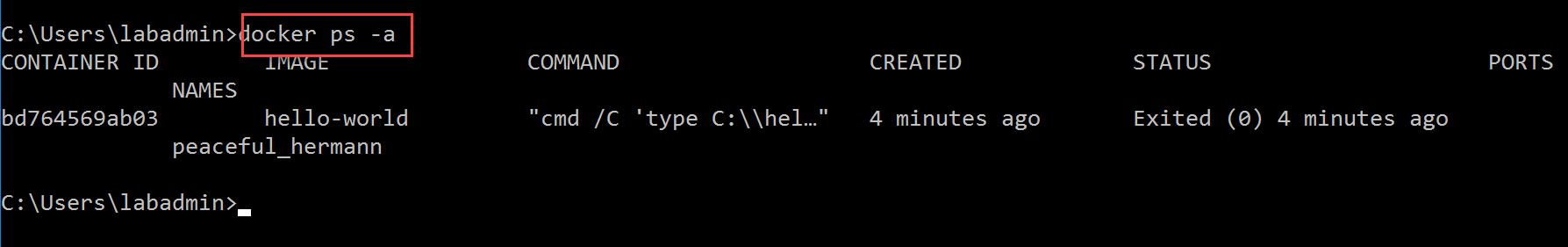
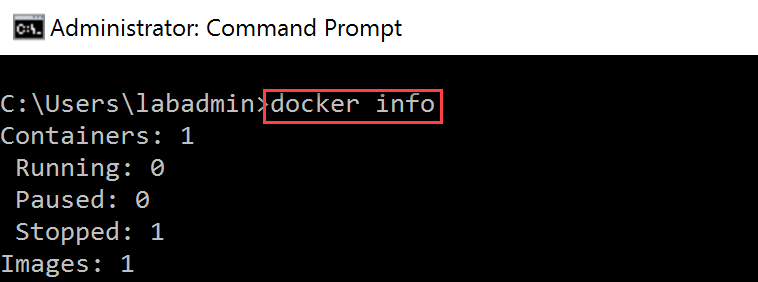
This lab duration is estimated 60 min.

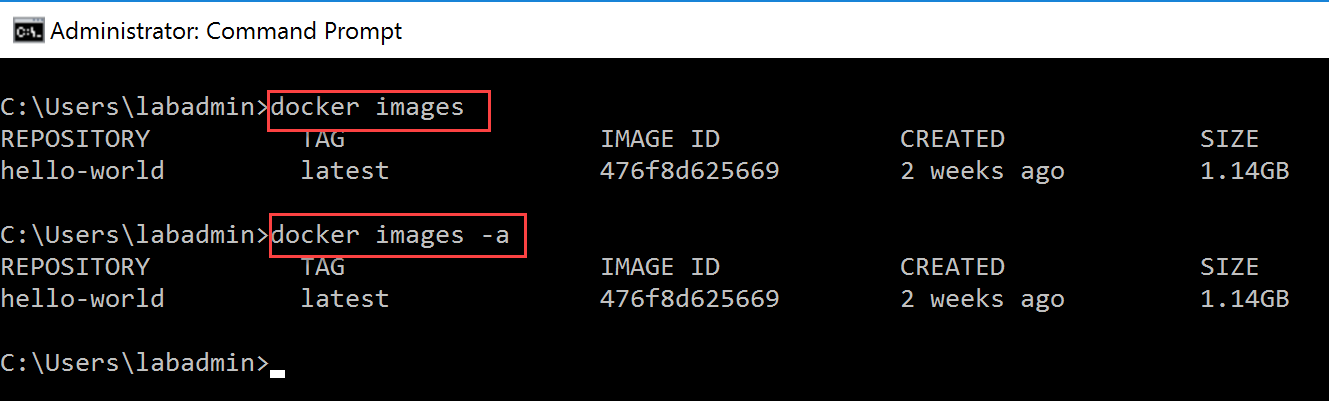
## Task 1: Installing Docker for Windows on the lab-jumpVM

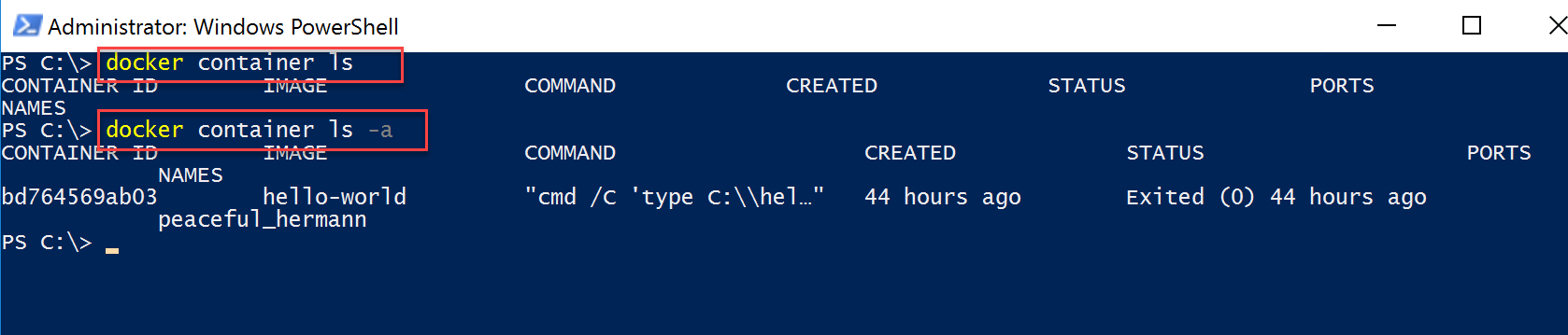
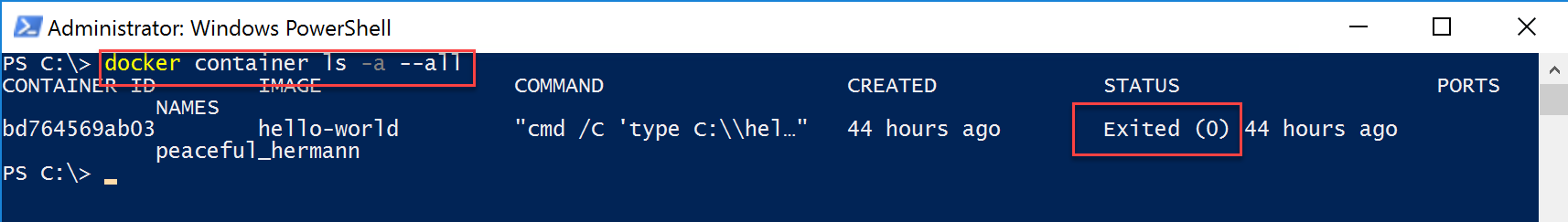
1. If not logged on anymore to the lab-jumpVM, open an RDP session to this Virtual Machine, using labadmin / [L@BadminPa55w.rd](mailto:L@BadminPa55w.rd) as credentials.
2. Connect to the Docker website <http://www.docker.com>; Here, **Click Products, and select Docker Desktop.**
3. **Click “Download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

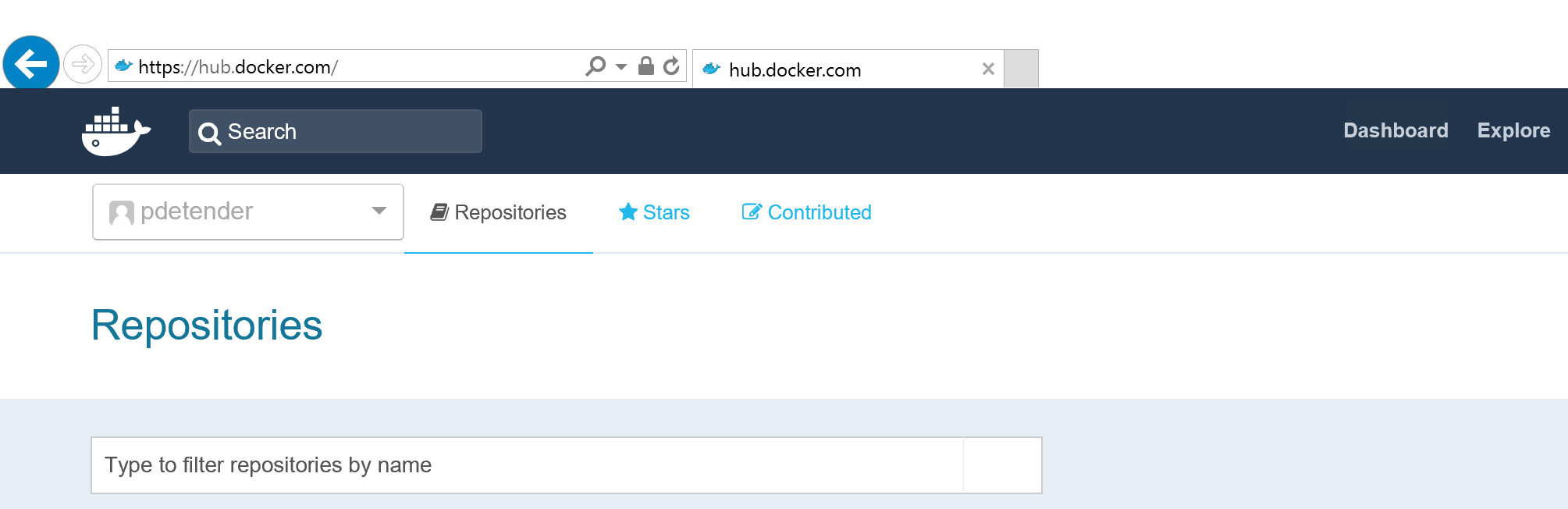
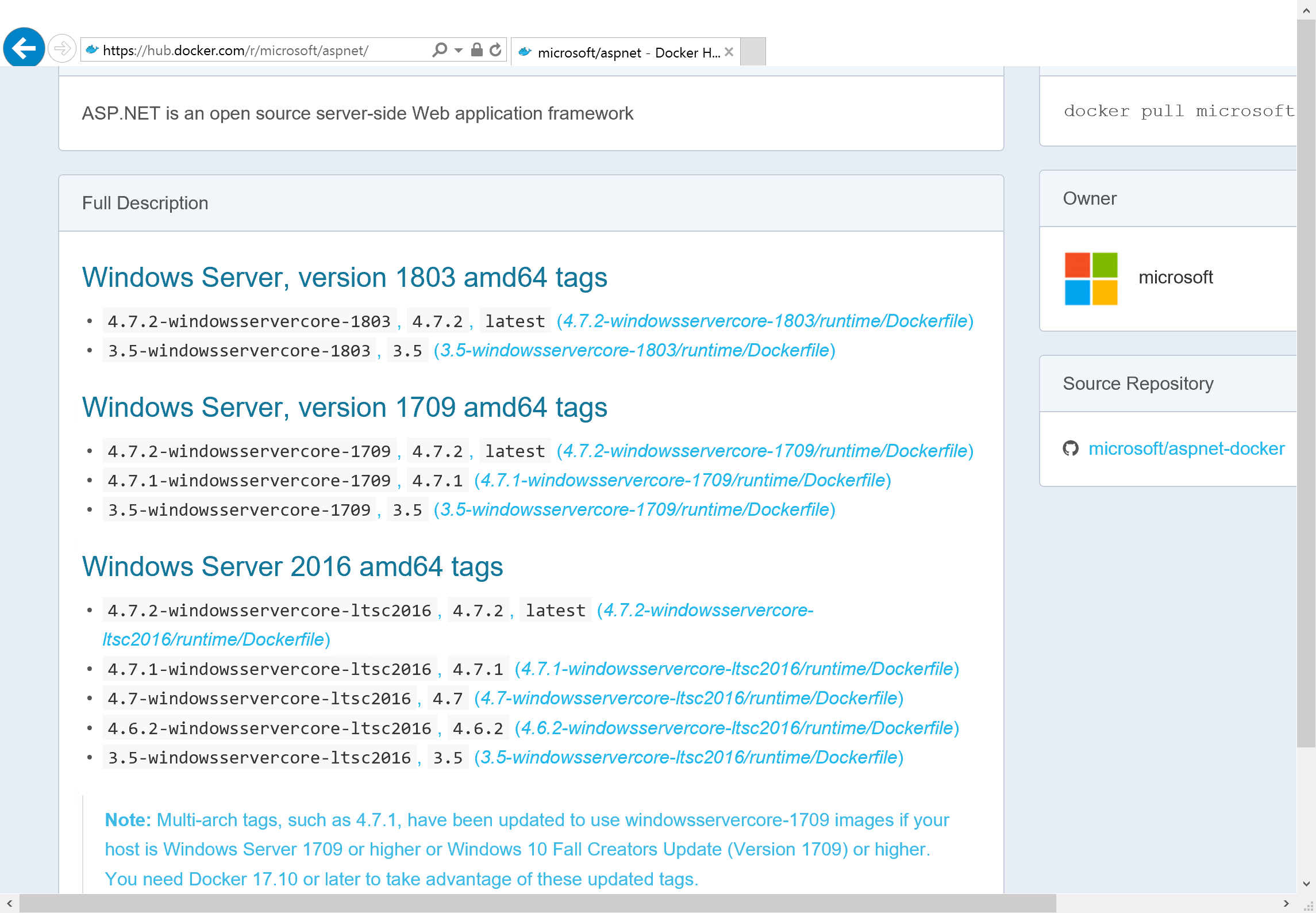
1. Another useful Docker command is to see what containers are running. Launch the following  
     
   docker ps  
     
     
     
   **Note we don’t have any Docker containers running; this was part of the hello-world container, which runs, and eventually automatically stops again.**
2. Like the 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



