

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

### What you will learn

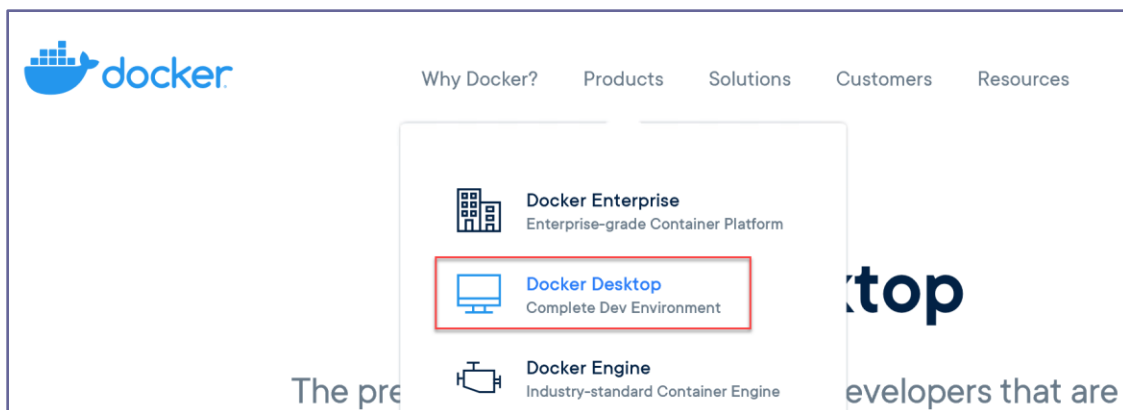
In this lab, we focus on Docker for Windows. Starting with installing the Docker for Windows application, you learn the basics of Docker commands using the Docker Command Line interface. Next, students learn how to 'Dockerize' the legacy ASP.NET code that has been used in former labs. Finally, you learn 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

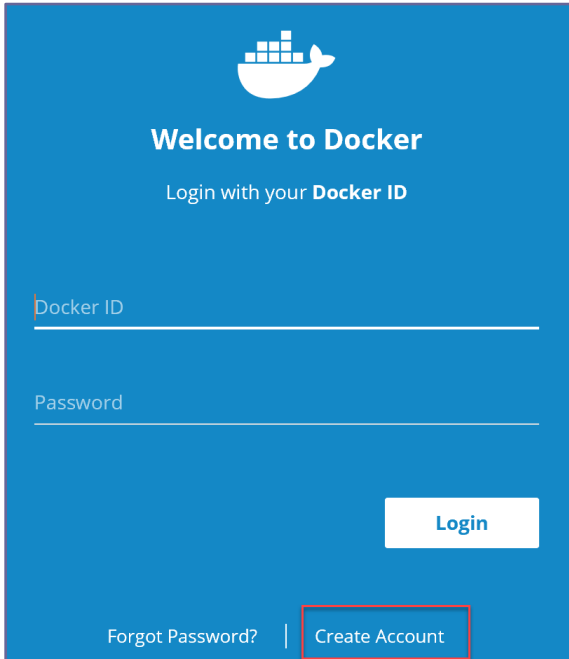
1. If not logged on anymore to the lab-jumpVM, open an RDP session to this Virtual Machine, using labadmin / [L@BadminPa55w.rd](mailto:L@BadminPa55w.rd) as credentials.
2. Connect to the Docker website <http://www.docker.com>; Here, Click **Products**, and select **Docker Desktop**.



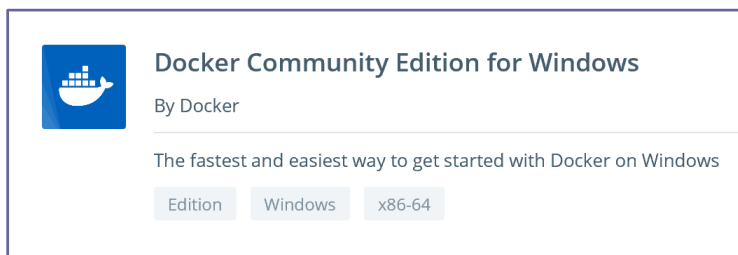
3. Click "Download for Windows".
4. This redirects you to the Docker Store. (the direct link at the time of writing this lab guide is <https://store.docker.com/editions/community/docker-ce-desktop-windows>)
5. Before you can download the source install files, you need to create a **Docker Login**. Since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

details here.

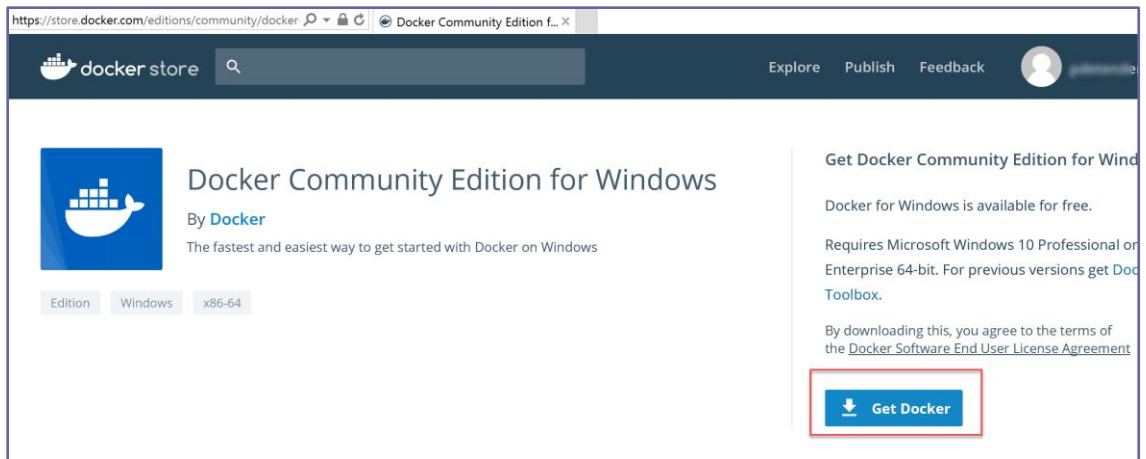
6. Click the **Login to Portal** button; in the login portal, choose **Create Account**.



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**.



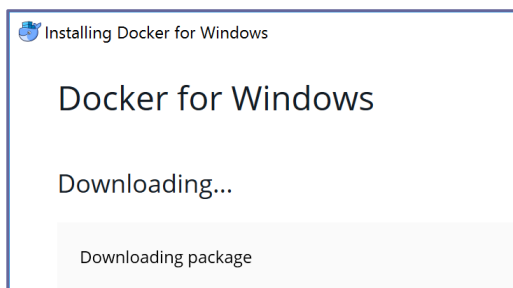
10. Click on the **object title**, which opens the download page for this solution. Click the **Get Docker** button.



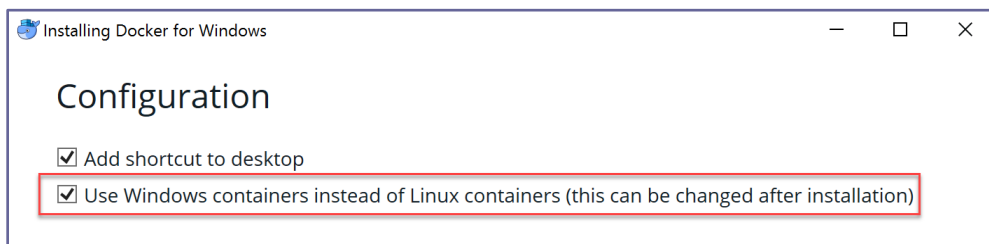
11. **Save** the install files to the local machine's Downloads folder; wait for the download to complete, and run the actual install process.



