Identify your data source and format

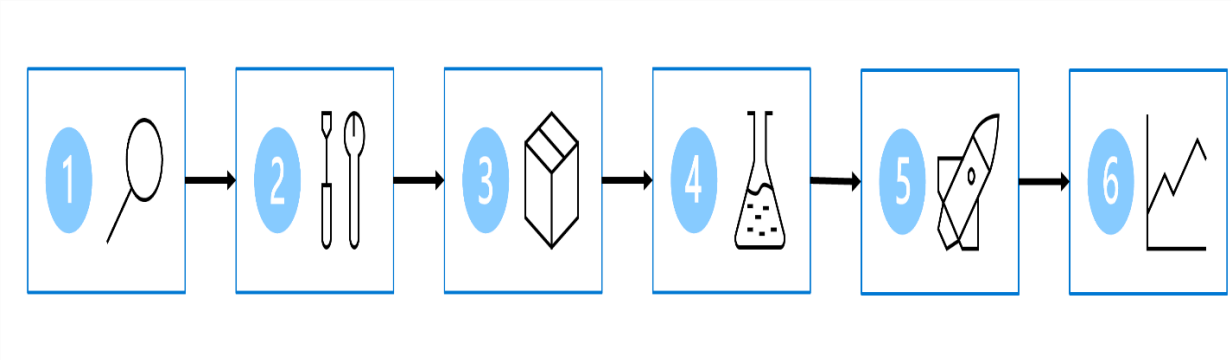
Completed100 XP

- 6 minutes

Data is the most important input for your machine learning models. You'll need access to data when training machine learning models, and the trained model needs data as input to generate predictions.

Imagine you're a data scientist and have been asked to train a machine learning model.

You aim to go through the following six steps to plan, train, deploy, and monitor the model:



1. **Define the problem**: Decide on what the model should predict and when it's successful.
2. **Get the data**: Find data sources and get access.
3. **Prepare the data**: Explore the data. Clean and transform the data based on the model's requirements.
4. **Train the model**: Choose an algorithm and hyperparameter values based on trial and error.
5. **Integrate the model**: Deploy the model to an endpoint to generate predictions.
6. **Monitor the model**: Track the model's performance.

Note

The diagram is a simplified representation of the machine learning process. Typically, the process is iterative and continuous. For example, when monitoring the model you may decide to go back and retrain the model.

To get and prepare the data you'll use to train the machine learning model, you'll need to extract data from a source and make it available to the Azure service you want to use to train models or make predictions.

In general, it's a best practice to extract data from its source before analyzing it. Whether you're using the data for data engineering, data analysis, or data science, you'll want to **extract** the data from its source, **transform** it, and **load** it into a serving layer. Such a process is also referred to as **Extract, Transform, and Load** (**ETL**) or **Extract, Load, and Transform** (**ELT**). The serving layer makes your data available for the service you'll use for further data processing like training machine learning models.

Before being able to design the ETL or ELT process, you'll need to **identify your data source and data format**.

Identify the data source

When you start with a new machine learning project, first identify *where the data you want to use is stored*.

The necessary data for your machine learning model may already be stored in a database or be generated by an application. For example, the data may be stored in a Customer Relationship Management (CRM) system, in a transactional database like an SQL database, or be generated by an Internet of Things (IoT) device.

In other words, your organization may already have business processes in place, which generate and store the data. If you don't have access to the data you need, there are alternative methods. You can collect new data by implementing a new process, acquire new data by using publicly available datasets, or buy curated datasets.

Identify the data format

Based on the source of your data, your data may be stored in a specific format. You need to understand in which format the data currently has and which format you'll need for your machine learning workloads.

Commonly, we refer to three different formats:

- **Tabular** or **structured** data: All data has the same fields or properties, which are defined in a schema. Tabular data is often represented in one or more tables where columns represent features and rows represent data points. For example, an Excel or CSV file can be interpreted as tabular data:

  Expand table

Patient ID	Pregnancies	Diastolic Blood Pressure	BMI	Diabetes Pedigree
1354778	0	80	43.50973	1.213191
1147438	8	93	21.24058	0.158365

- **Semi-structured** data: Not all data has the same fields or properties. Instead, each data point is represented by a collection of *key-value pairs*. The keys represent the features, and the values represent the properties for the individual data point. For example, real-time applications like Internet of Things (IoT) devices generate a JSON object:

  JSONCopy
  ```
  { "deviceId": 29482, "location": "Office1", "time":"2021-07-14T12:47:39Z",
  "temperature": 23 }
  ```

- **Unstructured** data: Files that don't adhere to any rules when it comes to structure. For example, documents, images, audio, and video files are considered unstructured data. Storing them as unstructured files ensures you don’t have to define any schema or structure, but also means you can't query the data in the database. You'll need to specify how to read such a file when consuming the data.

Tip

Learn more about **core data concepts on Learn**

Identify the desired data format

When extracting the data from a source, you may want to transform the data to change the data format and make it more suitable for model training.

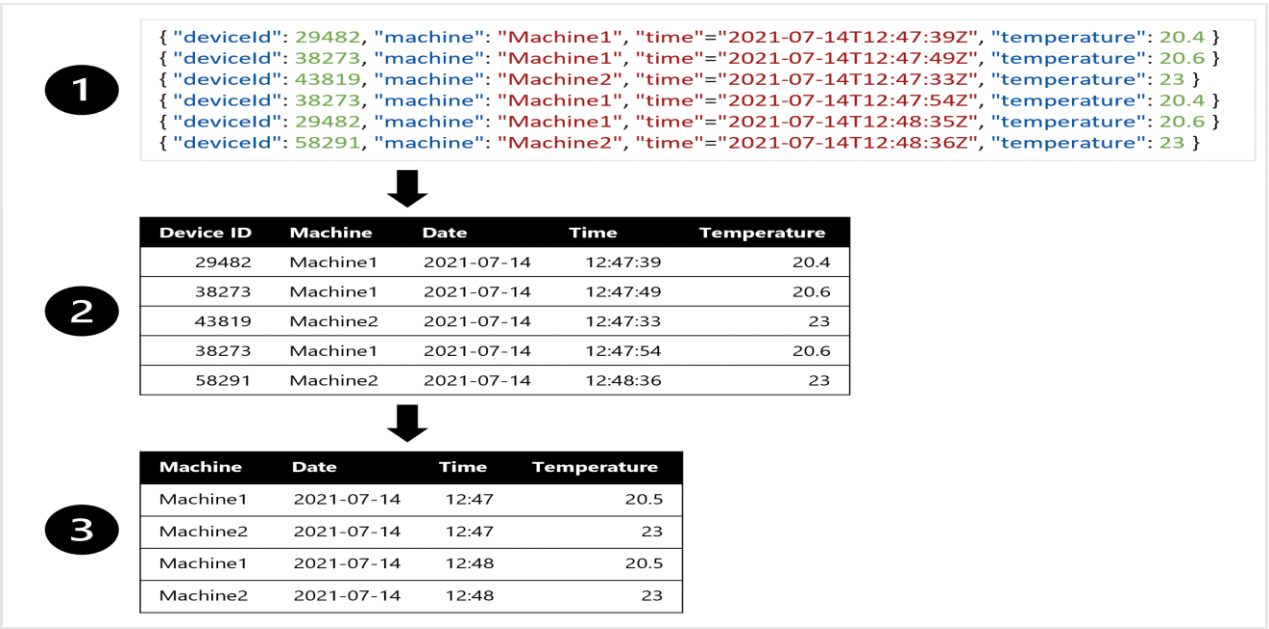
For example, you may want to train a forecasting model to perform predictive maintenance on a machine. You want to use features such as the machine's temperature to predict a problem with the machine. If you get an alert that a problem is arising, before the machine breaks down, you can save costs by fixing the problem early on.

Imagine the machine has a sensor that measures the temperature every minute. Each minute, every measurement or entry can be stored as a JSON object or file.

To train the forecasting model, you may prefer one table in which all temperature measurements of each minute are combined. You may want to create aggregates of the data and have a table of the average temperature per hour. To create the table, you'll want to transform the semi-structured data ingested from the IoT device to tabular data.

To create a dataset you can use to train the forecasting model, you may:

1. Extract data measurements as JSON objects from the IoT devices.
2. Convert the JSON objects to a table.
3. Transform the data to get the temperature per machine per minute.



Once you've identified the data source, the original data format, and the desired data format, you can think about how you want to serve the data. Then, you can design a data ingestion pipeline to automatically extract and transform the data you need.

Choose how to serve data to machine learning workflows

Completed100 XP

- 5 minutes

To access data when training machine learning models, you'll want to serve the data by storing it in a cloud data service. By storing data separately from your compute, you'll *minimize costs* and *be more flexible*.

Separate compute from storage

One of the benefits of the cloud is the ability to scale compute up or down according to your demands. In addition, you can shut down compute when you don't need it and restart it when you want to use it again.

Especially when training machine learning models, you'll have periods of time during which you'll need a lot of compute power, and times when you don't. When shutting down the compute you use for training machine learning models, you want to ensure your data isn't lost, and can still be accessed for other purposes (like reporting).

Therefore, it's a best practice to store your data in one tool, which is separate from another tool you use to train your models. Which tool or service is best to store your data depends on the data you have and the service you use for model training.

Store data for model training workloads

When you use **Azure Machine Learning**, **Azure Databricks**, or **Azure Synapse Analytics** for *model training*, there are three common options for storing data, which are easily connected to all three services:

- **Azure Blob Storage**: Cheapest option for storing data as *unstructured* data. Ideal for storing files like images, text, and JSON. Often also used to store data as CSV files, as data scientists prefer working with CSV files.
- **Azure Data Lake Storage (Gen 2)**: A more advanced version of the Azure Blob Storage. Also stores files like CSV files and images as *unstructured* data. A data lake also implements a hierarchical namespace, which means it's easier to give someone access to a specific file or folder. Storage capacity is virtually limitless so ideal for storing large data.
- **Azure SQL Database**: Stores data as *structured* data. Data is read as a table and schema is defined when a table in the database is created. Ideal for data that doesn't change over time.

Note

There are other Azure services for storing and serving data to services such as Azure Machine Learning, Azure Databricks, and Azure Synapse Analytics. The three storage options listed here are the most commonly used data storage solutions in combination with machine learning, especially for new projects. To learn about when to use which option, **explore this guide on Azure data stores**.

By storing your data in one of these Azure storage solutions, you can easily serve the data to whichever Azure service you use for machine learning workloads. To load the data into one of these storage solutions, you can set up a pipeline to extract, transform, and load the data.

Design a data ingestion solution

Completed100 XP

- 7 minutes

To move and transform data, you can use a **data ingestion pipeline**. A data ingestion pipeline is a sequence of tasks that move and transform the data. By creating a pipeline, you can choose to trigger the tasks manually, or schedule the pipeline when you want the tasks to be automated.

Create a data ingestion pipeline

To create a data ingestion pipeline, you can choose which Azure service to use.

Azure Synapse Analytics

A commonly used approach to create and run pipelines for data ingestion is using the data integration feature of **Azure Synapse Analytics**, also known as **Azure Synapse Pipelines**. With Azure Synapse Pipelines you can create and schedule data ingestion pipelines through the easy-to-use UI, or by defining the pipeline in JSON format.

When you create an Azure Synapse pipeline, you can easily copy data from one source to a data store by using one of the many standard connectors.

Tip

Learn more about the **copy activity in Azure Synapse Analytics, and all supported data stores and formats**.

To add a data transformation task to your pipeline, you can use a UI tool like **mapping data flow** or use a language like SQL, Python, or R.

Azure Synapse Analytics allows you to choose between different types of compute that can handle large data transformations at scale: serverless SQL pools, dedicated SQL pools, or Spark pools.

Tip

Learn more about how to **perform data integration at scale with Azure Synapse Analytics**.

Azure Databricks

Whenever you prefer a code-first tool and to use SQL, Python, or R to create your pipelines, you can also use **Azure Databricks**. Azure Databricks allows you to define your pipelines in a notebook, which you can schedule to run.

Azure Databricks uses Spark clusters, which distribute the compute to transform large amounts of data in less time than when you don't use distributed compute.

Tip

Learn more about **data engineering with Azure Databricks** and how to **prepare data for machine learning with Azure Databricks**

Azure Machine Learning

Azure Machine Learning provides compute clusters, which automatically scale up and down when needed. You can create a pipeline with the Designer, or by creating a collection of scripts. Though Azure Machine Learning pipelines are commonly used to train machine learning models, you could also use it to extract, transform, and store the data in preparation for training a machine learning model.

Whenever you want to perform all tasks within the same tool, creating and scheduling an Azure Machine Learning pipeline to run with the on-demand compute cluster may best suit your needs.

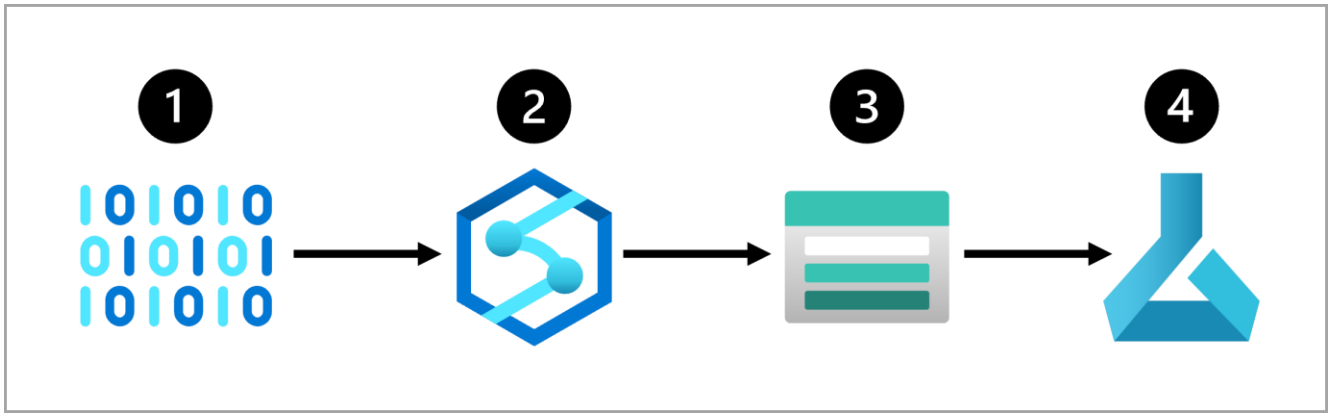
However, Azure Synapse Analytics and Azure Databricks offer more scalable compute that allow for transformations to be distributed across compute nodes. Therefore, your data transformations may perform better when you execute them with either Azure Synapse Analytics or Azure Databricks instead of using Azure Machine Learning.

