**A Developers Perspective: Building and deploying machine learning models using Microsoft Azure Machine Learning Workbench with containers on a Kubernetes cluster**

Contents

[Synopsis 3](#_Toc504060774)

[Developer to Developer 3](#_Toc504060775)

[Data Science – It’s not magic! 4](#_Toc504060776)

[Technologies Utilized 5](#_Toc504060777)

[Azure Machine Learning Workbench 5](#_Toc504060778)

[Programming Languages 5](#_Toc504060779)

[Azure and Other Services 6](#_Toc504060780)

[Building a Machine Learning model for a Factory 6](#_Toc504060781)

[Preparing Development Environment 7](#_Toc504060782)

[Azure Resource Group 7](#_Toc504060783)

[Machine Learning accounts 7](#_Toc504060784)

[Windows Data Science Virtual Machine(DSVM) 8](#_Toc504060785)

[Linux Data Science Virtual Machine (DSVM) 8](#_Toc504060786)

[Azure Storage Account 8](#_Toc504060787)

[Azure Machine Learning Workbench 8](#_Toc504060788)

[Copy Project files to Windows DSVM 8](#_Toc504060789)

[Configuring Azure Machine Learning Workbench 8](#_Toc504060790)

[Creating the first project 9](#_Toc504060791)

[Prepare the project source files 9](#_Toc504060792)

[Prepare dependencies 9](#_Toc504060793)

[Prepare the project to use Ubuntu Docker 10](#_Toc504060794)

[Working with the data 11](#_Toc504060795)

[Importing Data 11](#_Toc504060796)

[Splitting Data and Building a Model 12](#_Toc504060797)

[Creating a schema 13](#_Toc504060798)

[Scoring 13](#_Toc504060799)

[Deployment 14](#_Toc504060800)

[Consuming the Model 17](#_Toc504060801)

[Summary 17](#_Toc504060802)

[Appendix A: Azure CLI 19](#_Toc504060803)

[Azure Login 19](#_Toc504060804)

[AML Workbench : Configure Remote Docker Host 19](#_Toc504060805)

[Create Kubernetes Cluster and associate with Workbench Experiment 19](#_Toc504060806)

[Set the Model Management Account for the Experiment 20](#_Toc504060807)

[Create a Service within Model Management 20](#_Toc504060808)

[Multiple Commands 21](#_Toc504060809)

[Single Command 21](#_Toc504060810)

[Get Deployed Service Information 21](#_Toc504060811)

[Appendix B: Kubernetes kubectl.exe 21](#_Toc504060812)

[Prepare /.kube/config file 21](#_Toc504060813)

[Setting the context in kubectl 22](#_Toc504060814)

[Run the k8s Dashboard 22](#_Toc504060815)

# Synopsis

As machine learning becomes more ubiquitous it is imperative that developers understand the basics of data science and how it can enhance applications and services. Traditional software development doesn’t become a relic, far from it. Utilizing machine learning in conjunction with traditional software development offers a unique and powerful way create applications and services with embedded but remote intelligence.

With the ability to programmatically update that remote intelligence, well written applications reduce the need to update as core algorithms are improved.

To be sure, data science can seem daunting to a developer who has not been exposed to it. It is viewed by many inside and outside of engineering organizations as some sort of magic. That misconception creates anxiety, or worse, resistance to understanding it.

Many of the examples, demos and tutorials online that introduce users to data science concepts can be incredibly complex, or conversely so automated that it’s difficult to understand what is happening and why.

This goal of this paper is to explain basic concepts of data science by working through a simple but complete example of identifying a need, creating a machine learning model, and deploying it for consumption in a production environment.

# Developer to Developer

Machine learning has many applications in the real world. Predictive maintenance, churn detection, product recommendations, filtering, optical recognition, and many more.

The power of machine learning comes partially from the intelligence that can be applied to data in instances where traditional rules based software is difficult or infeasible. The power also comes from the ability to offload those complex computing tasks from the applications or service code.

Many enterprise grade solutions implement machine learning as a REST API residing on either an edge server or cloud service that hosts the machine learning model as a service.

Machine learning deployments can be managed by a cloud service provider that automatically scales up and down the service such as Azure ML Studio (<https://studio.azureml.net/>) . For more complex models, they can be hosted behind a function app or compiled into a Docker images that can be published into a cluster.

In the end, as a developer the result is that machine learning becomes just another API to be called in the course of the other logic contained in an application or service.

With the intelligence deployed to the cloud or an edge server, all sorts of products can take advantage be it dashboards, applications, and local or cloud services. By deploying it behind a REST endpoint, the intelligence can be modified or upgraded, also known as re-training, in place making the code more intelligent over time without the need to upgrade the original source.

By reading and/or following along with this paper you will get a better understanding of data science, machine learning and consumption of the machine learning models. The goal is to present the information in layman’s terms to make AI and ML more accessible to everyone.

# Data Science – It’s not magic!

Data science is not magic. It is the orchestration of talent that can identify a question, acquire data that supports the question, and implement the correct algorithms across that data to answer a question.