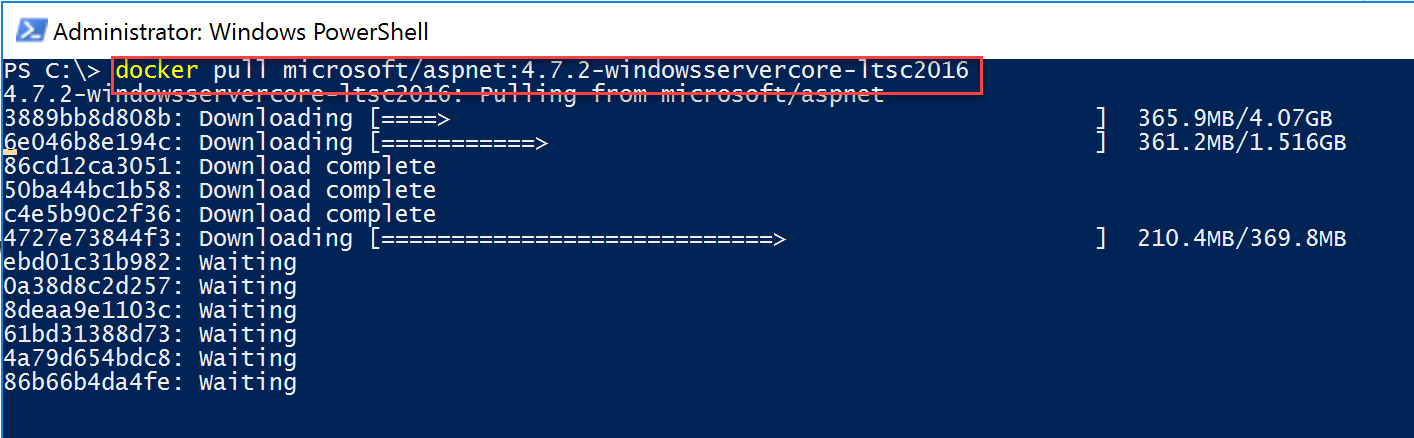
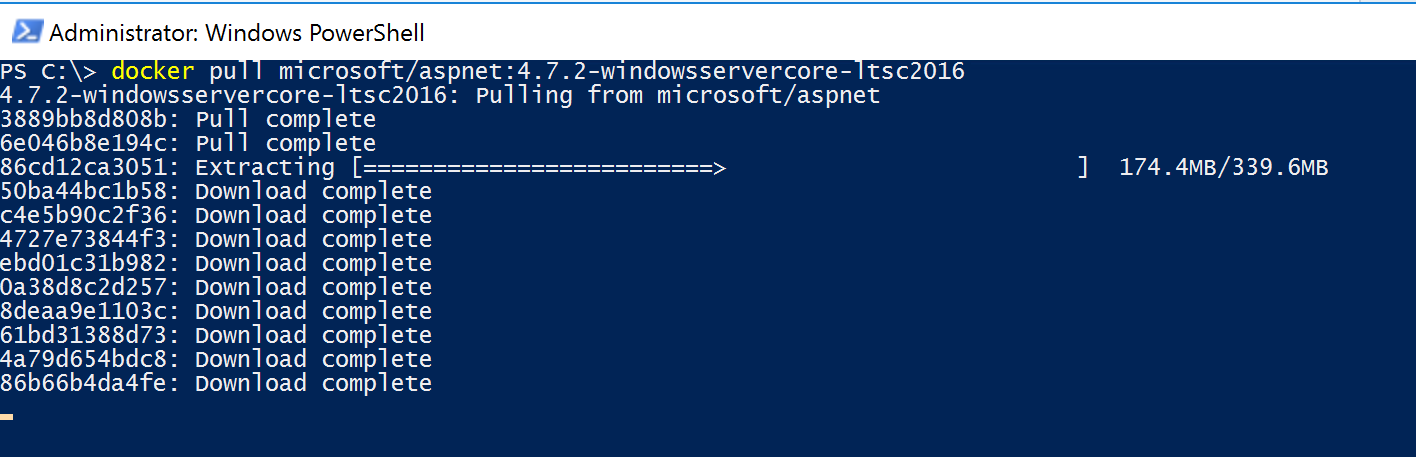
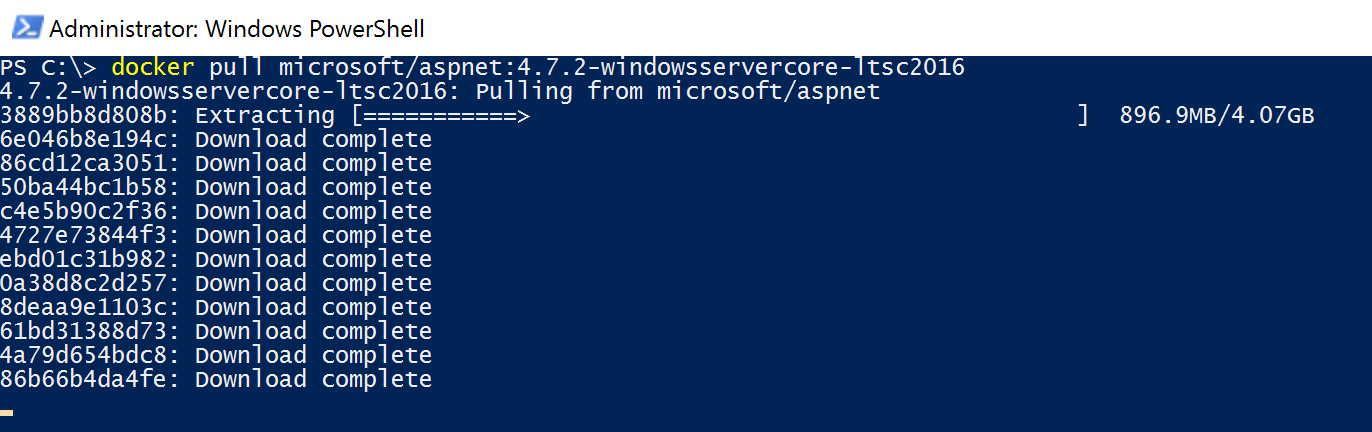
1. To get a view on the Docker Containers we have on your system, use the following command:  
     
   docker container ls or docker container ls -a  
     
   (**Note I switched to PowerShell from here on, to show you both command line tools are transparent across Docker usage)**  
   Notice the difference 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.
2. 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 😊).

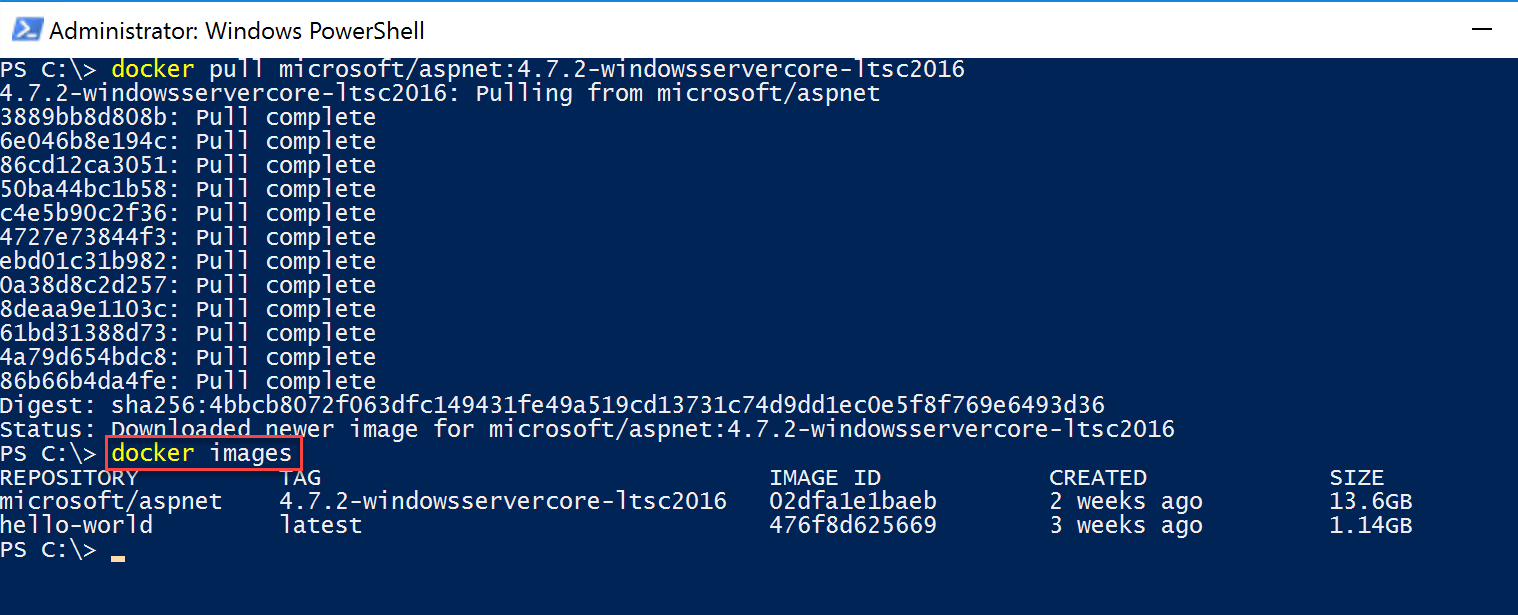
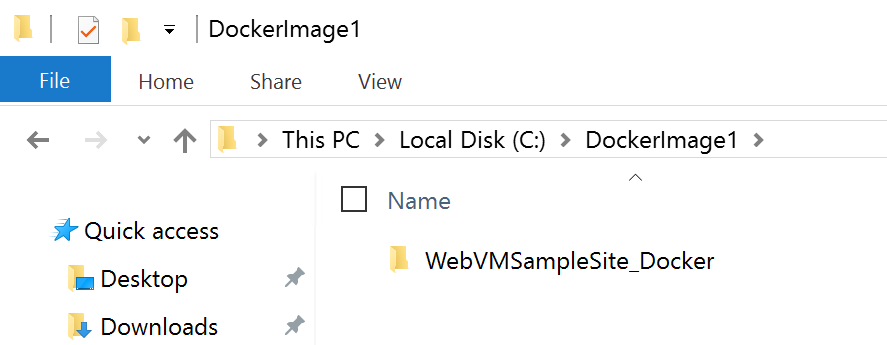
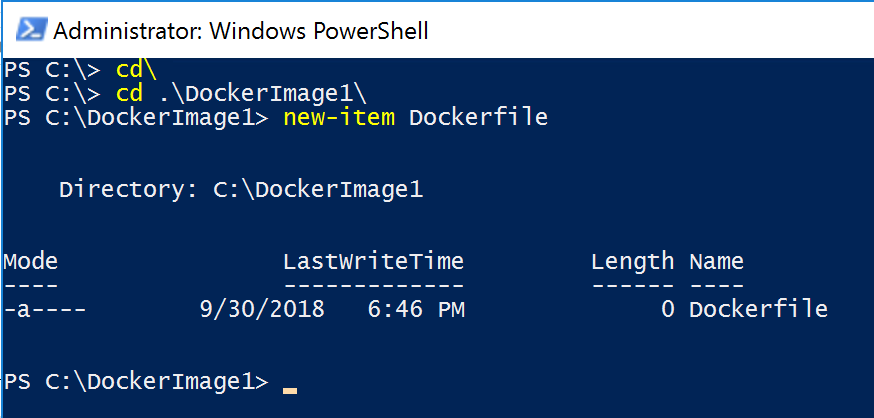
Similar to the hello-world container images that got pulled down before running, let’s move on with exploring the different Windows OS-based container images in Docker Store, and pulling them to our local lab-jumpVM.

1. Connect to hub.docker.com from your internet browser, and log on with your previously created DockerID credentials.   
     
   
2. In the upper **Search field, type “aspnet”;** this results in a list of repositories.   
     
   
3. **Select the “microsoft/aspnet”** repository. Which shows you the different ASPNET-based Docker images, provided by Microsoft. Important difference to note is the several Windows Server Operating System versions offered here.  
     
   

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

1. **From the Command Prompt or PowerShell**, initiate the following Docker command to pull the public Docker image to our lab-JumpVM:  
     
   docker pull microsoft/aspnet:4.7.2-windowsservercore-ltsc2016  
     
     
   Notice this image is based on different “layers” (the 3889bb…, 50ba44…, ebd01…)  
   
2. **Wait for the image to finish downloading** and **extracting**. Depending on your internet connection speed, this might take several minutes. From a container perspective, this is based on the Microsoft/aspnet, having a **tag** pointing to the latest edition (the 4.7.2-windowsservercore-ltsc2016 part in the name).

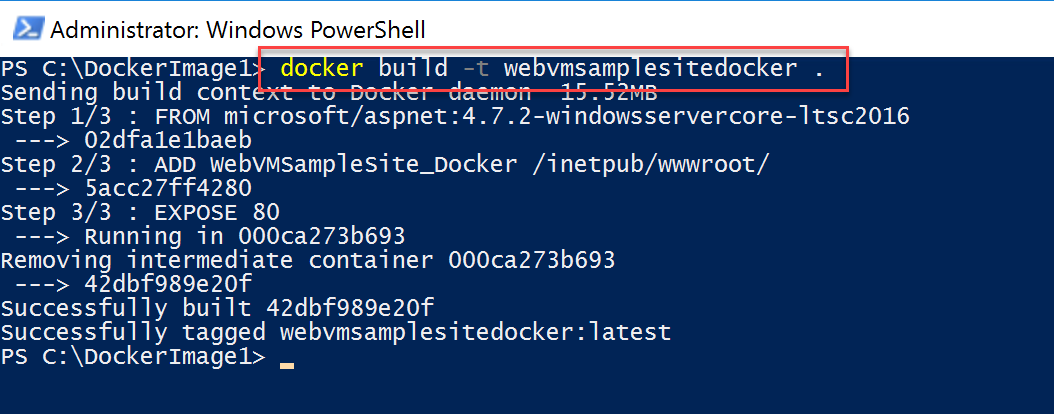
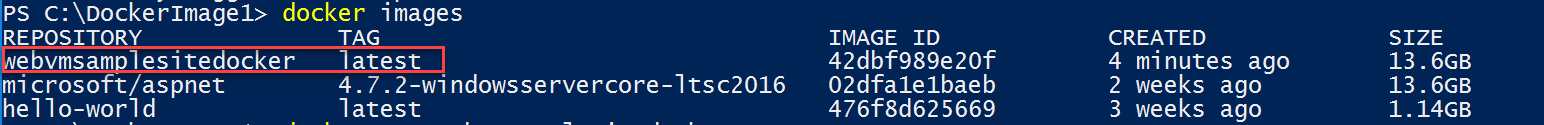
## Task 3: Containerize your ASP.NET application using Visual Studio and Docker

1. **From the Docker command line (Command Prompt or PowerShell)**, run the following Docker command:   
     
   docker images   
     
   and validate the Docker image for microsoft/aspnet is showing up.  
     
   
2. The next step involves creating our **Dockerfile**, which holds the different steps and actions to get our web application published into a Docker container. To do this in some sort of structured, but yet easy way, start with **creating a new folder in the C:\, called DockerImage1, and copy the folder c:\WebVMSampleSite** into it, renaming this folder after the copy is completed to **webvmsamplesite\_docker**. (as such, you can detect the differences in source files when something would go wrong).  
     
   
3. Next, initiate the creation of a new Dockerfile document, by running the following Powershell commands:  
     
   cd\ (this moves you to the root of C:\)  
   cd DockerImage1 (this moves you to the C:\DockerImage1 folder)  
   new-item Dockerfile (this creates a new file called “Dockerfile”, with no extension)  
     
   
4. **Open** the Dockerfile file in Notepad for easy editing. From PowerShell, you can run “notepad Dockerfile”, or browse to the file from File explorer and open it with Notepad.
5. **Type in** the following lines **into the Notepad window** (copy/paste might work, but sometimes injects garbage code, which breaks the file – therefore, typing is the safest option).

