Introduction to Azure Machine Learning: In a Weekend

By

Ajit Jaokar

and

Ayse Mutlu

Contents

Introduction

This book is the second book in the 'in a weekend' series – after Classification and Regression in a weekend. The idea of the 'in a weekend' series of books is to study one complex section of code in a weekend to master the concept. Cloud computing changes the development paradigm. Specifically, it combines development and deployment (the DevOps approach). In complex environments, the developer has to know more than the coding. Rather, she has to be familiar with both the data engineering and the DevOps. This book helps you to get started with coding a complex AI application for the Cloud(Azure). If you don't have an Azure subscription, you can use the free or paid version of Azure Machine Learning service. Note that It is important to delete resources after use.

Azure Machine Learning service is a cloud service that you use to train, deploy, automate, and manage machine learning models – availing all the benefits of a Cloud deployment.

In terms of development, you can use Jupyter notebooks and the Azure SDKs. You can see more details of the Azure Machine Learning Python SDK which allow you to choose environments like Scikit-learn, Tensorflow, PyTorch, and MXNet. We wish to acknowledge the contributions from Amy Kate Boyd - Twitter: @AmyKateNicho (we used her code as a foundation - https://github.com/amynic/azureml-sdk-fashion) and Dr Amita Kapoor for her feedback

The best way to use this book is to work with the Code – which is in three notebooks:

Local - https://colab.research.google.com/drive/1FzIJFgkBC0IXJMnbS6p_COWB-g9-piPr

Training - https://colab.research.google.com/drive/1UBwXWEgB0muz-Ng9oodamnhvjHnH5fqk

Deployment - https://colab.research.google.com/drive/1Ir0WHBpICIEbrkrrQEooNu0prS_YcSae

It is important to delete resources after use.

The problem we solve: Azure ML SDK - Fashion MNIST

In this code, we used the Fashion MNIST dataset. The objective is to classify clothing into categories such as dresses, t-shirts, sandals, trainers etc. The dataset used is the Fashion MNIST dataset from Zalando the online fashion brand, find out more about this dataset here: https://github.com/zalandoresearch/fashion-mnist

We use Keras and Tensorflow with Convolutional Neural Network (CNN).

We first attempt to run the code on a local machine. With 60,000 training images, it is likely to be slow. We next use the Azure ML SDK and set the target compute as a GPU machine in the Cloud.

Next, we use the trained model (which has been registered and versioned) and then deploy that model into Production using a container.

The overall steps are for training and deployment are:

- Creating a training and scoring script for your model
- Creating a bespoke environment for your model to run in (libraries and packages to install)
- Setting up a container service to host your model
- Deploying your model to the cloud container service
- Retrieving the service scoring URL for your hosted model
- Sending test data to your hosted model to be scored and returned to compare for the accuracy of your model on your validation set

What you will learn

In this book, you will learn how to train and deploy a model in the Azure Cloud using containers.

Community for the book

The community link for the book is HERE. We welcome your comments in the community

Pre-requisites - What you need to know

Besides a basic knowledge of Python and data science (see our previous book if you are not familiar with creating a model) – you need to know the basics of MNIST. However, these are explained in the appendix. Assuming you know Python and MNIST (which is effectively the ‘hello world’ of Data Science) – the book helps you to understand deployment to the Azure Cloud

Caution in using the Cloud – Deleting resources

Please exercise caution when using the Cloud with respect to the Billing. Neither the authors nor the publisher are responsible for any issues relating to Billing. Make sure to delete your resources

Make sure you delete the compute when you are not using it to save money

A couple of options ...

Delete the compute items

make sure you delete the batch AI cluster and any container instances you created in your resource group.

Go to your resource group in the Azure Portal (left panel 'resource groups' and select your resource group name)

Next select the compute resources (such as the 'batch AI' cluster) and choose delete

Delete the whole resource group

Or you can delete all resources created

Go to your resource group in the Azure Portal (left panel 'resource groups' and select your resource group name)

Select 'Delete Resource group'

confirm by writing the resource group name in the box

Shutdown/Delete the Virtual Machine

finally, you can shut down your Virtual machine (VM)

In the Azure portal - navigate to the virtual machines tab

select the VM you created

click 'shutdown' or 'delete'

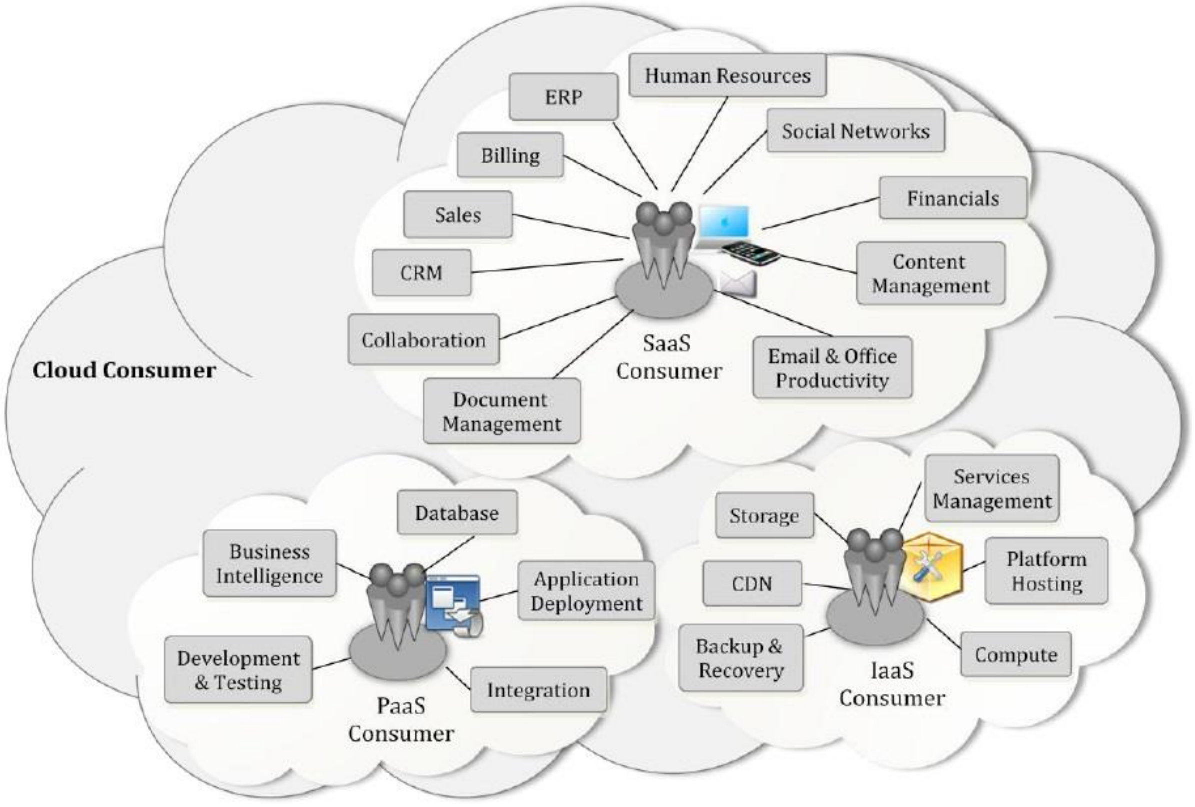
More details below - click here to see how to delete resources

Cloud - Glossary

The Cloud changes development because we look at development and deployment together (a DevOps strategy). In this section, we provide a brief introduction to Cloud terminology as applicable for the content in this book

The Cloud is a metaphor for a global network, first used in reference to the telephone network and now commonly used to represent the Internet. **Cloud computing** is a delivery model for computing resources in which various servers, applications, data and other resources are integrated and provided as a service over the Internet. Resources are often virtualised. The **Microsoft Azure** cloud platform is a growing collection of integrated services, including infrastructure as a service (IaaS) and platform as a service (PaaS) offerings The Cloud offers many benefits.
Benefits of Cloud computing include: Low up-front investment, Cost efficiency, Highly elastic capacity, Ease of use and maintenance, Easier innovation, Better business continuity etc

A big picture of the Cloud is as shown below



Example Services Available to a Cloud Consumer (Source: NIST)

However, more relevant to this discussion is: how the Cloud changes development by incorporating the ideas behind DevOps. **DevOps** is the union of people, process and technology to enable continuous delivery of value to customers. The practice of DevOps brings development and operations teams together to speed software delivery and make products more secure and reliable.

We need to understand some basic terms used in DevOps

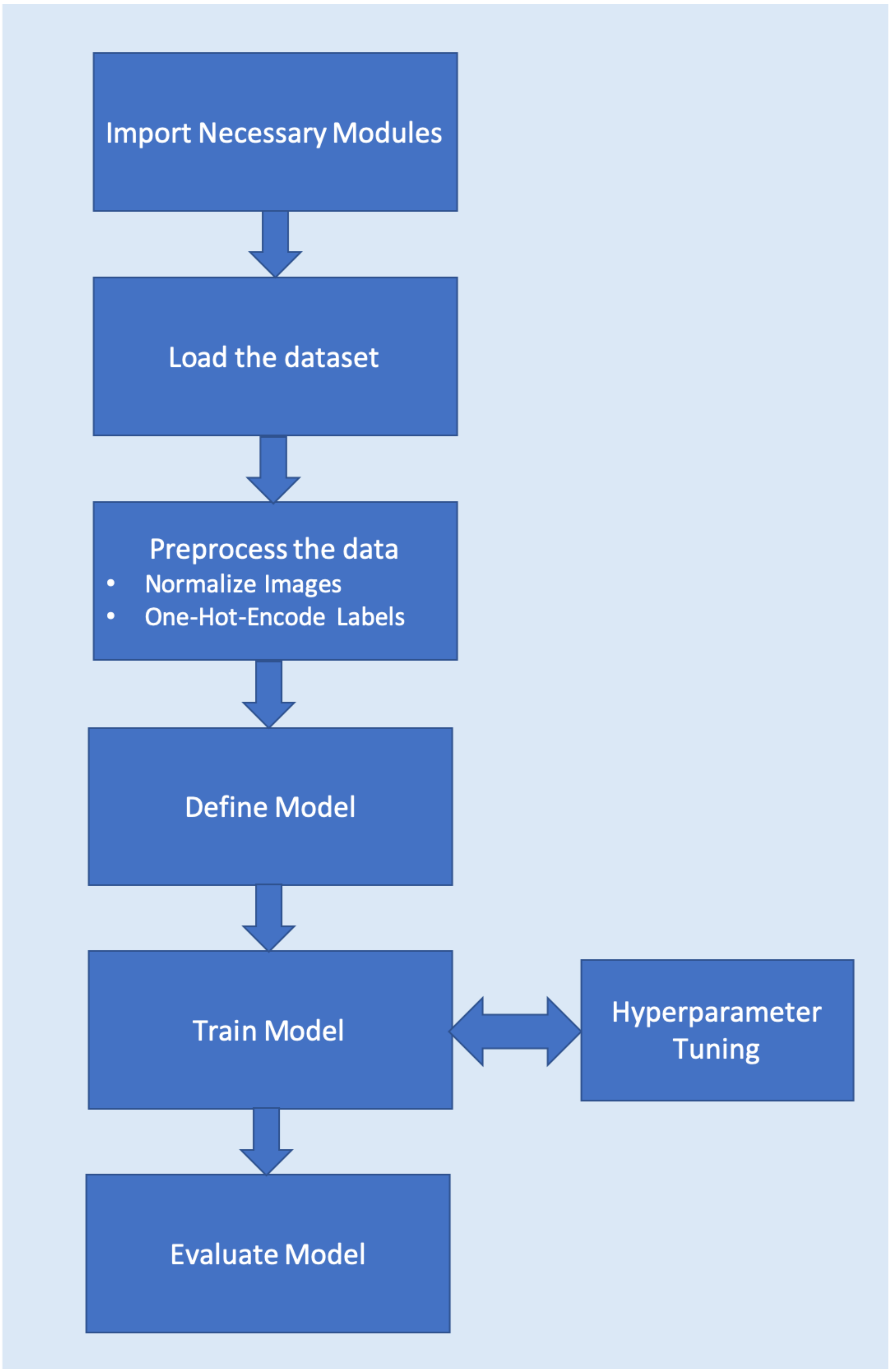
A virtual machine is a computer file (typically called an image) that behaves like an actual computer. Multiple virtual machines can run simultaneously on the same physical computer.

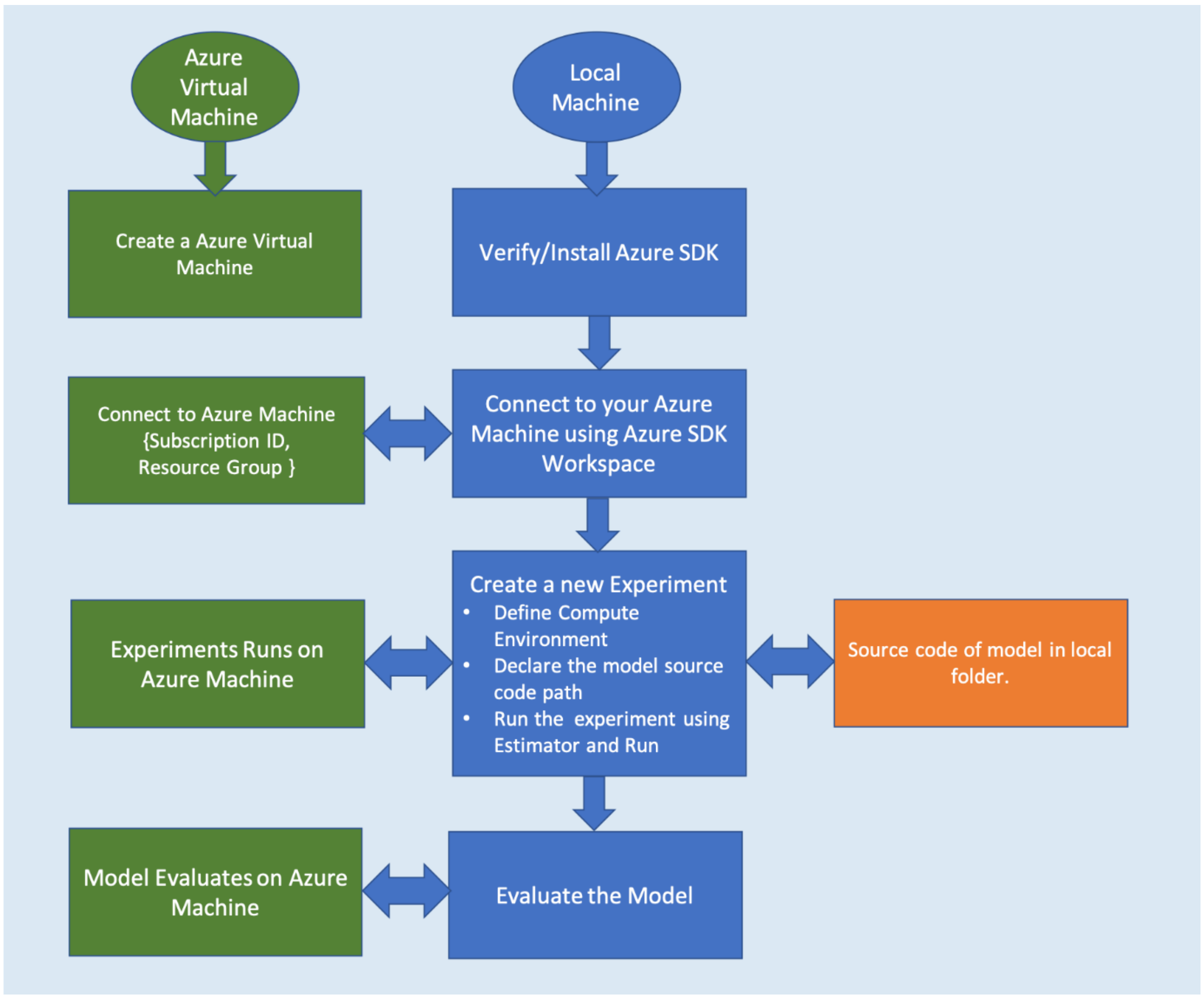
Containers: Containers enable the virtualisation of software applications by providing lightweight runtime environments that include everything apps need to run, making them highly portable. This is foundational to “cloud native” computing.
Docker: An open source platform aimed to deploy and manage virtualized containers.
Dockerfile: A file that contains one or more instructions that dictate how a container is to be created.
Microservices architecture: Describes applications built as collection of single-process services communicating over constrained and easily managed channels (often HTTP), where each service does one well-defined business level task or set of tasks and scales independently of other services. Microservice component boundaries map onto bounded contexts in Domain-Driven Design. The aim is to make changes easier, deployment faster, technology<->business match tighter, infrastructure more automated, conceptual and data models more flexible, and applications more resilient to failure.