So, what is this question? To start down a path of using data science and machine learning we need to form a sharp question. A sharp question can be answered with a name or a number such as “which one of the vehicles in the fleet will fail first?”, or “how many miles per gallon will I get today?”. Vague questions, or ones that can’t be answered with a name or a number should never be used. Vague questions lead you down a path with unattainable results. An example of a vague question would be “how can I improve my profits?”.

So we identify a sharp question, now what? The next step is data acquisition. Acquisition of data does not happen in a vacuum with a data engineer or a data scientist. It requires a combined effort with a subject matter expert (SME) who has a deep understanding of where the supporting data is and how to acquire it.

During this iterative process of identifying and testing data against the sharp question, there are two key ingredients that the team is looking for while building the data set that will be used to build a model.

1. Features: Data points that relate to the sharp question.
2. Labels: Data points that identify historic outcomes

Labels can be optional, and if labels are not present this is called *unsupervised learning* simply because we do not know what we are predicting. This may sound odd, but in fact this can be used for creating models for anomaly detection. Finding out when something “weird” happens.

When both features and labels are included in the dataset it is called *supervised learning.* That is, we know what we are trying to predict.

Whatever is there for the data, each row is called an observation. Again, an observation may or may not include labels.

While collecting the data, there are several questions that the team needs to ask of the data:

* Is the data relevant to the sharp question?
  + For example, if our sharp question is “if I drop this glass will it break?”. Relevant data is the type of glass, the height it is dropped from, does it contain liquid? Irrelevant data would be the day of week or time of day it’s dropped.
* Is the data connected?
  + Connected simply means that the columns that were identified as features contain enough data. Data is considered disconnected if there are too many holes in the data.
* Is the data accurate?
  + This is as simple as it sounds. Does the data reflect the ground truth? Are the sensors calibrated, is the height appropriately noted, etc.
* Is there enough data?
  + Again, while it seems like a straightforward question, it is very important to model accuracy. This question tries to determine if we have enough information to make an intelligent decision using a model.

With the sharp question identified, data collected and run through the above series of questions, the work of tool selection and model building begin in earnest.

# Technologies Utilized

This paper will utilize many different technologies and expertise in each is not required to gain useful insights. It is required, however, that if you do wish to follow through the example that you have an Azure Subscription that can support the services listed in [Azure and Other Services](#_Azure_and_Other). It is not recommended to manually deploy the resources if you have a free or student Azure subscription as this deployment will cost approximately $1.5K a month.

The reader is strongly encouraged to follow along with the setup and deployment steps that follow for retention purposes, but doing so is not required to understand the content in this paper. If the reader does want to implement the solution as the paper progresses a valid Azure Subscription will be required. Subscriptions can be bought [here](https://azure.microsoft.com/en-us/pricing/purchase-options/?v=17.28), or a free subscription can be created [here](https://azure.microsoft.com/en-us/free/?v=17.39a).

## Azure Machine Learning Workbench

The [Azure Machine Learning Workbench](http://azureml.azureedge.net/content/apphome/index.html) (AML Workbench) is the latest incarnation of the machine and deep learning toolset offered by Microsoft offering more flexibility and finer control over the first iteration of [Azure Machine Learning Studio](https://studio.azureml.net/) and is targeted towards the more experienced data scientist.

Using the AML Workbench, the data scientist can choose the language and libraries that best solve the challenge at hand. The workbench also comes with new Azure CLI extensions for machine learning that make creating and deploying Docker containers that host the model trivial.

## Programming Languages

This list of languages are used in developing the machine learning model as well as consumption of the model after deployment.

* Python
* Jupyter IPython Notebooks
* C#

## Azure and Other Services

* [Azure Resource Groups](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview)
* [Azure Storage](https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction)
* [Ubuntu Azure Data Science Virtual Machine](https://azure.microsoft.com/en-us/services/virtual-machines/data-science-virtual-machines/)
* [Windows Azure Data Science Virtual Machine](https://azure.microsoft.com/en-us/services/virtual-machines/data-science-virtual-machines/)
* [Container Registry](https://azure.microsoft.com/en-us/services/container-registry/)
* [Machine Learning Experimentation (new)](https://azuremarketplace.microsoft.com/en-us/marketplace/apps/Microsoft.MachineLearningExperimentation?tab=Overview)
* [Machine Learning Model Management (new)](https://docs.microsoft.com/en-us/azure/machine-learning/preview/model-management-overview)
* [Docker](https://www.docker.com/)

# Building a Machine Learning model for a Factory

Contoso Manufacturing has a facility with 100 devices producing widgets for the worlds leading pet-rock industries.

When the production line stops, they lose millions in both production and lost inventory.

Contoso would like to build a model that can tell them when a line device is experiencing issues that lead to a failure so that the device can be fixed prior to a production run.

The sharp question they have identified is “which device(s) are in a state of near failure”? Identifying the answer to this question could save the company many millions of dollars on an annual basis.

