LESSON 6: Managed Services for ML

Section 1: Lesson Overview

This lesson covers **managed services for Machine Learning**, which are services you use to enhance your Machine Learning processes. We will use services provided by Azure Machine Learning as examples throughout the lesson.

You will learn about **various types of computing resources** made available through managed services, including:

- **Training compute**
- **Inferencing compute**
- **Notebook environments Compute resources** are **clusters of computers** running in the **Cloud**, providing you the **raw computing power needed** by your ML workloads.

You will also study the **main concepts** involved in the **modeling process**, including:

- **Basic modeling**
- How **parts of the modeling process interact** when used together
- More **advanced aspects of the modeling process**, like **automation via pipelines** and **end-to-end integrated processes** (also known as **DevOps for Machine Learning** or simply, **MLOps**)
- How to move the results of your modeling work to **production environments** and make them **operational** (**operationalising models**)

Finally, you will be introduced to the world of **programming the managed services** via the **Azure Machine Learning SDK (Software Development Kit) for Python** (**programatically accessing the managed services**).

Section 2: Managed Services for ML

The machine learning process can be **labor intensive**. Machine learning requires a number of tools to prepare the data, train the models, and deploy the models. Most of the work usually takes place within **web-based, interactive notebooks**, such as Jupyter notebooks. Although notebooks are lightweight and easily run in a web browser, you still need a **server to to host them**. Typically, this involves installing several applications and libraries on a machine, configuring the environment settings, and then loading any additional resources required to begin working within notebooks or integrated development environments (IDEs).

Conventional ML approach:

- **Lengthy installation and setup** process (i.e. installing several applications and libraries on the machine, configuring the environment setting and loading all the resources needed to begin working within a notebook or IDE)
- **Expertise** to **configure hardware** (for deep learning you require expertise to configure hardware-related aspects, such as **GPUs**)
- Fair amount of **troubleshooting** (you need the right combination of software versions, compatible with one another)

All this setup takes time, and there is sometimes a fair amount of troubleshooting involved to make sure you have the right combination of software versions that are compatible with one another. This is the advantage of **managed services for machine learning, which provide a ready-made environment** that is pre-optimized for your machine learning development.

for your machine learning development.

Managed Service approach:

- **Very little setup**
- **Easy configuration** for any needed **hardware**
- MS Azure being cloud based can be run from anywhere and any machine

When you are ready to run your experiment you only need to specify a **compute target**, a computer resurce to run your experiments and host your service deployments.

MS Azure managed service go beyond **compute resource management** and also offer support for **datastore and datasets management**, **model registry**, **deployed service endpoint management**, ...

Examples of **compute resources**:

- **Training clusters**: for training the model
- **Inferencing clusters**: for operationalising the model
- **Compute instances**: they contain notebook environments to run notebook-based code
- **Attached compute**: e.g. you can attach extra virtual machines to Azure ML environment
- **Local compute**

Examples of **other services**:

- Notebooks gallery
- AutoML configurator
- Pipeline designer
- Datasets and datastores managers
- Experiments manager
- Pipelines manager
- Models registry
- Endpoints manager

Section 4: Compute Resources

A **compute target** is a **designated compute resource or environment** where you run training scripts or host your service deployment. There are two different variations on compute targets that we will discuss below: **training compute targets** and **inferencing compute targets**.

Training Compute

Compute resources that can be used for **model training**:

- **Training clusters**: usually the primary choice, especially for large-scale tasks, as they have multi-node scaling capabilities. Characteristics:
  - You can use them for training and batch inferencing (e.g. generic purposes of running ML Python code)
  - **Single** or **multi-node** cluster
  - **Automatic cluster management and job scheduling**
  - **Support** for both **CPU** and **GPU** resources
- **Compute instances**: primarily intended as a notebook environment
- **Local compute**

Inferencing/Scoring Compute

Once you have a trained model, you'll want to be able to **deploy it for inferencing**:

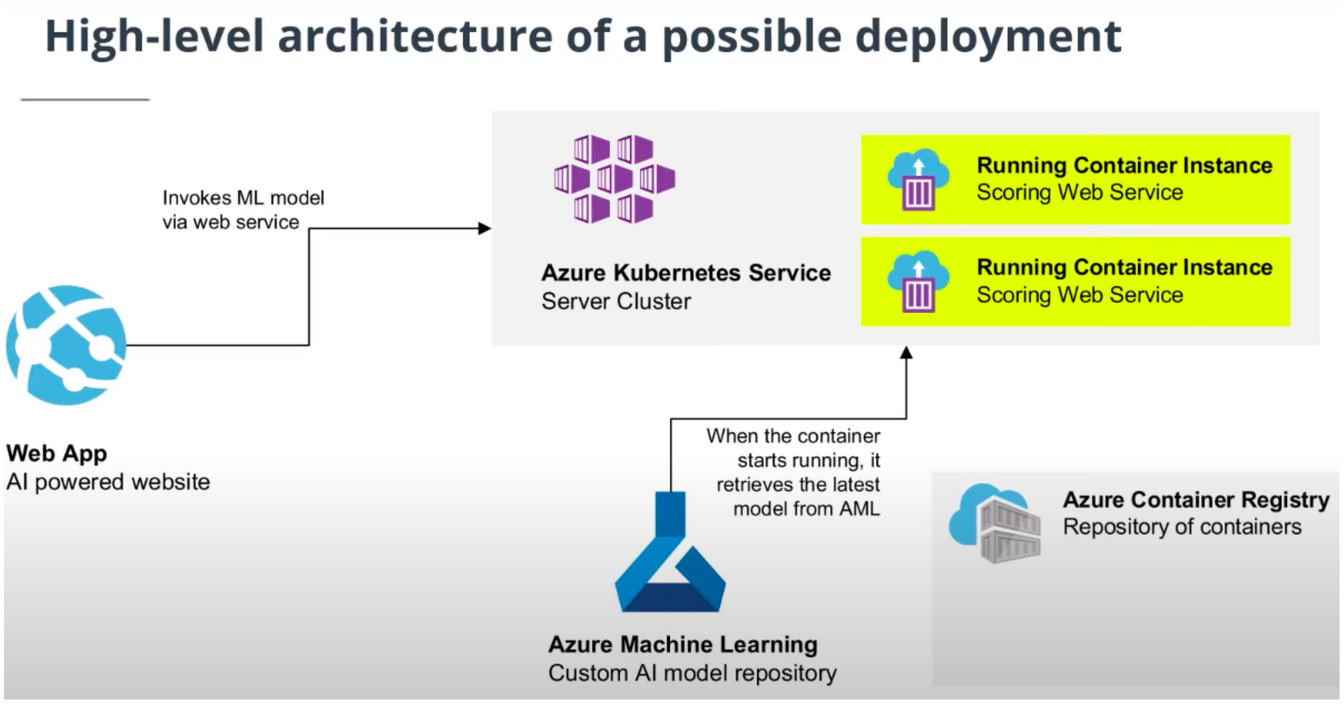
- **Real-time inferencing**: inferences for each new row of data, usually in real-time (the model is usually packaged in a **web service**; other possible packages approaches are deployment to an IoT device)
- **Batch inferencing**: inference on multiple rolls of data (batches). The resources required can sometimes be significant.

Let's take a look at the compute resources you can use for different types of inferencing.

- **Inferencing clusters**: dedicated computed resource for **real-time inferencing**. After you have trained your model you can deploy it to a web hosting environment or IoT device. Using the model, it infers things about the new data
- **Any compute resource** can be used for **batch inferencing**. However, for **inferencing in production**, you can choose compute targets specific for **production workloads**:
  - **Azure Kubernetes Services (AKS)**: fast response times, auto-scaling for deployed services, hardware acceleration options (such as GPUs or FPGAs = Field Programmable Gate Arrays)
  - **Azure ML training cluster**
- **Specialised inferencing scenarios**: trained models are **packaged in containers** using computer targets specialised for **deployment**:
  - **Azure Functions**
  - **Azure IoT Edge**
  - **Azure Data Box Edge**

High-level architecture of a possible deployment:

- A web app the users visit to make predictions with the deployed ML models.
- The models are deployed as a container instance in an Azure Kubernetes Service Clusters for high scalability.
- Azure ML was used to train these models, registering and scoring them in Web Service Docker Images in the Azure Container Registry.
- The model images are deployed to designated inferencing clusters. The models are stored within Azure Container Registry, a service that hosts Docker images from managed image deployments to containers.


High-level architecture of a possible deployment
Invokes ML model via web service
Azure Kubernetes Service Server Cluster
Running Container Instance Scoring Web Service
Running Container Instance Scoring Web Service
Web App AI powered website
When the container starts running, it retrieves the latest model from AML
Azure Container Registry Repository of containers
Azure Machine Learning Custom AI model repository

Section 5: Lab (Managing a Compute Instance)

Machine learning requires several tools to prepare data, and train and deploy models. Most of the work usually takes place within web-based, interactive notebooks, such as Jupyter notebooks. Although notebooks are lightweight and easily run in a web browser, you still need a server to to host them.

So, the setup process for most users is to install several applications and libraries on a machine, configure the environment settings, then loaed any additional resources to begin working within notebooks or integrated development environments (IDEs). All this setup takes time, and there is sometimes a fair amount of troubleshooting involved to make sure you have the right combination of software versions that are compatible with one another.

What if you could use a **ready-made environment** that is **pre-optimized for your machine learning development**?

Azure Machine Learning compute instance (https://docs.microsoft.com/en-gb/azure/machine-learning/concept-compute-instance (https://docs.microsoft.com/en-gb/azure/machine-learning/concept-compute-instance)) provides this type of environment for you, and is **fully managed**, meaning you don't have to worry about setup and applying patches and updates to the underlying virtual machine. Plus, since it is **cloud-based**, you can run it from anywhere and from any machine. All you need to do is **specify the type of virtual machine, including GPUs and I/O-optimized options**, then you have what you need to start working.

The **managed services**, such as **computer instance and compute cluster**, can be used as a training compute target to scale out training resources to handle larger data sets. When you are ready to run your experiments and build your models, you need to specify a compute target. **Compute targets are compute resources where you run your experiments or host your service deployment.** The target may be your local machine or a cloud-based resource. This is another example of where managed services like compute instance and computer cluster really shine.

A managed compute resource is created and managed by Azure Machine Learning. This compute is optimized for machine learning workloads.

Azure Machine Learning compute clusters and compute instances are the only managed computes. Additional managed compute resources may be added in the future.

In this lab, you will explore **different actions** you can take to **manage a compute instance in Azure Machine Learning Studio**.