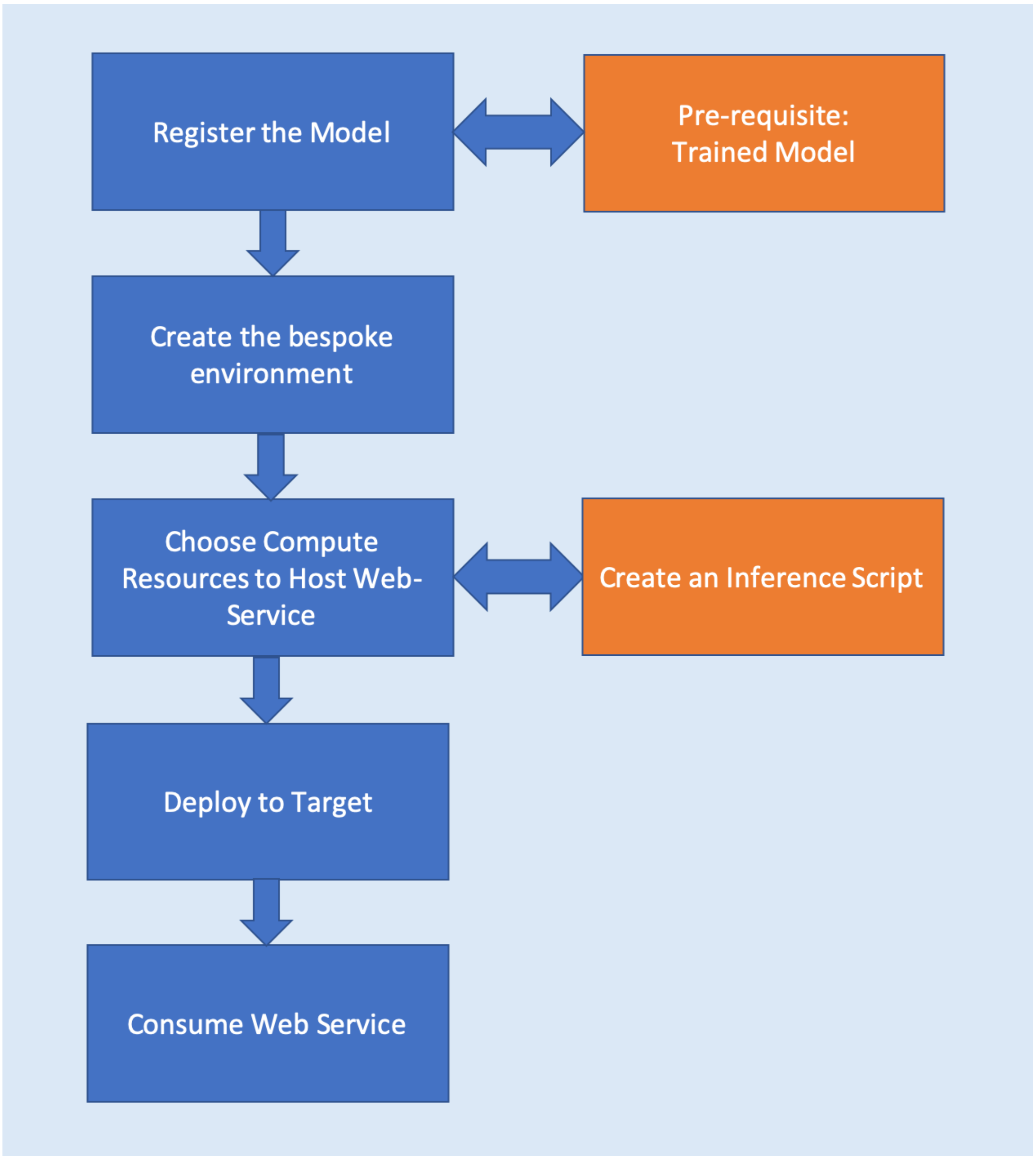
References:
https://azure.microsoft.com/en-gb/overview/cloud-computing-dictionary/
https://www.techrepublic.com/blog/the-enterprise-cloud/mini-glossary-cloud-computing-terms-you-should-know/
https://dzone.com/articles/52-cloud-terms-you-need-to-know
https://www.hpe.com/uk/en/what-is/cloud-computing.html

Code

Flow Diagrams


Import Necessary Modules
Load the dataset
Preprocess the data
• Normalize Images
• One-Hot-Encode Labels
Define Model
Train Model
Hyperparameter Tuning
Evaluate Model


Azure Virtual Machine
Local Machine
Create a Azure Virtual Machine
Verify/Install Azure SDK
Connect to Azure Machine {Subscription ID, Resource Group }
Connect to your Azure Machine using Azure SDK Workspace
Experiments Runs on Azure Machine
Create a new Experiment
• Define Compute Environment
• Declare the model source code path
• Run the experiment using Estimator and Run
Source code of model in local folder.
Model Evaluates on Azure Machine
Evaluate the Model


Register the Model
Pre-requisite: Trained Model
Create the bespoke environment
Choose Compute Resources to Host Web-Service
Create an Inference Script
Deploy to Target
Consume Web Service

Code and links to code

The code links are

Local

https://colab.research.google.com/drive/1FzIJFgkBC0IXJMnbS6p_COWB-g9-piPr

Training

https://colab.research.google.com/drive/1UBwXWEgB0muz-Ng9oodamnhvjHnH5fqk

Deployment

https://colab.research.google.com/drive/1Ir0WHBpICIEbrkrrQEooNu0prS_YcSae

The actual code is presented as below

Local implementation
https://colab.research.google.com/drive/1FzIJFgkBC0IXJMnbS6p_COWB-g9-piPr

```
# -*- coding: utf-8 -*-
"""Local Machine Fashion Mnist.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1FzIJFgkBC0IXJMnbS6p_COWB-
g9-piPr

# Classifying Fashion-MNIST images

In this tutorial, you will be learning about how to classify the
Fashion-MNIST dataset by building a simple convolutional neural network
(CNN) in [Keras](https://keras.io/). Before  going into the details of
model construction, let us learn a little bit more about our dataset:
**Fashion-MNIST**.

The Fashion MNISt dataset contains 70,000 grayscale images of fashion
products from 10 different categories, there are 7,000 images per
category. Each image is of size 28 x 28. The entire dataset is divided
into two: a training set comprising of 60,000 images and the test set
with 10,000 images. The data can be found here:
https://github.com/zalandoresearch/fashion-mnist

## Step 1: Import the necessary modules

We are using here TensorFlow as backend for Keras. If you want to
change the backend used in Keras from Tensorflow to CNTK you can do it
by simply changing the `os.environ['KERAS_BACKEND']` variable to `cntk`
"""

import tensorflow as tf
import os
import time
os.environ["TF_CPP_MIN_LOG_LEVEL"]= "2"
print("tensorflow Version is: " + str(tf.__version__))

import numpy as np
os.environ['KERAS_BACKEND'] = 'tensorflow'
from keras import backend as K
print(os.environ['KERAS_BACKEND'])

"""* We also import all the Keras functions we will need to use to
create a Convolutional Neural Network (CNN)"""

#Fashion MNIST Dataset CNN model development:
https://github.com/zalandoresearch/fashion-mnist
from keras.datasets import fashion_mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import utils, losses, optimizers
import matplotlib.pyplot as plt

"""## Step 2: Declare variables
* We setup some variables for example how many classes there are [0-9]
as well as batch size to send the training sample of data in to the
model and epochs is how many iterations/run thoroughs of the data there
are
* Each image is of size 28 x 28 pixels
"""

#no. of classes
num_classes = 10

# batch size and training iterations (epochs)
batch_size = 128
epochs = 24

#input image dimensions
img_rows,img_cols = 28,28

"""## Step 3: Load the dataset
* Now we load the dataset from the Keras Library. And explore the
dataset. `x_train` and `x_test` are the training and test dataset
features and `y_train` and `y_test` the respective labels.

* We also plot one of the images from the training set with its
corresponding text label.

#### You can change the img_index field to any number between 0 - 60000
to see different images
"""

#data for train and testing
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

print(x_train.shape, 'train set')
print(x_test.shape, 'test set')

# Define the text labels
fashion_mnist_labels = ["Top",          # index 0
                        "Trouser",      # index 1
                        "Jumper",       # index 2
                        "Dress",        # index 3
                        "Coat",         # index 4
                        "Sandal",       # index 5
                        "Shirt",        # index 6
                        "Trainer",      # index 7
                        "Bag",          # index 8
                        "Ankle boot"]   # index 9

img_index=90
label_index = y_train[img_index]
plt.imshow(x_train[img_index])
print('Label Index: ' + str(label_index) + " Fashion Labels: " +
(fashion_mnist_labels[label_index]))
```

```
"""## Step 4: Preprocess the dataset
* Now in step 4 we preprocess the images in both training and test
dataset. The first step in preprocessing is **normalizing** the images,
at present each pixel vale is a number between `1-255`, we divide it by
255 to ensure that the pixel values lie between `0-1`.  Normalization
helps the model to converge as the math becomes easier with smaller
numbers

* Also we need to [one-hot-
encode](https://www.aimldl.org/ml/Recognizing_Handwritten_Digits.html#O
ne-hot-encoding)
the labels, this converts them from numeric labels of `0-9` to binary
vector of size 10.

* Lastly, since we are deadling with greyscale images we have a depth
number = 1 that might be interpreted different dpending on how the
images are treated in the framework used (CNTK, Tensorflow etc)
"""

#type convert and scale the test and training data
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

#one-hot encoding
y_train = utils.to_categorical(y_train, num_classes)
y_test = utils.to_categorical(y_test,  num_classes)

#formatting issues for depth of image (greyscale = 1) with different
kernels (tensorflow, cntk, etc)
if K.image_data_format()== 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0],1,img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols,1)
    x_test = x_test.reshape(x_test.shape[0],img_rows, img_cols,1)
    input_shape = (img_rows, img_cols,1)

"""## Step 5: Define Model Architecture

Now we use the Keras Sequential API to build a simple Convolutional
Neural Network (CNN) layer by layer.

* We build  a **sequential model** meaning every layer passes
information forward to the next layer of the network
* **1st Convoltuional Layer** - extracts features from data source,
these are kernels/filters and feature maps. Feature maps passed to the
next layer. This layer also has a ReLu activation function - Y = max(0,
x) this removes any value <0 and prevents vanishing gradients or
weights <0
* **2nd pooling layer ** - reduces dimensionality, reduce compute and
helps with overfitting of the data.
* **3rd Convoltuional Layer ** -we add a Convoltuional Layer - extracts
features from data source, these are kernels/filters and feature maps.
Feature maps passed to the  next layer. This layer also has a ReLu
```

```
activation function - Y = max(0, x) this removes any value <0 and
prevents vanishing gradients or weights <0
* **4th Pooling Layer ** - reduces dimensionality, reduce compute and
helps with overfitting of the data.
* **5th/6th Dense fully connected layer with softmax function:** put
features together and classify what item of clothing is used

> **Run some experiments to see how when you change the model below and
rerun all the code the accuarcy and model will change:**
* add a dropout layer after the first pooling layer and also before the
final dense layer: `model.add(Dropout(0.5))`
* change the value of dropout between 0 and 1: `model.add(Dropout(X))`
* change the 2 Conv2D layer first variable to 32 instead of 64:
`model.add(Conv2D(32, kernel_size=(3,3), activation = 'relu'))`
* Add padding to each of the Conv2D layers: `model.add(Conv2D(32,
kernel_size=(3,3), padding = 'same', activation = 'relu'))`
"""

#Define the CNN model
model = Sequential()

model.add(Conv2D(64, kernel_size=(3,3), activation = 'relu',
input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, kernel_size=(3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))

#model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

"""## Step 6: Train and evaluate the model
* Now we compile the CNN model and assigns loss/optimiser functions as
well as metrics we wish to view
* We also add a timer so we know how long the model takes to run
* Fit the training data to the model using 24 epoches and batches of 64
images. Pass in the test data as your validation set so we can see how
the accuracy differs on the training set to the validation set as the
model runs through 24 epochs
* Finally we evaluate the model using the test/validation set

> You can look at different optimisers available in Keras and see what
happens when you change this value:
[https://keras.io/optimizers/](https://keras.io/optimizers/)
"""

#compile - how to measure loss
model.compile(loss=losses.categorical_crossentropy,
optimizer=optimizers.Adam(), metrics=['accuracy'])

#train the model and return loss and accuracy for each epoch - history
dictionary
start = time.time()
```

```
hist = model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs, verbose=1, validation_data=(x_test, y_test))
end = time.time()

#evaluate the model on the test data
score = model.evaluate(x_test, y_test, verbose=0)
print('Test Loss: ', score[0])
print('Test Accuracy: ', score[1])
print('Time to run: ', (end-start))

"""* This code plots the training and validation accuracy across 24
epochs
* The training accuracy is often higher but the validation accuracy is
deemed a more real world value
* In case you are not satisfied with your model, you can tune
hyperparameter to get a better performance. The hyperparameters for
this model being:
 * Batch Size
 * Epochs
 * Optimizer
 * Model Architecture
"""

epoch_list = list(range(1, len(hist.history['acc']) + 1))
plt.plot(epoch_list, hist.history['acc'], epoch_list,
hist.history['val_acc'])
plt.legend(('Training Accuracy', "Validation Accuracy"))
plt.show()

"""## Step 7: Test the model

Now we run this code to see a set of 15 images from the test set and
whether the labels are assigned correctly. We made use of the code
given in this [Medium post](https://medium.com/tensorflow/hello-deep-
learning-fashion-mnist-with-keras-50fcff8cd74a) for visualization.
"""

predictions = model.predict(x_test)

# Plot a random sample of 10 test images, their predicted labels and
ground truth
figure = plt.figure(figsize=(20, 8))
for i, index in enumerate(np.random.choice(x_test.shape[0], size=15,
replace=False)):
    ax = figure.add_subplot(3, 5, i + 1, xticks=[], yticks=[])
    # Display each image
    ax.imshow(np.squeeze(x_test[index]))
    predict_index = np.argmax(predictions[index])
    true_index = np.argmax(y_test[index])
    # Set the title for each image
    ax.set_title("{} ({})".format(fashion_mnist_labels[predict_index],
                                  fashion_mnist_labels[true_index]),
                                  color=("green" if predict_index ==
true_index else "red"))

"""### Congratulations you succesfully trained and evaluated the model
on Fashion-MNIST dataset."""
```

Training
https://colab.research.google.com/drive/1UBwXWEgB0muz-Ng9oodamnhvjHnH5fqk

```
# -*- coding: utf-8 -*-
"""Azure Fashion Mnist Training.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1jVJ12Hd9i1pb7DnonmP6bIlNDlPS43
XA

# Running your code on Azure Virtual Machine using Azure SDK

With the availability of manifold cloud computing resources the work of
training your model on cloud has become very easy. In an [earlier
tutorial](https://github.com/amita-kapoor/ailabs/blob/master/Azure-
Tutorial.md) we learned how to create a virtual machine on Microsoft
Azure portal and use it to train a simple model on MNIST dataset using
Jupyter Notebook.

Jupyter Notebooks provide an easy interface for writing the code and
see how it is performing immediately, however for the purpose of
deployment the `.py` files are better as they run faster.

And so in this tutorial we will see how we can combine the flexibility
and ease of working on Jupyter Notebook, along with the speed of `.py`
files, on the Microsoft Azure virtual machine with the help of **Azure
machine Learning SDK**.

### Pre-requisite

* The tutorial assumes that you already have a virtual machine instance
ready and running. In case, it is not so then please refer to this
[step](https://github.com/amita-kapoor/ailabs/blob/master/Azure-
Tutorial.md#creating-a-virtual-machine-for-deep-learning).

* On your local machine install Azure machine learning SDK (_We
recommend using an Anaconda environment_) using pip on CLI:


  ` pip install azureml-sdk[notebooks]`

### Step 1: Verify that your local machine is properly configured

Verify that your local machine is properly configured. Open a command
terminal on your local machine and run  `jupyter notebook`. The command
would open the Jupyter in a browser.

![pic1](pic1.png)

 Now create a new notebook. (or open this notebook
`Using_Azure_SDK.ipynb`)

 The first step we vireify if Azure SDK is correctly installed. To do
it we import it and print its version.
```

```
"""

import azureml.core
print("Azure ML SDK Version: ",azureml.core.VERSION)

"""### Step 2: Connect to your virtual machine on Azure

We will use the `Workspace` class of Azure ML module to connect to the
virtual machine. Please remember that once you run this cell a popoup
window will open and you will need to provide your microsoft account
username and password.

To ensure you connect with the right virtual machine from the Azure
dashboard note down:

* subscription_id
* Resource_group (if you have created one)

![pic2](pic2.png)

To create the workspace we need a unique name, we set the
`create_resource_group` to `True` so that in case it is not created,
the program will create it. Only few US locations support AzureML at
present, so choose the location `eastus2`.
"""

from azureml.core import Workspace
ws = Workspace.create(name='AiLabs2019ver3',
                      subscription_id='11d739b0-c8e5-4754-836a-
9488de187dfe',
                      resource_group='AILabs2_2019',
                      create_resource_group=True,
                      location='eastus2' # supports only few Us
locations for now
                     )

"""Now that workspace is created you can save it for future
refrence."""

# Let us save the configuration file for future
ws.write_config(file_name="ws_config.json")

"""In case you want to work on the same VM (virtual machine) you can
load the config file using:

`ws = Workspace.from_config()`

Let us see the details of our workspace.
"""

import azureml
from azureml.core import Run
# Check workspace config
print(ws.name, ws.location, ws.resource_group, ws.location, sep = '\t')

"""Now each code that you run, runs as an experiment, so we need to
provide it a name, and start an new experiment on the virtual
machine."""
```

```
experiment_name = 'mnist_azure'
from azureml.core import Experiment
exp = Experiment(workspace=ws, name=experiment_name)
print(exp)

"""### Step 3: Create Compute Target

To run the experiment we will need to specify the environment we want
our code to run. Since our machine is on GPU we select the compute
cluster name `gpucluster`, and also decide maximum and minimum number
of nodes. While making the virtual machine we had chosen
[Standard_NC6](https://github.com/amita-
kapoor/ailabs/blob/master/images/vm5.png) machine so we set the compute
SKU to `STANDARD_NC6`. If instead of GPU, you selected a CPU machine
you should set "AML_COMPUTE_CLUSTER_SKU" to `STANDARD_D2_V2`.
"""

from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTarget
import os

# choose a name for your cluster
compute_name = os.environ.get("AML_COMPUTE_CLUSTER_NAME", "gpucluster")
compute_min_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MAX_NODES", 2)

# This example uses GPU VM. For using CPU VM, set SKU to STANDARD_D2_V2
vm_size = os.environ.get("AML_COMPUTE_CLUSTER_SKU", "STANDARD_NC6")


if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print('found compute target. just use it. ' + compute_name)
else:
    print('creating a new compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(vm_size
= vm_size,

min_nodes = compute_min_nodes,

max_nodes = compute_max_nodes)

    # create the cluster
    compute_target = ComputeTarget.create(ws, compute_name,
provisioning_config)

    # can poll for a minimum number of nodes and for a specific
timeout.
    # if no min node count is provided it will use the scale settings
for the cluster
    compute_target.wait_for_completion(show_output=True,
min_node_count=None, timeout_in_minutes=20)

     # For a more detailed view of current AmlCompute status, use the
'status' property
    print(compute_target.status.serialize())
```

```
"""### Step 4: Building your model.
Now that the machine is set, you need to define the folder where your
source code of model training will be.
"""

import os
script_folder = './keras-mnist'
os.makedirs(script_folder, exist_ok=True)

ds = ws.get_default_datastore()
print(ds.datastore_type, ds.account_name, ds.container_name)

"""In the folder you specified above using any text editor of your
choice create a Python file, we have name our `train.py`. The file
contains:
* The import for all necessary modules for building the model and
reading the data.
* Also additionally import `azureml` module.
    ```
    import azureml
    from azureml.core import Workspace, Run
    ```
* The `train.py` reads in the data, pre-process it, define the model.
* Then we instantiate the AzureML Run object to submit it to the VM,
using: `run = Run.get_submitted_run()`
* Now, as you would normally do, compile the model and train it on
training data.
* Finally, save the model so that you can use it in future.

Below you can see our `train.py` code:

%%writefile $script_folder/train.py

import numpy as np
from keras.datasets import mnist
from keras.models import Sequential

from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense,
Dropout
from keras import utils, losses, optimizers
from keras import backend as K

import azureml
from azureml.core import Workspace, Run

# Define Hyper Parameters of the model:
num_classes = 10
batch_size = 128
epochs = 5 #20 to reduce time

# input image dimensions
img_rows, img_cols = 28, 28

#data for train and testing
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# One hot encode labels
y_train = utils.to_categorical(y_train,num_classes)
y_test = utils.to_categorical(y_test,num_classes)
num_classes = y_test.shape[1]


# Create model
model = Sequential()
model.add(Convolution2D(filters=32, kernel_size=3, padding='same',
activation='relu', input_shape=input_shape))
model.add(Convolution2D(filters=64, kernel_size=4, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.4))
model.add(Convolution2D(filters=128, kernel_size=4, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.4))
model.add(Convolution2D(filters=256, kernel_size=4, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=256, kernel_size=4, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))


# get hold of the current run
run = Run.get_submitted_run()

print('Train a deep learning model')
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

history = model.fit(x_train, y_train, validation_data=(x_test, y_test),
                    epochs=epochs, batch_size=batch_size, verbose=2)


#evaluate the model on the test data
print('Predict the test set')
score = model.evaluate(x_test, y_test, verbose=0)
```

```
print('Test Loss: ', score[0])
print('Test Accuracy: ', score[1])

# calculate accuracy on the prediction
print('Accuracy is', score[1])

run.log('accuracy', np.float(score[1]))

os.makedirs('outputs', exist_ok=True)
# note file saved in the outputs folder is automatically uploaded into
experiment record
model.save('outputs/model.h5')

### Step 5: Run the model on cloud compute machine

Now that all the basic steps are in place. We are ready to submit our
code to the VM for execution.
"""

from azureml.train.estimator import Estimator


est = Estimator(source_directory=script_folder,
                script_params=None,
                compute_target=compute_target,
                entry_script='train.py',
                conda_packages=['keras', 'scikit-learn'])

run = exp.submit(config=est)

from azureml.widgets import RunDetails
RunDetails(run).show()

run.wait_for_completion(show_output=True) # specify True for a verbose
log

"""### Step 6: Evaluate the model
Finally, the model is ready let us evaluate it.
"""

print(run.get_metrics())
print(run.get_file_names())

# Don't forget to delete all resources in the end
ws.delete(delete_dependent_resources=True)

"""### Next Steps

Now that the model is trained, we will need to deploy it. In the next
tutorial we will cover steps on how to deploy the saved model.
"""
```