Design a data ingestion solution

A benefit of using cloud technologies is the flexibility to create and use the services that best suit your needs. To create a solution, you can link services to each other and represent the solution in an **architecture**.

For example, a common approach for a data ingestion solution is to:

1. Extract raw data from its source (like a CRM system or IoT device).
2. Copy and transform the data with Azure Synapse Analytics.
3. Store the prepared data in an Azure Blob Storage.
4. Train the model with Azure Machine Learning.



It's a best practice to think about the architecture of a data ingestion solution before training your model. Thinking about how the data is automatically extracted and prepared for model training will help you to prepare for when your model is ready to go to production.

2.

When a data scientist extracts JSON objects from an IoT device, and combines all transformed data in a CSV file, which data store would be best to use?

Azure Blob Storage

Incorrect. Though you can store a CSV in an Azure Blob Storage, an IoT device generates many data points and you may reach the storage limits quickly.

Azure Data Lake Storage

Correct. You can store CSV files in a data lake without having any capacity constraints.

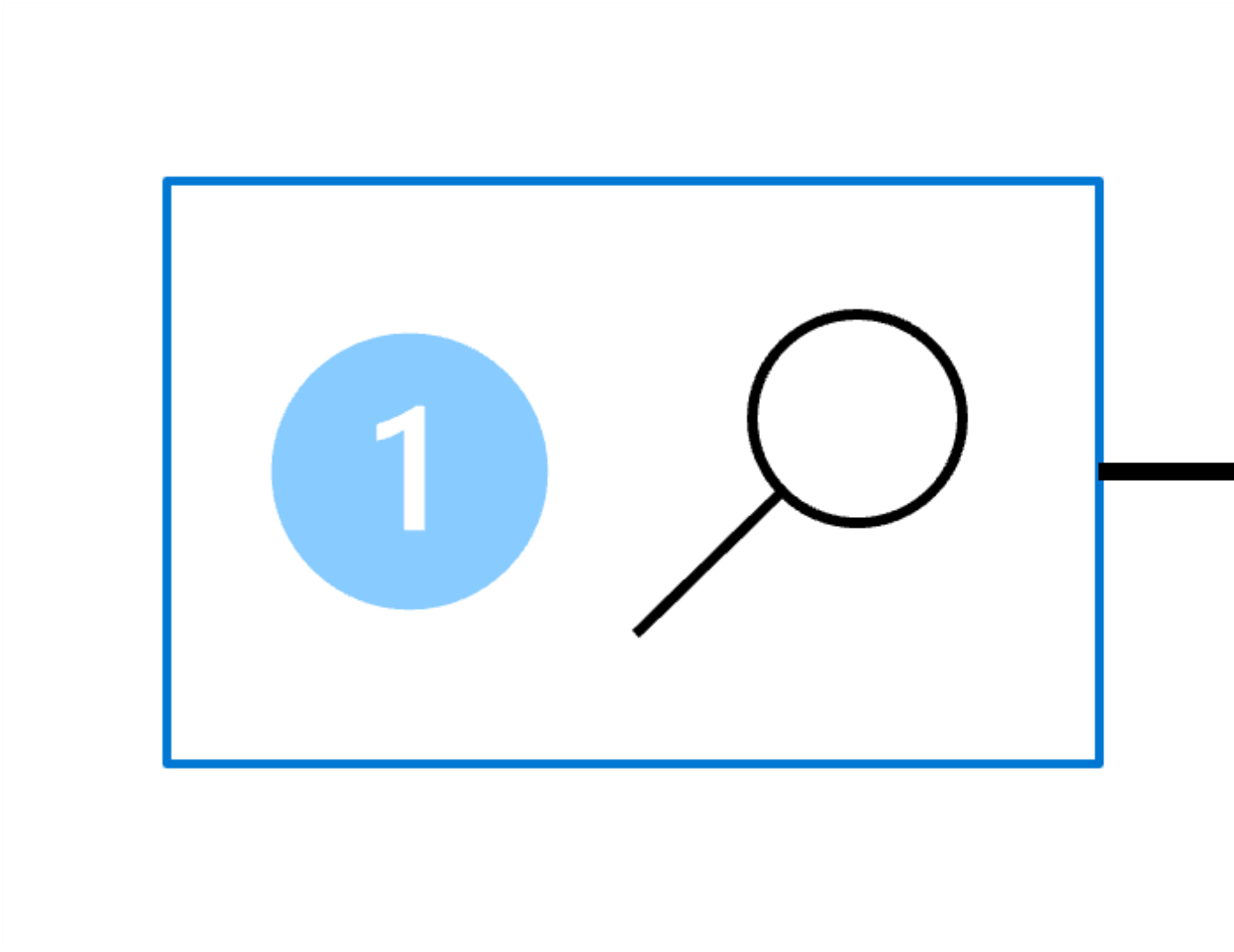
Identify machine learning tasks

Completed 100 XP

- 3 minutes

Imagine you're a data scientist and have been asked to train a machine learning model.

You aim to go through the following six steps to plan, train, deploy, and monitor the model:



1. **Define the problem**: Decide on what the model should predict and when it's successful.
2. **Get the data**: Find data sources and get access.
3. **Prepare the data**: Explore the data. Clean and transform the data based on the model's requirements.
4. **Train the model**: Choose an algorithm and hyperparameter values based on trial and error.
5. **Integrate the model**: Deploy the model to an endpoint to generate predictions.
6. **Monitor the model**: Track the model's performance.

Note

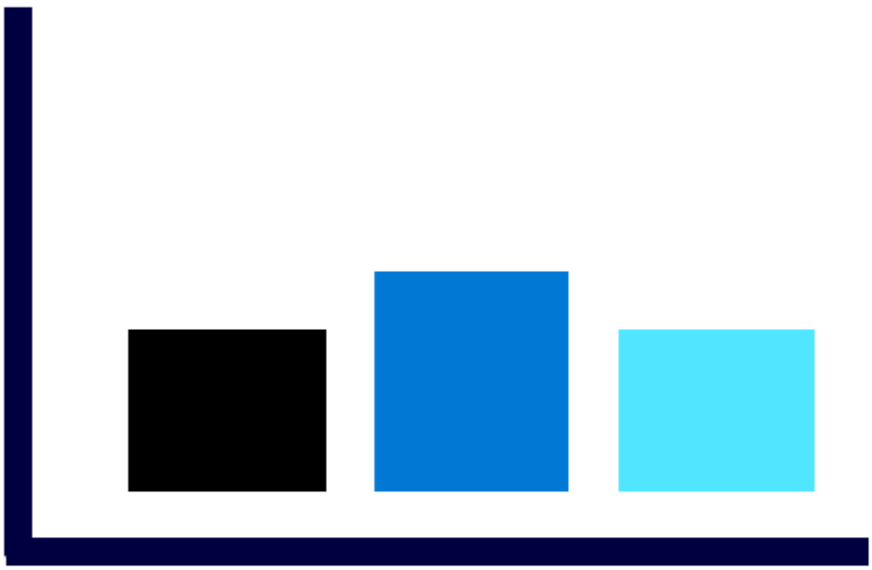
The diagram is a simplified representation of the machine learning process. Typically, the process is iterative and continuous. For example, when monitoring the model you may decide to go back and retrain the model.

Starting with the first step, you want to **define the problem** the model will solve by understanding:

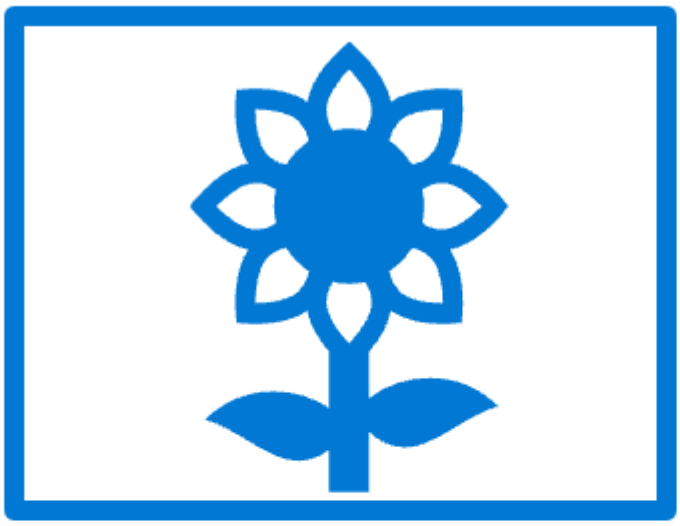
- What the model's output should be.
- What type of machine learning task you'll use.
- What criteria makes a model successful.

Depending on the data you have and the expected output of the model, you can identify the machine learning task. The task will determine which types of algorithms you can use to **train the model**.

Some common machine learning tasks are:



1



1. **Classification**: Predict a categorical value.
2. **Regression**: Predict a numerical value.
3. **Time-series forecasting**: Predict future numerical values based on time-series data.
4. **Computer vision**: Classify images or detect objects in images.
5. **Natural language processing** (**NLP**): Extract insights from text.

To train a model, you have a set of algorithms that you can use, depending on the task you want to perform. To evaluate the model, you can calculate performance metrics such as accuracy or precision. The metrics available will also depend on the task your model needs to perform and will help you to decide whether a model is successful in its task.

When you know what the problem is you're trying to solve and how you'll assess the success of your model, you can choose the service to train and manage your model.

Choose a service to train a machine learning model

Completed100 XP

- 7 minutes

There are many services available to train machine learning models. Which service you use depends on factors like:

- What type of model you need to train.
- Whether you need full control over model training.
- How much time you want to invest in model training.
- Which services are already within your organization.
- Which programming language you're comfortable with.

Within Azure, there are several services available for training machine learning models. When you choose to work with Azure instead of training a model on a local device, you'll have access to scalable and cost-effective compute. For example, you'll be able to use compute only for the time needed to train a model, and not pay for the compute when it's not used.

Some commonly used services in Azure to train machine learning models are:

Expand table

Icon	Description
	**Azure Machine Learning** gives you many different options to train and manage your machin can choose to work with the Studio for a UI-based experience, or manage your machine learni Python SDK, or CLI for a code-first experience. Learn more about Azure Machine Learning.
	**Azure Databricks** is a data analytics platform that you can use for data engineering and data s Databricks uses distributed Spark compute to efficiently process your data. You can choose to models with Azure Databricks or by integrating Azure Databricks with other services such as Learning. Learn more about Azure Databricks.
	**Azure Synapse Analytics** is an analytics service, which uses distributed compute for big data Synapse Analytics is primarily designed to ingest and transform data at scale but also includes learning capabilities. To train models with Azure Synapse Analytics, you can train models on or use the integrated Automated Machine Learning feature from Azure Machine Learning. Lea Synapse Analytics, and specifically about the machine learning capabilities in Azure Synapse
	**Azure AI Services** is a collection of prebuilt machine learning models you can use for commo such as object detection in images. The models are offered as an application programming inte easily integrate a model with your application. Some models can be customized with your own time and resources to train a new model from scratch. Learn more about Azure AI Services.

Understand the difference between services

Choosing a service to use for training your machine learning models may be challenging. Often, multiple services would fit your scenario. There are some general guidelines to help you:

- Use Azure AI Services whenever one of the customizable prebuilt models suits your requirements, to **save time and effort**.
- Use Azure Synapse Analytics or Azure Databricks if you want to **keep all data-related** (data engineering and data science) **projects within the same service**.
- Use Azure Synapse Analytics or Azure Databricks if you need **distributed compute** for working with large datasets (datasets are large when you experience capacity constraints with standard compute). You'll need to work with PySpark to use the distributed compute.
- Use Azure Machine Learning or Azure Databricks when you want **full control** over model training and management.
- Use Azure Machine Learning when **Python** is your preferred programming language.
- Use Azure Machine Learning when you want an **intuitive user interface** to manage your machine learning lifecycle.

Important

There are many factors which may influence your choice of service. Ultimately, it is up to you and your organization to decide what's the best fit. These are simply guidelines to help you understand how to differentiate between services.

Decide between compute options

Completed100 XP

- 7 minutes

When you want to train your own model, the most valuable resource you'll consume is compute. Especially during model training, it's important to choose the most suitable compute. Additionally, you should monitor compute utilization to know when to scale up or down to save on time and costs.

Though finding which virtual machine size best fits your needs is an iterative process, there are some guidelines you can follow when you start developing.

CPU or GPU

One important decision to make when configuring compute is whether you want to use a **central processing unit** (**CPU**) or a **graphics processing unit** (**GPU**). For smaller tabular datasets, CPU will be sufficient and cheaper to use. Whenever working with unstructured data like images or text, GPUs will be more powerful and effective.

For larger amounts of tabular data, it may also be beneficial to use GPUs. When processing your data and training your model takes a long time, even with the largest CPUs compute available, you may want to consider using GPUs compute instead. There are libraries such as RAPIDs (developed by NVIDIA) which allow you to efficiently perform data preparation and model training with larger tabular datasets. As GPUs come at a higher cost than CPUs, it may require some experimentation to explore whether using GPU will be beneficial for your situation.

Tip

Learn how to **train compute-intensive models with Azure Machine Learning**

General purpose or memory optimized

When you create compute resources for machine learning workloads, there are two common types of virtual machine sizes you can choose from:

- **General purpose**: Have a balanced CPU-to-memory ratio. Ideal for testing and development with smaller datasets.
- **Memory optimized**: Have a high memory-to-CPU ratio. Great for in-memory analytics, which is ideal when you have larger datasets or when you're working in notebooks.

The size of compute in Azure Machine Learning is shown as the **virtual machine size**. The sizes follow the same naming conventions as Azure Virtual Machines.

Tip

Learn more about **sizes for virtual machines in Azure**.

