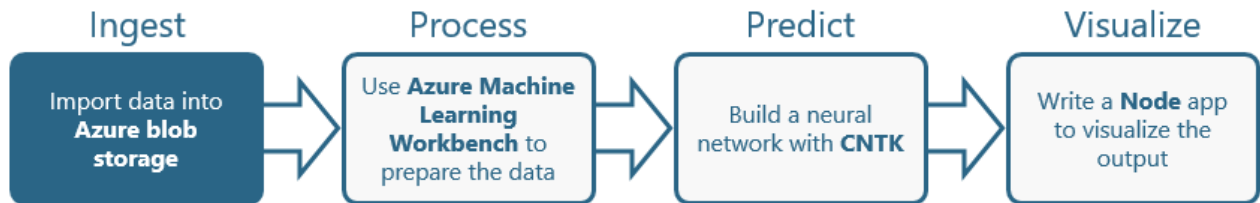


# Creating a Windows Virtual Machine and Building a Neural Network using Cognitive Toolkit CNKT



## INGEST

### Objectives

In this hands-on lab, you will learn how to:

- Create an Azure storage account
- Upload files to Azure blob storage

### Prerequisites

The following are required to complete this hands-on lab:

- An active Microsoft Azure subscription.
- A computer running Windows 10, Windows Server 2016, macOS Sierra, or macOS High Sierra
- [Azure Machine Learning Workbench](#)
- [Docker](#)

### Exercises

This hands-on lab includes the following exercises:

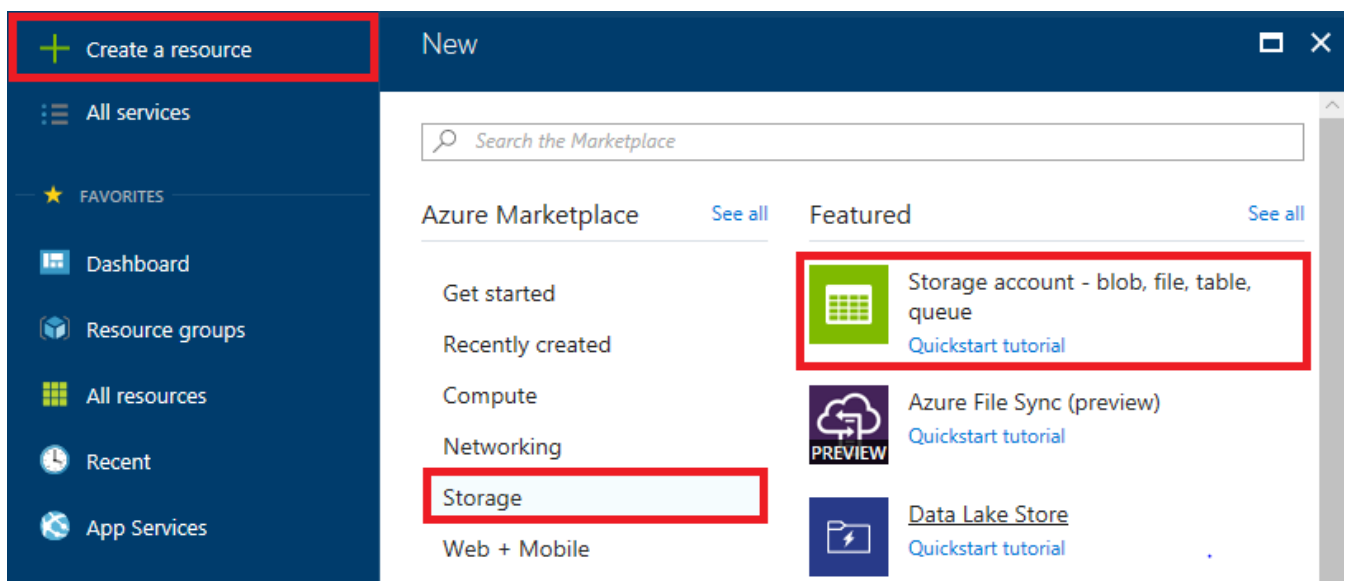
- [Exercise 1: Create a storage account](#)
- [Exercise 2: Upload data to blob storage](#)

Estimated time to complete this lab: **15** minutes.

## Exercise 1: Create a storage account

The [Azure Portal](#) allows you to perform basic storage operations such as creating storage accounts, viewing what's stored under those accounts, and managing the access keys associated with the accounts. In this exercise, you'll use the portal to create a storage account.

1. Open the [Azure Portal](#) in your browser. If you are asked to sign in, do so with your Microsoft account.
2. To create a storage account, click **+ Create a resource** in the ribbon on the left. Then click **Storage**, followed by **Storage account**.



### *Adding a storage account*

3. In the ensuing blade, enter a name for the new storage account in the **Name** field. The name is important, because it forms one part of the URL through which blobs created under this account can be accessed.

Storage account names can be 3 to 24 characters in length and can only contain numbers and lowercase letters. In addition, the name you enter must be unique within Azure; if someone else has chosen the same name, you'll be notified that the name isn't available with a red exclamation mark in the **Name** field.

Once you have a unique name that Azure will accept, select **Create new** under **Resource group** and type "CntkResources" (without quotation marks) into the box below to name the resource group that will be created for the storage account. Select the location nearest you in the **Location** box. Then click the **Create** button at the bottom of the blade.

Create storage account

The cost of your storage account depends on the usage and the options you choose below.  
[Learn more](#)

\* Name ⓘ

cntklab

✓

.core.windows.net

Deployment model ⓘ

Resource manager

Classic

Account kind ⓘ

General purpose

▼

Performance ⓘ

Standard

Premium

Replication ⓘ

Read-access geo-redundant storage (RA-...

▼

\* Secure transfer required ⓘ

Disabled

Enabled

\* Subscription

Microsoft Azure

▼

\* Resource group

☒ Create new

☐ Use existing

CntkResources

✓

\* Location

South Central US

▼

Virtual networks (Preview)

Configure virtual networks ⓘ

Disabled

Enabled

☐ Pin to dashboard

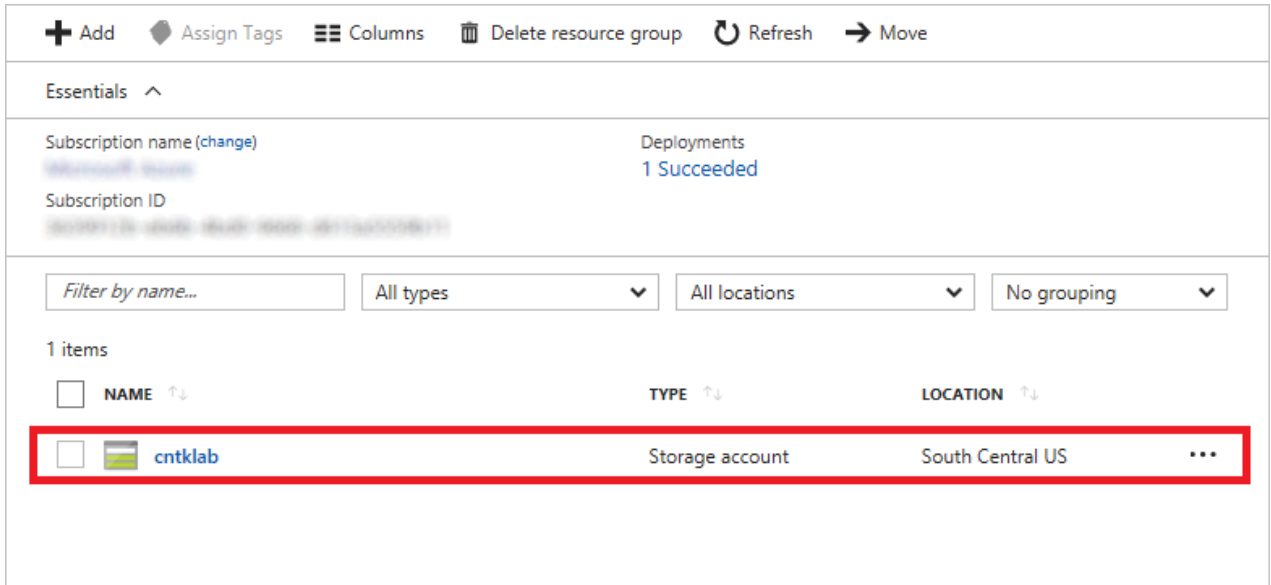
Create

Automation options

### Creating a storage account

- Click **Resource groups** in the ribbon on the left side of the portal to list all of your resource groups. In the "Resource groups" blade, click the resource group whose name you entered in the previous step.

- Wait until "Deploying" changes to "Succeeded," indicating that the storage account has been deployed. (You can click the **Refresh** button at the top of the blade to refresh the deployment status.) Then click the storage account.




Essentials ^

Subscription name (change) Microsoft Azure Deployments 1 Succeeded

Subscription ID 00000000-0000-0000-0000-000000000000

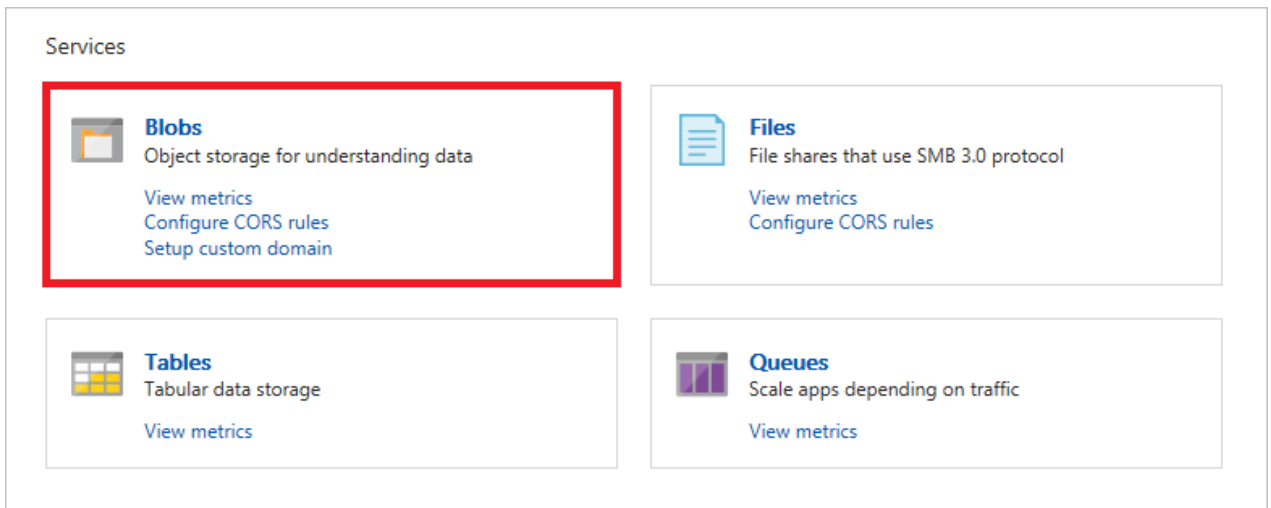
Filter by name... All types All locations No grouping

1 items


NAME ↑↓	TYPE ↑↓	LOCATION ↑↓
 cntklab	Storage account	South Central US


### Opening the storage account


- In the blade for the storage account, click **Blobs** to view a list of blob containers.




Services

**Blobs**  
Object storage for understanding data  
[View metrics](#)  
[Configure CORS rules](#)  
[Setup custom domain](#)

**Files**  
File shares that use SMB 3.0 protocol  
[View metrics](#)  
[Configure CORS rules](#)

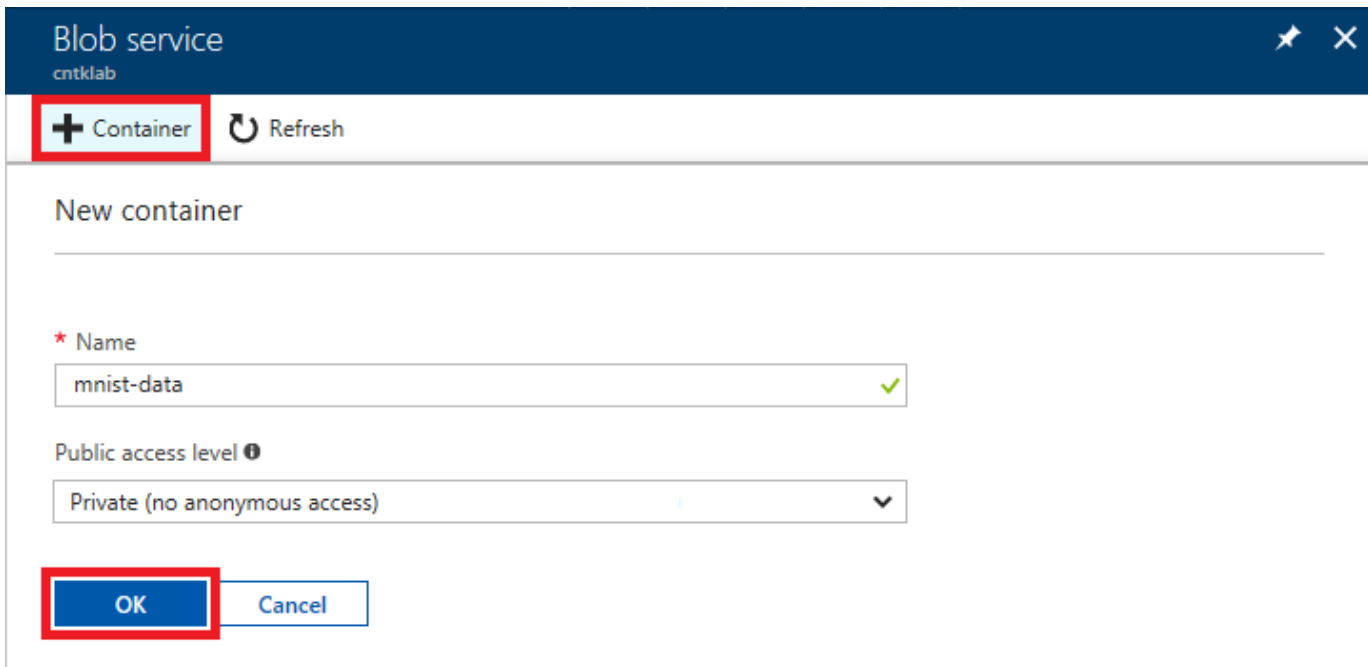
**Tables**  
Tabular data storage  
[View metrics](#)

**Queues**  
Scale apps depending on traffic  
[View metrics](#)

### Viewing blob containers

- The storage account currently has no containers. Before you create a blob, you must create a container to store it in. A container is similar to a folder in a file system. A storage account can have an unlimited number of containers, and a container can store an unlimited number of blobs. Container names must be from 3 to 63 characters in length and may contain numbers, dashes, and lowercase letters.

Click **+ Container** to create a new container. Enter "mnist-data" as the container name, and accept the default **Public access level** of **Private** so blobs stored in this container are not publicly accessible. Then click **OK**.