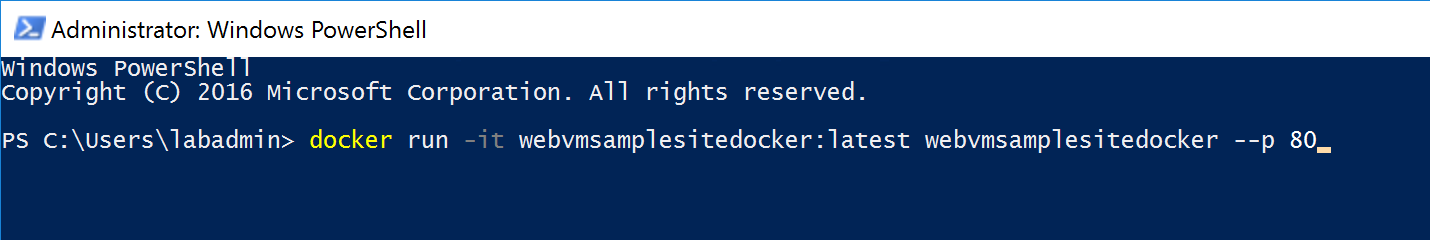
FROM microsoft/aspnet:4.7.2-windowsservercore-ltsc2016

ADD WebVMSampleSite\_Docker /inetpub/wwwroot/

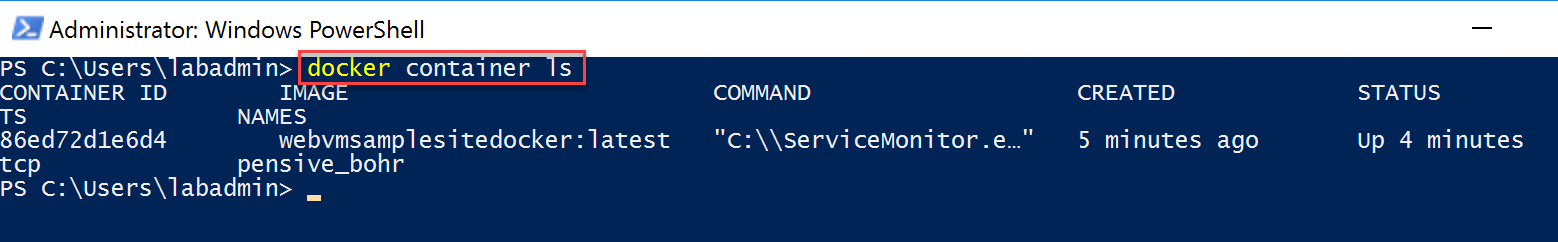
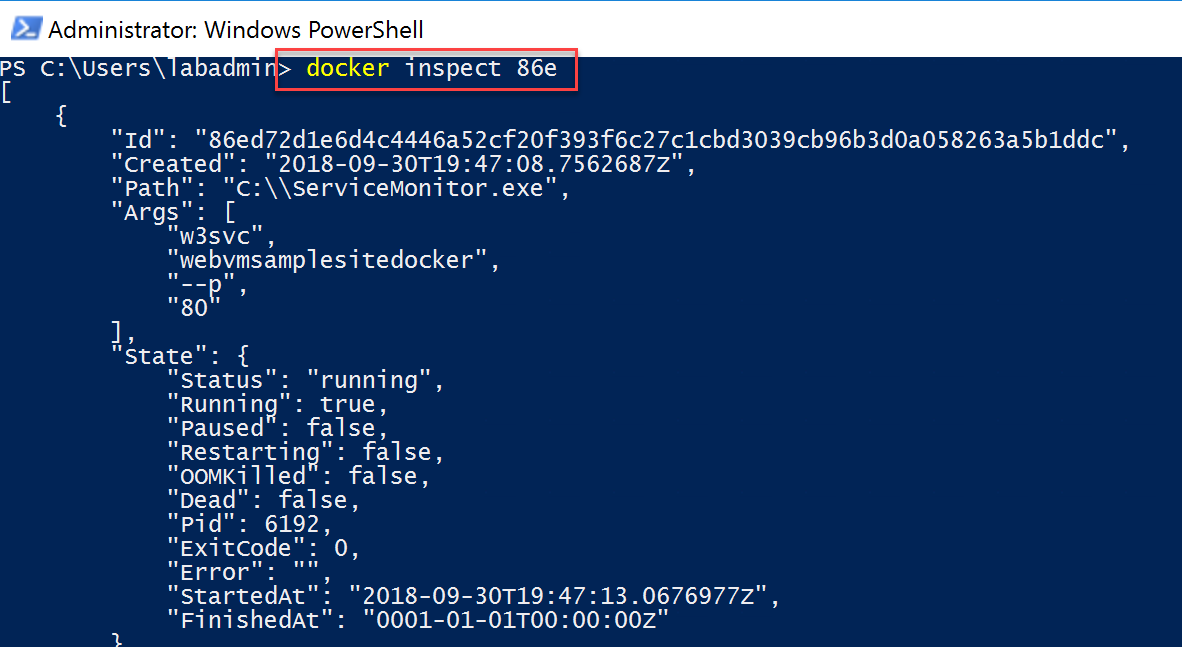
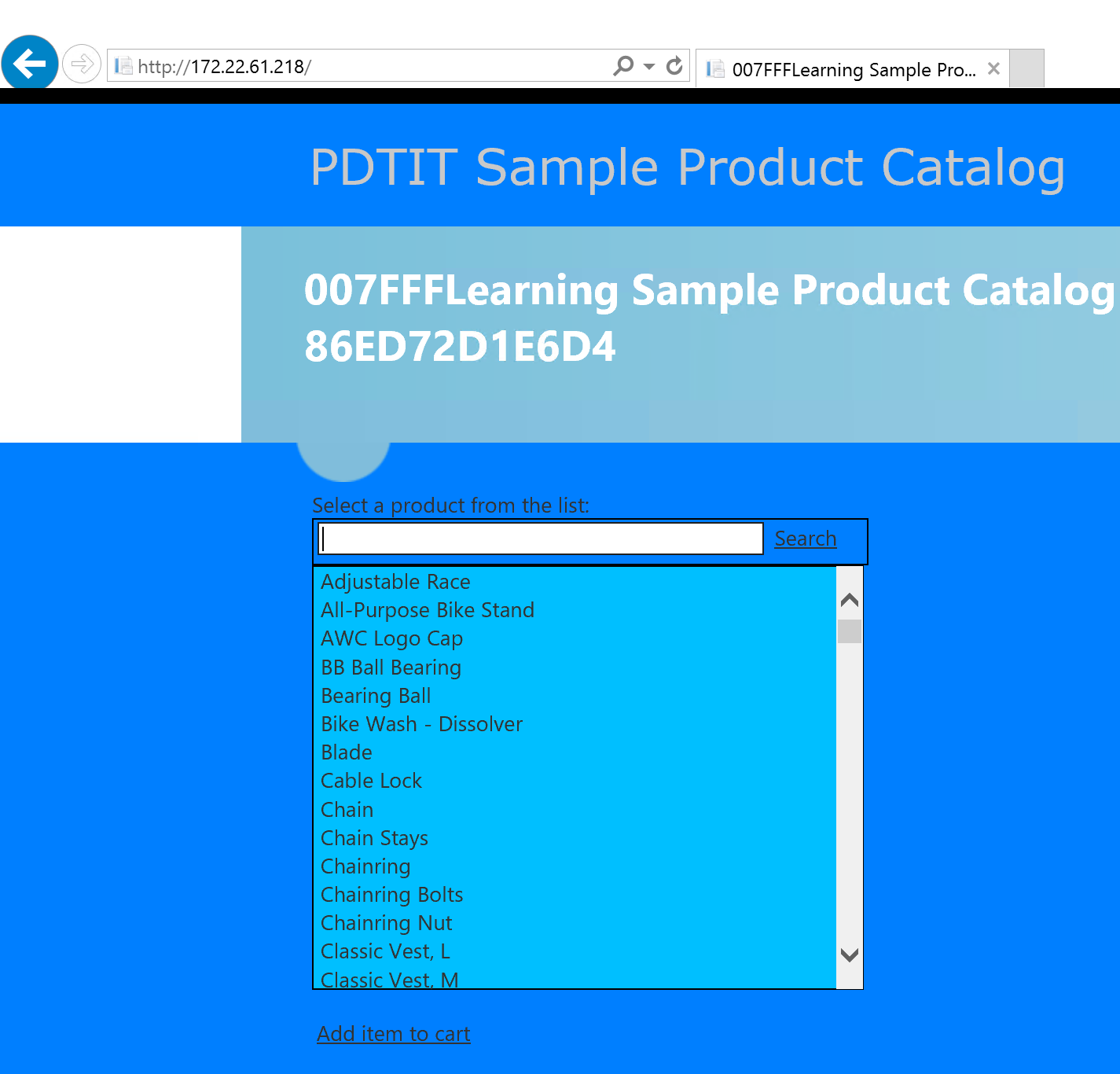
EXPOSE 80  
  
and **save the file**.

1. Some words of explanation what this Dockerfile does:  
   - the **FROM** statement defines what Docker image we will use as a starting point, meaning, this Container should be based on the Windows Server Cored LTSC2016 version  
   - the **ADD** statement tells the Docker image to copy all contents from the subfolder “WebVMSampleSite\_Docker to the inetpub/wwwroot folder, which is running inside the container (=the default IIS folder on a regular Windows machine)  
   - the **EXPOSE 80** statement allows running the web site on port 80, allowing us to connect to the IP-address of the container, and browsing to it on port 80, validating the web site is running as expected.
2. **Back in the PowerShell window,** initiate the following command to get the Docker image build:  
     
   docker build -t webvmsamplesitedocker .  
     
   (Note the dot at the end – what this points at is creating a Docker image from all content in the current folder, which technically is everything in the \WebVMSampleSite\_Docker folder)  
   
3. **Validate** the image is built successfully by running the following command:  
     
   docker images  
     
   and notice the webvmsamplesitedocker is in the list, having a tag of “latest”  
     
   

Now our image is available, let us **spin up a new container** using this image, by executing the following command:  
  
docker run -it -d --publish 80 --name webvmsamplesitedocker webvmsamplesitedocker:latest

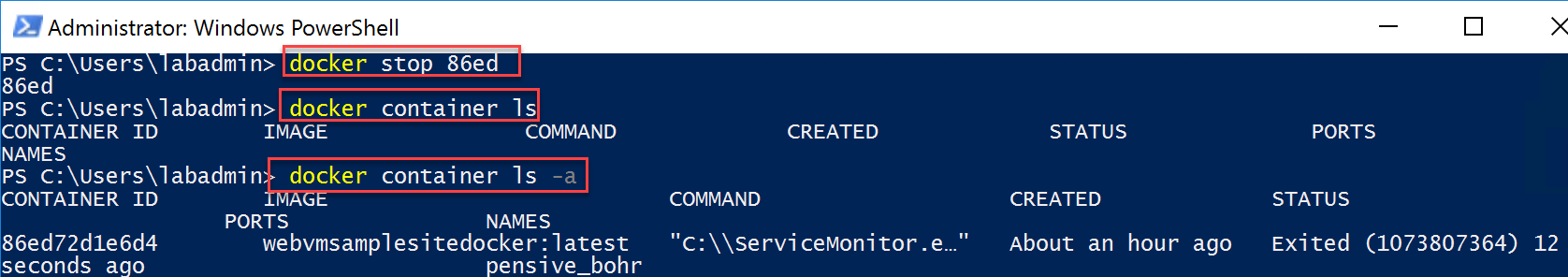
**Where:**- “docker run” start the container run process  
- “-it” defines the container needs to run in interactive mode (output)  
- “webvm…:latest” the name of the image to use, note the tag  
- “webvm…” the name for the container  
-“—p 80” run the container on port 80  
  


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


1. **From a new PowerShell window**, let us check on the details of this running container, by executing the following command:  
     
   docker container ls  
     
     
     
   This shows our running container.
2. **Next,** let’s check on **some additional and interesting technical details** for this specific container, by running the following command:   
     
   **Note: each container has an ID (the 86ed721e6d4 in my example), where we can reuse this in the docker commands, referring to that object. Easier than using the long names 😊**  
   docker inspect 86ed (where 86ed should be replaced with the ID of your container)  
     
     
     
   
3. In this JSON output file, scroll down and search for the **Networking** section:  
     
   
4. Here, check for the IPAddress under the Networks / nat section specifically. This reveals the IP-address that is used by our running container.
5. From your internet browser, connect to this IP-address, which shows you the running web application. Yeah! (**Notice it also refers to 86ED… as hostname, which is the technical ID of our container)**  
     
   
6. To **stop** our running Docker container, **execute** the following command, **one after the other**:  
     
   Docker stop 86ed (where 86ed should be replaced with the ID of your container)