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



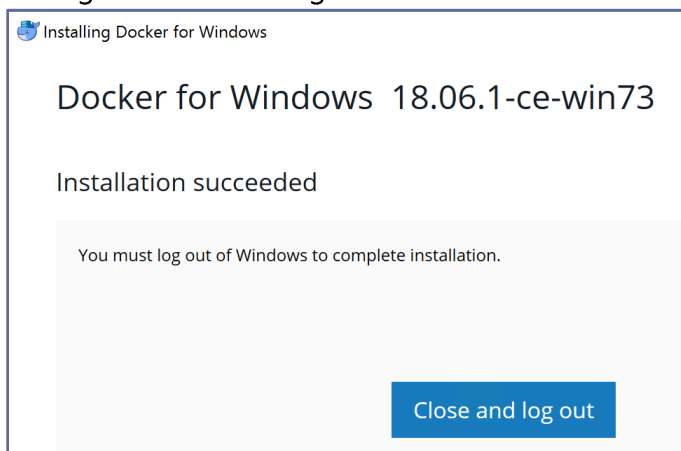
13. Wait for the background download to complete, until you see the **Configuration** step of the install wizard. **Here, select "Use Windows containers instead of Linux containers"**. Press OK to continue.



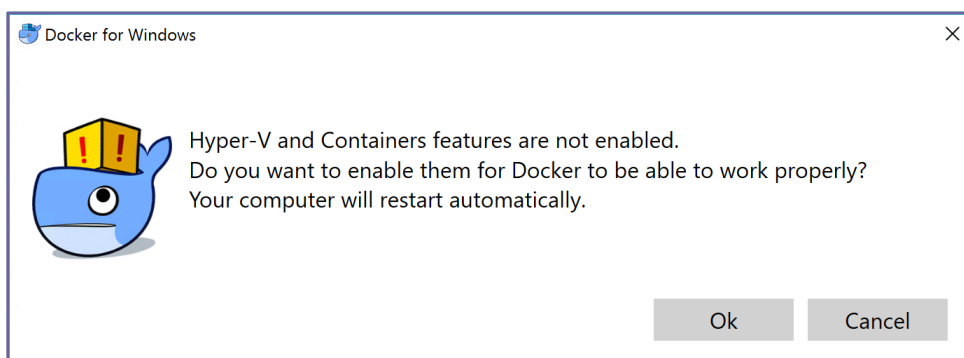
14. The Docker install will update the install packages and run the actual tool installation. Wait for this step to complete.



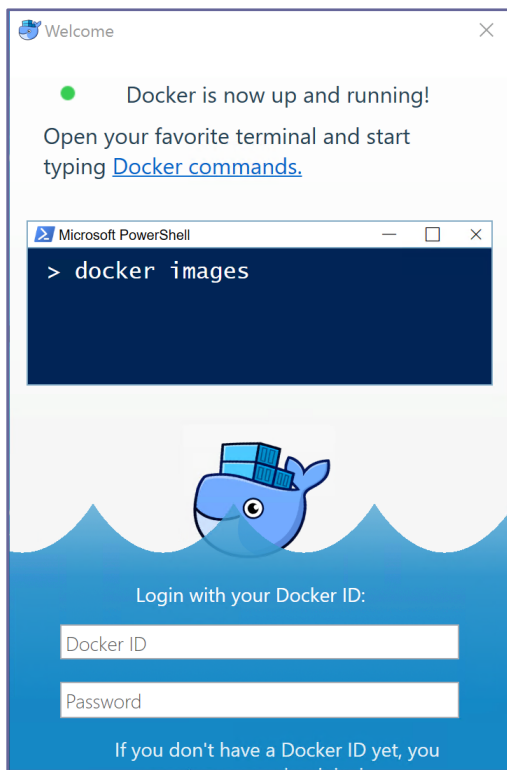
15. **Wait** for the installation to **complete**; once prompted, **log out** from the RDP session, by clicking the Close and Log out button.



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
```

```
Select Administrator: Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

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

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

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

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

C:\Users\labadmin>
```

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

```
docker run hello-world
```

Some explanation what this command executes:

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

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

C:\Users\labadmin>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
bce2fbc256ea: Extracting [==>] 12.26MB/252.7MB
4a14bdf6da80: Download complete
842bcb9bd6e: Download complete
3c15d2487d42: Download complete
```

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

```
Administrator: Command Prompt

bce2fbc256ea: Pull complete
4a14bdf6da80: Pull complete
842bcb9bd6e: Pull complete
3c15d2487d42: Pull complete
Digest: sha256:0add3ace90ecb4adbf777e9aacf18357296e799f81cab9fde470971e499788
Status: Downloaded newer image for hello-world:latest

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

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

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

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

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

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

## Task 2: Operating Docker from the command line

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

`docker ps`

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

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

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

```
docker ps -a
```

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

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

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

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

```
docker images
```

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

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

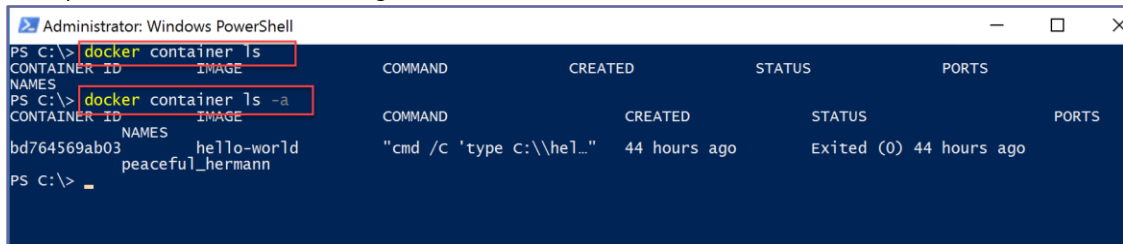
C:\Users\labadmin>
```



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

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

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



The screenshot shows a Windows PowerShell window with the following output:

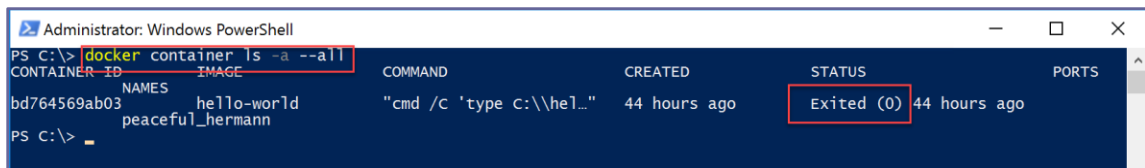
```
PS C:\> docker container ls
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS
bd764569ab03   hello-world "cmd /c 'type c:\\hel..." 44 hours ago    Exited (0)     44 hours ago
peaceful_hermann
```

The second command, `docker container ls -a`, shows the same output, including the container that has exited.