Spark

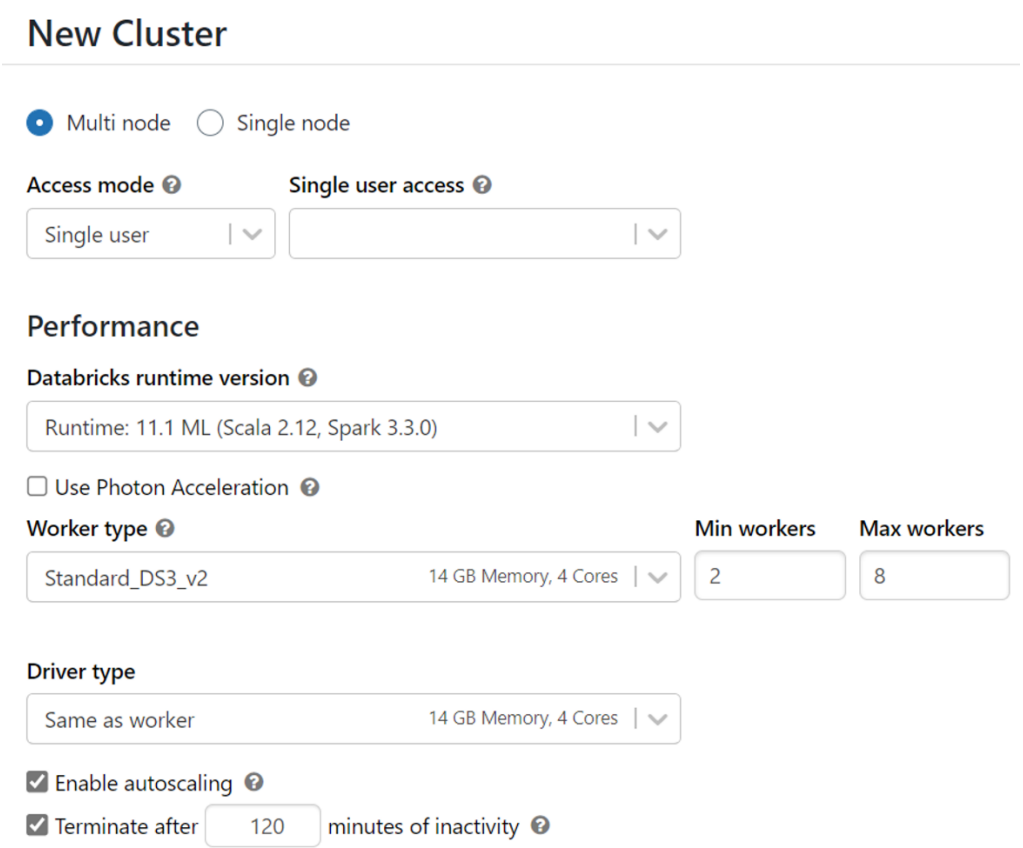
Services like Azure Synapse Analytics and Azure Databricks offer Spark compute. Spark compute or clusters use the same sizing as virtual machines in Azure but distribute the workloads.

A Spark cluster consists of a driver node and worker nodes. Your code will initially communicate with the driver node. The work is then distributed across the worker nodes. When you use a service that distributes the work, parts of the workload can be executed in parallel, reducing the processing time. Finally, the work is summarized and the driver node communicates the result back to you.

Important

To make optimal use of a Spark cluster, your code needs to be written in a Spark-friendly language like Scala, SQL, RSpark, or PySpark in order to distribute the workload. If you write in Python, you'll only use the driver node and leave the worker nodes unused.

When you create a Spark cluster, you'll have to choose whether you want to use CPU, or GPU compute. You'll also have to choose the virtual machine size for the driver and the worker nodes.



Monitor the compute utilization

Configuring your compute resources for training a machine learning model is an iterative process. When you know how much data you have and how you want to train your model, you'll have an idea of which compute options may best suit training your model.

Every time you train a model, you should monitor how long it takes to train the model and how much compute is used to execute your code. By monitoring the compute utilization, you'll know whether to scale your compute up or down. If training your model takes too long, even with the largest compute size, you may want to use GPUs instead of CPUs. Alternatively, you can choose to distribute model training by using Spark compute which may require you to rewrite your training scripts.

Decide on real-time or batch deployment

Completed100 XP

- 8 minutes

When you deploy a model to an endpoint to integrate with an application, you can choose to design it for real-time or batch predictions.

The type of predictions you need depends on how you want to use the model's predictions

To decide whether to design a real-time or batch deployment solution, you need to consider the following questions:

- How often should predictions be generated?
- How soon are the results needed?
- Should predictions be generated individually or in batches?
- How much compute power is needed to execute the model?

Identify the necessary frequency of scoring

A common scenario is that you're using a model to score new data. Before you can get predictions in real-time or in batch, you must first collect the new data.

There are various ways to generate or collect data. New data can also be collected at different time intervals.

For example, you can collect temperature data from an Internet of Things (IoT) device every minute. You can get transactional data every time a customer buys a product from your web shop. Or you can extract financial data from a database every three months.

Generally, there are two types of use cases:

1. You need the model to score the new data as soon as it comes in.
2. You can schedule or trigger the model to score the new data that you've collected over time.


1

Whether you want real-time or batch predictions *doesn't necessarily depend on how often new data is collected*. Instead, it depends on how often and how quickly you need the predictions to be generated.

If you need the model's predictions immediately when new data is collected, you need real-time predictions. If the model's predictions are only consumed at certain times, you need batch predictions.

Decide on the number of predictions

Another important question to ask yourself is whether you need the predictions to be generated individually or in batches.

A simple way to illustrate the difference between individual and batch predictions is to imagine a table. Suppose you have a table of customer data where each row represents a customer. For each customer, you have some demographic data and behavioral data, such as how many products they've purchased from your web shop and when their last purchase was.

Based on this data, you can predict customer churn: whether a customer will buy from your web shop again or not.

Once you've trained the model, you can decide if you want to generate predictions:

- **Individually**: The model receives a *single row of data* and returns whether or not that individual customer will buy again.
- **Batch**: The model receives *multiple rows of data* in one table and returns whether or not each customer will buy again. The results are collated in a table that contains all predictions.

You can also generate individual or batch predictions when working with files. For example, when working with a computer vision model you may need to score a single image individually, or a collection of images in one batch.

Consider the cost of compute

In addition to using compute when training a model, you also need compute when deploying a model. Depending on whether you deploy the model to a real-time or batch endpoint, you'll use different types of compute. To decide whether to deploy your model to a real-time or batch endpoint, you must consider the cost of each type of compute.

If you need **real-time predictions**, you need compute that is always available and able to return the results (almost) immediately. **Container** technologies like *Azure Container Instance* (ACI) and *Azure Kubernetes Service* (AKS) are ideal for such scenarios as they provide a lightweight infrastructure for your deployed model.

However, when you deploy a model to a real-time endpoint and use such container technology, the compute is *always on*. Once a model is deployed, you're continuously paying for the compute as you can't pause, or stop the compute as the model must always be available for immediate predictions.

Alternatively, if you need **batch predictions**, you need compute that can handle a large workload. Ideally, you'd use a **compute cluster** that can score the data in *parallel* batches by using multiple nodes.

When working with compute clusters that can process data in parallel batches, the compute is provisioned by the workspace when the batch scoring is triggered, and scaled down to 0 nodes when there's no new data to process. By letting the workspace scale down an idle compute cluster, you can save significant costs.

Decide on real-time or batch deployment

Choosing a deployment strategy for your machine learning models may be challenging, as different factors may influence your decision.

In general, if you need individual predictions immediately when new data is collected, you need real-time predictions.

If you need the model to score new data when a batch of data is available, you should get batch predictions.

There are scenarios where you expect to need real-time predictions when batch predictions can be more cost-effective. Remember that you're continuously paying for compute with real-time deployments, even when no new predictions are generated.

If you can allow for a 5-10 minutes delay when needing immediate predictions, you can opt to deploy your model to a batch endpoint. The delay is caused in the time it needs to start the compute cluster after the endpoint is triggered. However, the compute cluster will also stop after the prediction is generated, minimizing costs and potentially being a more cost-effective solution.

Finally, you also have to consider the required compute for your model to score new data. Simpler models require less cost and time to generate predictions. More

complex models may require more compute power and processing time to generate predictions. Therefore, you should consider how you'll deploy your model before deciding on how to train your model.

Explore an MLOps architecture

Completed100 XP

- 10 minutes

As a data scientist, you want to train the best machine learning model. To implement the model, you want to deploy it to an endpoint and integrate it with an application.

Over time, you may want to retrain the model. For example, you can retrain the model when you have more training data.

In general, once you've trained a machine learning model, you want to get the model ready for enterprise-scale. To prepare the model and operationalize it, you want to:

- Convert the model training to a **robust** and **reproducible** pipeline.
- Test the code and the model in a **development** environment.
- Deploy the model in a **production** environment.
- **Automate** the end-to-end process.

Set up environments for development and production

Within MLOps, similarly to DevOps, an **environment** refers to a collection of resources. These resources are used to deploy an application, or with machine learning projects, to deploy a model.

Note

In this module, we refer to the DevOps interpretation of environments. Note that Azure Machine Learning also uses the term environments to describe a collection of Python packages needed to run a script. These two concepts of environments are independent from each other.

How many environments you work with, depends on your organization. Commonly, there are at least two environments: *development* or *dev* and *production* or *prod*. Plus, you can add environments in between like a staging or *pre-production* (*pre-prod*) environment.

A typical approach is to:

- Experiment with model training in the *development* environment.
- Move the best model to the *staging* or *pre-prod* environment to deploy and test the model.
- Finally release the model to the *production* environment to deploy the model so that end-users can consume it.

Organize Azure Machine Learning environments

When you implement MLOps, and work with machine learning models at a large scale, it's a best practice to work with separate environments for different stages.

Imagine your team uses a dev, pre-prod, and prod environment. Not everyone on your team should get access to all environments. Data scientists may only work within the dev environment with non-production data, while machine learning engineers work on deploying the model in the pre-prod and prod environment with production data.

Having separate environments makes it easier to control access to resources. Each environment can then be associated with a separate Azure Machine Learning workspace.

Within Azure, you use role-based access control (RBAC) to give colleagues the right level of access to the subset of resources they need to work with.

Alternatively, you can use only one Azure Machine Learning workspace. When you use one workspace for development and production, you have a smaller Azure footprint and less management overhead. However, RBAC applies to both dev and prod environments, which may mean that you're giving people too little or too much access to resources.

Tip

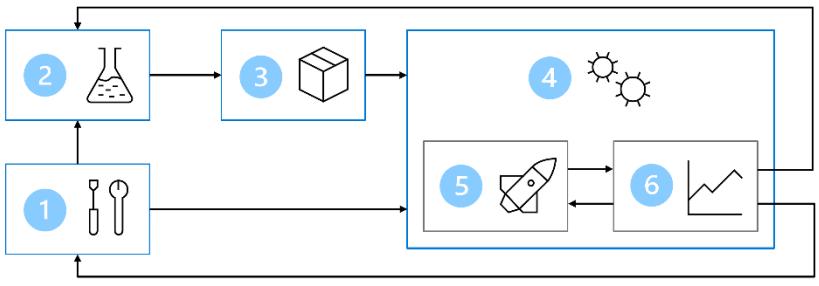
Learn more about **best practices to organize Azure Machine Learning resources**.

Design an MLOps architecture

Bringing a model to production means you need to scale your solution and work together with other teams. Together with other data scientists, data engineers and an infrastructure team, you may decide on using the following approach:

- Store all data in an Azure Blob storage, managed by the data engineer.
- The infrastructure team creates all necessary Azure resources, like the Azure Machine Learning workspace.
- Data scientists focus on what they do best: developing and training the model (inner loop).
- Machine learning engineers deploy the trained models (outer loop).

As a result, your MLOps architecture includes the following parts:



1. **Setup**: Create all necessary Azure resources for the solution.
2. **Model development (inner loop)**: Explore and process the data to train and evaluate the model.
3. **Continuous integration**: Package and register the model.
4. **Model deployment (outer loop)**: Deploy the model.
5. **Continuous deployment**: Test the model and promote to production environment.
6. **Monitoring**: Monitor model and endpoint performance.

When you're working with larger teams, you're not expected to be responsible of all parts of the MLOps architecture as a data scientist. To prepare your model for MLOps however, you should think about how to design for monitoring and retraining.

Design for monitoring

Completed100 XP

- 9 minutes

As part of a machine learning operations (MLOps) architecture, you should think about how to monitor your machine learning solution.

Monitoring is beneficial in any MLOps environment. You'll want to monitor the *model*, the *data*, and the *infrastructure* to collect metrics that help you decide on any necessary next steps.

Monitor the model

Most commonly, you want to monitor the performance of your model. During development, you use MLflow to train and track your machine learning models. Depending on the model you train, there are different metrics you can use to evaluate whether the model is performing as expected.

To monitor a model in production, you can use the trained model to generate predictions on a small subset of new incoming data. By generating the performance metrics on that test data, you're able to verify whether the model is still achieving its goal.

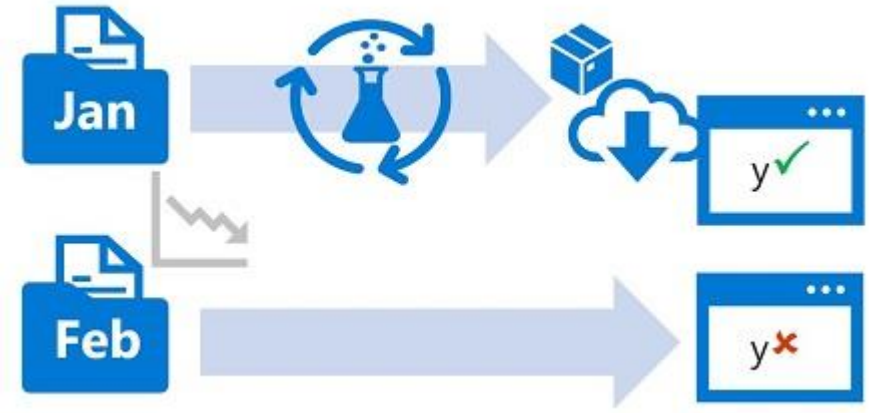
Additionally, you may also want to monitor for any responsible artificial intelligence (AI) issues. For example, whether the model is making fair predictions.

Before you can monitor a model, it's important to decide which performance metrics you want to monitor and what the benchmark for each metric should be. When should you be alerted that the model isn't accurate anymore?

Monitor the data

You typically train a machine learning model using a historical dataset that is representative of the new data that your model receives when deployed. However, over time there may be trends that change the profile of the data, making your model less accurate.

For example, suppose a model is trained to predict the expected gas mileage of an automobile based on the number of cylinders, engine size, weight, and other features. Over time, as car manufacturing and engine technologies advance, the typical fuel-efficiency of vehicles might improve dramatically; making the model's predictions trained on older data less accurate.



This change in data profiles between current and the training data is known as data drift, and it can be a significant issue for predictive models used in production. It's therefore important to be able to monitor data drift over time, and retrain models as required to maintain predictive accuracy.

Monitor the infrastructure

Next to monitoring the model and data, you should also monitor the infrastructure to minimize cost and optimize performance.

Throughout the machine learning lifecycle, you use compute to train and deploy models. With machine learning projects in the cloud, compute may be one of your biggest expenses. You therefore want to monitor whether you are efficiently using your compute.

For example, you can monitor the compute utilization of your compute during training and during deployment. By reviewing compute utilization, you know whether you can scale down your provisioned compute, or whether you need to scale out to avoid capacity constraints.