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

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



## Summary

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

# Lab 5: Running Azure Container Instance (ACI) from an Azure Container Registry (ACR) image

## What you will learn

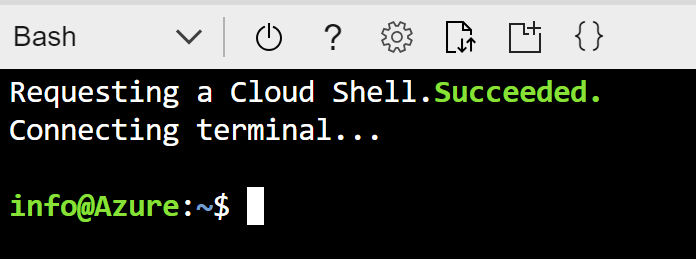
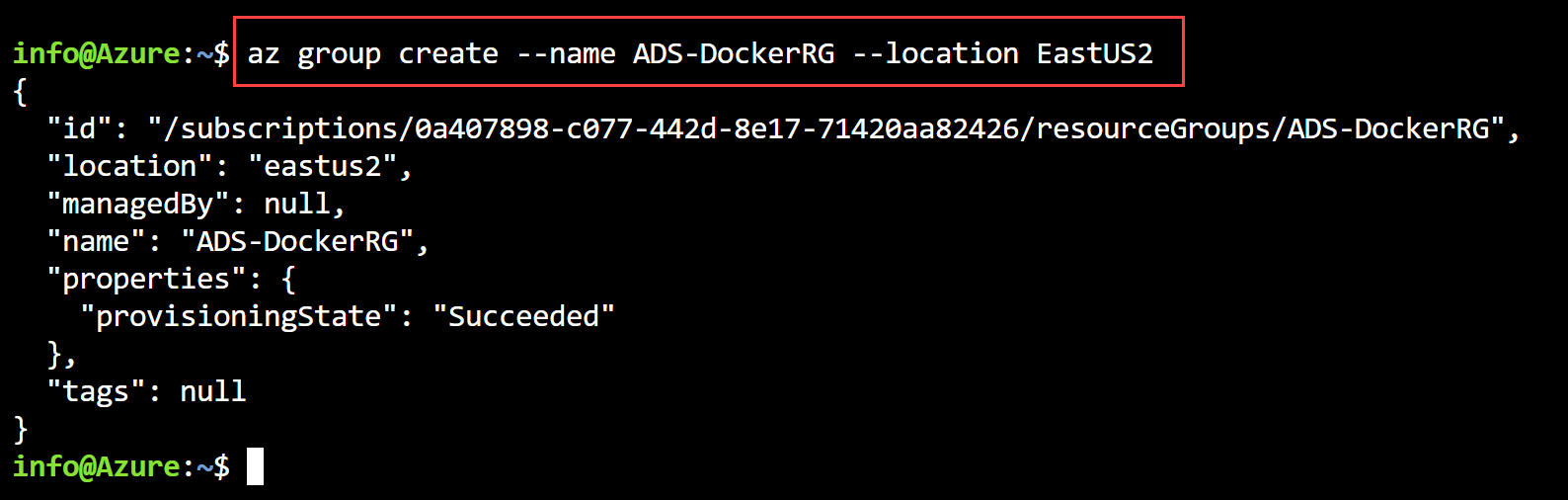
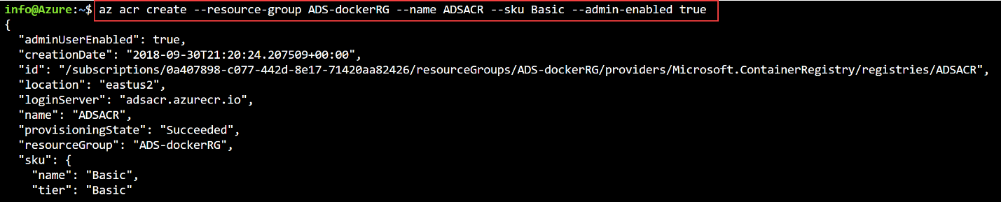
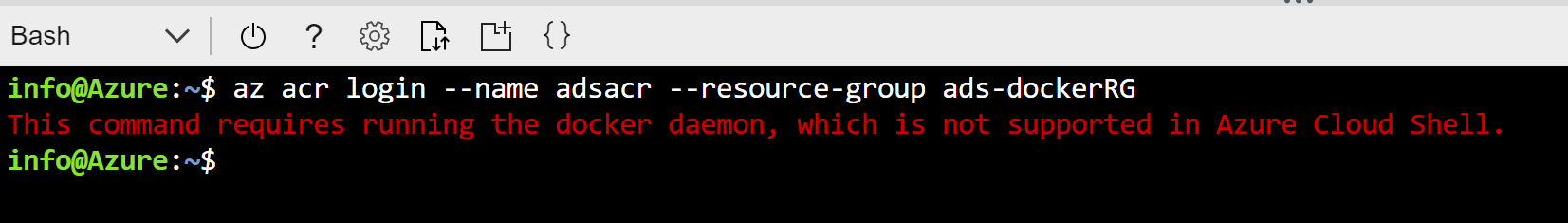
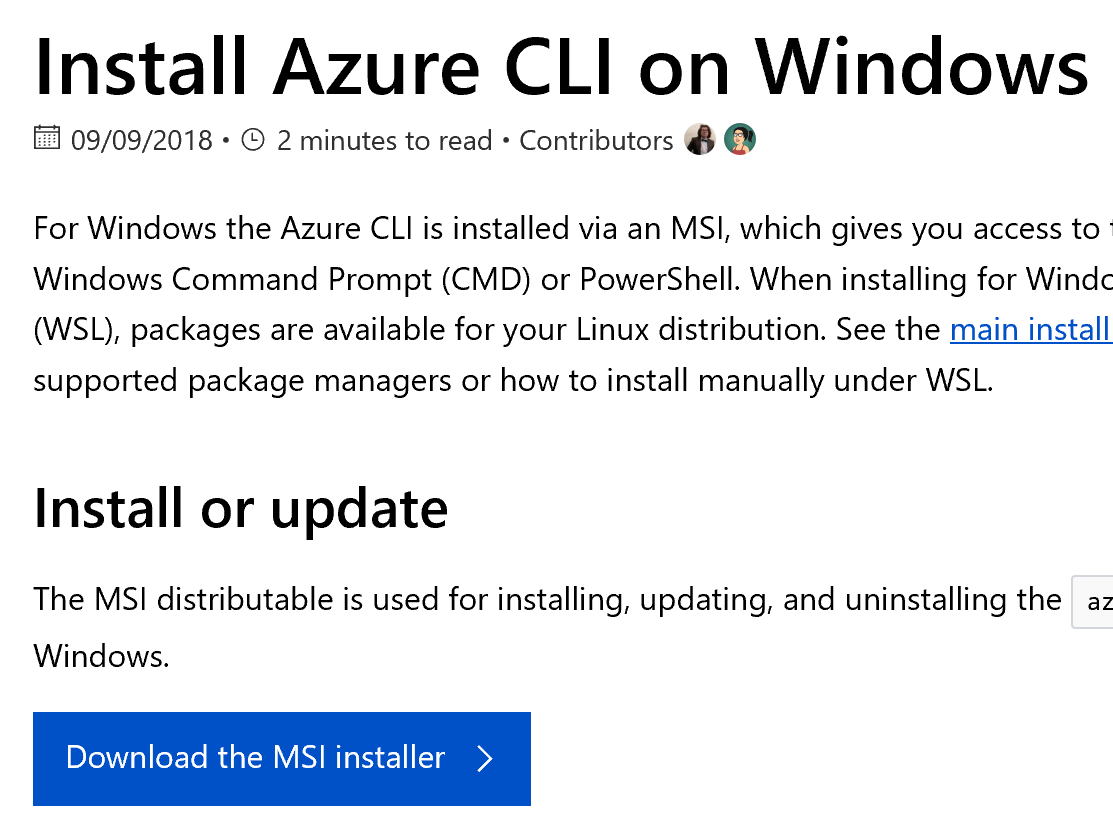
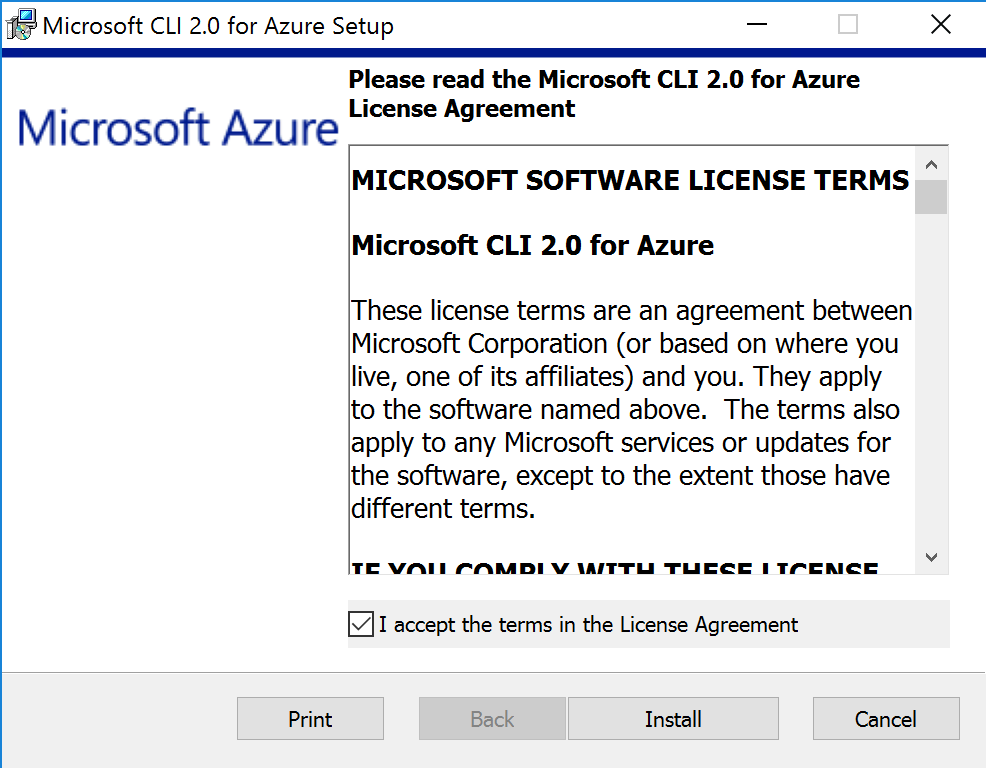
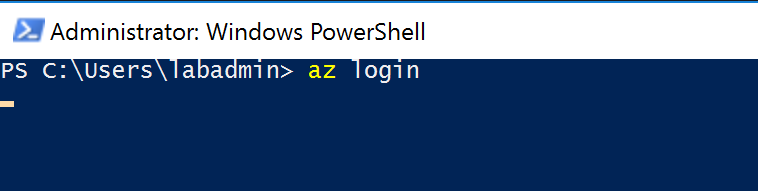
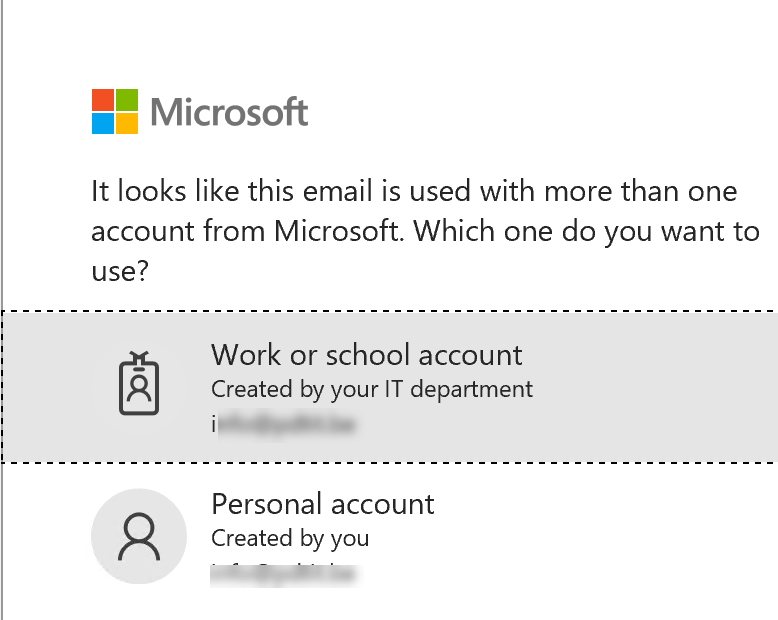
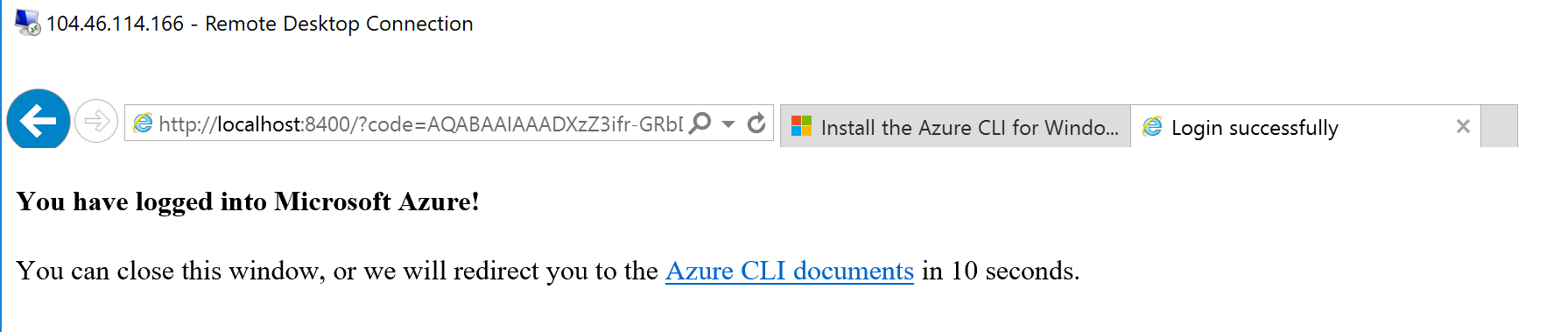
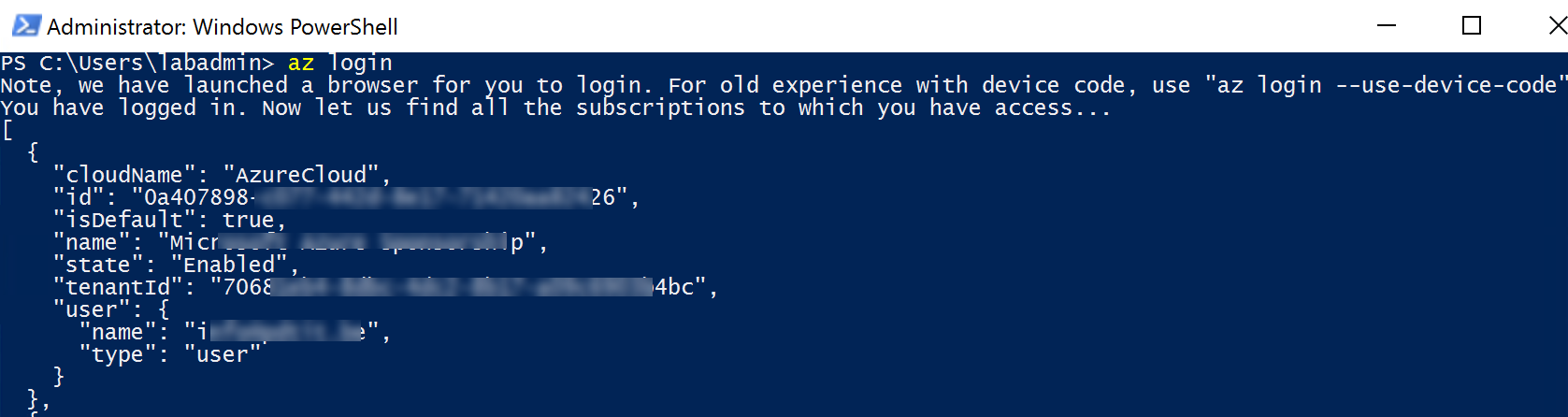
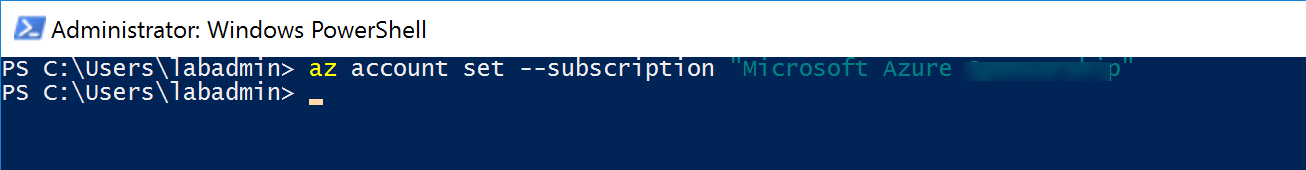
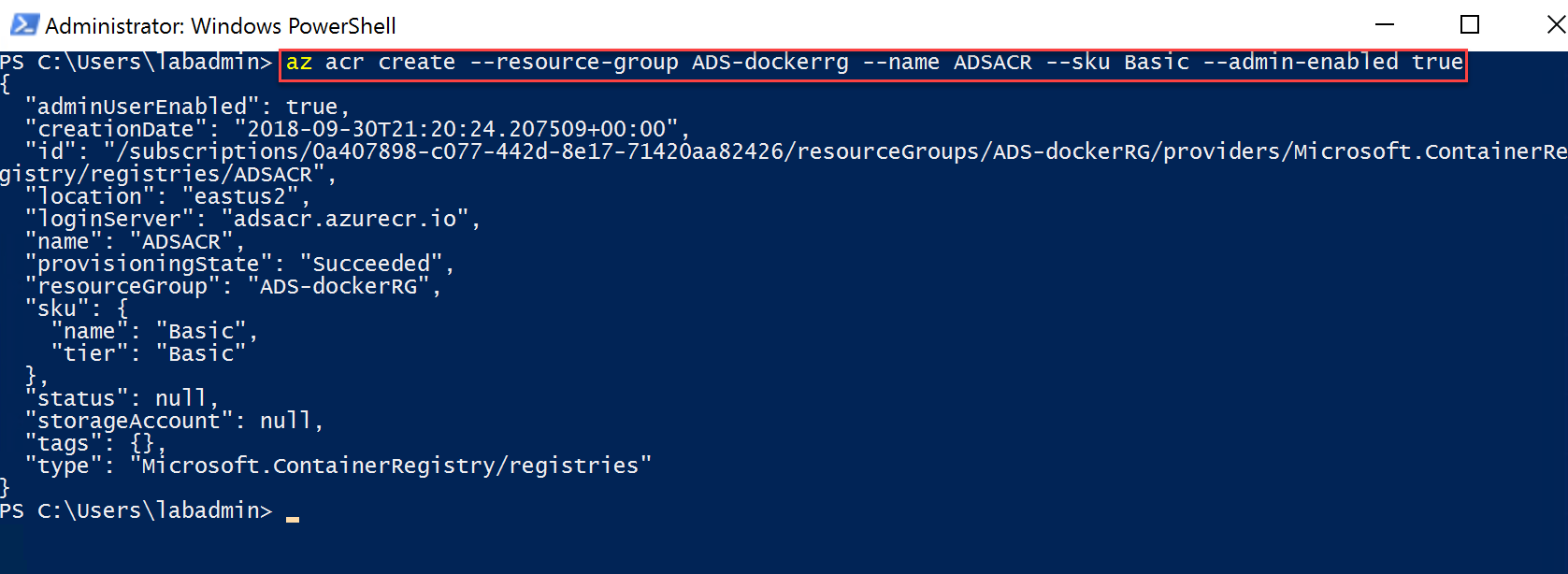
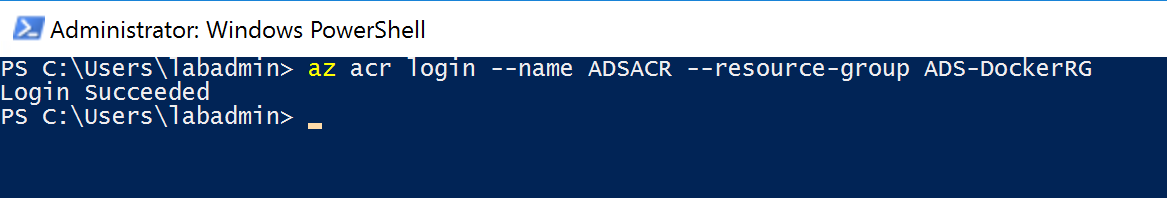
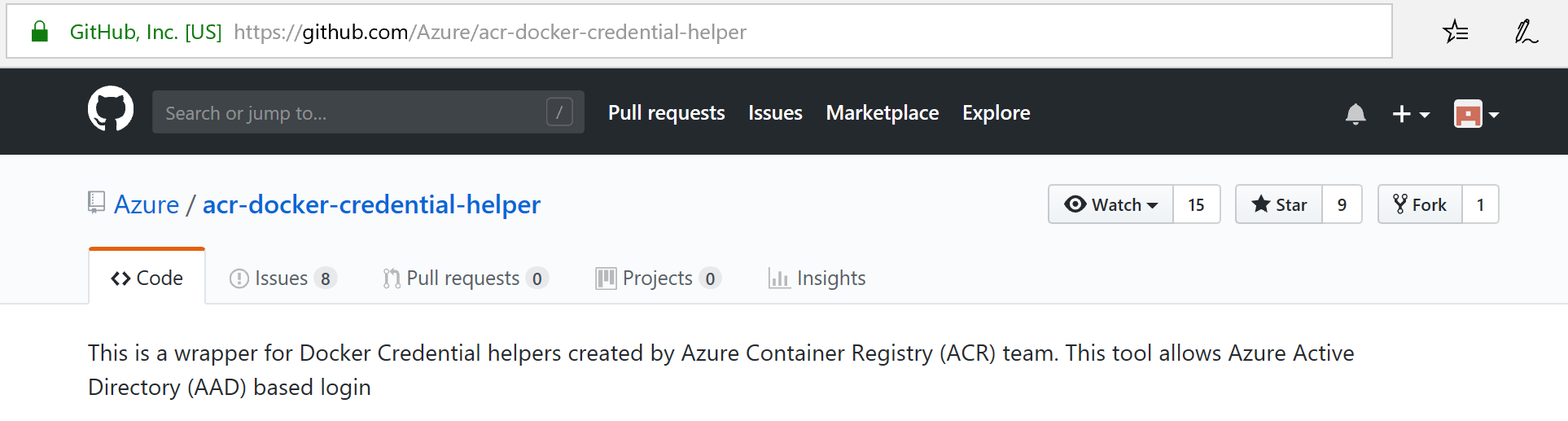
In this lab, you start from creating a new Azure Container Registry resource. Next, after authenticating to this ACR, you learn how to tag and push the Docker image you created in the previous lab to an Azure Container Registry. Lastly, you run an Azure Container Instance, based on the Docker image that is stored in the Azure Container Registry.