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
```



The screenshot shows a Windows PowerShell window with the following output:

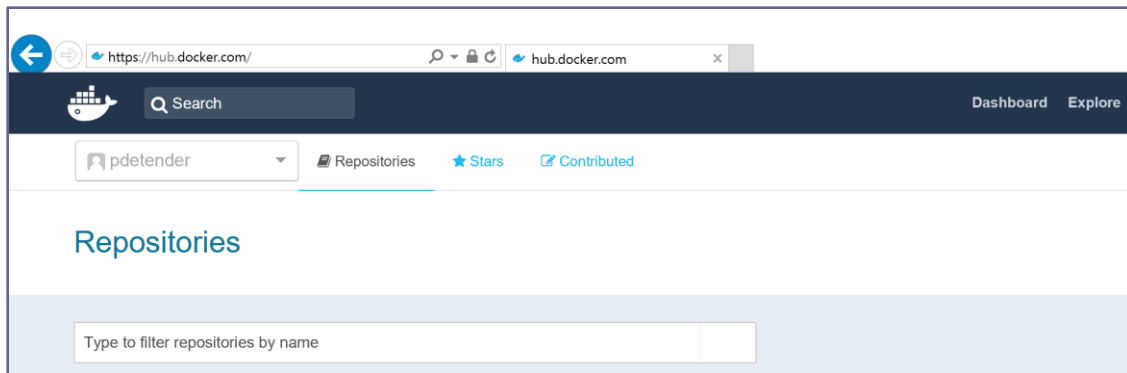
```
PS C:\> docker container ls -a --all
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS
bd764569ab03   hello-world "cmd /c 'type c:\\hel..." 44 hours ago    Exited (0)     44 hours ago
peaceful_hermann
```

The output is the same as the previous screenshot, but the command used is `docker container ls -a --all`.

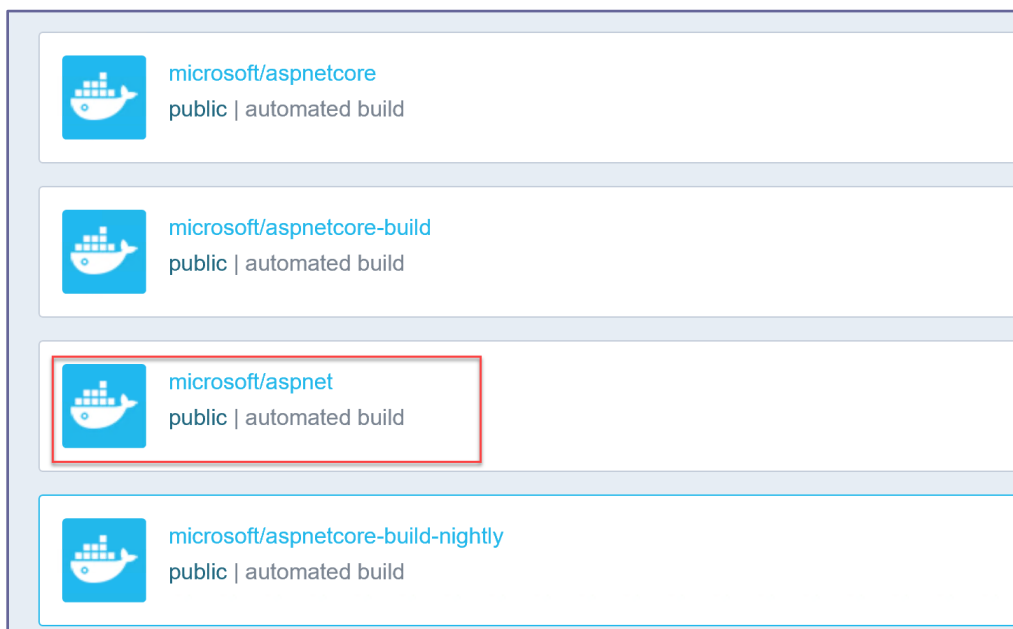
Which shows you this container is in an exited (=stopped) state since 44 hours. (if you are wondering where the 44 hours comes from, no worries, lab guide authors have a life too 😊).

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.

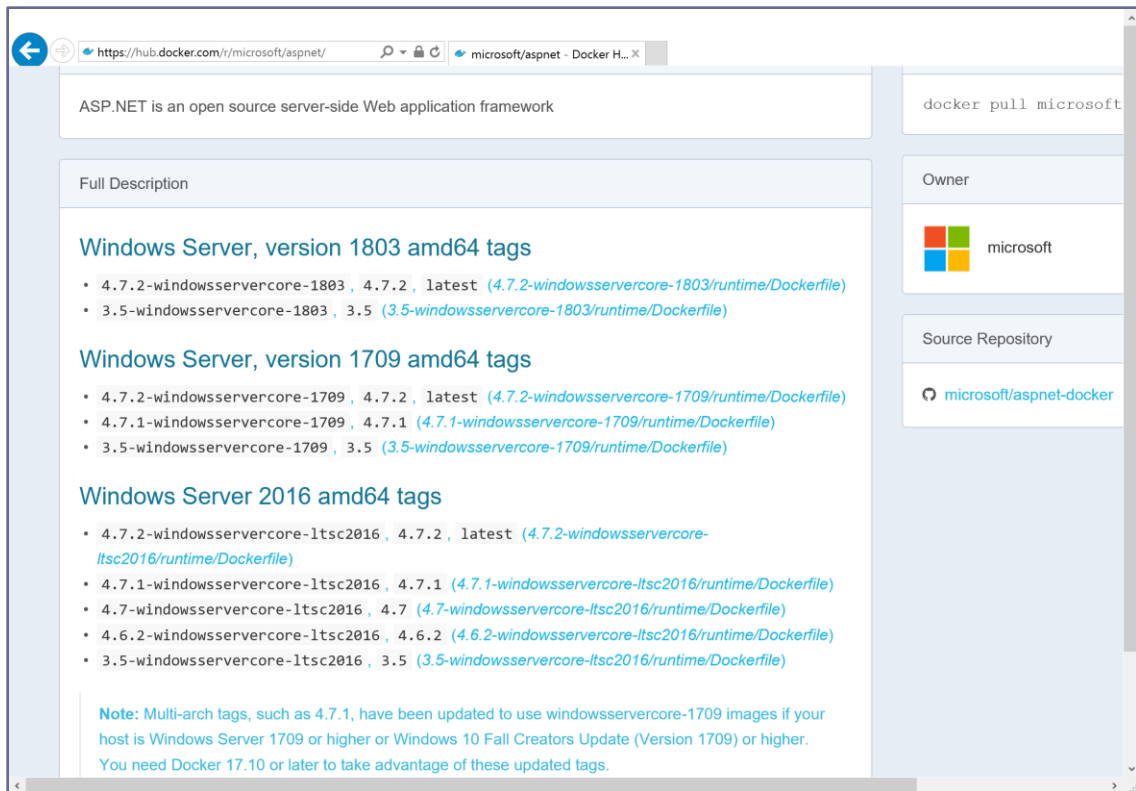
7. Connect to [hub.docker.com](https://hub.docker.com) from your internet browser, and log on with your previously created DockerID credentials.



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



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



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

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

```
docker pull microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
```

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

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

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

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

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

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

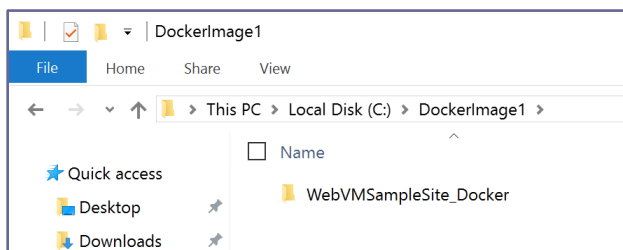
1. From the Docker command line (Command Prompt or PowerShell), run the following Docker command:

```
docker images
```

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

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

- 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).



- 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)
```

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

Directory: C:\DockerImage1

Mode                LastWriteTime         Length Name
----                -
-a----          9/30/2018   6:46 PM              0 Dockerfile
PS C:\DockerImage1>
```

4. **Open** the Dockerfile file in Notepad for easy editing. From PowerShell, you can run “notepad Dockerfile”, or browse to the file from File explorer and open it with Notepad.
5. **Type in** the following lines **into the Notepad window** (copy/paste might work, but sometimes injects garbage code, which breaks the file – therefore, typing is the safest option).

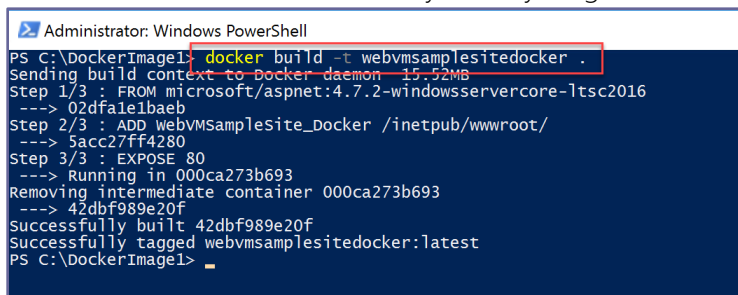
```
FROM microsoft/aspnet:4.7.2-windowsservercore-ltsc2016
ADD WebVMSampleSite_Docker /inetpub/wwwroot/
EXPOSE 80
```

and **save the file**.

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

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

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



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

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