The screenshot shows the 'Blob service' interface for 'cntklab'. At the top, there is a '+ Container' button (highlighted with a red box) and a 'Refresh' button. Below this is the 'New container' form. It has a 'Name' field with the value 'mnist-data' and a green checkmark. Below the name field is a 'Public access level' dropdown menu, currently set to 'Private (no anonymous access)'. At the bottom of the form, there are two buttons: 'OK' (highlighted with a red box) and 'Cancel'.

*Creating a container*

You now have a storage account in which you can store blobs, and a container to store them in. The next step is to upload a dataset to the container.

## Exercise 2: Upload data to blob storage

In this exercise, you will download the MNIST dataset to your local computer, and then upload it to the container you created in the previous exercise. The dataset consists of four compressed GZ files containing images and labels for training and testing.

1. Download the files from the following URLs and save them on your hard disk:
  - <http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz>
  - <http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz>
  - <http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz>
  - <http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz>

These files contain the raw data for the MNIST dataset.

2. Return to the Azure Portal in your browser and click the container you created in [Exercise 1](#).

Blob service  
cntklab

+ Container Refresh

Essentials

Search containers by prefix

NAME	LAST MODIFIED	PUBLIC ACCESS...	LEASE STATE
mnist-data	12/5/2017 4:15:53 PM	Container	Available ...

*Opening the container*

3. Click **Upload**.

mnist-data  
Container

Upload Refresh Delete container Container properties Access policy

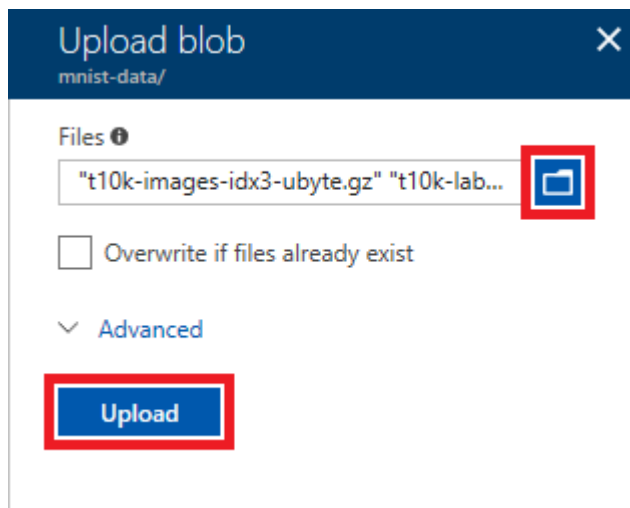
Location: mnist-data

Search blobs by prefix (case-sensitive)

NAME	MODIFIED	BLOB TYPE	SIZE	LEASE STATE
No blobs found.				

*Uploading the dataset*

4. Click the folder icon and browse to the directory in which you saved the four GZ files. Then select all four files and click the **Upload** button to upload them to the container.



*Uploading the dataset*

5. Close the "Upload Blob" blade and confirm that all four files uploaded successfully.

NAME	MODIFIED	BLOB TYPE	SIZE	LEASE STATE
t10k-images-idx3-ubyte.gz	12/5/2017 4:31:33 PM	Block blob	1.57 MiB	Available ...
t10k-labels-idx1-ubyte.gz	12/5/2017 4:31:28 PM	Block blob	4.44 KiB	Available ...
train-images-idx3-ubyte.gz	12/5/2017 4:31:44 PM	Block blob	9.45 MiB	Available ...
train-labels-idx1-ubyte.gz	12/5/2017 4:31:28 PM	Block blob	28.2 KiB	Available ...

*The uploaded MNIST data*

The MNIST data is now in blob storage, where it can be accessed by applications that are granted access to it using the storage account's access key or connection string. In the next lab, you will retrieve a connection string and use [Azure Machine Learning Workbench](#) to prepare the data for use in machine learning.

## PROCESS

## Objectives

In this hands-on lab, you will learn how to:

- Write Python code to wrangle data stored in Azure blob storage
- Execute that code in Docker containers using Azure Machine Learning Workbench
- Format labeled image data for input to CNTK

## Exercises

---

This hands-on lab includes the following exercises:

- [Exercise 1: Install Workbench and create a project](#)
- [Exercise 2: Customize the project](#)
- [Exercise 3: Convert MNIST data to CNTK format](#)

Estimated time to complete this lab: **30** minutes.

### Exercise 1: Install Workbench and create a project

---

In this exercise, you will install Azure Machine Learning Workbench and create a new project to house the files for a machine-learning project.

1. Azure Machine Learning Workbench runs jobs in Docker containers, and as such, it requires that Docker be installed on your computer. If you haven't installed Docker, go to <https://www.docker.com/> and download and install [Docker for Windows](#) or [Docker for Mac](#). If you are not sure whether Docker is installed on your computer, open a Command Prompt window (Windows) or a terminal window (macOS) and type the following command:

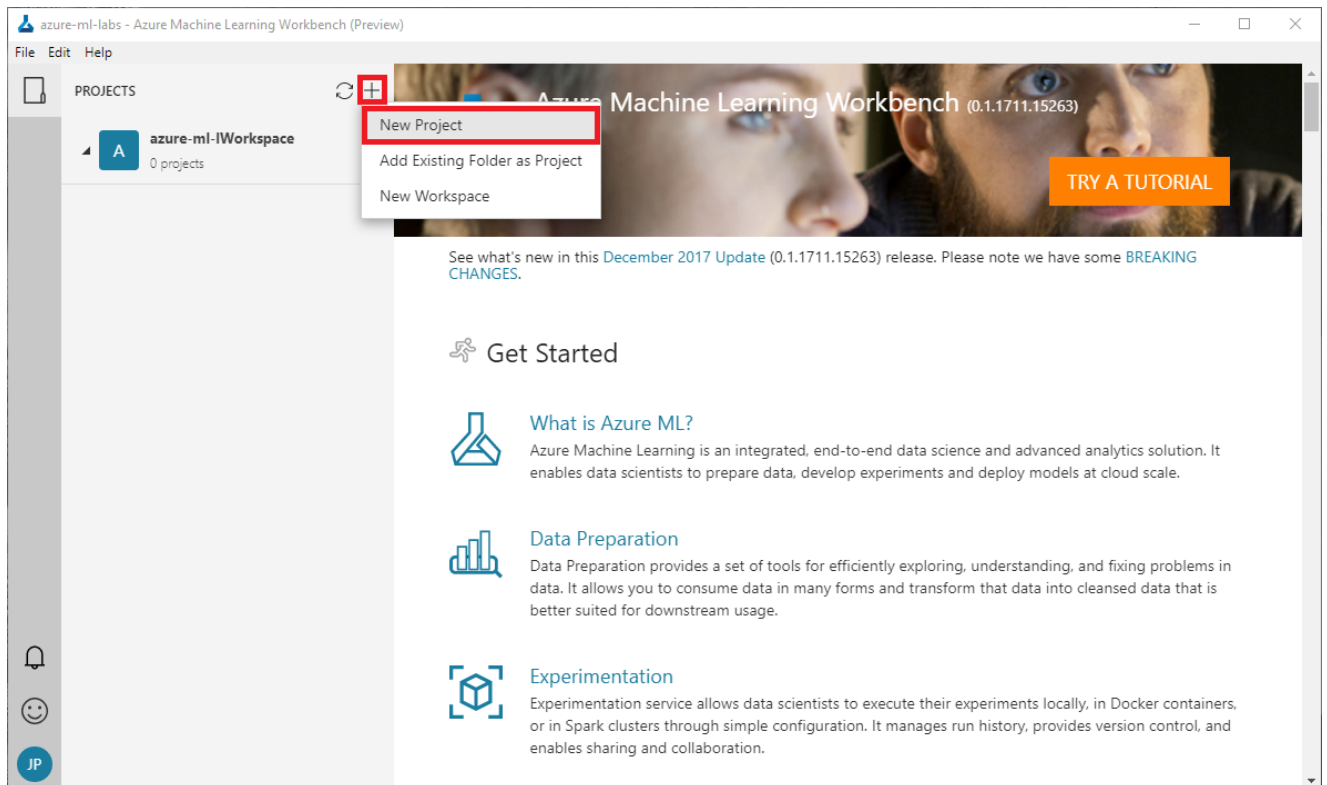
```
2. docker -v
```

If a Docker version number is displayed, then Docker is installed. If Docker is not installed, you should restart your PC after installing it.

3. If Azure Machine Learning Workbench isn't installed on your computer, go to <https://docs.microsoft.com/azure/machine-learning/preview/quickstart-installation> and follow the instructions there to install it, create a Machine Learning Experimentation account, and sign in to Machine Learning Workbench for the first time. The experimentation account is required in order to use Azure Machine Learning Workbench.

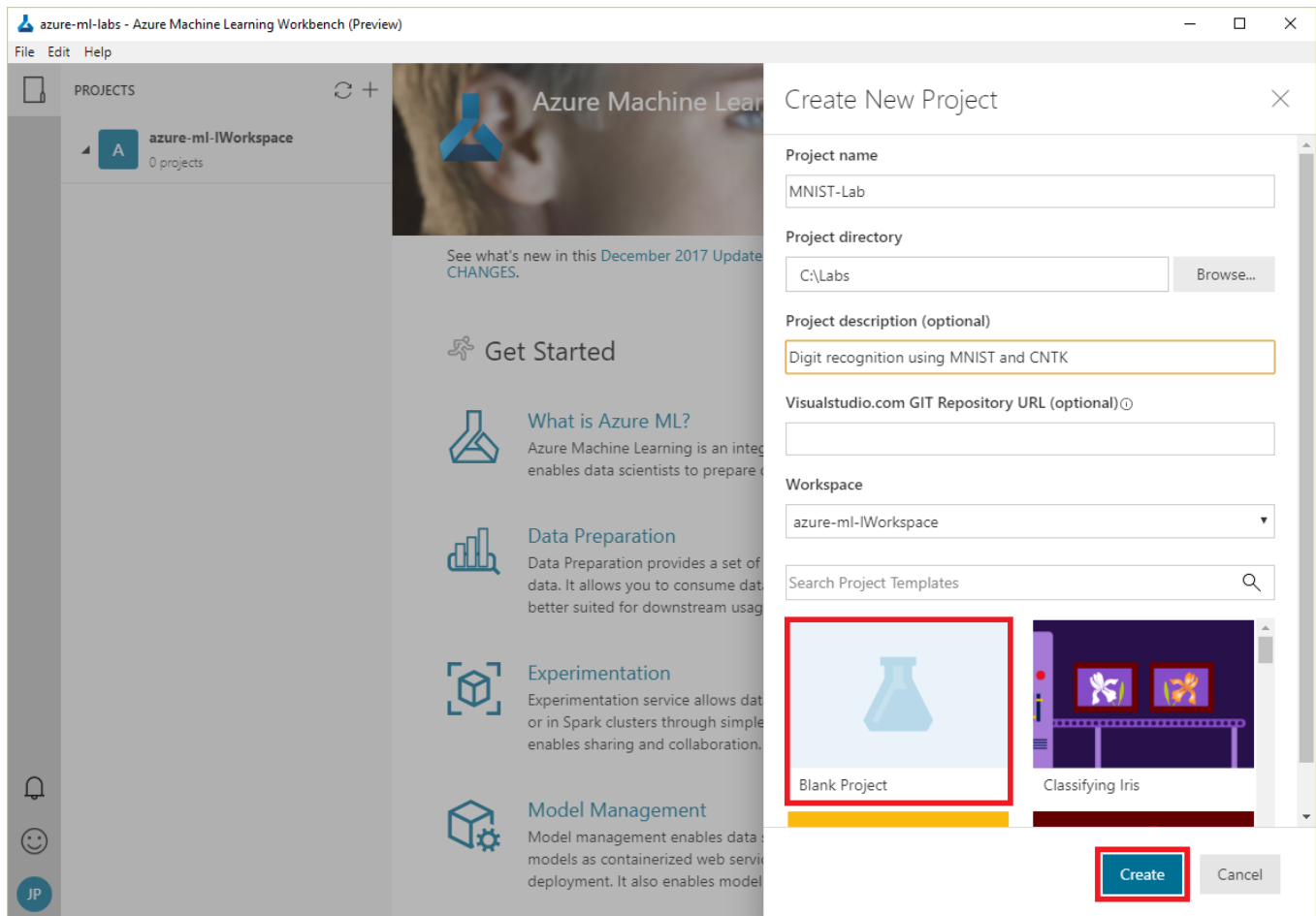


4. Launch Azure Machine Learning Workbench if it isn't already running. Then click the + sign in the "Projects" panel and select **New Project**.



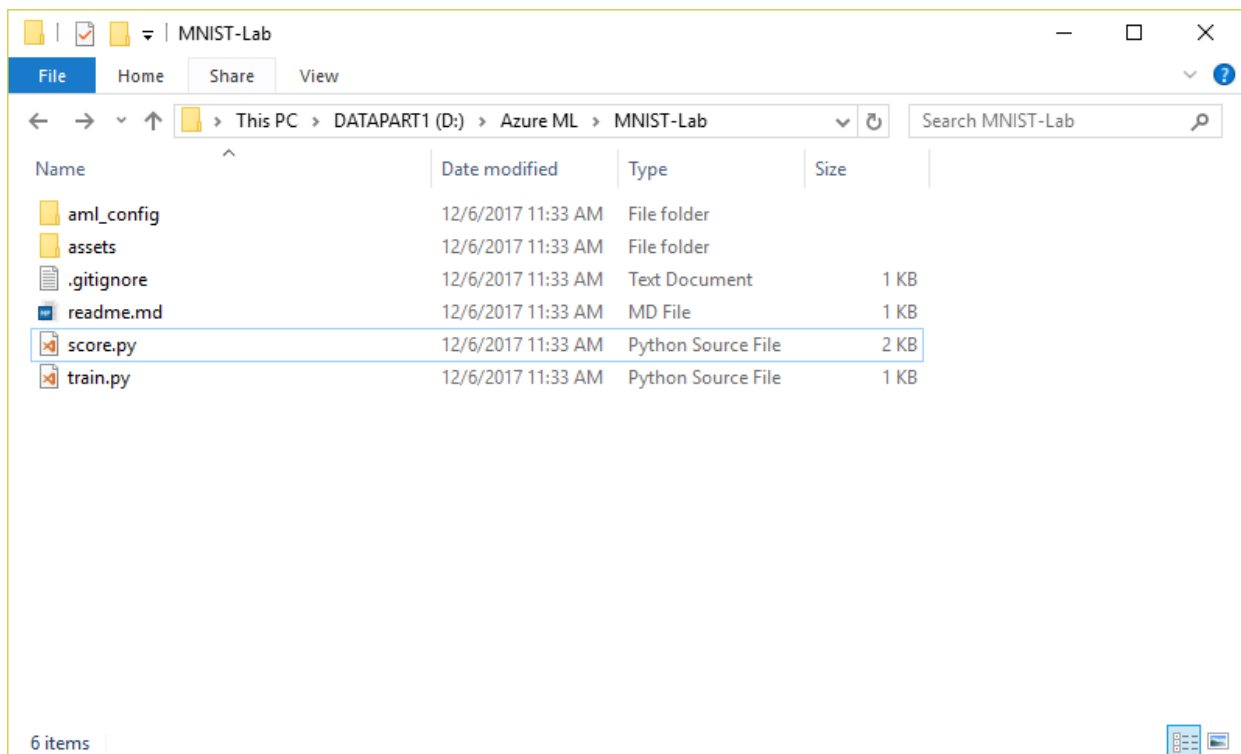
### *Creating a new project*

5. Enter a project name such as "MNIST-Lab" and a project description. For **Project directory**, specify the location where you would like for the project directory to be created. Make sure **Blank Project** is selected as the project type, and then click the **Create** button.