Deployment
https://colab.research.google.com/drive/1Ir0WHBpICIEbrkrrQEooNu0prS_YcSae

```
# -*- coding: utf-8 -*-
"""Azure Fashion Mnist Deployment.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1Ir0WHBpICIEbrkrrQEooNu0prS_YcS
ae

# Fashion MNIST Image Classification - Azure ML SDK Deployment

In this Fashion MNIST notebook we introduce how to take the models you
have registered and versioned, deploy them to production and monitor
them

This code will show how Azure ML SDK can support your machine learning
project with:
* creating a training and scoring script for your model
* creating a bespoke environment for your model to run in (libraries
and packages to install)
* setting up a container service to host your model
* deploying your model to the cloud container service
* retrieving the service scoring URL for your hosted model
* sending test data to your hosted model to be scored and returned to
compare for the accuracy of your model on your validation set

This notebook is based off the great documentation tutorial here:
[https://docs.microsoft.com/en-us/azure/machine-
learning/service/tutorial-deploy-models-with-aml]

## Import packages
"""

# %matplotlib inline
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

import azureml
from azureml.core import Workspace, Run

# display the core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)

"""## Load workspace and download registered model"""

from azureml.core import Workspace
from azureml.core.model import Model

ws = Workspace.from_config()
model=Model(ws, 'keras_dl_fashion_test')
model.download(target_dir = '.', exist_ok = 'True')
import os
# verify the downloaded model file
os.stat('model.h5')

"""## Load and format Test dataset"""
```

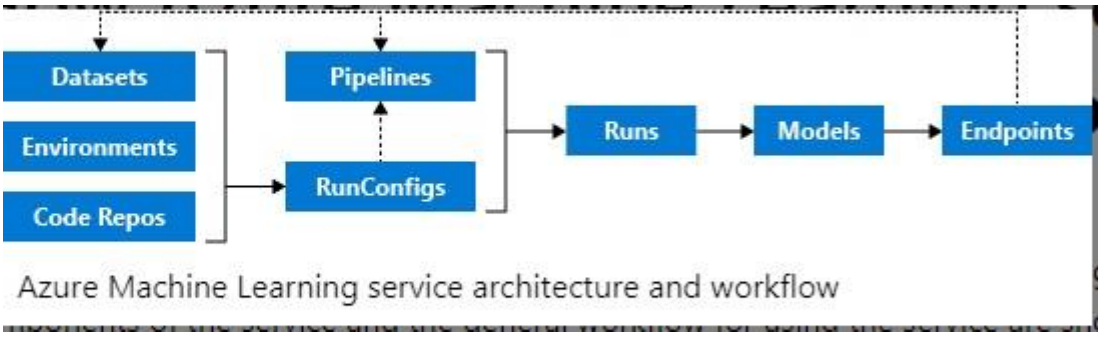
```
import tensorflow as tf
import os
import time
os.environ["TF_CPP_MIN_LOG_LEVEL"]= "2"
print("tensorflow Version is: " + str(tf.__version__))

import numpy as np
os.environ['KERAS_BACKEND'] = 'tensorflow'
from keras import backend as K
print(os.environ['KERAS_BACKEND'])

#Fashion MNIST Dataset CNN model development:
https://github.com/zalandoresearch/fashion-mnist
from keras.datasets import fashion_mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import utils, losses, optimizers
import matplotlib.pyplot as plt
from utils import load_data

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

#no. of classes
num_classes = 10
# Define the text labels
fashion_mnist_labels = ["Top",          # index 0
                        "Trouser",      # index 1
                        "Jumper",       # index 2
                        "Dress",        # index 3
                        "Coat",         # index 4
                        "Sandal",       # index 5
                        "Shirt",        # index 6
                        "Trainer",      # index 7
                        "Bag",          # index 8
                        "Ankle boot"]   # index 9

x_test = x_test.astype('float32')
x_test /= 255
y_test = utils.to_categorical(y_test,  num_classes)

#X_test = load_data('./data/test-images.gz', False) / 255.0
#y_test = load_data('./data/test-labels.gz', True).reshape(-1)

"""## Create a score.py script

%%writefile score.py
import keras as K
from azureml.core.model import Model
from keras.models import Sequential
from keras.models import load_model
import h5py
import sys
import os
import numpy as np
import json
```

```
def init():
    global model
    print("Python version: " + str(sys.version) + ", keras version: " +
K.__version__)
    print("Executing init() method...")
    model_path = Model.get_model_path('model.h5')
    print("got model...")
    model = K.models.load_model(model_path)
    print("loaded model...")

def run(raw_data):
    print("Executing run(raw_data) method...")
    data = np.array(json.loads(raw_data)['data'])
    data = np.reshape(data, (30,28,28,1))
    y_hat = model.predict(data)
    print("Eexcuted predictions...")
    return json.dumps(y_hat.tolist())

## Create a bespoke environment to run your model in
"""

from azureml.core.conda_dependencies import CondaDependencies

myenv = CondaDependencies()
myenv.add_conda_package("scikit-learn")
myenv.add_conda_package("keras")
myenv.add_tensorflow_pip_package(core_type='cpu', version='1.11.0')

with open("myenv.yml","w") as f:
    f.write(myenv.serialize_to_string())

with open("myenv.yml","r") as f:
    print(f.read())

"""## Setup a Container service to host your model in"""

from azureml.core.webservice import AciWebservice

aciconfig = AciWebservice.deploy_configuration(cpu_cores=1,
                                               memory_gb=1,
                                               tags={"data": "fashion-
MNIST",  "method" : "keras"},
                                               description='Deep
Learning to classify fashion clothing items, using Keras')

"""%%time
from azureml.core.webservice import Webservice
from azureml.core.image import ContainerImage

image_config = ContainerImage.image_configuration(execution_script =
"score.py",
                                    runtime = "python",
                                    conda_file = "myenv.yml",
                                    description = "Deep Learning to
classify fashion clothing items, using Keras",
                                    tags = {"data": "fashionmnist",
"type": "classification"}
)
```

```
#the service name must not already exist in your workspace, else rename
it
service_name = 'deployfashion2'
service = Webservice.deploy(deployment_config = aciconfig,
                                image_config = image_config,
                                model_paths = ['model.h5'],
                                name = service_name,
                                workspace = ws)

service.wait_for_deployment(show_output = True)
print(service.state)
"""

print(service.scoring_uri)

"""## Test your model by calling the hosted API with test data"""

import json
s = 100
e = s + 30

#sample_indices = (x_test[0:n])

test_samples = json.dumps({"data": x_test[s:e].tolist()})
test_samples = bytes(test_samples, encoding = 'utf8')

test_labels = y_test[s:e]

##testing input correct
body = np.array(json.loads(test_samples)['data'])

# predict using the deployed model
result = json.loads(service.run(input_data=test_samples))

for i in range(0, len(result)):
    indice_prediction = np.argmax(result[i])
    indice_label = np.argmax(test_labels[i])
    if indice_prediction == indice_label:
        print("correct prediction for sample " + str(i))
    else:
        print("INCORRECT prediction for sample" + str(i))
```

Azure Machine Learning concepts - an Introduction

Sequence of flow



Source: https://docs.microsoft.com/en-gb/azure/machine-learning/service/ove...

The Azure machine learning workflow generally follows this sequence:

1. Develop machine learning training scripts in **Python**.
2. Create and configure a **compute target**.
3. **Submit the scripts** to the configured compute target to run in that environment. During training, the scripts can read from or write to **datastore**. And the records of execution are saved as **runs** in the **workspace** and grouped under **experiments**.
4. **Query the experiment** for logged metrics from the current and past runs. If the metrics don't indicate a desired outcome, loop back to step 1 and iterate on your scripts.
5. After a satisfactory run is found, register the persisted model in the **model registry**.
6. Develop a scoring script that uses the model and **Deploy the model** as a **web service** in Azure, or to an **IoT Edge device**.

Concepts of Azure Machine Learning – an overview

Below, we explain some concepts relating to running machine learning algorithms in the Azure Cloud as per the above flow

The workspace is the top-level resource for Azure Machine Learning service. It provides a centralized place to work with all the artefacts you create when you use Azure Machine Learning service. The workspace keeps a history of all training runs, including logs, metrics, output, and a snapshot of your scripts.

Once you have a model you like, you register it with the workspace. You then use the registered model and scoring scripts to deploy to Azure Container Instances, Azure Kubernetes Service, or to a field-programmable gate array (FPGA) as a REST-based HTTP endpoint. You can also deploy the model to an Azure IoT Edge device as a module.
When you create a new workspace, it automatically creates several Azure resources that are used by the workspace:

- Azure Container Registry: Registers docker containers that you use during training and when you deploy a model.
- Azure Storage account: Is used as the default datastore for the workspace
- Azure Application Insights: Stores monitoring information about your models.
- Azure Key Vault: Stores secrets that are used by compute targets and other sensitive information that's needed by the workspace.

An experiment is a grouping of many runs from a specified script. It is always associated to a workspace.

A model: has the same meaning as in traditional machine learning. In Azure, a model is produced by a run but models trained outside Azure can also be used. Azure Machine Learning service is framework agnostic i.e. popular machine learning frameworks such as Scikit-learn, XGBoost, PyTorch, TensorFlow etc can be used.

The model registry keeps track of all the models in your Azure Machine Learning service workspace.

A run configuration is a set of instructions that defines how a script should be run in a specified compute target. The configuration includes a wide set of behavior definitions, such as whether to use an existing Python environment or to use a Conda environment that's built from a specification

Azure Machine Learning Datasets manage data in various scenarios such as model training and pipeline creation. Using the Azure Machine Learning SDK, you can access underlying storage, explore and prepare data, manage the life cycle of different Dataset definitions, and compare between Datasets used in training and in production.

A datastore is a storage abstraction over an Azure storage account. The datastore can use either an Azure blob container or an Azure file share as the back-end storage. Each workspace has a default datastore, and you can register additional datastores. You can use the Python SDK API or the Azure Machine Learning CLI to store and retrieve files from the datastore.

A compute target is the compute resource that you use to run your training script or host your service deployment. Your local computer, A linux VM in Azure; Azure Databricks etc are all examples of compute targets

A training script – brings it all together. During training, the directory containing the training script and the associated files is copied to the compute target and executed there. A snapshot of the directory is also stored under the experiment in the workspace.

A run – A run is produced when you submit a script to train a model. A run is a record that contains information like the metadata about the run, output files, metrics etc

Github tracking and integration - When you start a training run where the source directory is a local Git repository, information about the repository is stored in the run history.
Snapshot – snapshot(of a run) represents compressed directory that contains the script as a zip file maintained as a record. The snapshot is also sent to the compute target where the zip file is extracted and the script is run there.

An activity represents a long running operation. Creating or deleting a compute target. Running a script on the compute target etc are all examples of activities.
Image Images provide a way to reliably deploy a model, along with all components you need to use the model. An image contains of a model, scoring script or application and dependencies that are needed by the model/scoring script/application
Azure Machine Learning can create two types of images:

- **FPGA image**: Used when you deploy to a field-programmable gate array in Azure.
- **Docker image**: Used when you deploy to compute targets other than FPGA. Examples are Azure Container Instances and Azure Kubernetes Service.

Image registry: The image registry keeps track of images that are created from your models. You can provide additional metadata tags when you create the image.

Deployment: A deployment is an instantiation of your model into either a web service that can be hosted in the cloud or an IoT module for integrated device deployments.

Web service: A deployed web service can use Azure Container Instances, Azure Kubernetes Service, or FPGAs. You create the service from your model, script, and associated files. These are encapsulated in an image, which provides the run time environment for the web service. The image has a load-balanced, HTTP endpoint that receives scoring requests that are sent to the web service.

Iot module: A deployed IoT module is a Docker container that includes your model and associated script or application and any additional dependencies. You deploy these modules by using Azure IoT Edge on edge devices.

Pipelines: You use machine learning pipelines to create and manage workflows that stitch together machine learning phases. For example, a pipeline might include data preparation, model training, model deployment, and inference/scoring phases. Each phase can encompass multiple steps, each of which can run unattended in various compute targets.

Source: Azure documentation here.

STEPS for Training

1. Create Virtual Machine
2. Installation azureml-sdk(on your local machine)
3. Import libraries
4. Create workspace
5. Save workspace configuration file for future
6. Define Experiment
7. Create Compute Target
8. Define Datastore
9. Write train.py and save model.h5
    1. Setup variables

    2. Data for training and testing
    3. Define the text labels
    4. Explore data
    5. Normalise data
    6. One-hot encoding
    7. Formatting issues for depth of image
    8. Define the CNN Model
    9. Instantiate AzureML Run object to submit it to VM (using run = Run.get_submitted_run())
    10. Compile the model
    11. Train the model
    12. Evaluate the model on the test data
    13. Save the model as model.h5
10. Run the model on cloud compute machine
    1. Define Estimator and submit it
11. Evaluate the model
12. Delete all resources

STEPS for deployment

1. Import libraries
2. Load workspace
3. Download registered model
4. Load and format test dataset
5. Create a score.py script
6. Create a bespoke environment to run your model in
7. Setup a container service to host your model in
8. Test your model by calling the hosted API with test data

Training and Deployment – extended discussion

In this section, we extend the previous discussion with more detail.

What is a Virtual Machine?

A virtual machine, or VM, is a software emulation of a physical computer. A virtual machine is defined by a number of factors, including its size and location. A VM's ***size*** defines its processor speed, amount of memory, initial amount of storage, and expected network bandwidth. Some sizes even include specialized hardware such as GPUs for heavy graphics rendering and video editing. Azure is made up of data centers distributed throughout the world. A ***region*** is a set of Azure data centers in a named geographic location. Every Azure resource, including virtual machines, is assigned a region. A ***virtual network*** is a logically

isolated network on Azure. Each virtual machine on Azure is associated with a virtual network. Azure provides cloud-level firewalls for your virtual networks called ***network security groups*****.** Virtual machines and other cloud resources are grouped into logical containers called *resource* ***groups***. Groups are typically used to organize sets of resources that are deployed together as part of an application or service. You refer to a resource group by its name.
Reference: https://docs.microsoft.com/en-us/learn/modules/welcome-to-azure/4-create-a-vm?pivots=windows-cloud

What is Azure Machine Learning SDK?