This particular production line has been in service for several years and there is a substantial amount data that has been collected through the sensors on these particular devices. This data is stored in multiple database tables that hold sensor readings from the line devices, production results, and a host of other business information.

Contoso’s SME has full access to the information stored in the database and has identified two distinct data sets that the core information to help answering the sharp question. The first dataset contains the sensor readings from the devices on the line, and the second dataset contains the production results.

Joining these two datasets the SME has identified both features and labels as described in the previous section - [Data Science – It’s not magic!](#_Data_Science_–)

The following table describes the dataset that the Contoso SME has accumulated for the data science project.

|  |  |  |
| --- | --- | --- |
| **Field** | **ML Importance** | **Data** |
| temp | Feature | Operating temperature of the device |
| volt | Feature | Voltage required to turn the device |
| rotate | Feature | Rotation speed of the device |
| state | Label | Operational state. 0 == Normal operation, 1 == failure mode |
| time | Feature | Time stamp of the recording, can be used to pass back from model with device id for deeper granularity in dashboards. |
| id | Feature | Device identity |

A small example of the data that was collected follows in the next table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| temp | volt | rotate | state | time | id |
| 45.98425945 | 150.5132231 | 277.294014 | 0 | 1 | 1 |
| 52.90392969 | 110.4340753 | 314.5867262 | 0 | 2 | 1 |
| 53.82550722 | 169.2595188 | 315.6025021 | 0 | 3 | 1 |
| 47.78267591 | 110.8295315 | 345.8946517 | 0 | 4 | 1 |
| 43.4792264 | 199.3516748 | 325.3640807 | 1 | 5 | 1 |
| 49.59376671 | 145.6429489 | 370.5231473 | 0 | 6 | 1 |
| 49.55370077 | 177.9368947 | 345.7171063 | 1 | 7 | 1 |
| 49.62468858 | 150.4729976 | 310.4098192 | 0 | 8 | 1 |
| 48.66065109 | 155.3300661 | 342.1765811 | 0 | 9 | 1 |
| 52.13914561 | 163.9654119 | 371.1262708 | 0 | 10 | 1 |
| 53.84428872 | 227.5308994 | 332.7407004 | 1 | 11 | 1 |
| 55.90476339 | 186.296285 | 344.5849566 | 1 | 12 | 1 |

Admittedly, this data set is small by any means. There are only 10000 samples across 100 devices. This set is kept purposefully small. In a real world scenario this dataset would need to be significantly larger to ensure we have enough data to properly build a model.

## Preparing Development Environment

Before we can really dig in to creating a model, we will need to create some resources and set up a development environment.

|  |
| --- |
| **NOTE:** If you do not plan to perform the tasks manually you should skip ahead to the sections under [Build the model with Azure Machine Learning Workbench](#_Build_the_model) |

### Azure Resource Group

A new Azure Resource Group will be used to house the machine learning services, virtual machines and a storage account complete this design.

To set up an Azure Resource group, , follow the instructions on [this](https://docs.microsoft.com/en-us/azure/azure-resource-manager/vs-azure-tools-resource-groups-deployment-projects-create-deploy) page.

### Machine Learning accounts

The Azure Machine Learning accounts service will house the machine learning workspaces, projects and model management that will be created in the following steps.

To set up an Azure Machine Learning account, follow the instructions on [this](https://docs.microsoft.com/en-us/azure/machine-learning/preview/quickstart-installation) page.

|  |
| --- |
| **NOTE:** Ensure that the Machine Learning Experimentation and Machine Learning Model Management are created in the same resource group created in the section [Azure Resource Group](#_Azure_Resource_Group) |

Do not follow the remaining instructions for installing the AML Workbench at this time.

### Windows Data Science Virtual Machine(DSVM)

The Windows DSVM is based on Windows server 2016 and will be used to house the AML Workbench.

To set up a Windows DSVM, follow the instructions on [this](https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/provision-vm) page.

|  |
| --- |
| **NOTE:** Ensure that the DSVM is created in the same resource group created in the section [Azure Resource Group](#_Azure_Resource_Group) |

### Linux Data Science Virtual Machine (DSVM)

The Windows DSVM is based on Ubuntu Linux operating system and will be used to house a remote Docker host to run the data science IPython notebooks.

To set up a Ubuntu DSVM, follow the instructions on [this](https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/dsvm-ubuntu-intro) page.

If you prefer to have the ability to remote desktop to your Ubuntu DSVM, follow the instructions [here](https://docs.microsoft.com/en-us/azure/virtual-machines/linux/use-remote-desktop) to install a desktop environment on it, otherwise you can use an SSH tool like PuTTy to access the machine.

|  |
| --- |
| **NOTE:** Ensure that the DSVM is created in the same resource group created in the section [Azure Resource Group](#_Azure_Resource_Group) |

### Azure Storage Account

The Azure Storage account will be used as a storage location between the two DSVM instances.