```
docker images
```

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

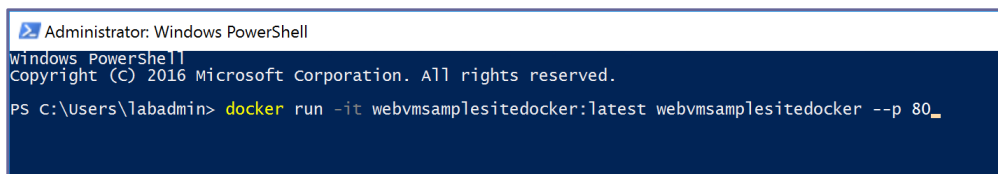
```
PS C:\DockerImage1> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
webvmsamplesitedocker latest             42dbf989e20f       4 minutes ago      13.6GB
microsoft/aspnet     4.7.2-windowsservercore-ltsc2016 02dfa1e1baeb       2 weeks ago       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

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

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

Where:

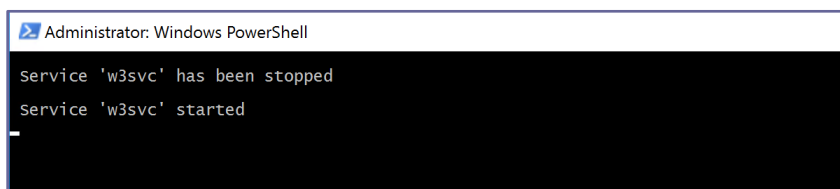
- "docker run" start the container run process
- "-it" defines the container needs to run in interactive mode (output)
- "webvm...:latest" the name of the image to use, note the tag
- "webvm..." the name for the container
- "--p 80" run the container on port 80



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

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

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



```
Administrator: Windows PowerShell

service 'w3svc' has been stopped
service 'w3svc' started
_
```

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

```
docker container ls
```

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

This shows our running container.

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

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

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

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

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

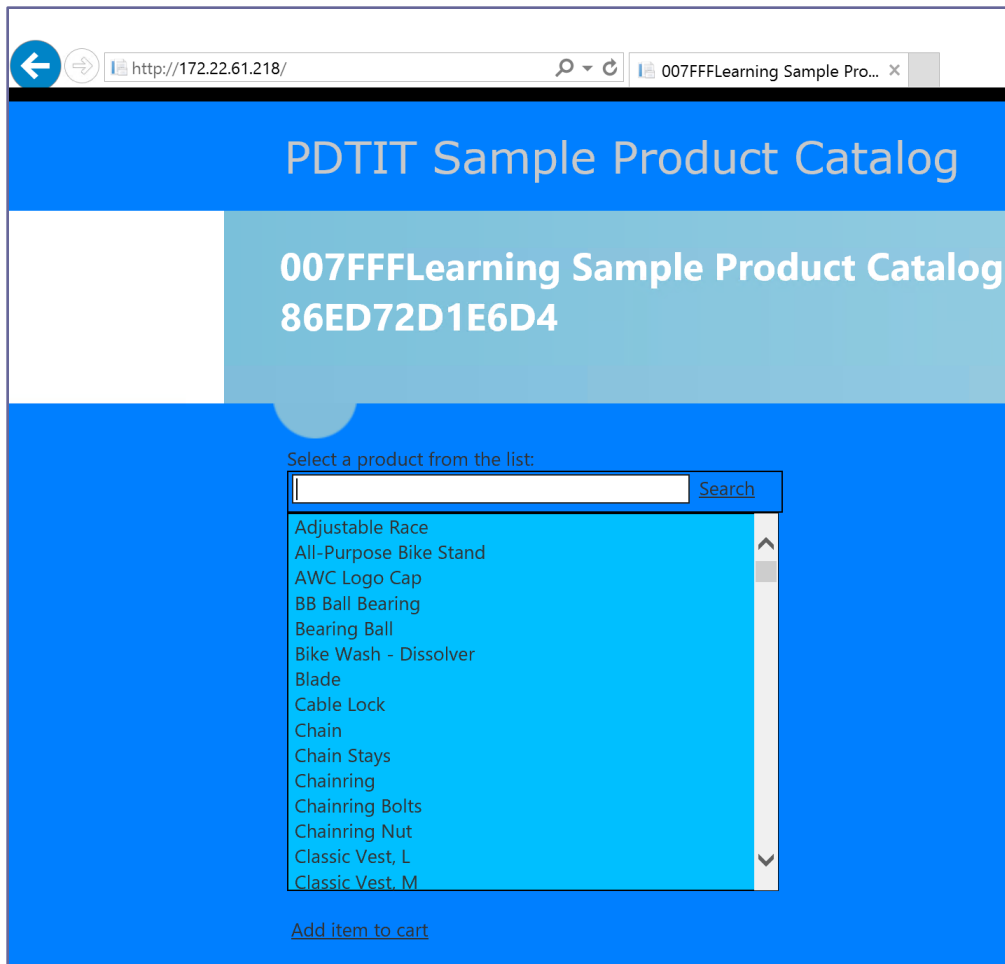


```

    "NetworkSettings": {
      "Bridge": "",
      "SandboxID": "86ed72d1e6d4c4446a52cf20f393f6c27c1cbd3039cb96b3d0a058263a5b1ddc",
      "HairpinMode": false,
      "LinkLocalIPv6Address": "",
      "LinkLocalIPv6PrefixLen": 0,
      "Ports": {
        "80/tcp": null
      },
      "SandboxKey": "86ed72d1e6d4c4446a52cf20f393f6c27c1cbd3039cb96b3d0a058263a5b1ddc",
      "SecondaryIPAddresses": null,
      "SecondaryIPv6Addresses": null,
      "EndpointID": "",
      "Gateway": "",
      "GlobalIPv6Address": "",
      "GlobalIPv6PrefixLen": 0,
      "IPAddress": "",
      "IPPrefixLen": 0,
      "IPv6Gateway": "",
      "MacAddress": "",
      "Networks": {
        "nat": {
          "IPAMConfig": null,
          "Links": null,
          "Aliases": null,
          "NetworkID": "075b37fba26fba20fb9eb4ee9fdea8580e401031b61b8d767a33fbc09fec7373",
          "EndpointID": "1acc8f1118e6f382ca3c96d6e61ee66e527f940d03601f62207a23aede66ea53",
          "Gateway": "172.22.48.1",
          "IPAddress": "172.22.61.218",
          "IPPrefixLen": 16,
          "IPv6Gateway": "",
          "GlobalIPv6Address": "",
          "GlobalIPv6PrefixLen": 0,
          "MacAddress": "00:15:5d:66:7f:0c",
          "DriverOpts": null
        }
      }
    }
  }
}

```

12. Here, check for the IPAddress under the Networks / nat section specifically. This reveals the IP-address that is used by our running container.
13. From your internet browser, connect to this IP-address, which shows you the running web application. Yeah! (Notice it also refers to 86ED... as hostname, which is the technical ID of our container)



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

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

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

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

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

## 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.

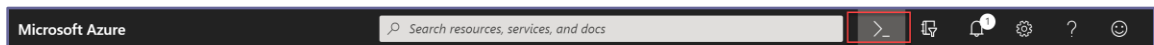
### Time Estimate

This lab should take about an hour to complete.