### *Creating a new project*

6. Open a File Explorer window (Windows) or a Finder window (macOS) and navigate to the project directory that you specified in the previous step. Confirm that it contains a subdirectory with the same name as the project. Open the subdirectory and examine its contents. Confirm that it contains a subdirectory named "aml\_config" and a pair of Python scripts named **train.py** and **score.py**, as shown below.



### *Inspecting the project directory*

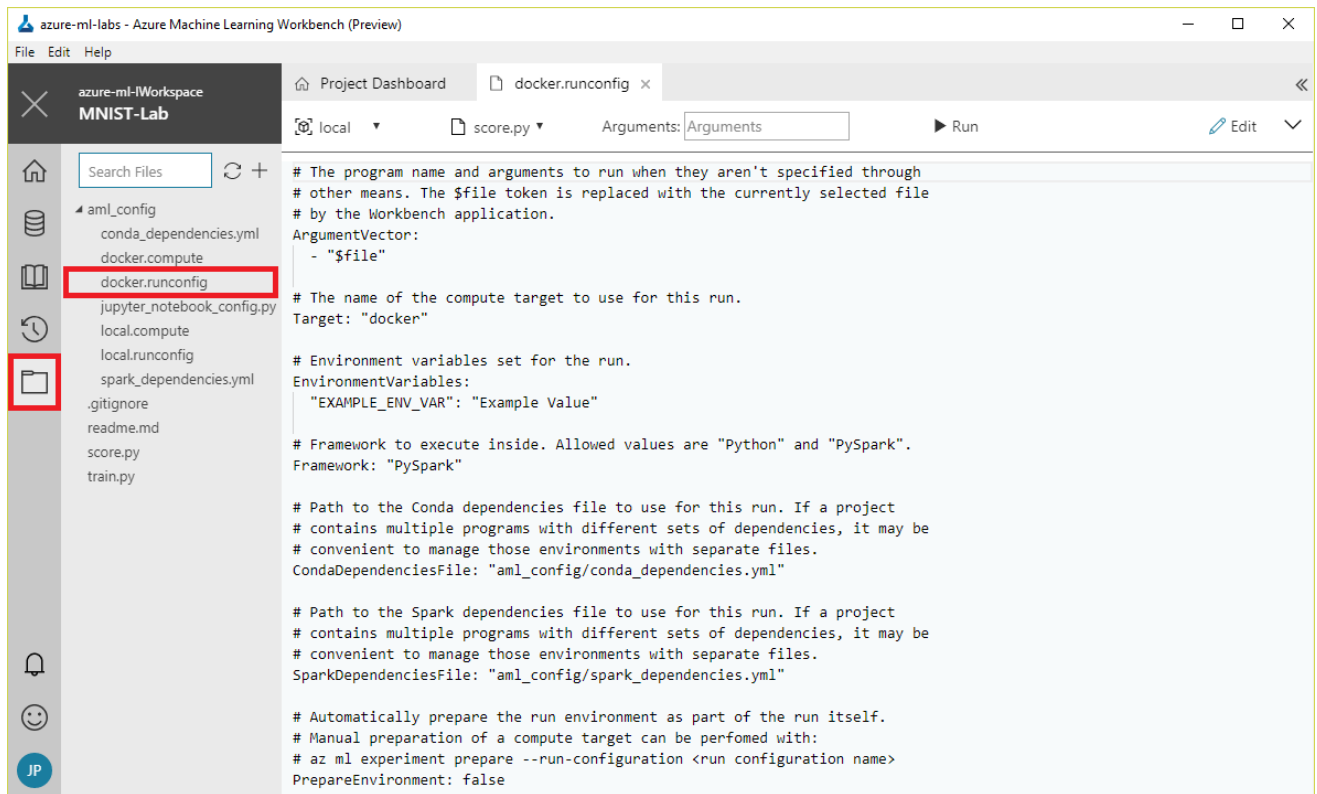
The files in the project directory are merely starter files that will need to be modified in order to build a machine-learning model. But before you can build a model, you need to prepare the data that will be used to train it.

## Exercise 2: Customize the project

One of the tasks at which Machine Learning Workbench excels is helping you prepare data for training machine-learning models. For example, its [Derive Column by Example](#) feature lets you create new feature columns with information derived from data in other columns, and it uses AI to learn by example. You do the first few transformations, and it does the rest. It also includes features for replacing missing values, trimming strings, and performing other common data-cleaning operations.

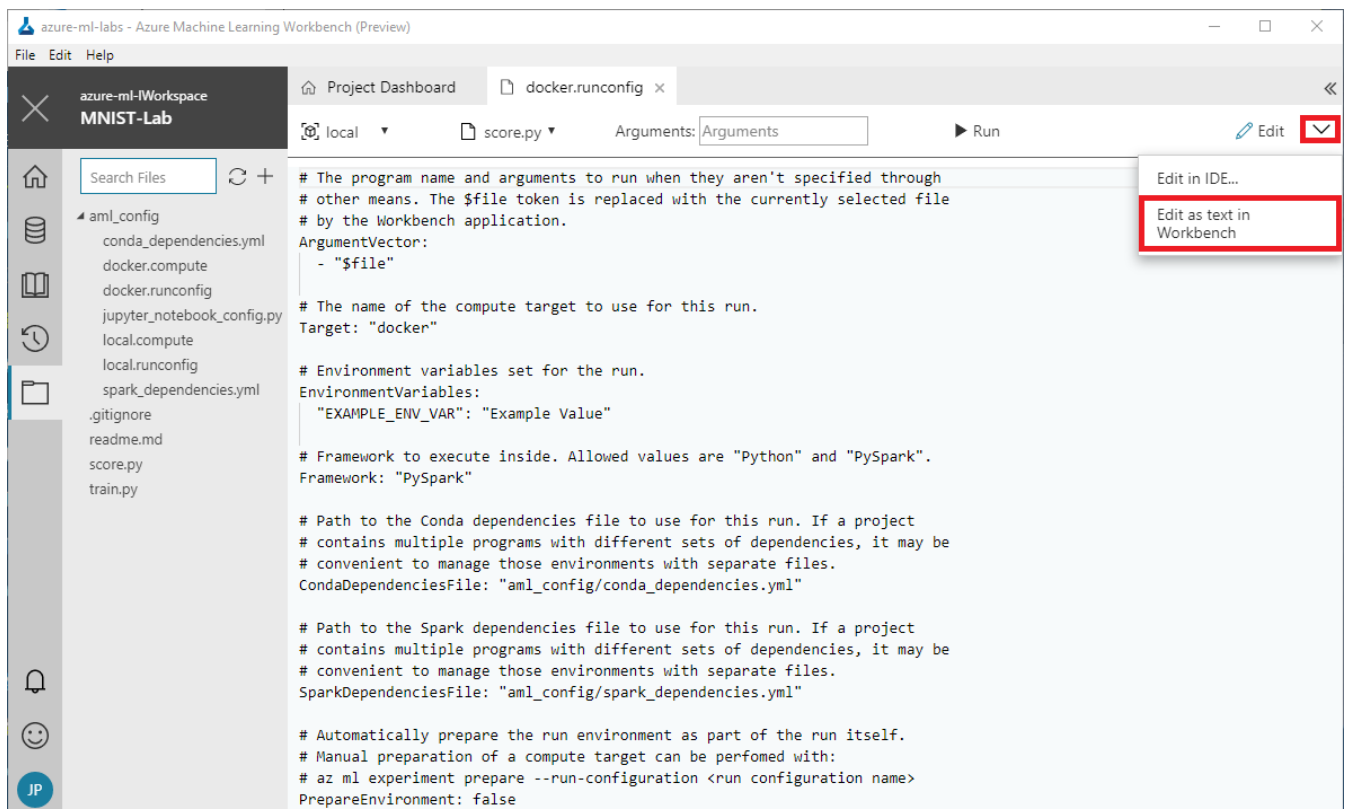
The MNIST data that you uploaded to blob storage in the previous lab doesn't require cleaning in the conventional sense, but it does need to be converted into a format that is compatible with CNTK. In this exercise, you will write Python scripts to perform the conversion, and configure the project to execute them.

1. Return to Machine Learning Workbench and click the folder icon in the ribbon on the left to display all the files in the project. Then expand the treeview to show the files in the "aml\_config" directory and click **docker.runconfig**. This file contains configuration information used when the project is executed in a Docker container.



### Opening docker.runconfig

2. Click the down-arrow next to **Edit** and select **Edit as text in Workbench** to enter editing mode.

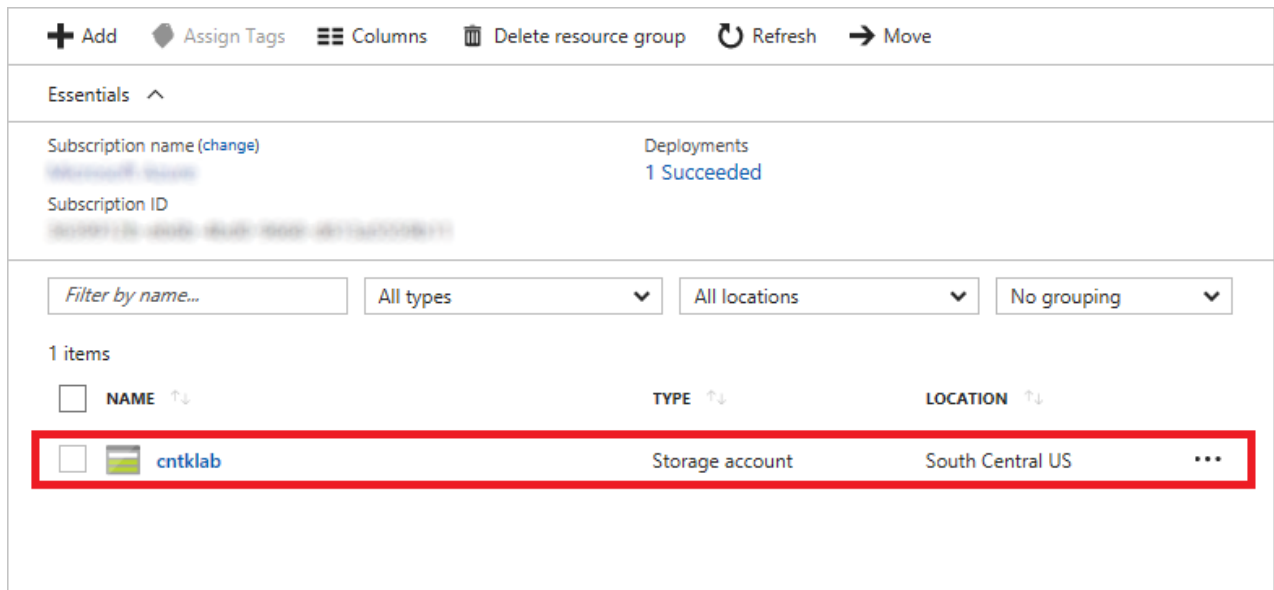


### Editing docker.runconfig

3. Replace lines 10-15 of **docker.runconfig** with the following statements:

```
4. # Environment variables set for the run.
5. EnvironmentVariables:
6.   "AZ_STORAGE_CONNSTR": "CONNECTION_STRING"
7.
8. # Framework to execute inside. Allowed values are "Python" and "PySpark".
9. Framework: "Python"
```

10. Open the [Azure Portal](#) in your browser and open the storage account that you created in the [previous lab](#).



The screenshot shows the Azure Portal interface. At the top, there are navigation buttons: Add, Assign Tags, Columns, Delete resource group, Refresh, and Move. Below this is the 'Essentials' section, which displays the subscription name (Microsoft Azure), subscription ID, and deployment status (1 Succeeded). A filter bar allows filtering by name, type, location, and grouping. Below the filter bar, a table lists resources. The table has columns for NAME, TYPE, and LOCATION. One item is listed: 'cntklab' (Storage account) located in 'South Central US'. This row is highlighted with a red border.

NAME	TYPE	LOCATION
cntklab	Storage account	South Central US

### Opening the storage account

11. Click **Access keys** in the menu on the left. Then click the **Copy** button to the right of the first connection string to copy the storage account's connection string to the clipboard. This connection string allows scripts and programs to access blobs stored in the storage account, even if the blobs are private.

cntklab - Access keys  
Storage account

Search (Ctrl+)

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

SETTINGS

- Access keys**
- Configuration
- Shared access signature
- Firewalls and virtual networks...

Use access keys to authenticate your applications when making requests to this Azure storage account. Store your access keys securely - for example, using Azure Key Vault - and don't share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you can maintain connections using one key while regenerating the other.

When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines. [Learn more](#)

Storage account name: cntklab

Default keys

NAME	KEY	CONNECTION STRING
key1	TeaGCaowgCxR4s4GHZa15hFaEtbzF...	DefaultEndpointsProtocol=https;Acc...
key2	VMRaEr6atETylLB1d0Kgx8k+u7KTe...	DefaultEndpointsProtocol=https;Acc...

### *Copying the connection string*

- Return to Machine Learning Workbench and replace CONNECTION\_STRING on line 12 of **docker.runconfig** with the connection string on the clipboard. This assigns the connection string to an environment variable so it can be retrieved at run-time by scripts executed by Machine Learning Workbench.
  - On line 30 of **docker.runconfig**, change the value of PrepareEnvironment from false to true:
- ```
14. PrepareEnvironment: true
```
- This configures the project to automatically prepare the environment by loading dependencies when the project is run.
- Select **Save** from the **File** menu to save the modified **docker.runconfig** file.
  - Open **docker.compute** for editing in Machine Learning Workbench. Then change the value of baseDockerImage on line 8 to "blaize/mlspark-brainscript," as shown here:
- ```
17. baseDockerImage: "blaize/mlspark-brainscript"
```

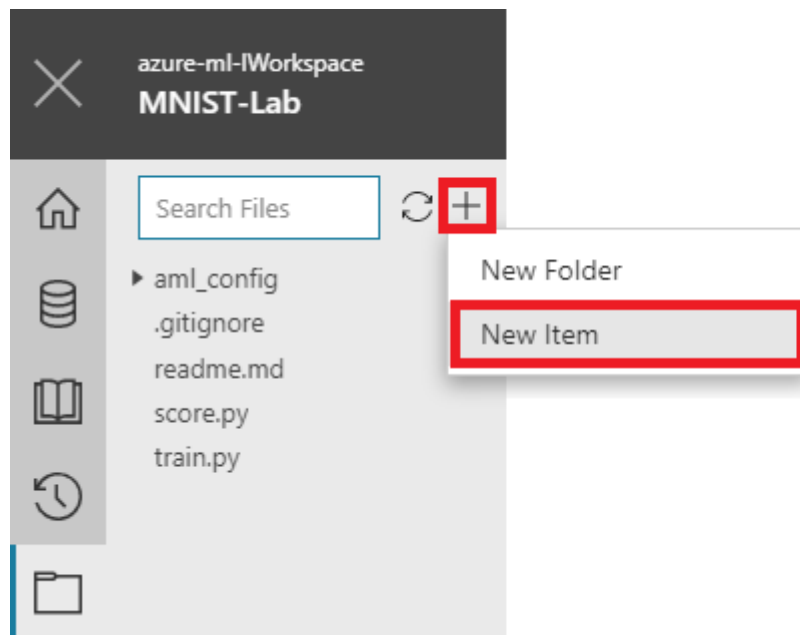
This changes the base Docker image used for the Docker container to a custom image that adds support for BrainScript to the Docker image that Machine Learning Workbench uses by default. [BrainScript](#) is the language used by CNTK to define neural networks.