To set up an Azure Storage account, follow the instructions on [this](https://docs.microsoft.com/en-us/azure/storage/common/storage-create-storage-account/) page.

|  |
| --- |
| **NOTE:** Ensure that the Azure Storage account is created in the same resource group created in the section [Azure Resource Group](#_Azure_Resource_Group) |

### Azure Machine Learning Workbench

AML Workbench is the core tool used to inspect the data, test models, and create the container that will be operationalized. In the latest versions of the Windows DSVM (2016) a shortcut on the desktop is provided to install workbench : AzureMLWorkbenchSetup.

### Copy Project files to Windows DSVM

In this repo, download and copy to the Windows DSVM the contents of the /Experiment folder. You will be instructed at a later point as to where to locate these files.

## Configuring Azure Machine Learning Workbench

|  |
| --- |
| **NOTE:** If you do not plan to perform the tasks manually you should skip ahead to the sections under [Working with the data](#_Working_with_the) |

With the development environment in place it is time to create and configure a new project.

### Creating the first project

When launching the AML Workbench for the first time you will be asked to login to your Azure account. Make sure this is the same account as the one in which you created the previous resources.

During the creation of the Machine Learning account you would have provided a workspace name and that workspace will appear on the ***PROJECTS*** pane.

* Click the + sign at the top of the *Projects* pane and select *New Project*
* Provide a project name and project directory. You will need to remember this directory for populating the project with the downloaded files.
* Ensure that your workspace is selected
* Select *Blank Project*
* Click *Create* and provide the required details of name and local disk location.

Upon creation of the project following items are created:

* A project directory on the Windows DSVM where you selected it
* A Microsoft.MachineLearningExperimentation/accounts/workspaces/projects resource in your Azure account in the same resource group in which you created the Azure Machine Learning account.

### Prepare the project source files

Open Windows Explorer and navigate to your project directory to view the structure. Navigate into the project directory so that you are seeing the readme.md file and then delete the readme.md, score.py and train.py files.

Copy the content from directory that was downloaded in the section [Copy Project files to Windows DSVM](#_Copy_Project_files) to the project directory. The following table describes the content of the directory:

|  |  |
| --- | --- |
| **Resource** | **Details** |
| Data directory | Contains the training data from the Contoso manufacturing plant. |
| readme.md | The project home page with details on the project |
| score.py | The Python script that will be deployed with the model |
| 1\_buildmodel.ipynb | This file loads and splits the data for consumption while building and testing a model. The model is built then uploaded to the Azure Storage account. |
| 2\_modelschema.ipynb | Generates the schema for the model for use with deployment |
| 3\_retrieve\_content.py | Retrieves the necessary files for submitting the model to a Model Management service to create a Docker image. |

Now that the data and source files are in place we have a few configuration items we need to attend to.

### Prepare dependencies

Extending the project dependencies list is crucial. There are certain libraries we will need for development and for deployment.

In the project click on the files button () on the project and navigate to aml\_config/ and then click on the conda\_dependencies.yml. Make the following additions/changes

|  |  |  |
| --- | --- | --- |
| **Section** | **Value** | **Details** |
| dependencies | - scikit-learn | Place this directly under the line python=XXXX  This will ensure that we have access to the python libray for Scikit learn that provides trainable Machine Learning algorithms.  To learn more about scikit-learn read here (<http://scikit-learn.org/stable/>) |
| pip | - azureml.datacollector | This library provides the ability to extract the schema from the model that will be built in preparation for deploying the model. |
|  | - azure-storage | This library, while present on both DSVM systems will not be present when we deploy the project to a Kubernetes cluster and hence is required for preparing the environment |
|  | - azure-ml-api-sdk | Ensure the version is at least 0.1.0a11 |

When done click the save button on the top of the file pane.

### Prepare the project to use Ubuntu Docker

With the project configuration complete it needs to be linked to the Ubuntu DSVM as the Docker execution environment for the scripts and notebooks. This step is completed by using the Azure CLI.

* Click on the *File* menu and select *Open Command Prompt*
* Enter the following command to login to azure for the following commands to work.
  + **az login**
* Enter the following command with your values replaced to create the compute environment.
  + **az ml computetarget attach remotedocker --name [name] --address [ip]--username [user] --password [password]** 
    - name = The name you want to provide for this execution environment, i.e. ubuntudsvm for example.
    - ip = The IP address of the Ubuntu server which can be found on the Azure Portal.
    - user = The admin user name supplied while creating the Ubuntu server.
    - password = The admin password provided for the user while creating the Ubuntu Server.
  + This step creates two new files in the aml\_config directory that describe the compute environment
    - [name].compute and [name].runconfig
  + The execution of this command may take several minutes.
* Prepare the project to use the newly created compute environment by entering the following command on the command prompt
  + **az ml experiment prepare --target [name] --run-configuration [name]**
    - name = The name provided when creating the compute environment
  + The execution of this command may take several minutes.

With these steps completed the environment is ready to start the development work.

