Table of Contents

Lesson 6 – Managed Services for Machine Learning

Chapter 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 [Clusters of computers running in Cloud, which provides raw computing power required by ML workloads.]** made available through managed services, including:

- Training compute – Training model
- Inferencing compute – Operationalizing model
- Notebook environments – Run notebook based codes

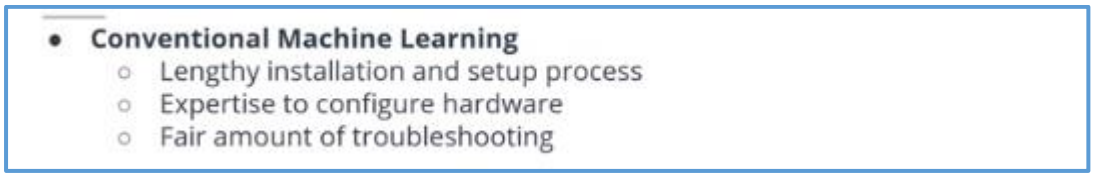
You will also study the main concepts involved in the **modelling process**, including:

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

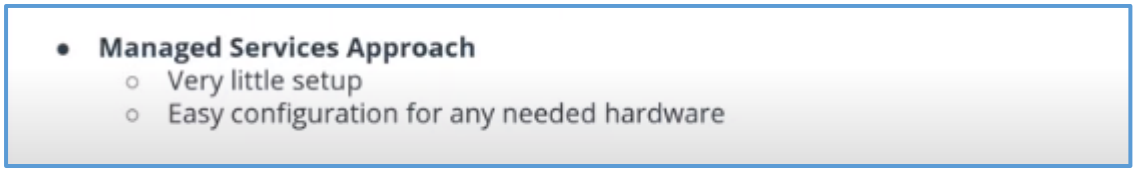
Finally, you will be, introduced to the world of programming the managed services via the **Azure Machine Learning SDK for Python**.

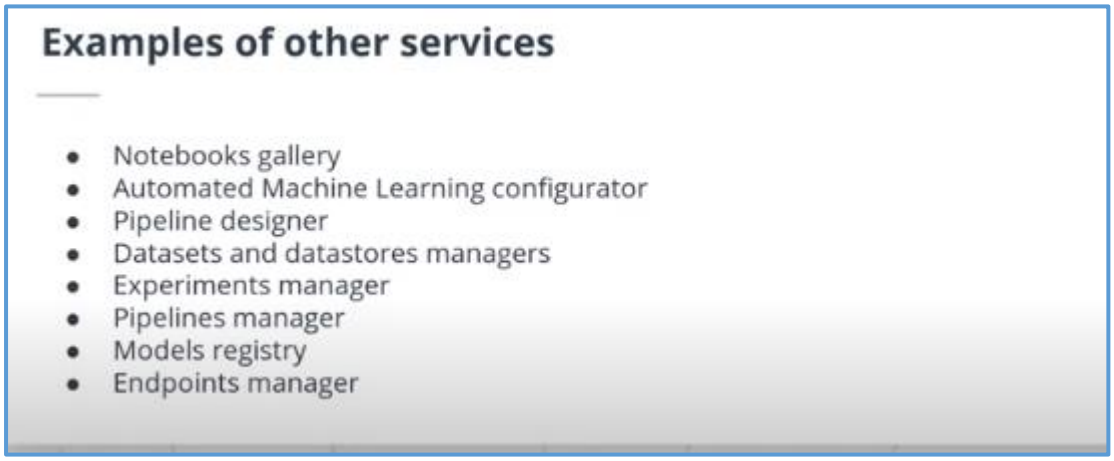
Chapter 2: Managed services for ML

The machine learning process can be labour 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 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).



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.





Chapter 3: Prelaunch Lab

Prelaunch your lab environment.

NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

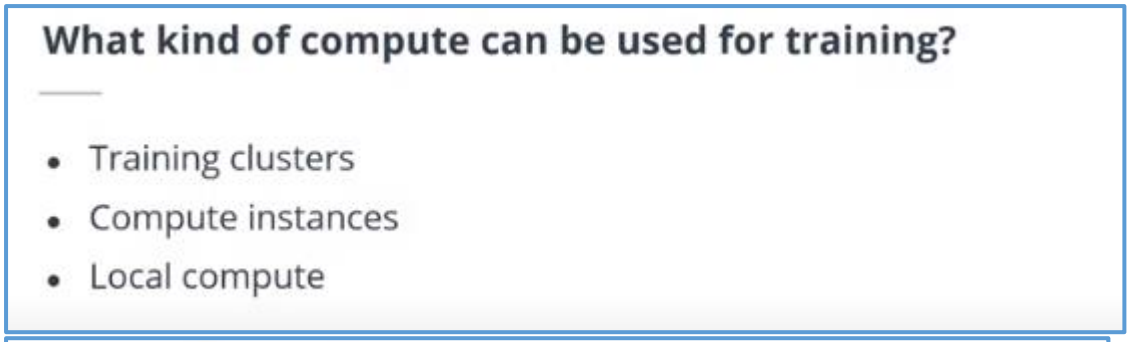
Your lab has been, reserved! Please continue to the next concept.

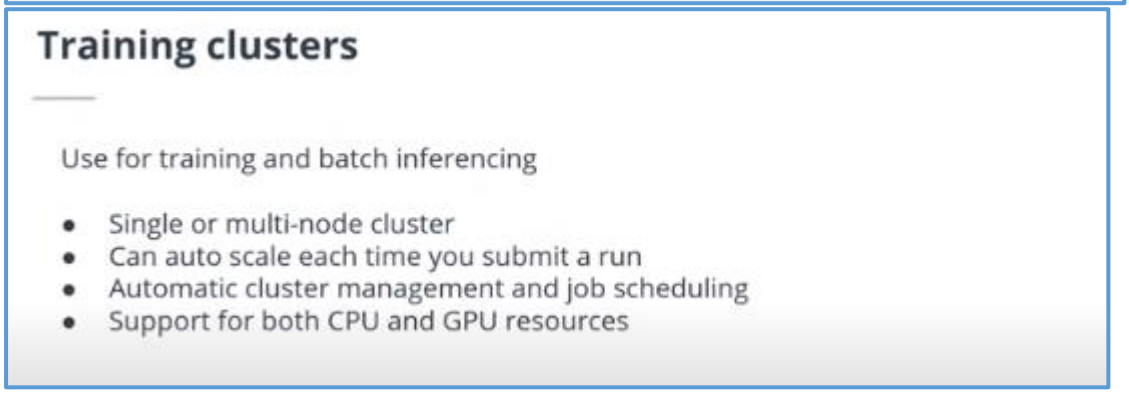
Chapter 4: Compute Resources

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

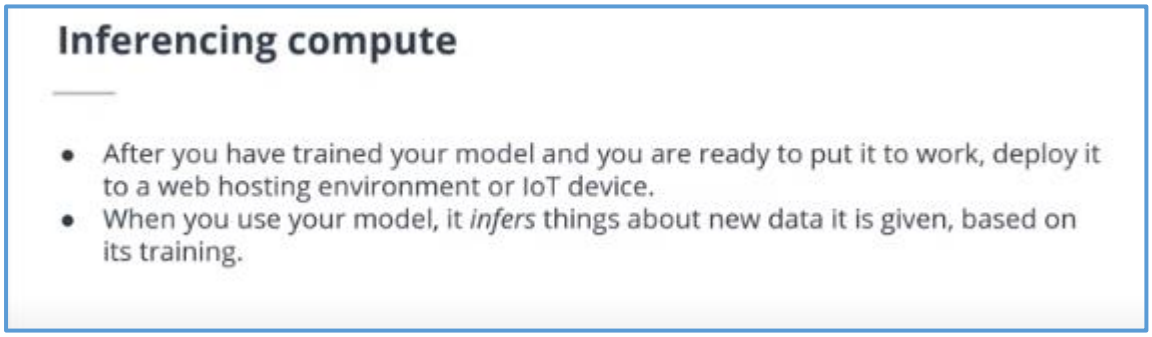
First, let us talk about compute resources that can be, used for model training.





Inferencing Compute

Once you have a trained model, you will want to be able to deploy it for inferencing for real time or batch inference. Let us take, a look at the compute resources you can use for different types of inferencing.
With real time inferencing, we make inferences for each new row of data usually in real time. For this, we package our model as a web service, however other options for packaging exists as well.
With Batch inferencing, we make inference for multiple rows of data named batches. Resources required for Batch scoring is often significant.
Azure ML has dedicated compute resource for real time inferencing called inferencing clusters. Virtually any compute resource can be, used for Batch inferencing.



Inferencing in production

Choose one of these compute targets for *production workloads*

Compute target for production workloads
Azure Kubernetes Service (AKS)
Azure Machine Learning training cluster

Azure ML training clusters allows us to run batch inferencing.
AKS provides first response time & auto-scaling for deployed service.

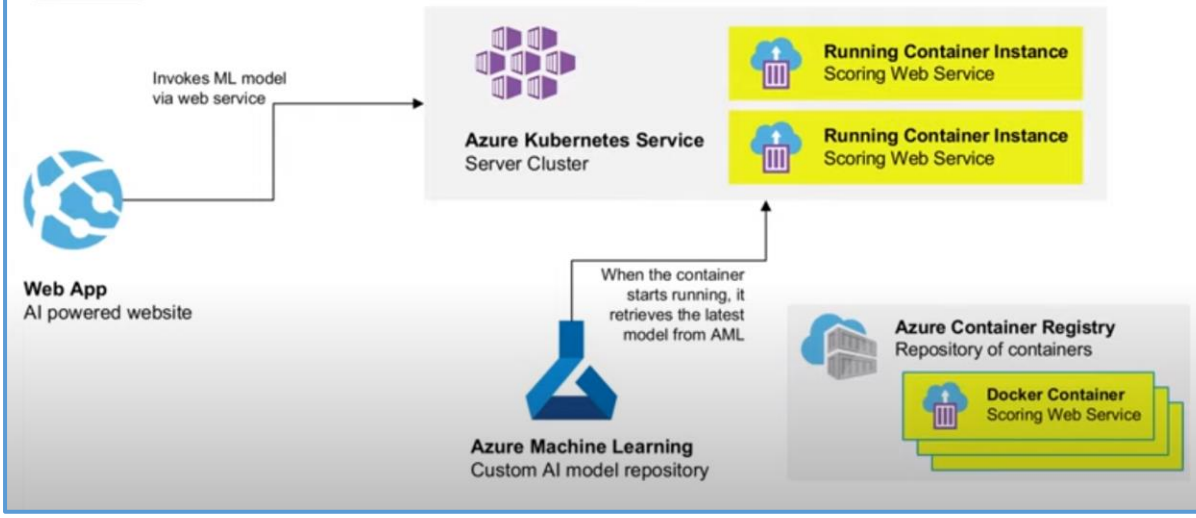
Specialized inferencing scenarios

Trained models are packaged in containers

Compute target for specialized deployment
Azure Functions
Azure IoT Edge
Azure Data Box Edge

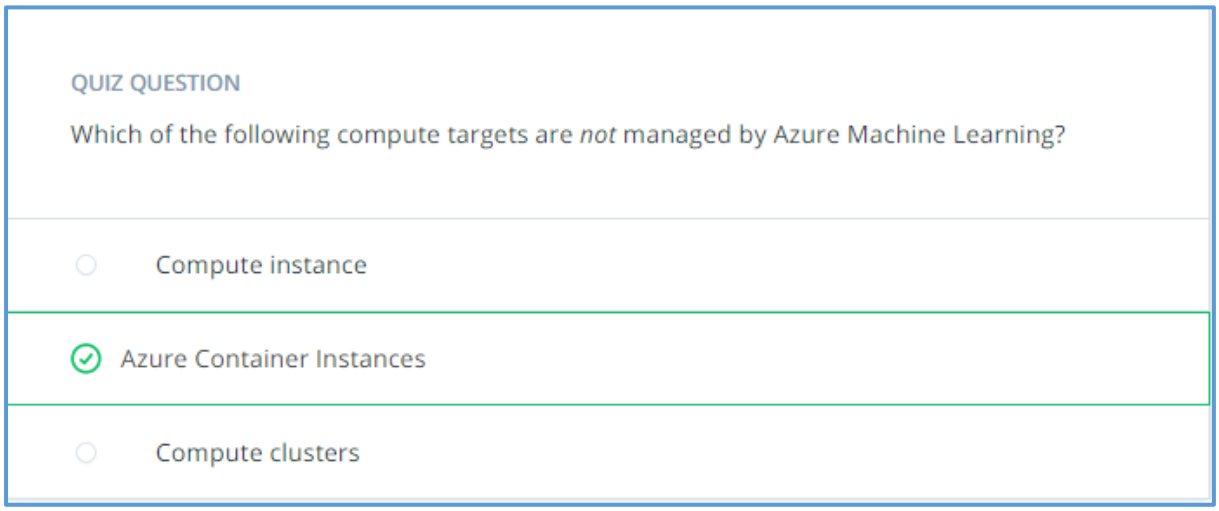
Azure ML enables us to package our trained model in a container, which enables us to use it in a wide range of destination for deployments. These models allows us to use our train models in a variety of environments.

High-level architecture of a possible deployment



- Web app uses visit to make prediction with the deployed ML models.
- Models are, deployed as container instance in AKS for high scalability.
- Azure ML was, used to train these models & register them a score in Web service Docker images in Azure container registry.
- Deploy the images in designated inferencing clusters.
- Models are stored within Azure container registry. This service hosts Docker images from managed deployments to containers.

Azure Container Instances, are not managed by, Azure Machine Learning. A managed compute resource is, created and managed by Azure Machine Learning. This type of compute is, optimized for machine learning workloads. Azure Machine Learning compute clusters and compute instances are the only managed computes.



Chapter 5: Lab: Managing Compute

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 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 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.

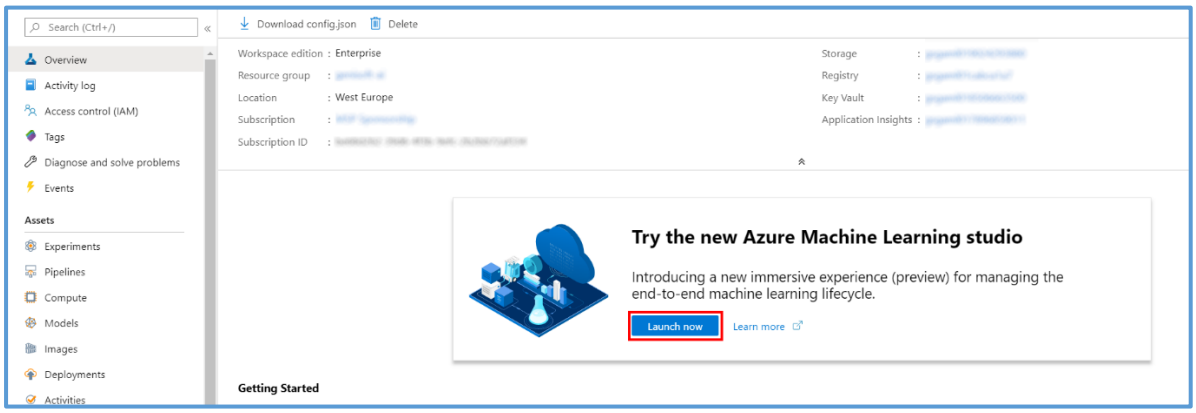
Overview

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

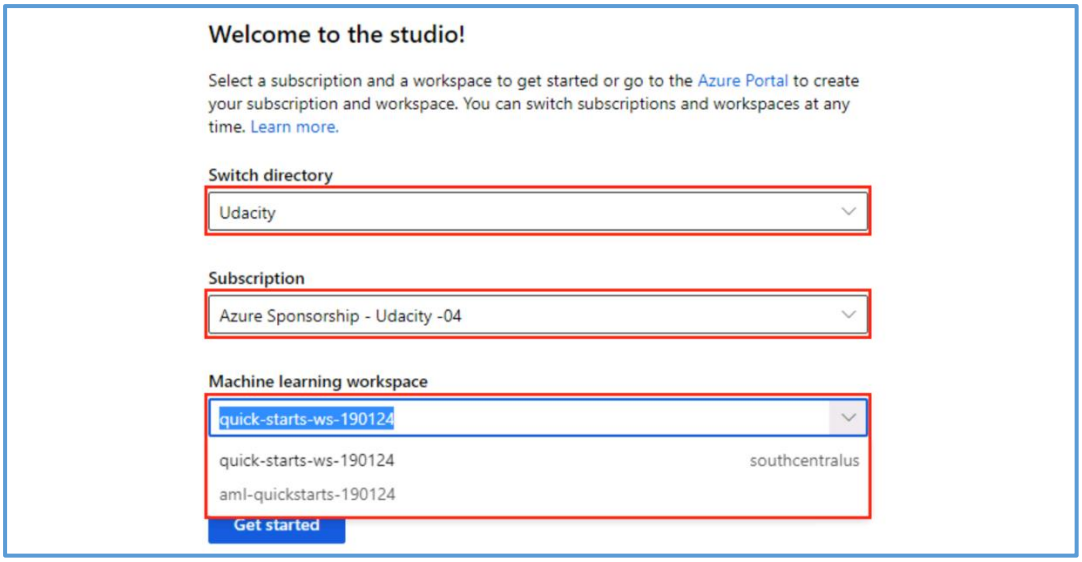
Exercise 1: Create New Compute Instance

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.

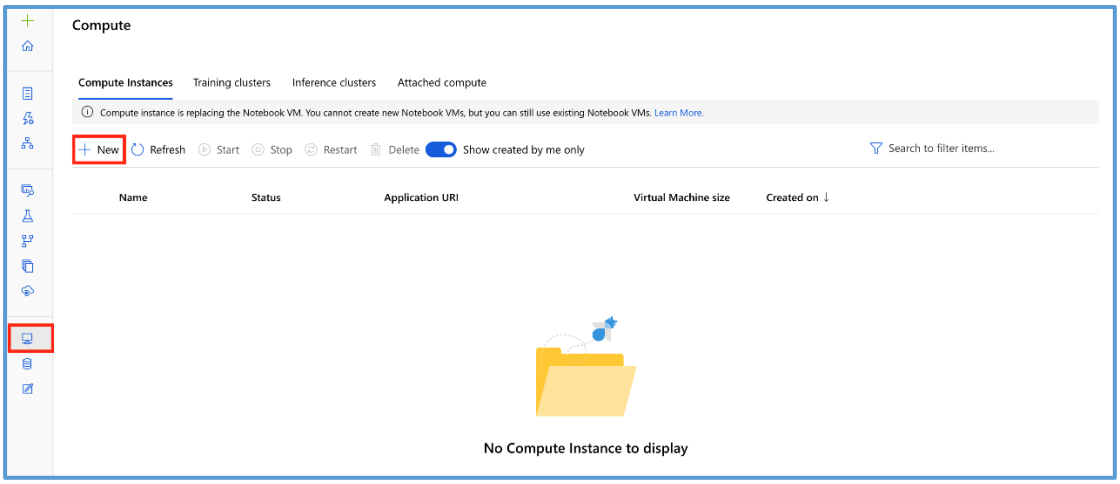


3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:

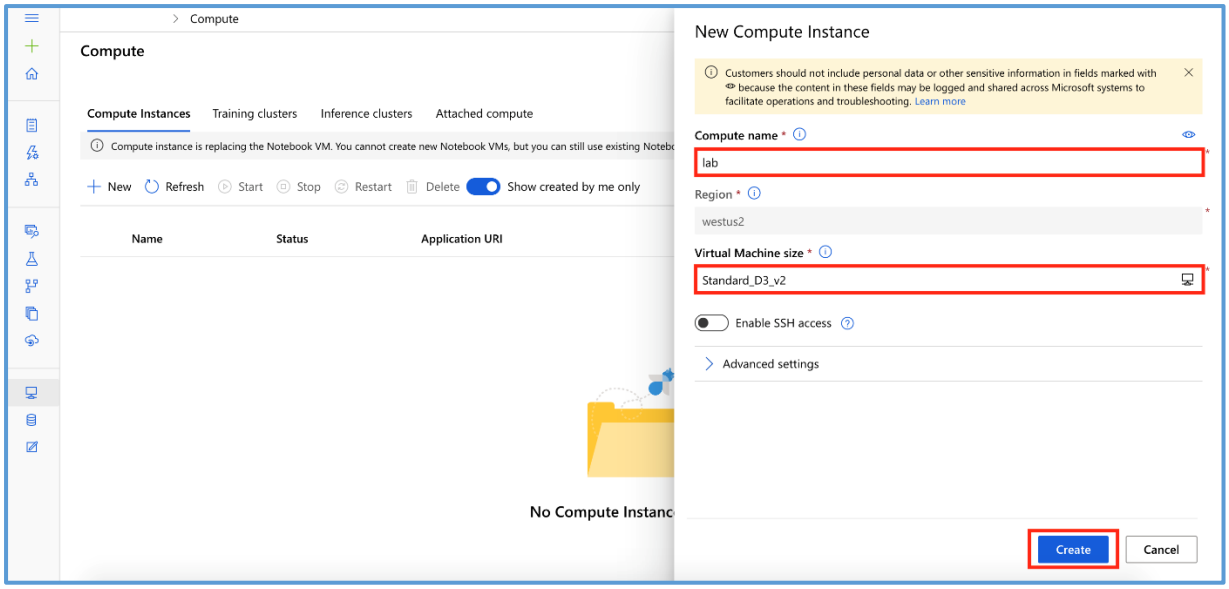


For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it does not matter which) and then click **Get started**.

4. From the studio, navigate to **Compute**, then select **+New**.



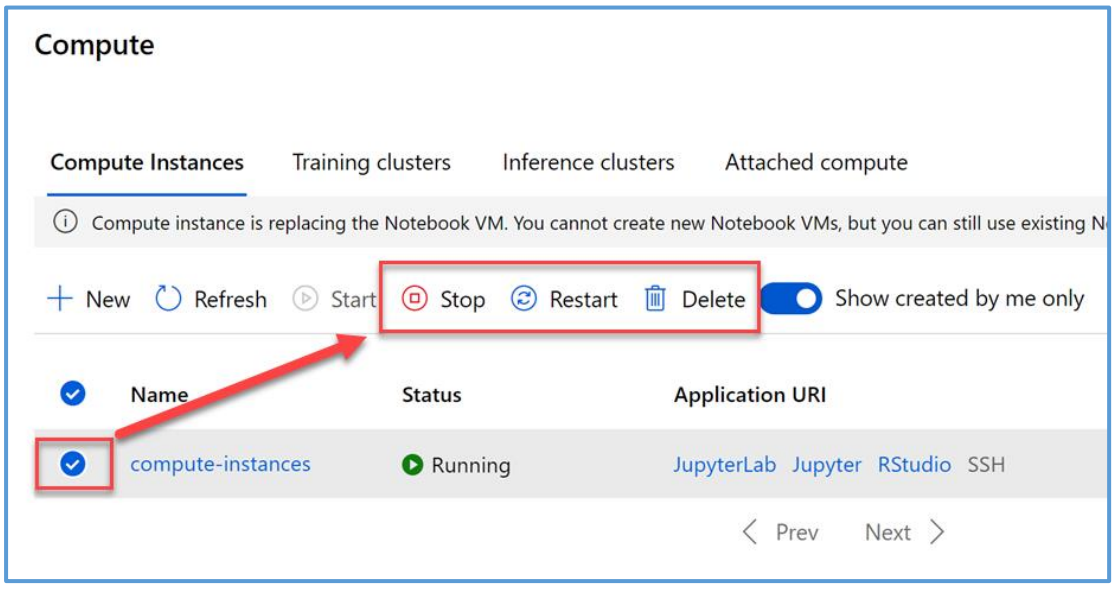
5. In the New Compute Instance pane, provide the following information and then select **Create**.
    - Compute name: provide an unique name
    - Virtual Machine size: Standard_D3_v2



6. It will take couple of minutes for your compute instance to be ready. Wait for your compute instance to be in status `Running`.

Exercise 2: Explore Compute Instances

1. 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.

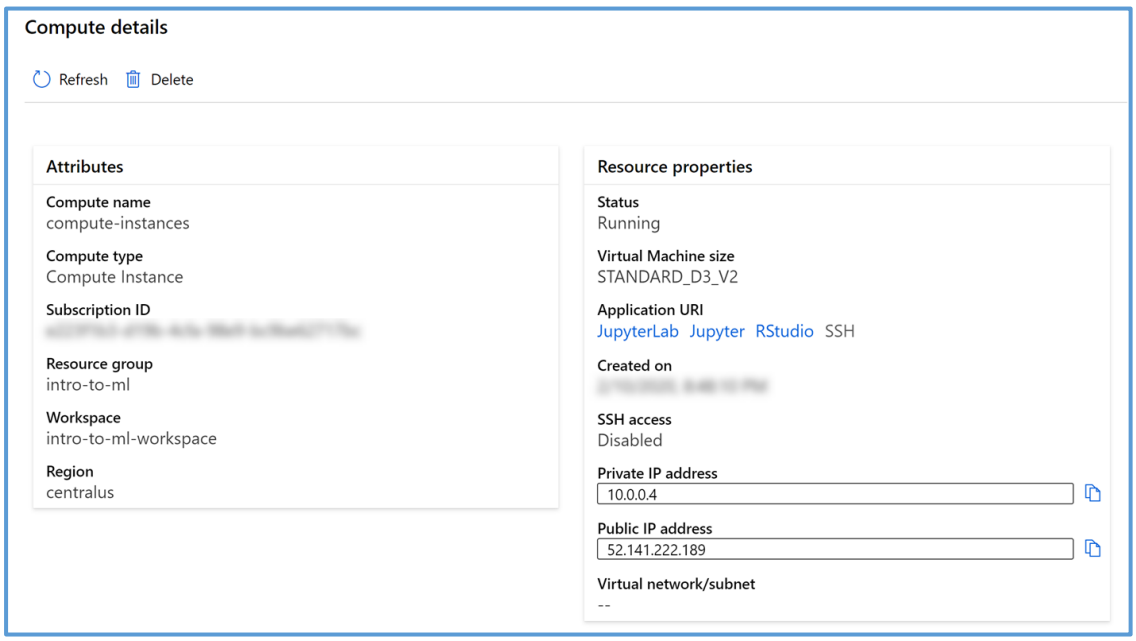


There are different scenarios in which you will want to perform these actions. Here are the actions you can take on a selected compute instance, and what they do:

- **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.