Design for retraining

Completed 100 XP

- 7 minutes

When preparing your model for production in a machine learning operations (MLOps) solution, you need to design for retraining.

Generally, there are two approaches to when you want to retrain a model:

- Based on a **schedule**: when you know you always need the latest version of the model, you can decide to retrain your model every week, or every month, based on a schedule.
- Based on **metrics**: if you only want to retrain your model when necessary, you can monitor the model's performance and data drift to decide when you need to retrain the model.

In either case, you need to design for retraining. To easily retrain your model, you should prepare your code for automation.

Prepare your code

Ideally, you should train models with **scripts** instead of notebooks. Scripts are better suited for automation. You can add **parameters** to a script and change input parameters like the training data or hyperparameter values. When you parameterize your scripts, you can easily retrain the model on new data if needed.

Another important thing to prepare your code is to host the code in a central repository. A repository refers to a location where all relevant files to a project are stored. With machine learning projects, Git-based repositories are ideal to achieve **source control**.

When you apply source control to your project, you can easily collaborate on a project. You can assign someone to improve the model by updating the code. You'll be able to see past changes, and you can review changes before they're committed to the main repository.

Automate your code

When you want to automatically execute your code, you can configure Azure Machine Learning jobs to run scripts. In Azure Machine Learning, you can create and schedule pipelines to run scripts too.

If you want scripts to run based on a trigger or event happening outside of Azure Machine Learning, you may want to trigger the Azure Machine Learning job from another tool.

Two tools that are commonly used in MLOps projects are Azure DevOps and GitHub (Actions). Both tools allow you to create automation pipelines and can trigger Azure Machine Learning pipelines.

As a data scientist, you may prefer to work with the Azure Machine Learning Python SDK. However, when working with tools like Azure DevOps and GitHub, you may prefer to configure the necessary resources and jobs with the Azure Machine Learning CLI extension instead. The Azure CLI is designed for automating tasks and may be easier to use with Azure DevOps and GitHub.

Tip

If you want to learn more about MLOps, explore the **introduction to machine learning operations (MLOps)** or try to build your first **MLOps automation pipeline with GitHub Actions**

When should we retrain the model?

Every week.

When the model's metrics are below the benchmark.

Correct. The most important thing is that the model performs as expected. When the model's performance is in jeopardy, we should retrain the model.

When there's data drift.

Incorrect. It's stated that new data is not seen as trustworthy. Currently, we shouldn't rely on monitoring the data to decide when to retrain the model.

Create an Azure Machine Learning workspace

Completed 100 XP

- 7 minutes

To get access to an Azure Machine Learning workspace, you first need to create the **Azure Machine Learning** service in your Azure subscription. The **workspace** is

central place where you can work with all resources and assets available to train and deploy machine learning models. For reproducibility, the workspace stores a history of all training jobs, including logs, metrics, outputs, and a snapshot of your code.

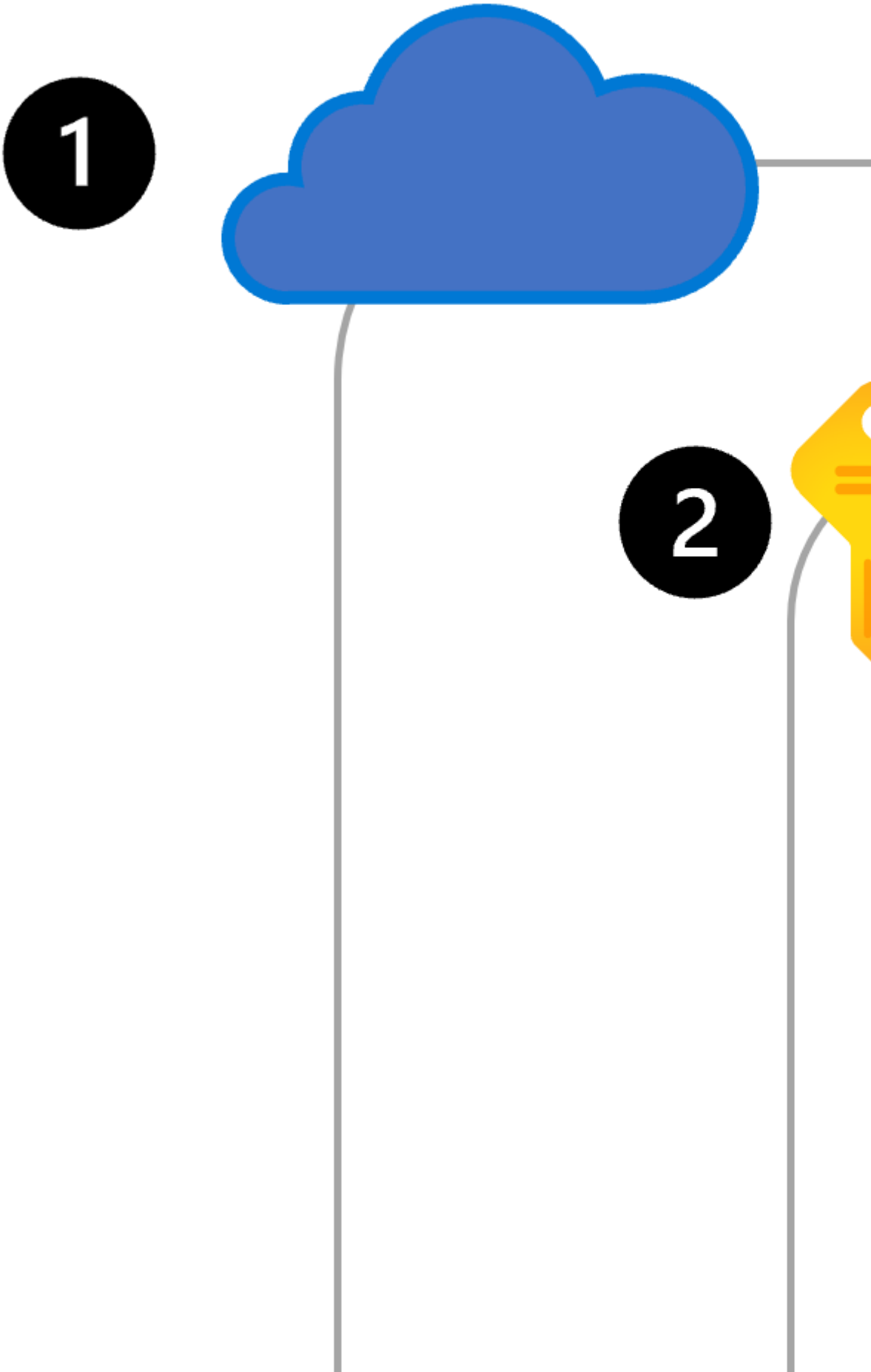
Understand the Azure Machine Learning service

To create an Azure Machine Learning service, you'll have to:

1. Get access to **Azure**, for example through the Azure portal.
2. Sign in to get access to an **Azure subscription**.
3. Create a **resource group** within your subscription.
4. Create an **Azure Machine Learning service** to create a workspace.

   When a workspace is provisioned, Azure will automatically create other Azure resources within the same resource group to support the workspace:

5. **Azure Storage Account**: To store files and notebooks used in the workspace, and to store metadata of jobs and models.
6. **Azure Key Vault**: To securely manage secrets such as authentication keys and credentials used by the workspace.
7. **Application Insights**: To monitor predictive services in the workspace.
8. **Azure Container Registry**: Created when needed to store images for Azure Machine Learning environments.


1
2

Create the workspace

You can create an Azure Machine Learning workspace in any of the following ways:

- Use the user interface in the **Azure portal** to create an Azure Machine Learning service.
- Create an **Azure Resource Manager** (**ARM**) template. Learn how to use an ARM template to create a workspace.
- Use the **Azure Command Line Interface** (**CLI**) with the Azure Machine Learning CLI extension. Learn how to create the workspace with the CLI v2.
- Use the **Azure Machine Learning Python SDK**.

For example, the following code uses the Python SDK to create a workspace named `mlw-example`:

PythonCopy

```
from azure.ai.ml.entities import Workspace

workspace_name = "mlw-example"

ws_basic = Workspace(
    name=workspace_name,
    location="eastus",
    display_name="Basic workspace-example",
    description="This example shows how to create a basic workspace",
)
ml_client.workspaces.begin_create(ws_basic)
```

Explore the workspace in the Azure portal

Creating an Azure Machine Learning workspace will typically take between 5-10 minutes to complete. When your workspace is created, you can select the workspace to view its details.

mlw-dp100-labs

Azure Machine Learning workspace

Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Events

Settings

Networking

Properties

Locks

Monitoring

Alerts

Metrics

Download

Essentials

Resource group
rg-dp100-labs

Location
East US

Subscription
Demo-Subscript

Subscription ID

Storage

From the **Overview** page of the Azure Machine Learning workspace in the Azure portal, you can launch the Azure Machine Learning studio. The Azure Machine Learning studio is a web portal and provides an easy-to-use interface to create, manage, and use resources and assets in the workspace.

From the Azure portal, you can also give others access to the Azure Machine Learning workspace, using the **Access control**.

Give access to the Azure Machine Learning workspace

You can give individual users or teams access to the Azure Machine Learning workspace. Access is granted in Azure using **role-based access control** (**RBAC**), which you can configure in the **Access control** tab of the resource or resource group.

In the access control tab, you can manage permissions to restrict what actions certain users or teams can perform. For example, you could create a policy that only allows users in the *Azure administrators group* to create compute targets and datastores. While users in the *data scientists group* can create and run jobs to train models, and register models.

There are three general built-in roles that you can use across resources and resource groups to assign permissions to other users:

- **Owner**: Gets full access to all resources, and can grant access to others using access control.
- **Contributor**: Gets full access to all resources, but can't grant access to others.
- **Reader**: Can only view the resource, but isn't allowed to make any changes.

Additionally, Azure Machine Learning has specific built-in roles you can use:

- **AzureML Data Scientist**: Can perform all actions within the workspace, except for creating or deleting compute resources, or editing the workspace settings.
- **AzureML Compute Operator**: Is allowed to create, change, and manage access the compute resources within a workspace.

Finally, if the built-in roles aren't meeting your needs, you can create a custom role to assign permissions to other users.

Tip

Learn more about **how to manage access to an Azure Machine Learning workspace, including creating custom roles**.

Organize your workspaces

Initially, you might only work with one workspace. However, when working on large-scale projects, you might choose to use multiple workspaces.

You can use workspaces to group machine learning assets based on projects, deployment environments (for example, test and production), teams, or some other organizing principle.

Identify Azure Machine Learning resources

Completed 100 XP

- 6 minutes

Resources in Azure Machine Learning refer to the infrastructure you need to run a machine learning workflow. Ideally, you want someone like an administrator to create and manage the resources.

The resources in Azure Machine Learning include:

- The workspace
- Compute resources
- Datastores

Create and manage the workspace

The **workspace** is the top-level resource for Azure Machine Learning. Data scientists need access to the workspace to train and track models, and to deploy the models to endpoints.

However, you want to be careful with who has *full* access to the workspace. Next to references to compute resources and datastores, you can find all logs, metrics, outputs, models, and snapshots of your code in the workspace.

Create and manage compute resources

One of the most important resources you need when training or deploying a model is **compute**. There are five types of compute in the Azure Machine Learning workspace:

- **Compute instances**: Similar to a virtual machine in the cloud, managed by the workspace. Ideal to use as a development environment to run (Jupyter) notebooks.
- **Compute clusters**: On-demand clusters of CPU or GPU compute nodes in the cloud, managed by the workspace. Ideal to use for production workloads as they automatically scale to your needs.
- **Kubernetes clusters**: Allows you to create or attach an Azure Kubernetes Service (AKS) cluster. Ideal to deploy trained machine learning models in production scenarios.
- **Attached computes**: Allows you to attach other Azure compute resources to the workspace, like Azure Databricks or Synapse Spark pools.
- **Serverless compute**: A fully managed, on-demand compute you can use for training jobs.

Note

As Azure Machine Learning creates and manages *serverless compute* for you, it's not listed on the compute page in the studio. Learn more about how to **use serverless compute for model training**

Though compute is the most important resource when working with machine learning workloads, it can also be the most cost-intensive. Therefore, a best practice is to only allow administrators to create and manage compute resources. Data scientists shouldn't be allowed to edit compute, but only use the available compute to run their workloads.

Create and manage datastores

The workspace doesn't store any data itself. Instead, all data is stored in **datastores**, which are references to Azure data services. The connection information to a data service that a datastore represents, is stored in the Azure Key Vault.

When a workspace is created, an Azure Storage account is created and automatically connected to the workspace. As a result, you have four datastores already added to your workspace:

- `workspaceartifactstore`: Connects to the `azureml` container of the Azure Storage account created with the workspace. Used to store compute and experiment logs when running jobs.
- `workspaceworkingdirectory`: Connects to the file share of the Azure Storage account created with the workspace used by the **Notebooks** section of the studio. Whenever you upload files or folders to access from a compute instance, it's uploaded to this file share.

- `workspaceblobstore`: Connects to the Blob Storage of the Azure Storage account created with the workspace. Specifically the `azureml-blobstore-...` container. Set as the default datastore, which means that whenever you create a data asset and upload data, it's stored in this container.
- `workspacefilestore`: Connects to the file share of the Azure Storage account created with the workspace. Specifically the `azureml-filestore-...` file share.

Additionally, you can create datastores to connect to other Azure data services. Most commonly, your datastores will connect to an Azure Storage Account or Azure Data Lake Storage (Gen2) as those data services are most often used in data science projects.

Identify Azure Machine Learning assets

Completed100 XP

- 7 minutes

As a data scientist, you'll mostly work with **assets** in the Azure Machine Learning workspace. Assets are created and used at various stages of a project and include:

- Models
- Environments
- Data
- Components

Create and manage models

The end product of training a model is the model itself. You can train machine learning models with various frameworks, like Scikit-learn or PyTorch. A common way to store such models is to package the model as a Python pickle file (`.pkl` extension).

Alternatively, you can use the open-source platform MLflow to store your model in the MLModel format.

Tip

Learn more about **logging workflow artifacts as models using MLflow and the MLModel format**.

Whatever format you choose, binary file(s) will represent the model and any corresponding metadata. To persist those files, you can create or register a model in the workspace.

When you create a **model** in the workspace, you'll specify the *name* and *version*. Especially useful when you deploy the registered model, versioning allows you to track the specific model you want to use.

Create and manage environments

When you work with cloud compute, it's important to ensure that your code runs on any compute that is available to you. Whether you want to run a script on a compute instance, or a compute cluster, the code should execute successfully.