Steps

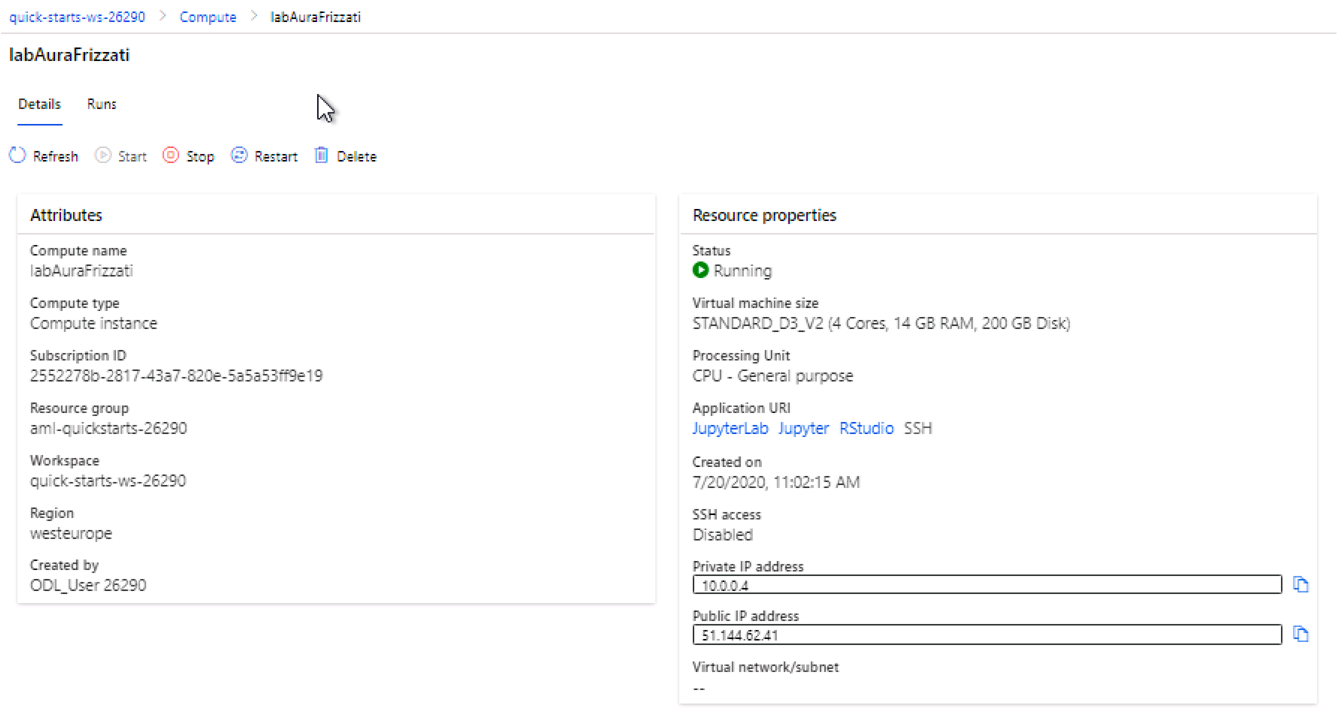
1) **Create New Compute Instance**: Select **Compute** in the left menu panel and then New+ (in this example Virtual Machine size: Standard_D3_v2)
2) **Explore Compute Instances**: Select the radio button next to the name of your compute instance. This will select the instance, as indicated by a checkmark. Selecting your instance in this way enables the toolbar options above that enable you to **Stop, Restart, or Delete** the instance. There are different scenarios in which you will want to perform these actions. Here are the actions you can take on a selected compute instance, and what they do:

- **Stop**: Since the compute instance runs on a virtual machine (VM), you pay for the instance as long as it is running. Naturally, it needs to run to perform compute tasks, but when you are done using it, be sure to stop it with this option to **prevent unnecessary costs**.
- **Restart**: Restarting an instance is sometimes necessary **after installing certain libraries or extensions**. There may be times, however, when the compute instance stops functioning as expected. When this happens, try restarting it before taking further action.
- **Delete**: You can create and delete instances as you see fit. The good news is, **all notebooks and R scripts are stored in the default storage account of your workspace in Azure file share**, within the

"User files" directory. This central storage allows all compute instances in the same workspace to access the same files so **you don't lose them** when you delete an instance you no longer need.

You can select the name of your compute instance and access the **Compute details** blade, revealing useful information about your compute instance.

- **The Attributes** describe the resource details of the compute instance, including the name, type, Azure subscription, the resource group to which it belongs, the Azure Machine Learning workspace that manages it, and the Azure region to which it is deployed. If you need to execute scripts that require details about your compute instance, this is where you can find most of what you need.
- **The Resource** properties show the status and configuration of the compute instance, including links to its applications and public and private endpoints. In this screenshot, you will see that SSH access is disabled. You cannot enable SSH access after creating a compute instance. You can only enable this option at the time of creation. SSH access allows you to securely connect to the VM from a terminal or command window. Use the public IP address to connect via SSH or an integrated development environment (IDE) like Visual Studio Code.



Attributes

Compute name
labAuraFrizzati

Compute type
Compute instance

Subscription ID
2552278b-2817-43a7-820e-5a5a53ff9e19

Resource group
aml-quickstarts-26290

Workspace
quick-starts-ws-26290

Region
westeurope

Created by
ODL_User 26290

Resource properties

Status
Running

Virtual machine size
STANDARD_D3_V2 (4 Cores, 14 GB RAM, 200 GB Disk)

Processing Unit
CPU - General purpose

Application URI
JupyterLab Jupyter RStudio SSH

Created on
7/20/2020, 11:02:15 AM

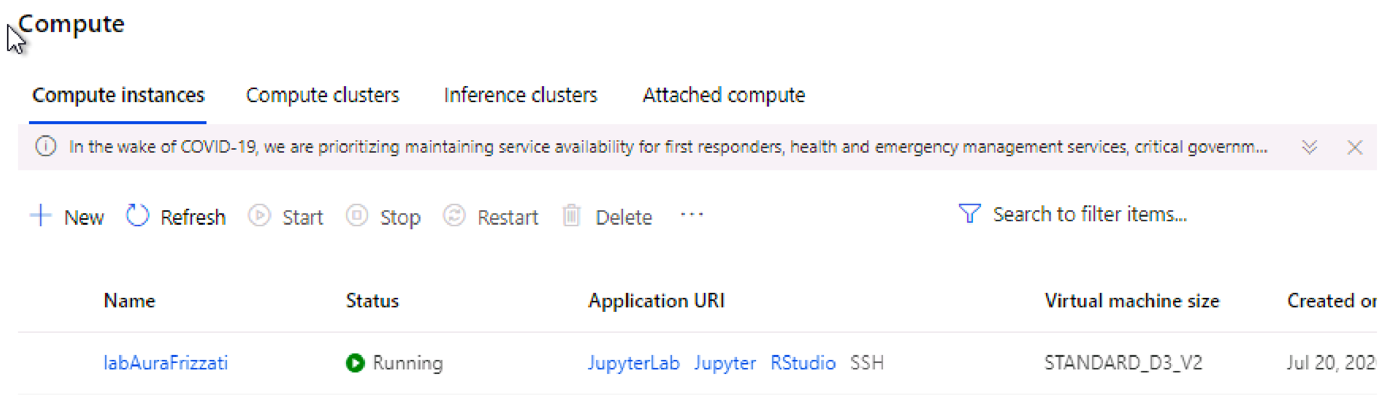
SSH access
Disabled

Private IP address
10.0.0.4

Public IP address
51.144.62.41

Virtual network/subnet
--

The compute instance comes preconfigured with tools and environments that enable you to author, train, and deploy models in a fully integrated notebook experience. You access these environments through the **Application URI links** located in the resource properties (as seen in the previous step), and next to each compute instance in the list.

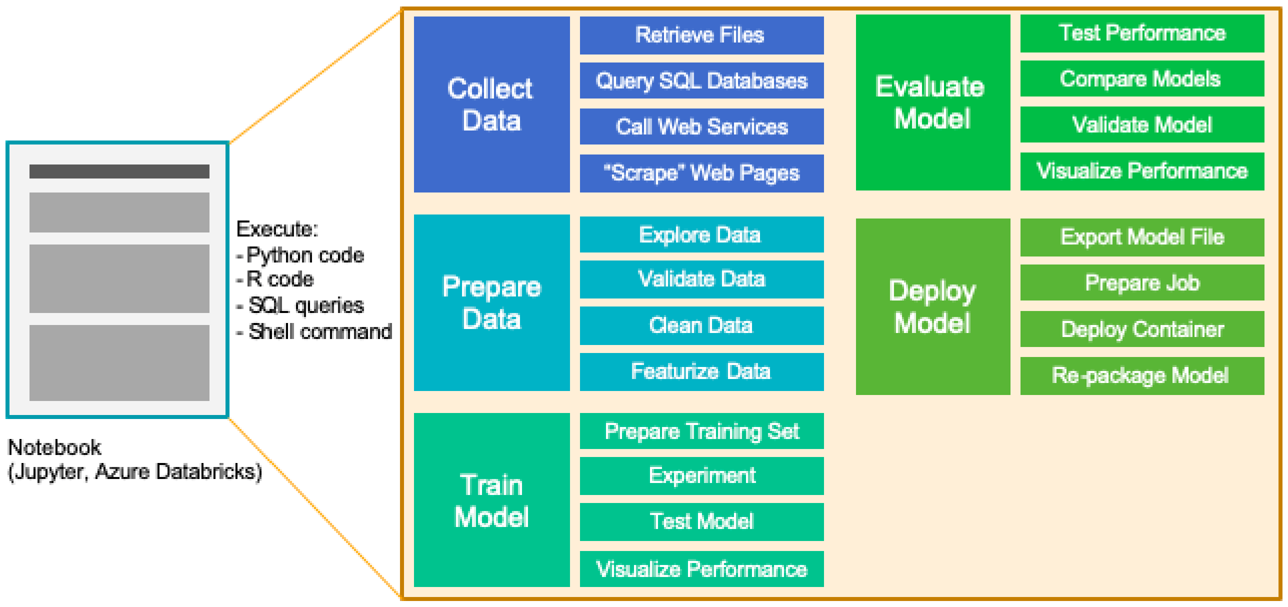


In the wake of COVID-19, we are prioritizing maintaining service availability for first responders, health and emergency management services, critical governm...

Name	Status	Application URI	Virtual machine size	Created or
labAuraFrizzati	Running	JupyterLab Jupyter RStudio SSH	STANDARD_D3_V2	Jul 20, 202

Section 8: Managed Notebook Environments

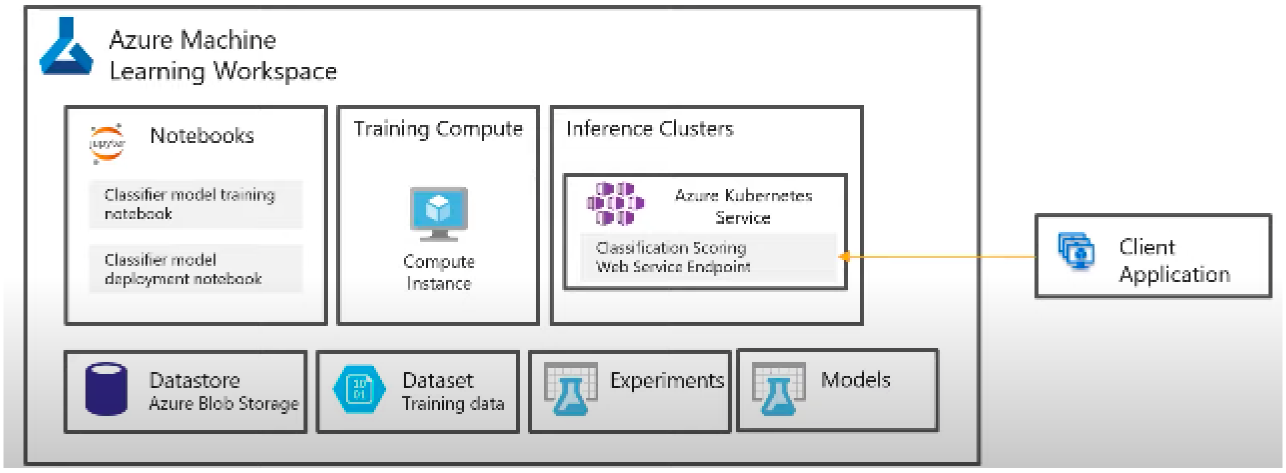
Notebooks are made up of one or more cells that allow for the **execution of the code snippets or commands within those cells**. They **store commands** and the **results** of running those commands. The notebooks can be shared with other users who can also visualise the intended outcome of each cell in the noteebook. In this diagram, you can see that we can use a notebook environment to perform the **five primary stages of model development**:



The most popular notebooks in use today are **Jupyter**, **Databrick** notebooks, **R Markdown** and **Apache Zeppelin**.