## Working with the data

### Importing Data

Now that the environment is configured, the next step is to import the data that we are going to need for the project. This has been provided in the zip file content in the Data directory.

* On the project pane, click on the data () icon
* Click the + button and choose Add Data Source
* Select Text Files and click Next
* Browse/File to the project/Data directory and choose dataset.csv and click Open
* Click Next
  + A preview of the data is shown.
* Continue to click Next looking through the data and finally click Finish.

|  |
| --- |
| **NOTE:** There are no settings to change in the data as it has previously been cleaned. To find out more about cleaning/working with data in the workbench tool, choose one of the predefined projects and walk through those tutorials. |

The data has been imported and we need to prepare it for use in the scripts. At the top of the pane showing the data that was just imported, click on the *Prepare* button. When prompted:

* Data Preparation Package - + New Data Preparation Package
* Data Preparation Package Name –dataset

Again, since this data has been cleansed there are no further preparation steps that are required.

To access the data in our source files we can obtain a snippet of Python code that will allow us to gain access to it by:

* On the *Data Preparation* filewe just created, in the files pane, right click and choose *Generate Data Access Code File*
* A new tab with the file dataset.py is presented with the Python code snippet to read in the dataset.

In this section we created a new dataset to work with inside of workbench and our source files. By doing so the following files were created in your project directory:

|  |  |
| --- | --- |
| **File** | **Details** |
| dataset.dsource | The datasource file from the data we imported. |
| dataset.dprep | The preparation steps on the datasource |
| dataset.py | The example Python code snippet to load in the data set from the dprep file. |

### Splitting Data and Building a Model

With the environment set and the data ingested to the workbench the actual work of building a model can be completed.

In the files pane click on the file *1\_buildmodel.ipynb* to open the Juypter IPython notebook and click *Start Notebook Server* at the tope of the file pane. Conversely, from the command prompt you can enter the command ***az ml notebook*** start. Either of these actions will start the Juypter Notebook Server locally.

When the server starts a dialog *Kernel not found* appears. From the drop list choose the option [project] [environment] where [project] is your project name and [environment] is the Ubuntu compute environment defined in the section [Prepare the project to use Ubuntu Docker](#_Prepare_the_project) and finally click *Set.*

|  |
| --- |
| **NOTE:** If you inspect the Ubuntu DSVM docker environment you will notice that a Docker container has been activated. This is the host for this notebook execution. |

In this notebook information is required to execute correctly, and each value is entered in the first code cell of the notebook:

|  |  |
| --- | --- |
| **Name** | **Description** |
| AZURE\_STORAGE\_ACCOUNT\_NAME | The storage account name that was created in the section [Azure Storage Account](#_Azure_Storage_Account) |
| AZURE\_STORAGE\_ACCOUNT\_KEY | The primary access key, found in the Azure Portal, associated with the storage account created in the section [Azure Storage Account](#_Azure_Storage_Account) |
| AZURE\_STORAGE\_CONTAINER\_NAME | The container name to be used to pass the model around. This container must be created before running the full notebook file. |

The dataset that was imported will be used for both training a model and testing it. Therefore, the data needs to be split to be useful. That is, we need to use some of the data to train the model and the remaining data to test it to see how we do. For this example we will use 70% of the data for training and 30% for testing.

Inspecting the data there are ~80% successful devices and ~20% failed devices. Now, if 70% were just grabbed from the whole data set we could end up with some seriously skewed data. What if they were all successful devices? What if the testing data had no failed devices?

To solve this we stratify the split by performing the following steps:

* Get all of the successful devices
* Get all of the failed devices
* Create two datasets
  + Training – 70% each of the successful and failed devices
  + Testing – The remainder of the data not used for training

With the data separated, we need to build a model. For this case we can use the scikit-learn DecisionTreeClassifier.

That model is trained with the training data and the results are inspected for accuracy, precision, recall and fscore because accuracy is not enough to base a decision on. Read this (<https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/>) article to learn more.

Once the model is trained it is then uploaded to the Azure Storage account for later deployment, but as a final sanity check, the model is downloaded and executed with the same testing data to ensure that nothing changed during the upload process.

### Creating a schema

The next step, once the model has been developed, is to create a schema that will be used when publishing the model as a REST service.

In the files section click on *2\_modelschema.ipynb*. Again select the kernel [project] [environment] where [project] is your project name and [environment] is the Ubuntu compute environment defined in the section [Prepare the project to use Ubuntu Docker](#_Prepare_the_project) and finally click *Set.*

If you have followed to this step immediately after the previous step, the Jupyter notebook server will still be running.

This notebook is essentially the same code that exists in the *score.py* file which is described later.

The model is downloaded from storage, and the model is initialized. Run is called on the model to get the prediction, but just as importantly it’s tested for functionality.

Finally the schema for the new service is generated and uploaded to the Azure Storage account.

### Scoring

Scoring with the model occurs while the service is deployed. For the service to function properly a *score.py* file is required. In many cases this is the same file that will be used to generate the schema and be published with the model as a service.