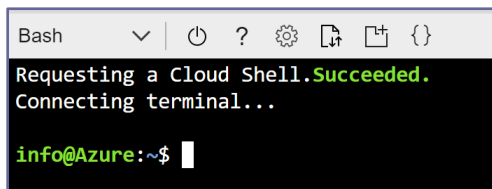
### Task 1: Creating an Azure Container Registry using Azure CloudShell

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
```

```
info@Azure:~$ az group create --name ADS-DockerRG --location EastUS2
{
  "id": "/subscriptions/0a407898-c077-442d-8e17-71420aa82426/resourceGroups/ADS-DockerRG",
  "location": "eastus2",
  "managedBy": null,
  "name": "ADS-DockerRG",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
info@Azure:~$
```

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

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

```
info@Azure:~$ az acr create --resource-group ADS-dockerRG --name ADSACR --sku Basic --admin-enabled true
{
  "adminUserEnabled": true,
  "creationDate": "2018-09-30T21:20:24.207509+00:00",
  "id": "/subscriptions/0a407898-c077-442d-8e17-71420aa82426/resourceGroups/ADS-dockerRG/providers/Microsoft.ContainerRegistry/registries/ADSACR",
  "location": "eastus2",
  "loginServer": "adsacr.azurecr.io",
  "name": "ADSACR",
  "provisioningState": "Succeeded",
  "resourceGroup": "ADS-dockerRG",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  }
}
```

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

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

```
Bash
info@Azure:~$ az acr login --name adsacr --resource-group ads-dockerRG
This command requires running the docker daemon, which is not supported in Azure Cloud Shell.
info@Azure:~$
```

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

# Install Azure CLI on Windows

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

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

## Install or update

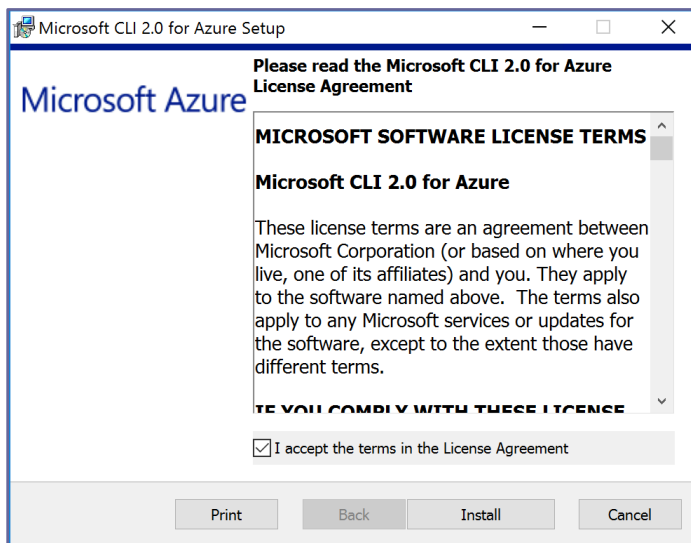
The MSI distributable is used for installing, updating, and uninstalling the Azure CLI on Windows.

[Download the MSI installer](#) >

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



- The Microsoft CLI 2.0 for Azure Setup Installer launches



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

az

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

PS C:\Users\labadmin> az

Welcome to Azure CLI!
Use 'az -h' to see available commands or go to https://aka.ms/cli.

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

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

AZURE

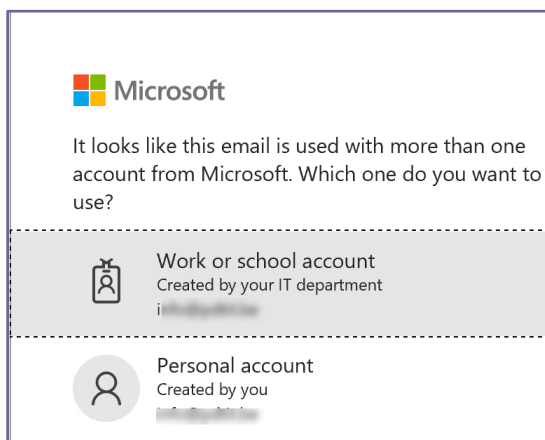
Welcome to the cool new Azure CLI!
Use 'az --version' to display the current version.
Here are the base commands:
```

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

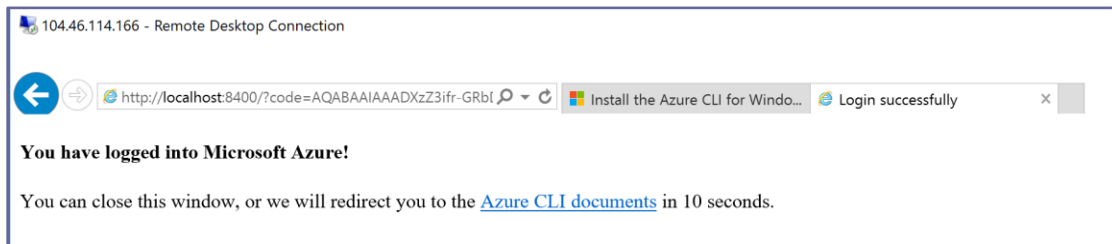
az login

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

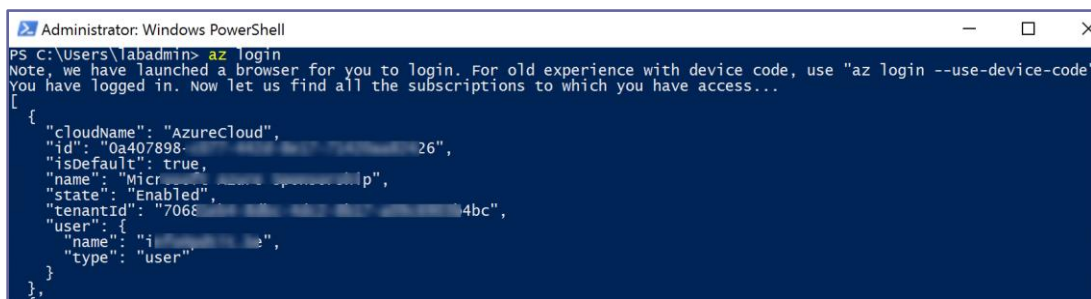
13. This opens your internet browsers, and prompts for your Azure admin credentials:



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

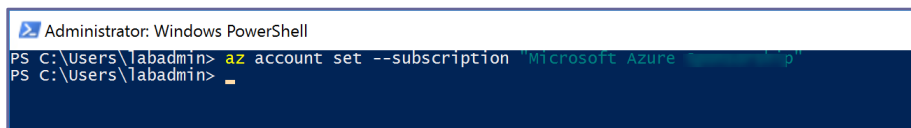


15. You can close the internet browser.
16. When you go back to the PowerShell window, it will show you the JSON output of your Azure subscription, related to this Azure admin user:



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
```



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

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

Home > Container registries			
<b>Container registries</b> infopdtit (Default Directory)			
+ Add   Edit columns   Refresh   Assign tags			
 <a href="#">Build, Run, Push and Patch containers in Azure with ACR Tasks</a>			
<b>Subscriptions:</b> Microsoft Azure Sponsorship – Don't see a subscription? <a href="#">Open Directory</a> + <a href="#">Subscription settings</a>			
Filter by name...   All resource groups   All locations   All tags			
3 items			
<input type="checkbox"/>	NAME ↑↓	TYPE ↑↓	RESOURCE GROUP ↑↓
<input type="checkbox"/>	ADSACR	Container registry	ADS-dockerrg
			East US 2

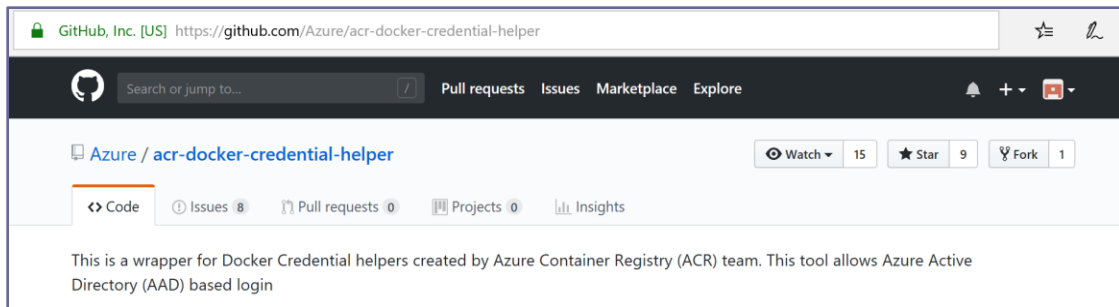
20. Next, we need to authenticate to the Azure Container Registry itself, using the following command:

```
az acr login --name ADSACR --resource-group ADS-DockerRG
```

```
Administrator: Windows PowerShell
PS C:\Users\labadmin> az acr login --name ADSACR --resource-group ADS-DockerRG
Login Succeeded
PS C:\Users\labadmin>
```

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>



## ACR Docker Credential Helper

The ACR Docker Credential Helper allows users to sign-in to the Azure Container Registry service using their Azure Active Directory (AAD) credentials. This credential helper is in charge of ensuring that the stored credentials are valid, and when required it also renews the credentials for a repository.

For now, this credential helper works in tandem with the Azure CLI, which is required in order to initiate the credential flow. Once you've successfully logged in to your container registry with the Azure CLI, the credential helper administers the life cycle of your locally stored credential.

### Prerequisites

- [Docker](#)
- [Azure CLI](#)

### Installation

For Windows, run the [powershell installation script](#) in administrator mode:

```
iex ([System.Text.Encoding]::UTF8.GetString((Invoke-WebRequest -Uri https://aka.ms/acr/installaad/win).Content))
```

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

```
Administrator: Windows PowerShell
PS C:\users\p\Downloads> .\install.ps1

Security warning
Run only scripts that you trust. While scripts from the internet can be useful, this script can potentially harm your
computer. If you trust this script, use the Unblock-File cmdlet to allow the script to run without this warning
message. Do you want to run C:\users\p\Downloads\install.ps1?
[D] Do not run [R] Run once [S] Suspend [?] Help (default is "D"): r
ACR Credential Helper currently does not support Windows Credential Manager because Windows Credential Manager only supp
ort saving tokens with less than 2.5KB blob size.
1. A json file will be used to store all your credentials.
2. You will have to re-login to any existing Docker registry after the installation.
Continue? [Y/n]: y

StatusCode      : 200
StatusDescription : OK
Content         : {80, 75, 3, 4...}
RawContent      : HTTP/1.1 200 OK
                  Content-MD5: N9ZEaAmDPB0IAyml+wTz2A==
                  x-ms-request-id: 83a0a5d2-201e-008e-1056-48613a000000
                  x-ms-version: 2014-02-14
                  x-ms-lease-status: unlocked
                  x-ms-lease-state: available
                  x-ms-...
Headers         : {[Content-MD5, N9ZEaAmDPB0IAyml+wTz2A==], [x-ms-request-id, 83a0a5d2-201e-008e-1056-48613a000000],
                  [x-ms-version, 2014-02-14], [x-ms-lease-status, unlocked]...}
RawContentLength : 4161209

PSPPath          : Microsoft.PowerShell.Core\FileSystem:C:\users\p\Downloads\deleteme
PSParentPath     : Microsoft.PowerShell.Core\FileSystem:C:\users\p\Downloads
PSChildName      : deleteme
PSDrive          : C
PSProvider       : Microsoft.PowerShell.Core\FileSystem
PSIsContainer    : True
Name             : deleteme
FullName         : C:\users\p\Downloads\deleteme
Parent           : Downloads
Exists           : True
Root             : C:\
Extension        :
CreationTime     : 9/9/2018 6:00:10 PM
CreationTimeUtc  : 9/9/2018 4:00:10 PM
LastAccessTime   : 9/9/2018 6:00:10 PM
LastAccessTimeUtc : 9/9/2018 4:00:10 PM
LastWriteTime    : 9/9/2018 6:00:10 PM
LastWriteTimeUtc : 9/9/2018 4:00:10 PM
Attributes       : Directory
Mode             : d-----
BaseName         : deleteme
```

22. 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)

```
Administrator: Windows PowerShell
PS C:\Users\labadmin> docker tag 42db ADSACR.azurecr.io/webvmsamplesitedocker
PS C:\Users\labadmin> docker images
REPOSITORY                                TAG                IMAGE ID           CREATED           SIZE
ADSACR.azurecr.io/webvmsamplesitedocker   latest            42dbf989e20f      3 hours ago      13.1GB
webvmsamplesitedocker                     latest            42dbf989e20f      3 hours ago      13.1GB
microsoft/aspnet                          4.7.2-windowsservercore-ltsc2016  02dfa1e1baeb      2 weeks ago      13.1GB
microsoft/windowsservercore               latest            0534c30e0e3e      2 weeks ago      10.1GB
hello-world                               latest            476f8d625669      3 weeks ago      1.1GB
PS C:\Users\labadmin>
```

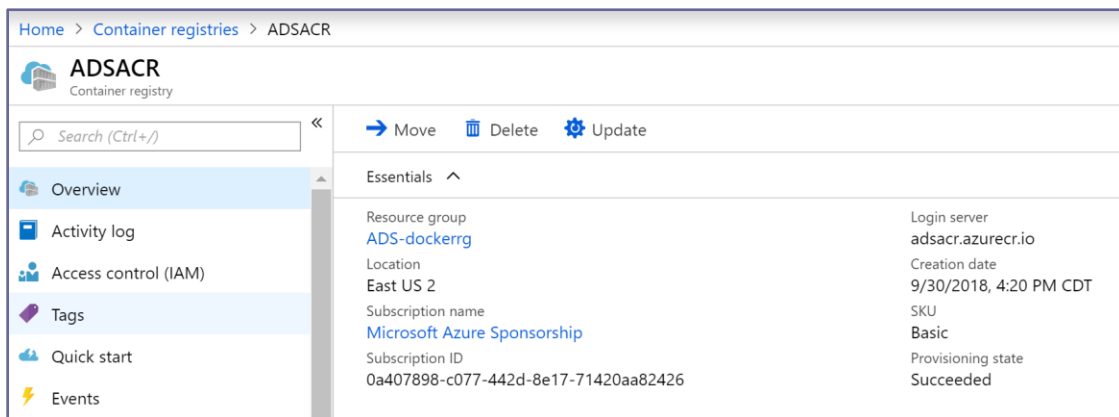
- Execute the following command to upload this image to the Azure Container Registry:

```
docker push [SUFFIX]ACR.azurecr.io/webvmsamplesitedocker
```

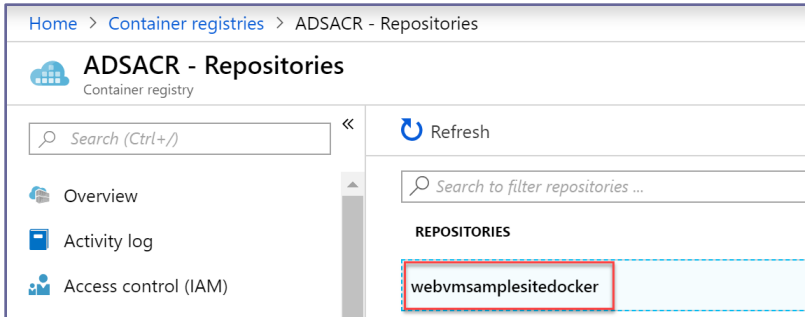
```
PS C:\Users\labadmin> docker push ADSACR.azurecr.io/webvmsamplesitedocker
The push refers to repository [ADSACR.azurecr.io/webvmsamplesitedocker]
93ab73aeac1f: Pushed
df94fe09aa40: Pushed
5803bbdcb6bf: Pushed
5246598eac8a: Pushed
8db0e5c054e8: Pushed
fb512a3ddb0d: Pushing [=====>] 27.6MB/193.2MB
2cc3f8bc43c8: Pushing [=====>] 29.31MB/347.6MB
9209a4810619: Pushed
_98389a318d8: Pushing [==>] 50.39MB/998.7MB
c60899c9c3f1: Pushing [=====>] 54.27kB
88ca726269bd: waiting
6aa7c5e61edd: waiting
7332b29f30e0: waiting
f358be10862c: waiting
```

Note: Since this image is about 13GB in size, this initial upload process might take some time.

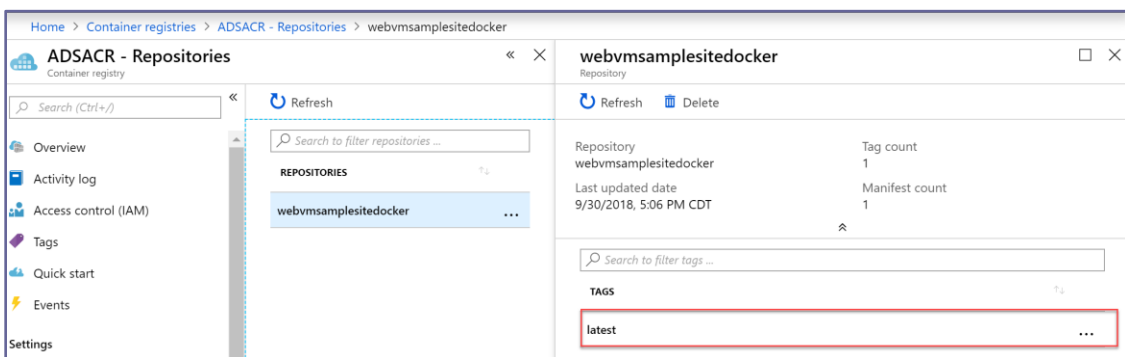
- From the Azure Portal | All Services | Azure Container Registries | select the ACR you created earlier.



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

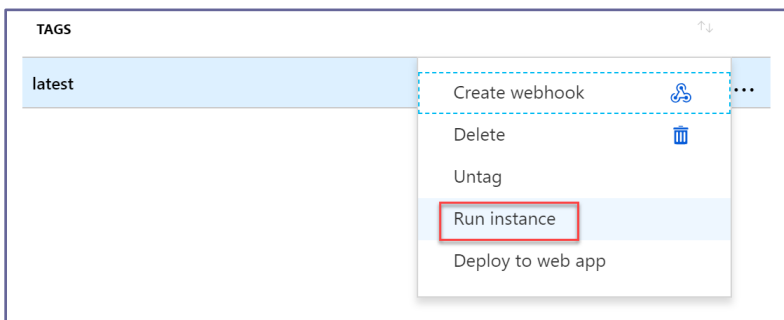


5. Click the `webvmsamplesitedocker` repository, which opens the specific details for this image, exposing its version:



### 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)

**webvmsamplesitedocker** Repository

Refresh Delete

Repository	Tag count
webvmsamplesite...	1
Last updated date	Manifest count
9/30/2018, 5:06 P...	1

Search to filter tags ...

**TAGS**

latest

**Create container instance**

\* Container name  
samplesitecontainer ✓

Container image  
adsacr.azurecr.io/webvmsamplesitedocker:la...

OS type  
Linux Windows

\* Subscription  
Microsoft Azure Sponsorship

\* Resource group  
ADS-DockerRG  
[Create new](#)

\* Location  
East US 2

Number of cores  
1

\* Memory (GB)  
1.5

Public IP address  
Yes No

\* Port  
80

OK

3. Press **OK** to have the Container Instance created.
4. You can follow the process from the notification area

Home > Microsoft.ContainerInstance - Overview

### Microsoft.ContainerInstance - Overview

Deployment

Search (Ctrl+[/](#))


Delete Cancel Redeploy Refresh

**Overview**

- Outputs
- Inputs
- Template


**... Your deployment is underway**

Check the status of your deployment, manage resources, or troubleshoot deployment issues. Pin this page to your dashboard to easily find it next time.

 Deployment name: Microsoft.ContainerInstance  
Subscription: [Microsoft Azure Sponsorship](#)  
Resource group: [ADS-DockerRG](#)

**DEPLOYMENT DETAILS** ([Download](#))

Start time: 9/30/2018, 5:16:20 PM  
Duration: 27 seconds  
Correlation ID: 68b378e9-c93d-45bb-b997-48ee3f4656d4

RESOURCE	TYPE	STATUS	OPERATION DETAILS
 <a href="#">samplesitecontainer</a>	Microsoft.ContainerI...	Created	<a href="#">Operation details</a>

- Wait for the deployment process to complete successfully.
- 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.

Home > Microsoft.ContainerInstance - Overview > samplesitecontainer

### samplesitecontainer

Container instances

Search (Ctrl+[/](#))

Restart Stop Delete Refresh

**Overview**

- Activity log
- Access control (IAM)
- Tags

**Settings**

- Containers
- Properties
- Locks

Resource group [\(change\)](#)  
[ADS-DockerRG](#)

Status  
Running

Location  
East US 2

Subscription [\(change\)](#)  