Example of how notebook are used for training and deploying models for classification:

- A Jupyter notebbok is created within a compute instance provision in the Azure ML Workspace.
- The notebook uses Azure ML and Python SDK to retrieve the training data (a set of flat files) from the dataset registered with the workspace and whose connection info (e.g. storage account name and keys to the Azure storage) is defined in the datastore.
- The supervised model training executes in the context of an experiment run, which logs the training duration and other metadata about training, collects the performance stats generated during model evaluation performed in the notebook and relates the metadata to the actual model that is uploaded to the model registry.
- A new notebook is used to deploy the model as a web service running in Azure Kubernetes service. This deployment notebook defines the web server logic that retrieves the model from the model registry. loads the model into memory and uses it for inferencing against every request for classification.
- Once deployed, the scoring web service handles classification of input samples arriving from the client app in form of an HTTP request



Section 9: Lab (Train a machine learning model from a managed notebook environment)

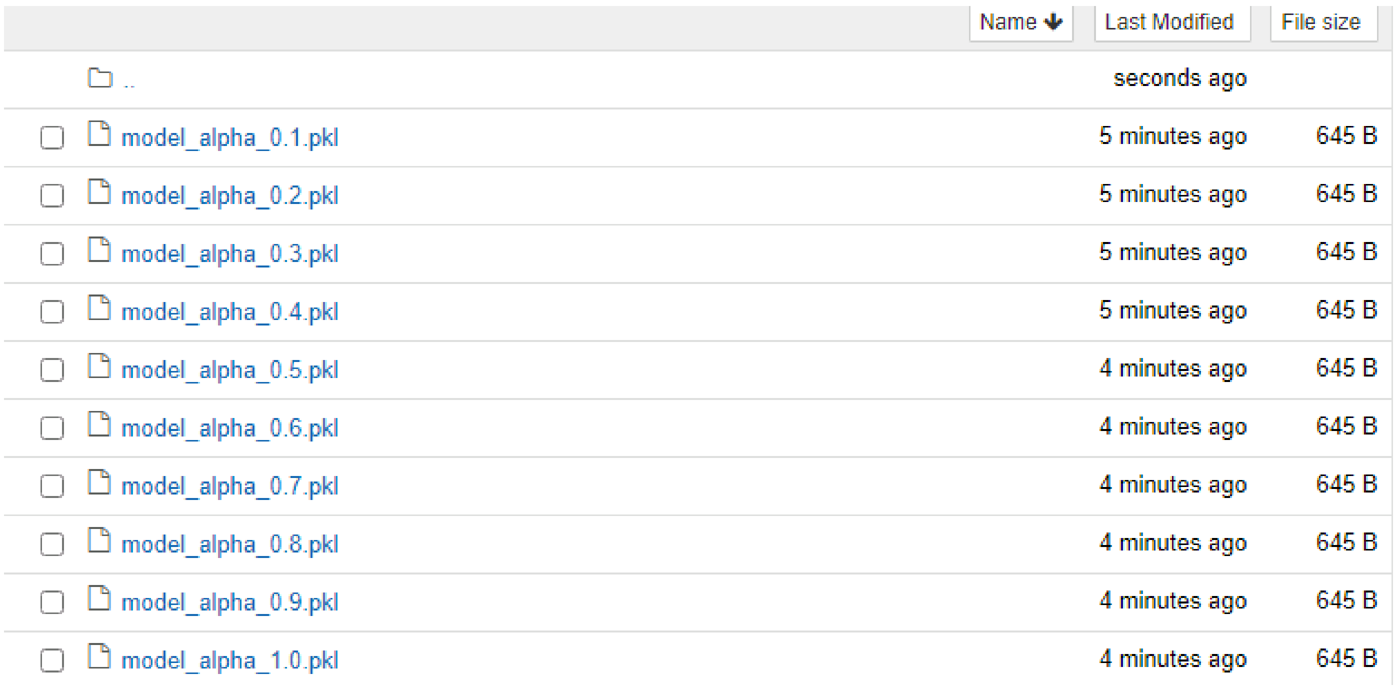
So far, the Managed Services for Azure Machine Learning lesson has covered compute instance and the benefits it provides through its fully managed environment containing everythng you need to run Azure Machine Learning. Now it is time to gain some hands-on experience by putting a **compute instance to work)).

In this lab, you learn the **foundational design patterns in Azure Machine Learning**, and **train a simple scikit-learn model based on the diabetes data set.** After completing this lab, you will have the practical knowledge of the SDK to scale up to developing more-complex experiments and workflows.

In this tutorial, you learn the following tasks:

- Connect your workspace and create an experiment
- Load data and train a scikit-learn model

Import notebook using: git clone https://github.com/solliancenet/udacity-intro-to-ml-labs.git (https://github.com/solliancenet/udacity-intro-to-ml-labs.git)
Follow on here: https://github.com/solliancenet/udacity-intro-to-ml-labs/tree/master/aml-visual-interface/lab-19 (https://github.com/solliancenet/udacity-intro-to-ml-labs/tree/master/aml-visual-interface/lab-19)
Output files generated:



Name	Last Modified	File size
..	seconds ago	
model_alpha_0.1.pkl	5 minutes ago	645 B
model_alpha_0.2.pkl	5 minutes ago	645 B
model_alpha_0.3.pkl	5 minutes ago	645 B
model_alpha_0.4.pkl	5 minutes ago	645 B
model_alpha_0.5.pkl	4 minutes ago	645 B
model_alpha_0.6.pkl	4 minutes ago	645 B
model_alpha_0.7.pkl	4 minutes ago	645 B
model_alpha_0.8.pkl	4 minutes ago	645 B
model_alpha_0.9.pkl	4 minutes ago	645 B
model_alpha_1.0.pkl	4 minutes ago	645 B

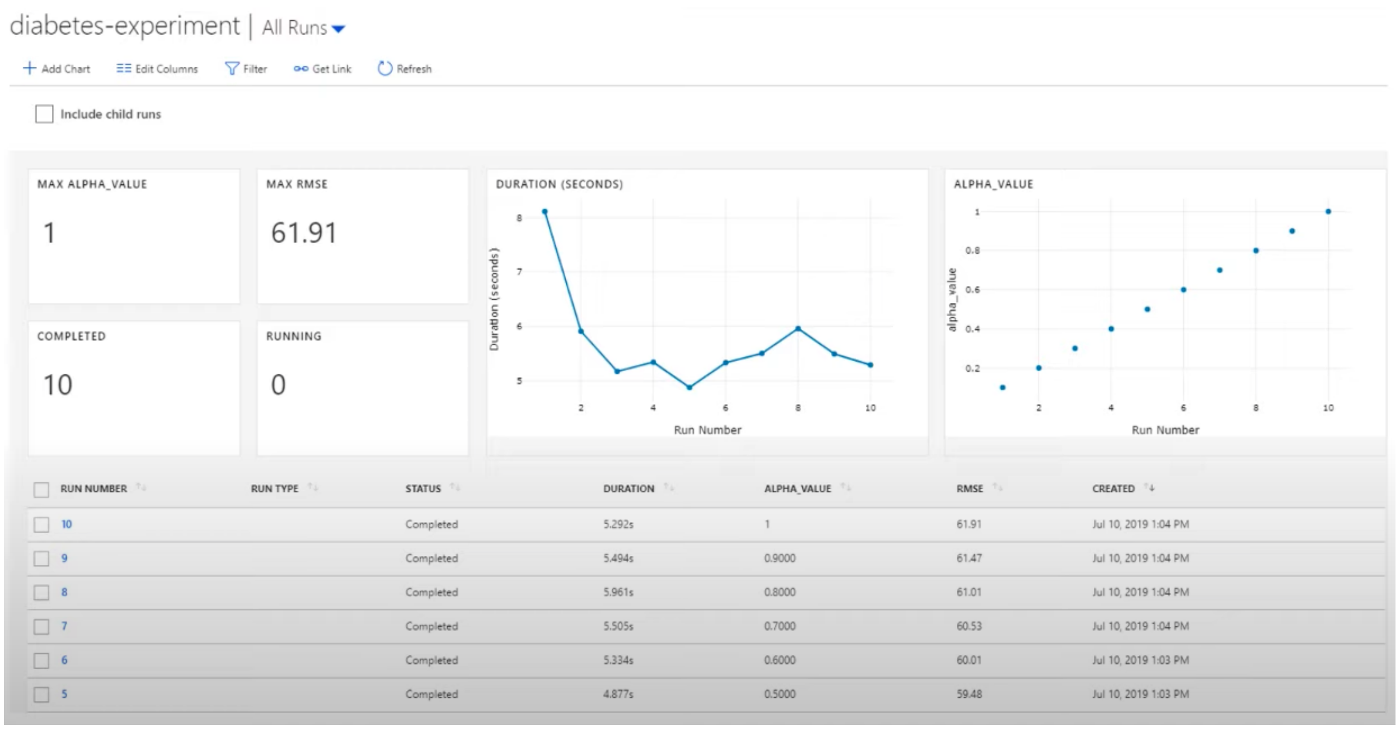
Section 12: Basic Modelling

Training, evaluating, and selecting the right Machine Learning models is at the core of each modern data science process. But what concrete steps do we need to go through to produce a trained model? In this section, we'll look at some important parts of the process and how we can use Azure Machine Learning to carry them out.

An **experiment** is a generic context for handling and organizing runs. Once you have an experiment, you'll want to create **one or more runs within it**, until you identify the best model. You then **register the final model** in a model registry. After registration, you can then **download or deploy the registered model** and receive all the files that were registered.