There are two critical functions needed in *score.py* when publishing a model

|  |  |
| --- | --- |
| **Function** | **Purpose** |
| init() | This function is responsible for obtaining the model, wherever it is, and creating a global *model* object. This object will be used for each call to the service on the specific container and init is run only once when the container is activated. |
| run(input\_df) | This function is called each time the service is requested to provide a prediction. The return of this call is the body of the HTTP POST request to the REST API. |

The *score.py* file will be utilized in the deployment.

### Deployment

|  |
| --- |
| **NOTE:** The deployment phase will create a resource of type Microsoft.ContainerRegistry. To ensure that this step succeeds, your subscription must have the Microsoft.ContainerRegistry resource provider registered. A resource provider need only be registered once per subscription.  For a description of providers and resource in azure read this article (<https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-supported-services>)  To validate your subscription is registered to use the Microsoft.ContainerRegistry provider run the following command from the Azure CLI:  az provider show -n Microsoft.ContainerRegistry | grep 'registrationState'  If the state comes as anything but “Registered”, run the following command  az provider register --namespace Microsoft.ContainerRegistry |

While there are several options for deployment, this paper will instruct the reader on how to deploy the model to an Azure hosted Kubernetes cluster.

The first step is to collect the necessary files in one place on the disk for the deployment command. To do so, open the 3\_retrieve\_content.py file in AML Workbech. Ensure that the Run Configuration () is set to local.

In the file modify the variable PROJECT\_DIRECTORY to the path of your project. Deployment files will be collected in PROJECT\_DIRECTORY \deploypackage. When the modification of the path is complete, click the *Run* button to collect the required files.

The list of files collected are:

|  |  |
| --- | --- |
| **File** | **Description** |
| conda\_dependencies.yml | The file that lists the requirements to run the package. Identified resources and packages in this file will be made available in the docker container. |
| factory.pkl | The model that was built in previous steps |
| factory.schema | The required schema of the model (??) |
| score.py | Source file that contains the run() function which will be run when the service is called. |

The following set of instructions create a Kubernetes cluster and deploy the service as a docker container to it.

When completed, the service will appear under the Model Management account created when the [Machine Learning accounts](#_Machine_Learning_accounts) was created.

|  |
| --- |
| **NOTE:** In the following commands another Azure Resource group will be created. While any existing resource group can be used for this purpose, creating the Kubernetes cluster will create several Azure services and it’s easier to maintain resources if they exist in separate Azure Resource groups. |

On the menu of the workbench, while in the project we are working on, open the command prompt from File\Open Command Prompt and perform the following actions, in order as they appear in this document.

|  |  |
| --- | --- |
| **Description** | Create a new Kubernetes cluster to host the Docker image that will serve up the new ML model as a REST API |
| **Command** | |
| az ml env setup -c --cluster --name [name] --location eastus2 -g [clusterrsrcgroup] | |
| **Parameters** | [name] – The user defined name to be applied to the new Kubernetes cluster to be deployed.  *eastus2* – The region to create the new Kubernetes cluster. Currently this can be eastus2, westeurope, westcentralus, australiaeast, and southeastasia.  *[clusterrsrcgroup]*– This is the name of the resource group that will be created to hold the Kubernetes cluster. |
| **NOTE** | This step will take 10-20 minutes to complete. Using the Azure CLI you can determine when the deployment has completed by issuing the following command:  az ml env show -g [clusterrsrcgroup] -n [name]  This step may also have the wrong subscription selected, ensure that the subscription is the one we have been working with until this point. |

|  |  |
| --- | --- |
| **Description** | Set the execution environment that the az ml engine will use to deploy the model. |
| **Command** | |
| az ml env set -n [name] -g [clusterrsrcgroup] | |
| **Parameters** | [name] – The user provided name for the Kubernetes cluster  [clusterrsrcgroup]– The name of the resource group that was provided when creating the Kubernetes cluster. |

|  |  |
| --- | --- |
| **Description** | Set the Microsoft.MachineLearningModelManagement resource for the az ml engine to register the Docker container to the correct Microsoft.ContainerRegistry resource. This model management resource was created when creating the [Machine Learning accounts](#_Machine_Learning_accounts) |
| **Command** | |
| az ml account modelmanagement set --name [mdlmgmt] --resource-group [mdlmgmtrsrc] | |
| **Parameters** | [mdlmgmt] – The name of the Microsoft.MachineLearningModelManagement resource created when creating the [Machine Learning accounts](#_Machine_Learning_accounts)  [mdlmgmtrsrc] – The resource group containing the Microsoft.MachineLearningModelManagement resource created when creating the [Machine Learning accounts](#_Machine_Learning_accounts) |
| **NOTE** | It is possible to reuse a single Microsoft.MachineLearningModelManagment across multiple projects, however delineating resources based on project makes simplifies maintenance and clean up tasks for resources associated with a particular service. |

|  |  |
| --- | --- |
| **Description** | The final deployment step performs multiple tasks but rely on the previous commands to appropriately set the execution environment.   1. Creates a manifest for the image to create. 2. Builds a Docker image 3. Registers the Docker image with the container registry 4. Deploys the container to the Kubernetes cluster 5. Enables the service to be called externally |
| **Command** | |
| At the command prompt, change the directory to the /deploypackage directory that was created earlier in your project directory.  az ml service create realtime --model-file factory.pkl -f score.py -n [servicename] -s factory.schema -r spark-py -c conda\_dependencies.yml | |
| **Parameters** | [servicename] – The user provided name for the service |
| **NOTE** | This step will take several minutes |