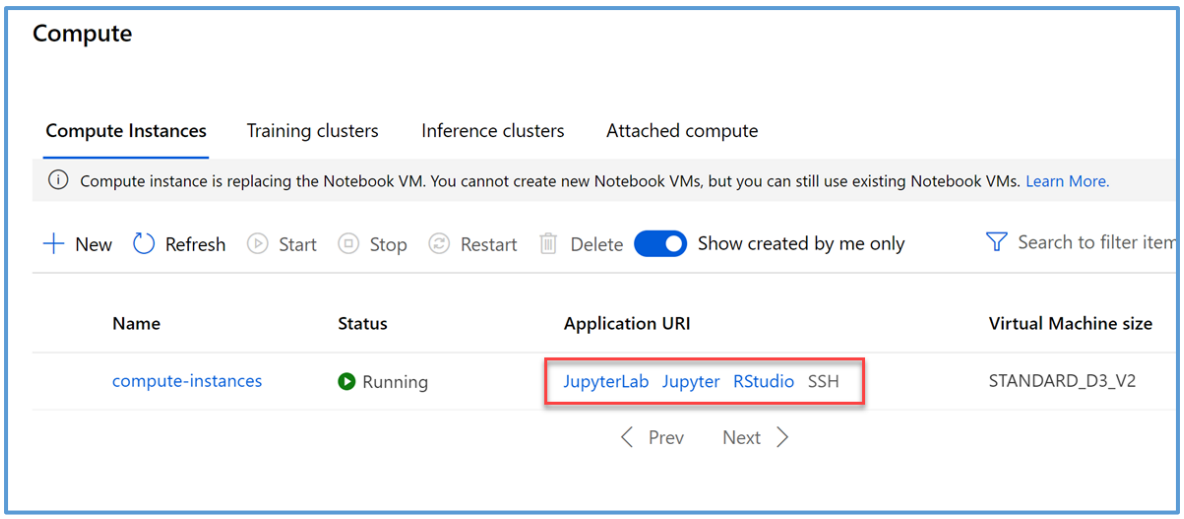
2. Select the name of your instance. This opens the **Compute details** blade, revealing useful information about your compute instance.



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.

3. Navigate back to **Compute**. 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.



4. Select each of the application links to sign in to the related environment. You may be, prompted to select your user account for each application.

Next Steps

Congratulations! You have completed the introduction to managing a compute instance lab. You can continue to experiment in the environment but are free to close the lab environment tab and return to the Udacity portal to continue with the lesson.

Chapter 6: Walkthrough: Managing Compute

- Within Azure ML studio, navigate to compute.
- Start was creating a new compute instance
- For each compute instance, there are several actions that, we can take like: Stop, Restart & Delete.
- Even after deleting a compute instance all NBs and files are still stored on the default storage account for your workspace and it can be, accessed by going to the File Explorer in the NB section.
- To get more information about the compute instance we need to click on the compute name.
- In compute, details there are 2 important things – Attributes and Resource properties.
- We can also launch different applications like – Jupyter, Jupyter Labs & RStudio. All of them can be, accessed via compute instance directly.

Chapter 7: Prelaunch Lab

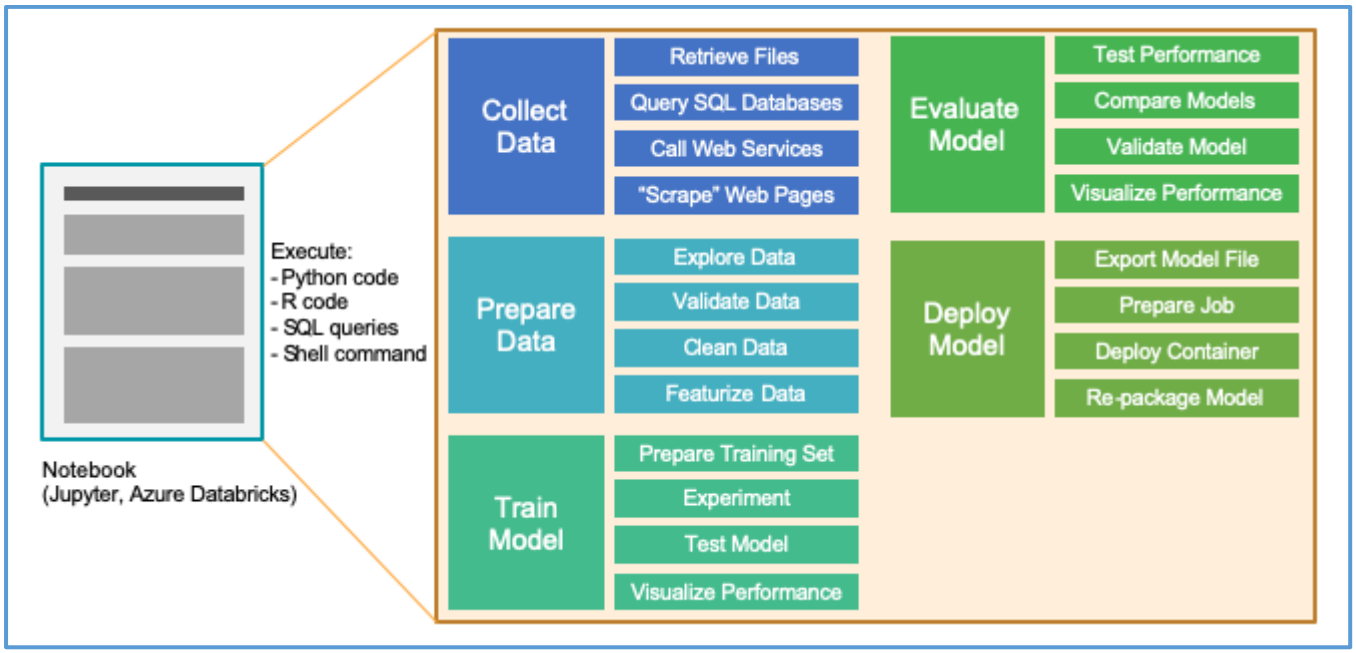
Prelaunch your lab environment.

NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

Your lab has been, reserved! Please continue to the next concept.

Chapter 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. In this diagram, you can see that we can use a notebook environment to perform the five primary stages of model development:

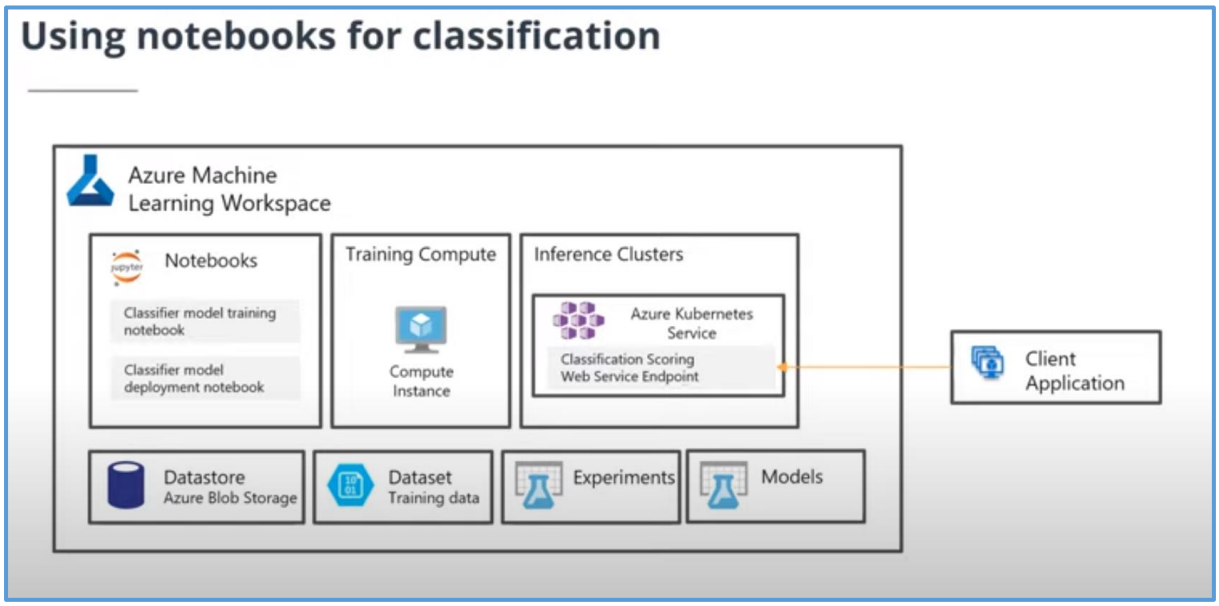


The best way to understand how managed notebook environments work is to jump in and work with one—so that's what we'll do in the next lab, by training a simple SciKit learn model on a medical dataset.

Notebooks are, provided by applications like Matlab to help scientists, students, professors to create self-documenting notebooks that others can use to reproduce experiments.

Notebooks contains runnable code, output, formatted text and visualizations.

Notebooks are, used to conduct exploratory data analysis and conduct model training using languages like Python, Scala, R and others. Popular Notebooks used are Jupyter, Databricks NBs, R Markdown & Apache Zeppelin.



- Jupyter NB is, created with Compute instance provisioned within Azure ML workspace.
- NB uses Azure ML and Python SDK to retrieve the training data as set of flat files from the dataset registered with the workspace and whose connection information such as storage account name, keys to the Azure storage is defined in the data store.
- Supervised model training executes in the context of an experiment run, which logs the training duration and other metadata about training. Collects the performance statistics collected during model evaluation performed in the NB & relates the metadata with the actual model that is, uploaded in the model registry.
- New NB is, used to deploy the model as a web service running in AKS.
- This NB defines the web service that retrieves the model from model registry, load model into memory & uses the loaded model for inferencing against every request for classification.
- Once deployed the scoring web service handles classification of input samples arriving from the client application in the form of HTTP request.

Chapter 9: Lab: Managed Notebook Environments

Compute Resources

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.

Overview

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.

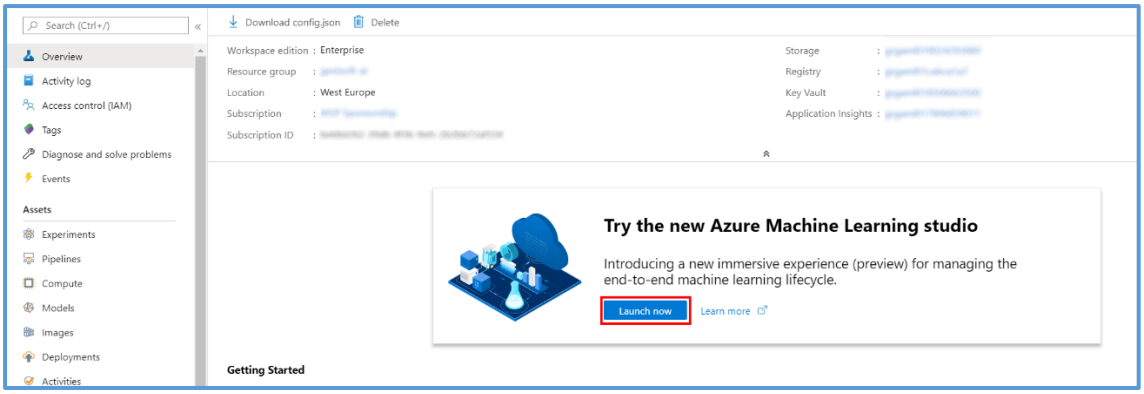
In this tutorial, you learn the following tasks:

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

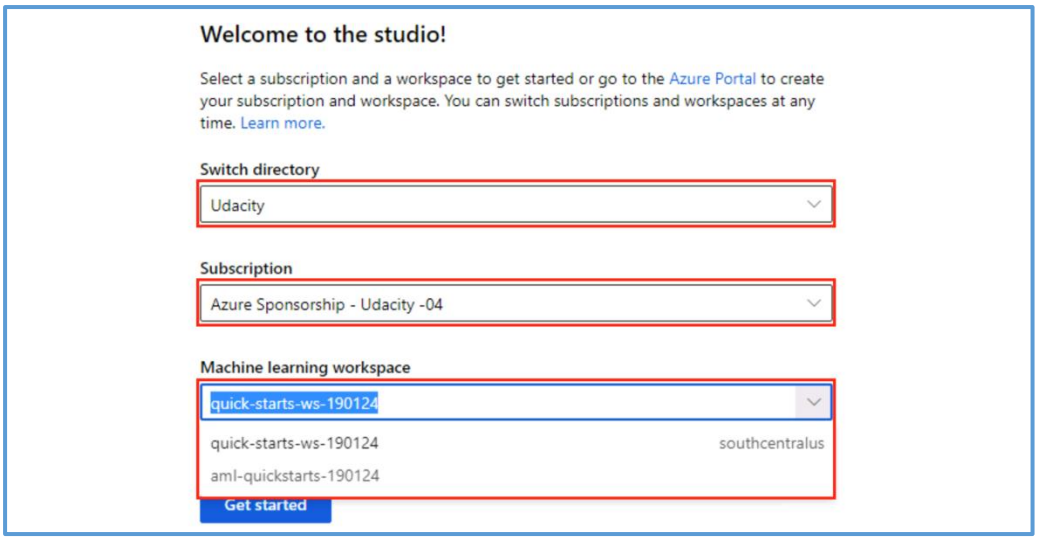
Exercise 1: Run the Notebook for this Lab

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.

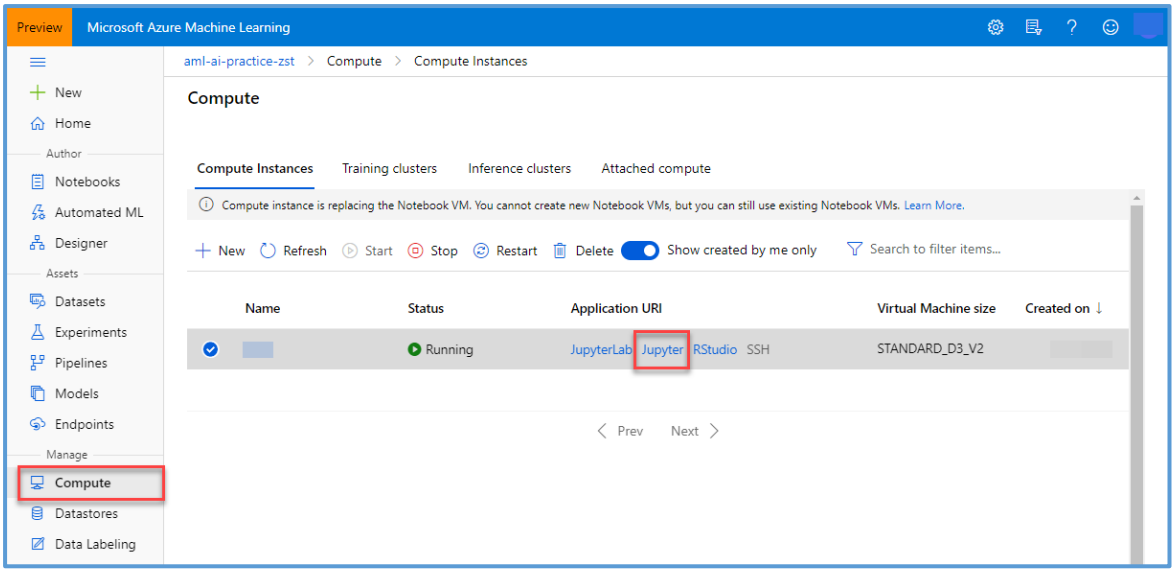


3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:

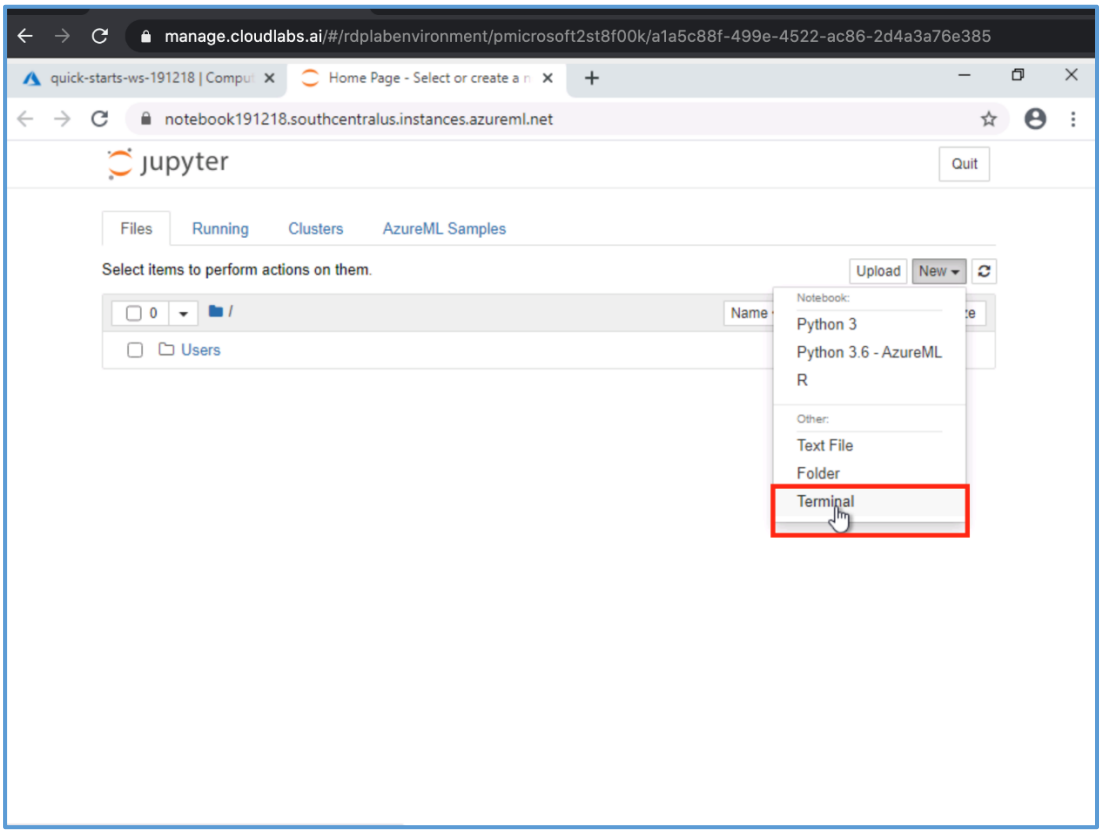


For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it does not matter which) and then click **Get started**.

4. From the studio, navigate to **Compute**. Next, for the available Compute Instance, under Application URI select `Jupyter`. Be sure to select `Jupyter` and not `JupterLab`.

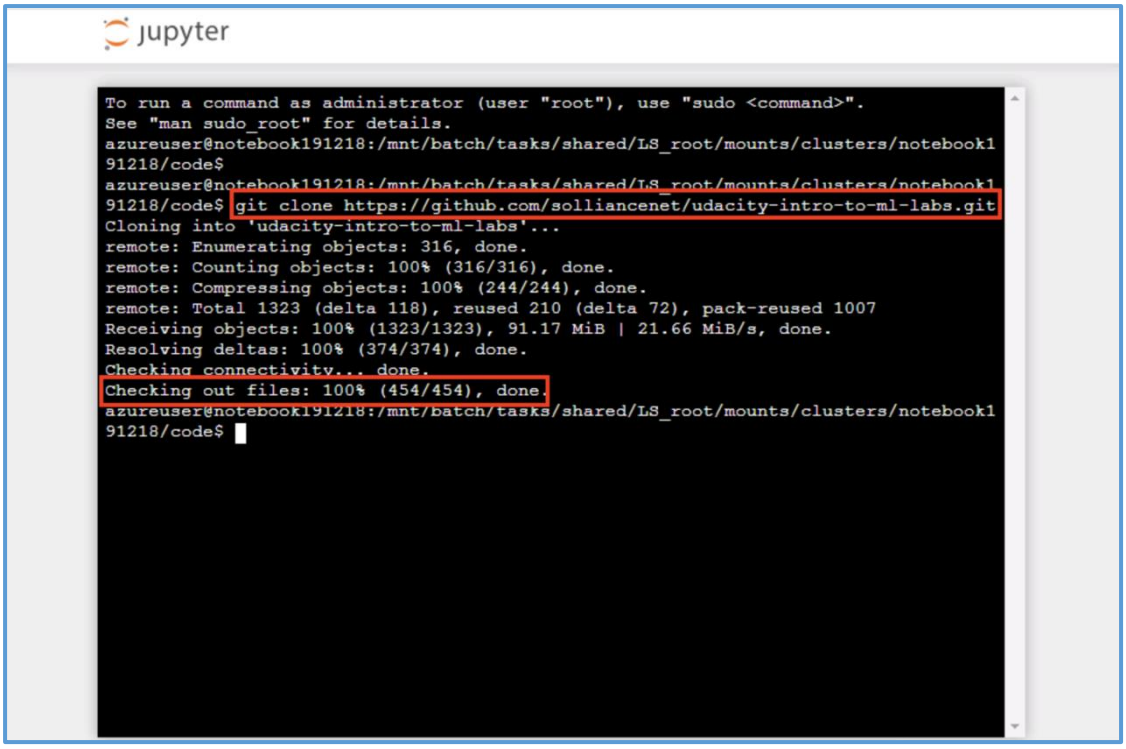


5. From within the Jupyter interface, select **New, Terminal**.

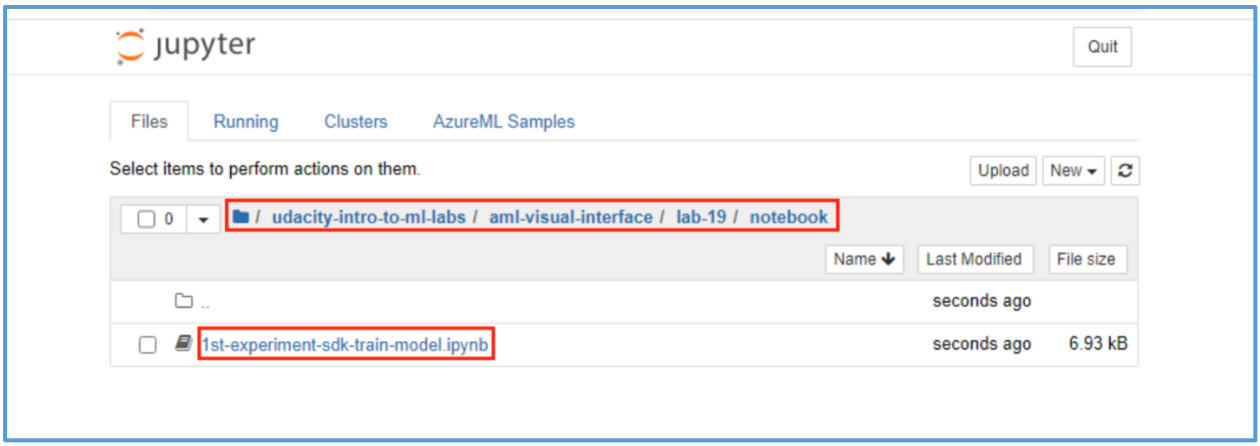


6. In the new terminal window run the following command and wait for it to finish:

```
git clone https://github.com/solliancenet/udacity-intro-to-ml-labs.git
```

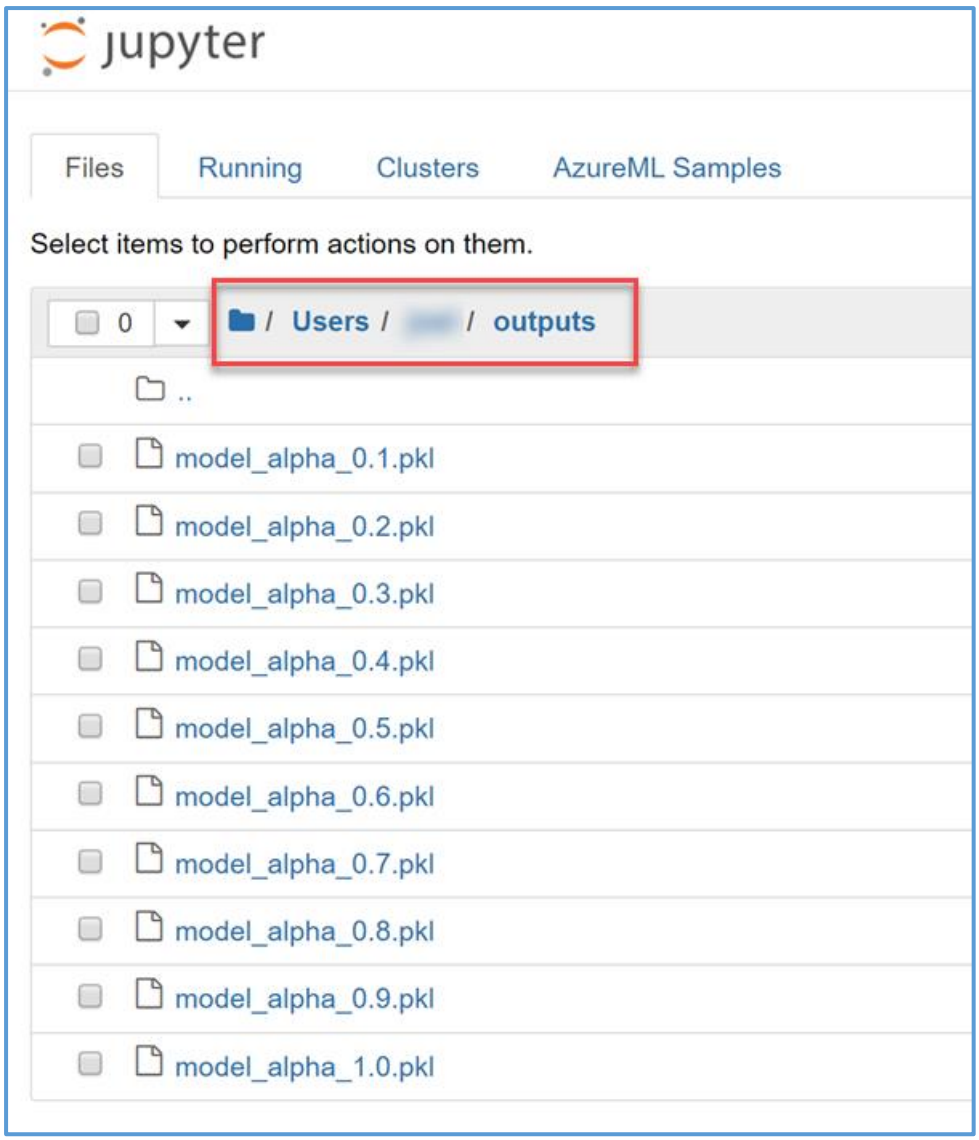


7. From within the Jupyter interface, navigate to directory `udacity-intro-to-ml-labs/aml-visual-interface/lab-19/notebook` and open `1st-experiment-sdk-train-model.ipynb`. This is the, Python notebook you will step through executing in this lab.



8. Follow the instructions within the notebook to complete the lab.

9. After completing the notebook, navigate back to the **Users** folder, select your assigned username, then select the newly created **outputs** sub-folder. Here you will see the trained models (`*.pkl` files) generated by the last cell you executed. In addition, the serialized model is uploaded to each run. This allows you to download the model file from the run in the portal as an alternative to downloading them from this folder.



Next Steps

Congratulations! You have just learned how to use the Jupyter application on a compute instance to train a model. You can now return to the Udacity portal to continue with the lesson.

Chapter 10: Walkthrough: Managed Notebook Environments

- We learn foundational design pattern of Azure ML
- We train simple Scikit-learn model based on diabetes dataset.
- We will have practical knowledge of SDK to scale up, to develop in more complex experiment and workflows.
- Connect workspace and create experiment.
- Load data and train Scikit-learn model
- Navigate to compute and select the compute instance.
- Select Jupyter
- Complete the Lab following the instructions in Python NB
- All trained models would be available within the Outputs sub-folder.

- Serial model is uploaded for each run and also we get the trained model generated by the last cell we executed

Chapter 11: Prelaunch Lab

Prelaunch your lab environment.

NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

Your lab has been, reserved! Please continue to the next concept.