You won't be using BrainScript in this lab, but you *will* use it in the next lab when you use CNTK to build a neural network.

For reference, and in case you want to create a similar Docker image of your own, here is the Dockerfile used to create the `mmlspark-brainscript` image:

```
FROM microsoft/mmlspark:plus-0.7.91
WORKDIR /home/mmlspark/
USER root
ENV PATH=/home/mmlspark/cntk/cntk/bin:$PATH
ENV LD_LIBRARY_PATH=/home/mmlspark/cntk/cntk/lib:/home/mmlspark/cntk/cntk/dependencies/lib:$LD_LIBRARY_PATH
RUN curl https://cntk.ai/BinaryDrop/CNTK-2-3-Linux-64bit-CPU-Only.tar.gz -o /home/mmlspark/CNTK.tar.gz && \
    tar -xzf /home/mmlspark/CNTK.tar.gz && \
    chown -R mmlspark:mmlspark cntk
USER mmlspark
```

18. Use the **File > Save** command to save the modified **docker.compute** file.
19. Click the **+** sign in the project panel and use the **New Item** command to add a file named **mnist\_utils.py** to the project.



*Adding a file to the project*

20. Open **mnist\_utils.py** for editing in Machine Learning Workbench and paste in the following Python code:

```
21. from __future__ import print_function
22. try:
23.     from urllib.request import urlretrieve
24. except ImportError:
25.     from urllib import urlretrieve
26. import sys
27. import gzip
28. import shutil
29. import os
```

```

30. import struct
31. import numpy as np
32.
33. def loadData(src, cimg):
34.     print ('Downloading ' + src)
35.     gzfname, h = urlretrieve(src, './delete.me')
36.     print ('Done.')
37.     try:
38.         with gzip.open(gzfname) as gz:
39.             n = struct.unpack('I', gz.read(4))
40.             # Read magic number.
41.             if n[0] != 0x3080000:
42.                 raise Exception('Invalid file: unexpected magic number.')
43.             # Read number of entries.
44.             n = struct.unpack('>I', gz.read(4))[0]
45.             if n != cimg:
46.                 raise Exception('Invalid file: expected {0}
entries.'.format(cimg))
47.             crow = struct.unpack('>I', gz.read(4))[0]
48.             ccol = struct.unpack('>I', gz.read(4))[0]
49.             if crow != 28 or ccol != 28:
50.                 raise Exception('Invalid file: expected 28 rows/cols per
image.')
51.             # Read data.
52.             res = np.fromstring(gz.read(cimg * crow * ccol), dtype =
np.uint8)
53.         finally:
54.             os.remove(gzfname)
55.         return res.reshape((cimg, crow * ccol))
56.
57. def loadLabels(src, cimg):
58.     print ('Downloading ' + src)
59.     gzfname, h = urlretrieve(src, './delete.me')
60.     print ('Done.')
61.     try:
62.         with gzip.open(gzfname) as gz:
63.             n = struct.unpack('I', gz.read(4))
64.             # Read magic number.
65.             if n[0] != 0x1080000:
66.                 raise Exception('Invalid file: unexpected magic number.')
67.             # Read number of entries.
68.             n = struct.unpack('>I', gz.read(4))
69.             if n[0] != cimg:
70.                 raise Exception('Invalid file: expected {0}
rows.'.format(cimg))
71.             # Read labels.
72.             res = np.fromstring(gz.read(cimg), dtype = np.uint8)
73.         finally:
74.             os.remove(gzfname)
75.         return res.reshape((cimg, 1))
76.
77. def load(dataSrc, labelsSrc, cimg):
78.     data = loadData(dataSrc, cimg)
79.     labels = loadLabels(labelsSrc, cimg)
80.     return np.hstack((data, labels))
81.
82. def savetxt(filename, ndarray):
83.     with open(filename, 'w') as f:
84.         labels = list(map(' '.join, np.eye(10, dtype=np.uint).astype(str)))

```



```

85.         for row in ndarray:
86.             row_str = row.astype(str)
87.             label_str = labels[row[-1]]
88.             feature_str = ' '.join(row_str[:-1])
            f.write('|labels {} |features {} \n'.format(label_str,
feature_str))

```

This file contains helper functions for loading data from blob storage and writing structured text files to the local file system.

89. Use the **File > Save** command to save the modified **mnist\_utils.py** file.

90. Click the + sign in the project panel and use the **New Item** command to add a file named **convert.py** to the project. Then open **convert.py** for editing and paste in the following code:

```

91. from __future__ import print_function
92. import os
93. import mnist_utils as ut
94. from azure.storage.blob import (
95.     BlockBlobService,
96.     ContainerPermissions,
97.     BlobPermissions,
98.     PublicAccess,
99. )
100. from datetime import datetime, timedelta
101.
102. block_blob_service =
    BlockBlobService(connection_string=os.environ['AZ_STORAGE_CONNSTR'])
103. block_blob_service.socket_timeout = 600
104.
105. # Function for getting a SAS URL for blobs
106. def getUrl(blobName):
107.     container = 'mnist-data'
108.     token = block_blob_service.generate_blob_shared_access_signature(
109.         container,
110.         blobName,
111.         BlobPermissions.READ,
112.         datetime.utcnow() + timedelta(hours=1),
113.     )
114.     return block_blob_service.make_blob_url(container, blobName,
sas_token=token)
115.
116. if __name__ == "__main__":
117.
118.     # Convert the data to the CNTK Format
119.     os.chdir(os.path.abspath(os.path.dirname(__file__)))
120.     train = ut.load(getUrl('train-images-idx3-ubyte.gz'),
121.         getUrl('train-labels-idx1-ubyte.gz'), 60000)
122.     print('Writing train text file...')
123.     ut.savetxt(r'./Train-28x28_cntk_text.txt', train)
124.     test = ut.load(getUrl('t10k-images-idx3-ubyte.gz'),
125.         getUrl('t10k-labels-idx1-ubyte.gz'), 10000)
126.     print('Writing test text file...')
127.     ut.savetxt(r'./Test-28x28_cntk_text.txt', test)
128.

```

```

129.     cntkContainer = 'cntk-data'
130.
131.     # Upload the CNTK-formatted data to blob storage
132.     print ('Uploading results to Azure Storage.')
133.     block_blob_service.create_container(cntkContainer, metadata=None,
        public_access=None, fail_on_exist=False, timeout=None)
134.     block_blob_service.create_blob_from_path(cntkContainer, 'Train-
        28x28_cntk_text.txt', r'./Train-28x28_cntk_text.txt')
        block_blob_service.create_blob_from_path(cntkContainer, 'Test-
        28x28_cntk_text.txt', r'./Test-28x28_cntk_text.txt')

```

This file contains code that loads MNIST training and testing data from blob storage, converts it to CNTK format, saves the converted data in text files in the local file system, and then uploads the text files from the local file system to a new container in blob storage.

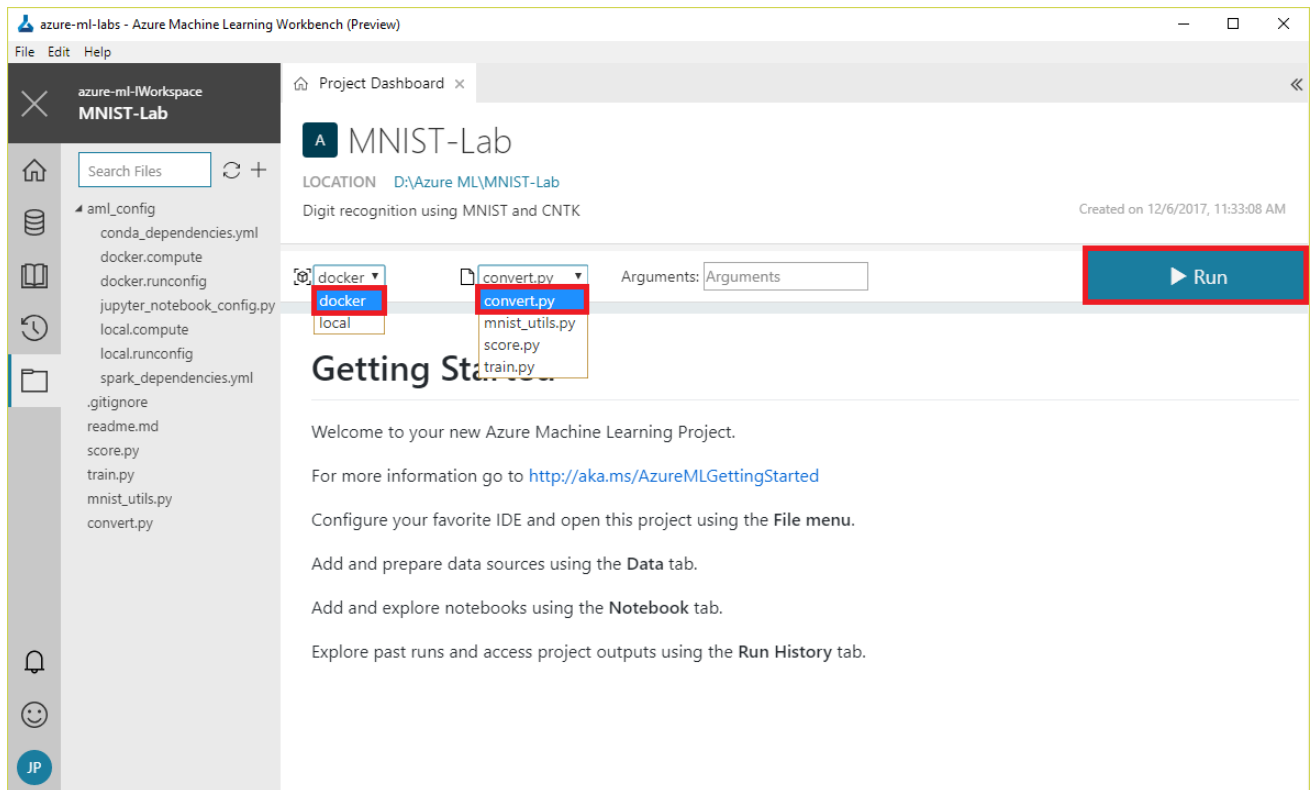
135. Use the **File > Save** command to save the modified **convert.py** file.

The project now contains a Python script that converts the MNIST data in blob storage into CNTK data in blob storage, as well as a Python script containing helper functions used by that script. It also contains a Docker configuration file that holds the connection string that the scripts use to access the storage account, and a Docker compute file that specifies the base Docker image to use for the container in which the scripts run. Now it's time to put these scripts to work.

## Exercise 3: Convert MNIST data to CNTK format

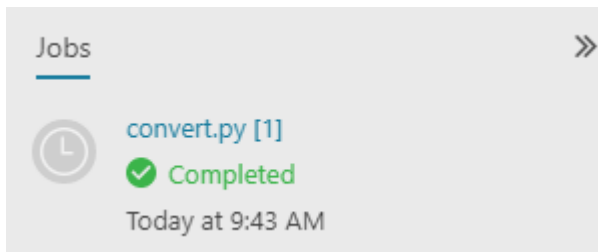
In this exercise, you will run the **convert.py** script that you created in the previous exercise in a local Docker container using Machine Learning Workbench to convert MNIST data from its native format to CNTK format. Then you will examine the output in blob storage and see what CNTK-formatted data looks like.

1. In Machine Learning Workbench, select **Docker** from the Run Configuration drop-down and **convert.py** from the Script drop-down to configure Workbench to run **convert.py** in a Docker container. Then click the **Run** button.



### *Running convert.py*

2. Wait for the job to complete. The first run may take a few minutes because Machine Learning Workbench has to download the base Docker image from Docker Hub. Subsequent runs should be much faster.



### *Successful run*

3. Return to the storage account that you created in the previous lab in the [Azure Portal](#). Confirm that the storage account now contains *two* containers: the "mnist-data" container you created in the previous lab, and a container named "cntk-data" that holds the output from **convert.py**.

Blob service				
cntklab				
+ Container Refresh				
Essentials				
Search containers by prefix				
NAME	LAST MODIFIED	PUBLIC ACCESS...	LEASE STATE	
cntk-data	12/7/2017 8:46:43 AM	Private	Available	...
mnist-data	12/7/2017 8:47:47 AM	Private	Available	...

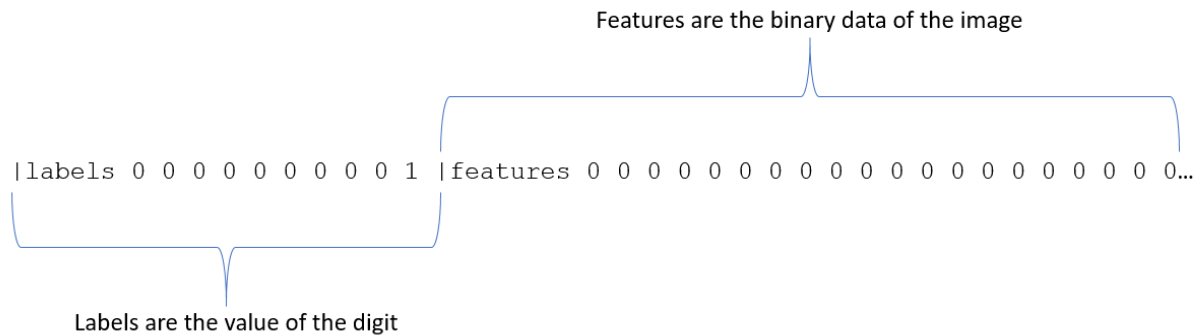
### Containers for converted and unconverted data

4. Open the "cntk-data" container and confirm that it contains two blobs. One contains CNTK-formatted data for training a model, and the other contains CNTK-formatted data for testing the model.

cntk-data					
Container					
Upload Refresh Delete container Container properties Access policy					
Location: cntk-data					
Search blobs by prefix (case-sensitive)					
NAME	MODIFIED	BLOB TYPE	SIZE	LEASE STATE	
Test-28x28_cntk_text.txt	12/7/2017 8:47:12 AM	Block blob	17.79 MiB	Available	...
Train-28x28_cntk_text.txt	12/7/2017 8:47:06 AM	Block blob	106.56 MiB	Available	...