Imagine working in Python or R, using open-source frameworks to train a model, on your local device. If you want to use a library such as Scikit-learn or PyTorch, you'll have to install it on your device.

Similarly, when you write code that uses any frameworks or libraries, you'll need to ensure the necessary components are installed on the compute that will execute the code. To list all necessary requirements, you can create **environments**. When you create an environment, you have to specify the *name* and *version*.

Environments specify software packages, environment variables, and software settings to run scripts. An environment is stored as an image in the Azure Container Registry created with the workspace when it's used for the first time.

Whenever you want to run a script, you can specify the environment that needs to be used by the compute target. The environment will install all necessary requirements on the compute before executing the script, making your code robust and reusable across compute targets.

Create and manage data

Whereas datastores contain the connection information to Azure data storage services, **data assets** refer to a specific file or folder.

You can use data assets to easily access data every time, without having to provide authentication every time you want to access it.

When you create a data asset in the workspace, you'll specify the path to point to the file or folder, and the *name* and *version*.

Create and manage components

To train machine learning models, you'll write code. Across projects, there may be code you can reuse. Instead of writing code from scratch, you'll want to reuse snippets of code from other projects.

To make it easier to share code, you can create a **component** in a workspace. To create a component, you have to specify the *name*, *version*, code, and *environment* needed to run the code.

You can use components when creating **pipelines**. A component therefore often represents a step in a pipeline, for example to normalize data, to train a regression model, or to test the trained model on a validation dataset.

Train models in the workspace

Completed 100 XP

- 6 minutes

To train models with the Azure Machine Learning workspace, you have several options:

- Use **Automated Machine Learning**.
- Run a Jupyter notebook.
- Run a script as a job.

Explore algorithms and hyperparameter values with Automated Machine Learning

When you have a training dataset and you're tasked with finding the best performing model, you might want to experiment with various algorithms and hyperparameter values.

Manually experimenting with different configurations to train a model might take long. Alternatively, you can use Automated Machine Learning to speed up the process.

Automated Machine Learning iterates through algorithms paired with feature selections to find the best performing model for your data.

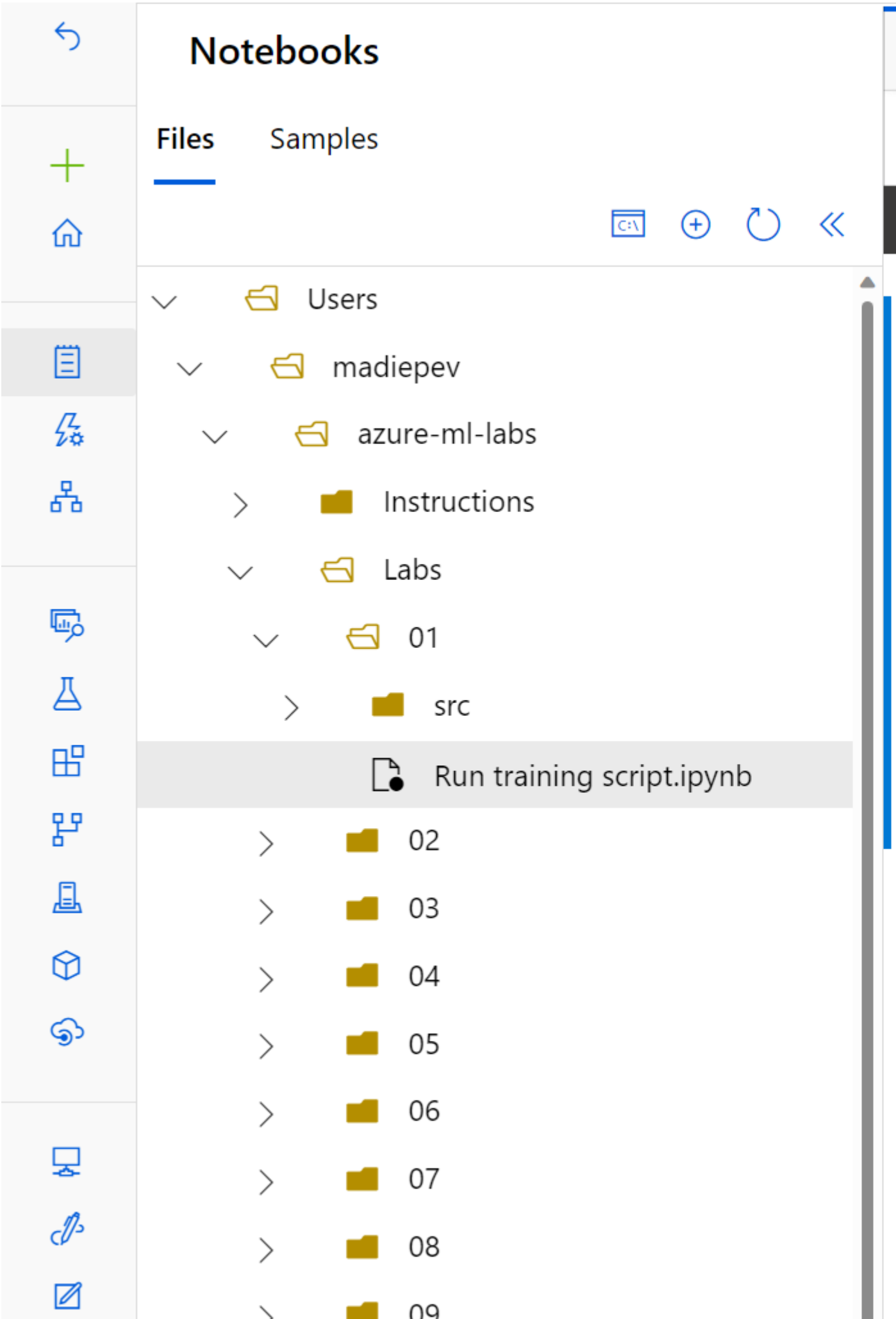
Create a new Automated ML job

- Select data asset
- Configure job
- **Select task and settings**
- Hyperparameter configuration (Computer Vision only)
- Validate and test

Run a notebook

When you prefer to develop by running code in notebooks, you can use the built-in notebook feature in the workspace.

The **Notebooks** page in the studio allows you to edit and run Jupyter notebooks.


Notebooks
Files
Samples
Users
madiepev
azure-ml-labs
Instructions
Labs
01
src
Run training script.ipynb
02
03
04
05
06
07
08
09

All files you clone or create in the notebooks section are stored in the file share of the Azure Storage account created with the workspace.

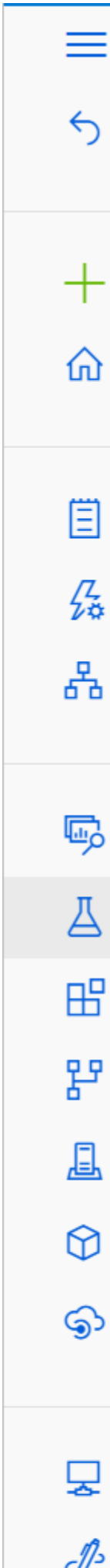
To run notebooks, you'll use a compute instance as they're ideal for development and work similar to a virtual machine.

You can also choose to edit and run notebooks in Visual Studio Code, while still using a compute instance to run the notebooks.

Run a script as a job

When you want to prepare your code to be production ready, it's better to use scripts. You can easily automate the execution of script to automate any machine learning workload.

You can run a script as a **job** in Azure Machine Learning. When you submit a job to the workspace, all inputs and outputs will be stored in the workspace.



Microsoft Non-Production > mlw-dp100-labs >

diabetes-train-mlflow

Completed

Overview | Metrics | Images | Child jobs

Refresh | Connect to compute | Edit and

Properties

Status
Completed

Created on
Nov 4, 2022

Start time
Nov 4, 2022

Duration
1m 36.99s

Compute duration
1m 36.99s

Name
yellow_head

Command
python train.py --training_data ${{inputs.diab

There are different types of jobs depending on how you want to execute a workload:

- **Command**: Execute a single script.
- **Sweep**: Perform hyperparameter tuning when executing a single script.
- **Pipeline**: Run a pipeline consisting of multiple scripts or components.

Note

When you submit a pipeline you created with the designer it will run as a pipeline job. When you submit an Automated Machine Learning experiment, it will also run as a job.

1.

A data scientist needs access to the Azure Machine Learning workspace to run a script as a job. Which role should be used to give the data scientist the necessary access to the workspace?

Reader.

Azure Machine Learning Data Scientist.

Correct. An Azure Machine Learning Data Scientist is allowed to submit a job.

Azure Machine Learning Compute Operator.

Incorrect. An Azure Machine Learning Data Scientist is allowed to submit a job.

Explore the studio

Completed 100 XP

- 6 minutes

The easiest and most intuitive way to interact with the Azure Machine Learning workspace, is by using the **studio**.

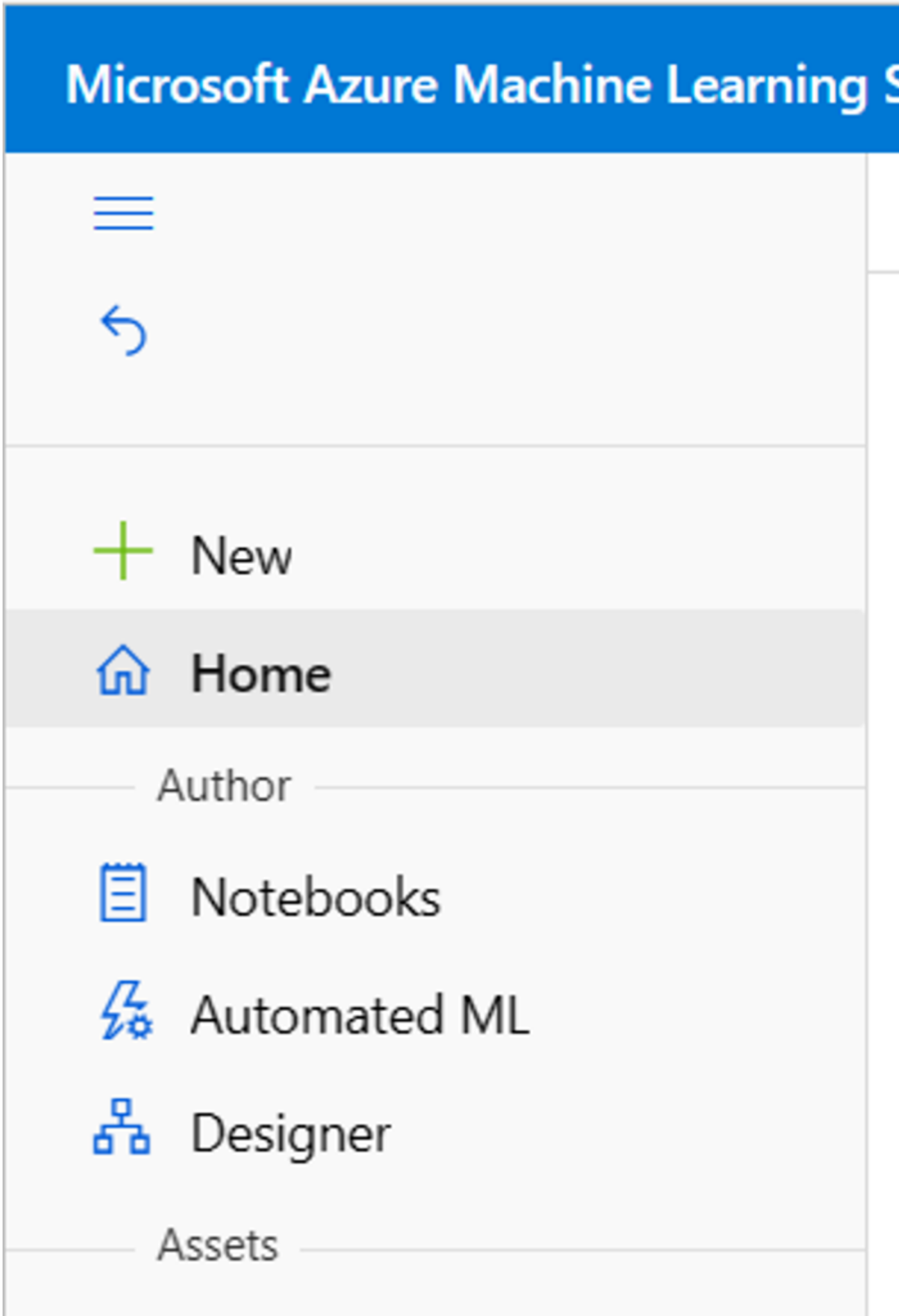
The Azure Machine Learning studio is a web portal, which provides an overview of all resources and assets available in the workspace.

Access the studio

After you've created an Azure Machine Learning workspace, there are two common ways to access the Azure Machine Learning studio:

- Launch the studio from the **Overview** page of the Azure Machine Learning workspace resource in the Azure portal.
- Navigate to the studio directly by signing in at https://ml.azure.com using the credentials associated with your Azure subscription.

When you've opened your workspace in the Azure Machine Learning studio, a menu appears in the sidebar.


Microsoft Azure Machine Learning S
New
Home
Author
Notebooks
Automated ML
Designer
Assets

The menu shows what you can do in the studio:

- **Author**: Create new jobs to train and track a machine learning model.
- **Assets**: Create and review assets you use when training models.
- **Manage**: Create and manage resources you need to train models.

Though you can use each tool at any time, the studio is ideal for quick experimentation or when you want to explore your past jobs.

For example, use the studio if you want to verify that your pipeline ran successfully. Or when a pipeline job has failed, you can use the studio to navigate to the logs and review the error messages.

For more repetitive work, or tasks that you'd like to automate, the Azure CLI or Python SDK are better suited as these tools allow you to define your work in code.

Explore the Python SDK

Completed100 XP

- 7 minutes

Important

Currently, there are two versions of the Python SDK: version 1 (v1) and version 2 (v2). For any new projects, you should use v2 and therefore, **the content in this unit only covers v2**. Learn more about **deciding between v1 and v2**.

Data scientists can use Azure Machine Learning to train, track, and manage machine learning models. As a data scientist, you'll mostly work with the assets within the Azure Machine Learning workspace for your machine learning workloads.

As most data scientists are familiar with Python, Azure Machine Learning offers a software development kit (SDK) so that you can interact with the workspace using Python.

The Python SDK for Azure Machine Learning is an ideal tool for data scientists that can be used in any Python environment. Whether you normally work with Jupyter notebooks, Visual Studio Code, you can install the Python SDK and connect to the workspace.

Install the Python SDK

To install the Python SDK within your Python environment, you need Python 3.7 or later. You can install the package with `pip`:

Copy

```
pip install azure-ai-ml
```

Note

When working with notebooks within the Azure Machine Learning studio, the new Python SDK is already installed when using Python 3.10 or later. You can use the Python SDK v2 with earlier versions of Python, but you'll have to install it first.