Chapter 12: Basic Modelling

Training, evaluating, and selecting the right Machine Learning models is at the core of each modern data science process. However, 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.

Training, evaluating, and selecting the right machine learning models is at the core of each modern data science process. When we talk about training a machine-learning model, what do we mean?

In essence, the training of a machine-learning model is the process through which a mathematical model is, built from data that contains both inputs and expected outputs, or only inputs in the case of unsupervised learning. There are several classes of algorithms available to build the model, like classification, regression, clustering, feature alerting, and others. Let us introduce some basic concepts that define the steps involved in the model training process. An experiment is a generic context of handling runs. Think about it as a folder that organizes the artifacts used in your model training process. Model training runs are, used to build a train model. A run contains all artifacts associated with a training process, like output files, metrics, logs, and snapshots of the directory that contains your script. The model registry keeps track of all the models in Azure Machine Learning workspace. Models are either produced by a run or imported outside Azure Machine Learning and are made available through model registration.

Experiments:

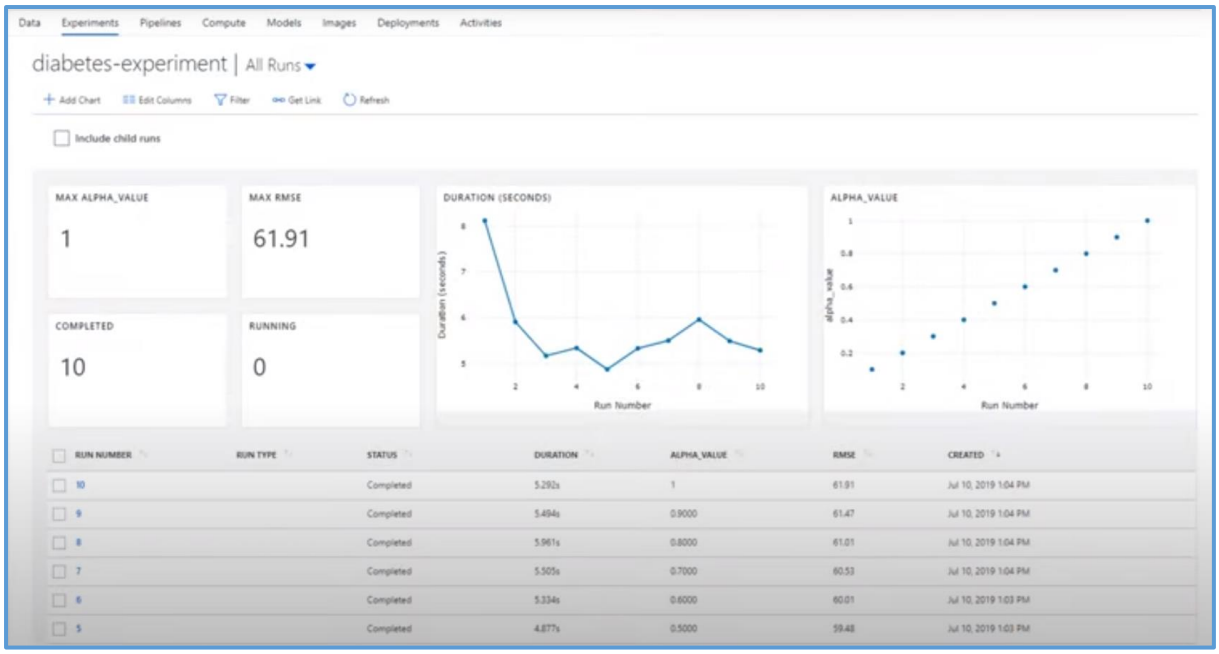
Azure Machine Learning experiments differ from an experiment in the machine learning sense.

In Azure Machine Learning, experiments are a generic context for organizer runs. If you have hundreds of runs, it can be difficult to keep track of them unless you can organize them in some way. Think about an Azure Machine Learning experiment as a folder that organizes the artifacts used in your model train and process.

When you submit a run, you provide an experiment name and information about the run is stored under that experiment.

If you submit a run and specify an experiment name that does not exist, a new experiment with the newly specified name is automatically, created.

This is a screenshot of the main experiment's page.



Here, you see all the individual runs in the experiment. Any custom log values, such as Alpha value or the RMSE becomes fields for each run, and also become available for charts and titles at the top of your experiment page.

To add a log metrics, to charts or titles, hover over it, click the "Edit" button and find your customer blogged metrics. When training models at scale with over hundreds and thousands of separate runs, this page makes it easy to see every model you have trained, specifically how they were, trained and how unique metrics have changed over time. Clicking on the run number link, in the run number column, in the previous screen, takes you to a page of each individual run.

The default tab Details shows you more detailed information on each run. Navigate to the Outputs tab and see the *.pkl file for the model that was, uploaded to the run during the training iteration.

Here, you can download the model file rather than having to retrain it manually.

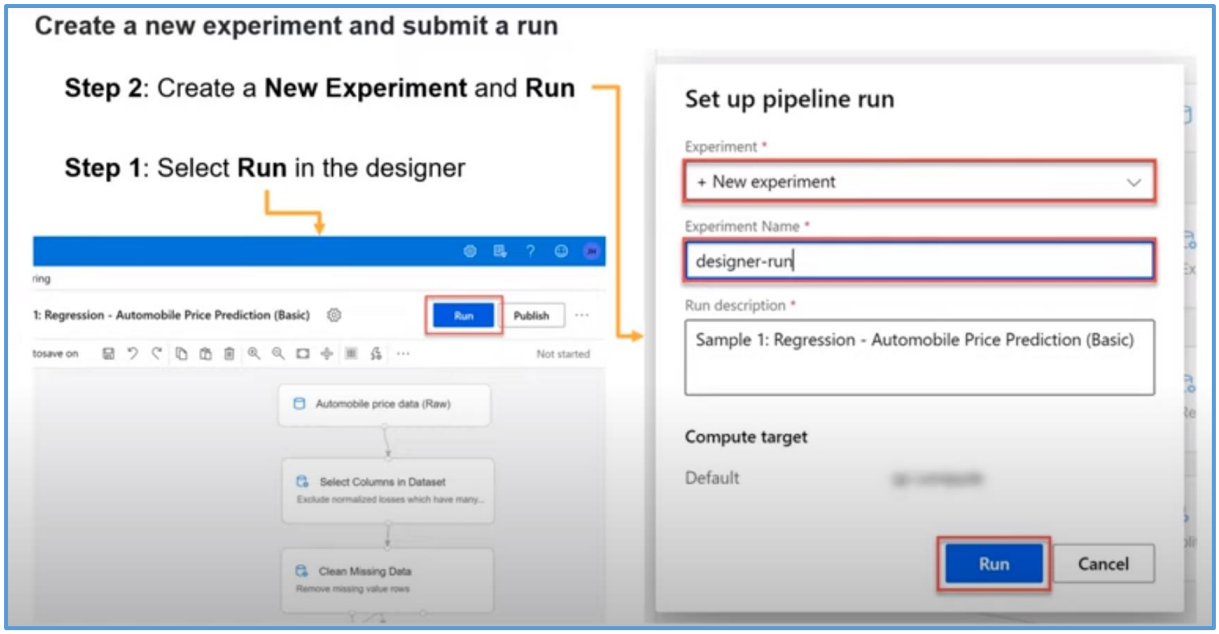
Runs

A run is a single execution of a training script. Azure Machine Learning records all runs and stores the following information: metadata about the run, things such as timestamps, durations, and so on, metrics that are logged with each of your script, output files that are auto collected by the experiment or explicitly uploaded by you, and a snapshot of the directory that contains your script prior to the run.

You produce a run when you submit a script to train a model. A run can have zero or more child runs. For example, the top, level run might have two child runs, each of which might have its own child run.

A run configuration is a set of instructions that defines how a script should run in a specific compute target. A run configuration can be persisted into a file inside the directory that contains your training script, or it can be constructed, as an in-memory object, and used to, submit a run. The configuration includes a wide set of behavior definitions, such as whether to use existing Python environments or to use a Conda environment that is, built from a specification. In case you are not familiar with the term, Conda is one of the most popular environments for running Python code. When you use the designer in Azure Machine Learning Studio, the process to create a new experiment and submit a run is quite simple and does not require you to write any code. The first step is to select "Run" above your experiment in the designer to open the setup, pipeline run editor. The next and final step is to create a new experiment or select an existing one. Then select "Run".

This creates the experiment if needed and submits the run. After submitting the run, you can view the progress live as a completed activity in your experiment.



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

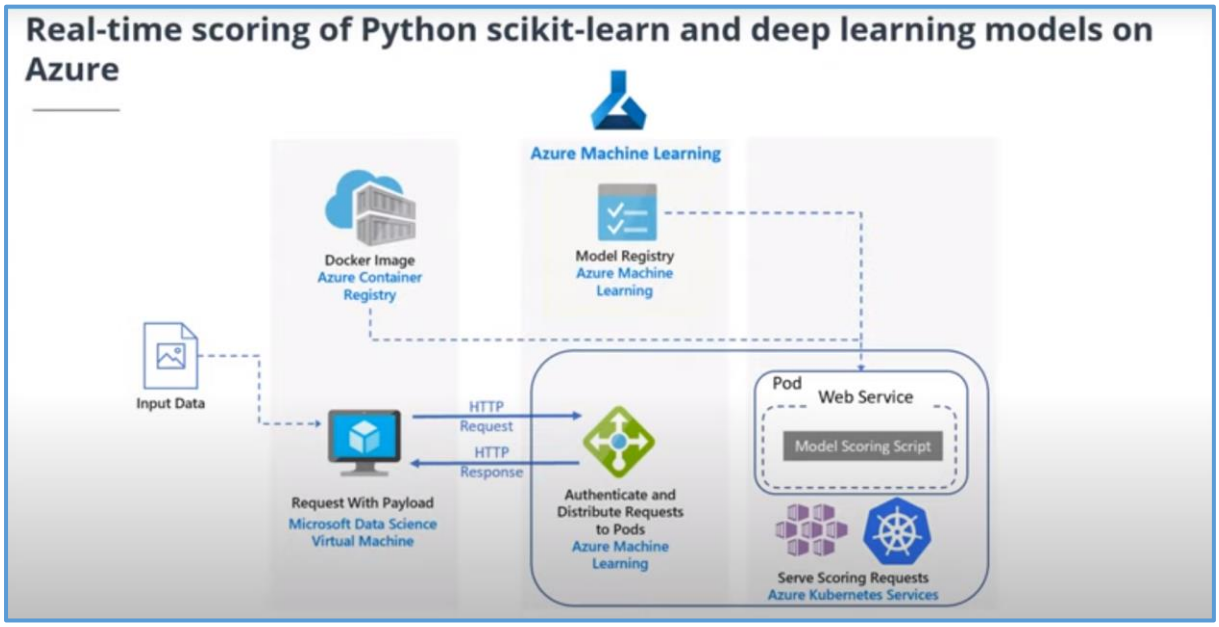
A model is, produced by a run in Azure Machine Learning. You can also use a model that is, trained outside Azure Machine Learning. You can register a model in the Azure Machine Learning workspace. **Creating a machine-learning model involves selecting an algorithm, providing it with data, and tuning the hyperparameters.**

Training is an iterative process that produces a trained model, which encapsulates what the model learned during the training process. It is important to note, Azure Machine Learning is library agnostic.

When you create a model, you can use any popular machine-learning library such as SciKit-learn, XGBoost, PyTorch, and TensorFlow

Do not worry if you are not familiar with these library names, learning them is part of the process through which you become a seasoned data scientist.

This reference architecture shows you how to deploy Python models as web services to make real-time prediction using the Azure Machine Learning service. The scenario shows how to deploy a custom ML model as a web service hosted in AKS.



Two scenarios are, covered; deploying regular Python models and the specific requirements of deploying deep learning models. Both scenarios use the architecture shown. Scikit-learn, is a simple and efficient Python library for data mining and analysis. The application flows for this architecture is as follows; the train model is, registered to the machine learning, model registry. Azure Machine Learning registers the train model as a scoring web service Docker image in the Azure Container Registry, then deploys the image to Azure Kubernetes Service, also known as AKS as a web service.

The models are stored within Azure Container Registry, which is a service that host Docker images for managed image deployments to containers. The client sends an HTTP post request with the encoded question data. The web service created by Azure Machine Learning extracts

the question from the request and the data is, sent to the model for scoring. The matching data with the associated scores are, returned to the client. This architecture consist of the following components; Azure Machine Learning which is a Cloud service that is used to train, deploy, automate, and manage machine learning models, all at the broad scale that the Cloud provides. It is, used in this architecture to manage the deployment of the models as well as authentication, routing, and load balancing of the web service.

Virtual machines, also known as VMs, the VM is shown as an example of a device local or in a Cloud that can send an HTTP request.

Azure Kubernetes Service, also known as AKS is, used to deploy the application on a Kubernetes cluster.

AKS simplifies the deployment and operations of Kubernetes. The cluster can be, configured using CPU only VMs for regular Python models or GPU enabled 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 Azure Kubernetes Service and use, to run the score in script.

The image used here is created by Azure Machine Learning, from the trained model, and scoring script, and then pushed, to Azure Container Registry.

Chapter 13: Lab: Explore Experiments and Runs

Compute Resources
Explore experiments and runs

In the previous lab (19), 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.

Overview

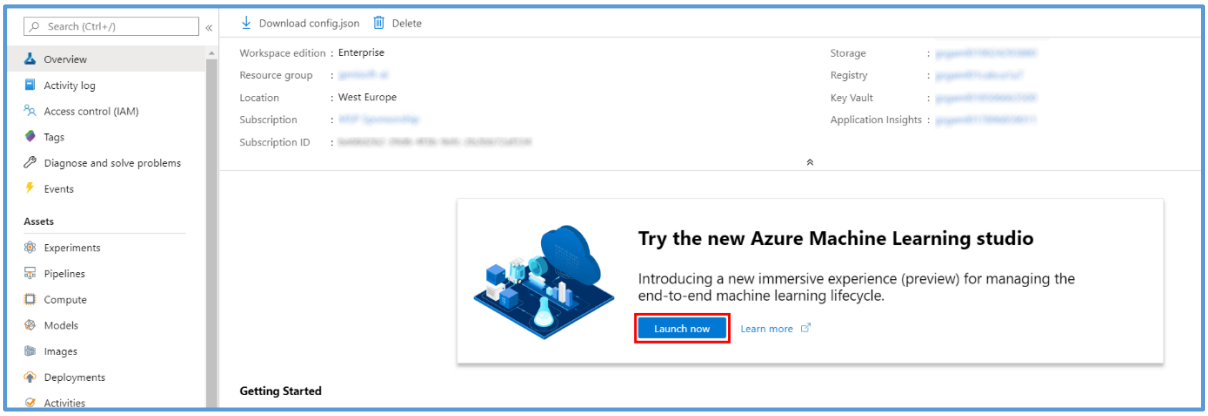
In this lab, you view the experiments and runs executed by a notebook. In the first part of the lab, you will use a notebook to create and run the experiments. In the second part of the lab, you will navigate to the **Experiments** blade in Azure Machine Learning Studio. Here you see all the individual 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.

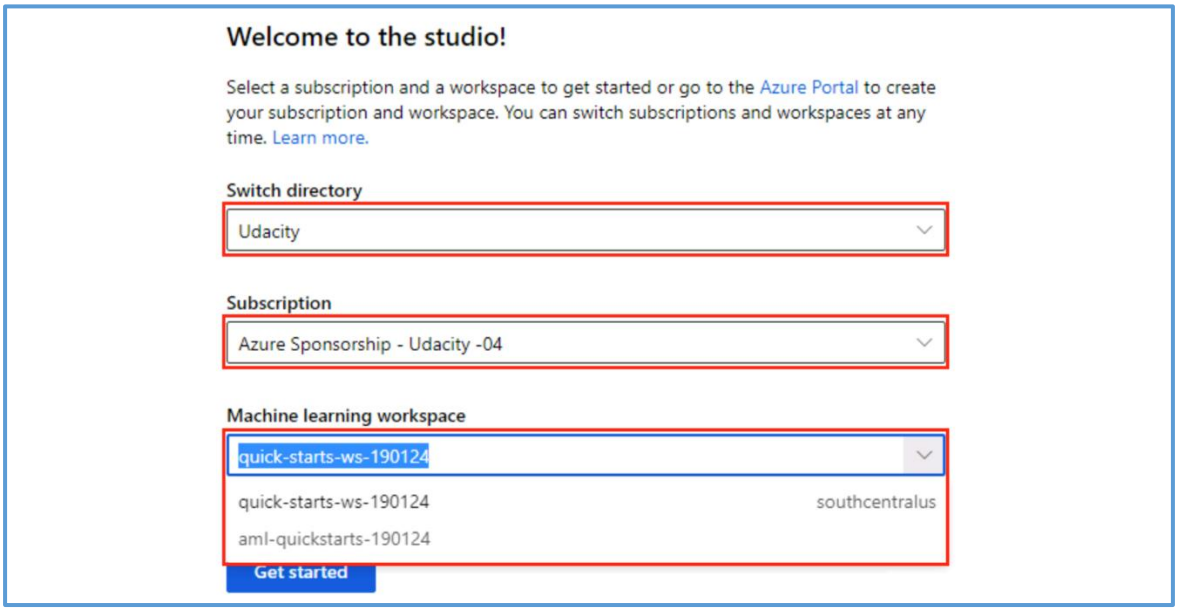
Exercise 1: Run the Notebook for this Lab

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.

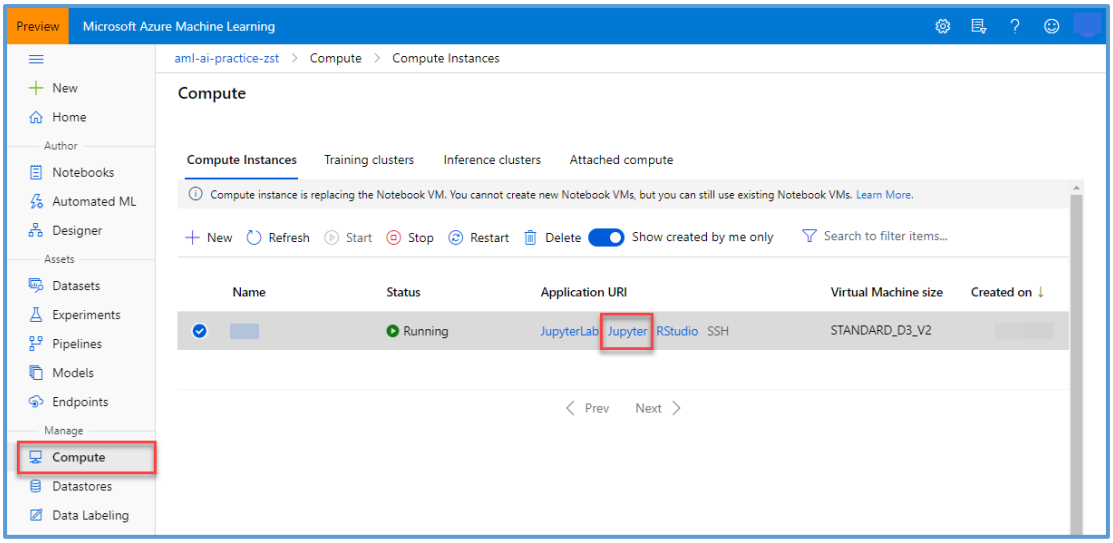


3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:

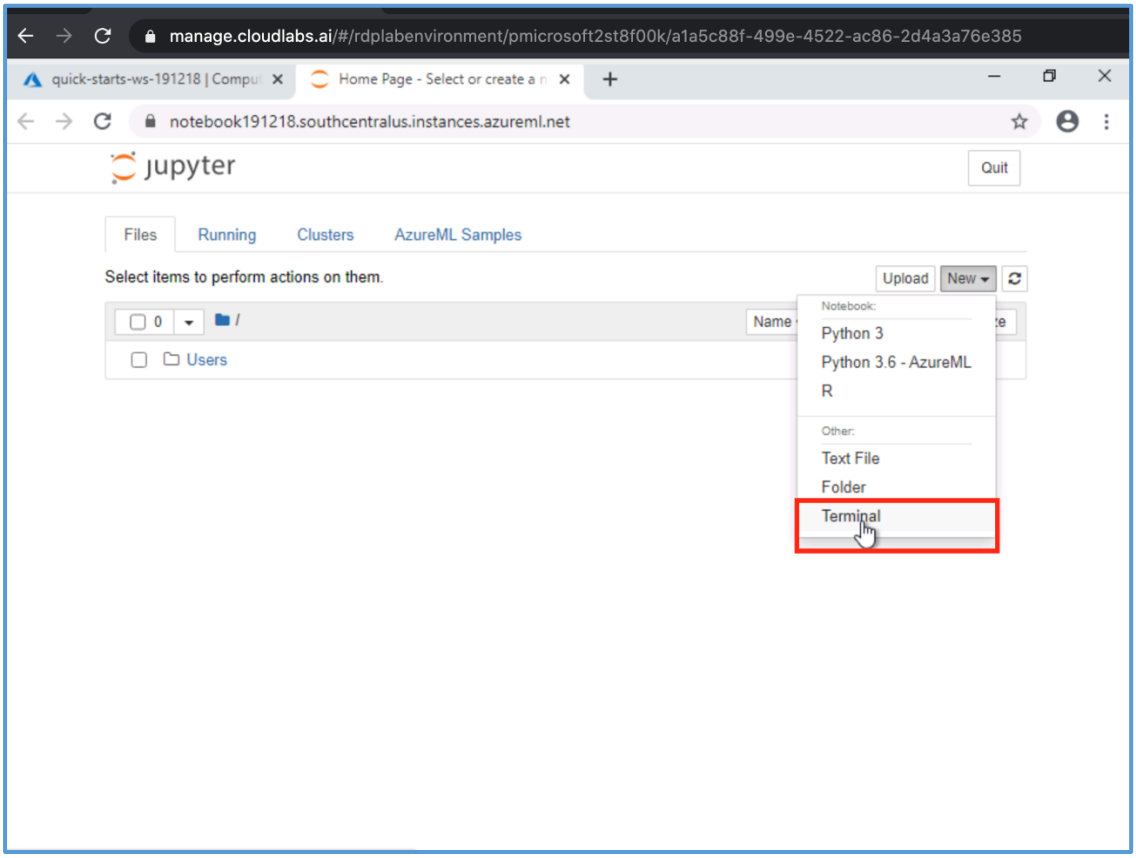


For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it doesn't matter which) and then click **Get started**.

4. From the studio, navigate to **Compute**. Next, for the available Compute Instance, under Application URI select `Jupyter`. Be sure to select `Jupyter` and not `JupterLab`.

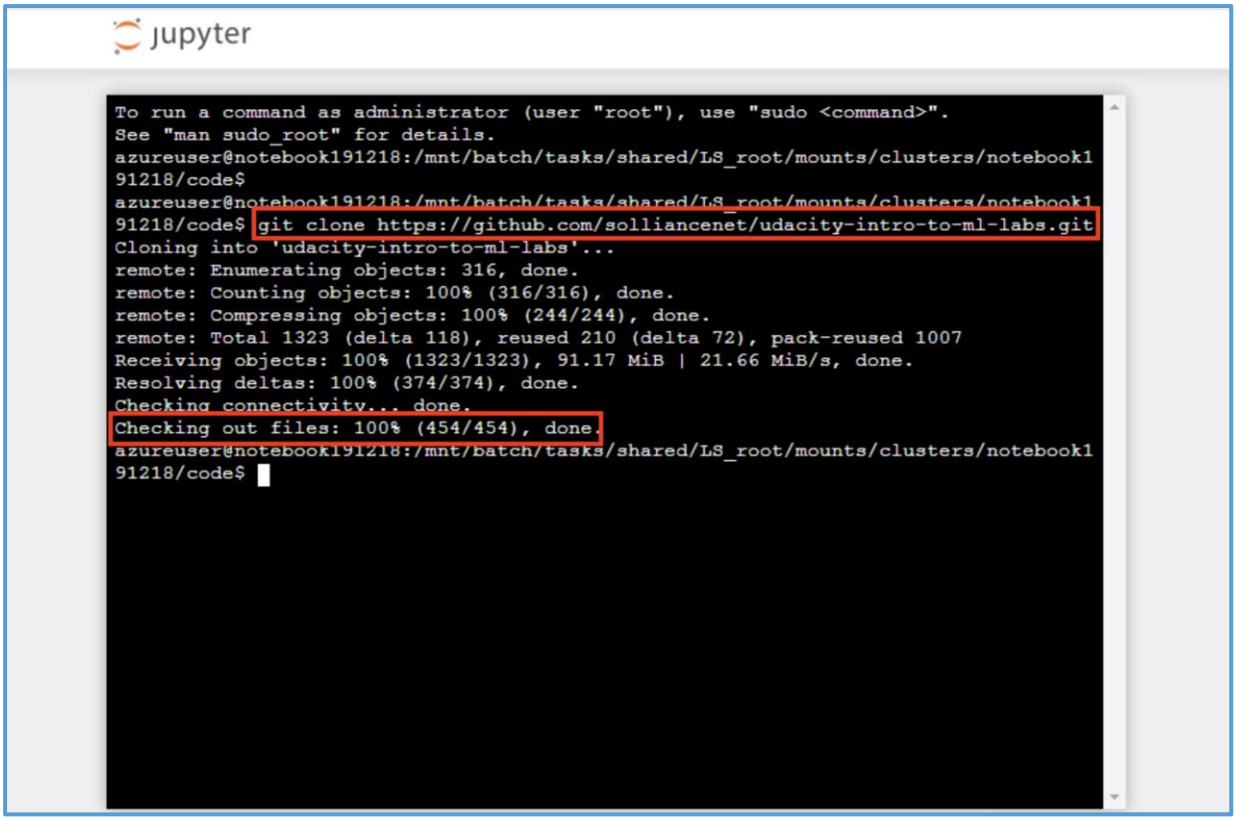


5. From within the Jupyter interface, select **New, Terminal**.

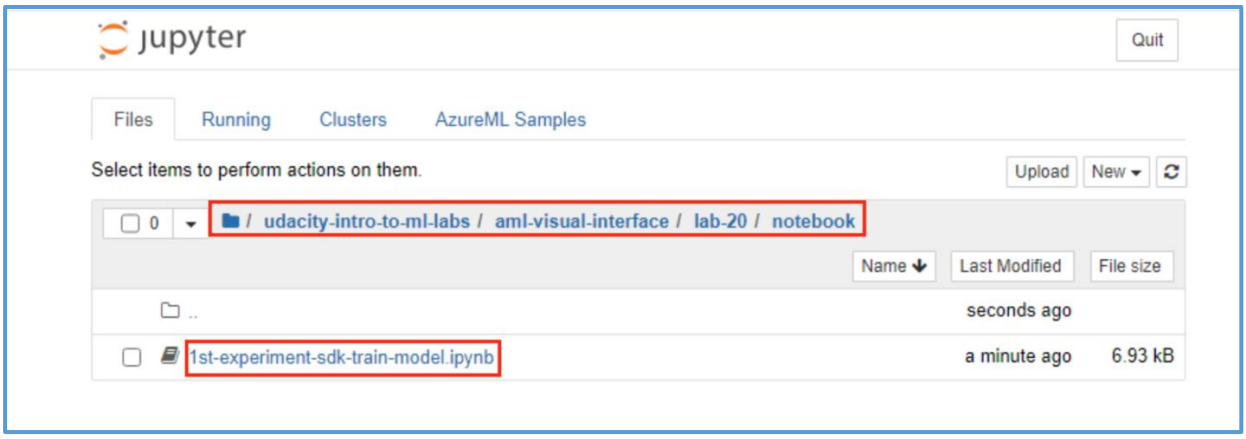


6. In the new terminal window run the following command and wait for it to finish:

```
git clone https://github.com/solliancenet/udacity-intro-to-ml-labs.git
```



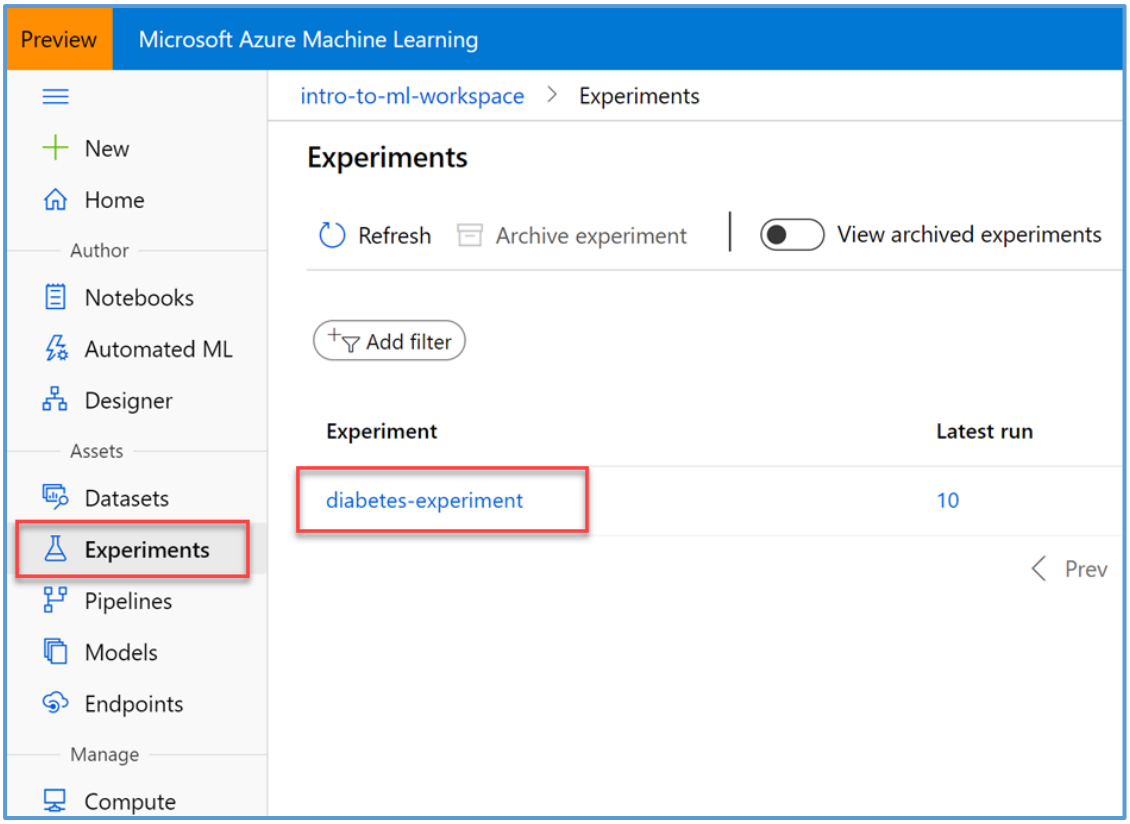
7. From within the Jupyter interface, navigate to directory `udacity-intro-to-ml-labs/aml-visual-interface/lab-20/notebook` and open `1st-experiment-sdk-train-model.ipynb`. This is the Python notebook you will step through executing in this lab.