### Blobs containing CNTK input

If you are curious to know what data formatted for input to CNTK looks like, download the blob named **Test-28x28\_cntk\_text.txt** and open it in your favorite text editor. The blob contains one row for each digitized image. Each row contains a section named "labels" and a section named "features." The "labels" data denotes the digit represented by the image. The first value is 1 or 0 for 0, the second is 1 or 0 for 1, and so on. For the digit 7, all of the values in "labels" will be 0 except the 8th, which will contain a 1. The "features" data represents the image itself. There are 784 values, one for each pixel in a 28 x 28 array, and each value is a grayscale value from 0 to 255, with 0 representing empty space and 255 representing filled space.



*CNTK-formatted data*

## PREDICT

### Objectives

In this hands-on lab, you will learn how to:

- Write Python code to train CNTK models
- Execute that code in Docker containers using Azure Machine Learning Workbench
- Determine the accuracy of a trained CNTK model

### Resources

[Click here](#) to download a zip file containing the resources used in this lab. Copy the contents of the zip file into a folder on your hard disk.

### Exercises

This hands-on lab includes the following exercises:

- [Exercise 1: Train a neural network and score it for accuracy](#)
- [Exercise 2: Train and score a second neural network](#)
- [Exercise 3: Train and score a third neural network](#)

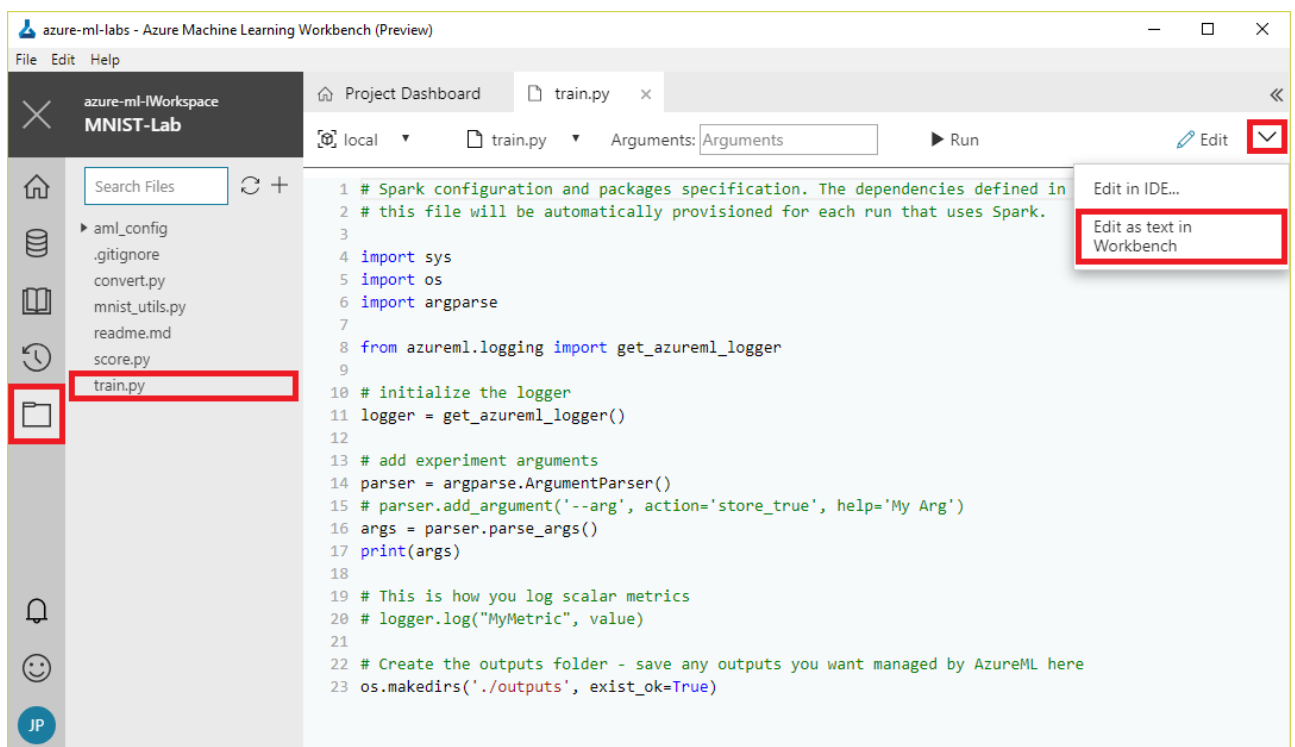
Estimated time to complete this lab: **30** minutes.

### Exercise 1: Train a neural network and score it for accuracy

It's time to leverage the work you performed in the first two labs by using the CNTK-formatted data that you generated to train and test a machine-learning model that utilizes a CNTK neural network. The Microsoft Cognitive Toolkit comes with several [BrainScript](#) files enabling various types of neural networks to be trained and tested with a single command. BrainScript provides a simple way to define a network in code-like fashion using expressions, variables, functions, and other constructs. CNTK networks can also be defined in pure Python, but BrainScript files are shorter, more concise, and generally more readable.

One of the simplest network types is the "One Hidden Layer" network, which is not a deep neural network, but rather one that contains a single layer of hidden nodes. In this exercise, you will write a Python script that uses BrainScript to train and score a "One Hidden Layer" network that recognizes hand-written digits, and execute the script in a Docker container from Azure Machine Learning Workbench.

1. Return to Machine Learning Workbench and to the project that you created in the previous lab. Then open the file named **train.py** for editing.



*Opening train.py for editing in Machine Learning Workbench*

2. Replace the contents of **train.py** with the Python code below. Then save the file.

```
3. from __future__ import print_function
4. import os
5. from shutil import copyfile
6. import mnist_utils as ut
```

```

7. from azure.storage.blob import (
8.     BlockBlobService,
9.     ContainerPermissions,
10.    BlobPermissions,
11.    PublicAccess,
12.)
13. from datetime import datetime, timedelta
14. from subprocess import Popen, PIPE
15. import subprocess
16.
17. print(os.environ['AZ_STORAGE_CONNSTR'])
18. block_blob_service =
19.     BlockBlobService(connection_string=os.environ['AZ_STORAGE_CONNSTR'])
20. block_blob_service.socket_timeout = 600
21.
22. def trainModel(modelFile):
23.     modelDir = 'models'
24.     modelName = os.path.splitext(modelFile)[0].lower()
25.     if not os.path.exists(os.path.join('.', modelDir)):
26.         os.makedirs(os.path.join('.', modelDir))
27.
28.     print('Training model ' + modelFile);
29.
30.     p = Popen(['cntk', 'configFile=./' + modelFile],
31.               stdout=subprocess.PIPE, stderr=subprocess.PIPE)
32.     std_out, std_err = p.communicate()
33.     print(std_err.decode('utf-8'))
34.     print('Uploading model ' + modelFile);
35.
36.     modelContainer = 'models' #os.path.splitext(modelFile)[0].lower()
37.     block_blob_service.create_container(modelContainer, metadata=None,
38.                                       public_access=None, fail_on_exist=False, timeout=None)
39.
40.     for filename in os.listdir(os.path.join('.', modelDir)):
41.         # copyfile('./Output/Models/' + filename, './outputs/Models/' +
42.         #         modelName + '/' + filename)
43.         block_blob_service.create_blob_from_path(modelContainer, filename,
44.         os.path.join('.', modelDir, filename))
45.         os.remove(os.path.join('.', modelDir, filename))
46.
47.     print ('Done.')
48.
49. if __name__ == "__main__":
50.     container = 'cntk-data'
51.     if not os.path.exists(os.path.join('.', 'mnist')):
52.         os.makedirs(os.path.join('.', 'mnist'))
53.     print('Downloading CNTK data...');
54.     block_blob_service.get_blob_to_path(container, 'Train-
55.     28x28_cntk_text.txt', r'./mnist/Train-28x28_cntk_text.txt');
56.     block_blob_service.get_blob_to_path(container, 'Test-
57.     28x28_cntk_text.txt', r'./mnist/Test-28x28_cntk_text.txt');
58.     print('Done.');
```

```

59.     trainModel('01-OneHidden.cntk')

```

This Python script downloads the CNTK-formatted training and testing data that you created in the previous lab from blob storage and invokes the `cntk` command from the Cognitive Toolkit (remember, you are executing code in a Docker container created from a Docker image that contains the Cognitive Toolkit) to train and score a neural network. It passes as input to the `cntk` command the path to a file named **01-OneHidden.cntk**. The output from the command is a set of binary files that comprise a "compiled" neural network. The script uploads these files as blobs to the storage account that you created in the first lab.

55. Add a file named **01-OneHidden.cntk** to the project and insert the following statements. Then save the file.

```
56. command = trainNetwork:testNetwork
57. precision = "float"; traceLevel = 1 ; deviceId = "auto"
58. rootDir = "." ; dataDir = "./mnist" ;
59. outputDir = "./models" ;
60.
61. modelPath = "$outputDir$/01_OneHidden"
62. #stderr = "$outputDir$/01_OneHidden_bs_out"
63.
64. # TRAINING CONFIG
65. trainNetwork = {
66.     action = "train"
67.
68.     BrainScriptNetworkBuilder = {
69.         imageShape = 28:28:1                                # image dimensions, 1
        channel only
70.         labelDim = 10                                       # number of distinct
        labels
71.         featScale = 1/256
72.
73.         # This model returns multiple nodes as a record, which
74.         # can be accessed using .x syntax.
75.         model(x) = {
76.             s1 = x * featScale
77.             h1 = DenseLayer {200, activation=ReLU} (s1)
78.             z = LinearLayer {labelDim} (h1)
79.         }
80.
81.         # inputs
82.         features = Input {imageShape}
83.         labels = Input {labelDim}
84.
85.         # apply model to features
86.         out = model (features)
87.
88.         # loss and error computation
89.         ce = CrossEntropyWithSoftmax (labels, out.z)
90.         errs = ClassificationError (labels, out.z)
91.
92.         # declare special nodes
93.         featureNodes = (features)
94.         labelNodes = (labels)
95.         criterionNodes = (ce)
```



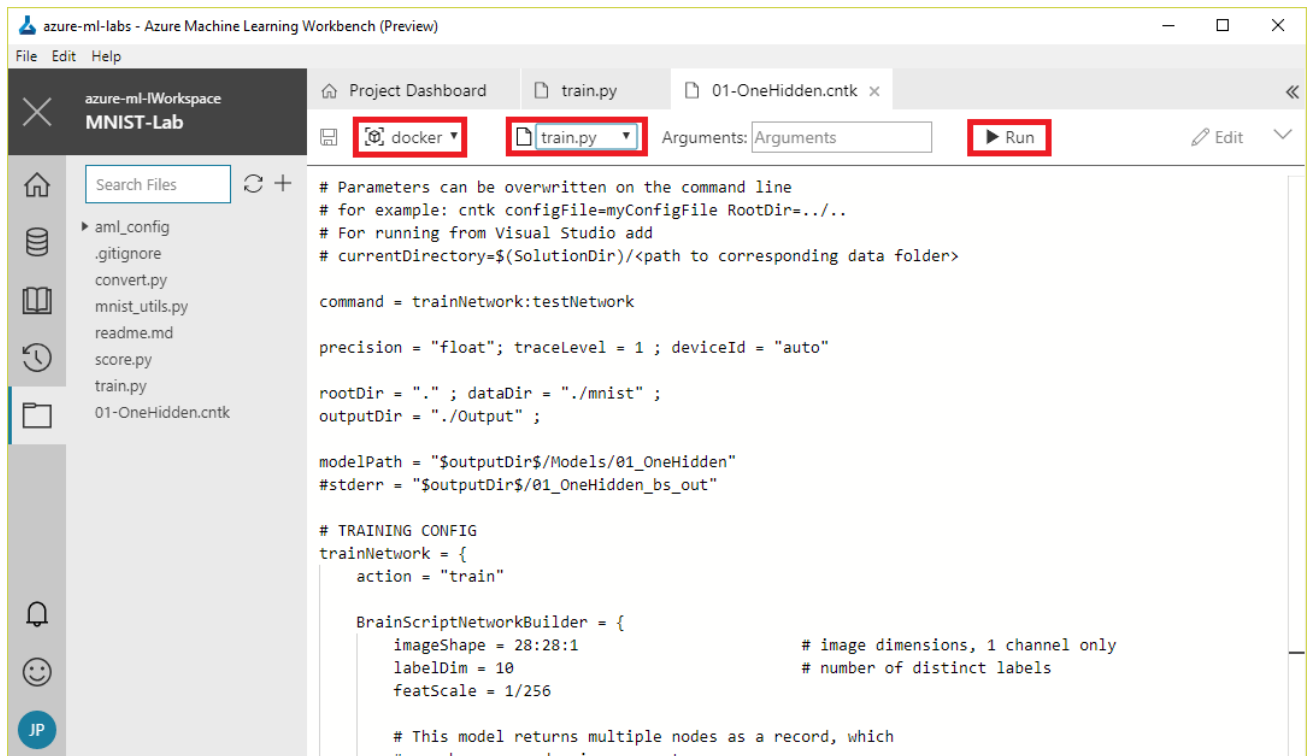
```

96.         evaluationNodes = (errs)
97.         outputNodes     = (out.z)
98.     }
99.
100.     SGD = {
101.         epochSize = 60000
102.         minibatchSize = 64
103.         maxEpochs = 10
104.         learningRatesPerSample = 0.01*5:0.005
105.         momentumAsTimeConstant = 0
106.         numMBsToShowResult = 500
107.     }
108.
109.     reader = {
110.         readerType = "CNTKTextFormatReader"
111.         file = "$DataDir$/Train-28x28_cntk_text.txt"
112.         input = {
113.             features = { dim = 784 ; format = "dense" }
114.             labels = { dim = 10 ; format = "dense" }
115.         }
116.     }
117. }
118.
119. # TEST CONFIG
120. testNetwork = {
121.     action = "test"
122.     minibatchSize = 1024    # reduce this if you run out of memory
123.
124.     reader = {
125.         readerType = "CNTKTextFormatReader"
126.         file = "$DataDir$/Test-28x28_cntk_text.txt"
127.         input = {
128.             features = { dim = 784 ; format = "dense" }
129.             labels = { dim = 10 ; format = "dense" }
130.         }
131.     }
132. }

```

This file contains [BrainScript](#) code that defines a neural network. It is a slightly modified version of a file of the same name that comes with CNTK. The first section — TRAINING CONFIG — defines a network with one hidden layer and identifies the shape and source of the training data. The TEST CONFIG section identifies the file containing testing data. The files containing the training and testing data are local versions of the files that you created in the previous lab and downloaded from blob storage.

133. Select **Docker** from the Run Configuration drop-down and **train.py** from the Script drop-down to configure Workbench to run **train.py** in a Docker container. Then click **Run**.



### *Training a neural network*

134. Wait for the run to complete and confirm that it completed successfully. It may take a few minutes to complete, and the duration will depend somewhat on your connection speed since the CNTK-formatted data has to be downloaded from blob storage.