Experiments

Before you create a new run, you must first create an experiment. Remember, an **experiment** in Azure ML is a **generic context for handling runs**. Think about it as a **logical entity** (a folder) you can use to **organize your model training processes.**
Screenshot of an example main Experiments page:



MAX ALPHA_VALUE	MAX RMSE	COMPLETED	RUNNING
1	61.91	10	0

RUN NUMBER	RUN TYPE	STATUS	DURATION	ALPHA_VALUE	RMSE	CREATED
10		Completed	5.292s	1	61.91	Jul 10, 2019 1:04 PM
9		Completed	5.494s	0.9000	61.47	Jul 10, 2019 1:04 PM
8		Completed	5.961s	0.8000	61.01	Jul 10, 2019 1:04 PM
7		Completed	5.505s	0.7000	60.53	Jul 10, 2019 1:04 PM
6		Completed	5.334s	0.6000	60.01	Jul 10, 2019 1:03 PM
5		Completed	4.877s	0.5000	59.48	Jul 10, 2019 1:03 PM

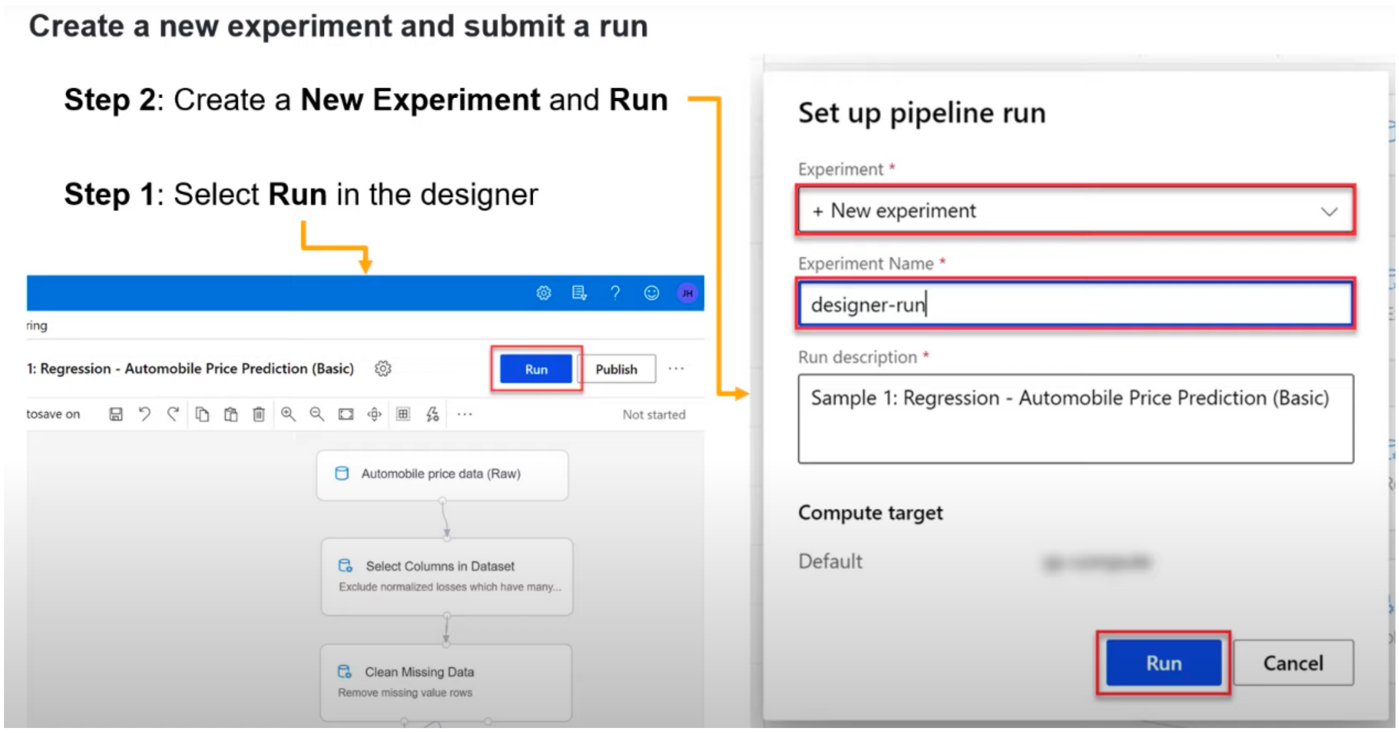
Any custom log value (such as alpha value or RMSE become fields for each experiment run).
You can also access from here detailed info for each individual run.

Runs

Once you have an experiment, you can create runs within that experiment. As we discussed above, **model training runs** are what you use to **build the trained model.** A run contains:

- **all artifacts associated with the training process**, like **output files, metrics, logs**
- a **snapshot of the directory that contains your scripts.**

A **run configuration** is a **Set of instructions** that defines **how a script should run** in a **specific compute target.** It includes a set of behaviour definitions (such as whether to use existing Python environments or to use a Conda environment built from a specification)

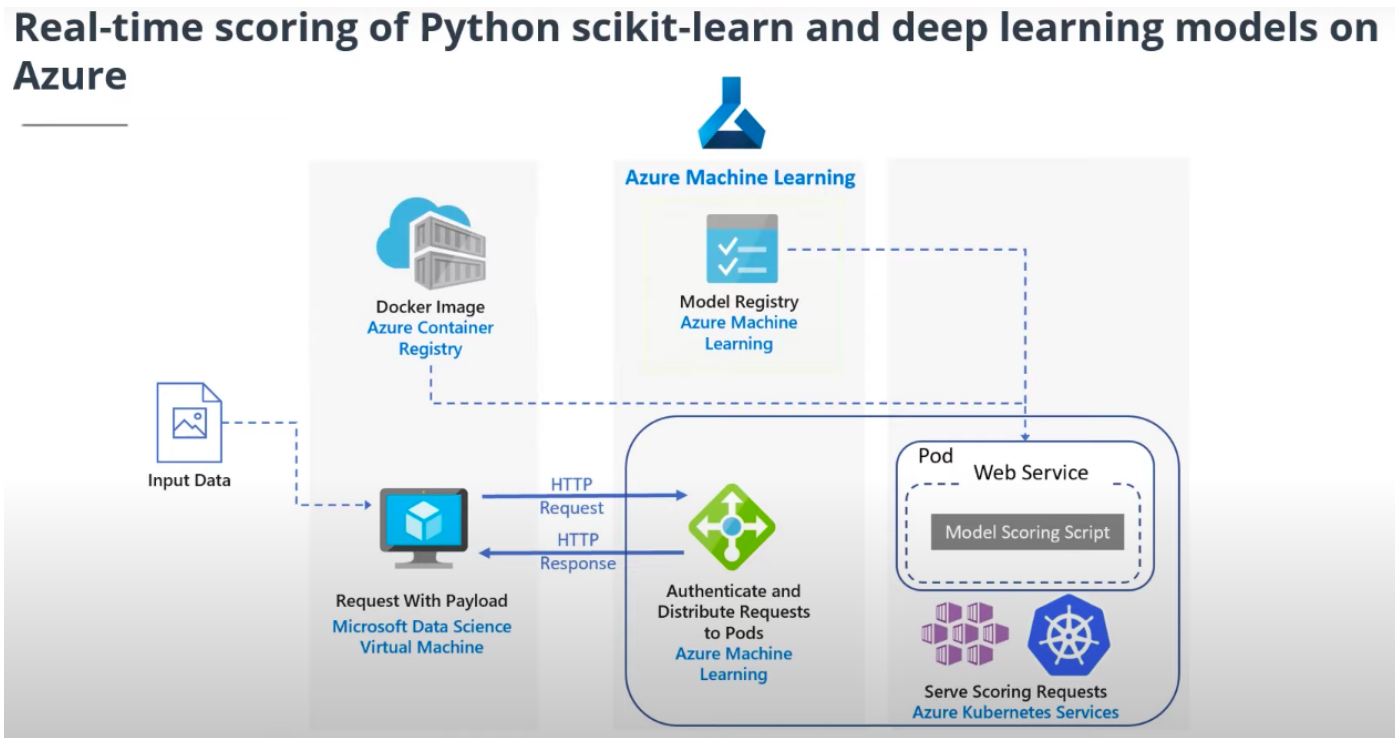

Create a new experiment and submit a run
Step 2: Create a New Experiment and Run
Step 1: Select Run in the designer
1: Regression - Automobile Price Prediction (Basic)
Run
Publish
Not started
Automobile price data (Raw)
Select Columns in Dataset
Exclude normalized losses which have many...
Clean Missing Data
Remove missing value rows
Set up pipeline run
Experiment *
+ New experiment
Experiment Name *
designer-run
Run description *
Sample 1: Regression - Automobile Price Prediction (Basic)
Compute target
Default
Run
Cancel

Models

A run is used to produce a model. Essentially, a **model** is a **piece of code that takes an input and produces output**. To get a model, we start with a more general algorithm. By **combining** this **algorithm** with the **training data**, as well as by **tuning the hyperparameters**, we produce a more specific function that is optimized for the particular task we need to do.
Put concisely: ***Model = algorithm + data + hyperparameters***

Azure ML is **library agnostic** (i.e. you can use any popular ML library, e.g. scikit-learn, XGBoost, PyTorch, TensorFlow and Theano).

Reference architecture on how to deploy Python models as web-services (hosted in AKS = Azure Kubernetes Service) to make real-time prediction using the Azure ML service. The diagram covers 2 scenarios: deploying regular Python models and the specific requirements of deploying deep learning models:



1) The trained model is registered to the **ML Model Registry**.
2) Azure ML registers the trained model as a **scoring web service docker image** in the **Azure Container Registry** (check this link for getting more info on Docker [https://stackify.com/docker-image-vs-container-everything-you-need-to-know/ (https://stackify.com/docker-image-vs-container-everything-you-need-to-know/)](https://stackify.com/docker-image-vs-container-everything-you-need-to-know/)). Azure Container Registry is a service that hosts docker images for managed image deployments to containers. 3) The docker image is deployed to the **Azure Kubernetes Service** (**AKS**) as a web service.
4) The **client sends an HTTP post request** with the **encoded question data**. The **web server** created by Azure ML **extracts** the **question** from the request and the **data** is **sent to the model for scoring**.
5) The **matching data with the associated scores** are **returned to the client**.

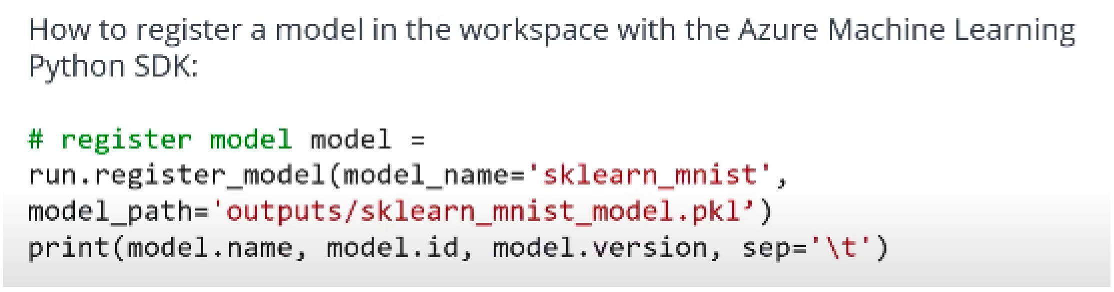
This **architecture** consists of the **following components**:

- **Azure ML**: a Cloud service used to train, deploy, automate and manage ML models, all at the broad scale that the Cloud provides. In this architecture it is used to manage the model deployment as well as the authentication, routing and load balancing of the web service.
- **Virtual Machines** (**VMs**): for example, a local or in the Cloud device that can send an HTTP request.
- **AKS**: used to deploy the application on a Kubernetes cluster, simplyfying the deployment and operations of Kubernetes. The cluster can be configured using CPU-only VMs (regular python models) or GPU-anabled VMs for deep learning models.
- **Azure Container Registry**: enables storage of images for all types of docker container deployments (including DC OS, Docker Swarm and Kubernetes). The scoring images are deployed as containers on AKS and use to run the scoring script; they were created using Azure ML from the trained model and scoring script and the pushed to Azure Container Registry.

Model Registry

A **registered model** is a **logical container** for all the **files** that make up your **model**. Once we have a trained model, we can turn to the **model registry**, which **keeps track of all registered models in an Azure Machine Learning workspace**.