Data scientists and AI developers use the Azure Machine Learning SDK for Python to build and run machine learning workflows with the Azure Machine Learning service.
Key areas of the SDK include:

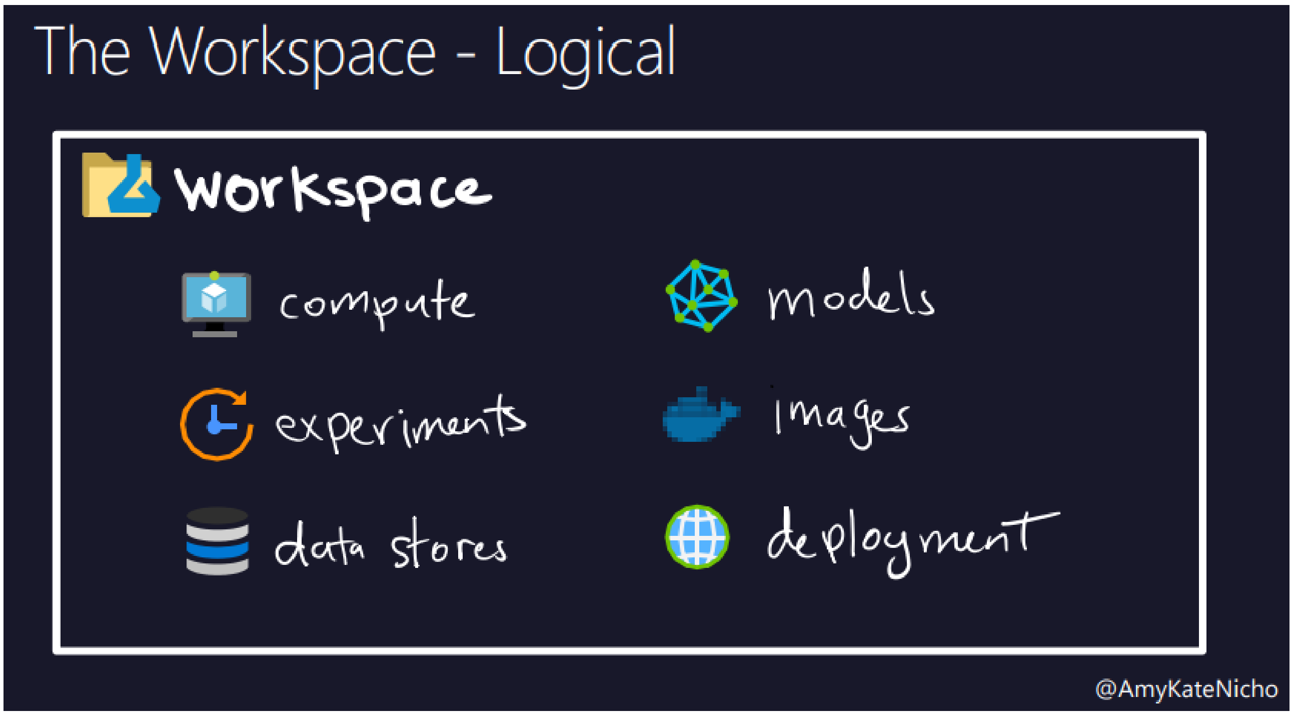
- Explore, prepare and manage the lifecycle of your datasets used in machine learning experiments.
- Manage cloud resources for monitoring, logging, and organizing your machine learning experiments.
- Train models either locally or by using cloud resources, including GPU-accelerated model training.
- Use automated machine learning, which accepts configuration parameters and training data. It automatically iterates through algorithms and hyperparameter settings to find the best model for running predictions.
- Deploy web services to convert your trained models into RESTful services that can be consumed in any application.

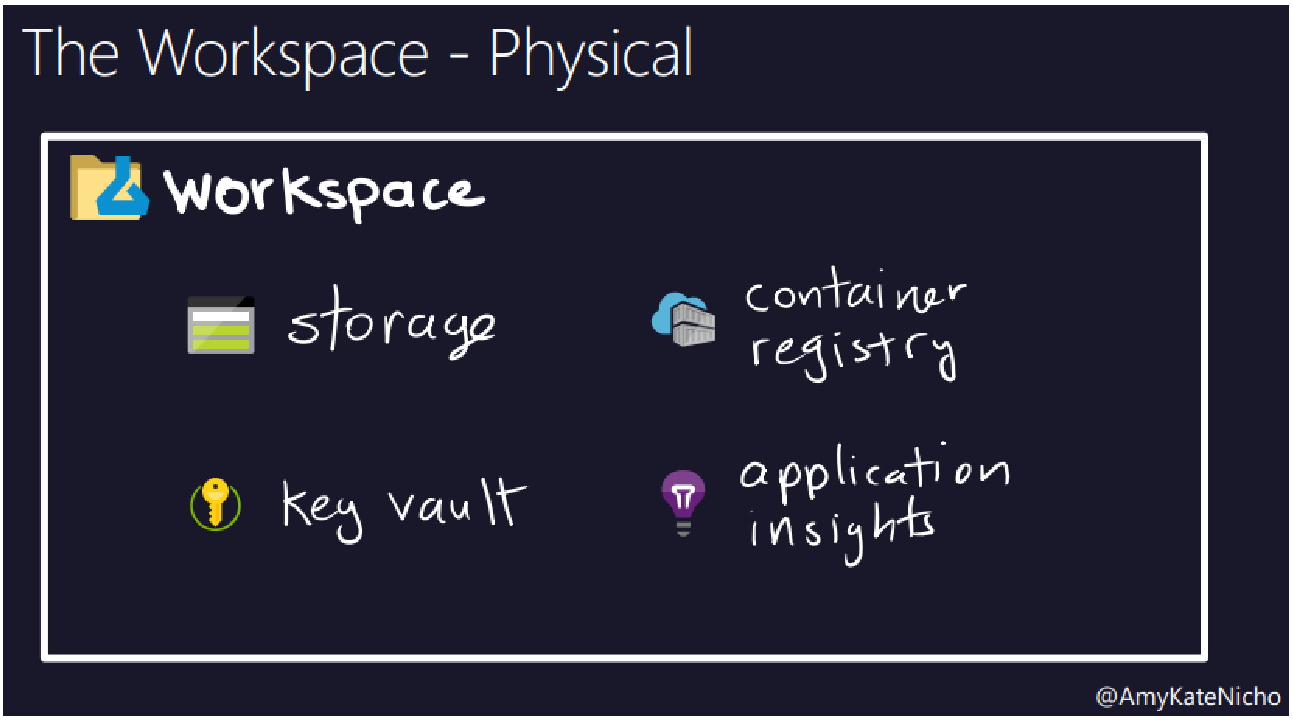
Source:
https://docs.microsoft.com/en-us/python/api/overview/azure/ml/intro?view=azure-ml-py

What is Workspace?

The workspace is the top-level resource for Azure Machine Learning service. It provides a centralized place to work with all the artefacts you create when you use Azure Machine Learning service. The workspace keeps a history of the training runs, including logs, metrics, output, and a snapshot of your scripts. You use this information to determine which training run produces the best model. Once you have a model you like, you register it with the workspace. You use the registered model and scoring scripts to deploy to Azure Container Instances, Azure Kubernetes Service, or to a field-programmable gate array (FPGA) as a REST-based HTTP endpoint. You can also deploy the model to an Azure IoT Edge device as a module.
https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#workspace

https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-workspace





In the code, we use the Workspace class of Azure ML module to connect to the virtual machine. Once you run this cell a popoup window will open and you will need to provide your microsoft account username and password and also the subscription_id and Resource_group (if you have created one). Also we need a unique name, we set the create_resource_group to True so that in case it is not created, the program will create it. Only few US locations support AzureML at present, so choose the location eastus2.

```
ws = Workspace.create(name='AiLabs2019ver3',
                subscription_id='11d739b0-c8e5-4754-836a-9488de187dfe',
                resource_group='AILabs2_2019',
                create_resource_group=True,
                location='eastus2' # supports only few Us locations for now
               )
```

To see the details of workspace.

```
print(ws.name, ws.location, ws.resource_group, ws.location, sep = '\t')
```

What is a resource group?

It is a container that holds related resources for an Azure solution. In Azure, you logically group related resources such as storage accounts, virtual networks, and virtual machines (VMs) to deploy, manage, and maintain them as a single entity.

Workspace.write_config(): Writes out the Workspace ARM properties to a config file. The method provides a simple way of reusing the same workspace across multiple python notebooks or projects. Users can save the workspace properties using this function, and use from_config to load the same workspace in different python notebooks or projects without retyping the workspace ARM properties. After you create workspace, you can save the configuration file for future reference.

```
ws.write_config(file_name="ws_config.json")
ws = Workspace.from_config()
```

What is Experiment in Azure?

An experiment is a grouping of many runs from a specified script. It always belongs to a workspace. When you submit a run, you provide an experiment name. Information for the run is stored under that experiment. If you submit a run and specify an experiment name that doesn't exist, a new experiment with that newly specified name is automatically created.

https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#experiment

An experiment is a way to keep your projects organised and allow you develop and iterate on your machine learning models keeping the history of each edit you make (we will see this in action shortly). We give the experiment a name and create that experiment within our connected workspace.

```
experiment_name = 'mnist_azure'
from azureml.core import Experiment
exp = Experiment(workspace=ws, name=experiment_name)
```

```
print(exp)
```

What is a Compute Target?

Azure Machine Learning Compute is a managed-compute infrastructure that allows the user to easily create a single or multi-node compute. The compute is created within your workspace region as a resource that can be shared with other users in your workspace. The compute scales up automatically when a job is submitted, and can be put in an Azure Virtual Network. The compute executes in a containerized environment and packages your model dependencies in a Docker container. You can use Azure Machine Learning Compute to distribute the training process across a cluster of CPU or GPU compute nodes in the cloud. For more information on the VM sizes that include GPUs, see GPU-optimized virtual machine sizes.

A compute target is the compute resource that you use to run your training script or host your service deployment.
https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#compute-target
https://docs.microsoft.com/en-us/azure/machine-learning/service/how-to-set-up-training-targets
We can create compute within the Azure Cloud and point our experiment at this compute and tell Azure to run our experiment there. This allow us to created GPU machines easily within the cloud and run our machine learning experiments against powerful compute.
We call this our Compute Target. AML Compute is a specific part of the Azure machine learning service where the compute nodes are spun up and down in size as and when you submit the experiment – this is autoscaling compute on demand.
The code below imports the need packages, sets a maximum and minimum number of nodes for your compute (if you don’t want to be charged all the time for compute set the minimum to 0, if you always need the compute to be ready to run, set the minimum node to > 0). You also have a chance to set the size and type of the machine. Our DSVM is a

STANDARD_D2_V2 size so let’s set out compute in the cloud to a GPU instance – STANDARD_NC6.

If the compute is already created in your Azure Machine Learning workspace – it will reuse the already created compute. We then create the target in our account and assign it to the compute_target variable to use later.

```
from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTarget
import os

# choose a name for your cluster
compute_name = os.environ.get("AML_COMPUTE_CLUSTER_NAME", "gpucluster")
compute_min_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MAX_NODES", 2)

# This example uses GPU VM. For using CPU VM, set SKU to STANDARD_D2_V2
vm_size = os.environ.get("AML_COMPUTE_CLUSTER_SKU", "STANDARD_NC6")


if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print('found compute target. just use it. ' + compute_name)
else:
    print('creating a new compute target...')
    provisioning_config = AmlCompute.provisioning_configuration(vm_size = vm_size,
                                                                min_nodes = compute_min_nodes,
                                                                max_nodes = compute_max_nodes)

    # create the cluster
    compute_target = ComputeTarget.create(ws, compute_name, provisioning_config)

    # can poll for a minimum number of nodes and for a specific timeout.
    # if no min node count is provided it will use the scale settings for the cluster
    compute_target.wait_for_completion(show_output=True, min_node_count=None, timeout_in_minutes=20)

     # For a more detailed view of current AmlCompute status, use the 'status' property
    print(compute_target.status.serialize())
```

Submit machine learning experiment to the cloud

To submit our machine learning experiment to the cloud, we will need to send a few files such as a training python script, a configuration file etc.

```
import os
script_folder = './keras-mnist'
```

```
os.makedirs(script_folder, exist_ok=True)
```

Datastore

A datastore is a storage abstraction over an Azure storage account. The datastore can use either an Azure blob container or an Azure file share as the back-end storage. Each workspace has a default datastore, and you can register additional datastores. Use the Python SDK API or the Azure Machine Learning CLI to store and retrieve files from the datastore

```
ds = ws.get_default_datastore()
print(ds.datastore_type, ds.account_name, ds.container_name)
```

https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#datastore

Training script

To train a model, you specify the directory that contains the training script and associated files. You also specify an experiment name, which is used to store information that's gathered during training. During training, the entire directory is copied to the training environment (compute target), and the script that's specified by the run configuration is started. A snapshot of the directory is also stored under the experiment in the workspace.
https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#training-script

While building the model, this time we are writing this code to a folder created above as train.py so we can submit this code to the cloud.

```
%%writefile $script_folder/train.py
```

Other specific additions to this file include:

- **Running:** you instantiate the AzureML Run object to submit it to the VM

  ```
  run = Run.get_submitted_run()
  ```

- **Logging:** you can log any variables in your code and then they are available to monitor within the cloud and in the output of the script

  ```
  run.log('accuracy', np.float(score[1]))
  ```

- **Saving model:** Save the model so that you can use it in future

  File saved in the outputs folder is automatically uploaded into experiment record

  ```
  model.save('outputs/model.h5')
  ```

Creating Estimator

We have our script now we need to state within our compute target what we need to environment to look like. So we create an estimator to do this, containing where the scripts are locally, any parameters we need to add, what is our compute target, what script do we want to run and also any packages that need to be installed in the environment to allow this code to run.

```
from azureml.train.estimator import Estimator
est = Estimator(source_directory=script_folder,
            script_params=None,
            compute_target=compute_target,
            entry_script='train.py',
            conda_packages=['keras', 'scikit-learn'])
```

Deploying model to the cloud

The model registry keeps track of all the models in your Azure Machine Learning service workspace.

https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#model

We are now ready to host our model as a web service so that applications can call into and use our model – therefore we are going to deploy our model to a container in the cloud.

First, we register our model within the Azure Machine Learning workspace using the register_model function, giving a name and pointing towards our model file.

```
model = run.register_model(model_name='keras_dl_fashion_test', model_path='outputs/')
```

Deployment

A deployment is an instantiation of your model into either a web service that can be hosted in the cloud or an IoT module for integrated device deployments.

Web service

A deployed web service can use Azure Container Instances, Azure Kubernetes Service, or FPGAs. You create the service from your model, script, and associated files. These are encapsulated in an image, which provides the run time environment for the web service. The image has a load-balanced, HTTP endpoint that receives scoring requests that are sent to the web service.

IoT module

A deployed IoT module is a Docker container that includes your model and associated script or application and any additional dependencies. You deploy these modules by using Azure IoT Edge on edge devices.

If you've enabled monitoring, Azure collects telemetry data from the model inside the Azure IoT Edge module. The telemetry data is accessible only to you, and it's stored in your storage account instance.

Azure IoT Edge ensures that your module is running, and it monitors the device that's hosting it.

https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#deployment

Load workspace and download registered model

Here is how you download the model from your cloud workspace.

```
ws = Workspace.from_config()
model=Model(ws, 'keras_dl_fashion_test')
model.download(target_dir = '.', exist_ok = 'True')
```

Create a score.py script

We loaded and formatted test dataset. Now we are creating score.py script**.**

Score file that will be ran within a container, this is the inference stage of a machine learning project.

- We import any packages needed to run the python script and model

- We build an initialise method that is called first, this is where we point at the model and load it to be used
- Next the run function will be called, this is where we pass the new input data and call the model with this data and return a JSON response for the prediction

```
%%writefile score.py
import keras as K
from azureml.core.model import Model
from keras.models import Sequential
from keras.models import load_model
import h5py
import sys
import os
import numpy as np
import json

def init():
    global model
    print("Python version: " + str(sys.version) + ", keras version: " + K.__version__)
    print("Executing init() method...")
    model_path = Model.get_model_path('model.h5')
    print("got model...")
    model = K.models.load_model(model_path)
    print("loaded model...")

def run(raw_data):
    print("Executing run(raw_data) method...")
    data = np.array(json.loads(raw_data)['data'])
    data = np.reshape(data, (30,28,28,1))
    y_hat = model.predict(data)
    print("Eexcuted predictions...")
    return json.dumps(y_hat.tolist())
```

Create a bespoke environment to run your model in

We have a script to run in the container, however, as we did with training computer target, we need to setup the environment configuration for the container. Setting up the container so that the machine learning model has everything it needs to run. In the code below we write a file called myenv.yml (a YAML file) and we add any dependency packages such as Scikit-learn, pandas and matplotlib

```
from azureml.core.conda_dependencies import CondaDependencies
myenv = CondaDependencies()
myenv.add_conda_package("scikit-learn")
myenv.add_conda_package("keras")
myenv.add_tensorflow_pip_package(core_type='cpu', version='1.11.0')

with open("myenv.yml","w") as f:
    f.write(myenv.serialize_to_string())
```

Image

Images provide a way to reliably deploy a model, along with all components you need to use the model. An image contains the following items:

- A model.
- A scoring script or application. You use the script to pass input to the model and return the output of the model.
- The dependencies that are needed by the model or scoring script or application. For example, you might include a Conda environment file that lists Python package dependencies.

Azure Machine Learning can create two types of images:

- **FPGA image**: Used when you deploy to a field-programmable gate array in Azure.
- **Docker image**: Used when you deploy to compute targets other than FPGA. Examples are Azure Container Instances and Azure Kubernetes Service.