Now click the completed run in the "Jobs" panel. Scroll to the bottom of the output window that appears and find the value named `errs`. This is the percentage of errors that were encountered when the model was tested. Lower percentages mean that the neural network was able to correctly identify more digits in the test dataset. It appears that this network is reasonably adept at identifying hand-written digits.

```

{out.z.W : [10 x 200]}
{out.z.b : [10]}
{errs : [1]}
{labels : [10 x *1]}

Minibatch[1-10]: errs = 1.790% * 10000; ce = 0.06163781 * 1
0000
Final Results: Minibatch[1-10]: errs = 1.790% 10000; ce =
0.06163781 * 10000; perplexity = 1.06357706

Action "test" complete.

COMPLETED.

Uploading model 01-OneHidden.cntk
Done.
```

*Gauging the accuracy of the neural network*

135. Close the output window and return in the Azure Portal to the storage account that you created in the first lab. Confirm that it now has a container named "models." Then click the container to view its contents.

Blob service				
cntklab				
+ Container    ↻ Refresh				
Essentials				
🔍 Search containers by prefix				
NAME	LAST MODIFIED	PUBLIC ACCESS...	LEASE STATE	
cntk-data	12/7/2017 8:46:43 AM	Private	Available	...
mnist-data	12/7/2017 8:47:47 AM	Private	Available	...
models	12/7/2017 8:57:49 PM	Private	Available	...

*Opening the "models" container*

136. Confirm that the "models" container holds the blobs pictured below. These files represent the trained network and can be used to deploy an operationalized model that performs predictive analytics.

models Container						
Upload          Refresh          Delete container          Container properties          Access policy						
Location: models						
<input type="text" value="Search blobs by prefix (case-sensitive)"/>						
NAME	MODIFIED	BLOB TYPE	SIZE	LEASE STATE		
01_OneHidden	12/7/2017 8:57:50 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.0	12/7/2017 8:57:54 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.1	12/7/2017 8:57:56 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.2	12/7/2017 8:57:58 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.3	12/7/2017 8:58:01 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.4	12/7/2017 8:57:59 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.5	12/7/2017 8:57:57 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.6	12/7/2017 8:58:00 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.7	12/7/2017 8:57:51 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.8	12/7/2017 8:57:52 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.9	12/7/2017 8:57:53 PM	Block blob	623.92 KiB	Available	...	
01_OneHidden.ckp	12/7/2017 8:57:55 PM	Block blob	621.68 KiB	Available	...	

### *Contents of the "models" container*

Writing the files that represent the trained network to blob storage is not a CNTK requirement — the files can be stored anywhere — but it does make it easy for you to download the files so you can put them to use elsewhere. This is something you will take advantage of in the next lab.

## Exercise 2: Train and score a second neural network

CNTK supports many different types of neural networks. Some are better than others at recognizing hand-written digits. In this exercise, you will train and score a second neural network and compare the results to the one you created in the previous exercise.

1. Return to Machine Learning Workbench and add a file named **02-OneConv.cntk** to the project. Insert the following BrainScript code and save the file. This script trains a "One Convolution" network, which is a [convolutional neural network](#) with a single convolutional layer. Convolutional networks are often used to perform image recognition and classification. Once more, this script is a slightly modified version of the file with the same name that is included in CNTK.

```

2. command = trainNetwork:testNetwork
3. precision = "float"; traceLevel = 1 ; deviceId = "auto"
4. rootDir = "." ; dataDir = "./mnist" ;
5. outputDir = "./models" ;
6.
7. modelPath = "$outputDir$/02_OneConv"
8. #stderr = "$outputDir$/02_OneConv_bs_out"
9.
10. # TRAINING CONFIG
11. trainNetwork = {
12.     action = "train"
13.
14.     BrainScriptNetworkBuilder = {
15.         imageShape = 28:28:1                                # image dimensions, 1
16.         channel only
17.         labelDim = 10                                         # number of distinct
18.         labels
19.         featScale = 1/256
20.         Scale{f} = x => Constant(f) .* x
21.
22.         model = Sequential (
23.             Scale {featScale} :
24.             ConvolutionalLayer {16, (5:5), pad = true} : ReLU :
25.             MaxPoolingLayer {(2:2), stride=(2:2)} :
26.             DenseLayer {64} : ReLU :
27.             LinearLayer {labelDim}
28.         )
29.
30.         # inputs
31.         features = Input {imageShape}
32.         labels = Input (labelDim)
33.
34.         # apply model to features
35.         ol = model (features)
36.
37.         # loss and error computation
38.         ce = CrossEntropyWithSoftmax (labels, ol)
39.         errs = ClassificationError (labels, ol)
40.
41.         # declare special nodes
42.         featureNodes = (features)
43.         labelNodes = (labels)
44.         criterionNodes = (ce)
45.         evaluationNodes = (errs)
46.         outputNodes = (ol)
47.     }
48.
49.     SGD = {
50.         epochSize = 60000
51.         minibatchSize = 64
52.         maxEpochs = 15
53.         learningRatesPerSample = 0.001*5:0.0005
54.         momentumAsTimeConstant = 0
55.         numMBsToShowResult = 500
56.     }
57.
58.     reader = {
59.         readerType = "CNTKTextFormatReader"
60.         file = "$DataDir$/Train-28x28_cntk_text.txt"

```

```

59.         input = {
60.             features = { dim = 784 ; format = "dense" }
61.             labels =   { dim = 10  ; format = "dense" }
62.         }
63.     }
64. }
65.
66. # TEST CONFIG
67. testNetwork = {
68.     action = "test"
69.     minibatchSize = 1024    # reduce this if you run out of memory
70.
71.     reader = {
72.         readerType = "CNTKTextFormatReader"
73.         file = "$DataDir$/Test-28x28_cntk_text.txt"
74.         input = {
75.             features = { dim = 784 ; format = "dense" }
76.             labels =   { dim = 10  ; format = "dense" }
77.         }
78.     }
79. }

```

80. Modify the final line of **train.py** to reference the file you just added. Then save the file.

```
trainModel('02-OneConv.cntk')
```

81. Run **train.py** in a Docker container and examine the output to determine how accurately the model is able to identify hand-written digits. Is this one more or less accurate than the "One Hidden Layer" network?

Rather than settle on the better of these two networks, let's train a third one and see how it compares. One of the benefits of CNTK is that you can easily experiment with different network types and the parameters that characterize them simply by varying the BrainScript passed to the `cntk` command.

## Exercise 3: Train and score a third neural network

In this exercise, you will train and score a third neural network and compare the results to the ones you created in [Exercise 1](#) and [Exercise 2](#).

1. Return to Machine Learning Workbench and add a file named **06-OneConvRegrMultiNode.cntk** to the project. Insert the following BrainScript code and save the file. This script trains a deep convolutional neural network and could require 10 minutes or more to run.

```

2. command = trainNetwork:testNetwork
3. precision = "float"; traceLevel = 1 ; deviceId = "auto"
4. rootDir = "." ; dataDir = "./mnist" ;
5. outputDir = "./models" ;
6.

```

```

7. modelPath = "$outputDir$/06_OneConvRegrMultiNode"
8. #stderr = "$outputDir$/06_OneConvRegr_bs_out"
9.
10.parallelizationMethod=DataParallelSGD
11.
12.# TRAINING CONFIG
13.trainNetwork = {
14.    action = "train"
15.
16.    BrainScriptNetworkBuilder = {
17.        imageShape = 28:28:1                                # image dimensions, 1
18.        channel only
19.        labelDim = 10                                        # number of distinct
20.        labels
21.        featScale = 1/256
22.        Scale{f} = x => Constant(f) .* x
23.
24.        model = Sequential (
25.            Scale {featScale} :
26.            ConvolutionalLayer {16, (5:5), pad = true} : ReLU :
27.            MaxPoolingLayer {(2:2), stride=(2:2)} :
28.            DenseLayer {64} : ReLU :
29.            LinearLayer {labelDim}
30.        )
31.
32.        # inputs
33.        features = Input {imageShape}
34.        labels = Input {labelDim}
35.
36.        # apply model to features
37.        z = model (features)
38.
39.        # loss and error computation
40.        diff = labels - z
41.        sqerr = ReduceSum (diff.*diff, axis=1)
42.        rmse = Sqrt (sqerr / labelDim)
43.
44.        # declare special nodes
45.        featureNodes = (features)
46.        labelNodes = (labels)
47.        criterionNodes = (rmse)
48.        evaluationNodes = (rmse)
49.        outputNodes = (z)
50.    }
51.
52.    SGD = {
53.        epochSize = 0
54.        minibatchSize = 64
55.        maxEpochs = 15
56.        learningRatesPerSample = 0.001*5:0.0005
57.        momentumAsTimeConstant = 1024
58.        numMBsToShowResult = 500
59.        ParallelTrain = [
60.            parallelizationMethod = $parallelizationMethod$
61.            distributedMBReading = "true"
62.            parallelizationStartEpoch = 1
63.            DataParallelSGD = [
64.                gradientBits = 32
65.            ]
66.        ]
67.    }
68. }

```

```

64.         ModelAveragingSGD = [
65.             blockSizePerWorker = 64
66.         ]
67.         DataParallelASGD = [
68.             syncPeriod = 64
69.             usePipeline = false
70.         ]
71.     ]
72. }
73.
74. reader = {
75.     readerType = "CNTKTextFormatReader"
76.     file = "$DataDir$/Train-28x28_cntk_text.txt"
77.     input = {
78.         features = { dim = 784 ; format = "dense" }
79.         labels = { dim = 10 ; format = "dense" }
80.     }
81. }
82. }
83.
84. # TEST CONFIG
85. testNetwork = {
86.     action = "test"
87.     minibatchSize = 1024    # reduce this if you run out of memory
88.
89.     reader = {
90.         readerType = "CNTKTextFormatReader"
91.         file = "$DataDir$/Test-28x28_cntk_text.txt"
92.         input = {
93.             features = { dim = 784 ; format = "dense" }
94.             labels = { dim = 10 ; format = "dense" }
95.         }
96.     }
97. }

```

98. Modify the final line of **train.py** to reference the file you just added. Then save the file.

```
trainModel('06-OneConvRegrMultiNode.cntk')
```

99. Run **train.py** in a Docker container and examine the output to determine how accurately the model is able to identify hand-written digits. Rather than present an `err` value, this model will present `rmse`, which stands for **root mean square error**. A simple way to compare `rmse` to `err` is to multiply the `rmse` value by 100, so that 0.02, for example, becomes 2%. Based on that, is this network more or less accurate than the other networks you trained?

The [resources that accompany this lab](#) include the three BrainScript files that you used to train neural networks, and four more that you didn't use. Feel free to train additional networks using the extra files. For a great tutorial on training various types of machine-learning models with CNTK and additional insights into BrainScript, see <https://github.com/Microsoft/CNTK/wiki/Tutorial>.

## VISUALISE



## Objectives

In this hands-on lab, you will learn how to:

- Create a Docker image containing a neural network
- Run the image in a Docker container
- Invoke the neural network inside the container

## Exercises

---

This hands-on lab includes the following exercises:

- [Exercise 1: Build a Docker image containing a neural network](#)
- [Exercise 2: Run the image in a local container](#)
- [Exercise 3: Use the neural network to identify digits](#)

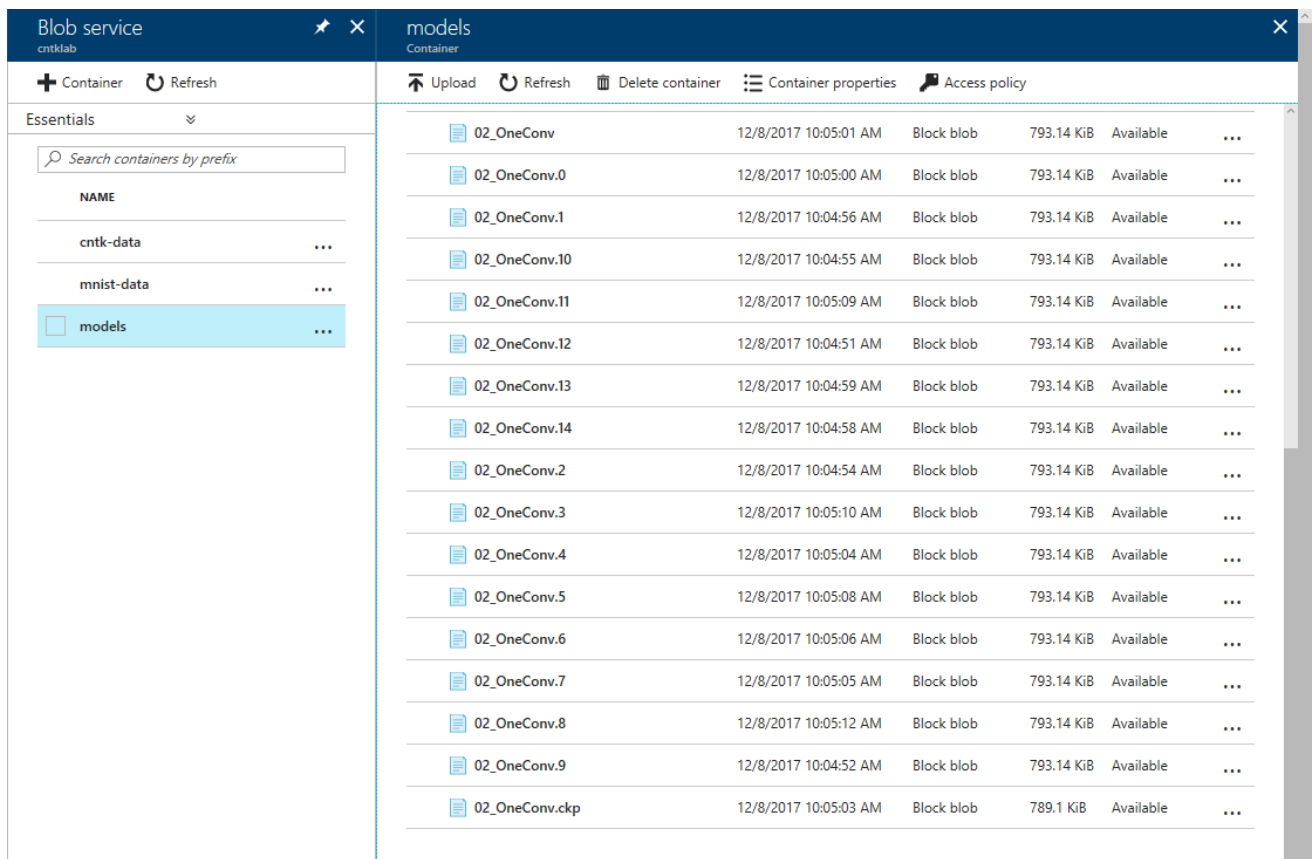
Estimated time to complete this lab: **30** minutes.