- Models are **identified** by their **name** and **version**. Also **additional metadata tags** can be provided when registreing a model (they can be used for searching a model).
- You **can't** delete a registered model that is being used in an active deployment.
- You can use Azure ML **Studio Designer** and create a deployment (the model is automatically registered for you)
- When using the **SDK**, you can register your model with code. "**outputs**" os a **special directory** as all its **content is automaticallly uploaded into the workspace**.



How to register a model in the workspace with the Azure Machine Learning Python SDK:

```
# register model model =
run.register_model(model_name='sklearn_mnist',
model_path='outputs/sklearn_mnist_model.pkl’)
print(model.name, model.id, model.version, sep='\t')
```

Note that models are either produced by a Run or originate from outside of Azure Machine Learning (and are made available via model registration).

Section 13: Lab (Explore Experiments and Runs)

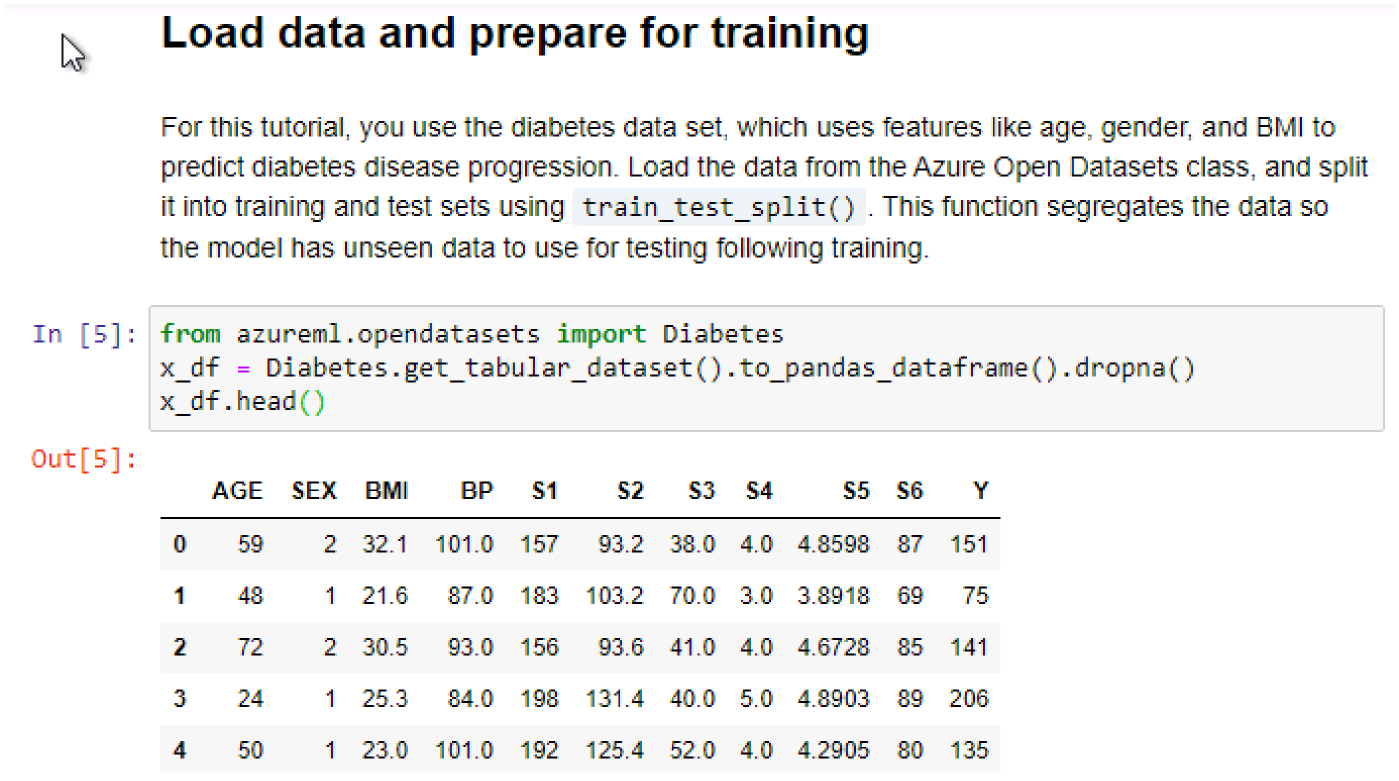
In the previous lab, you executed a Jupyter notebook that trained a model through a series of 10 different runs, each with a different alpha hyperparameter applied. These runs were created within the experiment you created at the beginning of the notebook. Because of this, Azure Machine Learning logged the details so you can review the result of each run and see how the alpha value is different between the them.

In this lab, you **view the experiments and runs executed by a notebook.** In the first part of the lab, you will use a notebook to create and run the experiments. In the second part of the lab, you will navigate to the **Experiments blade** in Azure Machine Learning Studio. Here you see **all the individual runs in the experiment**. Any **custom-logged values** (**alpha_value** and **rmse**, in this case) become **fields for each run**, and also become available for the charts and tiles at the top of the experiment page. To add a logged metric to a chart or tile, hover over it, click the edit button, and find your custom-logged metric.

When training models at scale over hundreds and thousands of separate runs, this page makes it easy to see every model you trained, specifically how they were trained, and how your unique metrics have changed over time.

Code to follow here: https://github.com/solliancenet/udacity-intro-to-ml-labs/blob/master/aml-visual-interface/lab-20/README.md (https://github.com/solliancenet/udacity-intro-to-ml-labs/blob/master/aml-visual-interface/lab-20/README.md)

Input dataset:



Load data and prepare for training

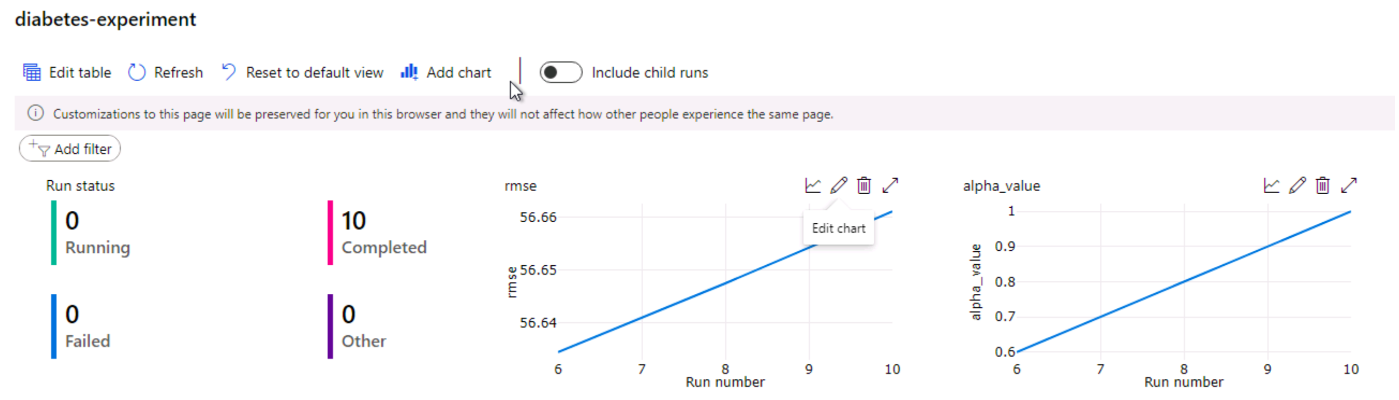
For this tutorial, you use the diabetes data set, which uses features like age, gender, and BMI to predict diabetes disease progression. Load the data from the Azure Open Datasets class, and split it into training and test sets using `train_test_split()`. This function segregates the data so the model has unseen data to use for testing following training.

In [5]:
```
from azureml.opendatasets import Diabetes
x_df = Diabetes.get_tabular_dataset().to_pandas_dataframe().dropna()
x_df.head()
```

Out[5]:

	AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6	Y
0	59	2	32.1	101.0	157	93.2	38.0	4.0	4.8598	87	151
1	48	1	21.6	87.0	183	103.2	70.0	3.0	3.8918	69	75
2	72	2	30.5	93.0	156	93.6	41.0	4.0	4.6728	85	141
3	24	1	25.3	84.0	198	131.4	40.0	5.0	4.8903	89	206
4	50	1	23.0	101.0	192	125.4	52.0	4.0	4.2905	80	135

Experiments view:



diabetes-experiment

Customizations to this page will be preserved for you in this browser and they will not affect how other people experience the same page.

Run status

0 Running

10 Completed

0 Failed

0 Other

Section 15: Advanced Modelling

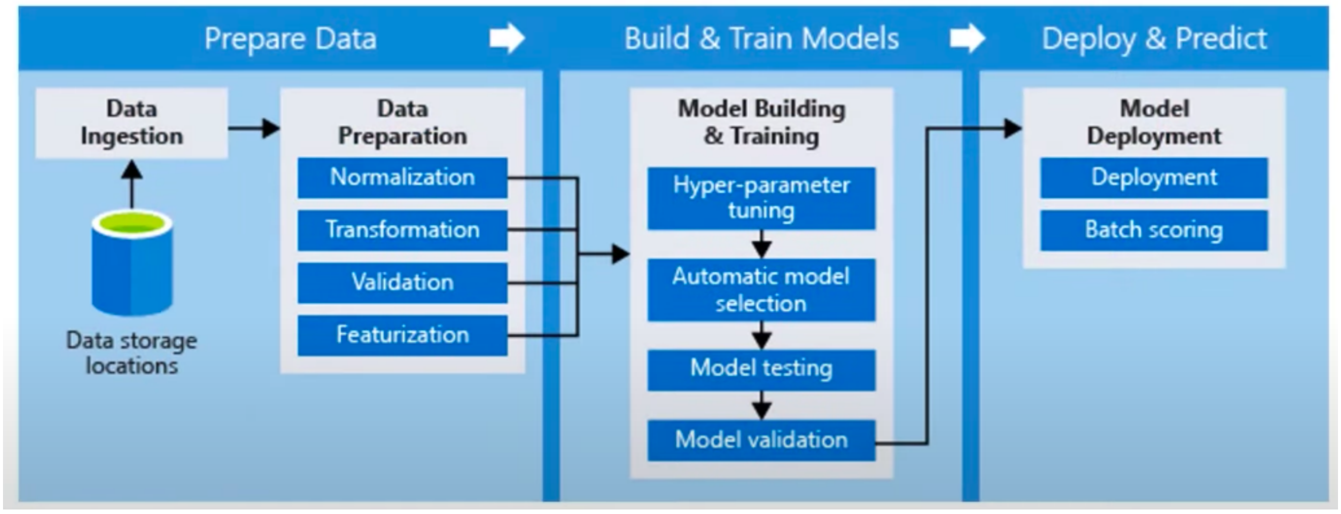
Any non-trivial project that includes data science components will need to take advantage of a combination of features from these two major platforms:

- **Azure ML**
- **Azure DevOps**: the DevOps applied to data science solutions is commonly referred to as ML operations (**MLOps**)

Machine Learning Pipelines

As the process of building your models becomes more complex, it becomes more important to get a handle on the steps to prepare your data and train your models in an organized way. In these scenarios, there can be many steps involved in the **end-to-end process**, including:

- **Data ingestion**
- **Data preparation**
- **Model building & training**
- **Model deployment**