The Azure Machine Learning service provides a base image, which is used by default. You can also provide your own custom images.
https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#image

Setup a Container service to host your model

Currently we have model, a score.py script and an enviroment setup file – this is what will run in the container, now lets setup the infrastructure of the container to run in the cloud.
Create the web service by combining the setup elements:

- Pull together the **image configuration (image_config)** using the scoring script, the python runtime needed, and the packages needed for the environment
- Deploy the **web service (service):** naming the deployment, add the container infrastructure configuration, add the model and add the image for the containers
- **Deploy the container** and track the progress

To achieve this goal, we use the following packages from the azureml library:
azureml.core packages:

It contains core packages, modules and classes for Azure Machine Learning.Main areas include managing compute targets, creating/managing workspaces and experiments, and submitting/accessing model runs and run output/logging.
https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core?view=azure-ml-py

azureml.core.webservice packages:

This package contains classes used to manage Webservice objects within Azure Machine Learning service.
https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.webservice?view=azure-ml-py

azureml.core.image packages:

This package is used for managing the Images you create with Azure Machine Learning service. These Images are used to deploy a user's Model as a Webservice. Images contain a Model, execution script, and any dependencies needed for Model deployment. This Images package has multiple subclasses such as ContainerImage for Docker Images, as well as preview Images like FPGA.
https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.image?view=azure-ml-py

from azureml.core.webservice import AciWebservice

AciWebservice class: Class for Azure Container Instances Webservices

AciWebservice.deploy_configuration():

Create a configuration object for deploying an ACI Webservice. It can take the parameters below. Default value of parameters are None. In the end, object returns a configuration object to use when deploying a Webservice object

Parameters:

cpu_cores: The number of cpu cores to allocate for this Webservice. Can be a decimal

memory_gb: The amount of memory (in GB) to allocate for this Webservice. Can be a

tags: Dictionary of key value tags to give this Webservice

description: A description to give this Webservice

Below we setup the configuration of creating a container instance. How many cores, how much memory is needed and adding a sensible description, so we know what is running in this container

We created configuration object for deploying an ACI(Azure Container Instances Webservices). We defined 1 cpu core, 1 gb memory, tags and description.

```
aciconfig = AciWebservice.deploy_configuration(cpu_cores=1,
                                               memory_gb=1,
                                               tags={"data": "fashion-MNIST",  "method" : "keras"},
                                               description='Deep Learning to classify fashion clothing items, using Keras')
```

https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.webservice.aciwebservice?view=azure-ml-py

from azureml.core.image import ContainerImage

ContainerImage class: Class for container images, currently only for Docker images.

The image contains the dependencies needed to run the model including:

- The runtime
- Python environment definitions specified in a Conda file
- Ability to enable GPU support
- Custom Docker file for specific run commands

ContainerImage.image_configuration()

Create and return a ContainerImageConfig object.

This function accepts parameters to define how your model should run within the Webservice, as well as the specific environment and dependencies it needs to be able to run.

Parameters:

execution_script: Path to local Python file that contains the code to run for the image. Must include both init() and run(input_data) functions that define the model execution steps for the Webservice.

runtime: The runtime to use for the image. Current supported runtimes are 'spark-py' and 'python'.

conda_file: Path to local .yml file containing a Conda environment definition to use for the image.

description: A text description to give this image.

tags: Dictionary of key value tags to give this image.

```
image_config = ContainerImage.image_configuration(execution_script = "score.py",
                        runtime = "python",
                        conda_file = "myenv.yml",
                        description = "Deep Learning to classify fashion clothing items, using Keras",
                        tags = {"data": "fashionmnist", "type": "classification"})
```

https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.image.containerimage?view=azure-ml-py#image-configuration-execution-script--runtime--conda-file-none--docker-file-none--schema-file-none--dependencies-none--enable-gpu-none--tags-none--properties-none--description-none--base-image-none--base-image-registry-none-

from azureml.core.webservice Webservice

Webservice class: Class for Azure Machine Learning Webservices.

Webservice constructor is used to retrieve a cloud representation of a Webservice object associated with the provided Workspace. Returns an instance of a child class corresponding to the specific type of the retrieved Webservice object. Class allows for deploying machine learning models from either a Model or Image object.

Webservice.deploy():

Deploy a Webservice from zero or more model files.

This will register any models files provided and create an image in the process, all associated with the specified Workspace. Use this function when you have a directory of models to deploy that haven't been previously registered.

It can take the parameters below and, it returns Webservice object corresponding to the deployed webservice

workspace: A Workspace object to associate the Webservice with

name: The name to give the deployed service. Must be unique to the workspace.

model_paths: A list of on-disk paths to model files or folder. Can be an empty list.

image_config: An ImageConfig object used to determine required Image properties.

deployment_config: A WebserviceDeploymentConfiguration used to configure the webservice. If one is not provided, an empty configuration object will be used based on the desired target.

We created Webservice. We defined configuration object for deploying an ACI(aciconfig), (image_config), path to our model files(model_paths), service name and workspace.

```
service_name = 'deployfashion2'
service = Webservice.deploy(deployment_config = aciconfig,
                            image_config = image_config,
                            model_paths = ['model.h5'],
                            name = service_name,
                            workspace = ws)


service.wait_for_deployment(show_output = True)
print(service.state)
```

We have a scoring URl we can send data to and retrieve a prediction

```
print(service.scoring_uri)
```

https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.webservice.webservice.webservice?view=azure-ml-py

Test your model by calling the hosted API with test data

Finally, we can test our web service by sending some sample data and retrieving a response. In the response we can see the input data, the actual label and the predicted label of the network intrusion.

```
import json
s = 100
e = s + 30

#sample_indices = (x_test[0:n])

test_samples = json.dumps({"data": x_test[s:e].tolist()})
test_samples = bytes(test_samples, encoding = 'utf8')

test_labels = y_test[s:e]

##testing input correct
body = np.array(json.loads(test_samples)['data'])

# predict using the deployed model
result = json.loads(service.run(input_data=test_samples))

for i in range(0, len(result)):
    indice_prediction = np.argmax(result[i])
    indice_label = np.argmax(test_labels[i])
    if indice_prediction == indice_label:
        print("correct prediction for sample " + str(i))
    else:
        print("INCORRECT prediction for sample" + str(i))
```

Clean Up your Resources

Once you have completed this exercise you will have items within your subscription that will be cost your subscription [GPU compute, container instance and VM]
To delete everything – you can select your resource group containing all resources and choose to delete *[recommended to incur no charges]*

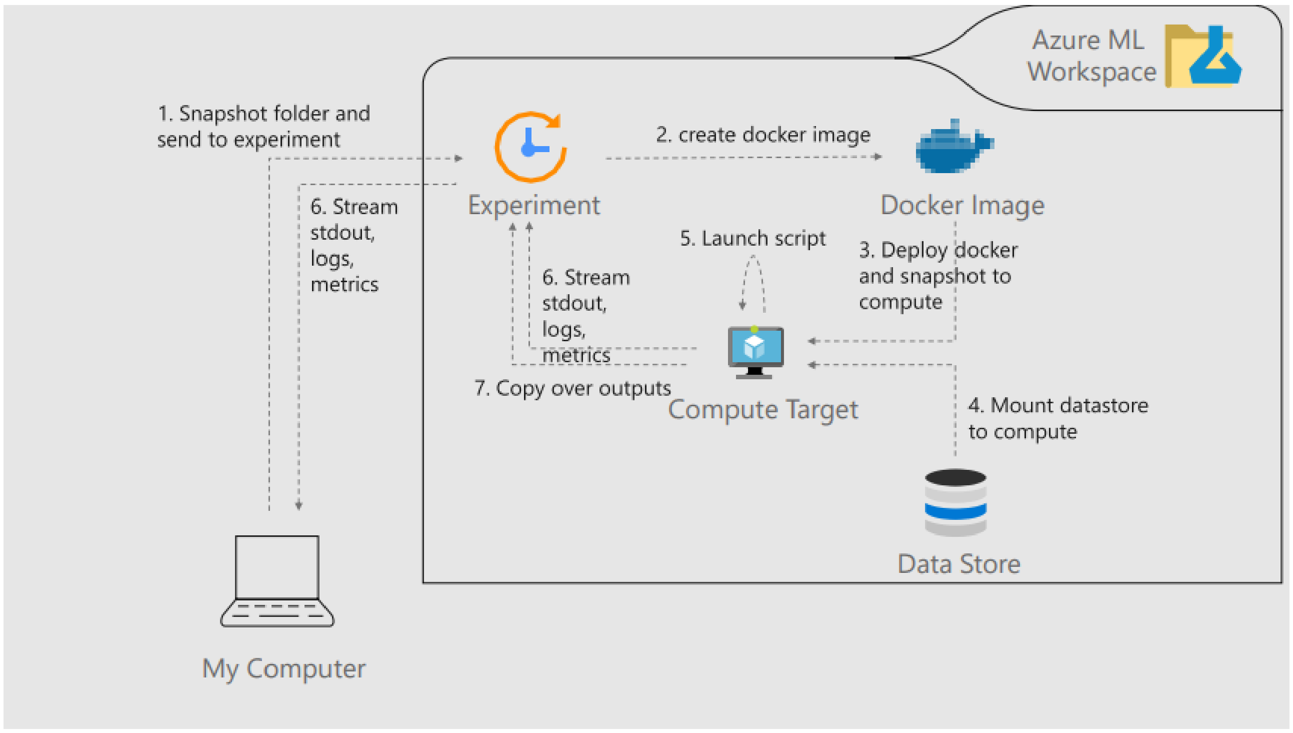
How the Azure Machine Learning service works: Architecture and concepts

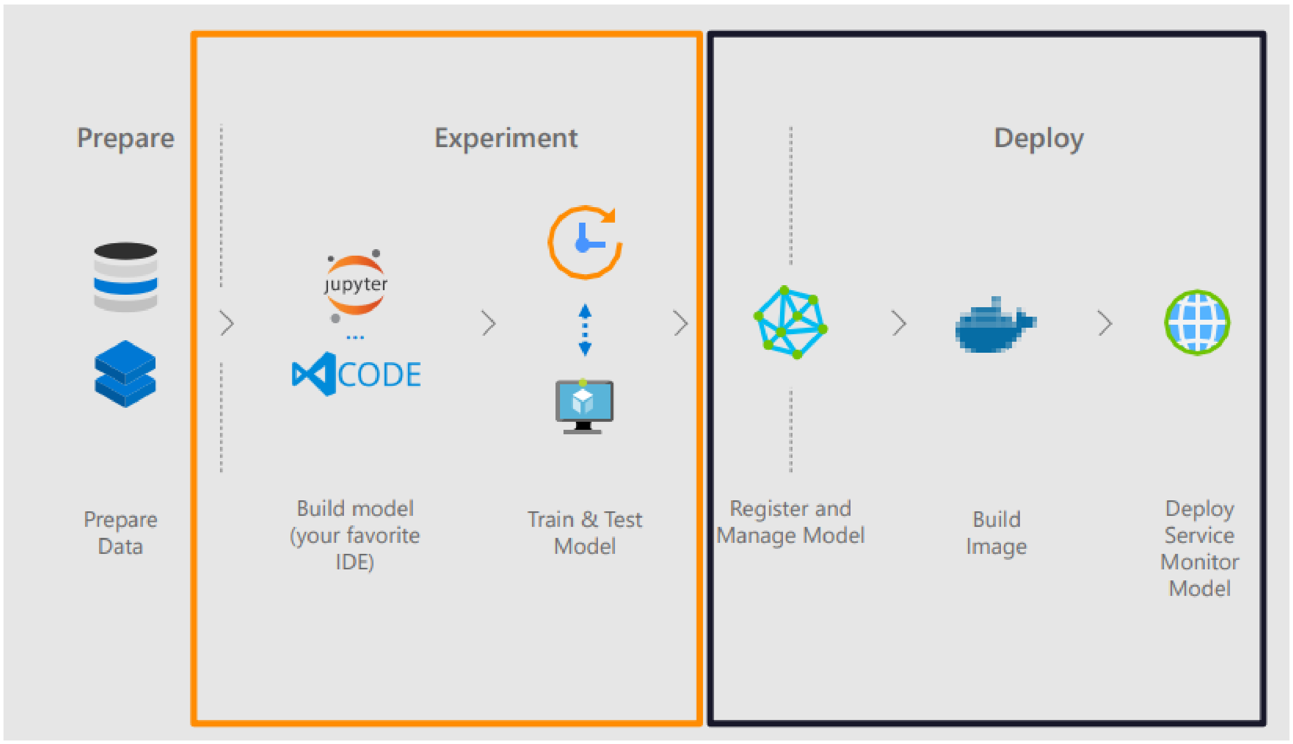
In our case, in the notebook, we create workspace using .create() function. Additionally you can create workspace via Azure portal. For the details, you could check the link.
https://docs.microsoft.com/en-gb/azure/machine-learning/service/setup-create-workspace
Use your own notebook server or cloud-based notebook server to get started with Azure Machine Learning

In the notebook, we use our own Python environment and Jupyter Notebook server to get started with Azure Machine Learning service.
https://docs.microsoft.com/en-gb/azure/machine-learning/service/quickstart-run-local-notebook
Also, you can use cloud-based notebook server to get started.
https://docs.microsoft.com/en-gb/azure/machine-learning/service/quickstart-run-cloud-notebook

Pipeline

You use machine learning pipelines to create and manage workflows that stitch together machine learning phases. For example, a pipeline might include data preparation, model training, model deployment, and inference/scoring phases. Each phase can encompass multiple steps, each of which can run unattended in various compute targets.


Azure ML Workspace
1. Snapshot folder and send to experiment
Experiment
2. create docker image
Docker Image
6. Stream stdout, logs, metrics
5. Launch script
3. Deploy docker and snapshot to compute
6. Stream stdout, logs, metrics
7. Copy over outputs
Compute Target
4. Mount datastore to compute
Data Store
My Computer


Prepare
Experiment
Deploy
jupyter
CODE
Prepare Data
Build model (your favorite IDE)
Train & Test Model
Register and Manage Model
Build Image
Deploy Service Monitor Model

Conclusion

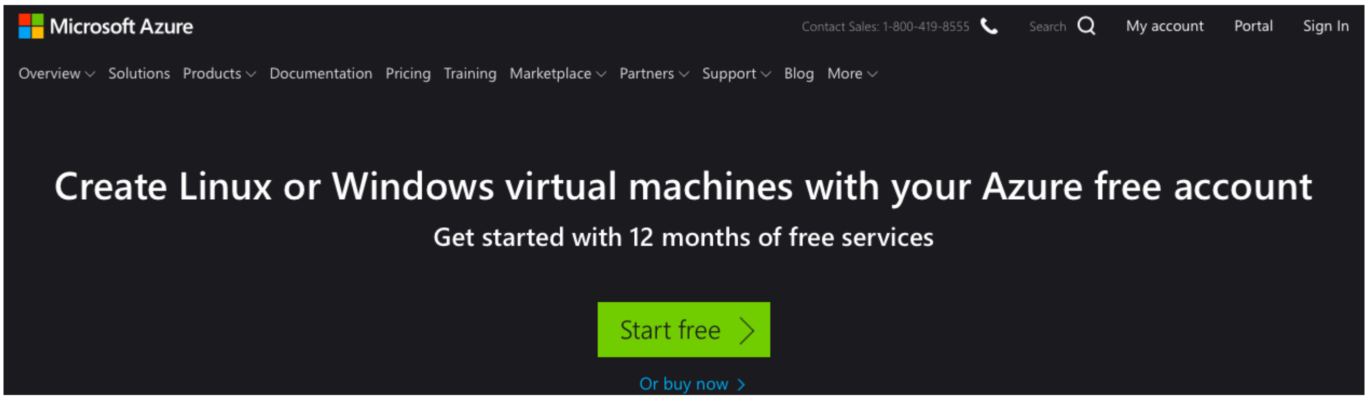
In this mini book, we have explored the idea of deploying a service into the Azure cloud. We welcome your comments in the community HERE

Appendix – setting up Azure and running a basic MNIST model

In this section, we explain the basics of running an MNIST code on Azure.

Opening an account on Azure

You need a Microsoft live account to start, it is available free of cost. Use this account to signup. Click on the **Start free** button, you will be redirected to open an account.