#### Debugging the deployment

Deployments don’t always go right the first time. Two typical culprits are the score.py file has bugs in it, or dependencies available on the development machine are not present in the Docker container and updates are required to the conda\_dependencies.yml file.

If there is an issue either deploying or calling a deployed service, you can gather logs about the service. First you need to gather the service id, which can be done by issuing the following command on the Azure CLI :

*az ml service list realtime*

From the list of returned services, find the service you would like to deploy, copy the service id and then issue the following command to retrieve the logs:

*az ml service logs realtime -i [serviceid]*

More information about debugging deployments can be found here (<https://docs.microsoft.com/en-us/azure/machine-learning/preview/how-to-deploy-troubleshooting-guide>)

## Consuming the Model

Consuming the model is straightforward, and typically this is where the engineering team becomes familiar with the service that was deployed in the previous sections.

The first step is to determine how to the service is called. The Azure ML CLI (installed on the Windows DSVM with Azure Machine Learning Workbench) can provide the service information using the following command, for general information about the Azure ML CLI read [this](https://docs.microsoft.com/en-us/azure/machine-learning/preview/model-management-cli-reference) documentation.

The first thing you will need is the service ID of your service. At the command prompt enter the following:

az ml service list realtime -o table

This will list out the services available. Find the service that was just published and copy the *servicid* and use it in the next command.

az ml service usage realtime -i *[serviceid]*

This will print out a bunch of information about the service. For example the URL of the endpoint, and examples on how to call it with data. The model created in this document, the output should contain the following :

Usage for cmd: az ml service run realtime -i [serviceid] -d "{\"input\_df\": [{\"rotate\": 277.294013981084, \"id\": 1.0, \"time\": 1.0, \"temp\": 45.9842594460449, \"volt\": 150.513223075022}]}"

This provides the input message format that the experiment is expecting. Determining the output is as easy as calling the service with the example input from the az ml service usage output.

Part of that output explains how to obtain the key needed for the header. The call will look like the following:

az ml service keys realtime -i *[serviceid]*

You will need to copy the URL and one of the service keys to call the endpoint. The provided code is in C# but as a REST endpoint, the service can be called from the developers language of choice. To find out more about calling from other languages click [here](https://docs.microsoft.com/en-us/azure/machine-learning/preview/model-management-consumption).

In this repository, a full working C# example can be found in the /Client directory.

# Summary

This document has introduced data science and machine learning at a very high level. Using that knowledge, a problem was identified by the Contoso Manufacturing company.

Azure Machine Learning Workbench was introduced, and utilized, to create a model using the Python library scikit-learn.

This model is then published using new Azure services to a Kubernetes cluster as a docker container and operationalized as a REST endpoint.

Finally, the document gives an example on how to consume this model through code that could be included in any application or service.

# Appendix A: Azure CLI

This section contains the needed Azure CLI commands you will need and that are sprinkled throughout the document.

For any command you can get help by simply typing the command with a -h in the argument list.

## Azure Login

When using the CLI you need to first log in. The following series of commands will accomplish that goal:

|  |
| --- |
| >az login  >az account show  >az account list --output table  >az account set --subscription [subscriptionname] |

Ensure you are in the right subscription using account show. To change subscriptions, use account set with a value found after executing account list.

## AML Workbench : Configure Remote Docker Host

Using a remote docker host for workbench is recommended. Once you have a machine set up and have collected the systems IP address [ip], admin user name [au], and admin password [ap], you can associate that host with the project. You just provide a connection name [cn] that you want to refer to this connection in the workbench.

Upon completion of identifying a target, new files [cn].compute and [cn].runconfig will be created in the /aml\_config folder.

The second step is to prepare the target with the appropriate base docker image and associated required dependencies.

When these steps are completed, .py and .ipynb files can be executed on the remote host.

|  |
| --- |
| >az ml computetarget attach remotedocker -n [cn] -a [ip] -u [au] -w [ap]  >az ml experiment prepare -t [cn] -c [cn] |

## Create Kubernetes Cluster and associate with Workbench Experiment

A Kubernetes cluster is used as the operational public endpoint for the experiment. The following commands create a Kubernetes based Azure Container Service.

The default configuration (shown) creates one master and two nodes.