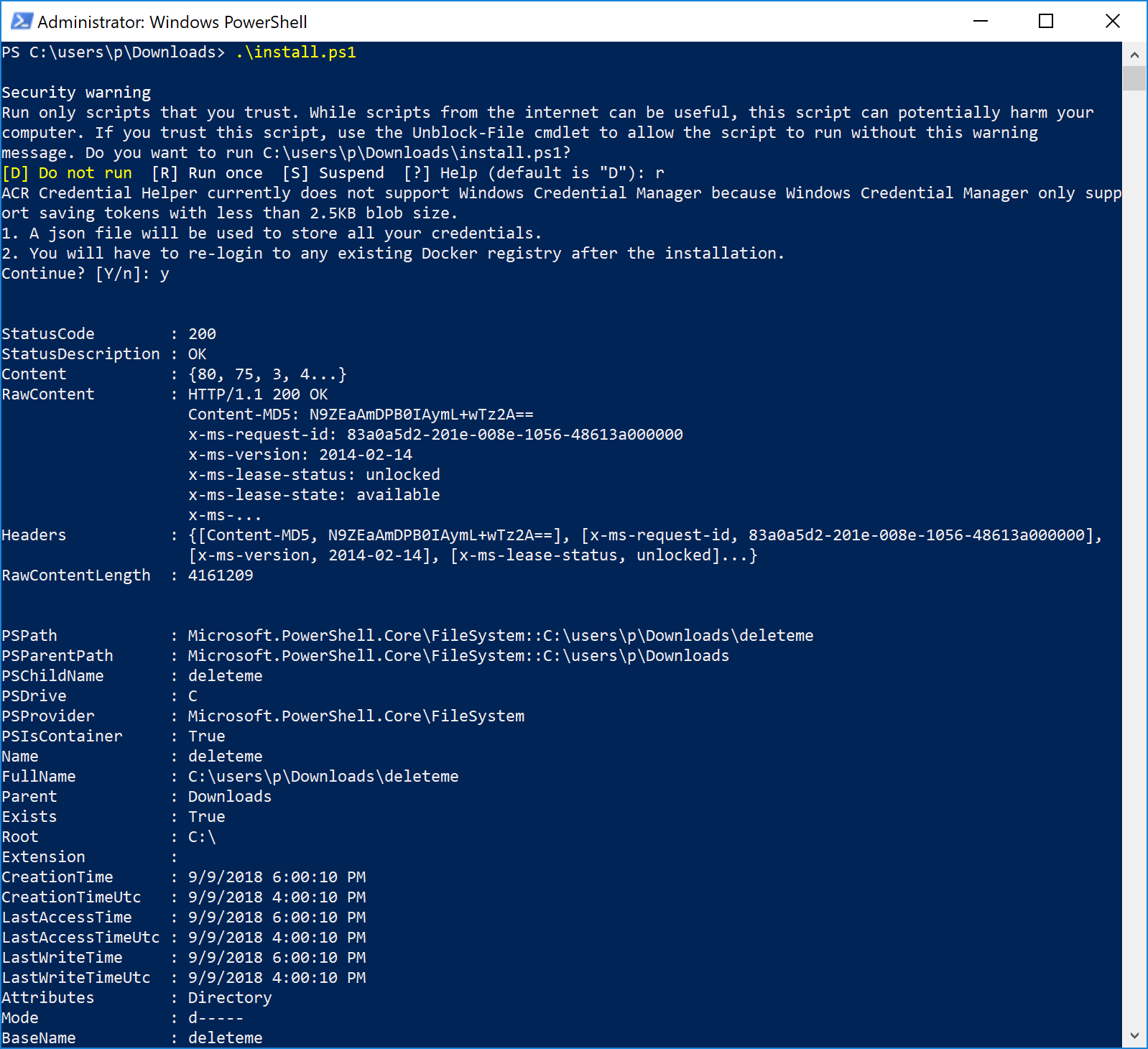
## Time Estimate

This lab should take about an hour to complete.

## Task 1: Creating an Azure Container Registry using Azure CloudShell

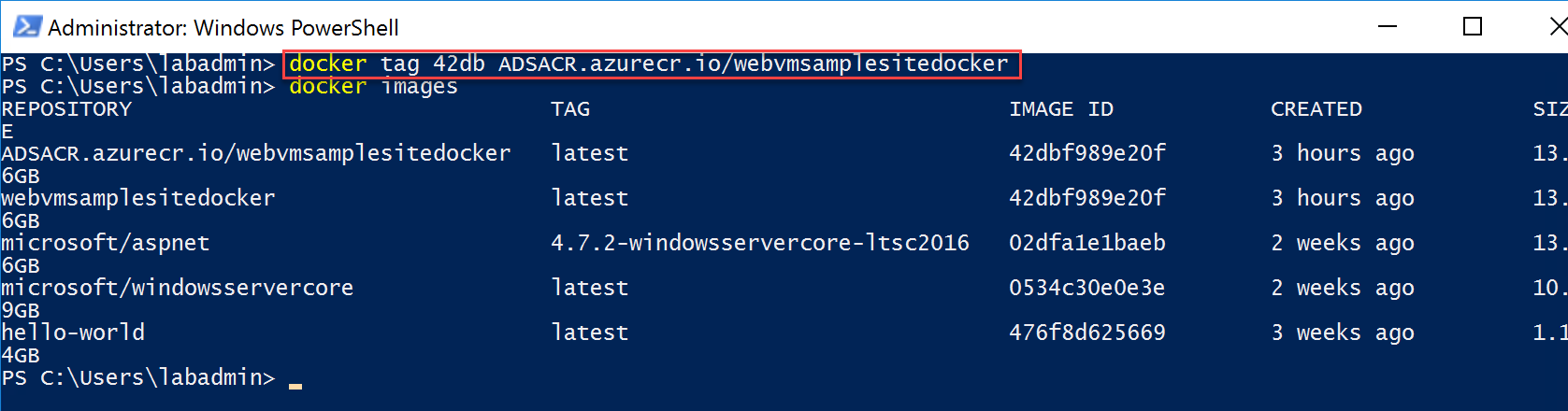
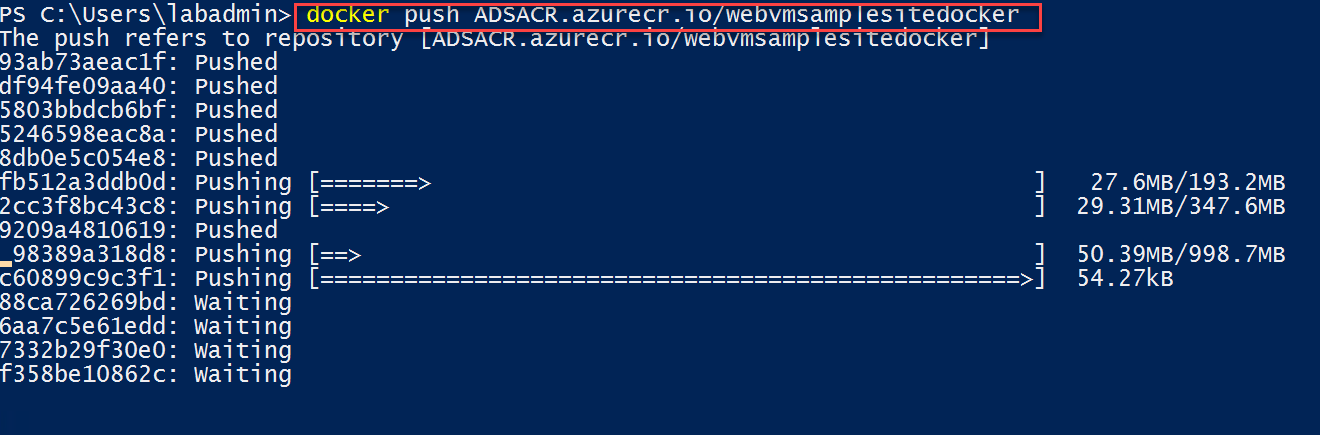
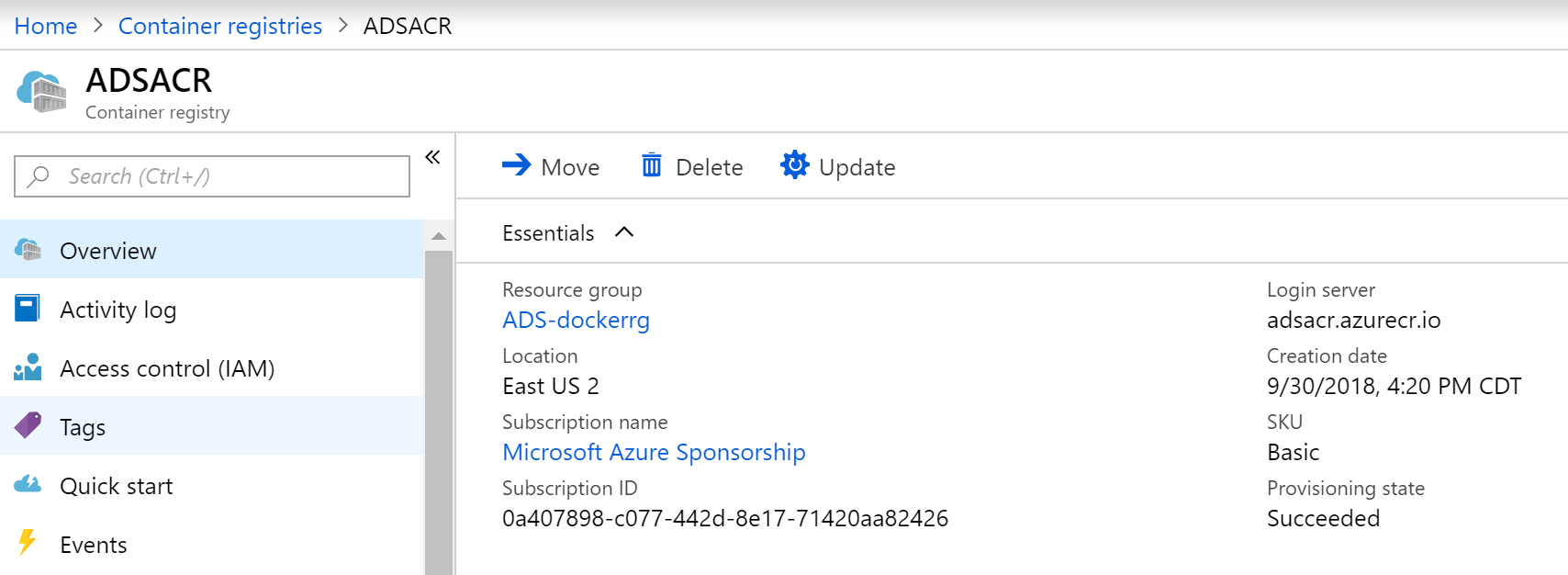
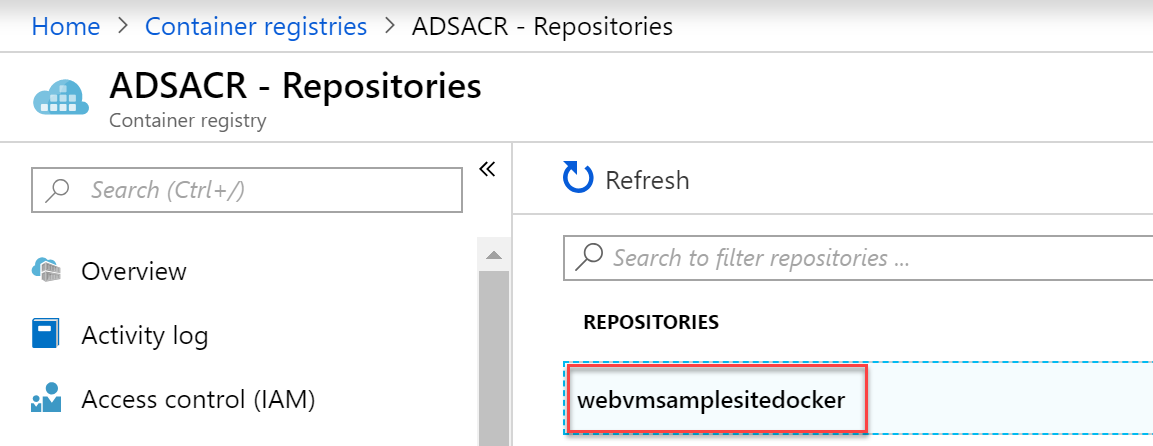
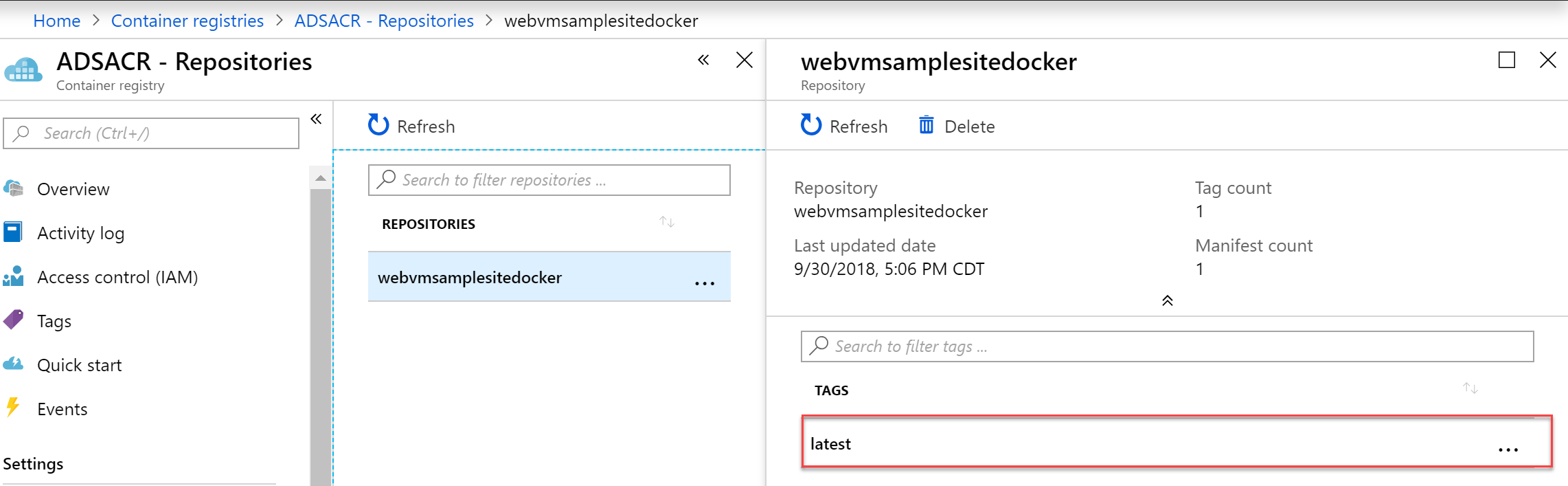
As we now validated our Docker image is running successfully in a container, we can migrate this image to an Azure Container Registry (ACR). This will allow to reuse this image for future Azure Container solutions we will be talking about.