Depending upon the country you choose the interface will require you to submit different information. Either case you will need to provide the following details:

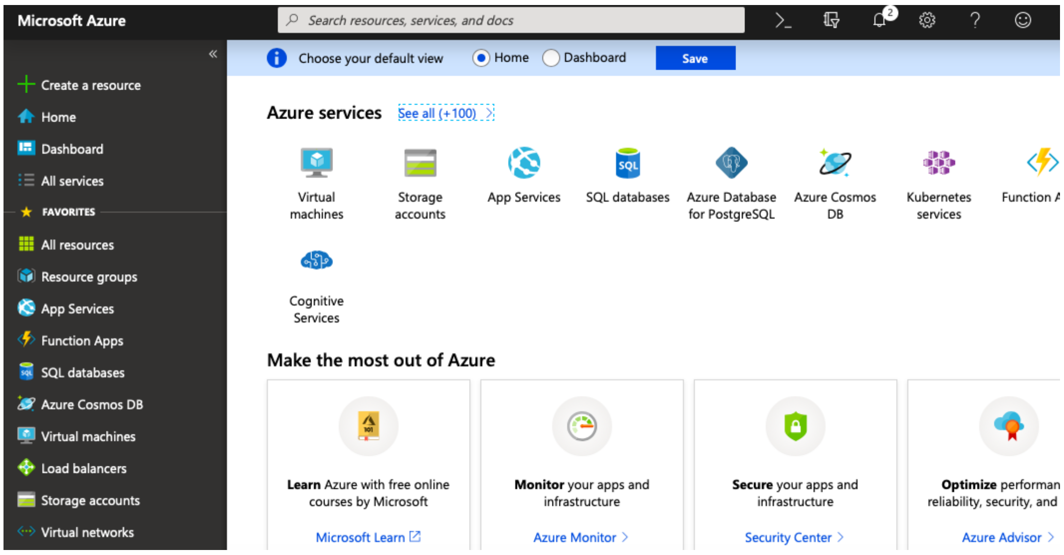
A working email,

A credit card/debit card number

A phone number

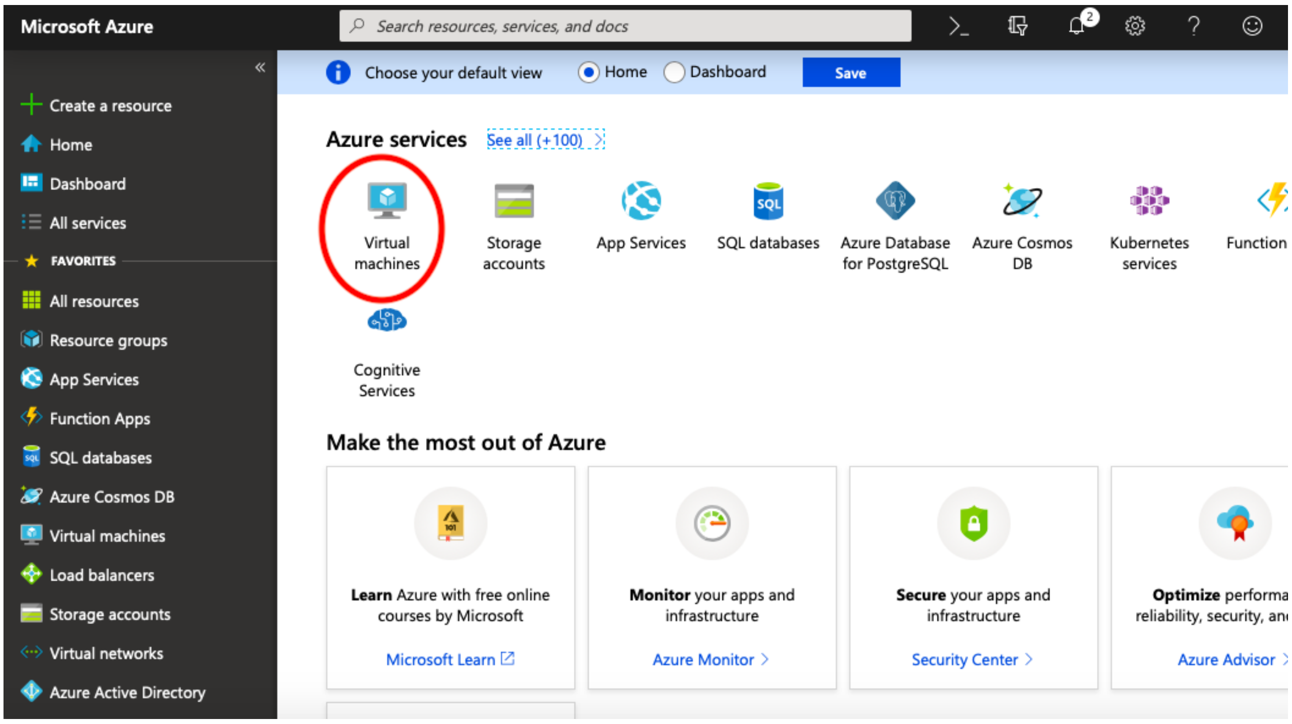
Microsoft provides $200 credit, valid for a month, plus a plethora of its free services.

After you complete the signup you reach Microsoft Azure management portal. In case this is your first time you have two options to choose from: the **free subscription** or **choose pay as you go** option. Once you choose to work on Azure on a regular basis you can choose from pay per minute to pay per months subscriptions. Below you can see the Azure Management Portal.

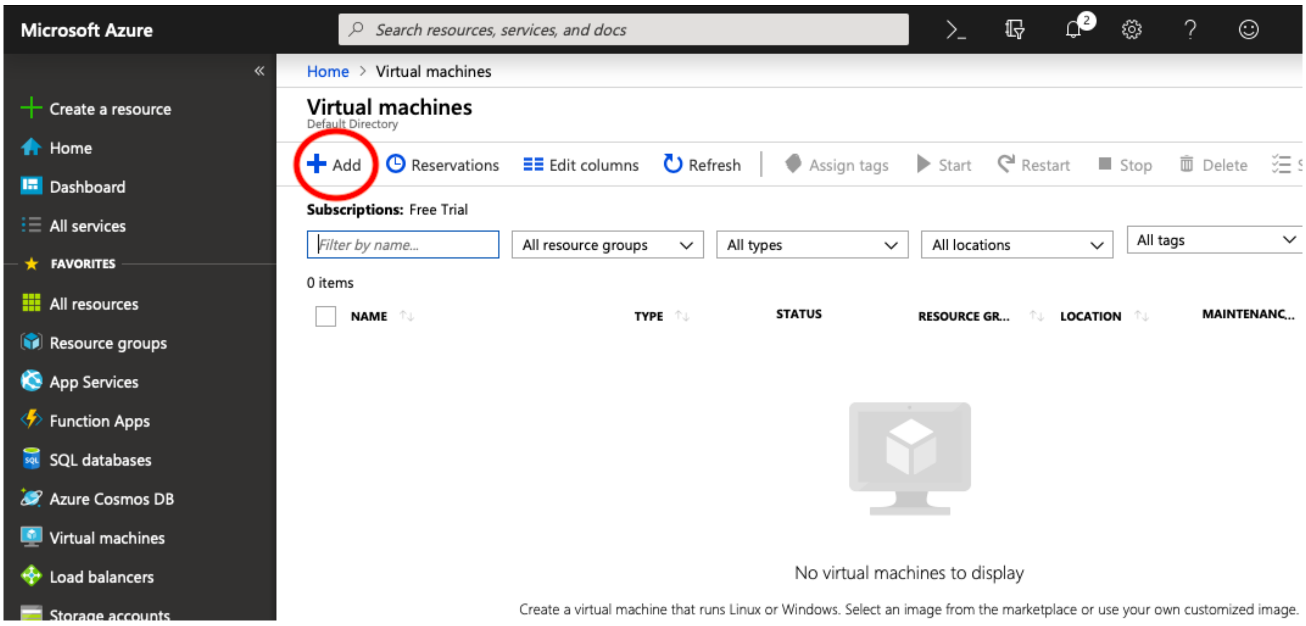


Creating a virtual machine for deep learning

Well so all is set, let us get going and create our first deep learning virtual machine (VM) on the Azure portal. **Remember for deep learning we will require GPU compute machine with the minumum configuration NC6 and it is not available in free subscription plan**. To begin let us click on **Virtual machines**.



In the next page click on the **Add** button.



Virtual machines
Default Directory

Subscriptions: Free Trial

0 items

No virtual machines to display

Create a virtual machine that runs Linux or Windows. Select an image from the marketplace or use your own customized image.

To create a virtual machine you need to:

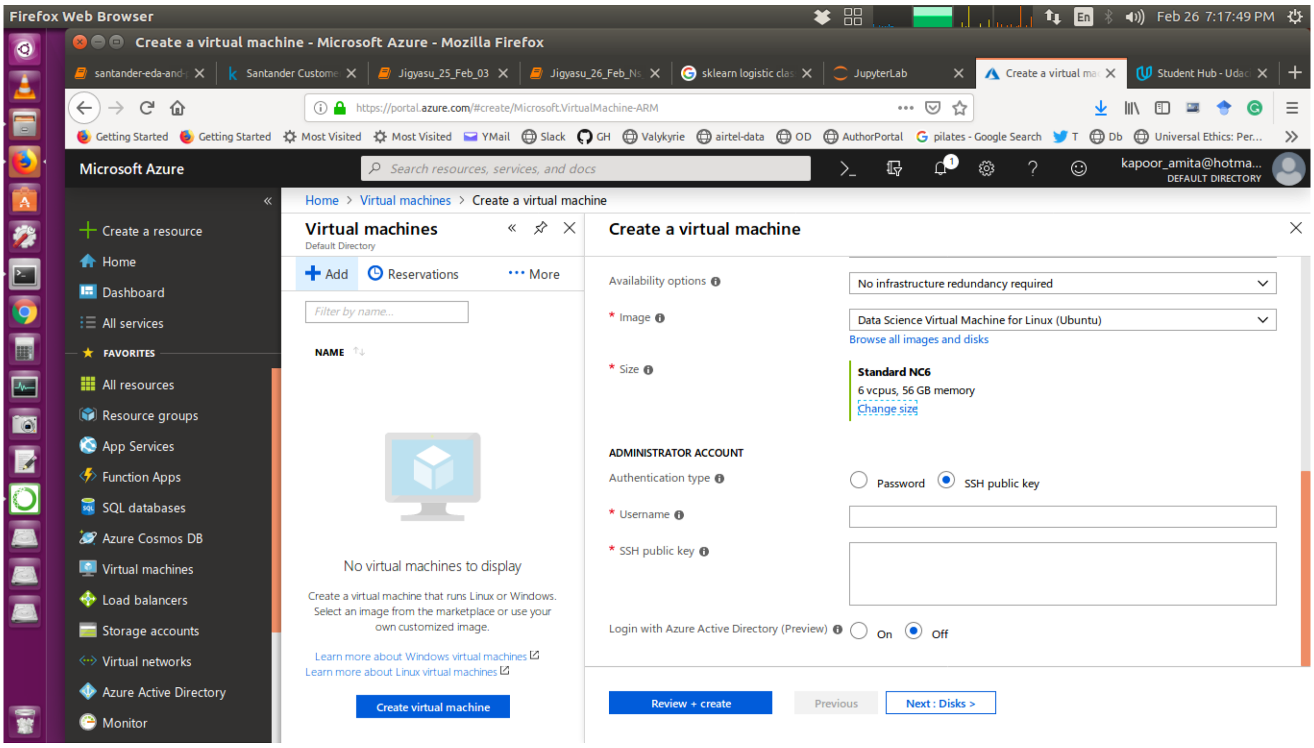
Specify the subscription plan

Specify the region you want to work in, we have selected *US East*.

Select virtual machine image. We will be choosing the [deep learning virtual machine]() image.

Confirm the compute resources you will be needing

Select proper authentication (password or ssh-key).



No virtual machines to display

Create a virtual machine that runs Linux or Windows. Select an image from the marketplace or use your own customized image.

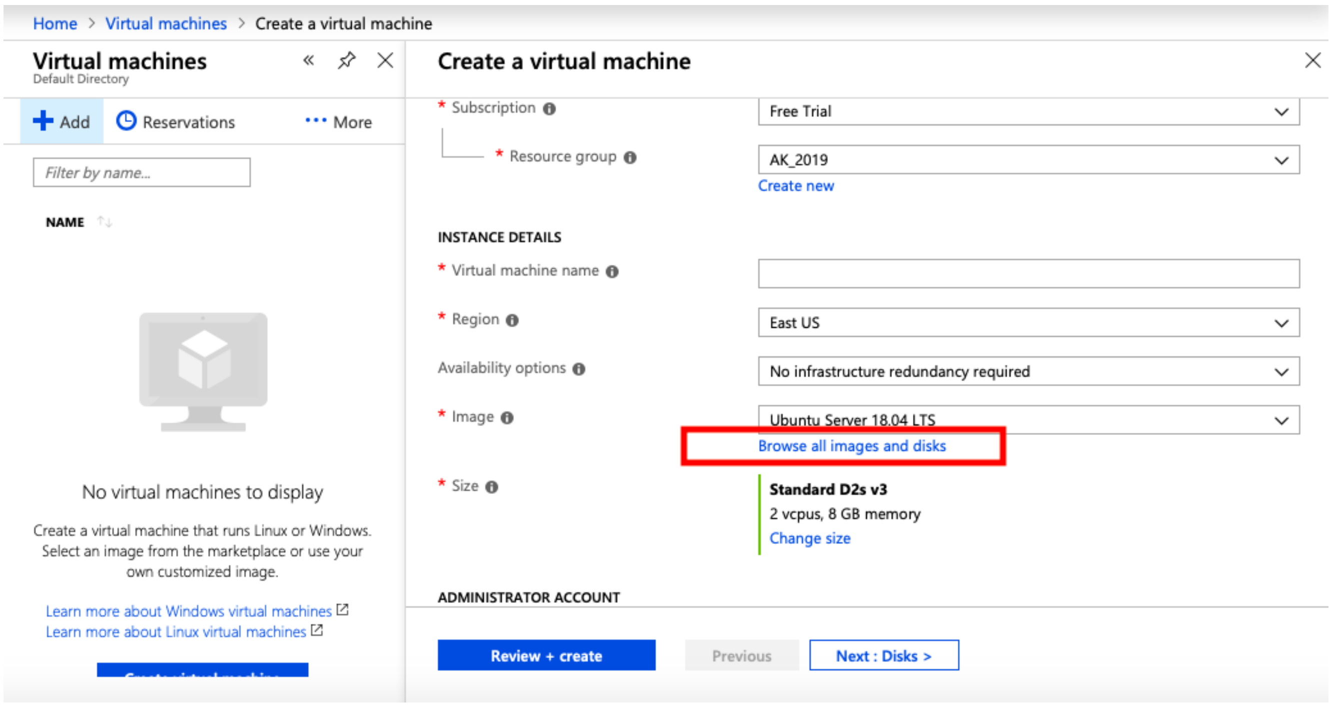
Create a virtual machine

Availability options	No infrastructure redundancy required
* Image	Data Science Virtual Machine for Linux (Ubuntu) Browse all images and disks
* Size	Standard NC6 6 vcpus, 56 GB memory Change size

ADMINISTRATOR ACCOUNT

Authentication type	Password [✓ SSH public key]
* Username	
* SSH public key	
Login with Azure Active Directory (Preview)	On [✓ Off]

Since we are interested in creating a deep learning virtual machine, we start with selecting the image first, click "Browse all images and disks".


Home > Virtual machines > Create a virtual machine

Virtual machines
Default Directory

Add | Reservations | More

Filter by name...

NAME

No virtual machines to display

Create a virtual machine that runs Linux or Windows. Select an image from the marketplace or use your own customized image.

Learn more about Windows virtual machines
Learn more about Linux virtual machines

Create a virtual machine

* Subscription	Free Trial
* Resource group	AK_2019
	Create new

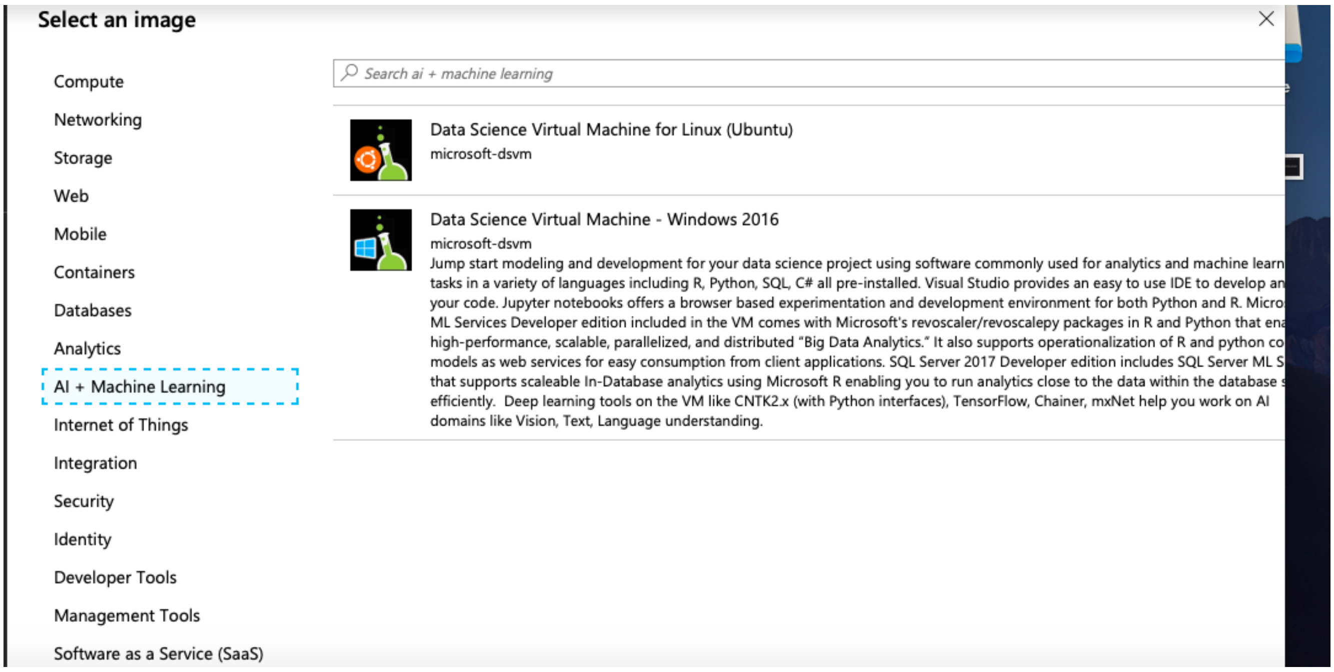
INSTANCE DETAILS

* Virtual machine name	
* Region	East US
Availability options	No infrastructure redundancy required
* Image	Ubuntu Server 18.04 LTS
	Browse all images and disks
* Size	**Standard D2s v3** 2 vcpus, 8 GB memory Change size

ADMINISTRATOR ACCOUNT

Review + create | Previous | Next : Disks >

On the next page select "AI + Machine Learning".