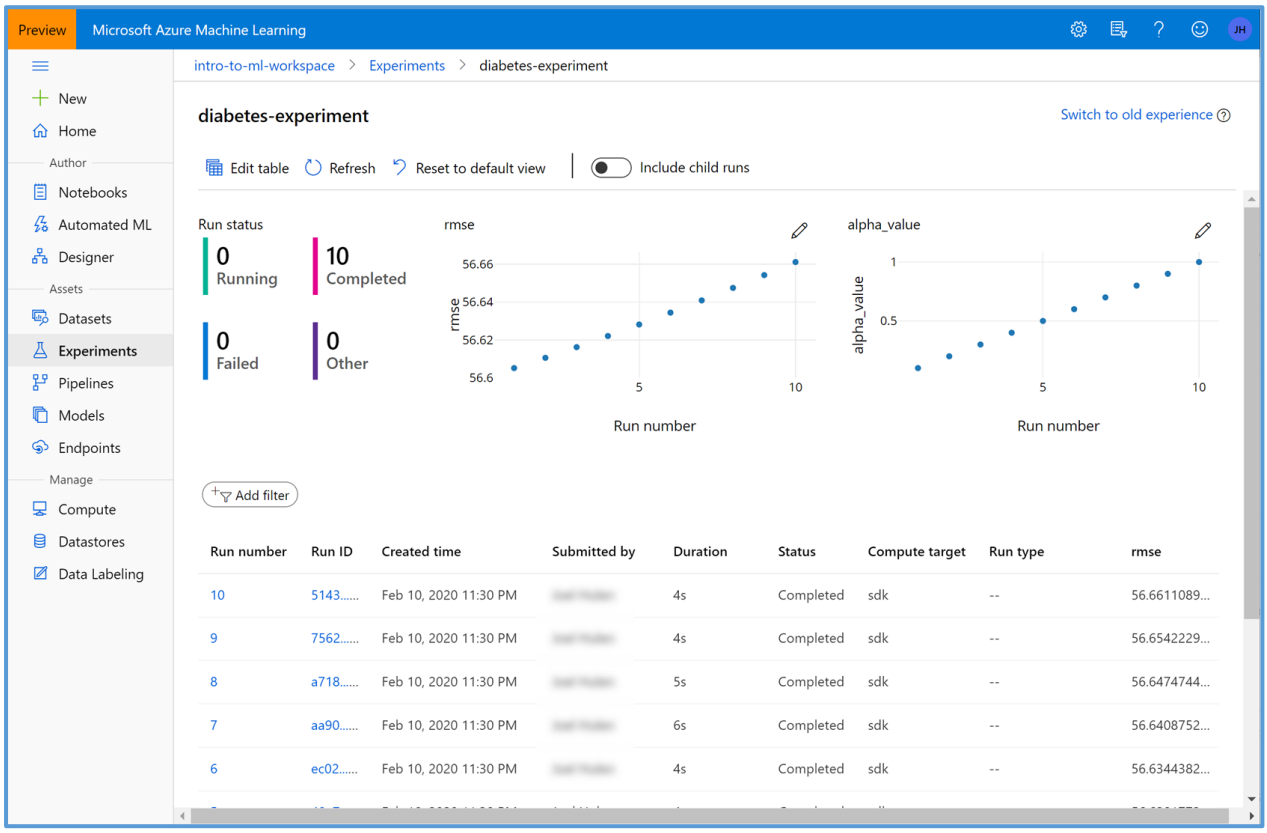
8. Follow the instructions within the notebook to complete the exercise.

Exercise 2: Open Experiments in the portal

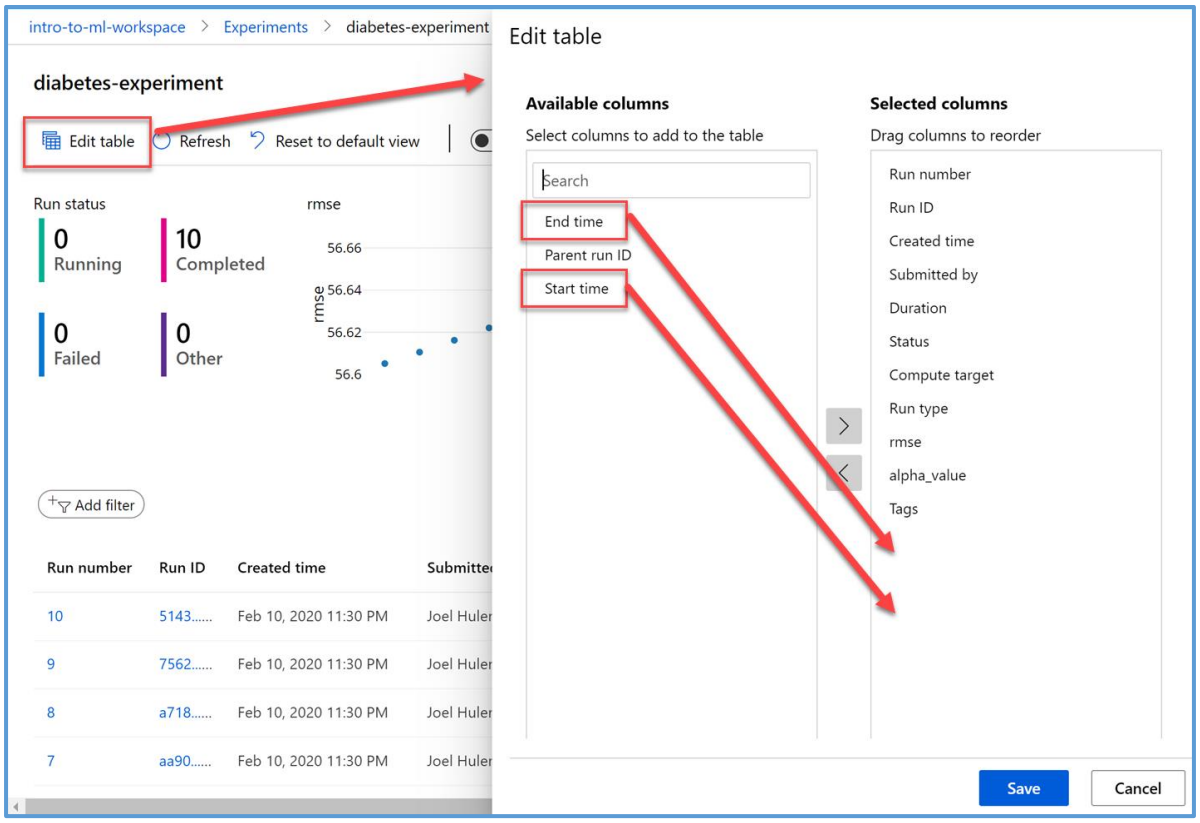
1. Within Azure Machine Learning Studio, select **Experiments** in the left-hand menu, then select the **diabetes-experiment** submitted by the notebook you executed in the previous lab (19).



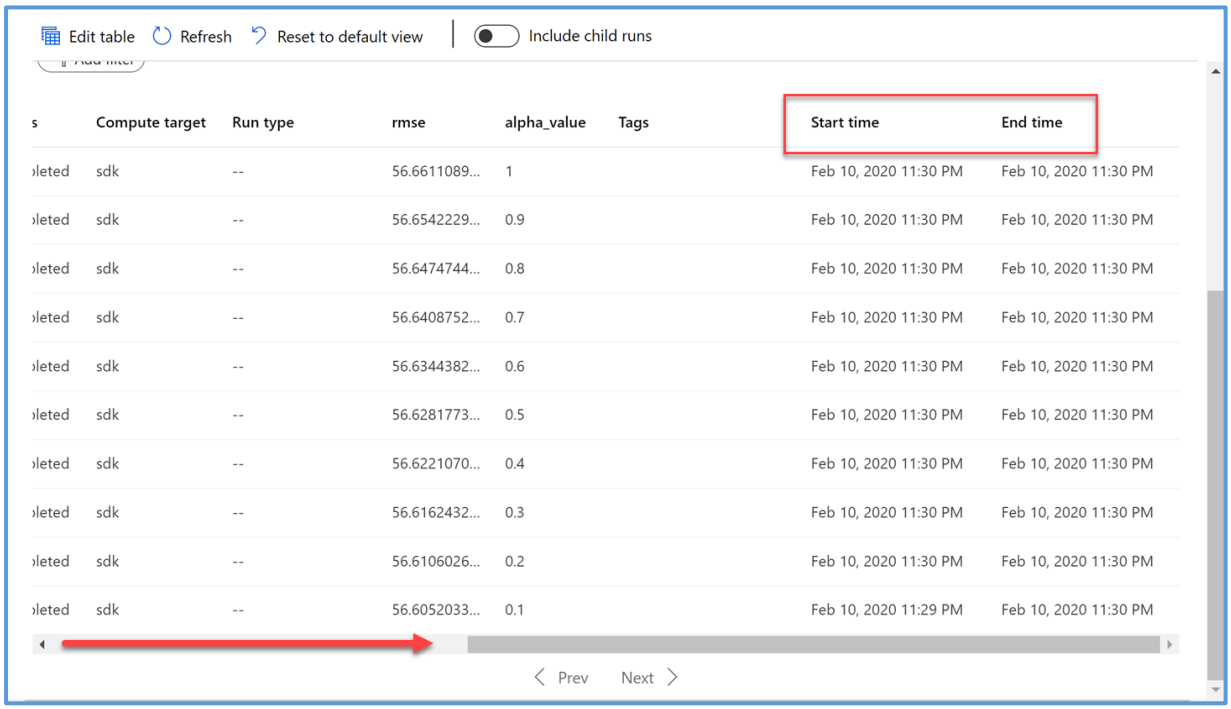
2. Here you can view details about the experiment and each of its runs, which created a new version of the model.



3. Select **Edit table** in the top toolbar. In the Edit table dialog that appears, add the **End time** and **Start time** columns to the Selected columns list, then select **Save**.

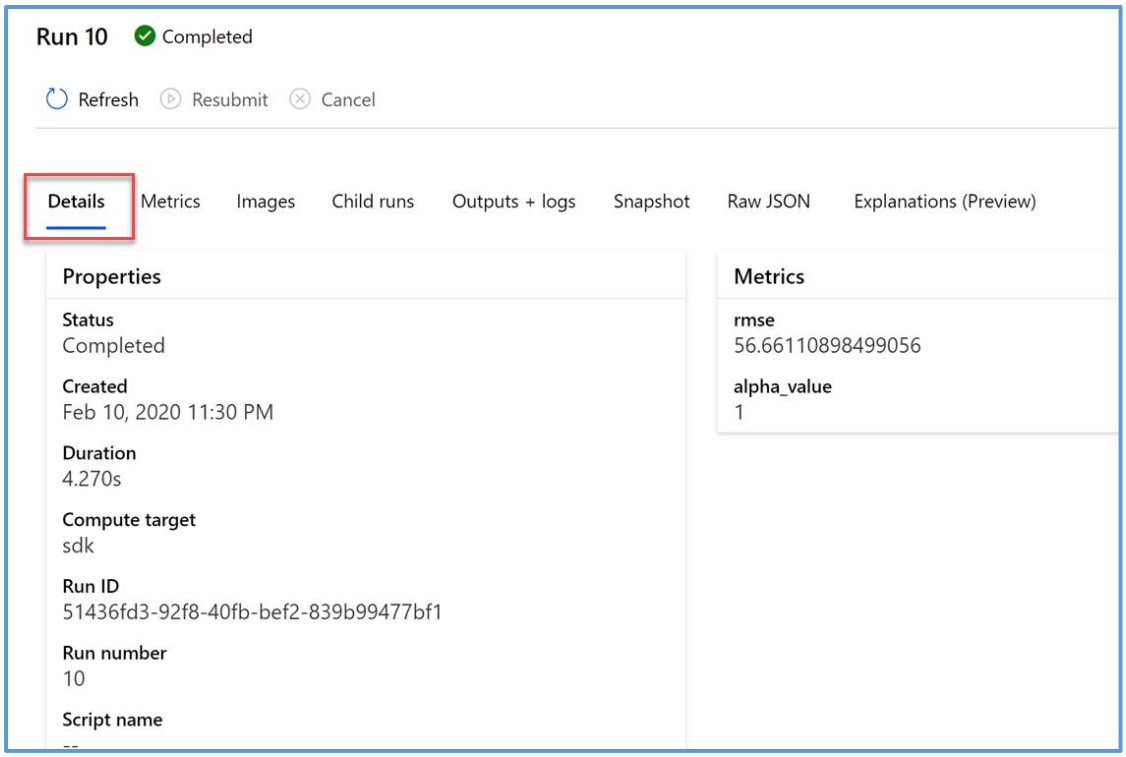


Depending on your screen resolution, you might need to scroll down the table to see the bottom horizontal scrollbar. When you scroll all the way to the right, you will see the new columns you added.

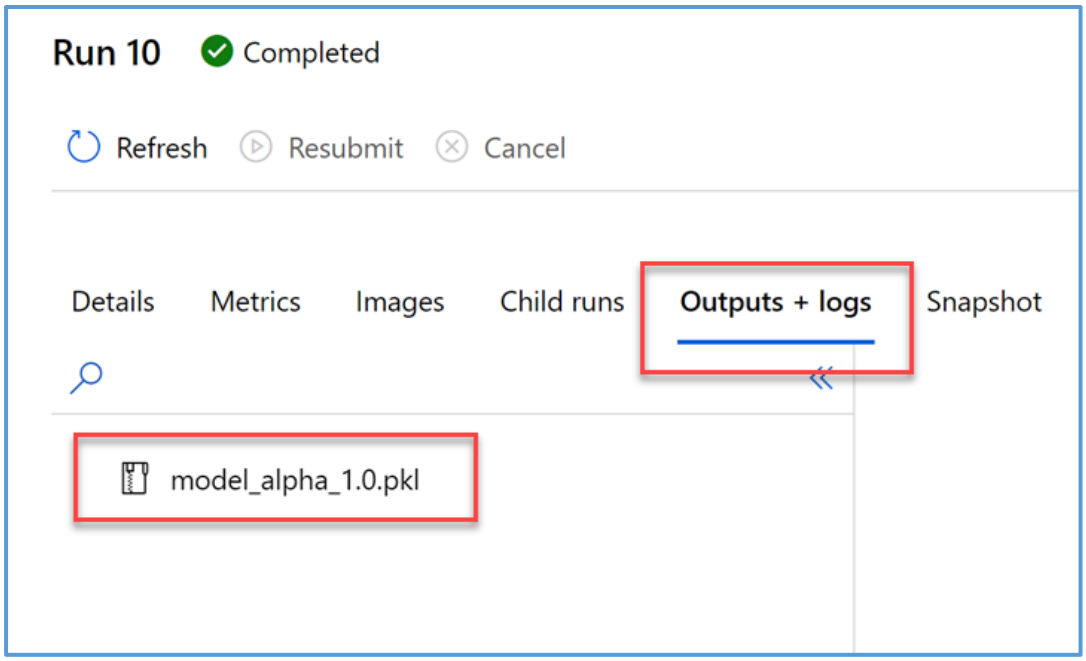


4. Select either the **Run number** *or* the **Run ID** of one of the runs to view its details. Both links on a run display the same dialog.

5. The **Details** tab shows you more detailed information about each run, including the run time and metrics.



6. Select the **Outputs + logs** tab. You see the `.pkl` file for the model that was uploaded to the run during each training iteration. This lets you download the model file rather than having to retrain it manually.



Next Steps

Congratulations! You have just learned how to use the Azure Machine Learning SDK to help you explain what influences the predictions a model makes. You can now return to the Udacity portal to continue with the lesson.

Chapter 14: Walkthrough: Explore Experiments and Runs

In the previous lab, you executed a Jupyter Notebook that trained a model through a series of ten different runs, each with a different alpha hyperparameter applied.

These runs are, created within the experiment you created at the beginning of the notebook. Because of this, Azure Machine Learning logged the detail so you can review the results of each run, and see how the alpha value is different between them.

In this lab, you viewed the experiments and runs executed by the notebook in the previous lab. You navigate to the experiments blade in the Azure Machine Learning Studio., Here you see all the individual new runs in the experiment, any custom values, alpha values or RMSE values in this case become fields for each run and also become available for the chart and tiles at the top of the experiment page. To add a log metric to the chart or tile, hover over it, and click the "Edit" button and find the Custom Log metrics.

When trained models are scaled over hundreds and thousands of separate run, this page makes it easy to see every model you trained, specifically how they were trained, and how you're unique metrics have changed over time.

To start, within the Azure Machine Learning Studio, select Experiments in the left-hand menu, and then select the "Diabetes-experiment" you submitted in the notebook you executed in the

previous lab. Here you can view details about the experiment and each of the trends, which created a new version of the model.

Select the "Edit Table" in the top toolbar.

In the edit table dialog that appears, add End time and "Start time" columns to the selected column list and select "Save". Depending on your screen resolution, you might need to scroll down the table to see the horizontal scroll bar and scroll all the way right to see the new added column. As you can see, the end time and Start time columns have not added. Select either the Run number or the Run ID to view details about the run. Both links lead to the same dialogue.

In the Details tab, it shows you, more detailed information about each run, including the run time and the metrics. Select the "Outputs plus logs" tab, you will see the *.pkl file for the model that was uploaded to the run during each training iteration. This lets you download the model rather than having to retrain it manually. Congratulations, you have just learned how to use Azure Machine Learning SDK to help you explain what influenced the prediction a model made.

Chapter 15: Advanced Modelling

As the process to build your model becomes more complex, it becomes more important to get a handle on the steps to prepare your data and train your model in an organized way. In these scenarios, there oftentimes there are many steps involved in the end-to-end process. These steps include; data ingestion, data preparation, model building and training, and model deployment.

These steps are, organized into what are, called, machine-learning pipelines. You can use tools like Azure Machine Learning Studio designer or the Azure Machine Learning SDK for Python to create reusable ML pipelines that optimize your workflow. All this automation of the pipeline infrastructure sounds like a lot of software development and automated builds and deployment. The software industry calls this process DevOps. Is there similar process for automated pipelines as well?

Actually, there's a term for this, MLOps. When it comes to DevOps principles, any non-trivial project that includes data science components will need to take advantage of a combination of features from the following two major platforms; Azure Machine Learning and Azure DevOps. The DevOps mindset applied to the data science solutions is commonly, referred to as machine learning operations, also known as MLOps.

Machine Learning Pipelines

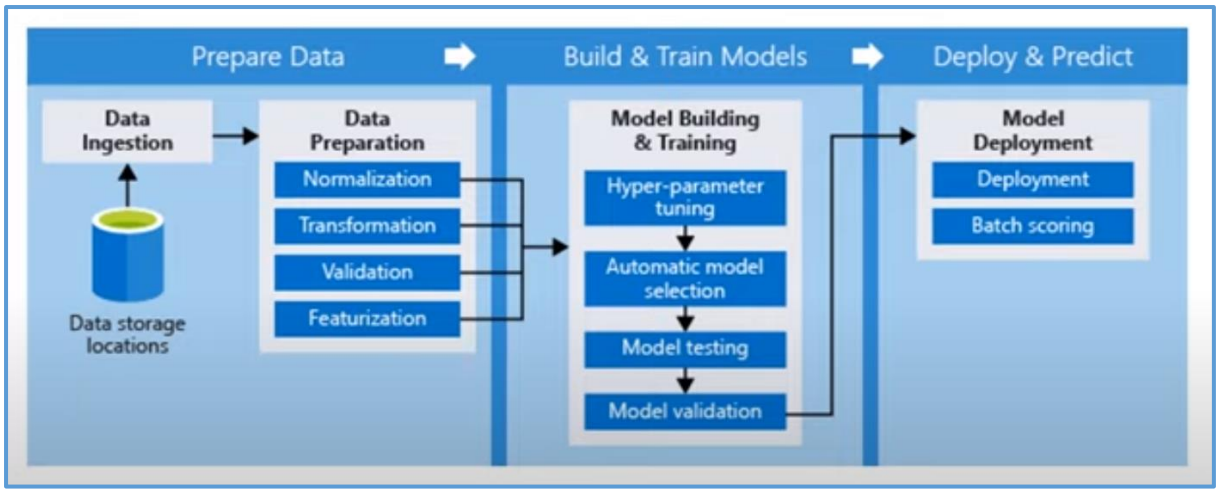
As the process of building your models becomes more complex, it becomes more important to get a handle on the steps to prepare your data and train your models in an organized way. In these scenarios, there can be many steps involved in the end-to-end process, including:

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

These steps are, organized into machine learning pipelines.

There are many steps one must take when training and deploying a machine-learning model.

From preparing data, to building and training the model, to deploying it for inferencing. You need to be able to organize the step so the end-to-end process is easily repeatable. This is where machine-learning pipelines can help. You use machine-learning pipelines to create and manage workflows that stitch together the machine-learning phases. There are cyclical and iterative in nature and facilitate continuous improvement of model performance, model deployment and making inferences over the best performing model to date. Machine-learning pipelines are, made up of distinct steps. Let us walk through a typical example. First, we have data preparation, where you ingest data from one or more sources, then prepare it for model training.

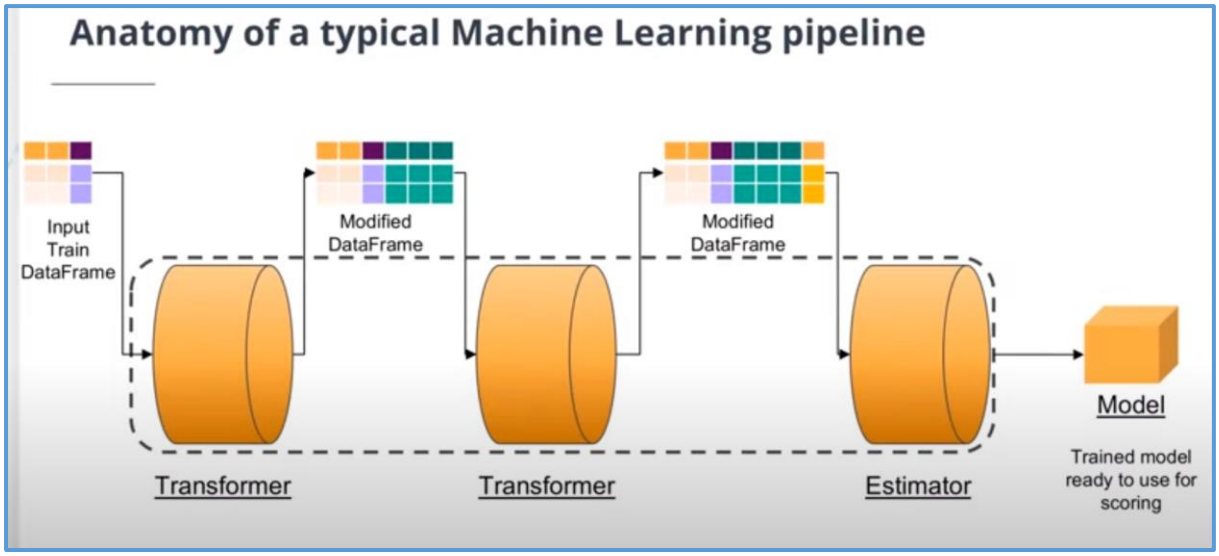


Some examples of data preparation include normalization, transformation, validation, and featurization. The next step is model training, where you take the output of the data preparation steps and use the output to build and train your model. Some steps you might take in this phase of the pipeline are hyper-parameter tuning, automatic model selection, model testing, and model validation, which is, used to find the best performing model.