### Exercise 1: Build a Docker image containing a neural network

---

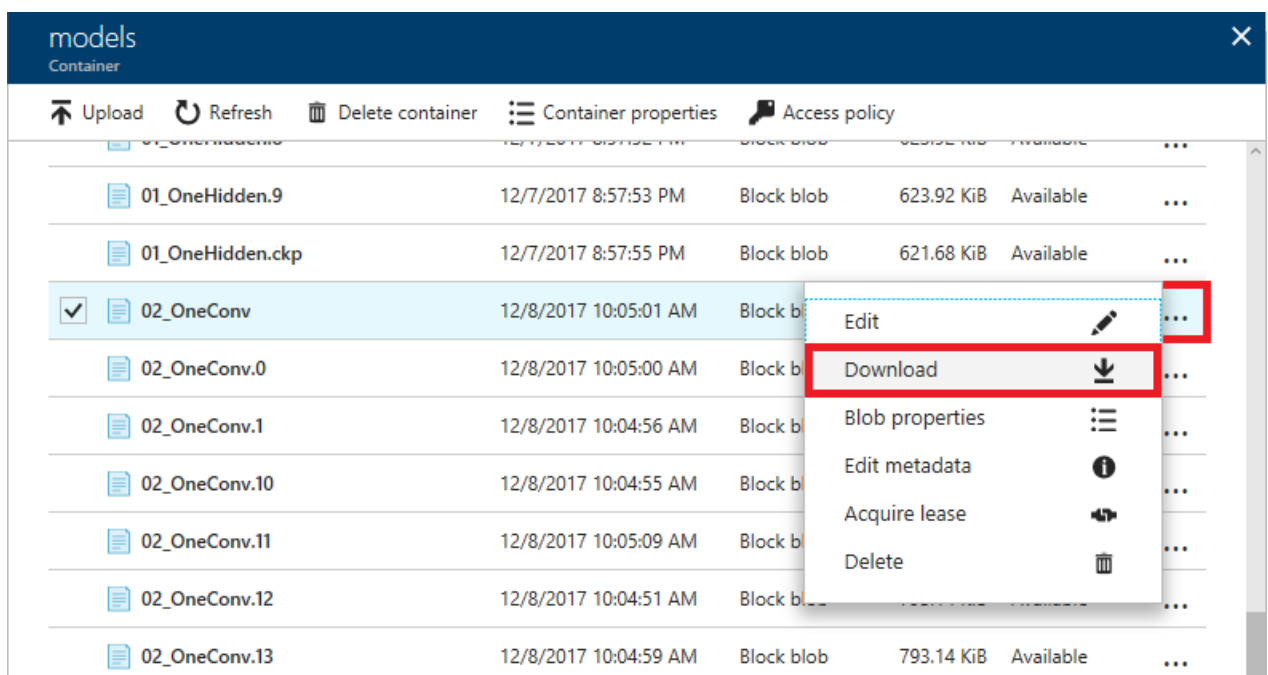
In this exercise, you will download the files created for the "One Convolution" network in the previous lab from blob storage and wrap them in a Docker container image. You will also write a Node.js app that listens for HTTP requests, serves up an HTML page for drawing digits, and implements a REST endpoint that the page can call to use the neural network to identify digits drawn by the user. The app, too, will be included in the container image.

1. Open the [Azure Portal](#) and return to the storage account that you created in the first lab. Open the container named "models" and confirm that among the many blobs there are ones named **02\_OneConv.\***.



*Blobs representing the "One Convolution" network*

- Download the blobs named **02\_OneConv.\*** one at a time and save them in a local directory on your hard disk.



*Downloading a blob*

3. When downloaded from the Azure Portal, the blob named **02\_OneConv** might have a **.txt** extension added to it. Check the downloaded file and if it's named **02\_OneConv.txt**, rename the file **02\_OneConv**.
4. Create a directory on your hard disk to hold all the files that will be built into a container image. Name the directory anything you want. Hereafter, this directory will be referred to as the "project directory."
5. Create a subdirectory named "models" in the project directory and copy all the files that you downloaded from blob storage in Step 2 into the "models" directory.
6. Create a subdirectory named "cntk" in the project directory and copy the file named **02-OneConv.cntk** from the [resources that you downloaded for the previous lab](#) into the "cntk" directory.
7. Create an empty subdirectory named "mnist" in the project directory.
8. Create a subdirectory named "public" in the project directory and add an HTML file named **index.html** containing the following statements:

```
9. <html>
10. <title>Draw an Image</title>
11. <style>
12. #canvas {
13.     border: 1px solid #000000;
14.     margin-top: 16px;
15.     margin-bottom: 16px;
16. }
17. button {
18.     width: 240px;
19.     height: 64px;
20.     margin-right: 20px;
21. }
22. </style>
23.
24. <script type="text/javascript">
25. var canvas, ctx, flag = false,
26.     prevX = 0,
27.     currX = 0,
28.     prevY = 0,
29.     currY = 0;
30.
31. function init() {
32.     canvas = document.getElementById('canvas');
33.     ctx = canvas.getContext("2d");
34.     var w = canvas.width;
35.     var h = canvas.height;
36.
37.     erase();
38.
39.     canvas.addEventListener("mousemove", function (e) {
```

```

40.     findxy('move', e);
41. }, false);
42. canvas.addEventListener("mousedown", function (e) {
43.     findxy('down', e);
44. }, false);
45. canvas.addEventListener("mouseup", function (e) {
46.     findxy('up', e);
47. }, false);
48. canvas.addEventListener("mouseout", function (e) {
49.     findxy('out', e);
50. }, false);
51. }
52.
53. function draw() {
54.     ctx.beginPath();
55.     ctx.arc(currX, currY, 12, 0, 2 * Math.PI, false);
56.     ctx.fillStyle = '#222222';
57.     ctx.fill();
58.     ctx.lineWidth = 12;
59.     ctx.strokeStyle = '#222222';
60.     ctx.stroke();
61. }
62.
63. function erase() {
64.     ctx.beginPath();
65.     ctx.rect(0, 0, 500, 500);
66.     ctx.fillStyle = "#ffffff";
67.     ctx.fill();
68. }
69.
70. function findxy(res, e) {
71.     if (res == 'down') {
72.         prevX = currX;
73.         prevY = currY;
74.         currX = e.clientX - canvas.offsetLeft;
75.         currY = e.clientY - canvas.offsetTop;
76.         flag = true;
77.     }
78.
79.     if (res == 'up' || res == "out") {
80.         flag = false;
81.     }
82.
83.     if (res == 'move') {
84.         if (flag) {
85.             prevX = currX;
86.             prevY = currY;
87.             currX = e.clientX - canvas.offsetLeft;
88.             currY = e.clientY - canvas.offsetTop;
89.             draw();
90.         }
91.     }
92. }
93.
94. var batchStr = "";
95.
96. function testImage() {
97.     var scaledCanvas = document.createElement('canvas');
98.     scaledCanvas.width = 28;

```

```

99.     scaledCanvas.height = 28;
100.     var scaledCtx = scaledCanvas.getContext('2d');
101.     scaledCtx.drawImage(canvas, 0,0,28,28);
102.     var cntkline = "|features";
103.
104.     for (var y = 0; y < 28; y++) {
105.         for (var x = 0; x < 28; x++) {
106.             var pixel = scaledCtx.getImageData(x, y, 1, 1).data;
107.             cntkline += " " + (255 - pixel[0]);
108.         }
109.     }
110.
111.     batchStr = cntkline + "\n";
112.
113.     var xhr = new XMLHttpRequest();
114.     xhr.open('POST', '/testImage', true);
115.
116.     xhr.onload = function () {
117.         console.log(this.responseText);
118.         var data = JSON.parse(this.responseText);
119.         alert("Detected a " + data.detected);
120.     };
121.
122.     xhr.send(batchStr);
123. }
124. </script>
125.
126. <body onload="init()">
127.     <div id="instructions">
128.         <span style="font-family: segoe ui">
129.             Use your pointing device (mouse, finger, or stylus) to draw a
130.             digit 0 through 9 in the box below.
131.             Then click the PREDICT button.
132.         </span>
133.     </div>
134.     <canvas id="canvas" width="500" height="500"></canvas>
135.
136.     <div id="controls">
137.         <button onclick="testImage()">PREDICT</button>
138.         <button onclick="erase()">CLEAR</button>
139.     </div>
140. </body>
</html>

```

This is the Web page that the Web server running inside the container serves up to allow users to draw digits and submit them to the model. It uses HTML5's [Canvas API](#) to do the drawing in response to mouse events. And it features a PREDICT button that, when clicked, uses AJAX to call a REST endpoint in the container and pass in the data representing the digit that was drawn.

141. Add a file named **index.js** containing the following statements to the project directory:

```

142. var express = require("express");

```

```

143. var bodyParser = require("body-parser");
144. var path = require("path");
145. var app = express();
146. var exec = require("child_process").exec;
147. var fs = require("fs");
148.
149. var port = 3003;
150.
151. app.use(express.static(path.join(__dirname, "public")));
152. app.use(bodyParser.text());
153.
154. app.post('/testImage', function (req, res) {
155.     fs.readdir("./cntk/", (err, files) => {
156.         var results = {};
157.         var testNetwork = function(file, digit) {
158.             var cntkline = "|labels ";
159.             for (var i = 0; i < 10; i++) {
160.                 var label = "0 ";
161.                 if (i == digit) {
162.                     label = "1 ";
163.                 }
164.                 cntkline += label;
165.             }
166.
167.             cntkline += req.body;
168.             fs.writeFileSync("./mnist/Test-28x28_cntk_text.txt" ,
169.                 cntkline);
170.
171.             exec("cntk configFile=./cntk/" + file + "
172.                 command=testNetwork", function(error, stdout, stderr) {
173.                 if (!error) {
174.                     var lines = stderr.split("\n");
175.
176.                     for (var j = 0; j < lines.length; j++) {
177.                         if (lines[j].startsWith("Final Results:")) {
178.                             results[digit] = lines[j];
179.                         }
180.                     }
181.
182.                     if (digit == 9) {
183.                         var detect = {detected: "Unknown!"};
184.
185.                         if (results["0"].indexOf("errs =") > 0) {
186.                             var regex = /(.*errs = )(.*%)(.*)/g;
187.                             for (var dig in results) {
188.                                 var matchDigitVal =
189.                                     parseFloat(results[dig].replace(regex, "$2"));
190.                                 if (matchDigitVal === 0) {
191.                                     detect.detected = dig;
192.                                 }
193.                             }
194.                             res.json(detect);
195.                         }
196.                     }
197.                     else {
198.                         var regex = /(.*rmse = )(.*)(.*)/g;
199.                         var val = 99;
200.                         for (var dig in results) {
201.                             var matchDigitVal =
202.                                 parseFloat(results[dig].replace(regex, "$2"));

```

```

198.             if (matchDigitVal < val) {
199.                 val = matchDigitVal;
200.                 detect.detected = dig;
201.             }
202.         }
203.         res.json(detect);
204.     }
205. }
206. else {
207.     testNework(file, digit + 1);
208. }
209. }
210. else {
211.     res.send(stderr);
212. }
213. });
214. };
215.
216. if (files.length > 0) {
217.     testNework(files[0], 0);
218. }
219. });
220. });
221.
222. app.listen(port);
    console.log("Listening on port " + port);

```

This file contains the code for the Node app. It uses the popular NPM package named [express](#) to implement a minimalist Web server. Then it listens for HTTP requests, responds to requests for static files in the "public" directory, and implements a REST method named `testImage` that invokes the CNTK model to identify the digit passed in the body of the request.

223. Add a file named **package.json** containing the following statements to the project directory:

```

224. {
225.   "name": "node-app",
226.   "version": "1.0.0",
227.   "description": "A Node app to identify hand-written digits",
228.   "main": "index.js",
229.   "scripts": {
230.     "test": "echo \"Error: no test specified\" && exit 1",
231.     "start": "node index.js"
232.   },
233.   "dependencies": {
234.     "body-parser": "^1.18.2",
235.     "express": "^4.16.2"
236.   }
237. }

```

This file includes information about the Node app that is built into the container image, including the file that runs when the app is started (**index.js**) and a list of dependencies — packages that must be downloaded and

installed in order for **index.js** to work. Package installation occurs when NPM INSTALL is executed as the container image is being built.

237. Add a file named **Dockerfile** (no file-name extension) containing the following statements to the project directory:

```
238. FROM microsoft/mmlspark:plus-0.7.91
239. USER root
240. RUN curl https://cntk.ai/BinaryDrop/CNTK-2-3-Linux-64bit-CPU-Only.tar.gz
    -o /home/mmlspark/CNTK.tar.gz && \
241.     tar -xzf /home/mmlspark/CNTK.tar.gz && \
242.     chown -R mmlspark:mmlspark cntk
243. COPY . /node-app
244. WORKDIR /node-app
245. ENV PATH=/home/mmlspark/cntk/cntk/bin:$PATH
246. ENV
    LD_LIBRARY_PATH=/home/mmlspark/cntk/cntk/lib:/home/mmlspark/cntk/cntk/depen
    dencies/lib:$LD_LIBRARY_PATH
247. RUN apt-get update && apt-get install -y curl && \
248.     curl -sL https://deb.nodesource.com/setup_8.x | bash - && \
249.     apt-get install -y nodejs && \
250.     npm install
251. EXPOSE 3003
    CMD npm start
```

This file contains instructions for building a Docker container image. It uses Microsoft's `microsoft/mmlspark:plus-0.7.91` as a base image and adds CNTK. It also installs Node.js in the container image, copies the files and subdirectories in the project directory into the container image, and opens port 3003 to HTTP traffic.