Provide a cluster name [cn], a new or existing Azure Resource Group name [rg] and a region [loc]. The region is currently required to be one of the following:

eastus2, westcentralus, australiaeast, westeurope, or southeastasia

The second command can be used to check the status of the deployment and when completed, run the third command to set the cluster environment for the current experiment.

|  |
| --- |
| >az ml env setup --cluster -n [cn] -l [loc] -g [rg]  >az ml env show -g [rg] -n [cn]  >az ml env set -g [rg] -n [cn] |

## Set the Model Management Account for the Experiment

The Azure Model Management tracks models, images, manifests and services that have been created and or deployed. Provide the model management account name [mma] and the resource group name [rg] that the model management account resides in.

This will associate the provided model management with this experiment and allow creating new services.

|  |
| --- |
| >az ml account modelmanagement set -n [mma] -g [rg] |

## Create a Service within Model Management

Creating a service can be done in discrete steps OR in a single command. Choosing one over the other depends on what you are trying to accomplish.

If you are updating an existing service, you would want to use multiple commands to re-build an image then update a service. If you are creating a new service, you would likely choose the single command.

At the successful completion of either path a service is created that is now accessible to callers.

In both cases there are some similar input parameters to the calls.

|  |  |
| --- | --- |
| Parameter ID | Usage/Meaning |
| [name] | The name you provide for the service. |
| [model] | The model created for the experiment, for example model.pkl |
| [manname] | A name associated with the manifest.  The manifest is registered with the model management service that identifies what is needed for docker image creation. |
| [imgname] | Docker Image Name |
| [score] | The score.py file name that is used in the deployed service that contains the init() and run() functions for the service. |
| [schema] | The schema file created for the service. |
| [conda] | The conda\_dependencies.yml file for the experiment. |
| [run] | Runtime of the service, valid selections python|spark-py |
|  |  |
| [modid] | Model ID, returned by az ml model register |
| [manid] | Manifest ID, returned by az ml manifest create |
| [imgid] | Docker image ID, returned by az ml image create |

### Multiple Commands

|  |
| --- |
| > az ml model register -m [model] -n [model]  > az ml manifest create -n [manname] -f [score] -r [run] -i [modid] -s [schema]  > az ml image create -n [imgname] --manifest-id [manid] -c [conda]  > az ml service create realtime --image-id [imgid] -n [name] |

### Single Command

|  |
| --- |
| > az ml service create realtime -m [model] -f [score] -n [name] -s [schema] -r [run] -c [conda] |

## Get Deployed Service Information

Once a service has been created it’s important to collect information about that service for clients. In particular there are 4 key items required:

|  |  |
| --- | --- |
| Service URL | The external URL of the service to call |
| Service API Key | The API key for the service |
| Service Input Format | The input JSON format |
| Service Output Format | The output JSON format |

To find out any of this information, you need to get the service ID. This is done in the first command, by listing out the available services. From this table, retrieve the service id of the service and follow with the commands that follow to get the url, key and input format.

From the az ml service usage call, you can retrieve the command to issue on the command line to test the service. Execute that command to find out the output format.

The final command can be used to see logs for the service to determine service state.

|  |
| --- |
| > az ml service list realtime -o table  > az ml service usage realtime -i [serviceid]  > az ml service keys realtime -i [serviceid]  > az ml service logs realtime -i [serviceid] |

# Appendix B: Kubernetes kubectl.exe

Kubectl is a very useful tool for working with a kubernetes cluster. It can be used for discovery, diagnosis, scaling, and opening a proxy to view the kubernetes dashboard.

## Prepare /.kube/config file

To use kubectl, you must configure it with your cluster information. This information can be found using the Azure ML CLI. First ensure that you have used az login.

The first command displays the environment currently configured for the Azure ML CLI. If this is incorrect for the cluster you are looking for, list out the environments. Capture the resource group [rg] and cluster name [cn] of the cluster of interest and then set the environment.

|  |
| --- |
| > az ml env show  > az ml env list  > az ml env set -g [rg] -n [cn] |

Once the ml environment is set, navigate the command prompt to the directory carrying the kubectl config file, which on a Windows DSVM is C:\users\[user]\.kube and run the following command. This command will update the config file with the information about your cluster.

|  |
| --- |
| > az ml env get-credentials -n [cn] -g [rg] -i |

## Setting the context in kubectl

With the config file updated, navigate the command prompt to the directory that has the file kubectl.ext which on the Windows DSVM is C:\users\[user]\bin

If running config current-context is not the context of your cluster, view the list of configurations and get the context name for the one that matches your cluster. Finally, set that context name as the current context.

|  |
| --- |
| > kubectl config current-context  > kubectl config view  > kubectl config use-context [context] |

## Run the k8s Dashboard

Now that kubectl is configured with your cluster as the current context, start the proxy to allow you to view the dashboard running this command

|  |
| --- |
| >kubectl proxy |

It will direct you to a URL, but generally that doesn’t work. Instead, past the following URL into a browser to bring up the dashboard:

|  |
| --- |
| <http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/kubernetes-dashboard/proxy/#!/node?namespace=default> |