The final phase of the machine-learning pipeline includes model deployment and optionally making batch predictions.

Often, data scientists, data engineers, and IT professionals need to collaborate on building robust, scalable, and reusable machine-learning pipelines.

The pipelines ensure you have a repeatable process when you need to retrain your models. Another important aspect of machine-learning pipelines is that they are modular. Because of this, pipeline steps are usable and can run without rerun in subsequent steps if the output of the steps have been, changed. For example, you can retrain a model without redoing costly data preparation steps if the data has not changed. Pipelines also allow data scientists to collaborate while working on separate areas of the machine learning workflow.



This diagram shows a pipeline represented by the dashed lines forming a rectangle around the transformers and estimators. In a typical scenario, you have multiple transformer steps and have an estimator at the end that works to modify the data. This shows a high-level view of creating a pipeline for repeatable data preparation and model training. There are typically several transformers. We are just showing two as an example. The entry point of the pipeline and the data input or ingestion of the training data.

The first transformer can represent data preparation steps, such as normalization and data transformation. For example, as a first step, you might convert datatypes so they are consistent from multiple sources. I reduced the columns to only those that you need. The output of the first transformer is a modified Dataframe that gets, used by the subsequent steps of the pipeline. The second transformer works to modify the Dataframe by applying additional data preparation steps such as validation and featurization. An example here is to take the fields from the first modified Dataframe that contains string data and apply one-hot encoding on them. Therefore, we have a numeric representation available to the estimator. The output of the second transformer is a new modified Dataframe. The last step of the pipeline is an estimator, which processes the modified Dataframe using the selected algorithm to train the model. The final output is the trained model. As the last step of this pipeline, you might register and deploy the model to one of the available deployment compute targets.

MLOps: Creating Automatic End-to-End Integrated Processes

As we said earlier, we do not 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.

DevOps has become ubiquitous in the world of classical development. Almost all projects that exceed a certain level of complexity become inevitably DevOps projects. Yet there is one category, of projects, that are stepping out, of the line. You guessed it right, it's the category of data science projects.

When it comes to DevOps, data science projects pulls a range of special challenges, whether it's about the technical side of things, the philosophy of the people involved, or the actors involved.

Think about one simple example, versioning. While in classical development projects, versioning refers almost exclusively to source code. In the world of data science, it gets another important aspect, data version. It's not enough to know the version of the code for your model, it's equally important to know the version of the data it was trained on.

Another interesting question is, for example, what does a build mean in the world of data science or a release? MLOps is, the discipline that deals with the application of classical DevOps principles to projects that have at least one data science component.

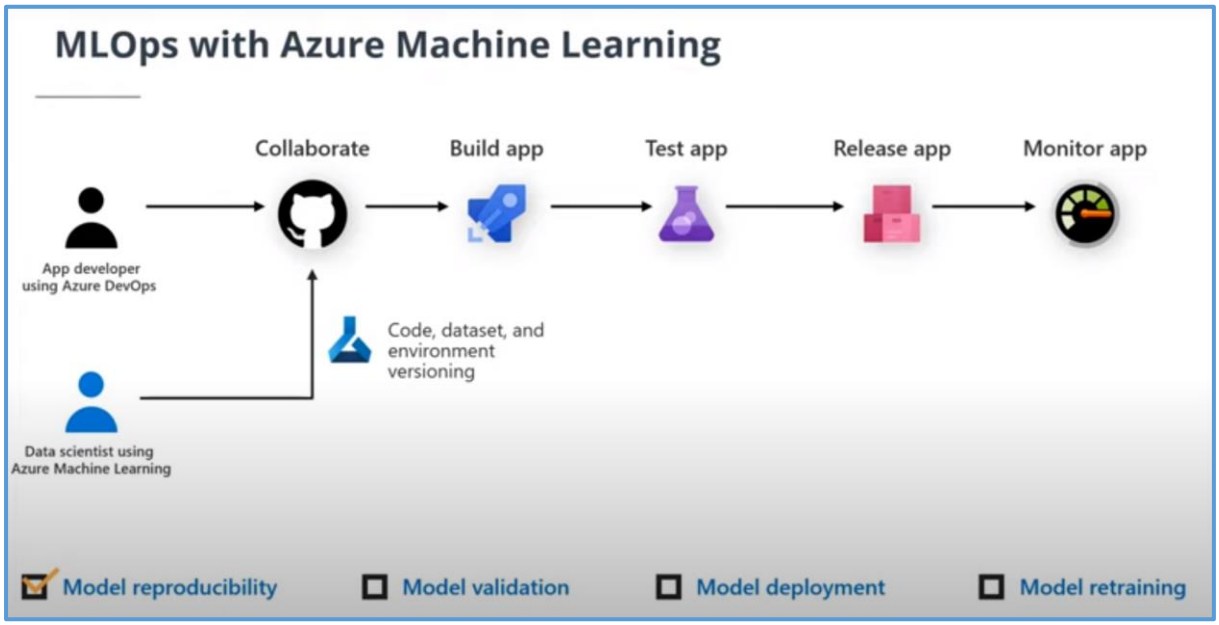
As the title implies, MLOps is DevOps for AI. Most important aspects of MLOps include: automating the end-to-end ML life cycle, using a combination of Azure Machine Learning services and Azure DevOps, monitoring ML solutions for both generic and ML specific operational issues, and capturing all data that is necessary for full traceability in the ML life cycle. A DevOps process enables repeatable and consistent code maintenance, testing an application deployment, capabilities within an organization. MLOps applies the same principles to machine learning.

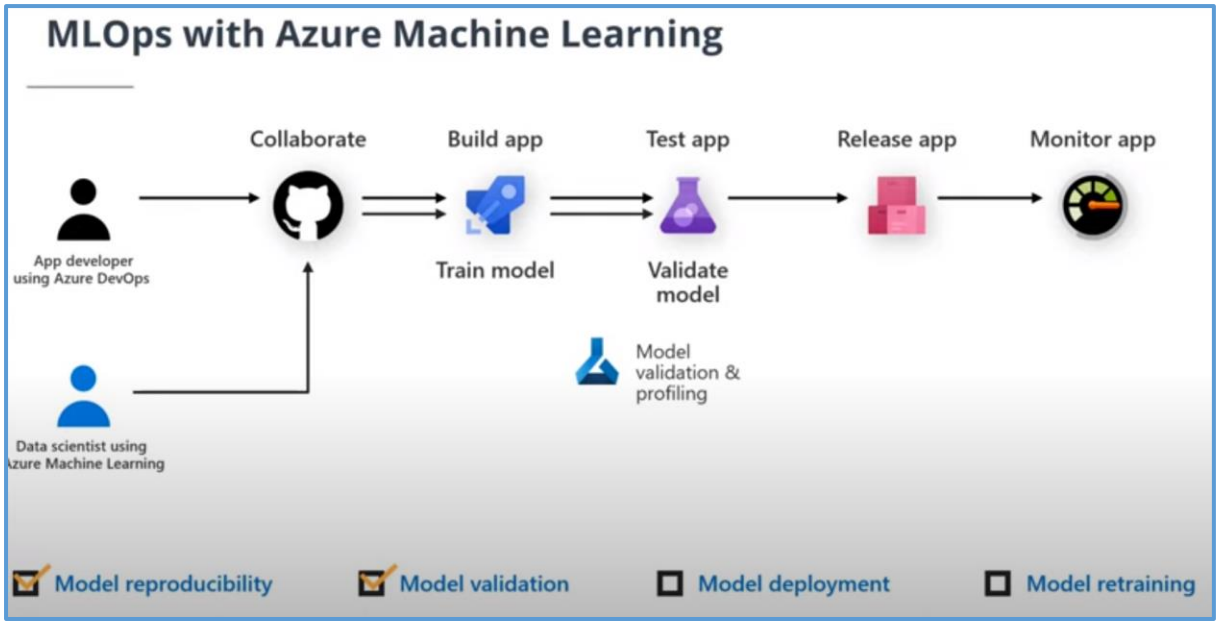
When viewed side-by-side, there are a lot, of parallels that are apparent between DevOps and MLOps. At the top of this diagram, you will see the steps an application developer follows when following standard DevOps processes.

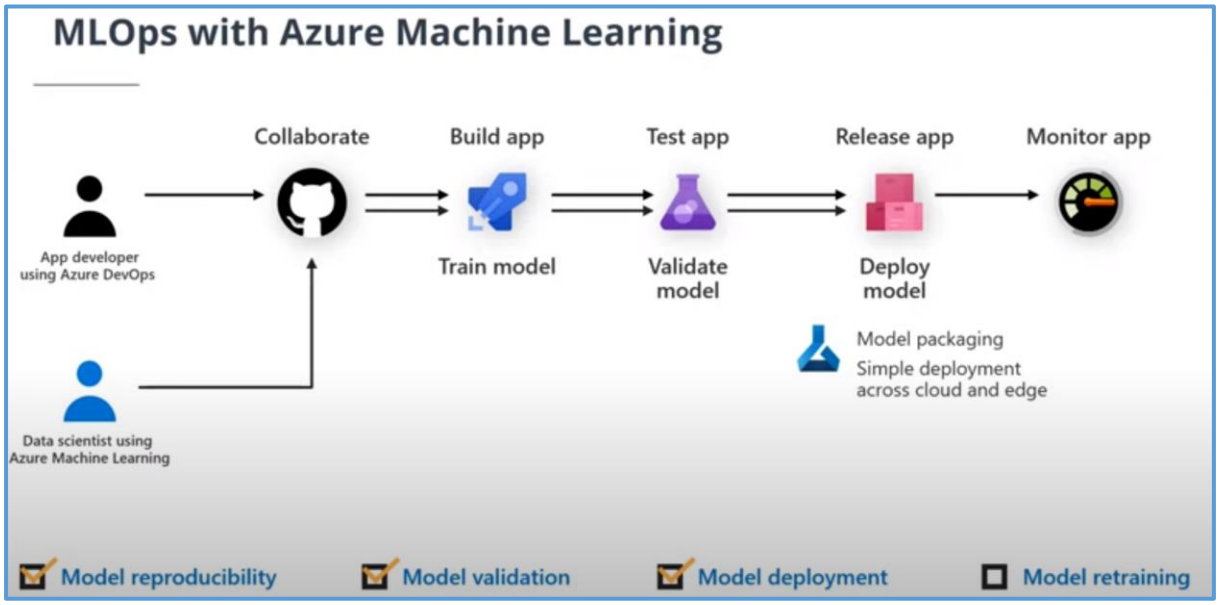
Developers collaborate on the application code within a code repository, automate the build process, an application test, then deploy the application as a new release to one or more environments.

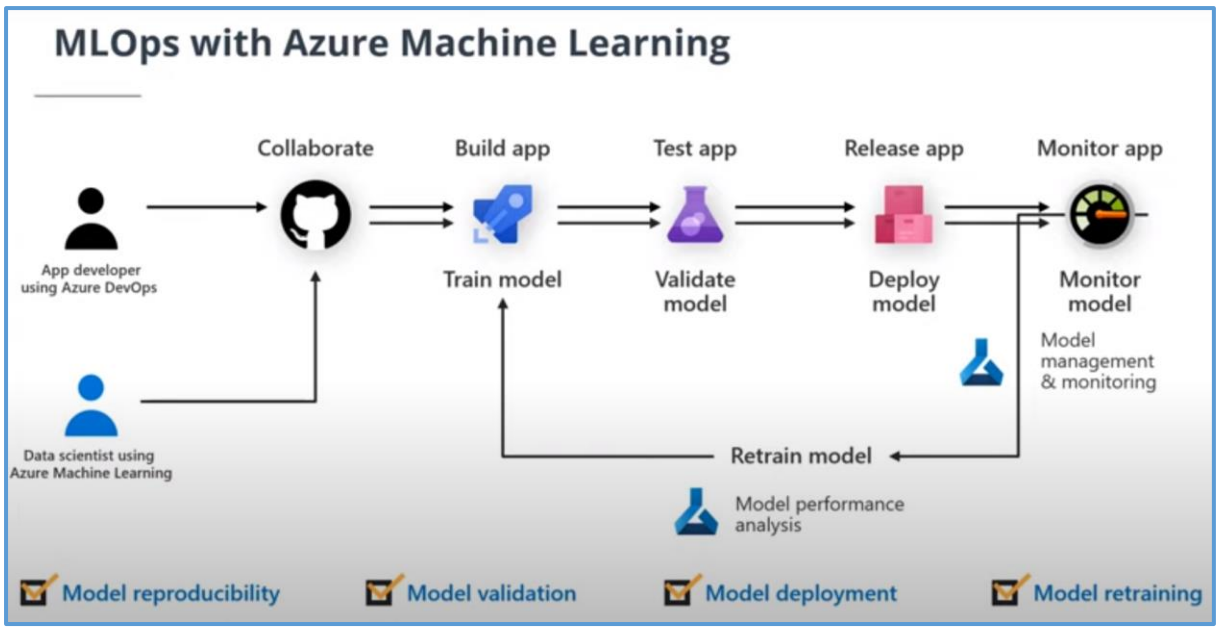
After the end of this process, they monitor the application's performance and ensure new deployments don't cause regressions or other problems. Data scientists following an MLOps process can obtain model reproducibility through code, dataset, and environment versioning. The resulting machine learning pipelines allow data scientists to define repeatable and reusable steps for data preparation, training, and scoring processes.

MLOps supports model validation and profiling within Azure Machine Learning experiments.




MLOps with Azure Machine Learning
Collaborate
Build app
Test app
Release app
Monitor app
App developer using Azure DevOps
Train model
Validate model
Model validation & profiling
Data scientist using Azure Machine Learning
Model reproducibility
Model validation
Model deployment
Model retraining


MLOps with Azure Machine Learning
Collaborate
Build app
Test app
Release app
Monitor app
App developer using Azure DevOps
Train model
Validate model
Deploy model
Model packaging
Simple deployment across cloud and edge
Data scientist using Azure Machine Learning
Model reproducibility
Model validation
Model deployment
Model retraining



From this diagram, it is clear to see the parallels between DevOps processes and MLOps. MLOps supports the packaging and deployment of the model across the cloud and edge devices. Model deployments is similar to the release of an application in a DevOps process. MLOps enables monitoring the model in production, which can kick-off retraining automatically.

Chapter 16: Prelaunch Lab

Prelaunch your lab environment.

NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

Your lab has been, reserved! Please continue to the next concept.

Chapter 17: Operationalizing Model

After you have trained your machine-learning model, and evaluated, it to a point where you are ready to use it outside your development or testing environment, you need to deploy it somewhere. Another term for this is operationalization. Azure Machine Learning Service makes this process easier than the traditional deployment steps. From a very high level, a typical model deployment goes like this:

- Get the model file of any format
- Create a scoring script
- Optionally, create a schema file describing the web service input
- Create a real-time scoring web service
- Call the web service from your application
- Repeat this process, each time, you re-train, the model.

Azure Machine Learning Service simplifies this process by providing tools that help automate these steps. When you deploy a model, you either deploy it to a service for real-time inferencing, or make it available for batch inferencing.

- Real-time inferencing means that you call the web service that hosts the model to quickly score against a relatively small set of data.
- If you need to score a large volume of data, you would normally use the model in a batch process.

Real time Inferencing

In the previous lessons, we spend 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 time spanning across many hours, days, and weeks depending on the amount of data, type of algorithm use 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 has given based on its training. Making these decisions or new data on-demand is, called real time inferencing.

Regardless of what you use to deploy model, you generally do the following.

- Get the model file in any format such as *.pkl format
- Create a scoring script, for instance, within a Python file(.py)
- Optionally, create a schema file describing the web service input, which is typically in json format.
- Create a real time score in web service
- Call the web service from your application and repeat this process each time you retrain the model.

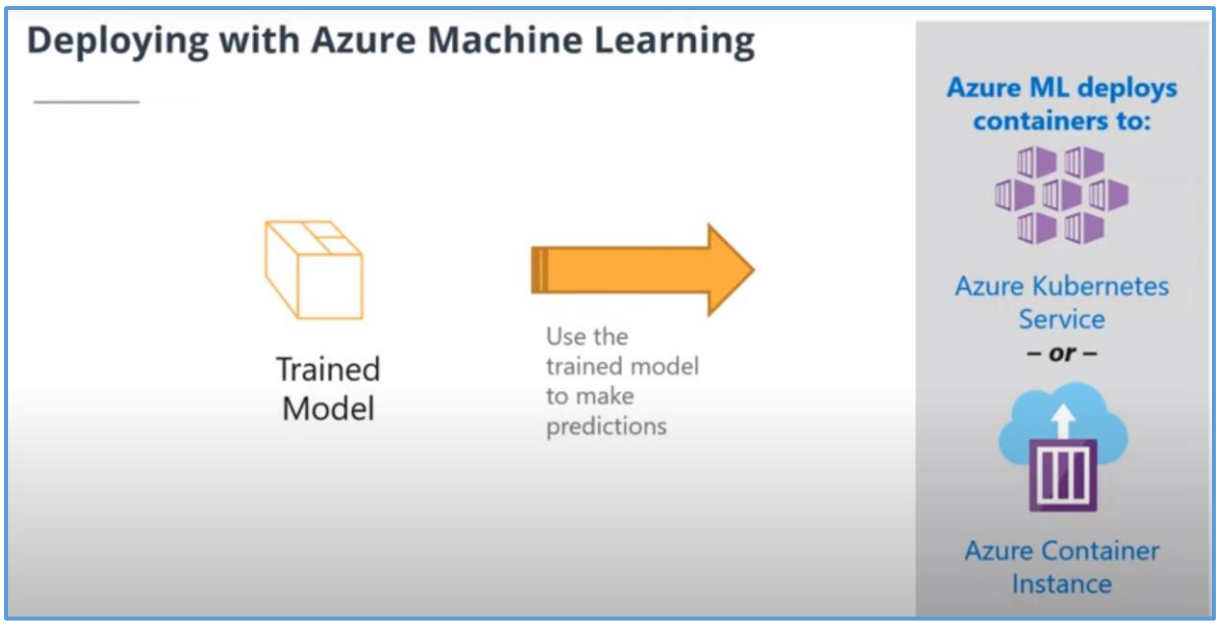
This process is very time consuming if done manually, especially in creating a real-time scoring web service. Azure Machine Learning simplifies the model deployment process by providing a framework that helps you perform manual steps and automatically deploys the model to one or more targets.

You can use Azure Machine Learning to deploy a model as a web service on Azure Kubernetes Service, also known as AKS.

Azure Kubernetes Service is good for high scale production deployments.

Use Azure Kubernetes Service if you need one or more of the following capabilities; fast response times, auto scaling of the deployed service, hardware acceleration options such as GPUs or field programmable gate arrays, also known as FPGAs.

Use Azure Container Instances if one of the following conditions is true, you need to quickly deploy and validate your model or you are testing a model that is under development.



Let us talk about the options for creating real-time inference in environments.

- One option is to create the environment manually using the Azure Machine Learning Studio user-interface.
- Another is to create the environment programmatically using the Azure Machine Learning Python SDK

Batch Inferencing

Batch inferencing or batch scoring is the process of using Machine Learning to make predictions on large quantities of existing data. Unlike real-time scoring, which makes predictions on data as it receives it, batch inference is commonly run on a reoccurring schedule against data stored in a database or other data stores. The resulting predictions are then, written to a data store for later use by applications, data scientist, developers, or end-users. Batch inference typically involves latency requirements of hours or days. It does not require using train models deployed to restful web services, as is done for real-time inference.

Instead, batch predictions performed using Azure Machine Learning can take advantage of a host of high throughput and scalable compute targets. This provides a cost effective means of making predictions on a large volume of data asynchronously. Consider using batch inference when you do not need to make or act on predictions in real time.

Inferences for large sets of existing data that can be, created on a schedule and written to data stores for later use.

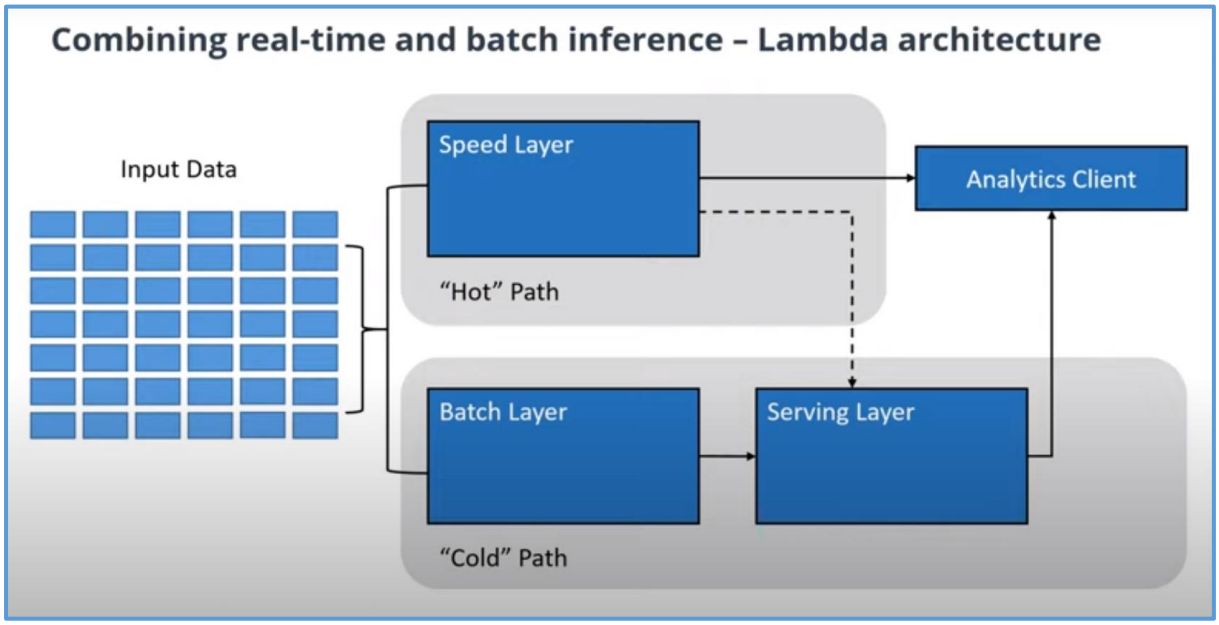
Post-processing or analysis of the predictions generated during inference are, needed before exposing the results for further use. The complexities of data processing requires long run in scoring operations or the scale out of compute resources to perform parallel processing. Due to the scheduled nature of batch inferencing, predictions are not usually available for new data.

However, in many common real-world scenarios, predictions are, needed on both newly arriving and existing historical data. In these cases, the Lambda architecture provides a good solution. The gist of the Lambda architecture is that ingested data is, processed at two different speeds.

A hot path that tries to make predictions against the data in real time, and a cold path that makes predictions in a batch fashion, which might take days to complete.

- In the cold path, the batch layer stores all the incoming data in its raw form and performs batch inference on the data in bulk fashion and usually on a predefined schedule. The prediction results are stored within the serving layer, which is typically a database for later use by the analytics client.
- In the hot path, the predictions are, made against the incoming data as quickly as it arrives; the predicted results are, made available to both the analytics client and stored in the serving layer.

Eventually, the hot and cold path converge at the analytics client application.



Let us put this in context with a concrete scenario, fraud detection in banking.

In this scenario, we use batch scoring to predict the total number of fraudulent transactions in a period, of time, such as within a given month.

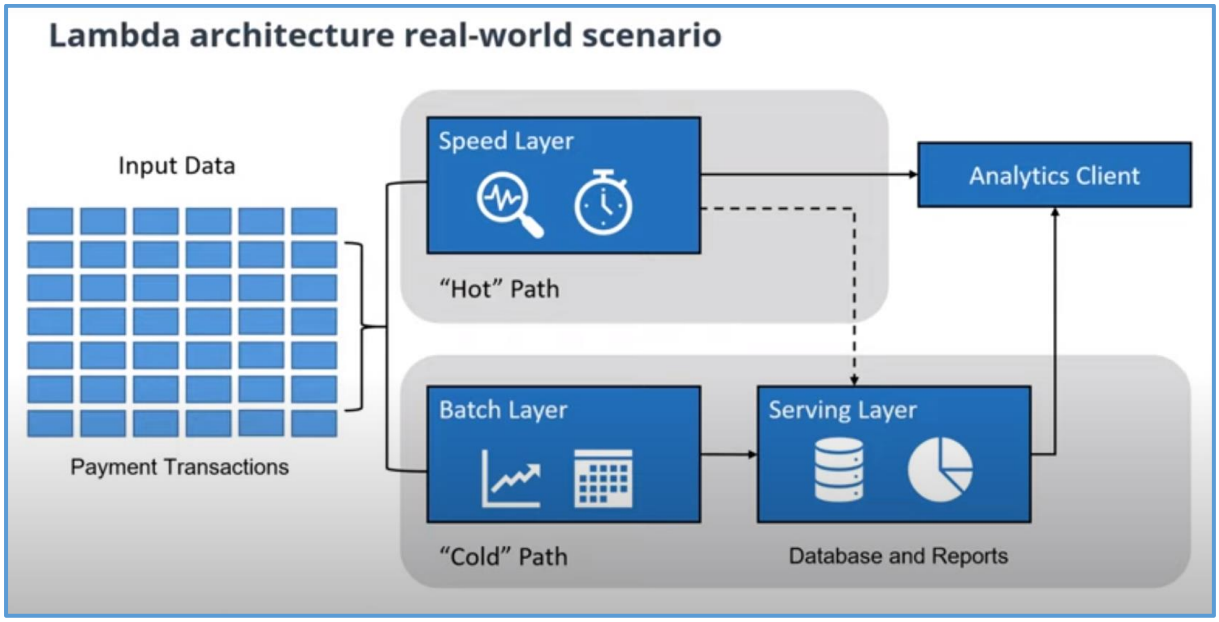
Batch scoring is, needed because you typically want to consider historical time-based data that are as long as possible; you want to predict the value at a certain level of granularity, like a category or subcategory of customers, for example. On the other hand, we could use real-time scoring to flag potentially fraudulent transaction based on properties of each transaction, as the transactions are happening in the real world. Results from both processes, batch and real-time scoring will then aggregate in the Analytic Engine where fraud related decisions would emerge.

For example, when the daily number of potential fraudulent transactions is surpassing the daily average resulting from the monthly prediction for three days in a row, an auditing action could be, triggered.

Let us say for our input data, we have payment transactions streaming in from one or more sources. As we see in the Lambda architecture diagram, the data splits into two paths.

A hot path where we process the transactions in real-time for fraud using our models for real-time scoring and a cold path where all the data persist in a long-term storage for batch

processing. We use different models to look for longer running trends in the transactions that are indicative of fraud. The batch and real-time scoring results are, saved to the database in the serving layer for reporting and auditing



QUIZ QUESTION

When you deploy a Machine Learning model used for real-time scoring, or batch inferencing, there are several steps you must follow. Select the *required* steps in the list below.

(Select all that apply.)

- [x] Save and retrieve the model file in any format
- [ ] Create a metadata file that describes the model
- [ ] Create a training script
- [x] Create a scoring script
- [ ] Create a schema file that describes the web service input
- [x] Create a real-time scoring web service

Chapter 18: Lab: Deploy a Model as Web Service

Compute Resources
Deploy a trained model as a web service

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 web service, hosted on an Azure Kubernetes Service (AKS) cluster. This process is what enables you to use your model for real-time inferencing.