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 EastUS2  
     
   
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 ADSACR --resource-group ADS-DockerRG  
      
    
21. **Note, ONLY if the above command should fail, it is most probably related to an issue with the Windows Credential Store. The work-around for now is running another PowerShell that is available on GitHub, allowing to install the ACR-Docker-Credential-Helper:**[**https://github.com/Azure/acr-docker-credential-helper**](https://github.com/Azure/acr-docker-credential-helper)  
      
      
    

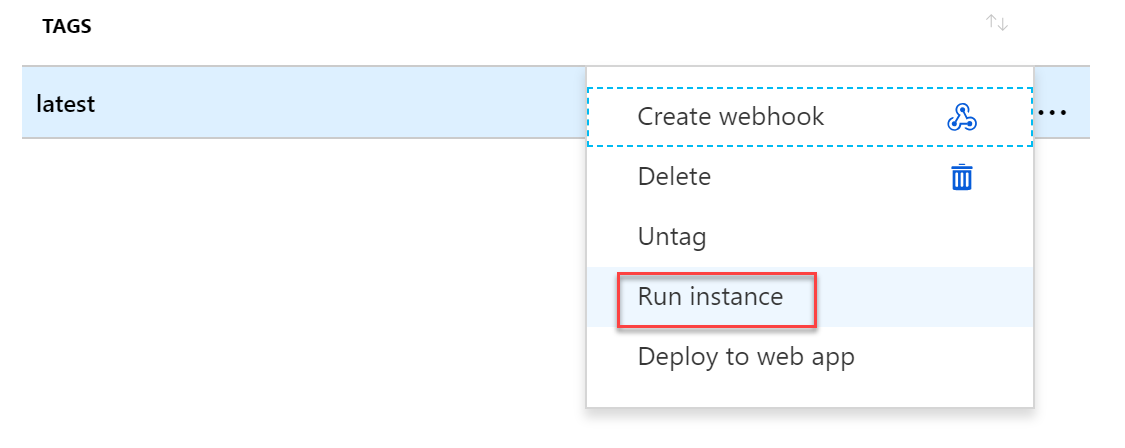
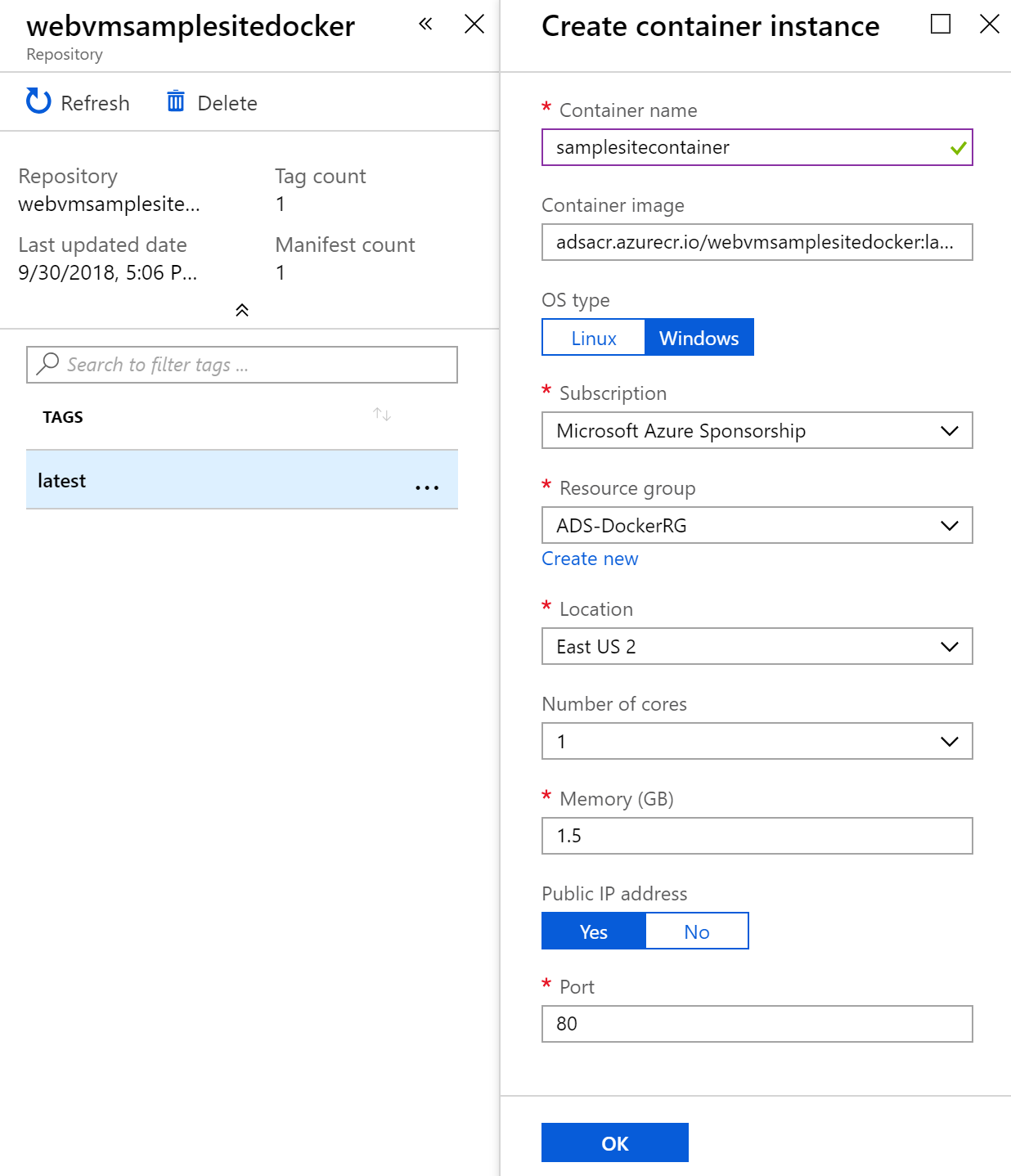
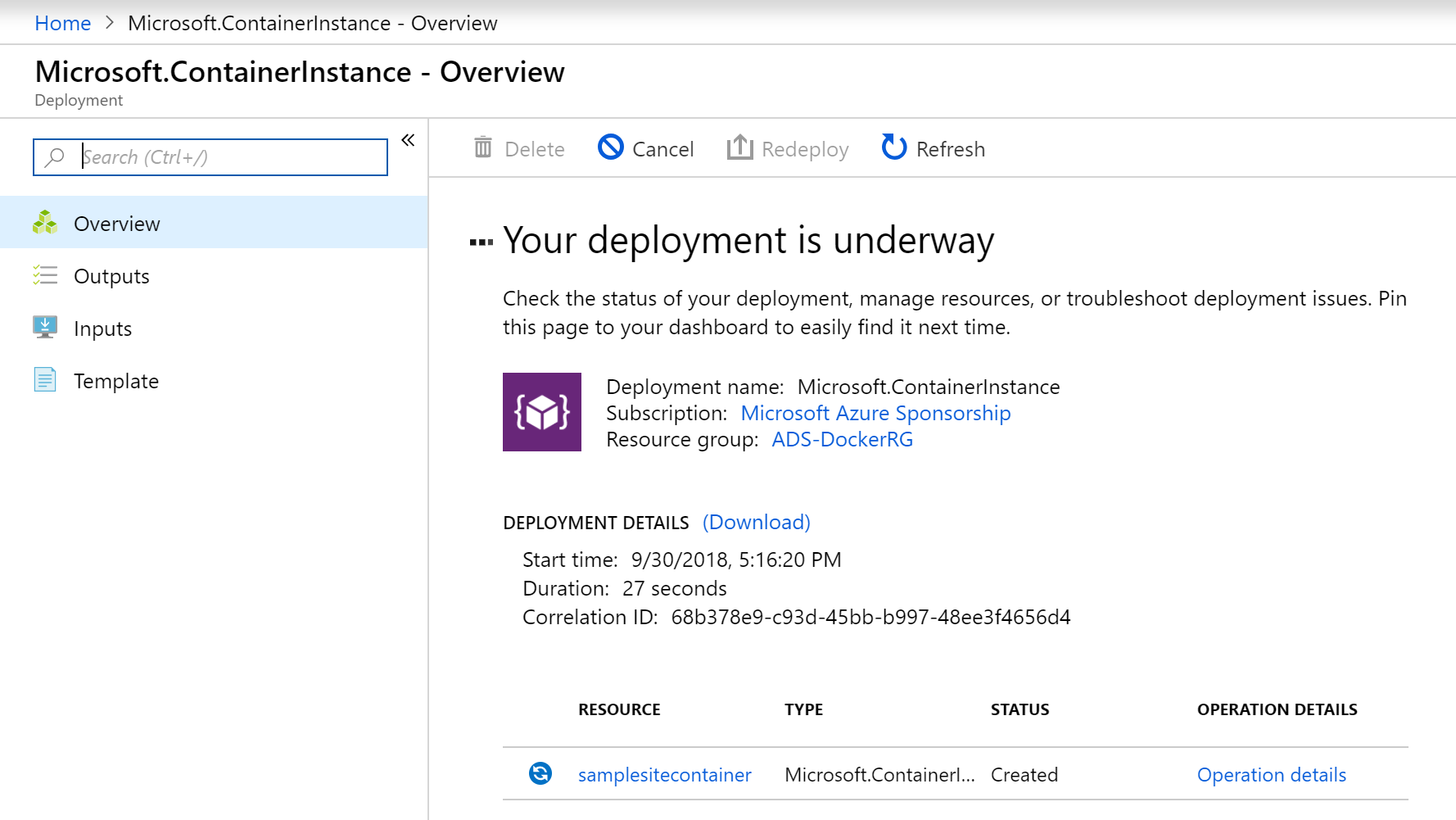
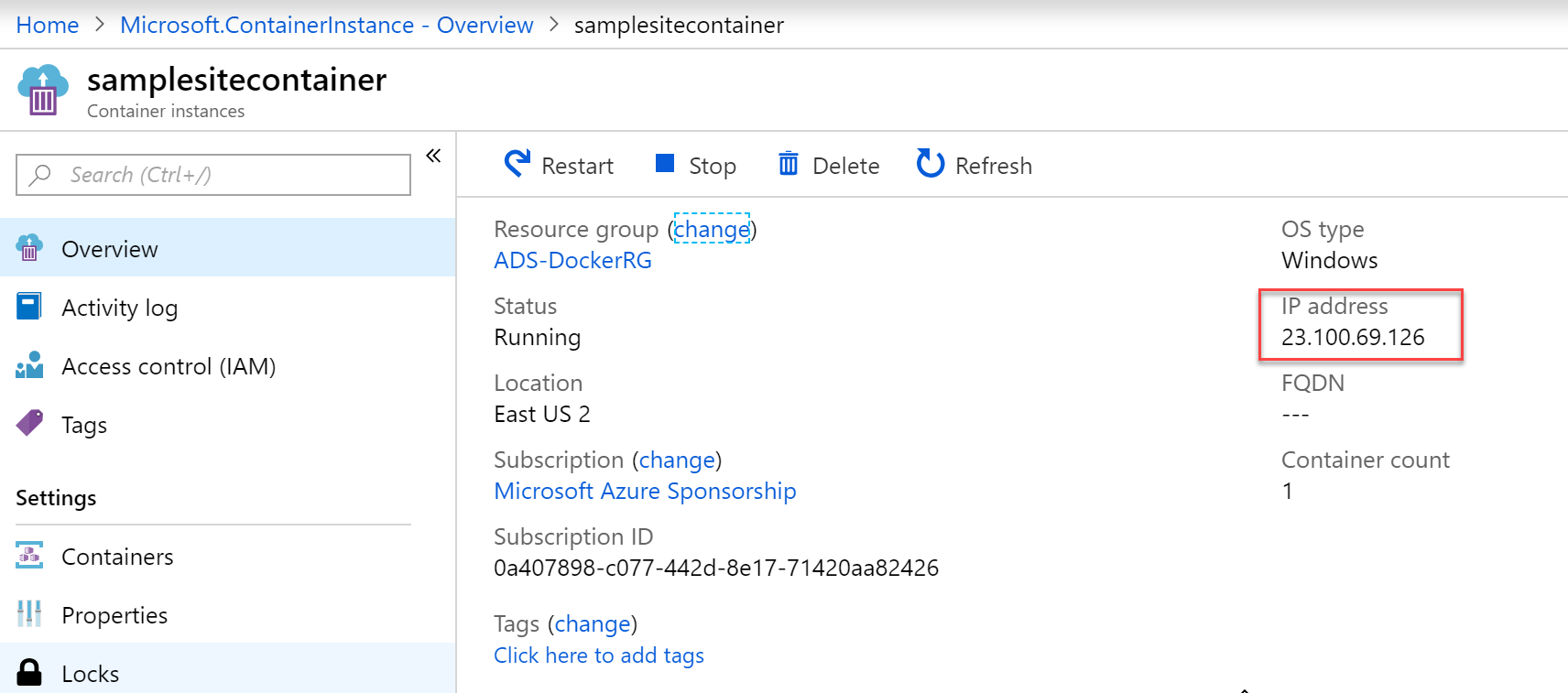
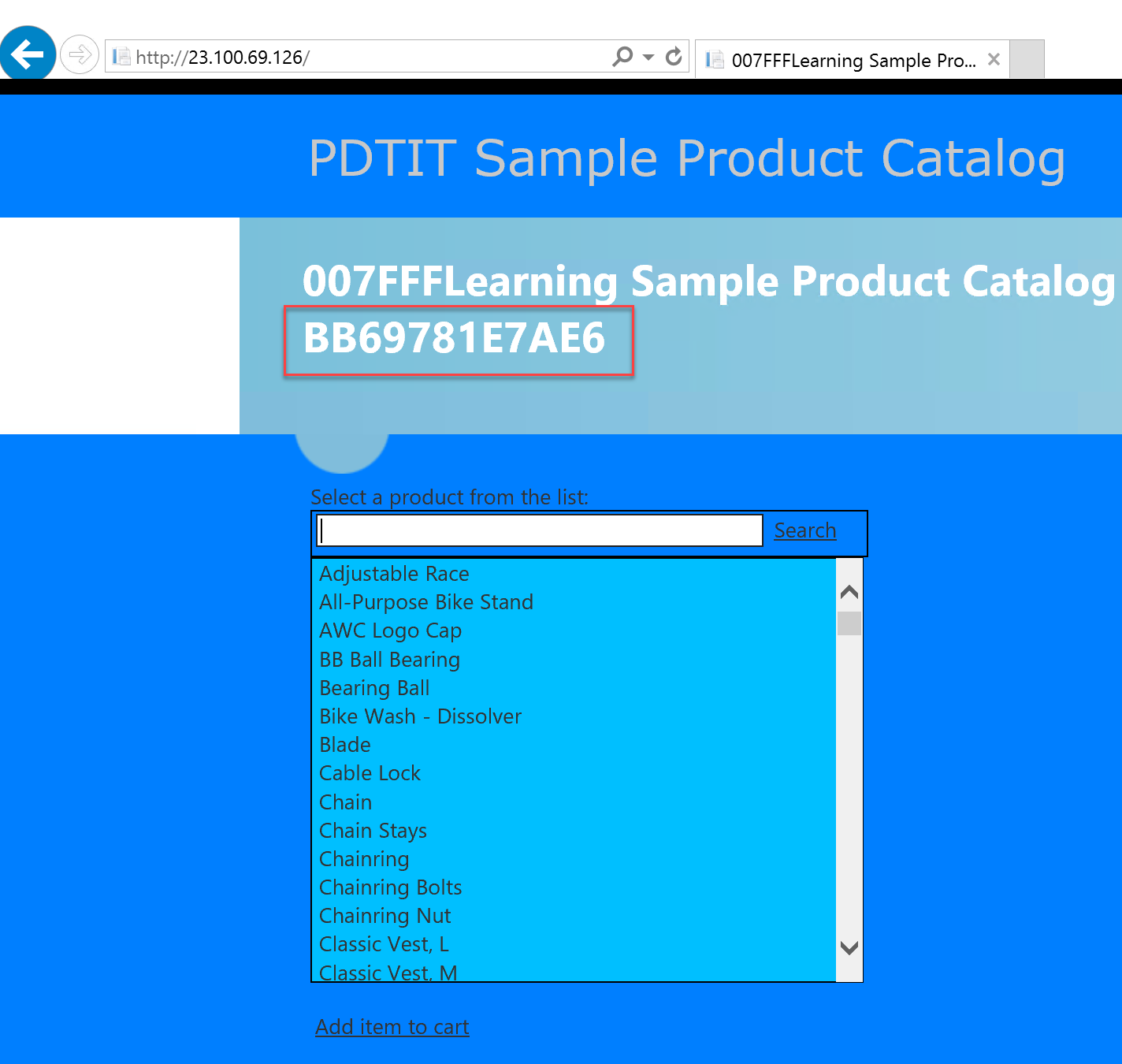
**Download the install.ps1 from the link (with the red arrow)**  