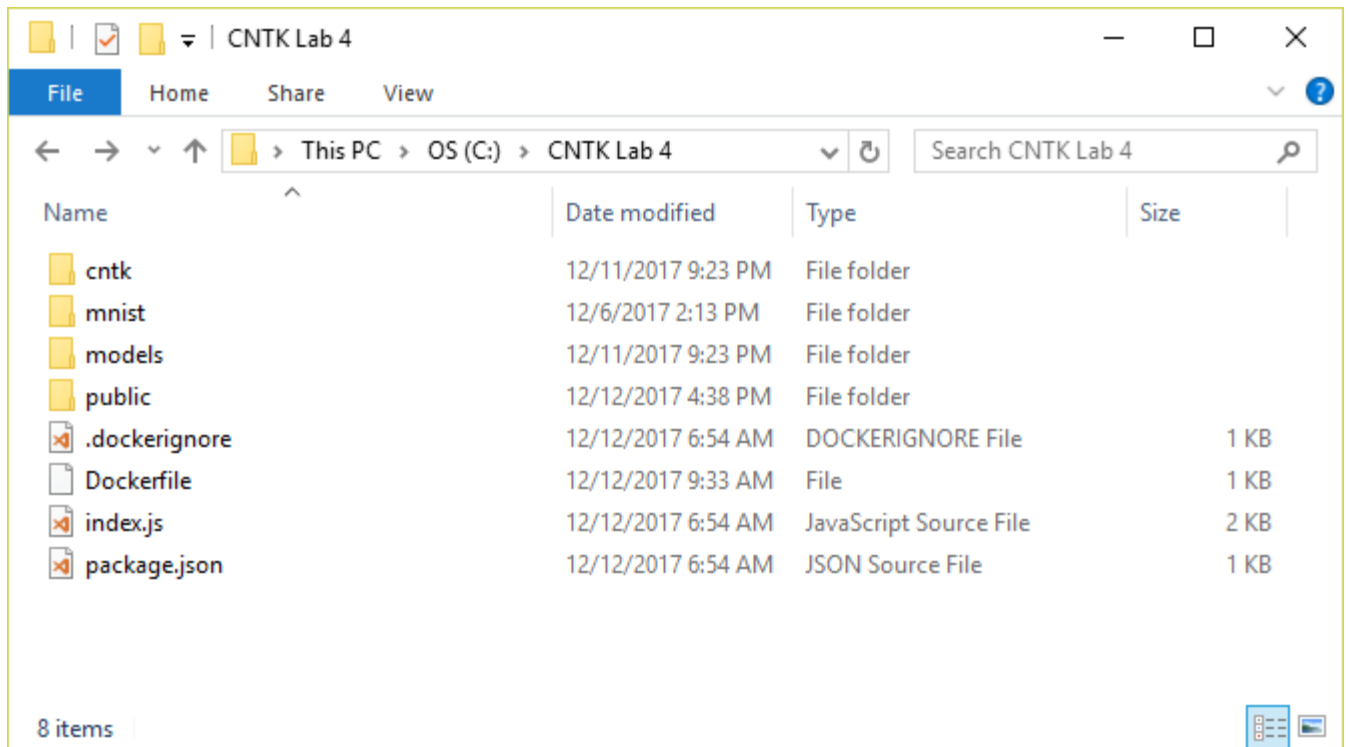
252. Add a file named **.dockerignore** containing the following statement to the project directory:

```
253. node_modules
```

Similar to **.gitignore** in a GitHub repository, this file tells Docker what NOT to include in the container image. In this case, it instructs Docker to ignore the "node\_modules" directory, which is created when NPM INSTALL is run during the build process.

254. Confirm that the project directory contains the files and subdirectories shown below. Also confirm that the "cntk" directory holds a file named **02-OneConv.cntk**, that the "public" directory contains a file named **index.html**, and that the "models" directory contains the **02\_Conv.\*** files that you downloaded from blob storage — 17 files in all.





### *Contents of the project directory*

255. Open a Command Prompt or terminal window and navigate to the project directory. Then execute the following command to build a container image using the files and subdirectories in the project directory and the **Dockerfile** containing build instructions:

```
256. docker build -t mycntk .
```

This command names the container image that is created "mycntk." It will probably take a few minutes to run. The faster your connection speed, the faster the command will execute.

Wait for the build process to complete. The container image is ready. Now let's run it in order to operationalize the neural network that you trained in the previous lab.

## Exercise 2: Run the image in a local container

In this exercise, you will operationalize the "One Convolution" network by running the image that you built in the previous exercise in a Docker container. You will run the container locally, but note that the same container could also be hosted in the cloud using the [Azure Container Service](#) or [Azure Container Instances](#).

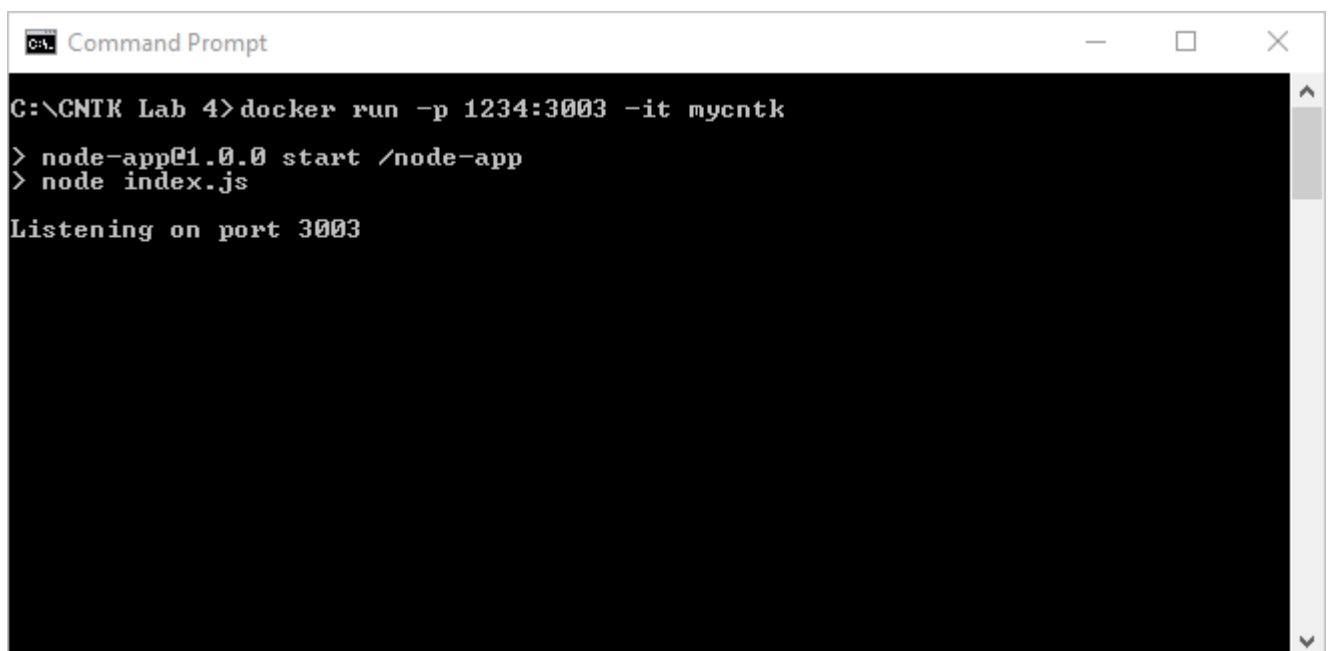
1. Return to the Command Prompt or terminal window in which you executed the `docker build` command in the previous exercise. Then execute the

following command to run a container that uses the Docker image that you built:

2. `docker run -p 1234:3003 -it mycntk`

This command creates a container from the image named "mycntk" and runs it in interactive mode so that output from the code running in the container can be seen in the Command Prompt or terminal window. The `-p` switch redirects traffic destined for port 1234 on the local machine to port 3003 in the container — the port that the Web server running inside the container is listening on.

3. Confirm that the following output appears in the Command Prompt or terminal window indicating that the container is running and that the Web server is listening on port 3003:

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following text: 

```
C:\CNTK Lab 4>docker run -p 1234:3003 -it mycntk
> node-app@1.0.0 start /node-app
> node index.js
Listening on port 3003
```

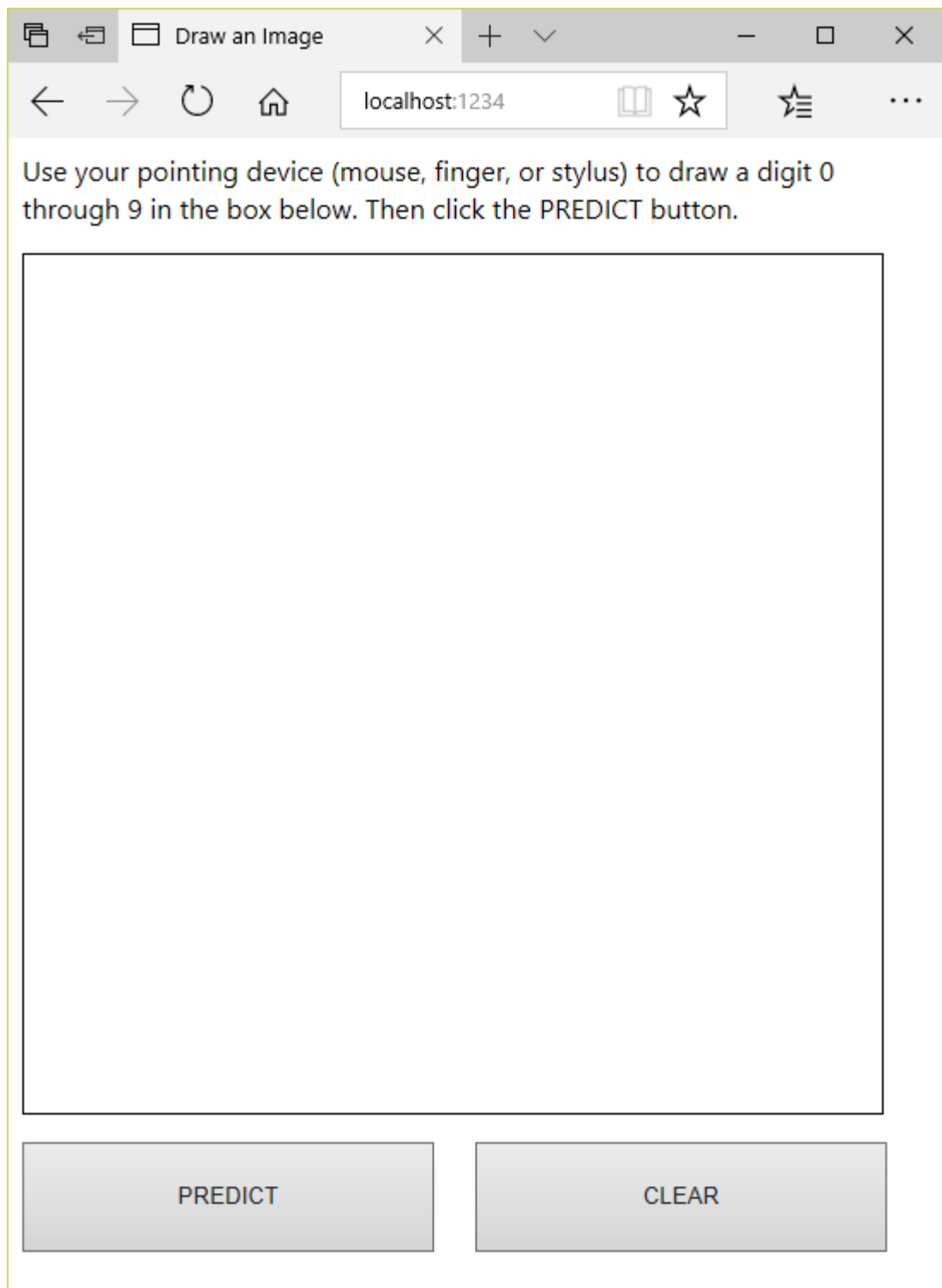
*Output from the running container*

The container is now running and listening for HTTP requests. The final step is to use the Web page inside it to draw a few digits and submit them to the neural network for identification.

## Exercise 3: Use the neural network to identify digits

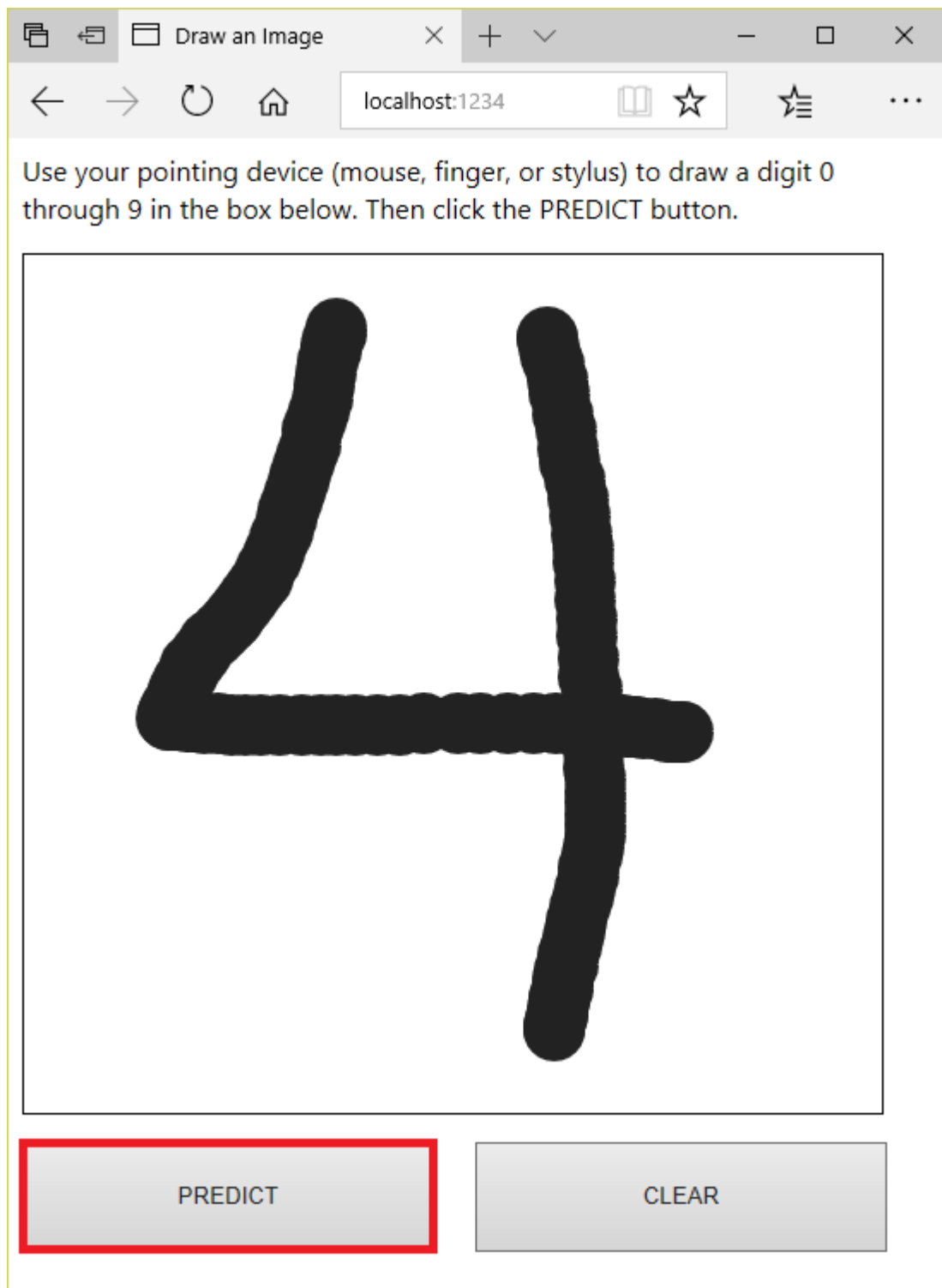
In this exercise, you will use the HTML page in the container to draw a few digits and submit them to the neural network to test its ability to identify the digits that you drew.

1. Open a browser and navigate to <http://localhost:1234>. Confirm that the page shown below appears in the browser



*Web page served up from the container*

2. Use your mouse to sketch a "4" into the grid, similar to the one shown below. Then click the **PREDICT** button.



*Submitting a digit to the neural network*

3. In a moment, a message appears telling you what digit you drew. Is it correct?
4. Click the **Clear** button to clear the display. Then try a few other digits. You'll probably find that the model you built is better at identifying some digits than others, and that you get the best results when the digits you draw fill the expanse of the drawing area as much as possible.

It's a pretty impressive feat for an app to perform basic OCR in this manner. And it's indicative of the kinds of apps you can build when you have machine learning working for you on the back end.