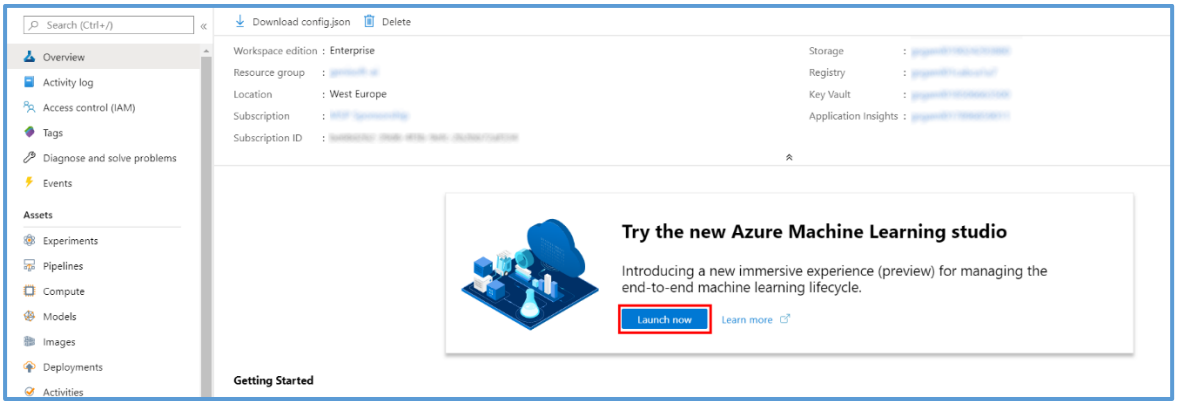
The Azure Machine Learning designer simplifies the process by enabling you to train and deploy your model without writing any code.

Exercise 1: Open a sample training pipeline

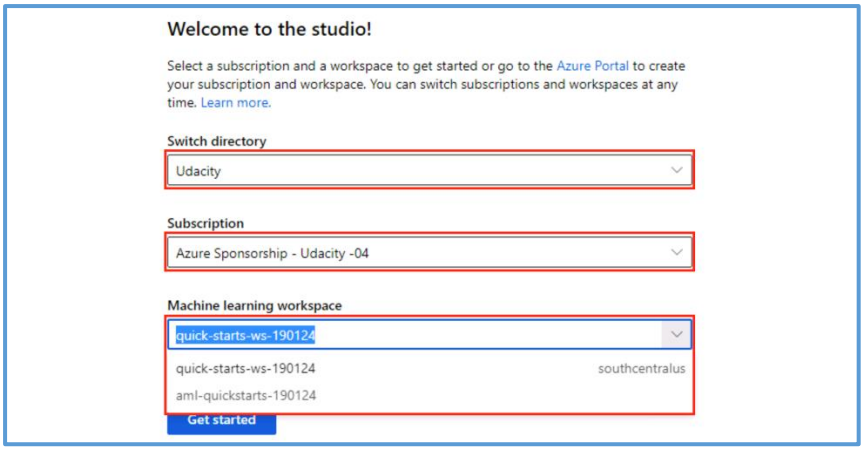
Task 1: Open the pipeline authoring editor

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.

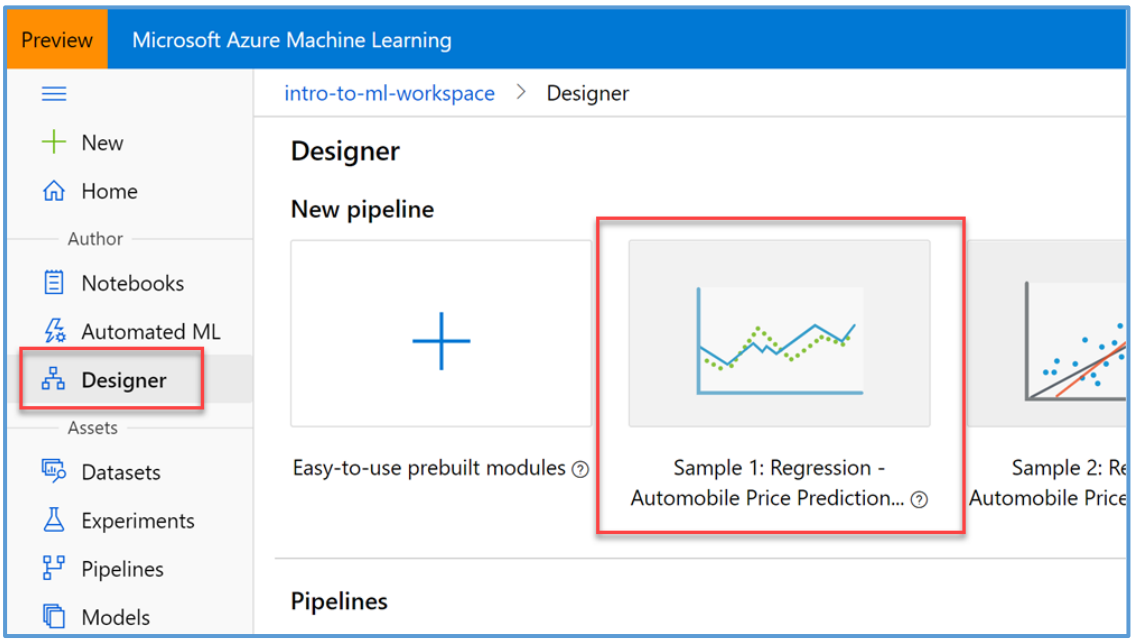


3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:



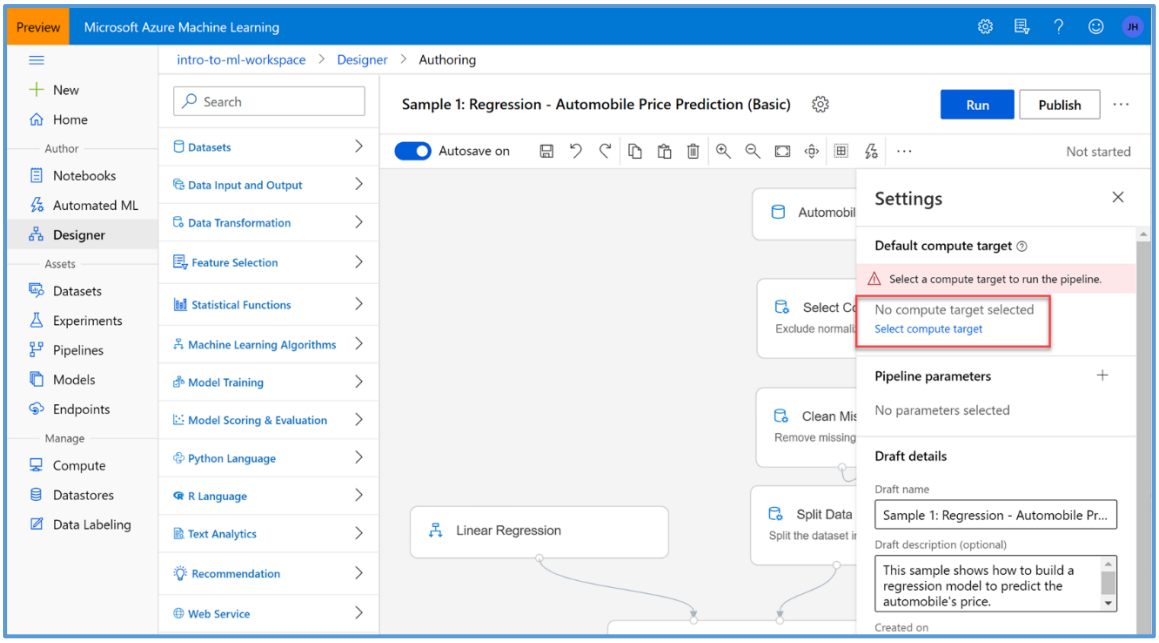
For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it doesn't matter which) and then click **Get started**.

4. From the studio, select **Designer** in the left-hand menu. Next, select **Sample 1: Regression - Automobile Price Prediction (Basic)** under the **New pipeline** section. This will open a visual pipeline authoring editor.



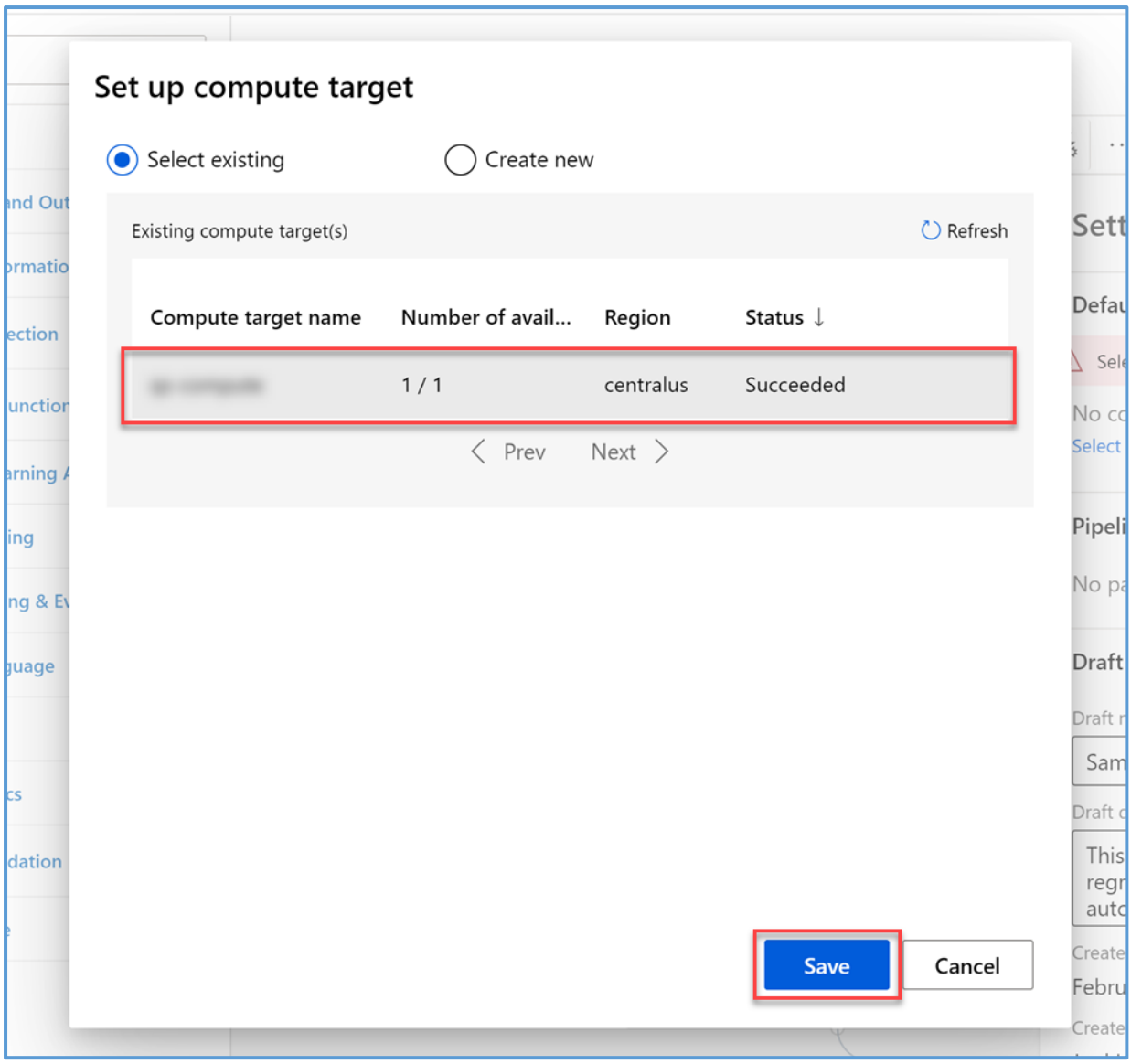
Task 2: Setup the compute target

1. In the settings panel on the right, select **Select compute target**.



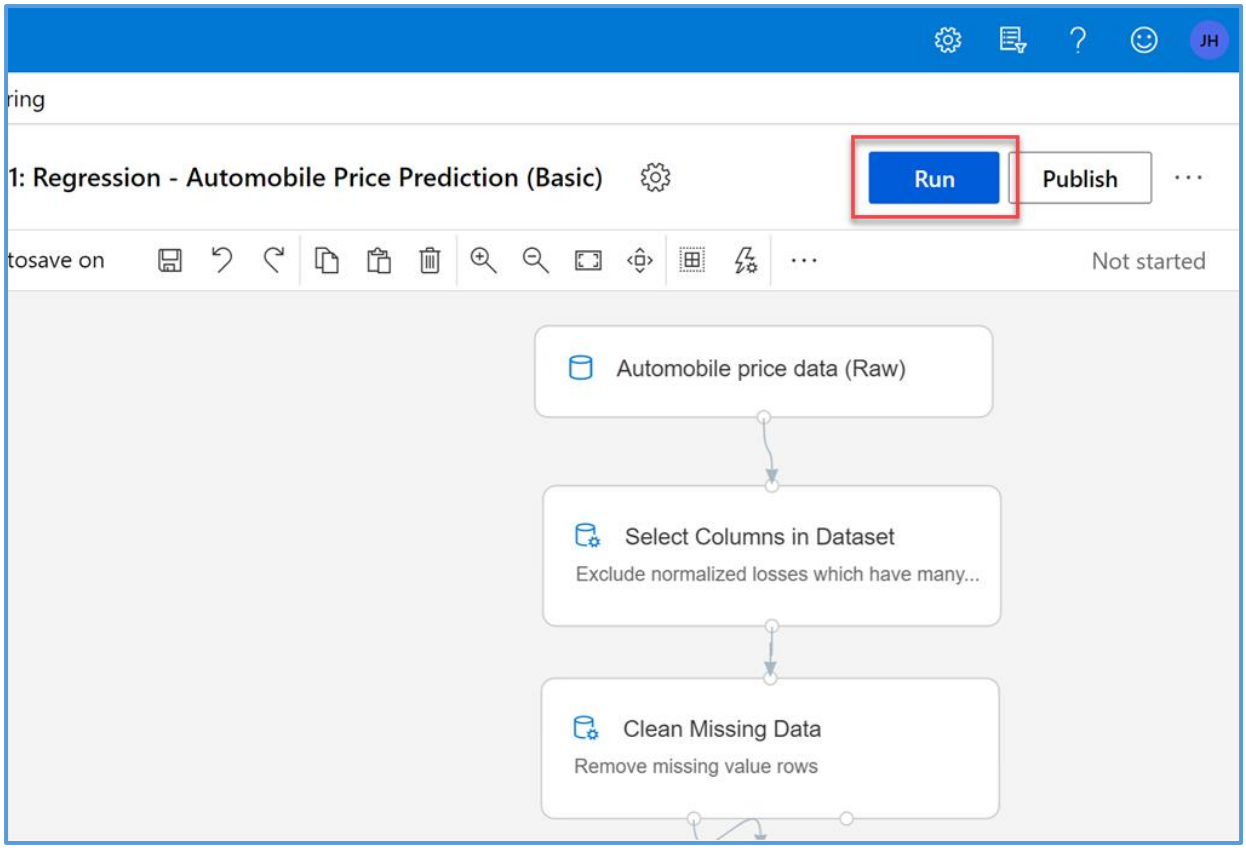
2. In the Set up compute target editor, select the existing compute target, then select **Save**.

Note: If you are facing difficulties in accessing pop-up windows or buttons in the user interface, please refer to the Help section in the lab environment.



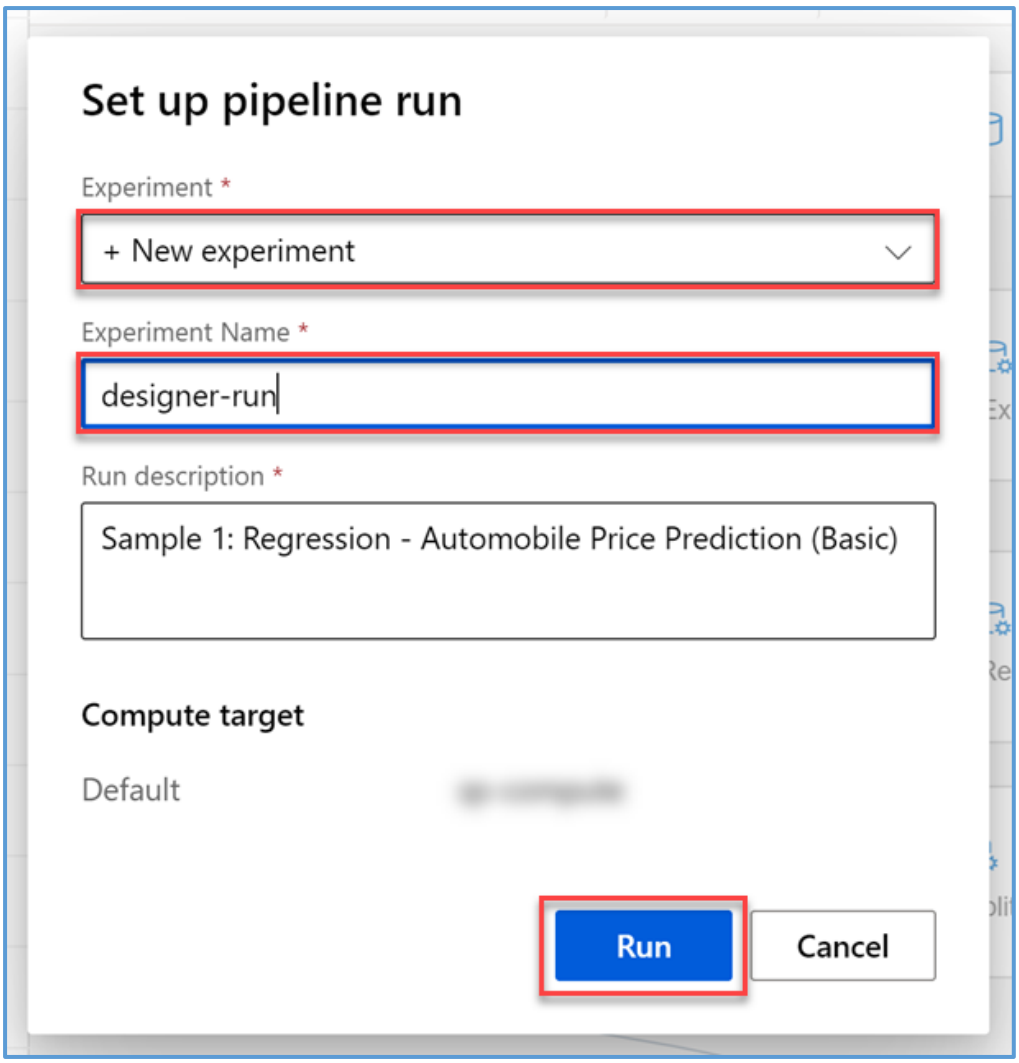
Task 3: Create a new experiment and submit the pipeline

1. Select **Submit** to open the Set up pipline run editor.



Please note that the button name in the UI is changed from **Run** to **Submit**.

2. In the Setup pipeline run editor, select **Experiment, Create new** and provide New experiment name : **designer-run**, and then select **Submit**.

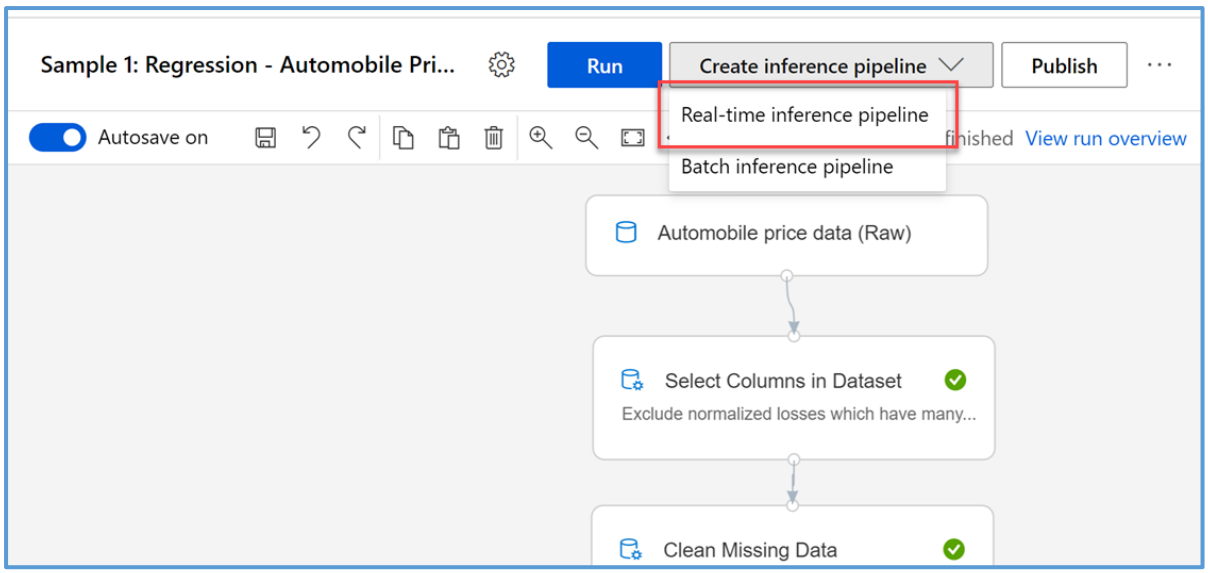


3. Wait for the pipeline run to complete. It will take around **10 minutes** to complete the run.

Exercise 2: Real-time inference pipeline

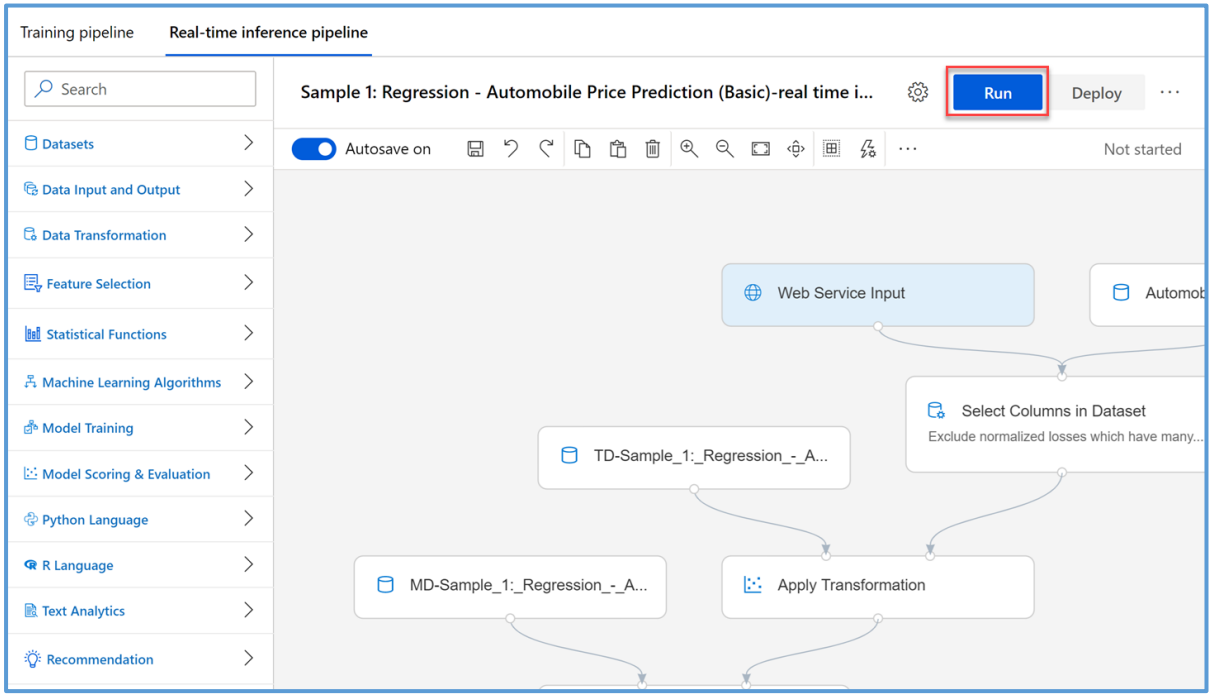
Task 1: Create pipeline

1. Select **Create inference pipeline**, then select **Real-time inference pipeline** from the list to create a new inference pipeline.



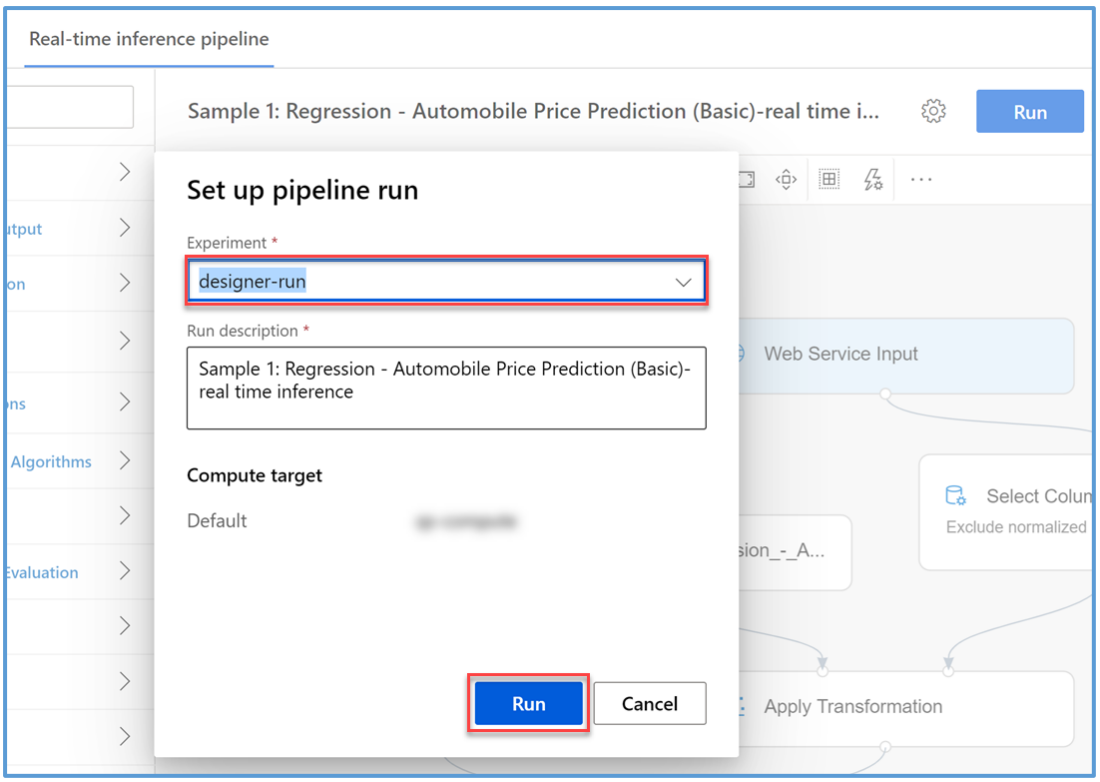
Task 2: Submit the pipeline

1. Select **Submit** to open the Set up pipeline run editor.



Please note that the button name in the UI is changed from **Run** to **Submit**.