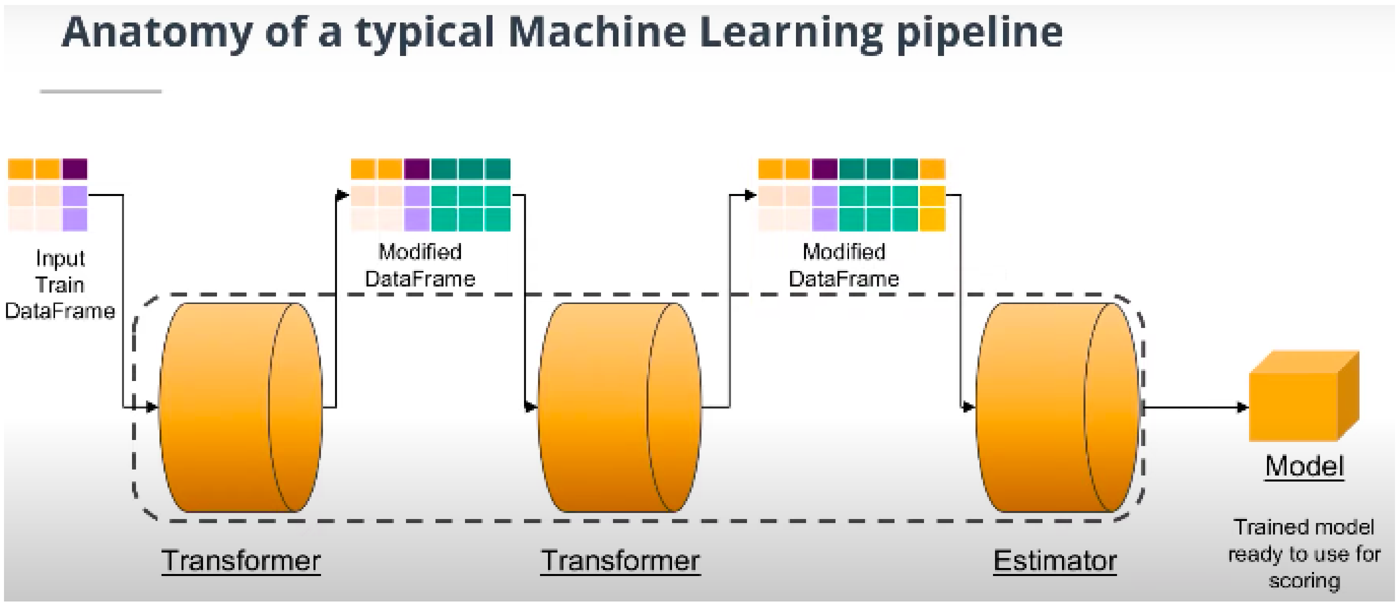


These steps are organized into **machine learning pipelines**, which are cyclical and iterative/repeatable in nature and facilitate continuous improvement of model performance, deployment and making inferences over the best performing model to date.

ML pipelines also need to be **modular**, thus their steps are re-usable and can re-run in subsequent steps if the output of the steps have been changed. They also allow data scientists to **collaborate** while they are **working on separate areas** of the ML workflow.

You can use tools like the Azure ML Designer or the Azure ML SDK for Python to create re-usable ML pipelines that optimise your workflow.

In the following diagram, the **pipeline** is **represented** by the **dashed lines** forming a rectangle around the transformers and estimator. In a typical scenario, you have **multiple transformer steps** and an **estimator** at the **end** which processes the transformed dataframe using the **selected algorithm** to train the model.

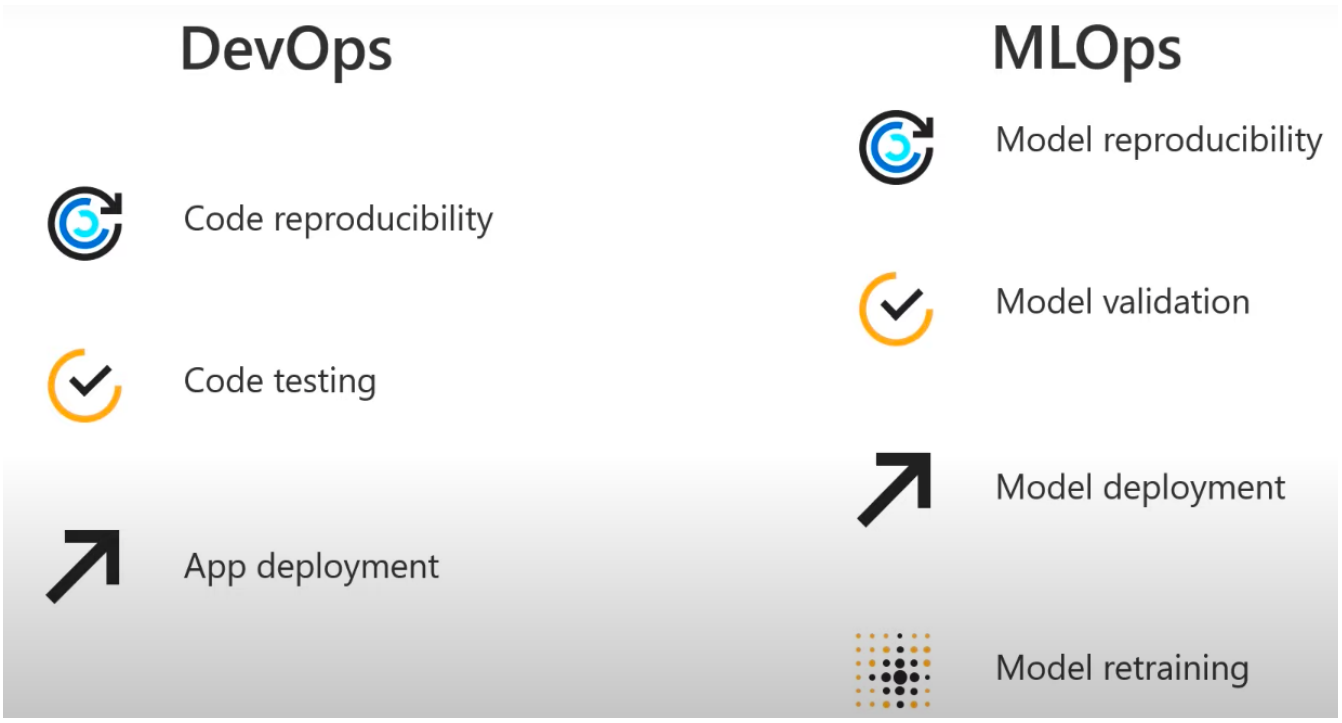


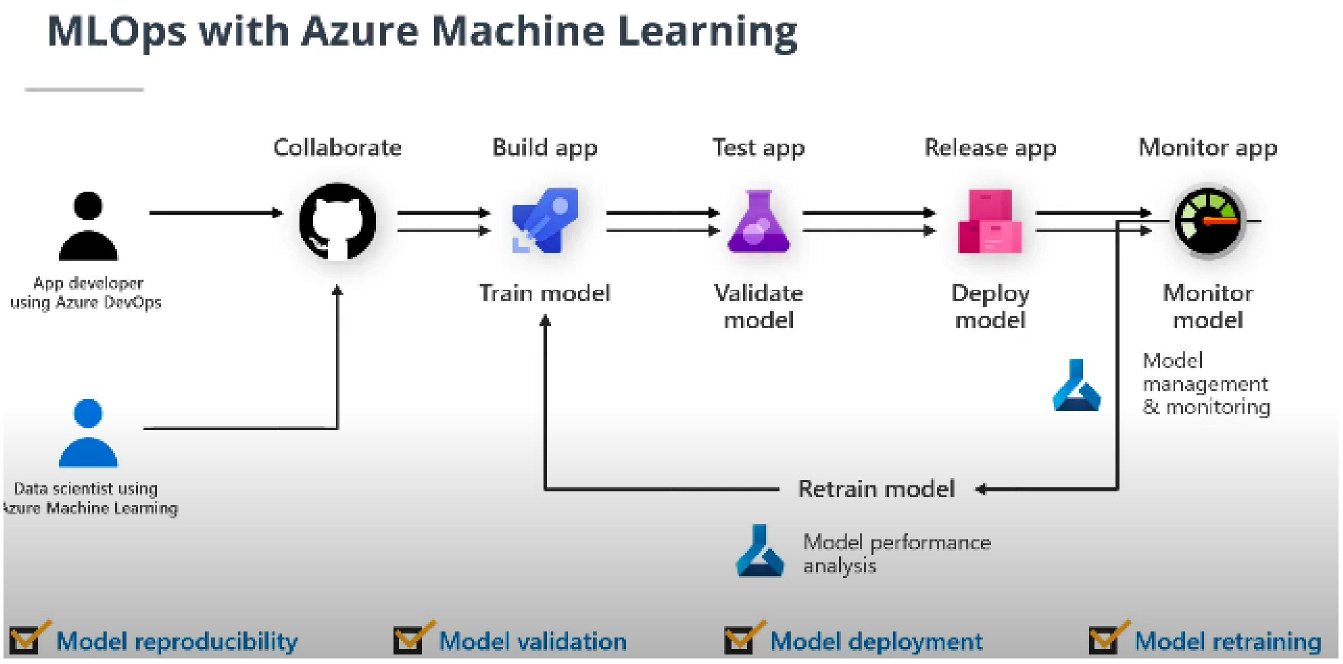
MLOps: Creating Automatic End-to-End Integrated Processes

As we said earlier, we don't want all the steps in the machine learning pipeline to be manual, rather, we want to develop processes that use **automated builds and deployments**. The general term for this approach is **DevOps**; when applied to machine learning, we refer to the **automation of machine learning pipelines** as **MLOps**.

The data science world has additional challenges to those present in classic software development: e.g. **versioning** includes not only the source code but also the version of the input dataset on which the model was trained. **MLOps** is the discipline that deals with the **application of classical DevOps principles** to projects that have **at least one data science component**, i.e. DevOPs for AI. Its most important aspects include:

- **Automate** the **end-to-end ML lifecycle**
- **Monitor ML processes**
- **Capture traceability data**





Section 17: Operationalizing Models

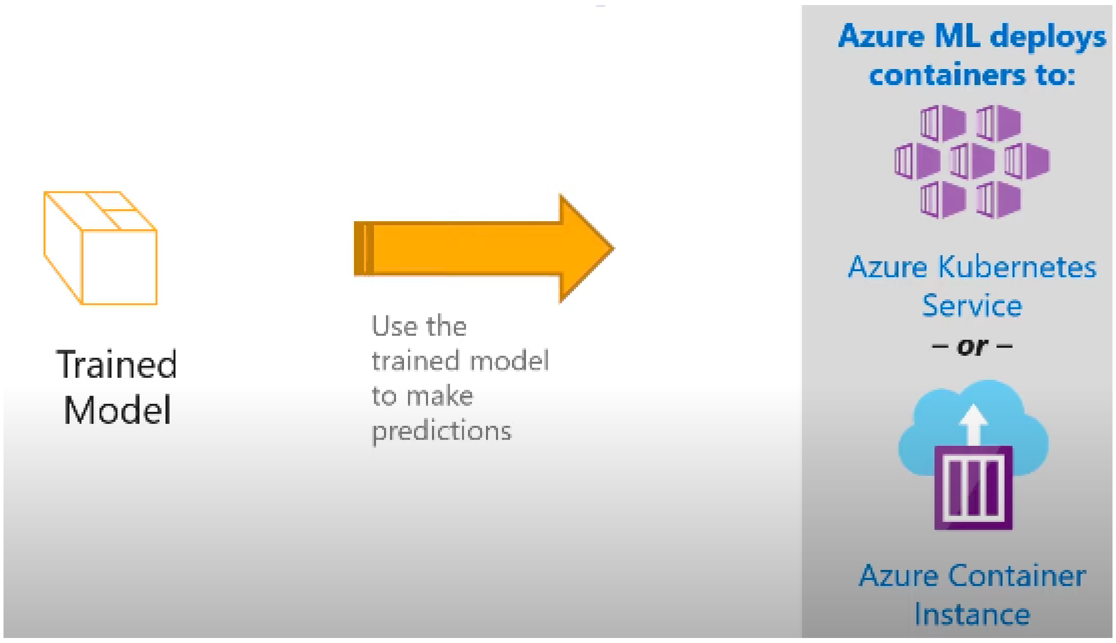
After you have trained your machine learning model and evaluated it to the point where you are ready to use it outside your own development or test environment, you need to **deploy** it somewhere. Another term for this is **operationalization**.