Select an image

Compute
Networking
Storage
Web
Mobile
Containers
Databases
Analytics
AI + Machine Learning
Internet of Things
Integration
Security
Identity
Developer Tools
Management Tools
Software as a Service (SaaS)

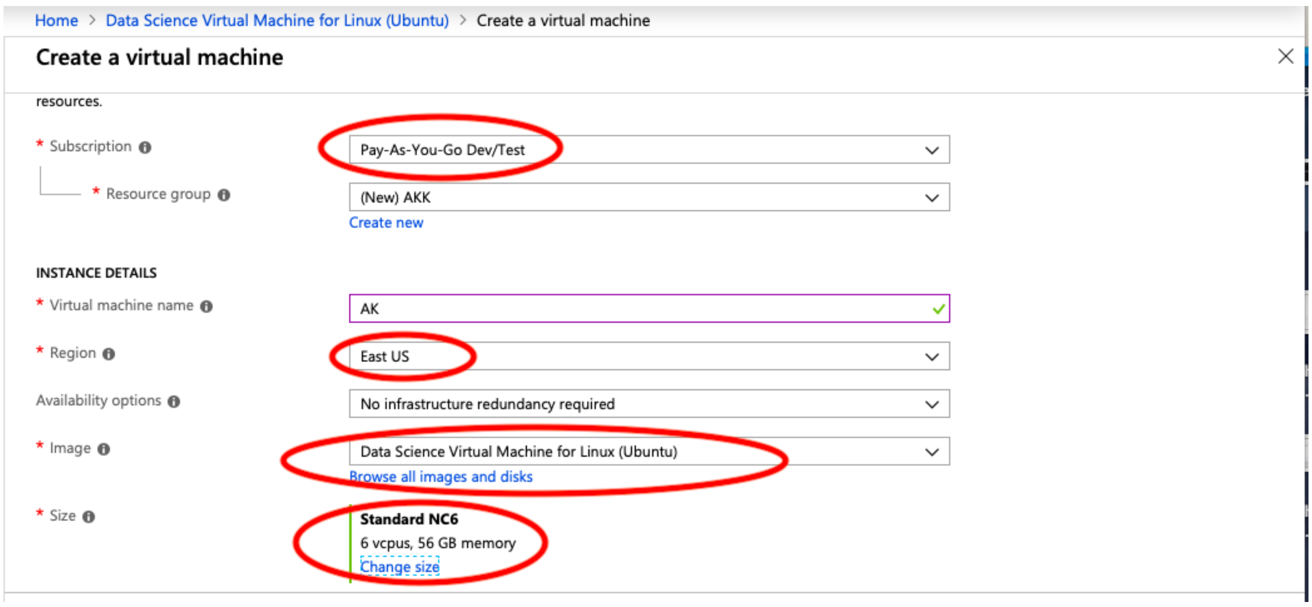
Search ai + machine learning

Data Science Virtual Machine for Linux (Ubuntu)
microsoft-dsvm

Data Science Virtual Machine - Windows 2016
microsoft-dsvm
Jump start modeling and development for your data science project using software commonly used for analytics and machine learn
tasks in a variety of languages including R, Python, SQL, C# all pre-installed. Visual Studio provides an easy to use IDE to develop an
your code. Jupyter notebooks offers a browser based experimentation and development environment for both Python and R. Micro
ML Services Developer edition included in the VM comes with Microsoft's revoscaler/revoscalepy packages in R and Python that ena
high-performance, scalable, parallelized, and distributed "Big Data Analytics." It also supports operationalization of R and python co
models as web services for easy consumption from client applications. SQL Server 2017 Developer edition includes SQL Server ML S
that supports scaleable In-Database analytics using Microsoft R enabling you to run analytics close to the data within the database
efficiently. Deep learning tools on the VM like CNTK2.x (with Python interfaces), TensorFlow, Chainer, mxNet help you work on AI
domains like Vision, Text, Language understanding.

The Azure provides two options a Ubuntu machine and a Windows machine. We will be selecting the Ubuntu one here. You can try windows version too. Once you have confirmed the image, you can click on **change size** option below it and choose the basic GPU machine **NC6**.

Once all is done verify with the selections encircled in the image below:


Home > Data Science Virtual Machine for Linux (Ubuntu) > Create a virtual machine

Create a virtual machine

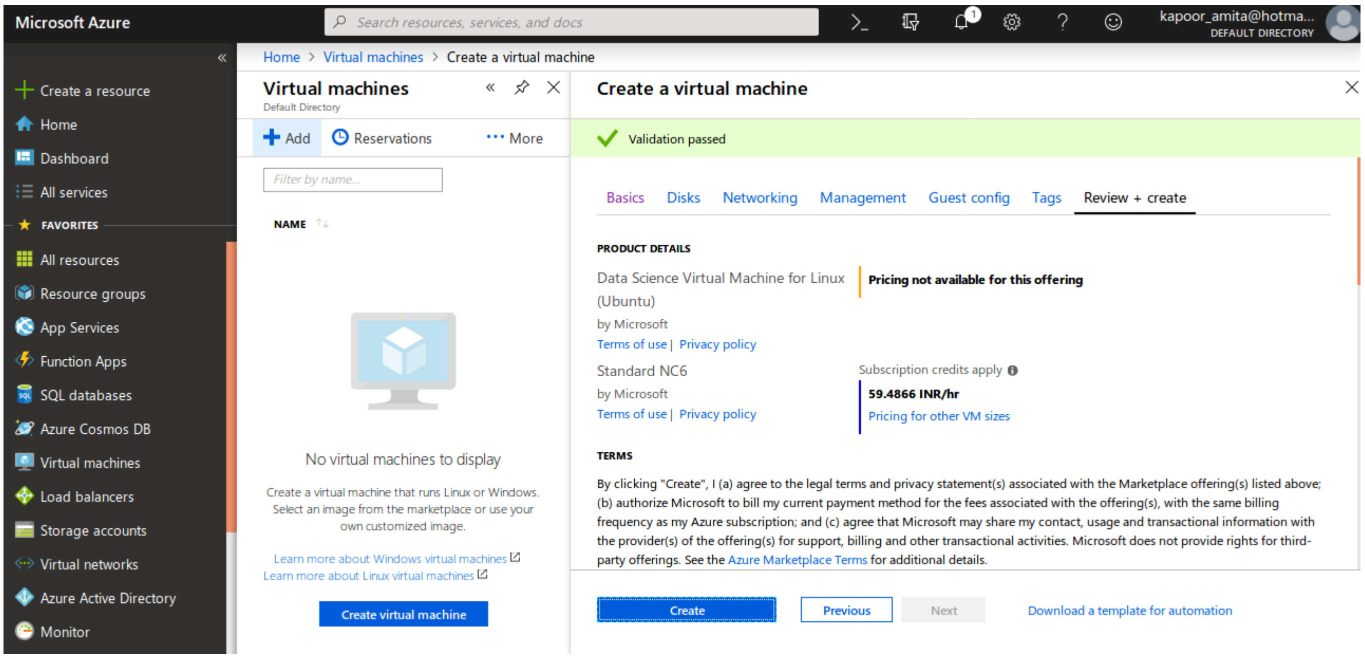
resources.

* Subscription	Pay-As-You-Go Dev/Test
* Resource group	(New) AKK Create new

INSTANCE DETAILS

* Virtual machine name	AK
* Region	East US
Availability options	No infrastructure redundancy required
* Image	Data Science Virtual Machine for Linux (Ubuntu) Browse all images and disks
* Size	Standard NC6 6 vcpus, 56 GB memory Change size

Also observe that subscription plan is no longer **Free Trial**. This is so because the **NC6** cloud machine is not available in free trial. Once you are satisfied choose the authentication type and click on the blue colored **Review + Create** button. Observe we have left everything else default. Once you click **Review + Create** button the Azure will check if the components you have selected is within your subscription plan and resource quota. If all goes well you will see a page like this:



Create a virtual machine

✓ Validation passed

Basics | Disks | Networking | Management | Guest config | Tags | Review + create

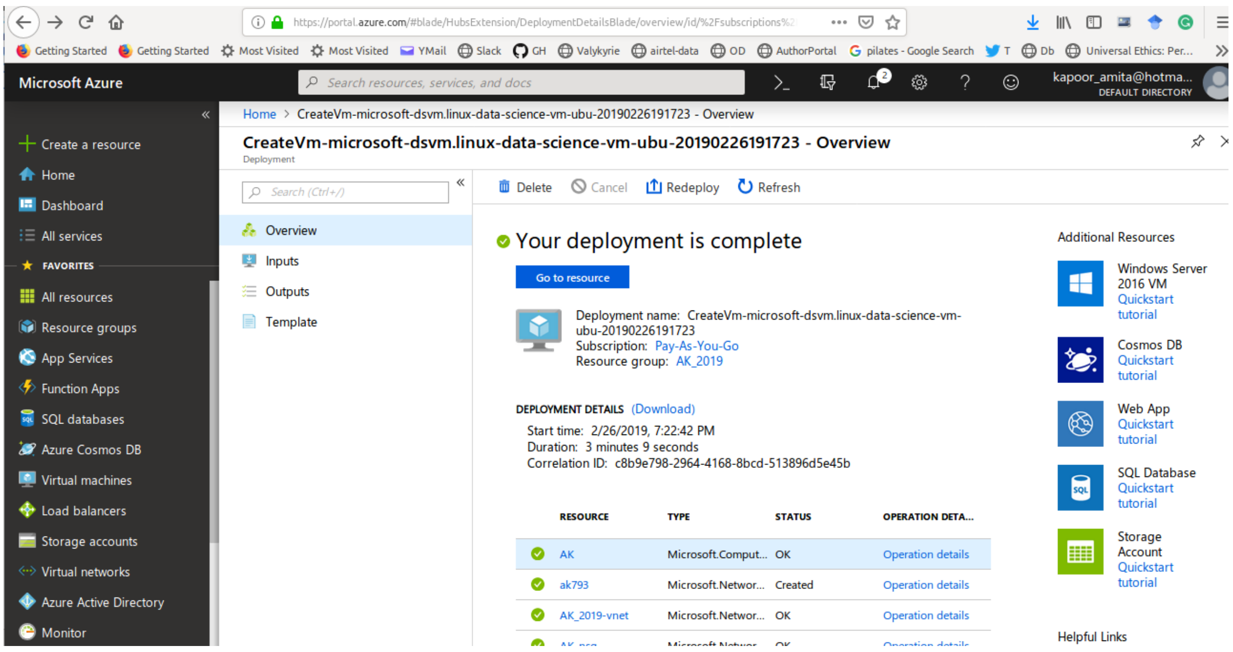
PRODUCT DETAILS

Data Science Virtual Machine for Linux (Ubuntu) by Microsoft Terms of use | Privacy policy	**Pricing not available for this offering**
Standard NC6 by Microsoft Terms of use | Privacy policy	Subscription credits apply **59.4866 INR/hr** Pricing for other VM sizes

TERMS

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. See the Azure Marketplace Terms for additional details.

The page contains a summary of your machine configuration and pricing. The moment you press create, Azure will start deploying the machine and you will be charged. After deployment you will see on the portal:



Home > CreateVm-microsoft-dsvm.linux-data-science-vm-ubu-20190226191723 - Overview

CreateVm-microsoft-dsvm.linux-data-science-vm-ubu-20190226191723 - Overview

Deployment

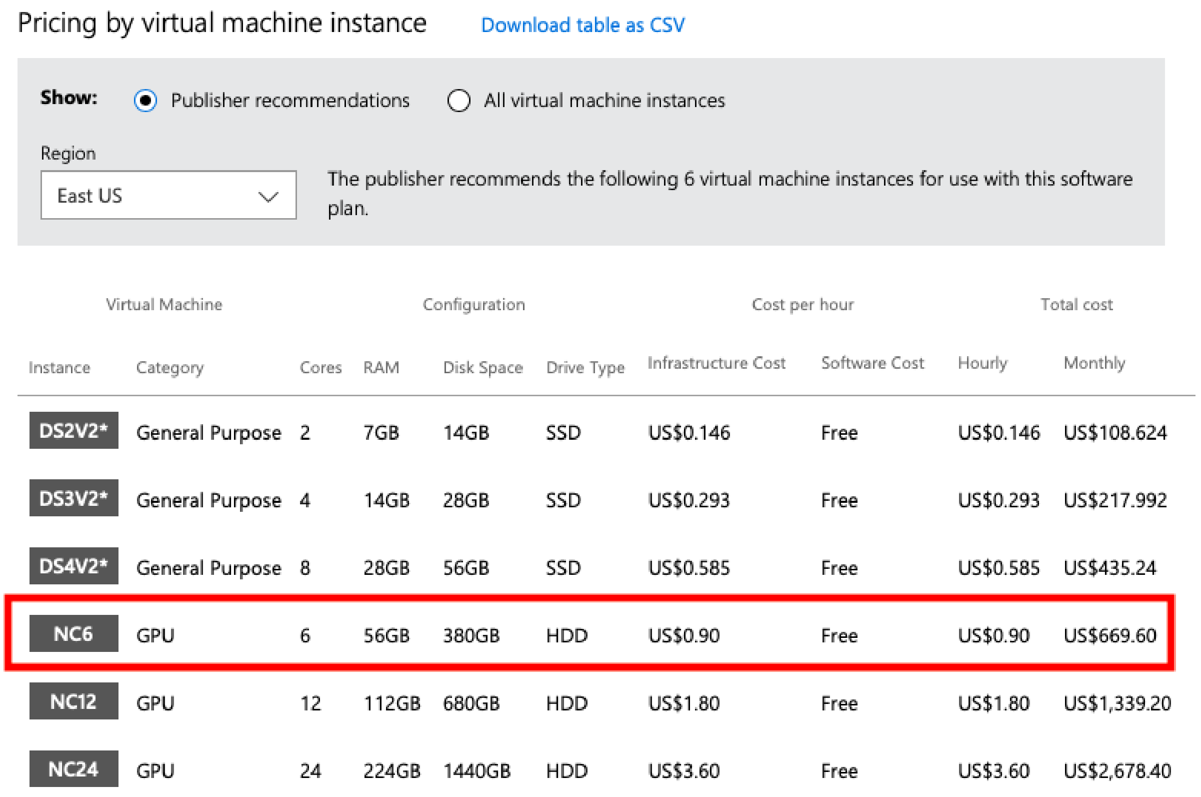
Your deployment is complete

Deployment name: CreateVm-microsoft-dsvm.linux-data-science-vm-ubu-20190226191723
Subscription: Pay-As-You-Go
Resource group: AK_2019

DEPLOYMENT DETAILS (Download)
Start time: 2/26/2019, 7:22:42 PM
Duration: 3 minutes 9 seconds
Correlation ID: c8b9e798-2964-4168-8bcd-513896d5e45b

RESOURCE	TYPE	STATUS	OPERATION DETA...
AK	Microsoft.Comput...	OK	Operation details
ak793	Microsoft.Networ...	Created	Operation details
AK_2019-vnet	Microsoft.Networ...	OK	Operation details

The net expense is roughly $0.90 per hour. An estimate of price for different GPU enabled machines on Azure is shown below.



Pricing by virtual machine instance   Download table as CSV

Show: [✓ Publisher recommendations] All virtual machine instances

Region: East US

The publisher recommends the following 6 virtual machine instances for use with this software plan.

Virtual Machine		Configuration				Cost per hour		Total cost	
Instance	Category	Cores	RAM	Disk Space	Drive Type	Infrastructure Cost	Software Cost	Hourly	Monthly
DS2V2*	General Purpose	2	7GB	14GB	SSD	US$0.146	Free	US$0.146	US$108.624
DS3V2*	General Purpose	4	14GB	28GB	SSD	US$0.293	Free	US$0.293	US$217.992
DS4V2*	General Purpose	8	28GB	56GB	SSD	US$0.585	Free	US$0.585	US$435.24
NC6	GPU	6	56GB	380GB	HDD	US$0.90	Free	US$0.90	US$669.60
NC12	GPU	12	112GB	680GB	HDD	US$1.80	Free	US$1.80	US$1,339.20
NC24	GPU	24	224GB	1440GB	HDD	US$3.60	Free	US$3.60	US$2,678.40

Connecting and terminating the VM

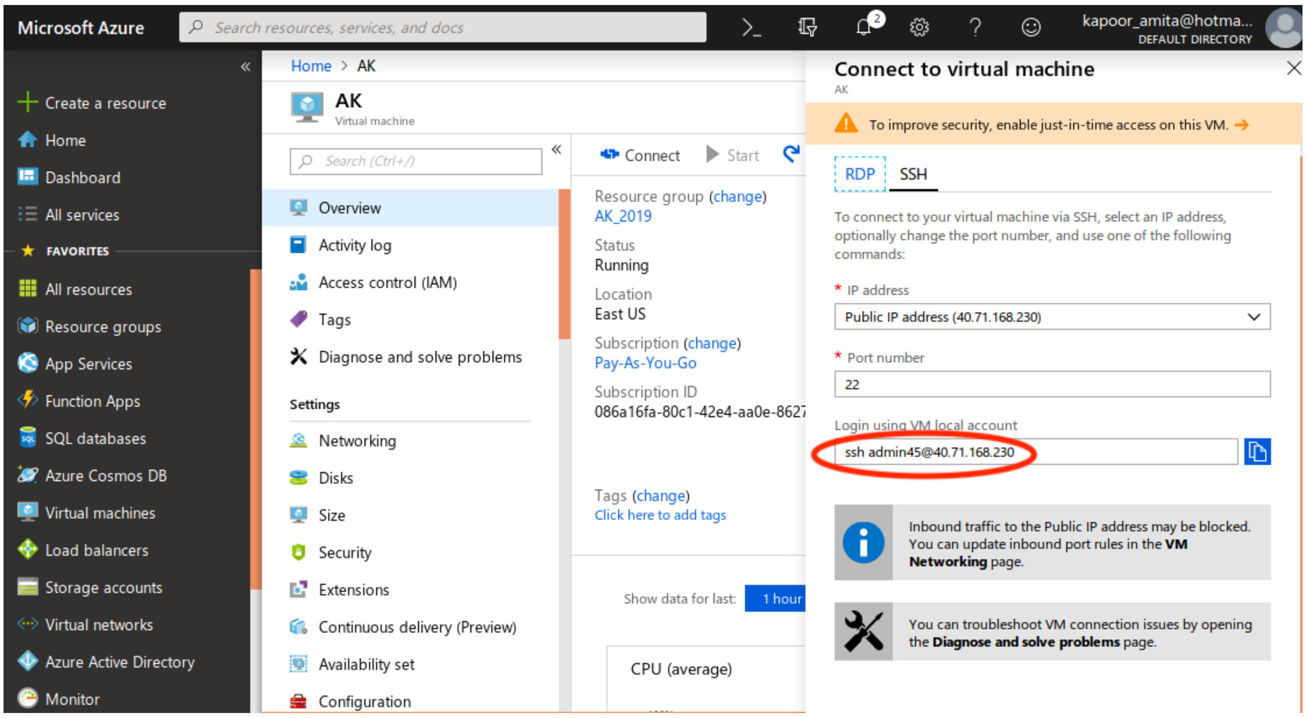
To connect to the VM we have two options, depending upon the authentication you had selected, if you click on **Connect** both options are visible to you:

You can use the username and password to login to the remote VM.