2. In the Setup pipeline run editor, select **Select existing**, then select the experiment you created in an earlier step: **designer-run**. Select **Submit** to start the pipeline.

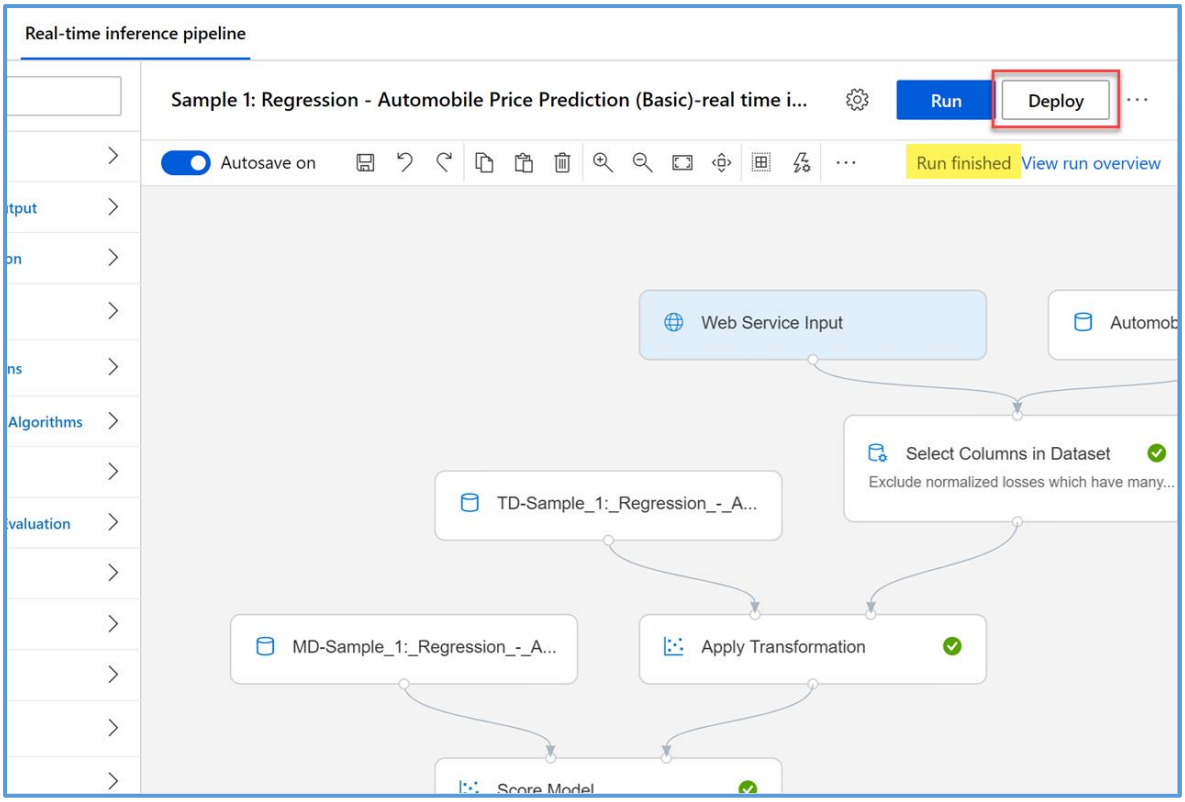


3. Wait for pipeline run to complete. It will take around **7 minutes** to complete the run.

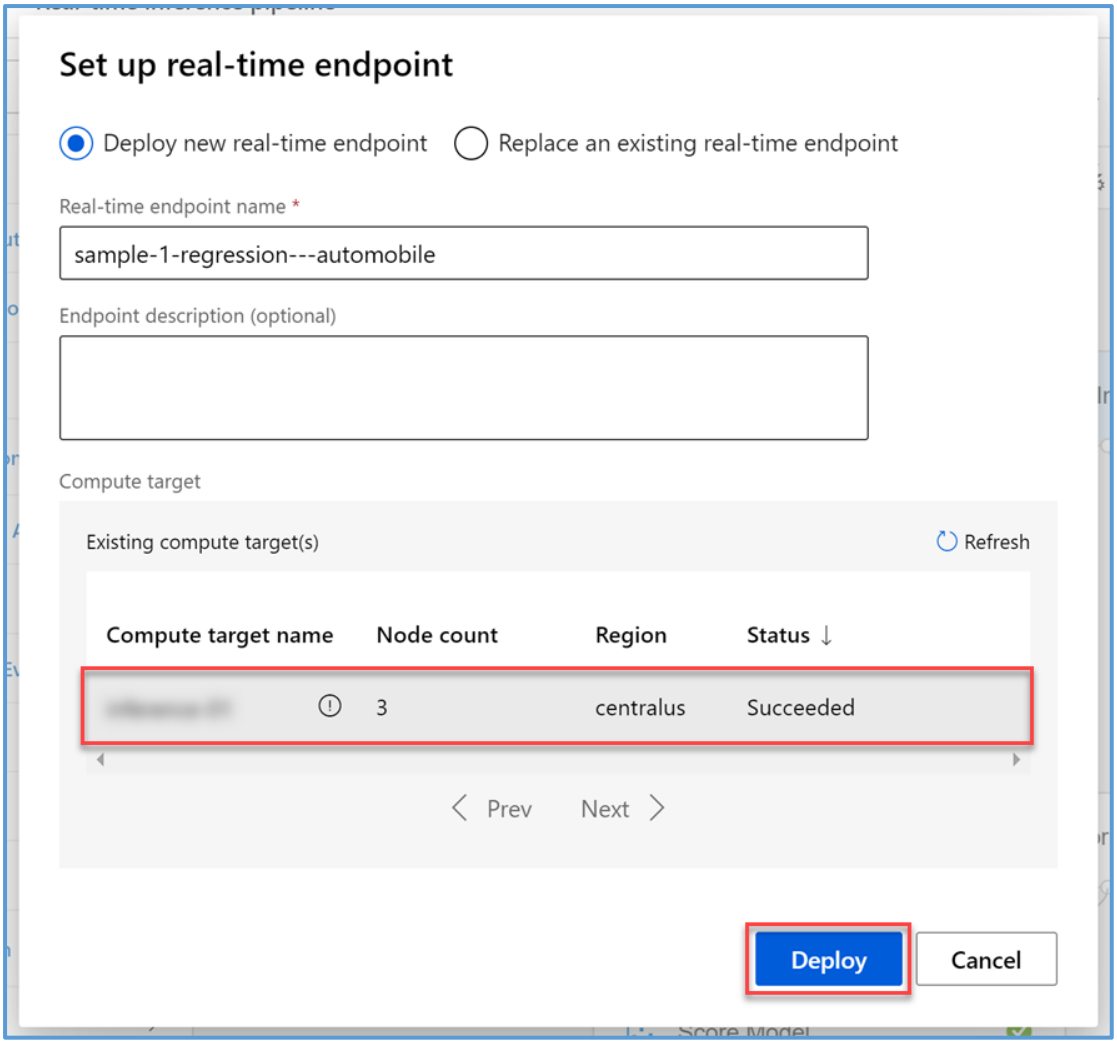
Exercise 3: Deploy web service on Azure Kubernetes Service compute

Task 1: Deploy the web service

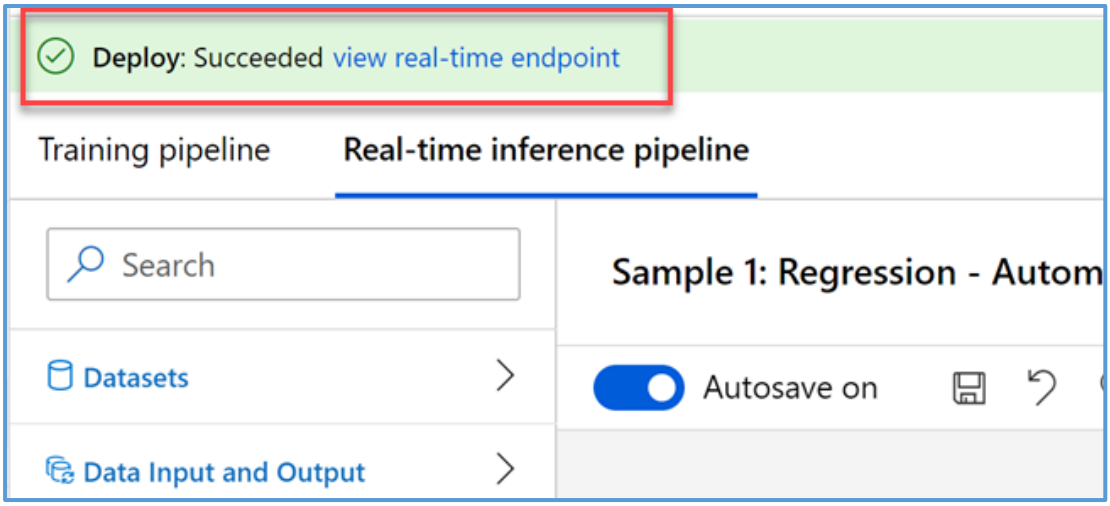
1. After the inference pipeline run is finished, select **Deploy** to open the Set up real-time endpoint editor.



2. In the Set up real-time endpoint editor, select your **existing compute target**, then select **Deploy**.



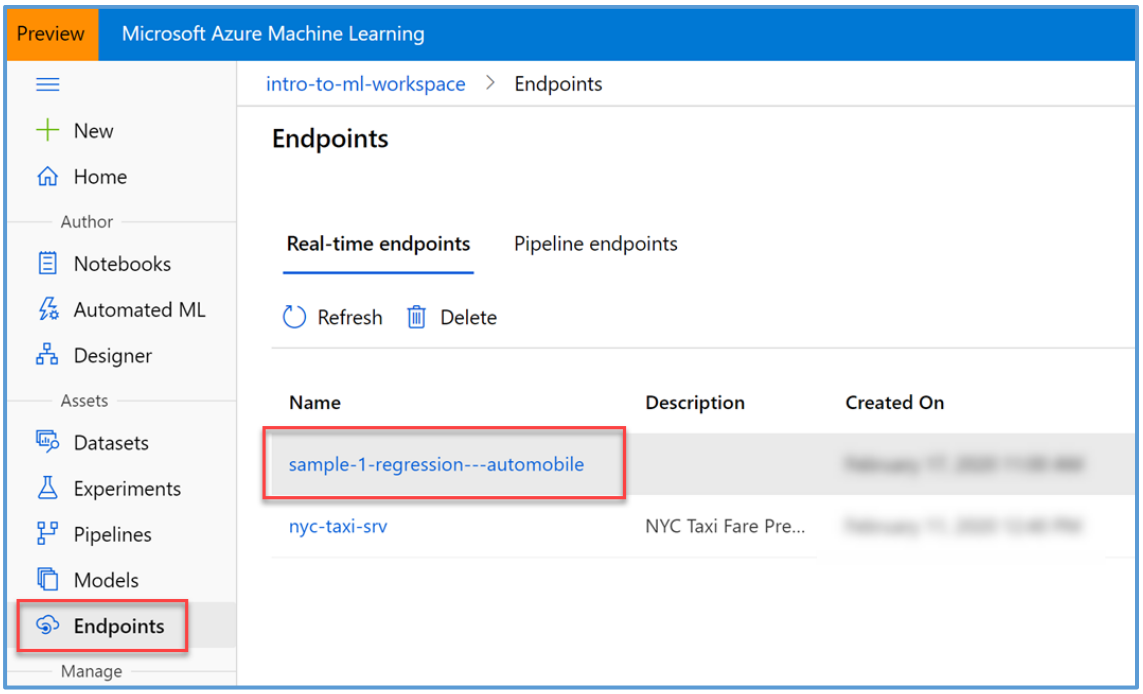
3. Wait for the deployment to complete. The status of the deployment can be observed above the Pipeline Authoring Editor.



Task 2: Review deployed web service

1. To view the deployed web service, select the **Endpoints** section in your Azure Portal Workspace.

2. Select the deployed web service: **sample-1-regression—-automobile** to open the deployment details page.



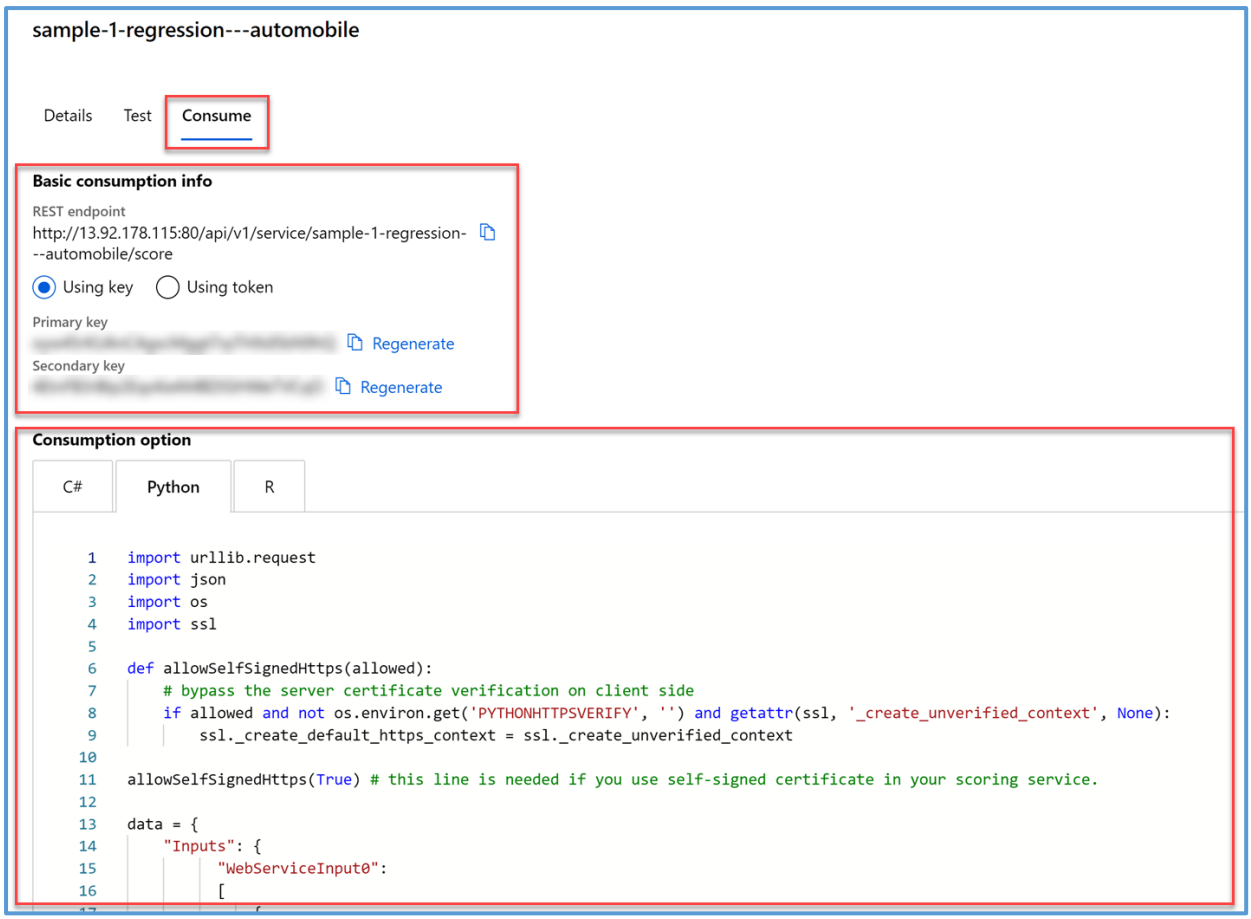
Note: you have to select the text of the service name to open the deployment details page

Task 3: Review how to consume the deployed web service

1. Select the **Consume** tab to observe the following information:

    1. Basic consumption info displays the **REST endpoint**, **Primary key**, and **Secondary key**.

    2. Consumption option shows code samples in **C#**, **Python**, and **R** on how to call the endpoint to consume the webservice.



Next Steps

Congratulations! You have just learned how to train and deploy a model to an Azure Kubernetes Service (AKS) cluster for real-time inferencing. You can now return to the Udacity portal to continue with the lesson.

Chapter 19: Walkthrough: Deploy a Model as Web Service

Section A:

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 time 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 data quickly.

In other words, it infers things about new data it has given based on its training. Making these decisions on new data on demand is, called real-time inferencing.

In this lab, you will learn how to deploy a trained model that can be, used as a web service hosted in Azure Kubernetes Service Cluster. This process is what enables you to use your model for real-time inferencing.

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

For the first task, within the Azure Machine Learning Studio, we will select "Designer" on the left-hand menu.

Next, we select "Sample 1: Regression - Automobile Price Prediction (Basic)" under the New pipeline section, we will click this. This will open the visual pipeline author in editor.

After that is, done, in the Settings panel on the right, we should select a compute target. We can select any of the compute targets we have available or we can create a new compute target. I have a few available, so I am going to select one of them.

After that, you hit "Save". In order to run the experiment, we will hit the "Submit" button. After we hit "Submit" button, we can create a new experiment and name the new experiment designer-run.

We will lead the description the same and hit "Submit".

After this experiment is, submitted, it will take around 10 minutes to complete this run.

Now we wait.

Section B:

Now we are going to create an inference pipeline. To create an inference pipeline, you hit "Create inference pipeline."

We click "Real-time inference pipeline," from the list to create a new inference pipeline, wait for it to load.

You can see right here, real-time inference pipeline has been added.

Now we can submit this pipeline.

Either we can, create a new experiment or we can select the experiment we already have set up in the earlier step. "Designer-run." We can hit "Submit," and after that is, submitted, this is going to take around seven minutes to run. You can see right here, there is a check mark. Therefore, we are going to wait around seven minutes for this to finish.

Section C:

Now that the real-time inference pipeline has finished, we want to deploy this as a real-time endpoint. As you can see, that pipeline was able to take a web service input, take some automobile price data, do some of the transformations, score the model, and create a web service output and evaluate the model.

Now this pipeline is finished, we can deploy this model. Therefore, we will click here to deploy. In order to set up a real-time endpoint and we have opened the setup real-time endpoint editor, I want to select a compute target. Therefore, I already have a bunch of existing compute targets, so we are going to pick one and we are going to hit "Deploy". This is going to take a bit of time to deploy. Therefore, we are just going to wait until this has succeeded.

After that model has been deployed, we can head over to the Endpoint section on the left menu, we can find the deployed endpoint right here, click it, and now we can review different things about the endpoint, such as the details. We can see that the deployment has been healthy. We can see it AKS. We can see the ID, and we can see when it was, created and how it was created.

This was, created through the studio. We can also see different things about the endpoints such as the model CPU, the memory and much more. Next, we can hit the other tab and we can test the endpoint.

Therefore, we have a button here to test, and it has all the inputs you need to test. Next, you can consume the endpoint. So this tab would show you the basic consumption information such as the REST endpoint and how to do so using a key or using a token, and the consuming options such as C#, Python, or R. You could easily copy all the template code here and paste it into whatever application you are using.

Congratulations.

You have just learned how to train and deploy a model to Azure Kubernetes Service Cluster for a real-time inferencing.

Chapter 20: Prelaunch Lab

Prelaunch your lab environment.

NOTE: When you click "Prelaunch Lab", we will begin preparing a lab environment for you. When it is time to access the lab environment, this ensures you will be able to start working. Once your lab is reserved, you can continue on to the next concepts.

Your lab has been, reserved! Please continue to the next concept.

Chapter 21: Programmatically Accessing Managed Services

Data scientists and AI developers use Azure Machine Learning SDK for Python to build and run machine-learning workflows with the Azure Machine Learning service. You can interact with the service in any Python environment, including Jupyter Notebooks, Visual Studio Code, and your favourite Python IDE.

Azure Machine Learning provides a code-first experience via the Azure Machine Learning SDK for Python. 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 and highly accurate machine learning models. Azure Machine Learning Service supports many of the popular open source machine learning and deep learning Python packages such as Scikit-learn, TensorFlow, PyTorch, and Keras.

Managing datasets.

You can explore, prepare, and manage the lifecycle of your datasets used in the machine learning experiments.

Organize and monitor experiments.

You can manage cloud resources for monitoring login and organizing your machine learning experiments.

Train models.

You can train models either locally or by using cloud resources including GPU accelerated model training.

Automated machine learning.

You can use automated machine learning which accepts configuration parameters and training data, then automatically iterates through algorithms and hyperparameter settings to find the best model for running predictions.

Model deployment.

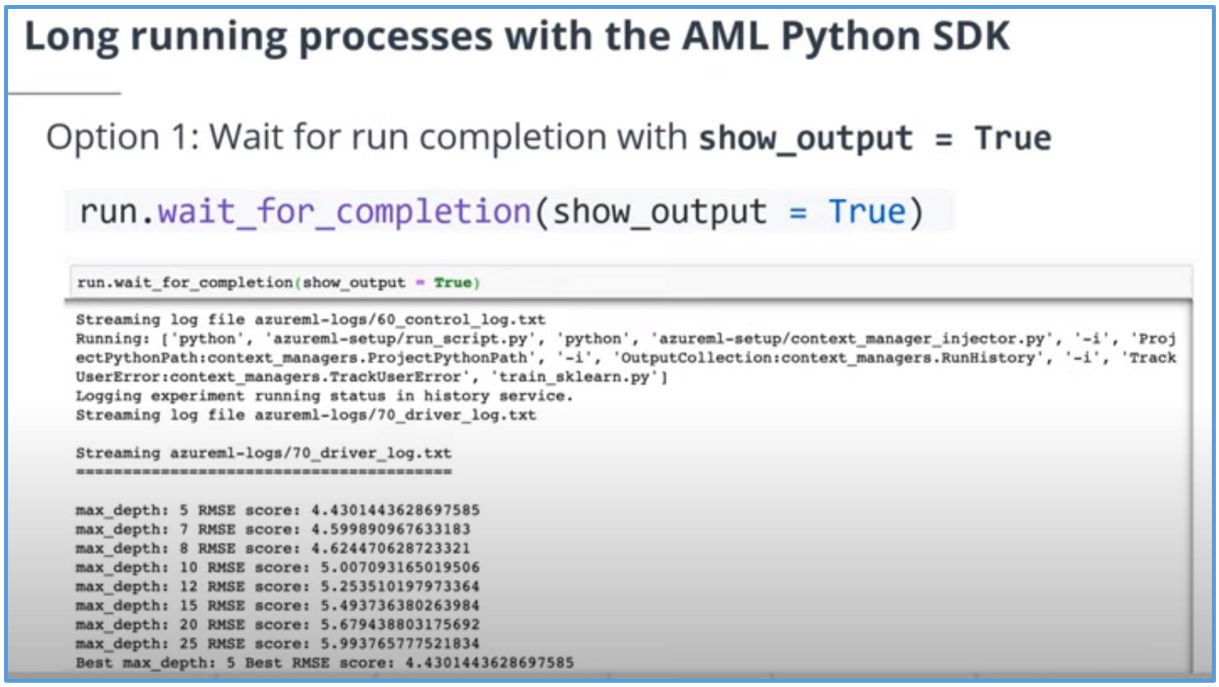
You can deploy Web services to convert your train model into a RESTful service that can be, consumed in any application. Most machine learning, processes are, long running, like model training, for example.

Azure Machine Learning Service provides the ability to manage your model training run from within your Python code. You can also query the run status, run details, cancel runs, or mark runs as complete.

Typically, the model training script is going to generate output and log metrics to the run as the model is, trained. You can monitor your training scripts output in real-time from within your Python notebooks in two common ways.

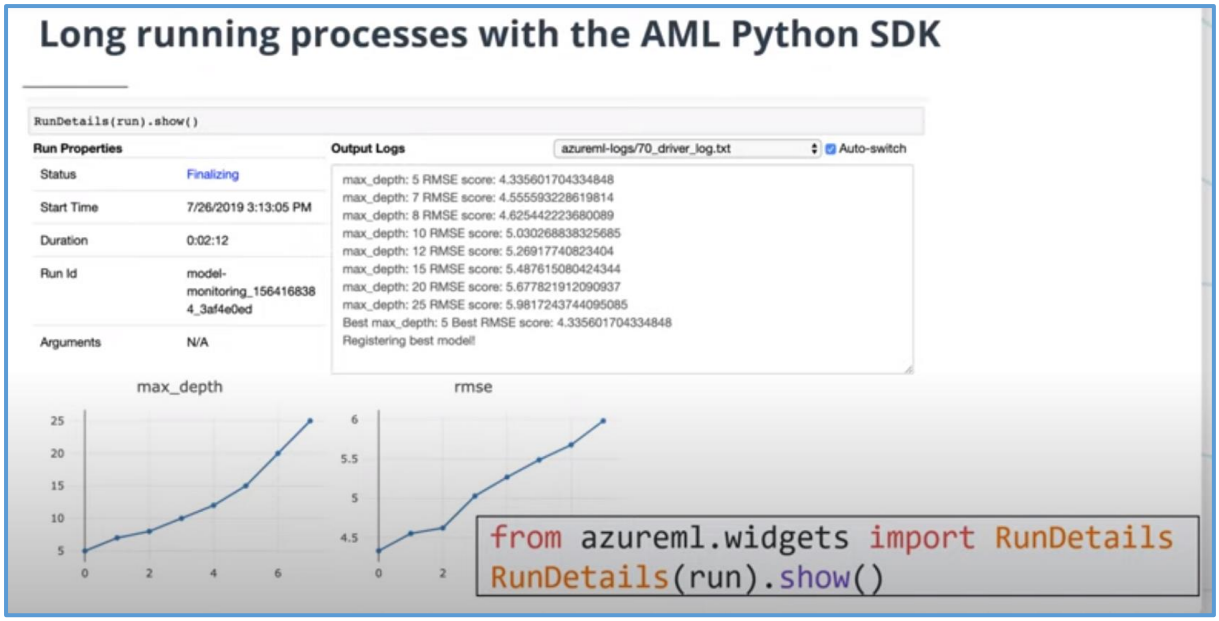
- **Option 1 is call run.wait_for_completion and set and show output to equal True.**

Notice how the generated output is progressively, displayed as its being, generated.



- **Option 2 is using a Jupyter Notebook widget named RunDetails.**

Here we show the second option for monitoring the model training progress from within your Notebook and its resulting outputs. Note how the displayed widget allows you to interact with the generated content.



Chapter 22: Lab: Training and Deploying from a Compute Instance

Compute Resources

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**.