1. And try to authenticate again to the Azure Container Registry.

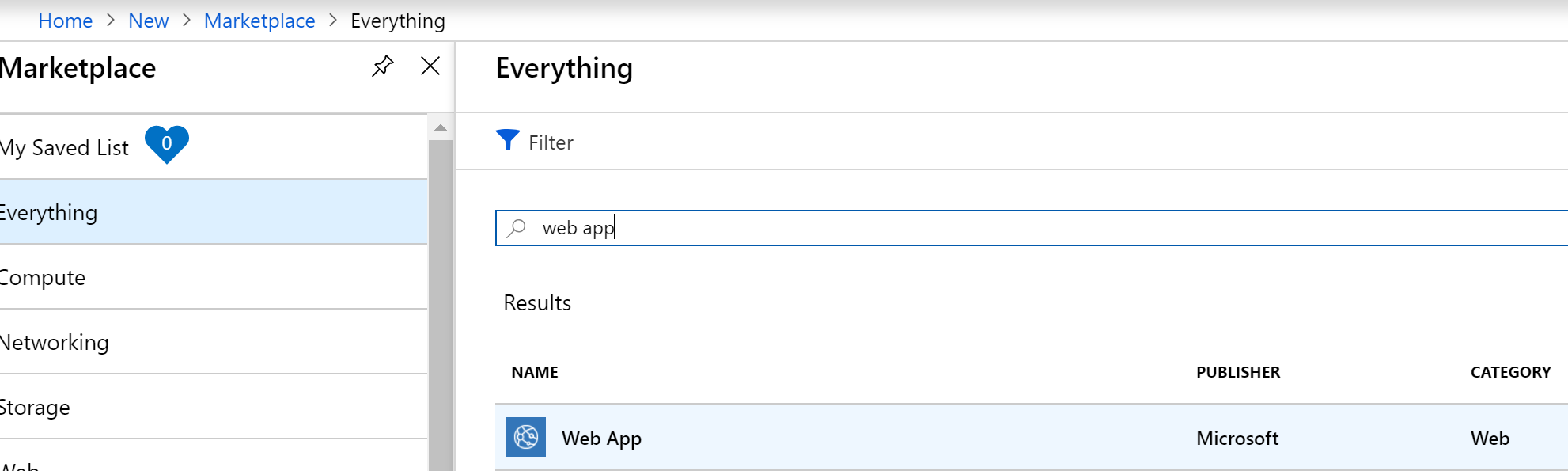
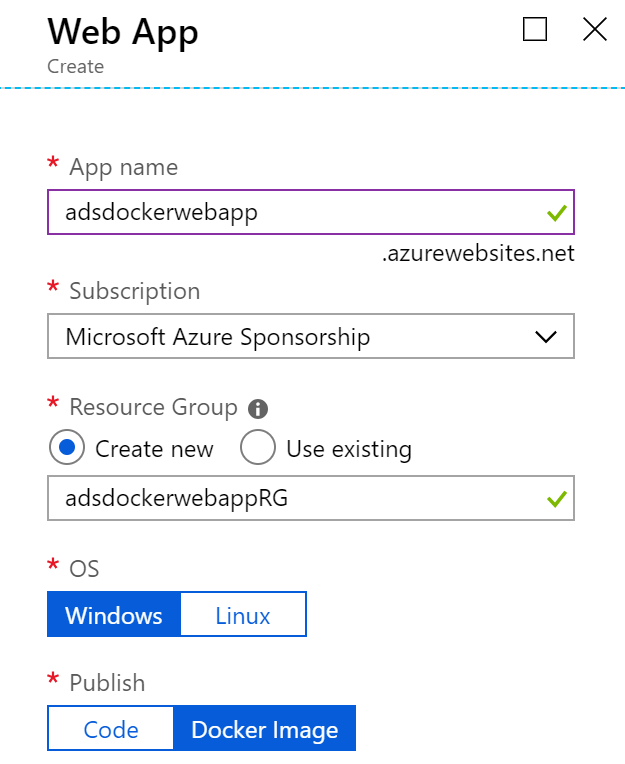
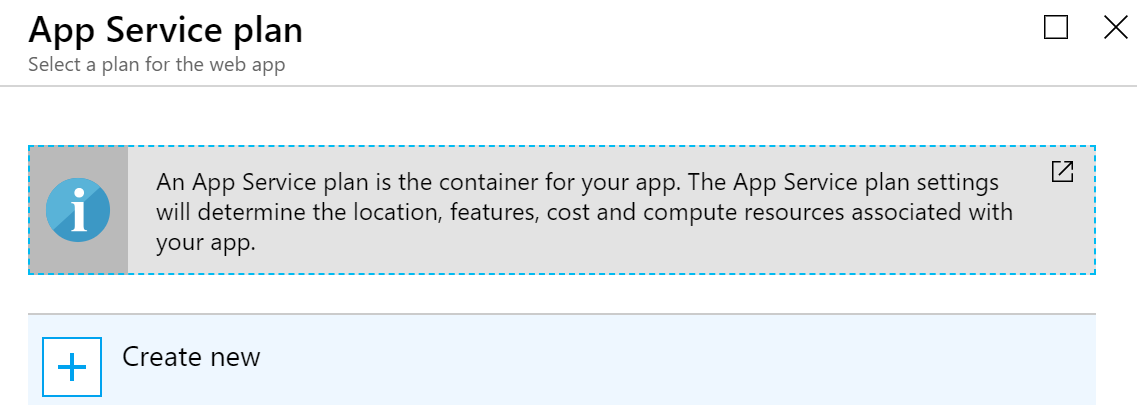
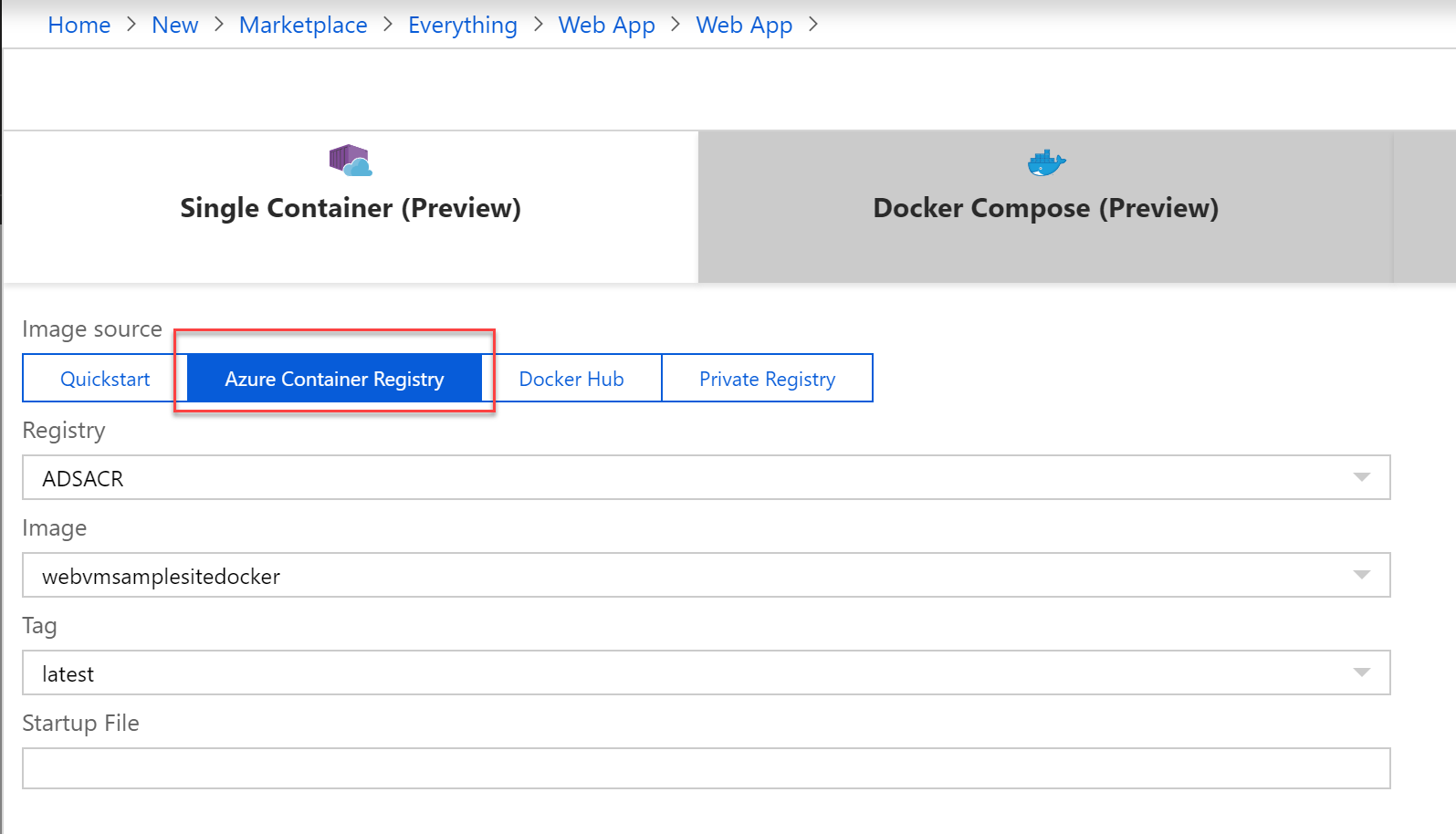
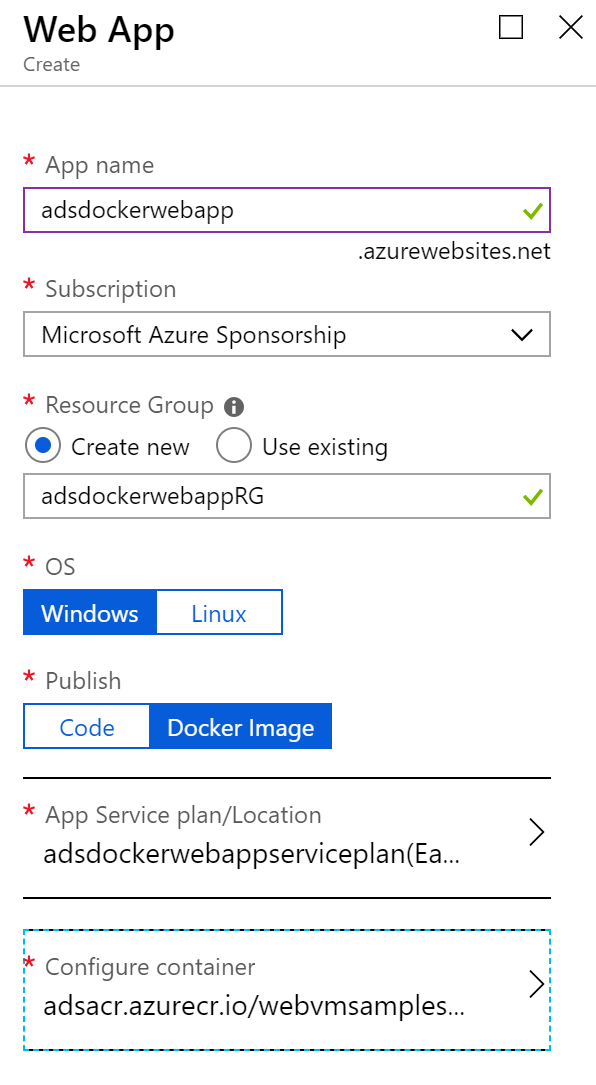
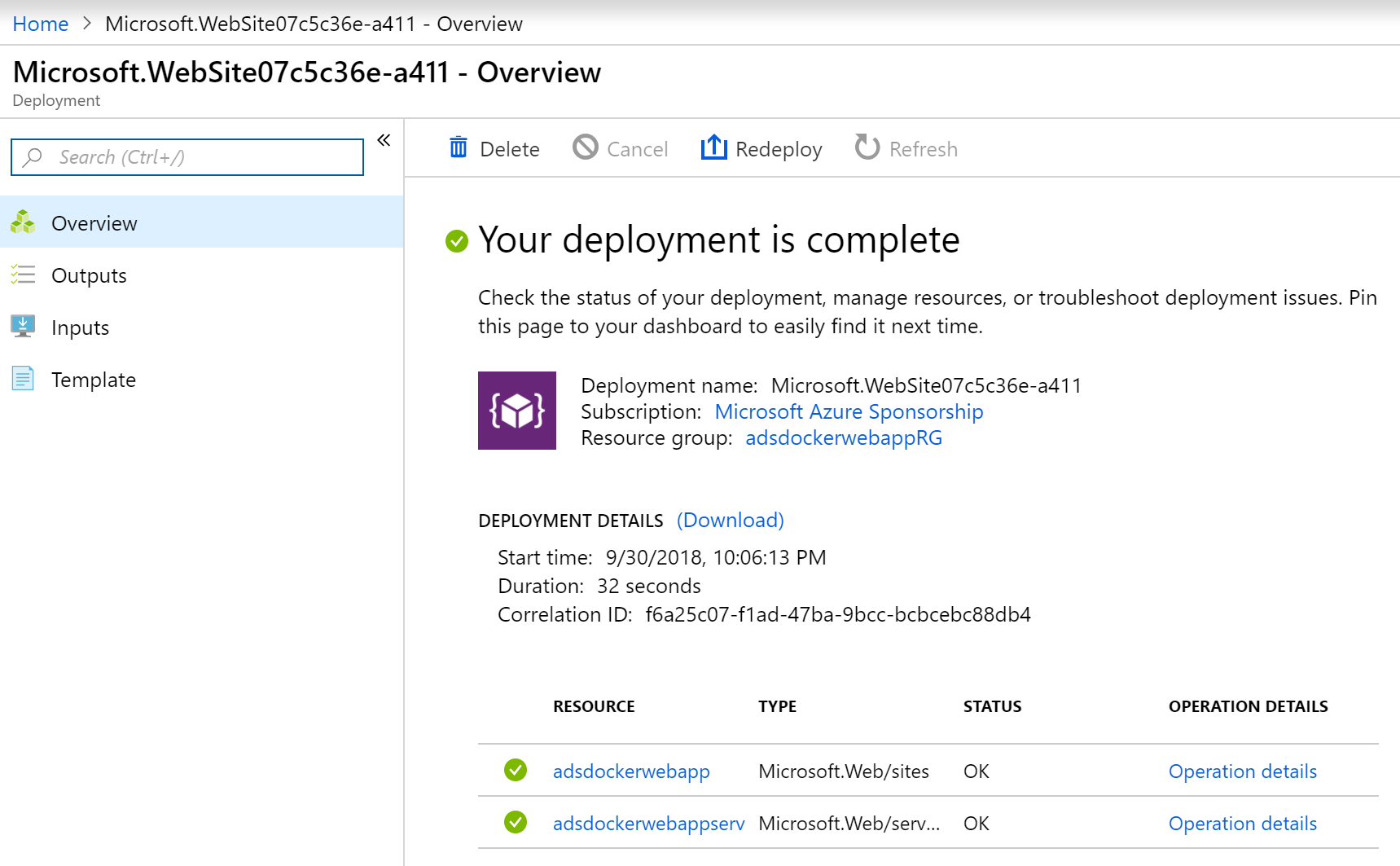
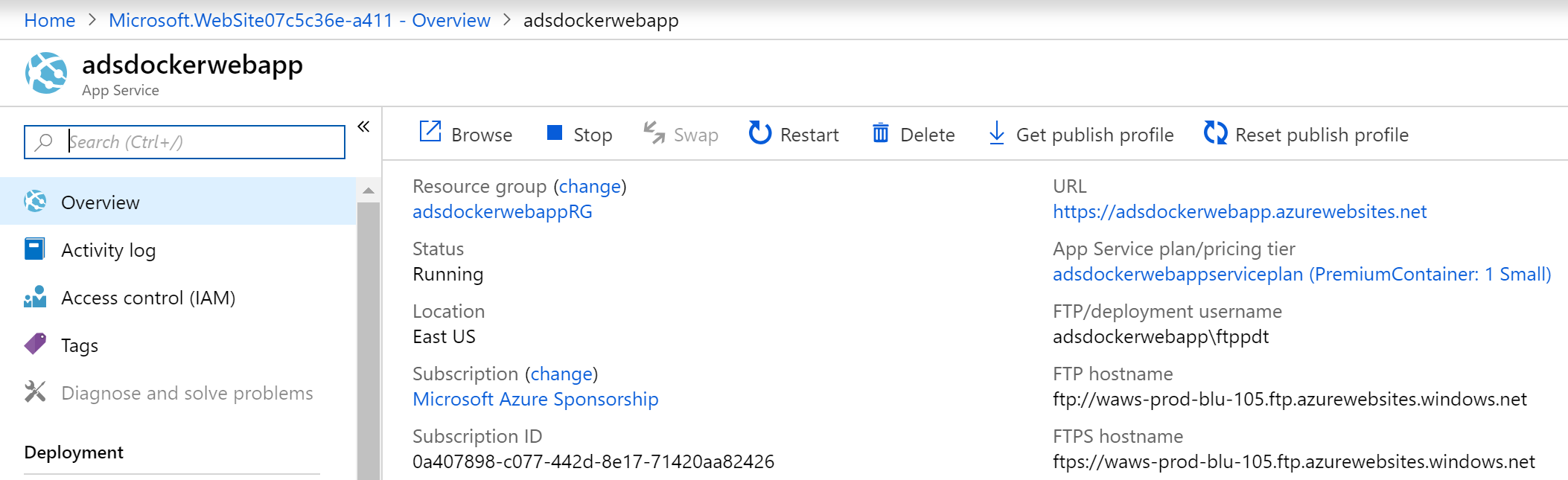
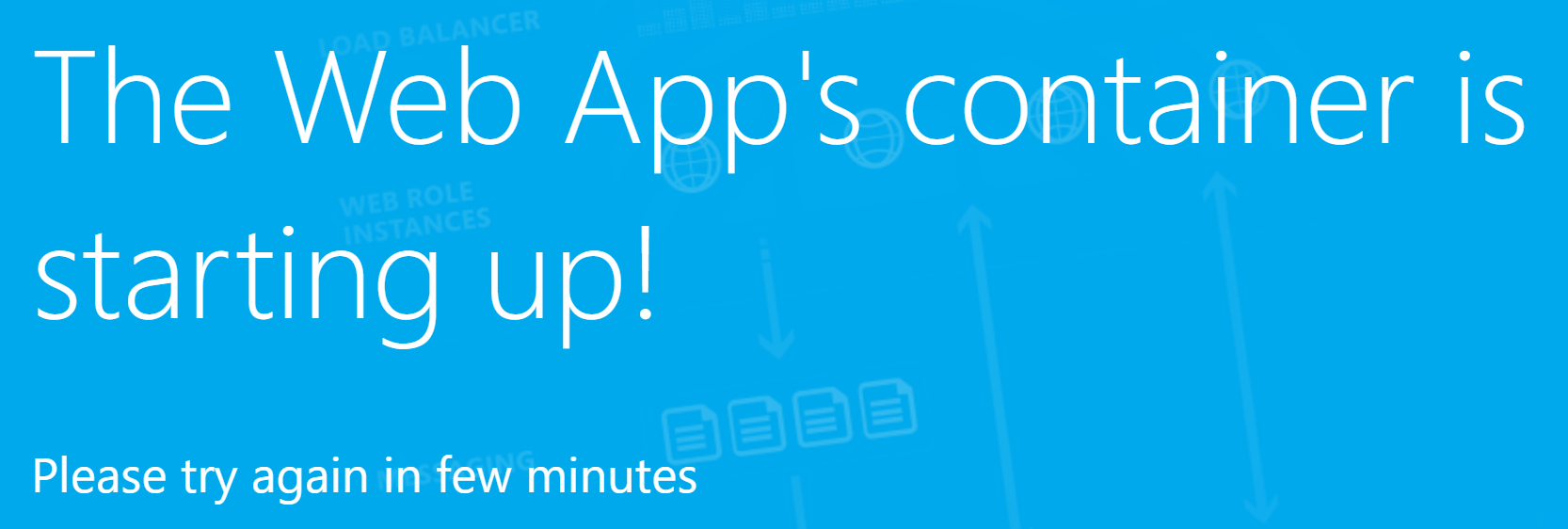
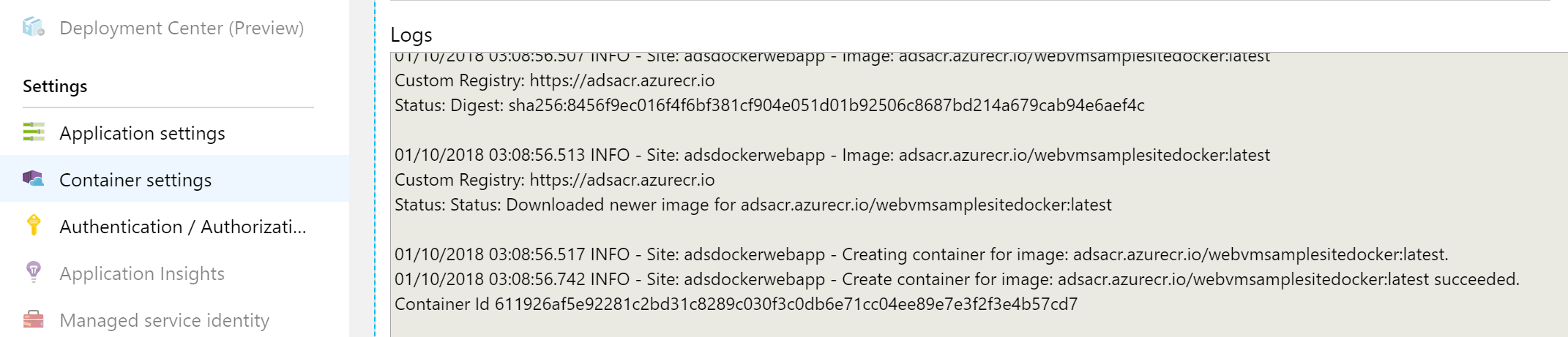
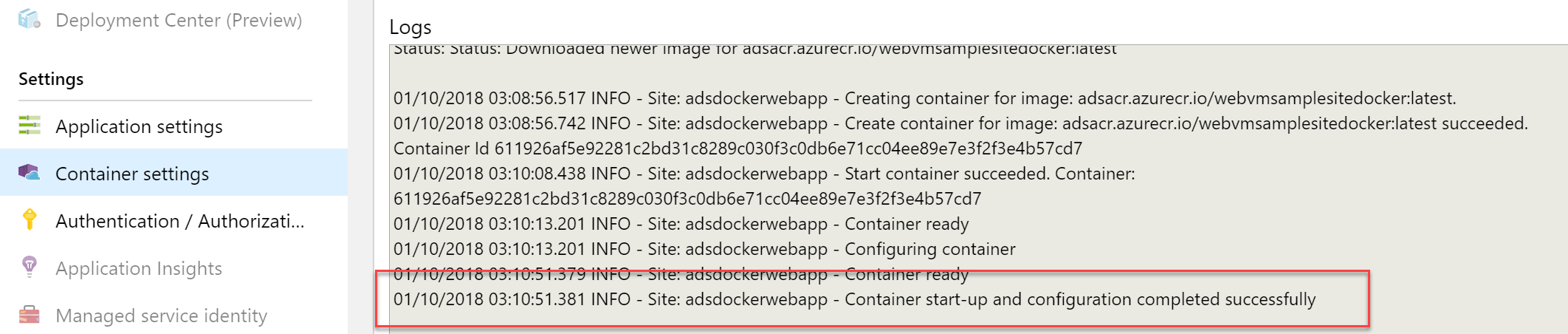
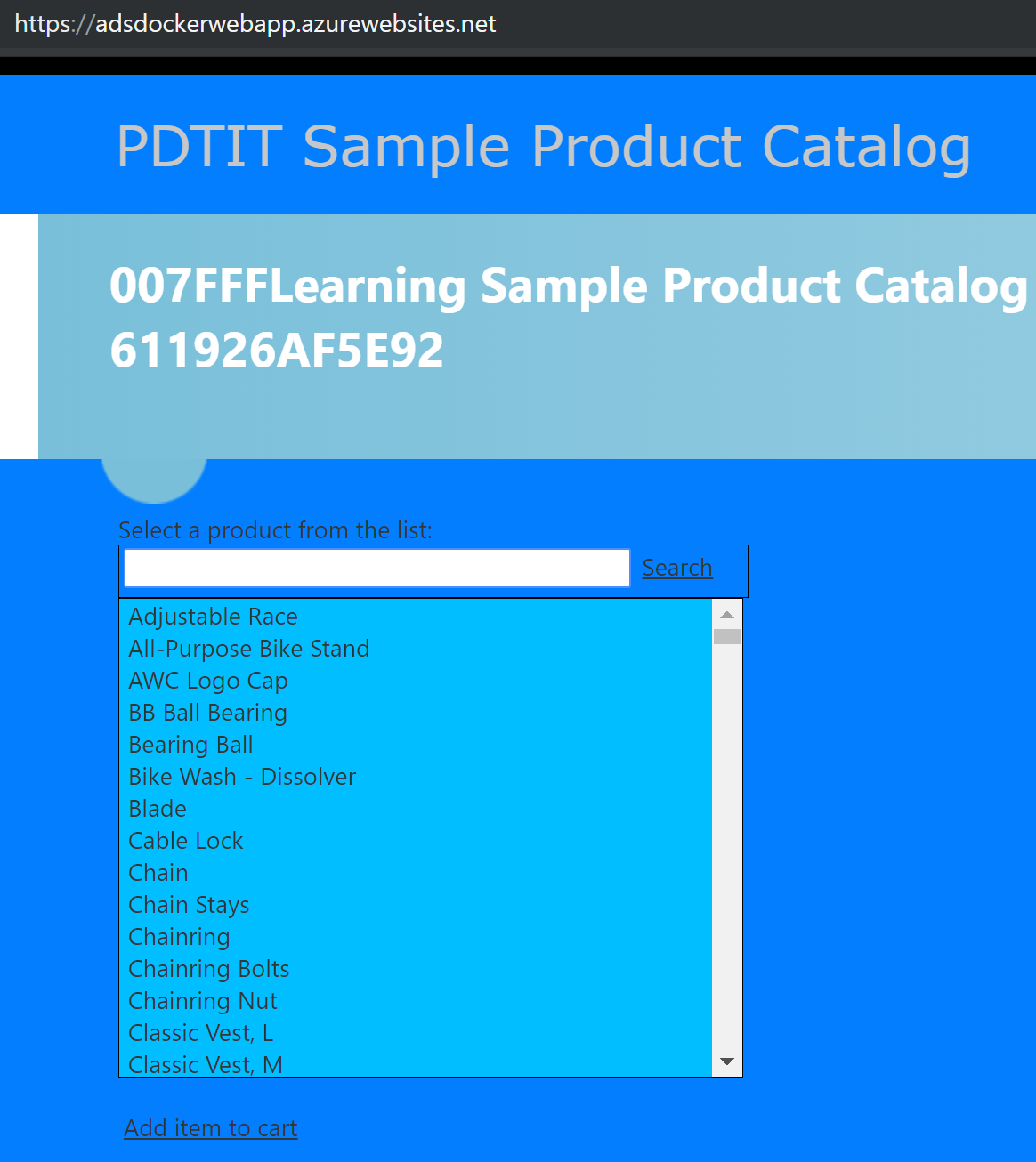
## Task 2: Pushing a Docker image into an Azure Container Registry

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 has 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/webvmsamplesitedocker  
     
   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/webvmsamplesitedocker  
     
     
    **Note: Since this image is about 13GB in size, this initial upload process might take some time.**
3. From the Azure Portal | All Services | Azure Container Registries | select the ACR you created earlier.   
     
   
4. In the blade menu to the leftunder the **Services** section, **click Repositories.** Notice the Docker image “webvmsamplesitedocker” is stored in here.  
     
   
5. **Click the** webvmsamplesitedocker repository, which opens the specific details for this image, exposing its version:  
     
   

## Task 3: 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. You can follow the process from the notification area  
     
   
5. Wait for the deployment process to complete successfully.
6. 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.   
     
   
7. **Copy the IP address** for this Azure Container Instance, or directly browse to it from your internet browser  
     
   
8. 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 4: Deploy a Container using Azure Web Apps

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**:  
   - **Single Container (Preview)  
   - Image Source** Azure Container Registry  
   - **Registry** [SUFFIX]ACR  
   - **image** webvmsamplesitedocker  
   - **Tag** latest  
   
6. **Confirm the creation** by pressing the **Apply** button. Your container web app settings are now like this:  
     
   
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 created an Azure Container Registry in Azure, pushed a local Docker image to the Azure Container Registry, and learned how to run an Azure Container Instance from this Docker image. You also learned how to deploy an Azure Web App for Containers for running your Docker image as an Azure Web App.