Connect to the workspace

After the Python SDK is installed, you'll need to connect to the workspace. By connecting, you're authenticating your environment to interact with the workspace to create and manage assets and resources.

To authenticate, you need the values to three necessary parameters:

- `subscription_id`: Your subscription ID.
- `resource_group`: The name of your resource group.
- `workspace_name`: The name of your workspace.

Next, you can define the authentication by using the following code:

PythonCopy

```python
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential

ml_client = MLClient(
    DefaultAzureCredential(), subscription_id, resource_group, workspace
)
```

After defining the authentication, you need to call `MLClient` for the environment to connect to the workspace. You'll call `MLClient` anytime you want to create or update an asset or resource in the workspace.

For example, you'll connect to the workspace when you create a new job to train a model:

PythonCopy

```python
from azure.ai.ml import command

# configure job
job = command(
    code="./src",
    command="python train.py",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    experiment_name="train-model"
)
```

```
# connect to workspace and submit job
returned_job = ml_client.create_or_update(job)
```

Use the reference documentation

To efficiently work with the Python SDK, you'll need to use the reference documentation. In the reference documentation, you'll find all possible classes, methods, and parameters available within the Python SDK.

The reference documentation on the MLClient class includes the methods you can use to connect and interact with the workspace. Moreover, it also links to the possible operations for the various entities like how to list the existing datastores in your workspace.

The reference documentation also includes a list of the classes for all entities you can interact with. For example, separate classes exist when you want to create a datastore that links to an Azure Blob Storage, or to an Azure Data Lake Gen 2.

By selecting a specific class like `AmlCompute` from the list of entities, you can find a more detailed page on how to use the class and what parameters it accepts.

Explore the CLI

Completed 100 XP

- 8 minutes

Important

The content in this unit only covers version 2 of the CLI. Learn more about **deciding between version 1 and 2**.

Another code-based approach to interact with the Azure Machine Learning workspace is the command-line interface (CLI). As a data scientist, you may not work with the CLI as much as you do with Python. The Azure CLI is commonly used by administrators and engineers to automate tasks in Azure.

There are many advantages to using the Azure CLI with Azure Machine Learning. The Azure CLI allows you to:

- Automate the creation and configuration of assets and resources to make it **repeatable**.
- Ensure **consistency** for assets and resources that must be replicated in multiple environments (for example, development, test, and production).

- Incorporate machine learning asset configuration into developer operations (**DevOps**) **workflows**, such as **continuous integration** and **continuous deployment** (**CI/CD**) pipelines.

To interact with the Azure Machine Learning workspace using the Azure CLI, you'll need to install the Azure CLI and the Azure Machine Learning extension.

Install the Azure CLI

You can install the Azure CLI on a Linux, Mac, or Windows computer. With the Azure CLI, you run commands or scripts to manage Azure resources. You can also use the Azure CLI from a browser through the Azure Cloud Shell. No matter which platform you choose, you can execute the same tasks. But, the installation of the Azure CLI, the commands, and scripts are different across platforms.

Important

To install the Azure CLI on your computer you can use a package manager. Here are the instructions to **install the Azure CLI**, based on the platform you choose. You don't need to install the Azure CLI if you use the Azure Cloud Shell. Learn more about how to use **the Azure Cloud Shell in this overview**.

Install the Azure Machine Learning extension

After you've installed the Azure CLI, or set up the Azure Cloud Shell, you need to install the Azure Machine Learning extension to manage Azure Machine Learning resources using the Azure CLI.

You can install the Azure Machine Learning extension `ml` with the following command:

Azure CLICopy
```
az extension add -n ml -y
```

You can then run the help command `-h` to check that the extension is installed and to get a list of commands available with this extension. The list gives an overview of the tasks you can execute with the Azure CLI extension for Azure Machine Learning:

Azure CLICopy
```
az ml -h
```

Work with the Azure CLI

To use the Azure CLI to interact with the Azure Machine Learning workspace, you'll use **commands**. Each command is prefixed with `az ml`. You can find the list of commands in the reference documentation of the CLI.

For example, to create a compute target, you can use the following command:

Azure CLICopy

```
az ml compute create --name aml-cluster --size STANDARD_DS3_v2 --min-instances 0 --max-instances 5 --type AmlCompute --resource-group my-resource-group --workspace-name my-workspace
```

To explore all possible parameters that you can use with a command, you can review the reference documentation for the specific command.

As you define the parameters for an asset or resource you want to create, you may prefer using YAML files to define the configuration instead. When you store all parameter values in a YAML file, it becomes easier to organize and automate tasks.

For example, you can also create the same compute target by first defining the configuration in a YAML file:

ymlCopy

```
$schema: https://azuremlschemas.azureedge.net/latest/amlCompute.schema.json
name: aml-cluster
type: amlcompute
size: STANDARD_DS3_v2
min_instances: 0
max_instances: 5
```

All possible parameters that you can include in the YAML file can be found in the reference documentation for the specific asset or resource you want to create like a compute cluster.

When you saved the YAML file as `compute.yml`, you can create the compute target with the following command:

Azure CLICopy

```
az ml compute create --file compute.yml --resource-group my-resource-group --workspace-name my-workspace
```

You can find an overview of all the YAML schemas in the reference documentation.

Tip

Learn more about **how to use the CLI (v2) with Azure Machine Learning to train models**.

Efficient data storage options

Completed 100 XP

- 5 minutes

As many machine-learning models benefit from large amounts of data, store your large dataset efficiently to reduce processing time.

Recall that you received a large dataset from the analytics team. You know you have to store it efficiently to optimize processing time. Whether it is for preparing and exploring the flight data, or to train a machine or deep learning model on the data.

You'll learn some best practices to store large amounts of data.

Choose an Azure Data Lake Storage Gen2

Together with the analytics team, you have decided to do all processing and model training in Azure Machine Learning. To easily and securely access the data from the Azure Machine Learning workspace, you want to store the data in Azure.

Although there are several options to store data in Azure, the best solution when working with Azure Machine Learning is to store the data in an **Azure Data Lake Storage Gen2**, no matter the size of the data.

Take advantage of the hierarchical namespace

Compared to an Azure Blob Storage, the Azure Data Lake Gen2 provides a **hierarchical namespace** to store your files.

With the hierarchical namespace, you can use a nested folder structure to optimize listing operations. Next to better scalability and performance, structuring your files like this will also allow for fine-granular access.

Tip

Learn more about **the capabilities of Azure Data Lake Storage Gen2**.

Use a nested folder structure

The reason why it's more efficient to use a data lake instead of a flat object storage, is because it's best to avoid putting all your files in one folder.

If all files are stored in one folder, regardless of which storage solution you choose, reading the files will be demanding for your compute.

The flight data you received, are a large collection of CSV files that show the flight information for each month. Based on these recommendations, you choose to migrate the data to an Azure Data Lake Storage and create a nested folder structure based on date. Doing so will allow you to easily select for which time period you want to load in the flight data.

Avoid small files

And finally, when storing your files you should avoid having many small files. Reading a 1000 small files is much slower than reading one file with 1000x the size.

Access data in Azure Machine Learning

After migrating the data and allowing Azure Machine Learning to connect to the Azure Data Lake, you want to use the flight data as input when running a job.

When working with data in Azure Machine Learning, you can either download or mount the data to the Compute Cluster assigned to run the job:

- Download the data if you estimate the dataset will fit onto the virtual machine's disk.
- Mount the data if you expect the dataset to be too large to be downloaded on to the disk.

Optimize data loading and preprocessing in Azure Machine Learning

Completed100 XP

- 8 minutes

Data scientists can load and process data more quickly using RAPIDS with GPU compute.

Recall the large volume of flight data you received as a data scientist for the air carrier company. One important performance metric of this company is whether a flight is delayed for more than 15 minutes. You want to train a classification model so you can predict when a future flight will be delayed. Before you can train the model, you need to load and process the data.

You'll learn how to load and process data in Azure Machine Learning by using the RAPIDS cuDF library.

RAPIDS

RAPIDS is a machine-learning framework created by NVIDIA. You can use the RAPIDS libraries for:

- Data loading and preprocessing
- Machine learning
- Graph analytics
- Visualization

Here, we'll focus on data loading and preprocessing with RAPIDS cuDF.

Note

To learn more about how to use RAPIDS for machine learning, graph analytics and visualization, go to the **RAPIDS Docs**.

cuDF

A typical approach for data scientists is to use the **pandas** library to process tabular data. When working with large volumes of data however, it can take a long time to load and manipulate your data with pandas on CPUs. Next to a decline in performance, you may run into memory issues. To improve performance and avoid memory issues on your compute, use GPUs to process large data more quickly. As pandas only works with CPUs, you'll have to use the **cuDF** library to work with GPUs.

cuDF is essentially pandas on GPU. Similar to pandas, cuDF is a Python DataFrame library for loading and processing data.

For example, let's look at how we can **load data** with both pandas and cuDF to see how similar they are:

Load a CSV file into a DataFrame with pandas:

PythonCopy

```
import pandas as pd

flight_data = pd.read_csv(csvfile)
```

Load a CSV file into a DataFrame with cuDF:

PythonCopy

```
import cudf

flight_data = cudf.read_csv(csvfile)
```

After you loaded the data, you may want to process it to prepare it for model training. Let's look at some data manipulations we can do with cuDF.

Tip

When working with data, use the **cuDF documentation** to find the methods you need.

For example, working for the air carrier company you've received all the flight data. The data includes where and when an airplane took off, and where and when it landed. The values for the origin and destination of a flight are the codes of the airports they visited.

You find an open dataset with information about the airports, like the full names of the airport, the codes, and the location on the map. To add the airport data to your flight data, you want to merge these two datasets.

You can **merge** your flight data with the airports dataset using the following code:

PythonCopy

```
data = cudf.merge(flight_data, airports, left_on='Dest', right_on='iata_code',
how='left')
data = cudf.merge(flight_data, airports, left_on='Origin', right_on='iata_code',
how='left')
```

To make columns more easily interpretable, you can **rename** them:

PythonCopy

```
data = data.rename(columns= { 'latitude_deg_y' : 'origin_lat', 'longitude_deg_y':
'origin_long',
                              'latitude_deg_x' : 'dest_lat', 'longitude_deg_x':
'dest_long',
                              'Description' : 'Airline'})
```

Machine learning models can be affected by missing data. An easy way to avoid this, especially when working with large datasets, is to remove the rows in which any of the columns have missing values.

Similar to when using pandas, you can **remove missing values** by dropping the rows using the `dropna` function:

PythonCopy

```
data = data.dropna()
```

For data scientists working with pandas, cuDF is a familiar approach to data processing that is compatible with GPU and will allow you to process data much faster.

Tip

If you want to improve performance even more, learn about **multi-GPU processing with Dask**.

Introduction

Completed 100 XP

- 1 minute

Data is a fundamental element in any machine learning workload. You need data to train a model and you create data when using a model to generate predictions.

To work with data in Azure Machine Learning, you can access data by using **Uniform Resource Identifiers** (**URIs**). When you work with a data source or a specific file or folder repeatedly, you can create **datastores** and **data assets** within the Azure Machine Learning workspace. Datastores and data assets allow you to securely store the connection information to your data.

In this module, you learn how to create and use URIs, datastores, and data assets in Azure Machine Learning.

Create a datastore

Completed 100 XP

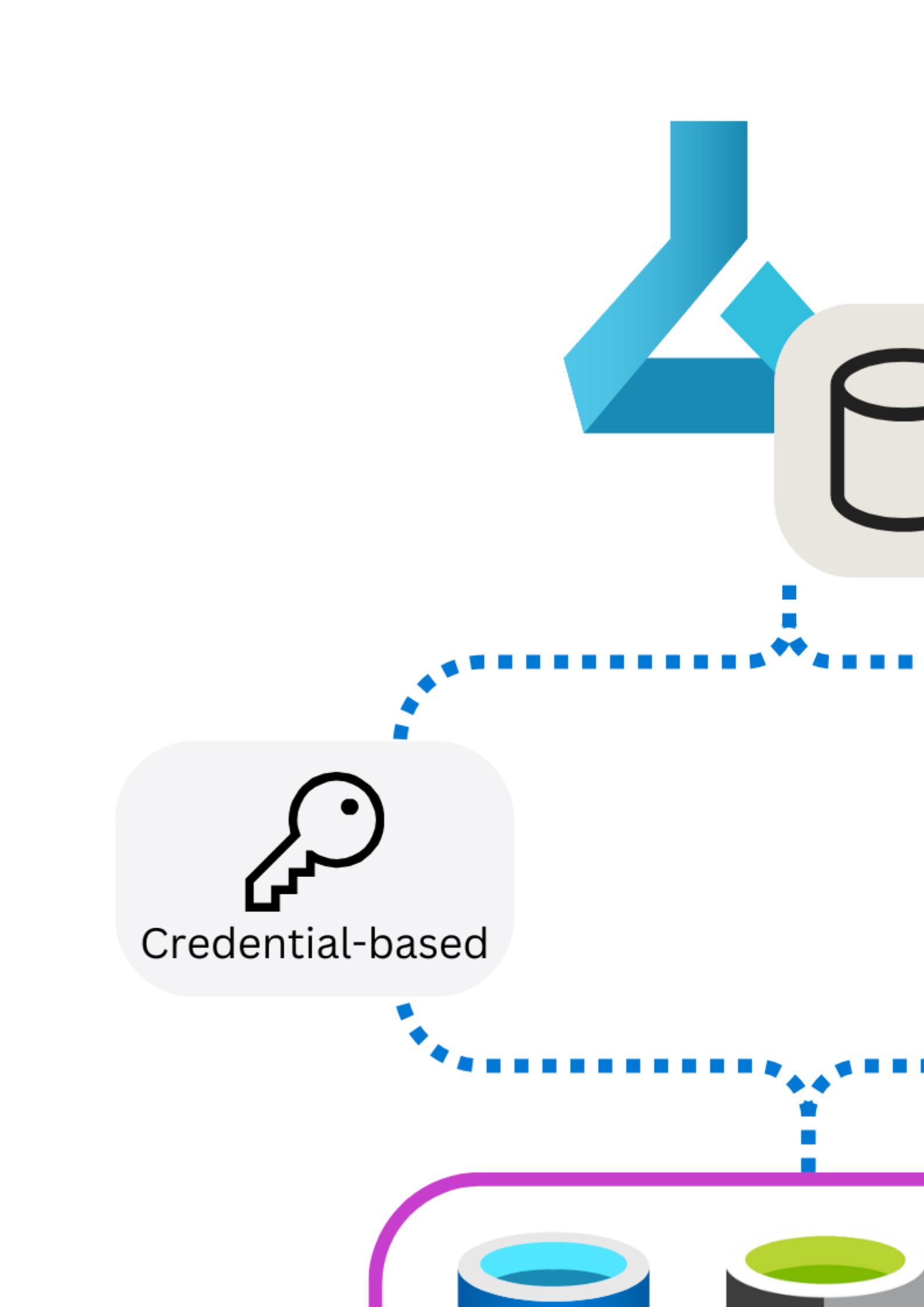
- 7 minutes

In Azure Machine Learning, **datastores** are abstractions for cloud data sources. They encapsulate the information needed to connect to data sources, and securely store this connection information so that you don't have to code it in your scripts.