[Microsoft Azure Sponsorship](#)

Subscription ID  
0a407898-c077-442d-8e17-71420aa82426

Tags [\(change\)](#)  
[Click here to add tags](#)

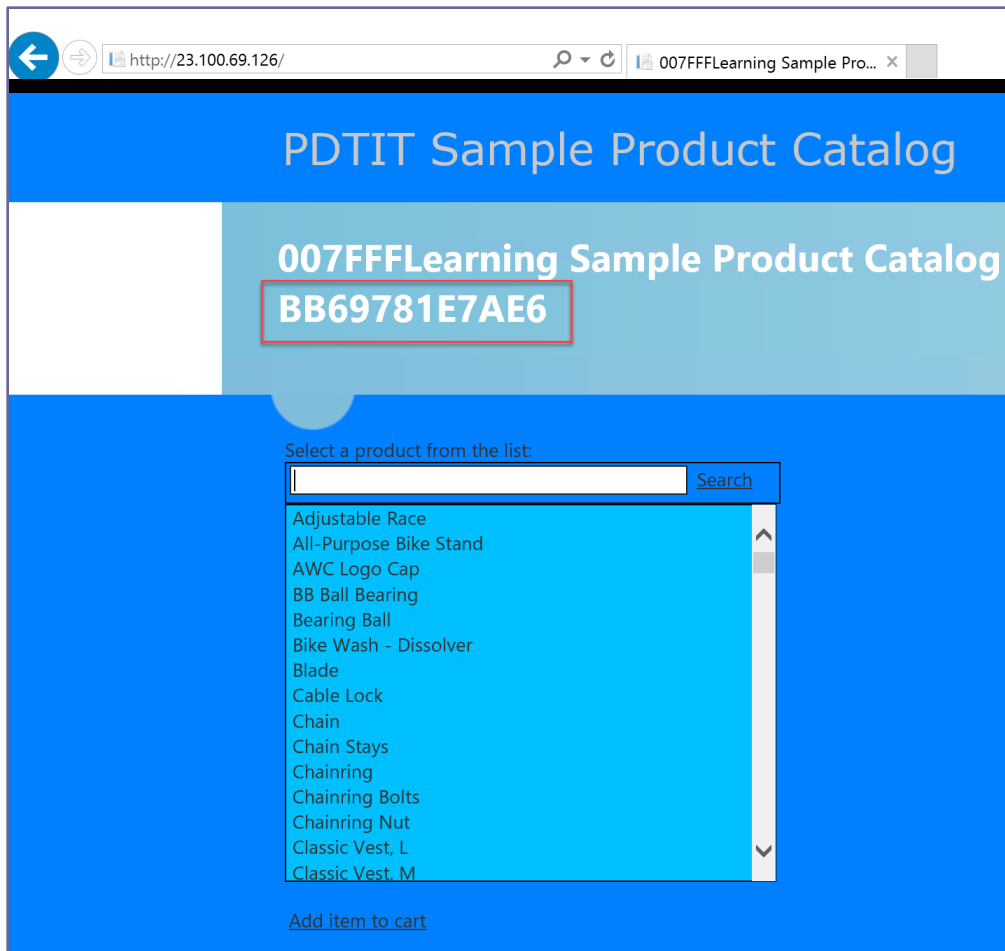
OS type  
Windows

IP address  
**23.100.69.126**

FQDN  
---

Container count  
1

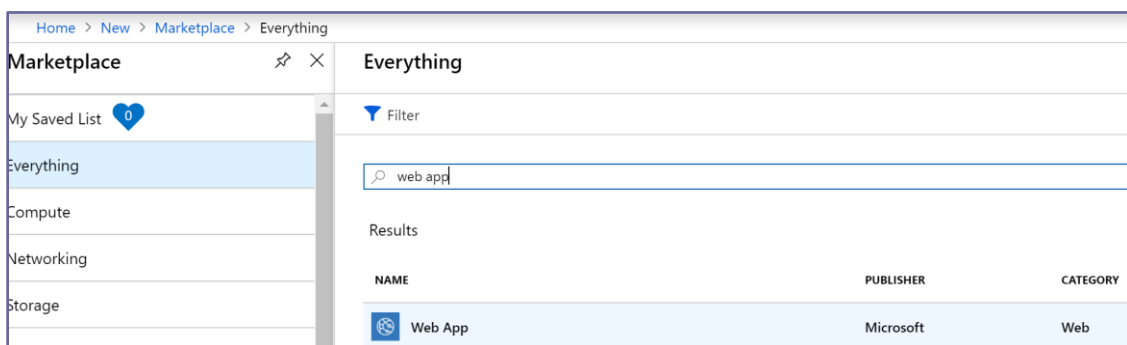
- Copy the IP address** for this Azure Container Instance, or directly browse to it from your internet browser



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

**New App Service Plan** □ ×

Create a plan for the web app

\* App Service plan  
adsdockerwebappserviceplan ✓

\* Location  
East US ▼

\* Pricing tier  
PC2 Premium Container >

And confirm the plan with **OK**.

- 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

Home > New > Marketplace > Everything > Web App > Web App >

**Single Container (Preview)** **Docker Compose (Preview)**

Image source

Quickstart **Azure Container Registry** Docker Hub Private Registry

Registry  
ADSACR ▼

Image  
webvmsamplesitedocker ▼

Tag  
latest ▼

Startup File

- Confirm the creation** by pressing the **Apply** button. Your container web app settings are now like this:

Web App

Create

\*

App name

adsdockerwebapp

✓

.azurewebsites.net

\*

Subscription

Microsoft Azure Sponsorship

▼

\*

Resource Group

Create new

Use existing

adsdockerwebappRG

✓

\*

OS

Windows

Linux

\*

Publish

Code

Docker Image

\*

App Service plan/Location

adsdockerwebappserviceplan(Ea...

>

\*

Configure container

adsacr.azurecr.io/webvmsamples...

>

- Press the **Create** button to start the deployment of the Azure Web App for Containers.
- Follow-up** on the deployment from the notification area.

Home > Microsoft.WebSite07c5c36e-a411 - Overview

Microsoft.WebSite07c5c36e-a411 - Overview

Deployment

Search (Ctrl+ /)

Overview

Outputs

Inputs

Template

✓

Your deployment is complete

Check the status of your deployment, manage resources, or troubleshoot deployment issues. Pin this page to your dashboard to easily find it next time.

Deployment name: Microsoft.WebSite07c5c36e-a411

Subscription: [Microsoft Azure Sponsorship](#)

Resource group: [adsdockerwebappRG](#)

DEPLOYMENT DETAILS [\(Download\)](#)

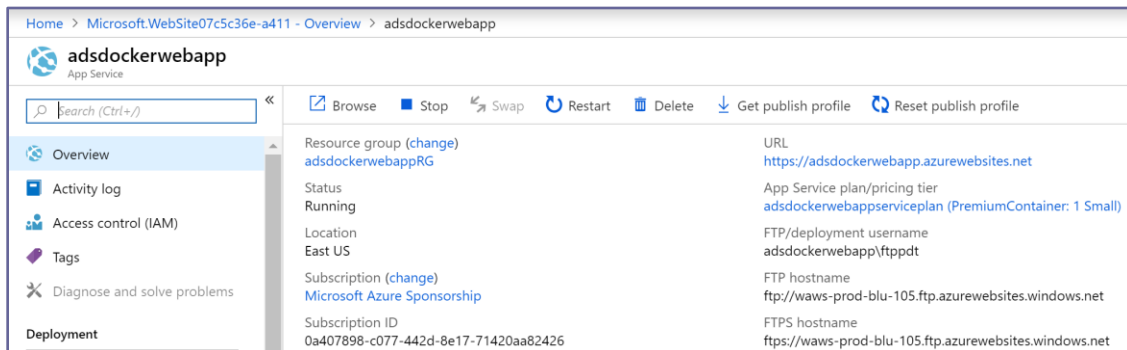
Start time: 9/30/2018, 10:06:13 PM

Duration: 32 seconds

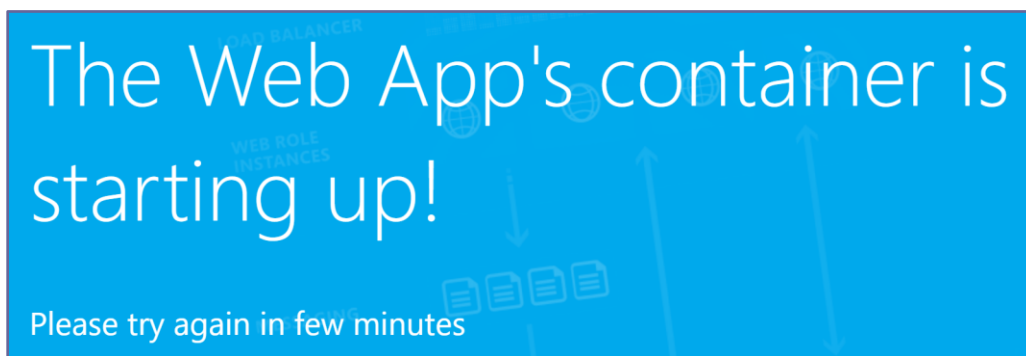
Correlation ID: f6a25c07-f1ad-47ba-9bcc-bcbcebc88db4

RESOURCE	TYPE	STATUS	OPERATION DETAILS
✓ <a href="#">adsdockerwebapp</a>	Microsoft.Web/sites	OK	<a href="#">Operation details</a>
✓ <a href="#">adsdockerwebappserv</a>	Microsoft.Web/serv...	OK	<a href="#">Operation details</a>

- Once deployed, browse to the [SUFFIX]dockerwebapp Azure Resource, which opens the detailed blade:



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



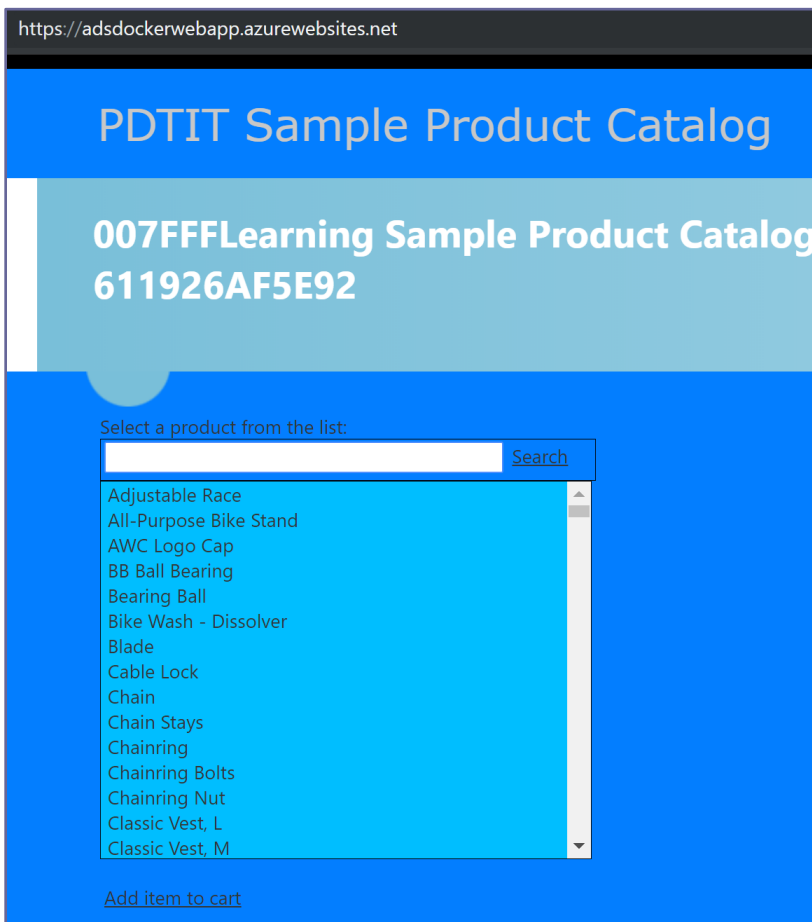
- Go back to the Azure Portal, which still has your Azure Web App for Containers open; here, browse to Settings | Container Settings and look at the LOGs section. This shows the different steps undergoing to get the container running.



- 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 containerized 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.