Real-time Inferencing

The model training process can be very compute-intensive, with training times that can potentially spann across many hours, days, or even weeks. A **trained model**, on the other hand, is used to **make decisions on new data quickly**. In other words, it infers things about new data it is given based on its training. Making these decisions on new data on-demand is called real-time inferencing.

Typical Real-Time Model Deployment:

- Get the model file (any format)
- Create a scoring script (.py)
- Optionally, create a schema file describing the web service input (.json)
- Create **a real-time scoring web service**
- Call the web service from your applications
- Repeat the process each time you re-train the model Azure ML Service **simplifies** this process by providing tools and a framework that automate these steps and deploy the model on one or more targets (e.g. as a web service on AKS; or Azure Container Instances if you need to quickly deploy/validate your model or testing a model under development).


Trained Model
Use the trained model to make predictions
Azure ML deploys containers to:
Azure Kubernetes Service
– or –
Azure Container Instance

Options for creating real-time inferencing environments:

- **Create manually** from the **Azure ML Studio user interface**
- **Create programmatically** using **code** (i.e. **Azure ML Python SDK**)

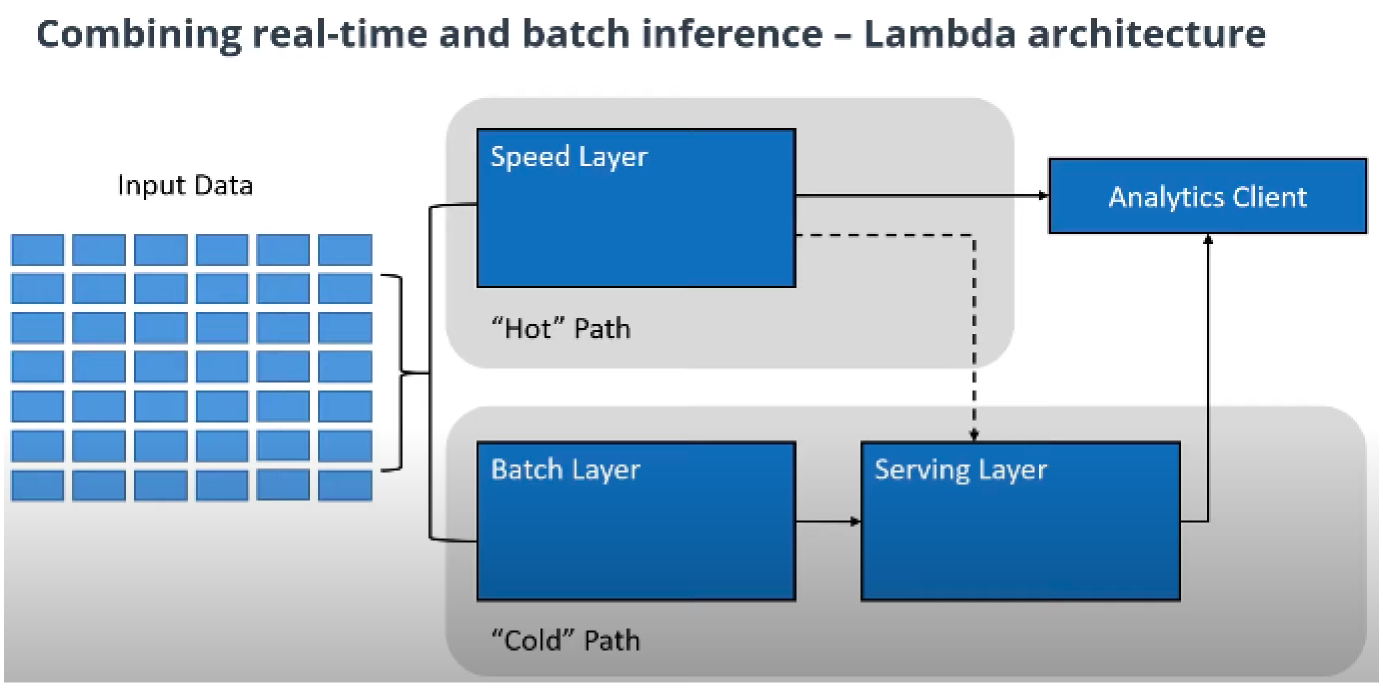
Batch Inferencing

Unlike real-time inferencing, which makes predictions on data as it is received, batch inferencing is **run on large quantities (batches)** of existing data. Typically, batch inferencing is **run on a recurring schedule against data stored in a database or other data store.** The resulting predictions are then written to a datastore for later use by applications, data scientists, developers, end users, ... It typically involves **latency requirements** of **hours** or **days** (predictions on a large data volume **asynchronously**). It doesn't require using train models deployed on restful web services (as done for real-time inferencing), instead it can take advantage of a host of high throughput and scalable compute targets.
Consider using batch inferecing when:

- You do not need to make actions/predictions in real time
- Inferecing results can be persisted (on datastores for later use)
- Post-processing or analysis on the predictions is needed before exposing the results to further use
- Inferencing is complex (it requires long-runs for scoring or to scale out for computer resources to perform parallel processing).

In many common real-world scenarios, **predictions** are needed on both **newly arriving** and **existing historical data.** In these cases, the **lamda architecture** provides a good solution, where **ingested data** is processed at **two different speeds**:

- A **hot path** that tries to make **predictions against the data in real time.** The predictions are made available directly to the analytics client and also stored in the serving layer.
- A **cold path** that makes **predictions** in a **batch fashion.** The predictions for this path are **stored** in the **serving layer** (typically a database) for later use by the analytics client



All information within the serving layer can be used for reporting and auditing

Section 18: Lab (Deploy a Model as a Webservice)

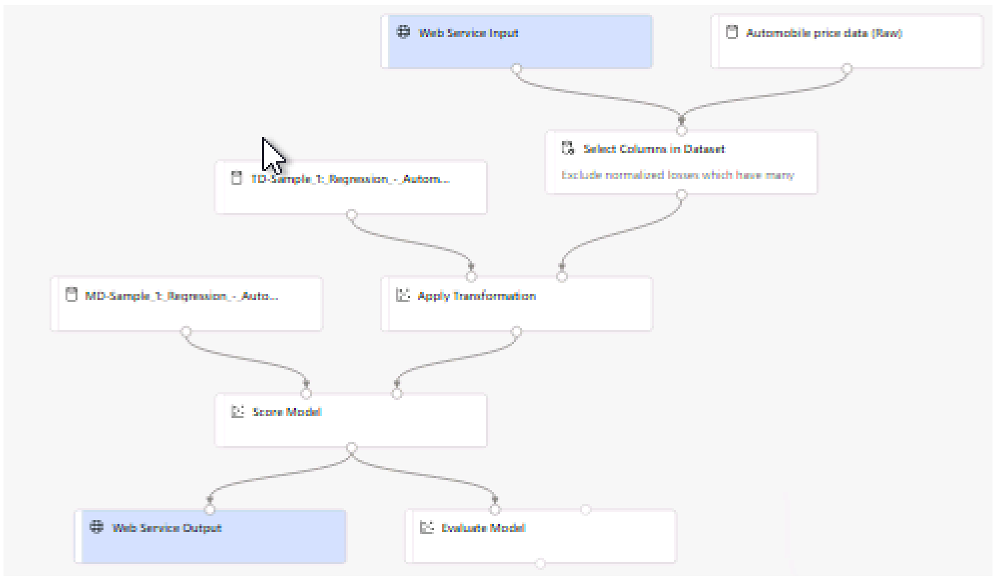
In previous lessons, we spent much time talking about training a machine learning model, which is a multi-step process involving data preparation, feature engineering, training, evaluation, and model selection. The model training process can be very compute-intensive, with training times spanning across many hours, days, or weeks depending on the amount of data, type of algorithm used, and other factors. A trained model, on the other hand, is used to make decisions on new data quickly. In other words, it infers things about new data it is given based on its training. Making these decisions on new data on-demand is called real-time inferencing.

Overview In this lab, you learn how to **deploy a trained model** that can be used as a **webservice**, hosted on an **Azure Kubernetes Service (AKS) cluster**. This process is what enables you to use your model for **real-time inferencing**.

The Azure Machine Learning designer simplifies the process by enabling you to train and deploy your model without writing any code.

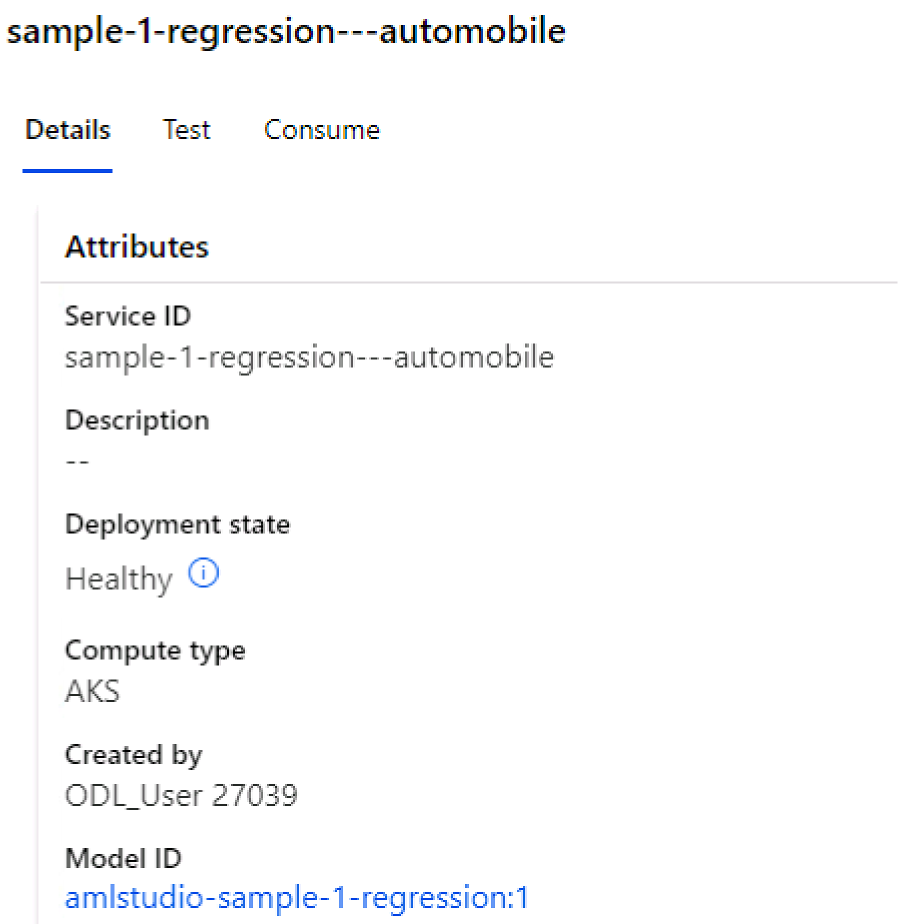
Real-time inference pipeline:

After you have run your model, select from the top **Create inference pipeline**, then select **Real-time inference pipeline** from the list to create a new inference pipeline.