The benefits of using datastores are:

- Provides easy-to-use URIs to your data storage.
- Facilitates data discovery within Azure Machine Learning.
- Securely stores connection information, without exposing secrets and keys to data scientists.

When you create a datastore with an existing storage account on Azure, you have the choice between two different authentication methods:


Credential-based

- **Credential-based**: Use a *service principal*, *shared access signature* (*SAS*) token or *account key* to authenticate access to your storage account.
- **Identity-based**: Use your *Microsoft Entra identity* or *managed identity*.

Understand types of datastores

Azure Machine Learning supports the creation of datastores for multiple kinds of Azure data source, including:

- Azure Blob Storage
- Azure File Share
- Azure Data Lake (Gen 2)

Use the built-in datastores

Every workspace has four built-in datastores (two connecting to Azure Storage blob containers, and two connecting to Azure Storage file shares), which are used as system storages by Azure Machine Learning.

In most machine learning projects, you need to work with data sources of your own. For example, you can integrate your machine learning solution with data from existing applications or data engineering pipelines.

Create a datastore

Datastores are attached to workspaces and are used to store connection information to storage services. When you create a datastore, you provide a name that can be used to retrieve the connection information.

Datastores allow you to easily connect to storage services without having to provide all necessary details every time you want to read or write data. It also creates a protective layer if you want users to use the data, but not connect to the underlying storage service directly.

Create a datastore for an Azure Blob Storage container

You can create a datastore through the graphical user interface, the Azure command-line interface (CLI), or the Python software development kit (SDK).

Depending on the storage service you want to connect to, there are different options for Azure Machine Learning to authenticate.

For example, when you want to create a datastore to connect to an Azure Blob Storage container, you can use an account key:

PythonCopy

```
blob_datastore = AzureBlobDatastore(
                         name = "blob_example",
                         description = "Datastore pointing to a blob container",
                         account_name = "mytestblobstore",
                         container_name = "data-container",
                         credentials = AccountKeyCredentials(
                         account_key="XXXxxxXXXxXXXXxxXXX"
                         ),
)
ml_client.create_or_update(blob_datastore)
```

Alternatively, you can create a datastore to connect to an Azure Blob Storage container by using a SAS token to authenticate:

PythonCopy

```
blob_datastore = AzureBlobDatastore(
name="blob_sas_example",
description="Datastore pointing to a blob container",
account_name="mytestblobstore",
container_name="data-container",
credentials=SasTokenCredentials(
sas_token="?xx=XXXX-XX-XX&xx=xxxx&xxx=xxx&xx=xxxxxxxxxxx&xx=XXXX-XX-
XXXXX:XX:XXX&xx=XXXX-XX-
XXXXX:XX:XXX&xxx=xxxxx&xxx=XXxXXXxxxxxXXXXXXXxXxxxXXXXXxxXXXXXxXXXXxXXXxXXxXX"
),
)
ml_client.create_or_update(blob_datastore)
```

Create a data asset

Completed100 XP

- 12 minutes

As a data scientist, you want to focus on training machine learning models. Though you need access to data as input for a machine learning model, you don't want to worry about *how* to get access. To simplify getting access to the data you want to work with, you can use **data assets**.

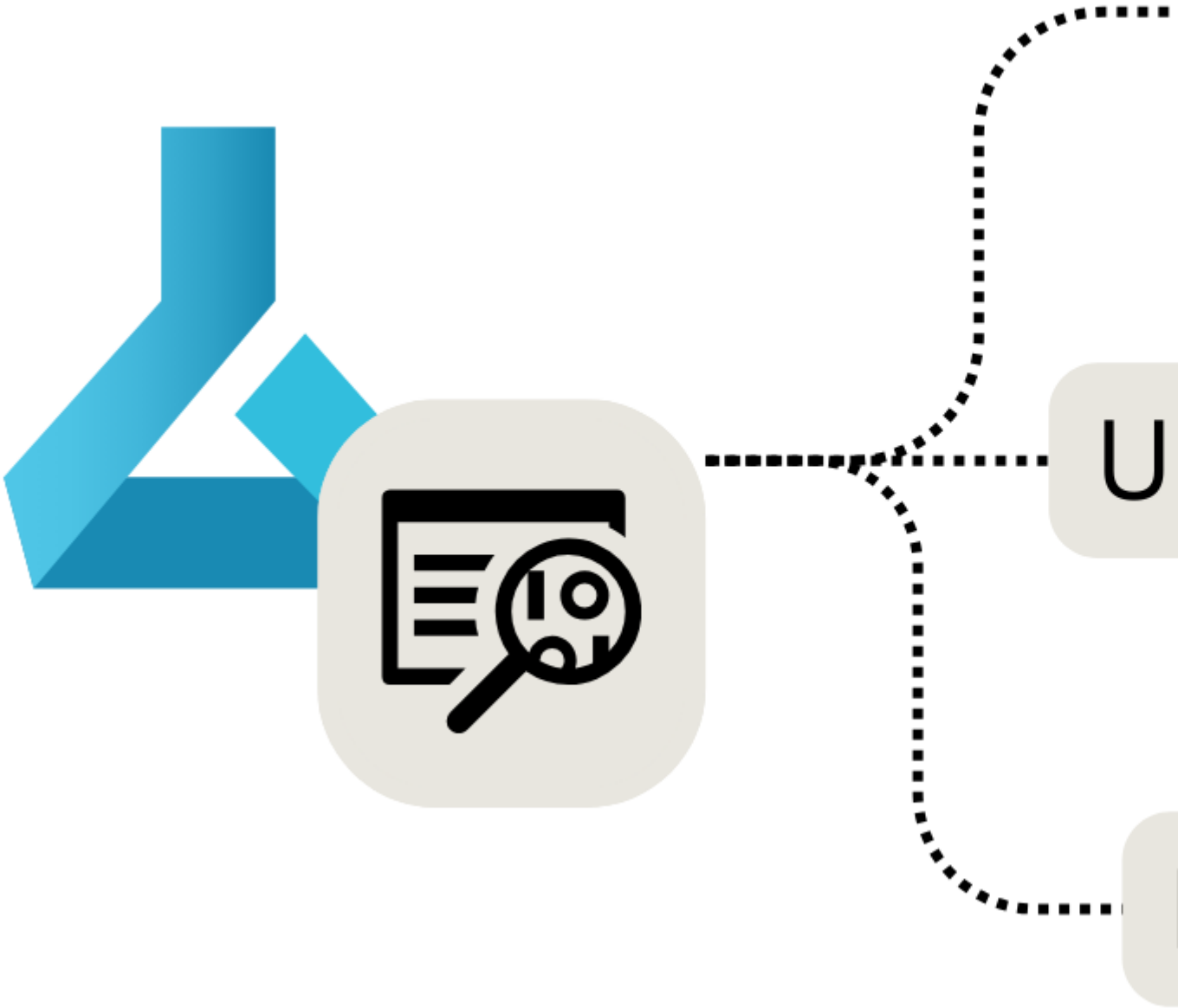
Understand data assets

In Azure Machine Learning, data assets are references to where the data is stored, how to get access, and any other relevant metadata. You can create data assets to get access to data in datastores, Azure storage services, public URLs, or data stored on your local device.

The benefits of using data assets are:

- You can **share and reuse data** with other members of the team such that they don't need to remember file locations.
- You can **seamlessly access data** during model training (on any supported compute type) without worrying about connection strings or data paths.
- You can **version** the metadata of the data asset.

There are three main types of data assets you can use:



- **URI file**: Points to a specific file.
- **URI folder**: Points to a folder.
- **MLTable**: Points to a folder or file, and includes a schema to read as tabular data.

Note

URI stands for **Uniform Resource Identifier** and stands for a storage location on your local computer, Azure Blob or Data Lake Storage, publicly available https location, or even an attached datastore.

When to use data assets

Data assets are most useful when executing machine learning tasks as Azure Machine Learning jobs. As a job, you can run a Python script that takes inputs and generates outputs. A data asset can be parsed as both an input or output of an Azure Machine Learning job.

Let's take a look at each of the types of data assets, how to create them, and how to use the data asset in a job.

Create a URI file data asset

A URI file data asset points to a specific file. Azure Machine Learning only stores the path to the file, which means you can point to any type of file. When you use the data asset, you specify how you want to read the data, which depends on the type of data you're connecting to.

The supported paths you can use when creating a URI file data asset are:

- Local: `./<path>`
- Azure Blob Storage: `wasbs://<account_name>.blob.core.windows.net/<container_name>/<folder>/<file>`
- Azure Data Lake Storage (Gen 2): `abfss://<file_system>@<account_name>.dfs.core.windows.net/<folder>/<file>`
- Datastore: `azureml://datastores/<datastore_name>/paths/<folder>/<file>`

Important

When you create a data asset and point to a file or folder stored on your local device, a copy of the file or folder will be uploaded to the default datastore `workspaceblobstore`. You can find the file or folder in the `LocalUpload` folder. By uploading a copy, you'll still be able to access the data from the Azure Machine

Learning workspace, even when the local device on which the data is stored is unavailable.

To create a URI file data asset, you can use the following code:

PythonCopy

```
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

my_path = '<supported-path>'

my_data = Data(
    path=my_path,
    type=AssetTypes.URI_FILE,
    description="<description>",
    name="<name>",
    version="<version>"
)

ml_client.data.create_or_update(my_data)
```

When you parse the URI file data asset as input in an Azure Machine Learning job, you first need to read the data before you can work with it.

Imagine you create a Python script you want to run as a job, and you set the value of the input parameter `input_data` to be the URI file data asset (which points to a CSV file). You can read the data by including the following code in your Python script:

PythonCopy

```
import argparse
import pandas as pd

parser = argparse.ArgumentParser()
parser.add_argument("--input_data", type=str)
args = parser.parse_args()

df = pd.read_csv(args.input_data)
print(df.head(10))
```

If your URI file data asset points to a different type of file, you need to use the appropriate Python code to read the data. For example, if instead of CSV files, you're working with JSON files, you'd use `pd.read_json()` instead.

Create a URI folder data asset

A URI folder data asset points to a specific folder. It works similar to a URI file data asset and supports the same paths.

To create a URI folder data asset with the Python SDK, you can use the following code:

PythonCopy

```python
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

my_path = '<supported-path>'

my_data = Data(
    path=my_path,
    type=AssetTypes.URI_FOLDER,
    description="<description>",
    name="<name>",
    version='<version>'
)

ml_client.data.create_or_update(my_data)
```

When you parse the URI folder data asset as input in an Azure Machine Learning job, you first need to read the data before you can work with it.

Imagine you create a Python script you want to run as a job, and you set the value of the input parameter `input_data` to be the URI folder data asset (which points to multiple CSV files). You can read all CSV files in the folder and concatenate them, which you can do by including the following code in your Python script:

PythonCopy

```python
import argparse
import glob
import pandas as pd

parser = argparse.ArgumentParser()
parser.add_argument("--input_data", type=str)
args = parser.parse_args()

data_path = args.input_data
all_files = glob.glob(data_path + "/*.csv")
df = pd.concat((pd.read_csv(f) for f in all_files), sort=False)
```

Depending on the type of data you're working with, the code you use to read the files can change.

Create a MLTable data asset

A MLTable data asset allows you to point to tabular data. When you create a MLTable data asset, you specify the schema definition to read the data. As the schema is already defined and stored with the data asset, you don't have to specify how to read the data when you use it.

Therefore, you want to use a MLTable data asset when the schema of your data is complex or changes frequently. Instead of changing how to read the data in every script that uses the data, you only have to change it in the data asset itself.

When you define the schema when creating a MLTable data asset, you can also choose to only specify a subset of the data.

For certain features in Azure Machine Learning, like Automated Machine Learning, you need to use a MLTable data asset, as Azure Machine Learning needs to know how to read the data.

To define the schema, you can include a **MLTable file** in the same folder as the data you want to read. The MLTable file includes the path pointing to the data you want to read, and how to read the data:

ymlCopy

```yaml
type: mltable

paths:
  - pattern: ./*.txt
transformations:
  - read_delimited:
      delimiter: ','
      encoding: ascii
      header: all_files_same_headers
```

Tip

Learn more on **how to create the MLTable file and which transformations you can include**.

To create a MLTable data asset with the Python SDK, you can use the following code:

PythonCopy

```python
from azure.ai.ml.entities import Data
from azure.ai.ml.constants import AssetTypes

my_path = '<path-including-mltable-file>'

my_data = Data(
    path=my_path,
    type=AssetTypes.MLTABLE,
    description="<description>",
    name="<name>",
    version='<version>'
)

ml_client.data.create_or_update(my_data)
```

When you parse a MLTable data asset as input to a Python script you want to run as an Azure Machine Learning job, you can include the following code to read the data:

PythonCopy

```python
import argparse
import mltable
import pandas

parser = argparse.ArgumentParser()
parser.add_argument("--input_data", type=str)
args = parser.parse_args()

tbl = mltable.load(args.input_data)
df = tbl.to_pandas_dataframe()

print(df.head(10))
```

A common approach is to convert the tabular data to a Pandas data frame. However, you can also convert the data to a Spark data frame if that suits your workload better.