Alternatively depending upon your OS, you can ssh to the VM machine using either terminal or PuTTy. To do this from your terminal type:

ssh -i path_to_your_ssh_key -L 8888:127.0.0.1:8888 myserverpaddress

here `myserverpaddress` is the IP of your launched instance. This information is available on the portal.



Connect to virtual machine

AK

To improve security, enable just-in-time access on this VM.

RDP SSH

To connect to your virtual machine via SSH, select an IP address, optionally change the port number, and use one of the following commands:

* IP address: Public IP address (40.71.168.230)

* Port number: 22

Login using VM local account: ssh admin45@40.71.168.230

Inbound traffic to the Public IP address may be blocked. You can update inbound port rules in the **VM Networking** page.

You can troubleshoot VM connection issues by opening the **Diagnose and solve problems** page.

Resource group (change): AK_2019

Status: Running

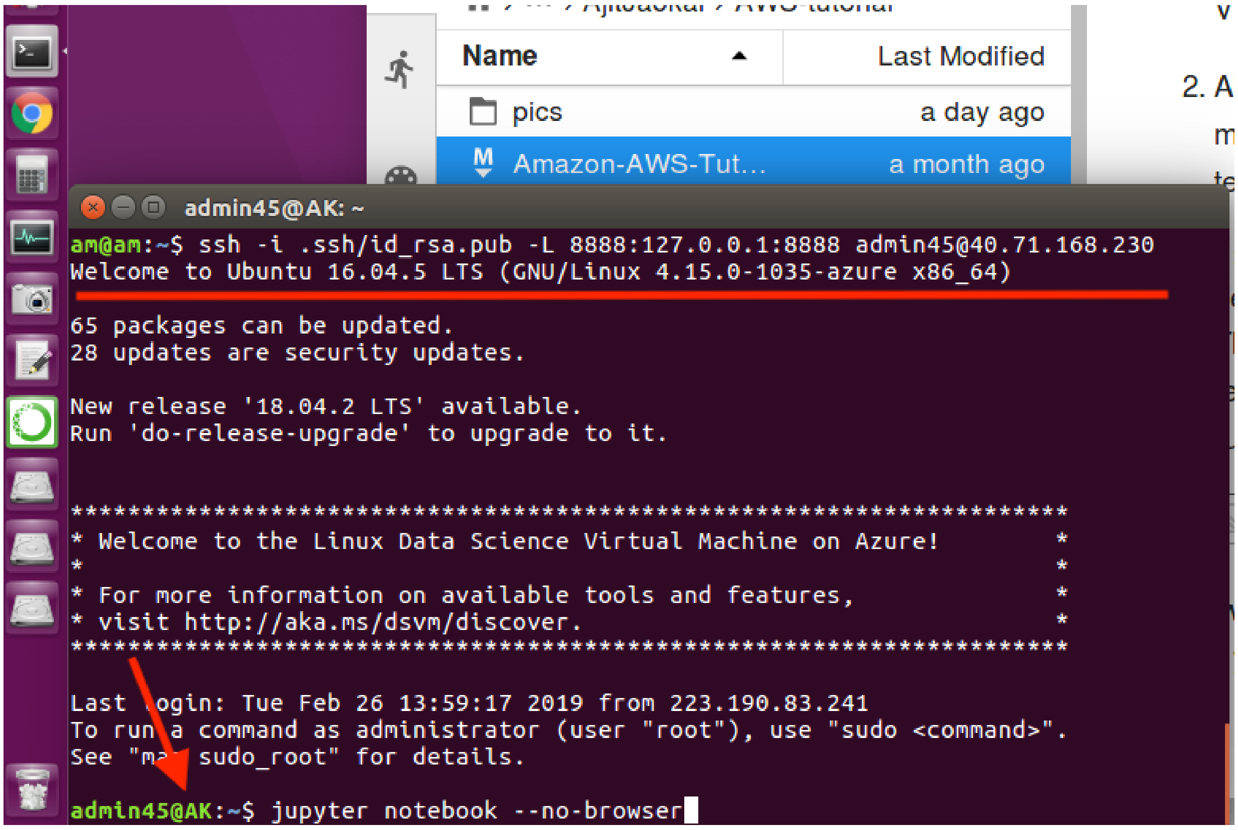
Location: East US

Subscription (change): Pay-As-You-Go

Subscription ID: 086a16fa-80c1-42e4-aa0e-8627

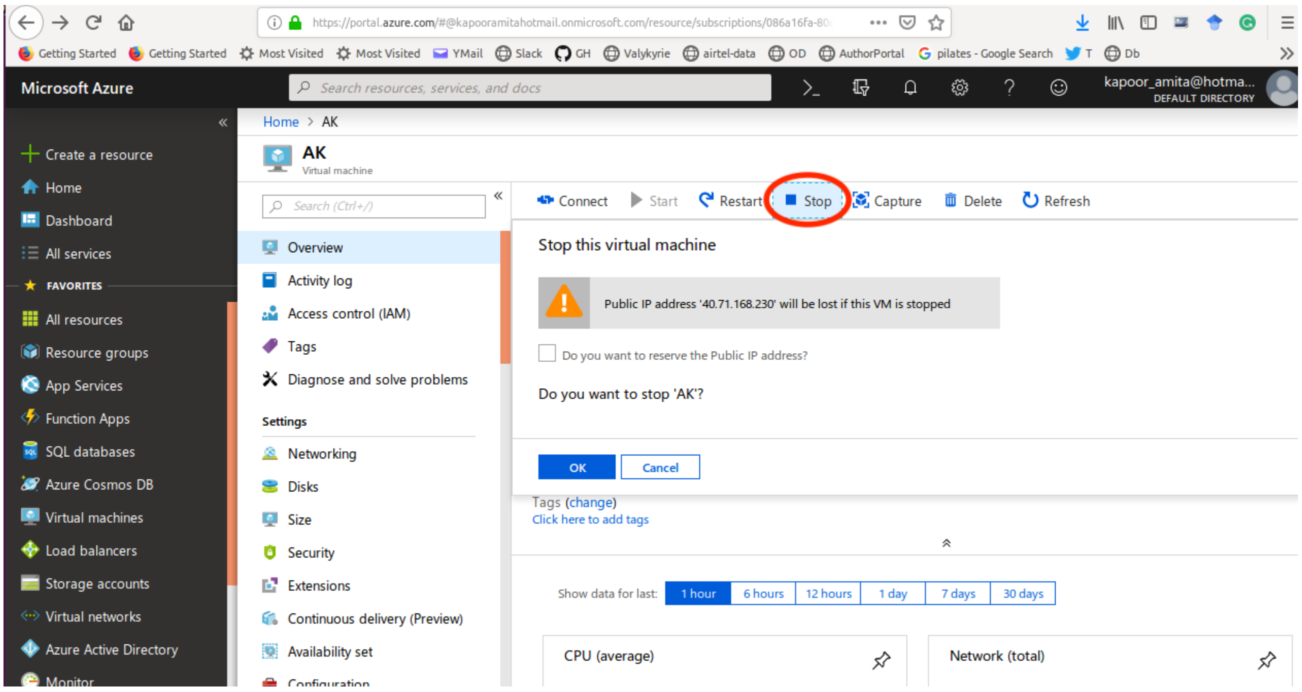
Tags (change): Click here to add tags

Below you can see the terminal on your local machine and the commands you need to run. Observe how the **prompt** changed after login.



```
am@am:~$ ssh -i .ssh/id_rsa.pub -L 8888:127.0.0.1:8888 admin45@40.71.168.230
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.15.0-1035-azure x86_64)

65 packages can be updated.
28 updates are security updates.

New release '18.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


**********************************************************************
* Welcome to the Linux Data Science Virtual Machine on Azure!        *
*                                                                    *
* For more information on available tools and features,             *
* visit http://aka.ms/dsvm/discover.                                 *
**********************************************************************

Last login: Tue Feb 26 13:59:17 2019 from 223.190.83.241
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

admin45@AK:~$ jupyter notebook --no-browser
```

Now we are connected and have terminal access to the Azure VM. Once your work is finish, do not forget to close the machine. And in case you do not need it again, it is safe to delete the machine along with all the resources.



Stop this virtual machine

Public IP address '40.71.168.230' will be lost if this VM is stopped

☐ Do you want to reserve the Public IP address?

Do you want to stop 'AK'?

Launching Jupyter Notebook and training the model

The deep learning image that we are using comes already loaded with Jupyter Notebook, so you are ready to go. Once you are connected to the VM to start Jupyter on the VM terminal run the command:

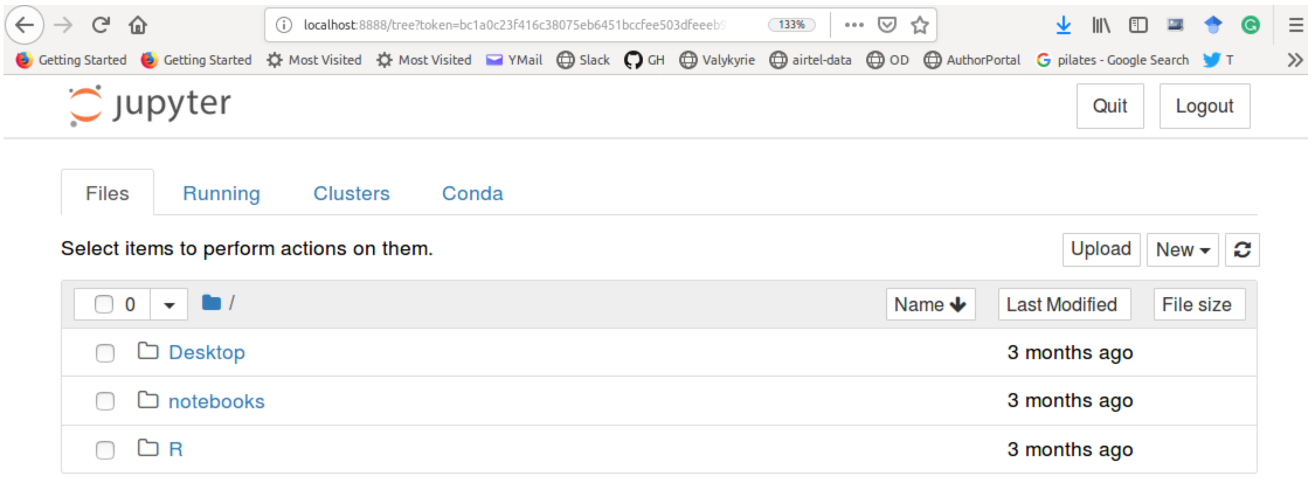
jupyter notebook --no-browser

You will see a message telling the notebook started and http address required to see it.

Copy the complete adddress including the complete token as shown below:

Copy the URL token from the command line and enter in your browser to access the jupyter notebook.

You are now ready to write your code.


localhost:8888/tree?token=bc1a0c23f416c38075eb6451bccfee503dfeeeb

Select items to perform actions on them.

Name	Last Modified	File size
Desktop	3 months ago	
notebooks	3 months ago	
R	3 months ago	

MNIST on Azure

Let us train a CNN to detect handwritten digits on the created VM.

First connect to the VM by ssh

On the Azure VM:

On the terminal type:

wget --no-check-certificate --content-disposition https://raw.githubusercontent.com/amita-kapoor/Tutorials-DL-SE/master/DeepLearning/MNISTSoftmax.ipynb

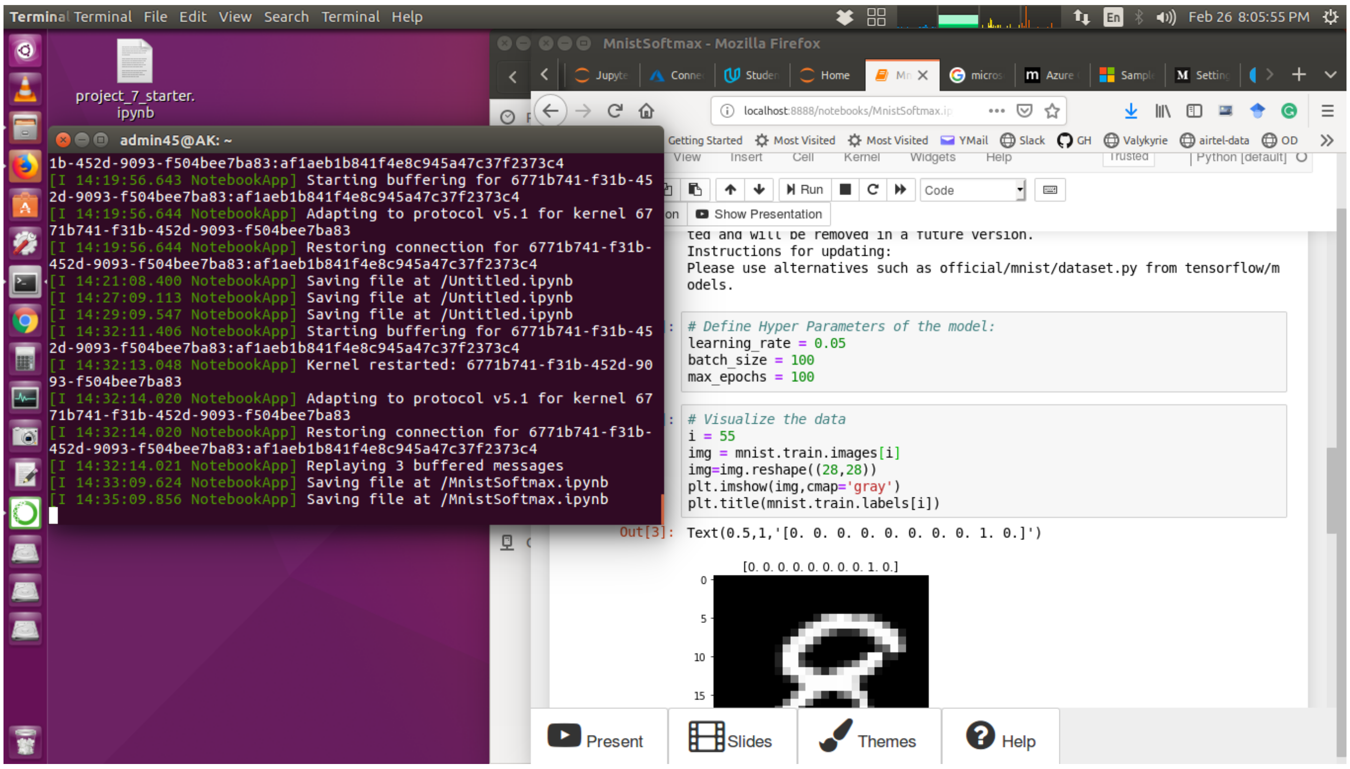
Start Jupyter Notebook: jupyter notebook --no-browser

You will need the token and the URL generated by the jupyter notebook to access it. On your instance terminal, there will be the following line: ”Copy/paste this URL into your browser when you connect for the first time, to login with a token:". Copy everything starting with the http://localhost:8888/?token=.

On your local machine:

Access the Jupyter notebook index from your web browser by pasting the complete address you copied above.

Click on the ipynb file: MNIST Softmax.ipynb.



```
1b-452d-9093-f504bee7ba83:af1aeb1b841f4e8c945a47c37f2373c4
[I 14:19:56.643 NotebookApp] Starting buffering for 6771b741-f31b-45
2d-9093-f504bee7ba83:af1aeb1b841f4e8c945a47c37f2373c4
[I 14:19:56.644 NotebookApp] Adapting to protocol v5.1 for kernel 67
71b741-f31b-452d-9093-f504bee7ba83
[I 14:19:56.644 NotebookApp] Restoring connection for 6771b741-f31b-
452d-9093-f504bee7ba83:af1aeb1b841f4e8c945a47c37f2373c4
[I 14:21:08.400 NotebookApp] Saving file at /Untitled.ipynb
[I 14:27:09.113 NotebookApp] Saving file at /Untitled.ipynb
[I 14:29:09.547 NotebookApp] Saving file at /Untitled.ipynb
[I 14:32:11.406 NotebookApp] Starting buffering for 6771b741-f31b-45
2d-9093-f504bee7ba83:af1aeb1b841f4e8c945a47c37f2373c4
[I 14:32:13.048 NotebookApp] Kernel restarted: 6771b741-f31b-452d-90
93-f504bee7ba83
[I 14:32:14.020 NotebookApp] Adapting to protocol v5.1 for kernel 67
71b741-f31b-452d-9093-f504bee7ba83
[I 14:32:14.020 NotebookApp] Restoring connection for 6771b741-f31b-
452d-9093-f504bee7ba83:af1aeb1b841f4e8c945a47c37f2373c4
[I 14:32:14.021 NotebookApp] Replaying 3 buffered messages
[I 14:33:09.624 NotebookApp] Saving file at /MnistSoftmax.ipynb
[I 14:35:09.856 NotebookApp] Saving file at /MnistSoftmax.ipynb
```

```
ted and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/m
odels.
```

```python
# Define Hyper Parameters of the model:
learning_rate = 0.05
batch_size = 100
max_epochs = 100
```

```python
# Visualize the data
i = 55
img = mnist.train.images[i]
img=img.reshape((28,28))
plt.imshow(img,cmap='gray')
plt.title(mnist.train.labels[i])
```

```
Out[3]: Text(0.5,1,'[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]')
```

3. Run each cell in the notebook 4. The code trains a simple MLP network on MNIST dataset.

Finally do not forget to Terminate/Stop your instance. Have fun training your models.