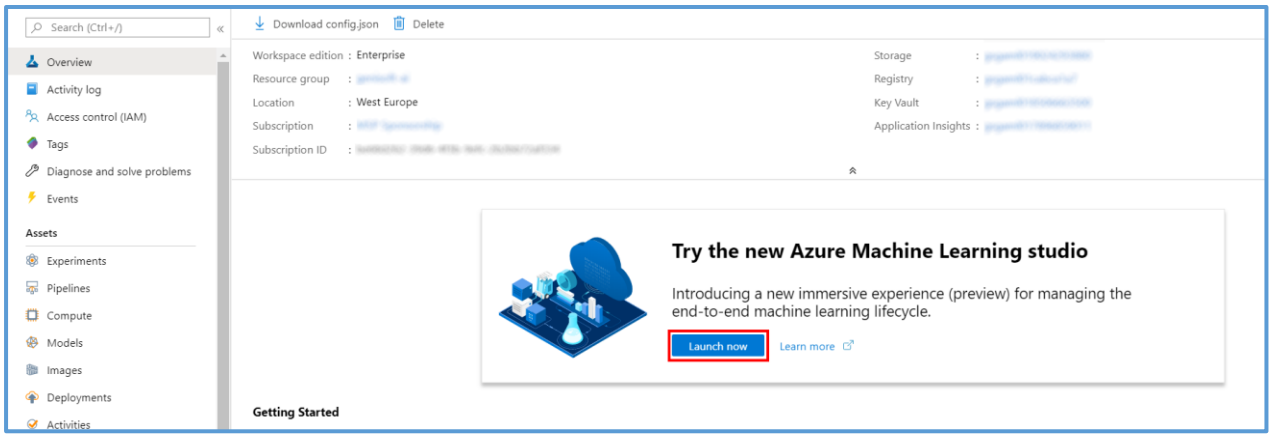
Overview

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.

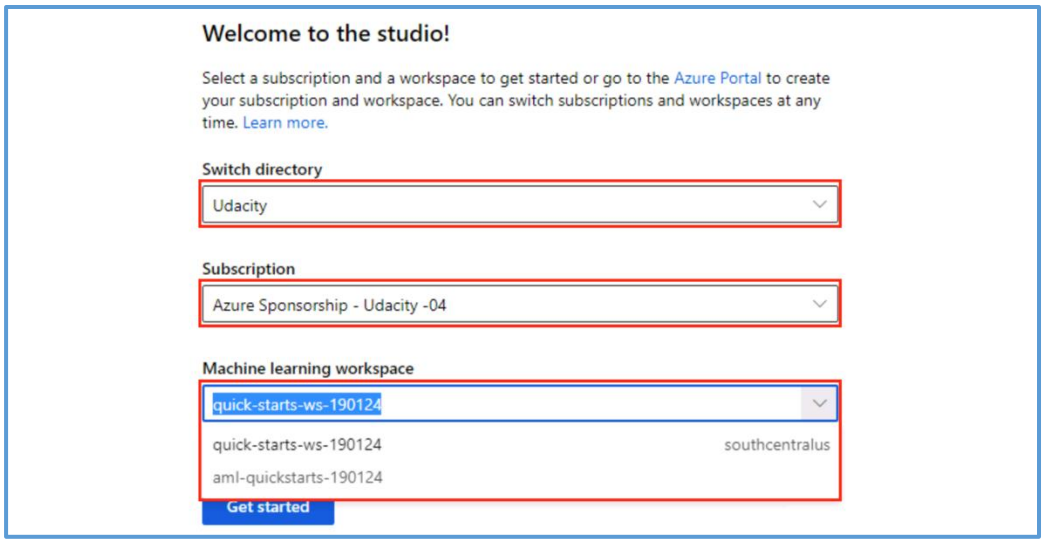
Exercise 1: Run the Notebook for this Lab

1. In Azure portal, open the available machine learning workspace.

2. Select **Launch now** under the **Try the new Azure Machine Learning studio** message.

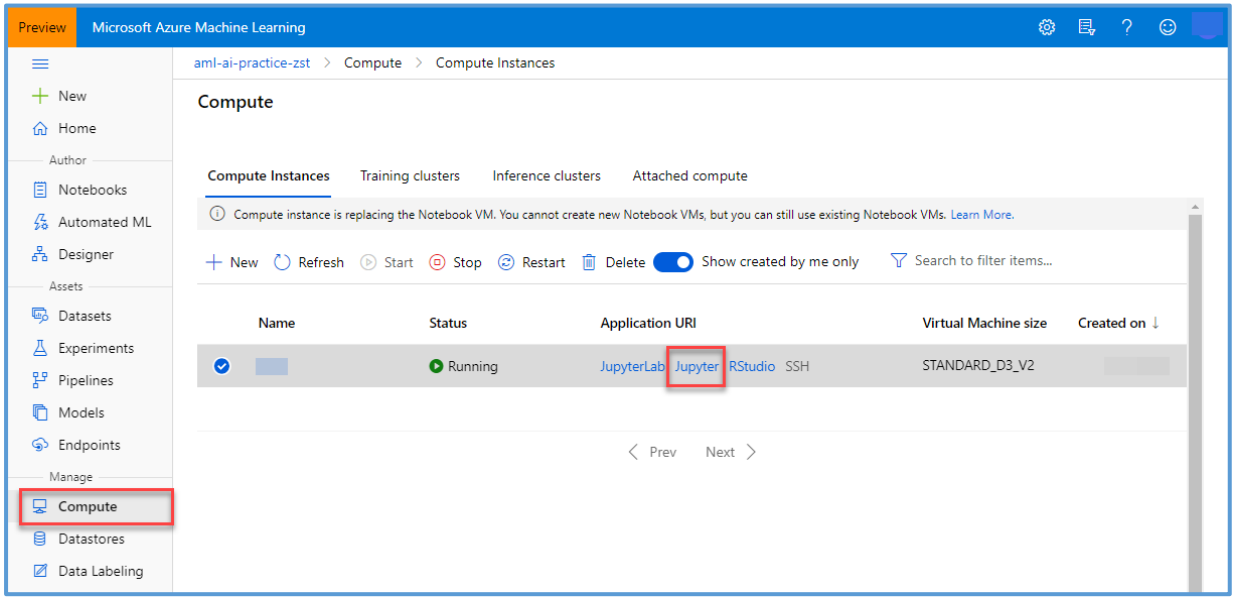


3. When you first launch the studio, you may need to set the directory and subscription. If so, you will see this screen:

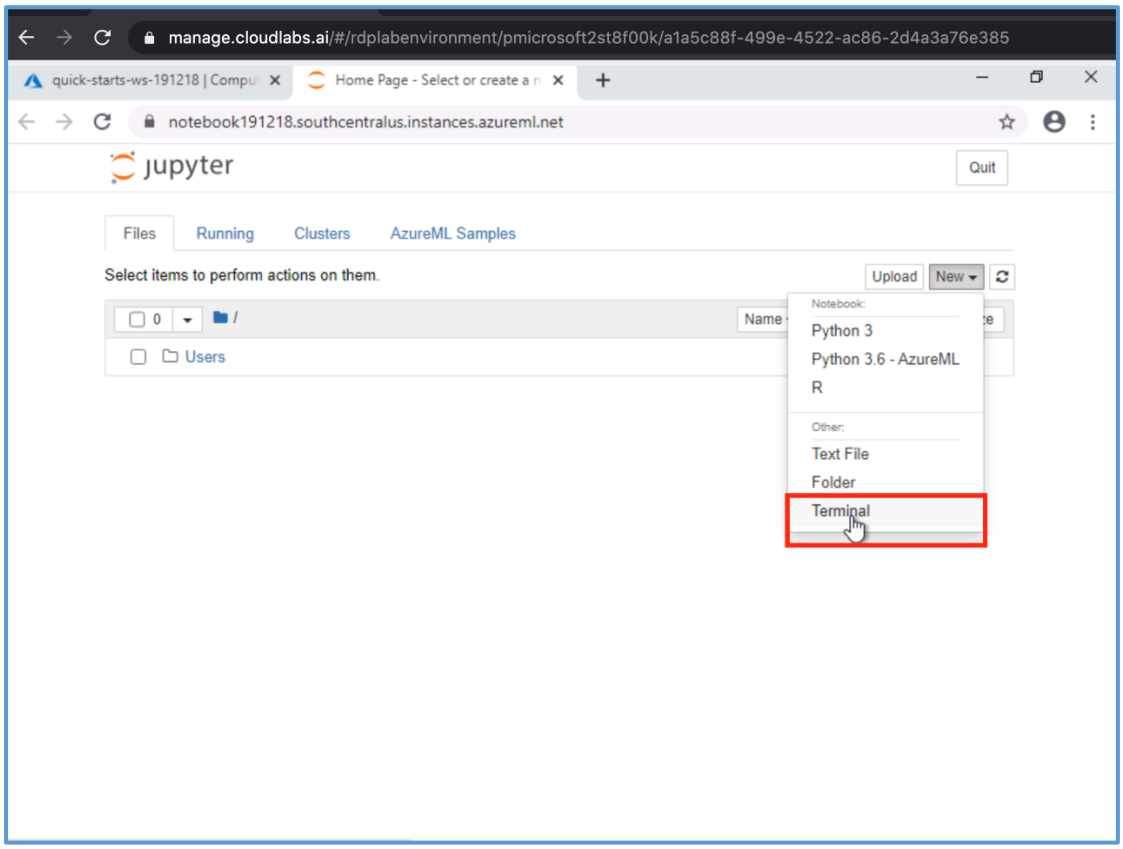


For the directory, select **Udacity** and for the subscription, select **Azure Sponsorship**. For the machine learning workspace, you may see multiple options listed. **Select any of these** (it does not matter which) and then click **Get started**.

4. From the studio, navigate to **Compute**. Next, for the available Compute Instance, under Application URI select `Jupyter`. Be sure to select `Jupyter` and not `JupterLab`.

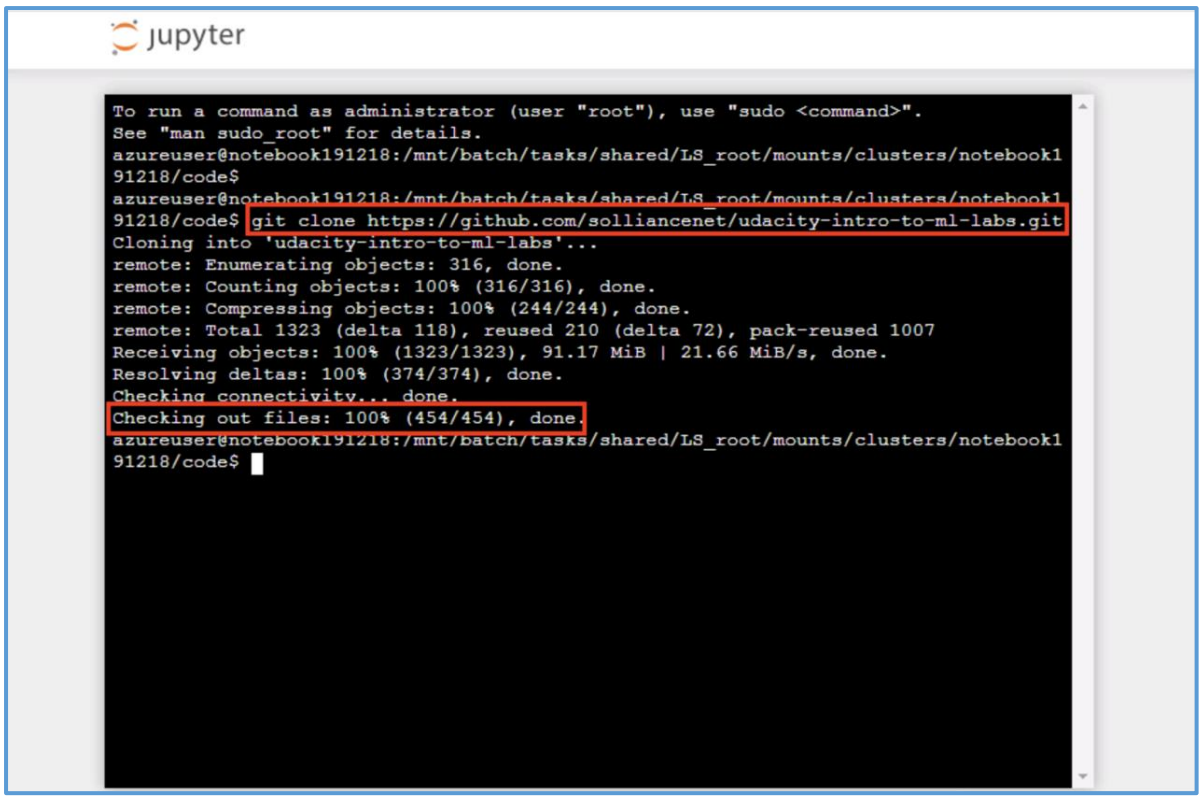


5. From within the Jupyter interface, select **New, Terminal**.

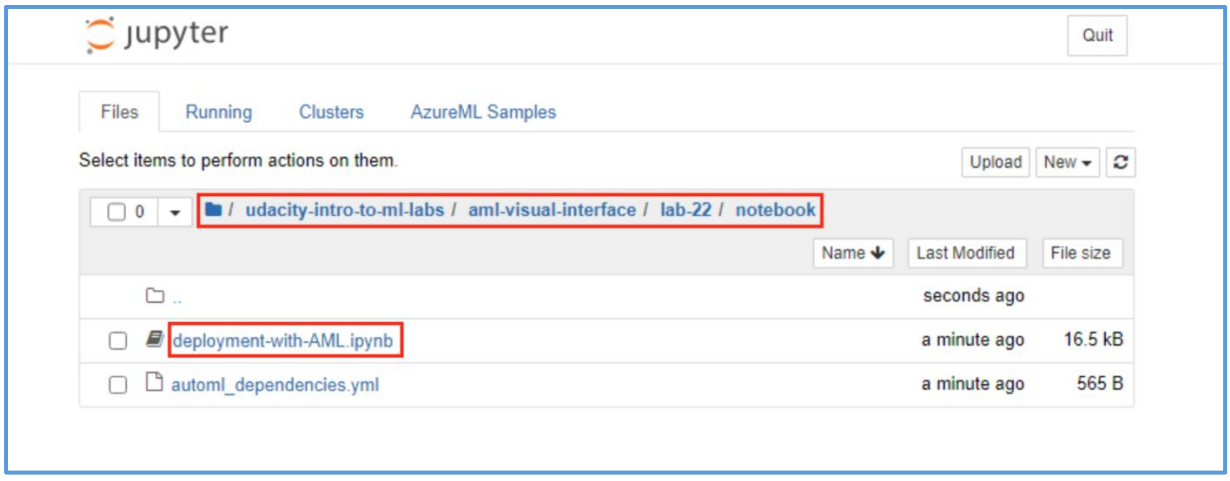


6. In the new terminal window run the following command and wait for it to finish:

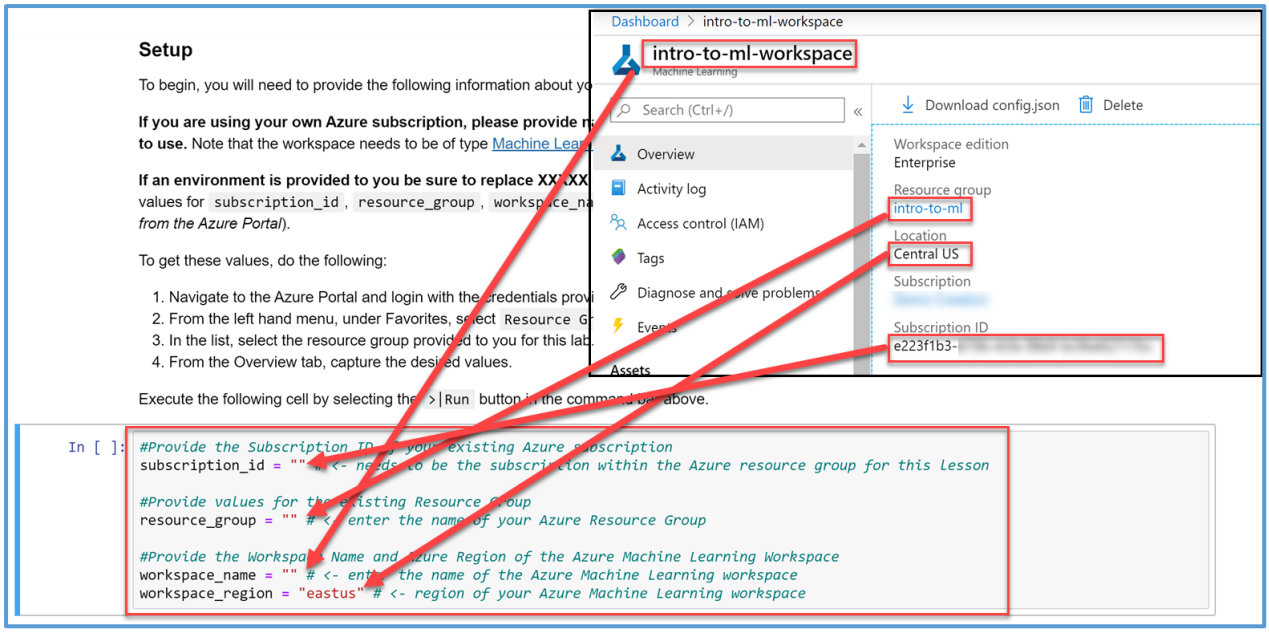
```
git clone https://github.com/solliancenet/udacity-intro-to-ml-labs.git
```



7. From within the Jupyter interface, navigate to directory `udacity-intro-to-ml-labs/aml-visual-interface/lab-22/notebook` and open `deployment-with-AML.ipynb`. This is the Python notebook; you will step through executing in this lab.



8. In the Setup portion of the notebook, you will be, asked to provide values for `subscription_id`, `resource_group`, `workspace_name`, and `workspace_region`. To find these, open your Azure Machine Learning workspace in the Azure portal and copy the values as shown:



9. Follow the instructions within the notebook to complete the lab.

Next Steps

Congratulations! You have just learned how to use the Jupyter application on a compute instance to deploy a trained model to Azure Container Instances (ACI) for real-time inferencing. You can now return to the Udacity portal to continue with the lesson.

Deployment of Automated Machine Learning Model

Lab Overview

In this lab, you will 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 Instance 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.

Because you will be using the Azure Machine Learning SDK, you will be able to provision all your required Azure resources directly from this notebook, without having to use the Azure Portal to create any resources.

Setup

To begin, you will need to provide the following information about your Azure Subscription.

If you are using your own Azure subscription, please provide names for subscription_id, resource_group, workspace_name and workspace_region to use. Note that the workspace needs to be of type Machine Learning Workspace.

If an environment is provided to you be sure to replace XXXXX in the values below with your unique identifier. `` In the following cell, be sure to set the values for `subscription_id`, `resource_group`, `workspace_name` and `workspace_region` as directed by the comments (*these values can be acquired from the Azure Portal*).

To get these values, do the following:

1. Navigate to the Azure Portal and login with the credentials provided.
2. From the left hand menu, under Favorites, select `Resource Groups`.
3. In the list, select the resource group provided to you for this lab.
4. From the Overview tab, capture the desired values.

Execute the following cell by selecting the `>|Run` button in the command bar above.

In [*]:
```
#Provide the Subscription ID of your existing Azure subscription
subscription_id = "b4a122b5-b4d5-40e7-9878-57b87adf4a8b" # <- needs to be the subscription within the Azure resource group for th

#Provide values for the existing Resource Group
resource_group = "aml-quickstarts-59497" # <- enter the name of your Azure Resource Group

#Provide the Workspace Name and Azure Region of the Azure Machine Learning Workspace
workspace_name = "quick-starts-ws-59497" # <- enter the name of the Azure Machine Learning workspace
workspace_region = "westeurope" # <- region of your Azure Machine Learning workspace
```

Download the model that was trained using Automated Machine Learning

In [*]:
```
import urllib.request
import os

model_folder = './automl-model'
model_file_name = 'model.pkl'
model_path = os.path.join(model_folder, model_file_name)

# this is the URL to download a model that was trained using Automated Machine Learning
model_url = ('https://quickstartsws9073123377.blob.core.windows.net/'
             'azureml-blobstore-0d1c4218-a5f9-418b-bf55-902b65277b85/'
             'quickstarts/automl-model/v2/model.pkl')

# Download the model to your local disk in the model_folder
os.makedirs(model_folder, exist_ok=True)
urllib.request.urlretrieve(model_url, model_path)
```

Import required packages

The Azure Machine Learning SDK provides a comprehensive set of a capabilities that you can use directly within a notebook 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**.

In Azure Notebooks, all of the libraries needed for Azure Machine Learning are pre-installed. To use them, you just need to import them. Run the following cell to do so:

In [*]:
```
import azureml.core
from azureml.core import Workspace
from azureml.core.webservice import Webservice, AksWebservice
from azureml.core.image import Image
from azureml.core.model import Model

print("Azure ML SDK version:", azureml.core.VERSION)
```

Create and connect to an Azure Machine Learning Workspace

Run the following cell to create a new Azure Machine Learning **Workspace**.

Important Note: You will be prompted to login in the text that is output below the cell. Be sure to navigate to the URL displayed and enter the code that is provided. Once you have entered the code, return to this notebook and wait for the output to read `Workspace configuration succeeded`.

In [*]:
```
ws = Workspace.create(
    name = workspace_name,
    subscription_id = subscription_id,
    resource_group = resource_group,
    location = workspace_region,
    exist_ok = True)

ws.write_config()

print('Workspace configuration succeeded')
```

In [*]:
```
# 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
```

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 [*]:
```
# 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 model is registered as
                       tags = {'area': "auto", 'type': "regression"},
                       description = "NYC Taxi Fare Predictor",
                       workspace = ws)

print()
print("Model registered: {} \nModel Description: {} \nModel Version: {}".format(model.name,
                                                                                model.description, model.version))
```

Deploy the Model as a Web Service

Create the Scoring Script

Azure Machine Learning SDK gives you control over the logic of the web service, so that you can define how it retrieves the model and how the model is used for scoring. This is an important bit of flexibility. For example, you often have to prepare any input data before sending it to your model for scoring. You can define this data preparation logic (as well as the model loading approach) in the scoring file.

Run the following cell to create a scoring file that will be included in the Docker Image that contains your deployed web service.

Important Please update the `model_name` variable in the script below. The model name should be the same as the `Model registered` printed above.

Deploy the Model as a Web Service

Create the Scoring Script

Azure Machine Learning SDK gives you control over the logic of the web service, so that you can define how it retrieves the model and how the model is used for scoring. This is an important bit of flexibility. For example, you often have to prepare any input data before sending it to your model for scoring. You can define this data preparation logic (as well as the model loading approach) in the scoring file.

Run the following cell to create a scoring file that will be included in the Docker Image that contains your deployed web service.

Important Please update the `model_name` variable in the script below. The model name should be the same as the `Model registered` printed above.

In [*]:
```
%%writefile scoring_service.py

import json
import numpy as np
import pandas as pd
import azureml.train.automl

columns = ['vendorID', 'passengerCount', 'tripDistance', 'hour_of_day', 'day_of_week', 'day_of_month',
           'month_num', 'normalizeHolidayName', 'isPaidTimeOff', 'snowDepth', 'precipTime',
           'precipDepth', 'temperature']

def init():
    try:
        # One-time initialization of predictive model and scaler
        from azureml.core.model import Model
        from sklearn.externals import joblib
        global model

        model_name = 'nyc-taxi-automl-predictor'
        print('Looking for model path for model: ', model_name)
        model_path = Model.get_model_path(model_name=model_name)
        print('Looking for model in: ', model_path)
        model = joblib.load(model_path)
        print('Model loaded...')

    except Exception as e:
        print('Exception during init: ', str(e))

def run(input_json):
    try:
        inputs = json.loads(input_json)
        data_df = pd.DataFrame(np.array(inputs).reshape(-1, len(columns)), columns = columns)
        # Get the predictions...
        prediction = model.predict(data_df)
        prediction = json.dumps(prediction.tolist())
    except Exception as e:
        prediction = str(e)
    return prediction
```

Package Model

Run the next two cells to create the deployment **Image**

WARNING: to install, build-essential needs to be available on the Docker image and is not by default. Thus, we will create a custom dockerfile with build-essential installed.

In [*]:
```
%%writefile dockerfile
RUN apt-get update && apt-get install -y build-essential
```

In [*]:
```
conda_file = 'automl_dependencies.yml'
runtime = 'python'

# create container image configuration
print("Creating container image configuration...")
from azureml.core.image import ContainerImage
image_config = ContainerImage.image_configuration(execution_script = 'scoring_service.py',
                                                   runtime = runtime,
                                                   conda_file = conda_file,
                                                   docker_file = 'dockerfile')

# create the image
image_name = 'nyc-taxi-automl-image'

from azureml.core import Image
image = Image.create(name=image_name, models=[model], image_config=image_config, workspace=ws)

# wait for image creation to finish
image.wait_for_creation(show_output=True)
```

Deploy Model to Azure Container Instance (ACI) as a Web Service

In [ ]:
```
from azureml.core.webservice import AciWebservice, Webservice

aci_name = 'aci-cluster'

aci_config = AciWebservice.deploy_configuration(
    cpu_cores = 1,
    memory_gb = 1,
    tags = {'name': aci_name},
    description = 'NYC Taxi Fare Predictor Web Service')

service_name = 'nyc-taxi-srv'

aci_service = Webservice.deploy_from_image(deployment_config=aci_config,
                                           image=image,
                                           name=service_name,
                                           workspace=ws)

aci_service.wait_for_deployment(show_output=True)
```

Test the deployed web service

Make direct calls on the service object

In [ ]:
```
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)
```

Consume the Deployed Web Service

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

In [ ]:
```
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)
```

Chapter 23: Walkthrough: Training and Deploying from a Compute Instance

In this lab, you start with a model that was trained using automated machine learning.

You will learn how to use Azure ML, Python SDK to register, package and deploy the trained model to an Azure Container Instances, ACI, as a scoring web service. Finally, we will test the deploy model by directly calling the service object and by calling the service endpoint URI over HTTP.

First, in the Azure machine-learning studio, we navigate to compute.

We click the JupyterLab. We opened the Jupyter link. In the Jupyter environment, we opened the Users folder, then navigate to the folder it or assign username, and then open the notebook deployment within AML.

This Python notebook will step you through execute in this lab.

In the setup portion of this notebook, we need to add some important values.

The subscription ID, the resource group, the workspace name, and the workspace region.

In order to get these values, we need to head into the Azure machine learning workspace in Azure Portal.

Here we can see all the values, the resource group; I am going to copy that, workspace name, the region location, make sure you remove the spaces, the capitalizations, and then lastly, the subscription ID.

After you have all that done, we can hit **run all** to run through the notebook.

As you can see in this notebook, we do several things.

First, we put another important setup details.

Then we download the model that was trained using automated machine learning.

Then we create and connect to the Azure machine learning workspace.

After we have connected, we register our model here.

We can name the model and add several tags.

After that is, done, we deploy the model as a web service using these SDK commands.

After that's finished, we package the model, and after the model is packaged, it is sent to the Azure Container Instance and deployed as a web service.

After the deployment is finished and your notebook is finished running, we can head to the endpoints tab on the left menu. Now we can see that the New York taxi SRV has launched is an ACI compute type.

We can click and we can see more details about the endpoint, and even how to, consume it.

Congratulations, you have just learned how to use the Jupyter application on a compute instance to deploy a trained model to Azure Container Instance for real-time inferencing.

Chapter 24: Lesson Summary

In this lesson, you have learned about **managed services for Machine Learning** and how these services are, used to enhance Machine Learning processes.

First, you learned about various types of **computing resources** made available through managed services, including:

- Training compute
- Inferencing compute
- Notebook environments

Next, you studied the main concepts involved in the **modelling process**, including:

- Basic modelling
- How parts of the modelling process interact when used together
- More advanced aspects of the modelling 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 modelling work to production environments and make them operational

Finally, you were, introduced to the world of programming the managed services via the **Azure Machine Learning SDK for Python**.