Web Service Input
Automobile price data (Raw)
Select Columns in Dataset
Exclude normalized losses which have many
TD-Sample_1:_Regression_-_Autom...
MD-Sample_1:_Regression_-_Auto...
Apply Transformation
Score Model
Web Service Output
Evaluate Model

Deploy web service on Azure Kubernetes Service compute

- After the inference pipeline run is finished, select **Deploy** to open the **Set up real-time endpoint editor**.
- To view the deployed web service, select the **Endpoints section** in your Azure Portal Workspace.



sample-1-regression---automobile

Details Test Consume

Attributes

Service ID
sample-1-regression---automobile

Description
--

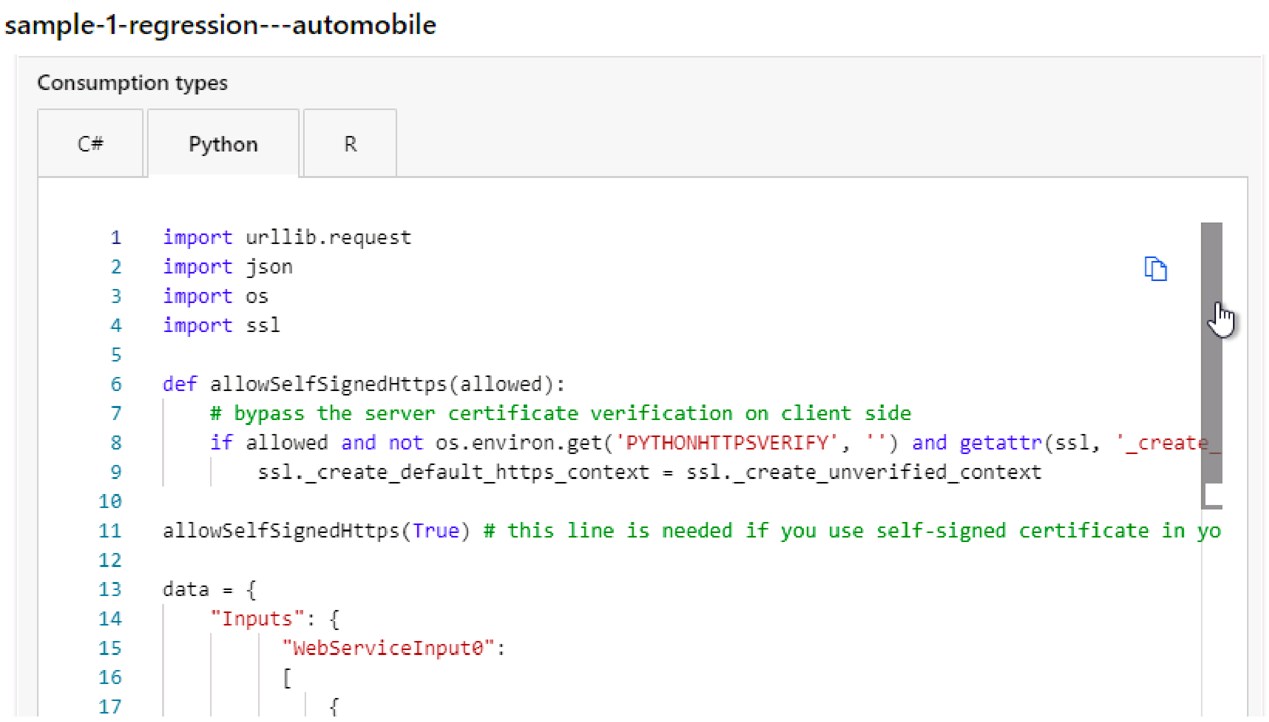
Deployment state
Healthy

Compute type
AKS

Created by
ODL_User 27039

Model ID
amlstudio-sample-1-regression:1

- Select the **Consume tab** to observe the following information:
  - Basic consumption info displays the REST endpoint, Primary key, and Secondary key.
  - Consumption option shows code samples in C#, Python, and R on how to call the endpoint to consume the webservice.


sample-1-regression---automobile

Consumption types

```
import urllib.request
import json
import os
import ssl

def allowSelfSignedHttps(allowed):
    # bypass the server certificate verification on client side
    if allowed and not os.environ.get('PYTHONHTTPSVERIFY', '') and getattr(ssl, '_create_
        ssl._create_default_https_context = ssl._create_unverified_context

allowSelfSignedHttps(True) # this line is needed if you use self-signed certificate in yo

data = {
    "Inputs": {
            "WebServiceInput0":
            [
                {
```

Section 21: Programmatically Accessing Managed Services

Azure Machine Learning provides a **code-first experience** via the **Azure Machine Learning SDK for Python** (https://docs.microsoft.com/en-us/python/api/overview/azure/ml/?view=azure-ml-py (https://docs.microsoft.com/en-us/python/api/overview/azure/ml/?view=azure-ml-py)). Using the SDK, you can start training your models on your local machine and then scale out to use Azure Machine Learning compute resources. This allows you to train better performing, highly accurate machine learning models.

You can interact with the service in any Python environment (including Jupyter notebooks, Visual Studio code and any Python IDE). You can start training your models on your local machine and then scale out using the Azure ML Compute resources.

Azure Machine Learning service supports many of the popular **open-source machine learning and deep learning Python packages** that we discussed earlier in the course, such as:

- Scikit-learn
- Tensorflow
- PyTorch
- Keras

Key areas of the **SDK** include:

- **Manage datasets**
- **Organise** and **monitor experiments**
- **Model training**: typically, the model traiing scripts is going to generate output and log metrics to the run as the model is trained. These outputs can be monitored in real time within a Python notebook in 2 main ways:
  - Wait for run completion and set ***run.wait_for_completion(show_output = True)***. The output gets progressively displayed as it is generated.
  - Using the Jupyter Notebook widget RunDetails:
    from azureml.widgets import RunDetails
    RunDetails(run).show()
- **Automated ML**

- **Model deployment**

Section 23: Lab (Training and Deploying a Model from a Notebook Running in a Compute Instance)

So far, the Managed Services for Azure Machine Learning lesson has covered compute instance and the benefits it provides through its fully managed environment containing everything you need to run Azure Machine Learning.

The compute instance provides a comprehensive set of a capabilities that you can use directly within a python notebook or python code including:

Creating a Workspace that acts as the root object to organize all artifacts and resources used by Azure Machine Learning. Creating Experiments in your Workspace that capture versions of the trained model along with any desired model performance telemetry. Each time you train a model and evaluate its results, you can capture that run (model and telemetry) within an Experiment. Creating Compute resources that can be used to scale out model training, so that while your notebook may be running in a lightweight container in Azure Notebooks, your model training can actually occur on a powerful cluster that can provide large amounts of memory, CPU or GPU. Using Automated Machine Learning (AutoML) to automatically train multiple versions of a model using a mix of different ways to prepare the data and different algorithms and hyperparameters (algorithm settings) in search of the model that performs best according to a performance metric that you specify. Packaging a Docker Image that contains everything your trained model needs for scoring (prediction) in order to run as a web service. Deploying your Image to either Azure Kubernetes or Azure Container Instances, effectively hosting the Web Service. Overview In this lab, you start with a model that was trained using Automated Machine Learning. Learn how to use the Azure ML Python SDK to register, package, and deploy the trained model to Azure Container Instances (ACI) as a scoring web service. Finally, test the deployed model (1) by make direct calls on service object, (2) by calling the service end point (Scoring URI) over http.

Jupyter notebook from: [https://github.com/solliancenet/udacity-intro-to-ml-labs/blob/master/aml-visual-interface/lab-22/notebook/deployment-with-AML.ipynb (https://github.com/solliancenet/udacity-intro-to-ml-labs/blob/master/aml-visual-interface/lab-22/notebook/deployment-with-AML.ipynb)](https://github.com/solliancenet/udacity-intro-to-ml-labs/blob/master/aml-visual-interface/lab-22/notebook/deployment-with-AML.ipynb)

In [6]:
```
# Display a summary of the current environment
import pandas as pd
output = {}
output['SDK version'] = azureml.core.VERSION
output['Workspace'] = ws.name
output['Resource Group'] = ws.resource_group
output['Location'] = ws.location
pd.set_option('display.max_colwidth', -1)
pd.DataFrame(data=output, index=['']).T
```

Out[6]:

SDK version	1.9.0
Workspace	quick-starts-ws-26821
Resource Group	aml-quickstarts-26821
Location	westus2

Register Model

Azure Machine Learning provides a Model Registry that acts like a version controlled repository for each of your trained models. To version a model, you use the SDK as follows. Run the following cell to register the best model with Azure Machine Learning.

In [7]:
```
# register the model for deployment
model = Model.register(model_path = model_path, # this points to a local file
                       model_name = "nyc-taxi-automl-predictor", # name the mode
                       tags = {'area': "auto", 'type': "regression"},
                       description = "NYC Taxi Fare Predictor",
                       workspace = ws)

print()
print("Model registered: {} \nModel Description: {} \nModel Version: {}".format(
```

```
Registering model nyc-taxi-automl-predictor

Model registered: nyc-taxi-automl-predictor
Model Description: NYC Taxi Fare Predictor
Model Version: 1
```

Make direct calls on the service object

In [12]:
```
import json

data1 = [1, 2, 5, 9, 4, 27, 5, 'Memorial Day', True, 0, 0.0, 0.0, 65]

data2 = [[1, 3, 10, 15, 4, 27, 7, 'None', False, 0, 2.0, 1.0, 80],
         [1, 2, 5, 9, 4, 27, 5, 'Memorial Day', True, 0, 0.0, 0.0, 65]]

result = aci_service.run(json.dumps(data1))
print('Predictions for data1')
print(result)

result = aci_service.run(json.dumps(data2))
print('Predictions for data2')
print(result)
```

```
Predictions for data1
[22.378127578013636]
Predictions for data2
[39.714977643580596, 22.378127578013636]
```

Consume the Deployed Web Service

Execute the code below to consume the deployed webservice over HTTP.

In [13]:
```
import requests

url = aci_service.scoring_uri
print('ACI Service: {} scoring URI is: {}'.format(service_name, url))
headers = {'Content-Type':'application/json'}

response = requests.post(url, json.dumps(data1), headers=headers)
print('Predictions for data1')
print(response.text)
response = requests.post(url, json.dumps(data2), headers=headers)
print('Predictions for data2')
print(response.text)
```

```
ACI Service: nyc-taxi-srv scoring URI is: http://84edf975-3954-42ef-875f-3cf00c
a9c491.westus2.azurecontainer.io/score
Predictions for data1
"[22.378127578013636]"
Predictions for data2
"[39.714977643580596, 22.378127578013636]"
```