Choose the appropriate compute target

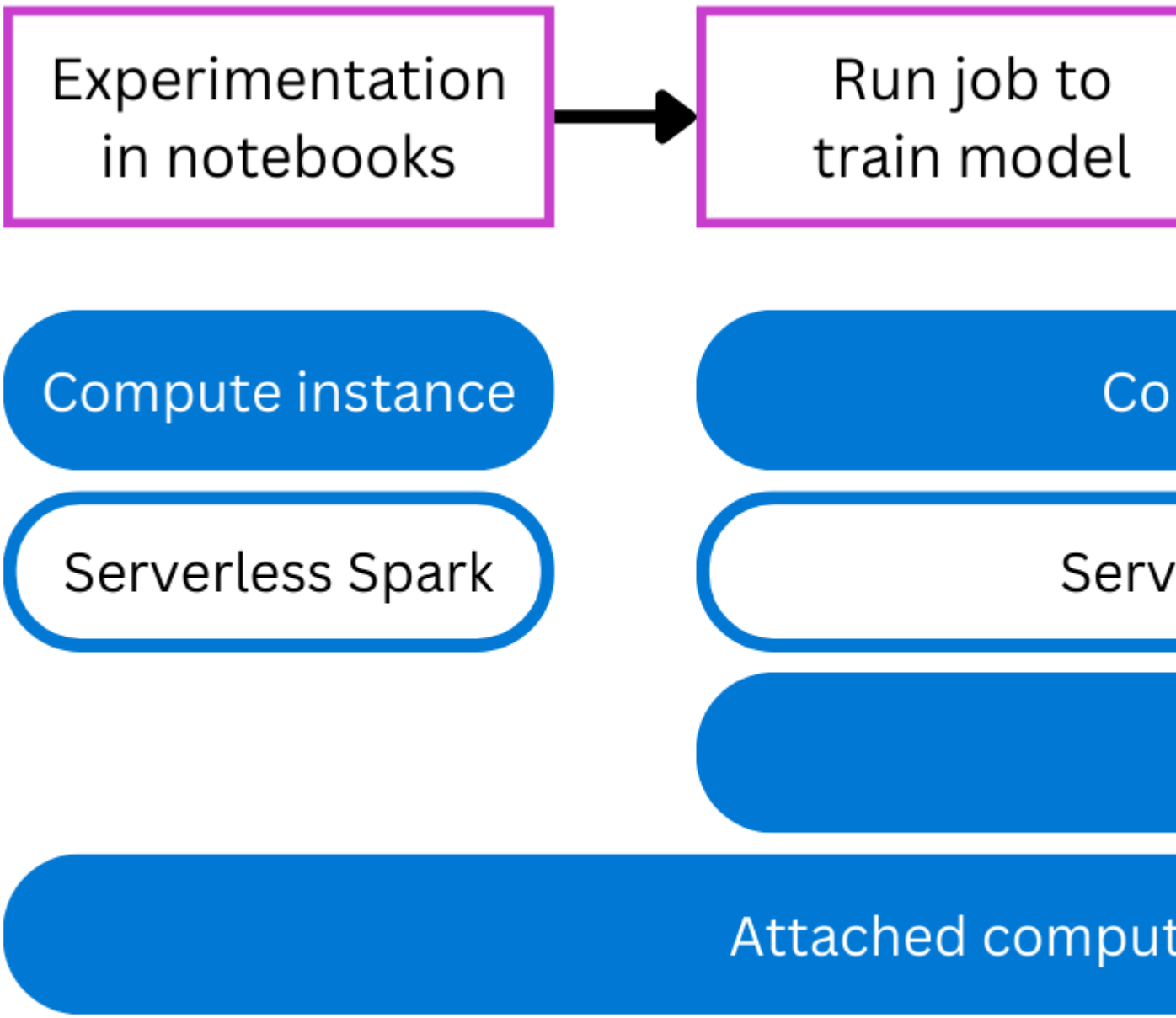
Completed100 XP

- 7 minutes

In Azure Machine Learning, *compute targets* are physical or virtual computers on which jobs are run.

Understand the available types of compute

Azure Machine Learning supports multiple types of compute for experimentation, training, and deployment. By having multiple types of compute, you can select the most appropriate type of compute target for your needs.



- **Compute instance**: Behaves similarly to a virtual machine and is primarily used to run notebooks. It's ideal for *experimentation*.
- **Compute clusters**: Multi-node clusters of virtual machines that automatically scale up or down to meet demand. A cost-effective way to run scripts that need to process

large volumes of data. Clusters also allow you to use parallel processing to distribute the workload and reduce the time it takes to run a script.
- **Kubernetes clusters**: Cluster based on Kubernetes technology, giving you more control over how the compute is configured and managed. You can attach your self-managed Azure Kubernetes (AKS) cluster for cloud compute, or an Arc Kubernetes cluster for on-premises workloads.
- **Attached compute**: Allows you to attach existing compute like Azure virtual machines or Azure Databricks clusters to your workspace.
- **Serverless compute**: A fully managed, on-demand compute you can use for training jobs.

Note

Azure Machine Learning offers you the option to create and manage your own compute or to use compute that is fully managed by Azure Machine Learning.

When to use which type of compute?

In general, there are some best practices that you can follow when working with compute targets. To understand how to choose the appropriate type of compute, several examples are provided. Remember that which type of compute you use always depends on your specific situation.

Choose a compute target for experimentation

Imagine you're a data scientist and you're asked to develop a new machine learning model. You likely have a small subset of the training data with which you can experiment.

During experimentation and development, you prefer working in a Jupyter notebook. A notebook experience benefits most from a compute that is continuously running.

Many data scientists are familiar with running notebooks on their local device. A cloud alternative managed by Azure Machine Learning is a *compute instance*. Alternatively, you can also opt for *Spark serverless compute* to run Spark code in notebooks, if you want to make use of Spark's distributed compute power.

Choose a compute target for production

After experimentation, you can train your models by running Python scripts to prepare for production. Scripts will be easier to automate and schedule for when you want to retrain your model continuously over time. You can run scripts as (pipeline) jobs.

When moving to production, you want the compute target to be ready to handle large volumes of data. The more data you use, the better the machine learning model is likely to be.

When training models with scripts, you want an on-demand compute target. A *compute cluster* automatically scales up when the script(s) need to be executed, and scales down when the script finishes executing. If you want an alternative that you don't have to create and manage, you can use Azure Machine Learning's *serverless compute*.

Choose a compute target for deployment

The type of compute you need when using your model to generate predictions depends on whether you want batch or real-time predictions.

For batch predictions, you can run a pipeline job in Azure Machine Learning. Compute targets like compute clusters and Azure Machine Learning's serverless compute are ideal for pipeline jobs as they're on-demand and scalable.

When you want real-time predictions, you need a type of compute that is running continuously. Real-time deployments therefore benefit from more lightweight (and thus more cost-efficient) compute. Containers are ideal for real-time deployments. When you deploy your model to a managed online endpoint, Azure Machine Learning creates and manages containers for you to run your model. Alternatively, you can attach Kubernetes clusters to manage the necessary compute to generate real-time predictions.

Create and use a compute cluster

Completed100 XP

- 8 minutes

After experimentation and development, you want your code to be production-ready. When you run code in production environments, it's better to use scripts instead of notebooks. When you run a script, you want to use a compute target that is scalable.

Within Azure Machine Learning, **compute clusters** are ideal for running scripts. You can create a compute cluster in the Azure Machine Learning studio, using the Azure command-line interface (CLI), or the Python software development kit (SDK).

Create a compute cluster with the Python SDK

To create a compute cluster with the Python SDK, you can use the following code:

PythonCopy

```python
from azure.ai.ml.entities import AmlCompute

cluster_basic = AmlCompute(
    name="cpu-cluster",
    type="amlcompute",
    size="STANDARD_DS3_v2",
    location="westus",
    min_instances=0,
    max_instances=2,
    idle_time_before_scale_down=120,
    tier="low_priority",
)
ml_client.begin_create_or_update(cluster_basic).result()
```

To understand which parameters the `AmlCompute` class expects, you can review the reference documentation.

When you create a compute cluster, there are three main parameters you need to consider:

- `size`: Specifies the *virtual machine type* of each node within the compute cluster. Based on the sizes for virtual machines in Azure. Next to size, you can also specify whether you want to use CPUs or GPUs.
- `max_instances`: Specifies the *maximum number of nodes* your compute cluster can scale out to. The number of parallel workloads your compute cluster can handle is analogous to the number of nodes your cluster can scale to.
- `tier`: Specifies whether your virtual machines are *low priority* or *dedicated*. Setting to low priority can lower costs as you're not guaranteed availability.

Use a compute cluster

There are three main scenarios in which you can use a compute cluster:

- Running a pipeline job you built in the Designer.
- Running an Automated Machine Learning job.
- Running a script as a job.

In each of these scenarios, a compute cluster is ideal as a compute cluster automatically scales up when a job is submitted, and automatically shut down when a job is completed.

A compute cluster also allows you to train multiple models in parallel, which is a common practice when using Automated Machine Learning.

You can run a Designer pipeline job and an Automated Machine Learning job through the Azure Machine Learning studio. When you submit the job through the studio, you can set the compute target to the compute cluster you created.

When you prefer a code-first approach, you can set the compute target to your compute cluster by using the Python SDK.

For example, when you run a script as a command job, you can set the compute target to your compute cluster with the following code:

PythonCopy

```python
from azure.ai.ml import command

# configure job
job = command(
    code="./src",
    command="python diabetes-training.py",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="cpu-cluster",
    display_name="train-with-cluster",
    experiment_name="diabetes-training"
    )

# submit job
returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

After submitting a job that uses a compute cluster, the compute cluster scales out to one or more nodes. Resizing takes a few minutes, and your job starts running once the necessary nodes are provisioned. When a job's status is *preparing*, the compute cluster is being prepared. When the status is *running*, the compute cluster is ready, and the job is running.

You're experimenting with component-based pipelines in the Designer. You want to quickly iterate and experiment, as you're trying out varying configurations of a pipeline. You're using a compute cluster. To minimize the start-up time each time you submit a pipeline, which parameter should you change?

Compute size

Idle time before scale down

Correct. By increasing the idle time before scale down, you can run multiple pipelines consecutively without the compute cluster resizing to zero nodes in between jobs.

Maximum number of instances

Incorrect. You shouldn't change the maximum number of instances when you want the compute cluster to remain available in between pipeline job runs.

Understand environments

Completed 100 XP

- 6 minutes

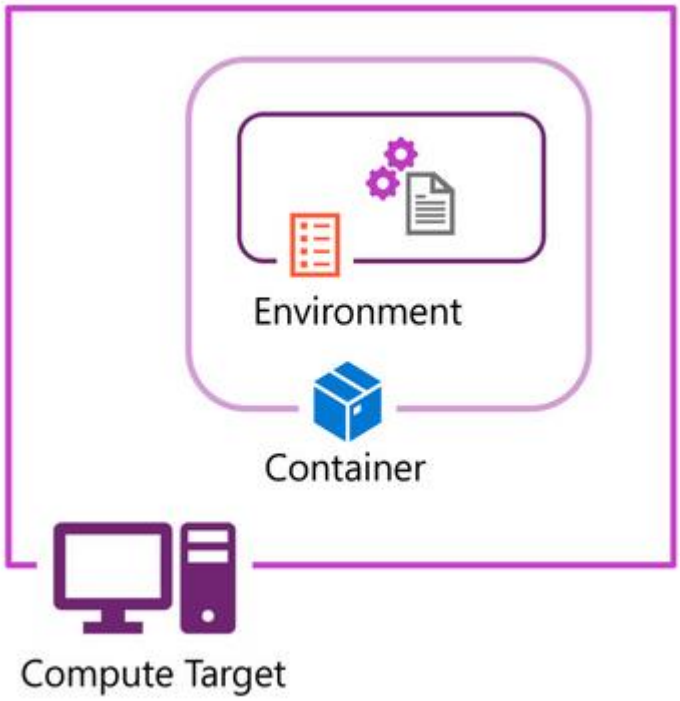
In an enterprise machine learning solution, where experiments may be run in various compute contexts, it can be important to be aware of the environments in which your experiment code is running. You can use Azure Machine Learning **environments** to create environments and specify runtime configuration for an experiment.

When you create an Azure Machine Learning workspace, **curated** environments are automatically created and made available to you. Alternatively, you can create and manage your own **custom** environments and register them in the workspace. Creating and registering custom environments makes it possible to define consistent, reusable runtime contexts for your experiments - regardless of where the experiment script is run.

What is an environment in Azure Machine Learning?

Python code runs in the context of a *virtual environment* that defines the version of the Python runtime to be used as well as the installed packages available to the code. In most Python installations, packages are installed and managed in environments using `conda` or `pip`.

To improve portability, you usually create environments in Docker containers that are in turn hosted on compute targets, such as your development computer, virtual machines, or clusters in the cloud.



Azure Machine Learning builds environment definitions into Docker images and conda environments. When you use an environment, Azure Machine Learning builds the environment on the **Azure Container registry** associated with the workspace.

Tip

When you create an Azure Machine Learning workspace, you can choose whether to use an existing Azure Container registry, or whether to let the workspace create a new registry for you when needed.

To view all available environments within the Azure Machine Learning workspace, you can list the environments in the studio, using the Azure CLI, or the Python SDK.

For example, to list the environments using the Python SDK:

PythonCopy

```
envs = ml_client.environments.list()
for env in envs:
    print(env.name)
```

To review the details of a specific environment, you can retrieve an environment by its registered name:

PythonCopy

```
env = ml_client.environments.get(name="my-environment", version="1")
print(env)
```

Explore and use curated environments

Completed100 XP

- 8 minutes

Curated environments are prebuilt environments for the most common machine learning workloads, available in your workspace by default.

Curated environments use the prefix **AzureML-** and are designed to provide for scripts that use popular machine learning frameworks and tooling.

For example, there are curated environments for when you want to run a script that trains a regression, clustering, or classification model with Scikit-Learn.

To explore a curated environment, you can view it in the studio, using the Azure CLI, or the Python SDK.

The following command allows you to retrieve the description and tags of a curated environment with the Python SDK:

PythonCopy

```
env = ml_client.environments.get("AzureML-sklearn-0.24-ubuntu18.04-py37-cpu",
version=44)
print(env. description, env.tags)
```

Use a curated environment

Most commonly, you use environments when you want to run a script as a (**command**) **job**.

To specify which environment you want to use to run your script, you reference an environment by its name and version.

For example, the following code shows how to configure a command job with the Python SDK, which uses a curated environment including Scikit-Learn:

PythonCopy

```
from azure.ai.ml import command

# configure job
job = command(
    code="./src",
    command="python train.py",
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",
    compute="aml-cluster",
    display_name="train-with-curated-environment",
    experiment_name="train-with-curated-environment"
```

```
)

# submit job
returned_job = ml_client.create_or_update(job)
```

Test and troubleshoot a curated environment

As curated environments allow for faster deployment time, it's a best practice to first explore whether one of the pre-created curated environments can be used to run your code.

You can verify that a curated environment includes all necessary packages by reviewing its details. Then, you can test by using the environment to run the script.

If an environment doesn't include all necessary packages to run your code, your job fails.

When a job fails, you can review the detailed error logs in the **Outputs + logs** tab of your job in the Azure Machine Learning studio.

A common error message that indicates your environment is incomplete, is `ModuleNotFoundError`. The module that isn't found is listed in the error message. By reviewing the error message, you can update the environment to include the libraries to ensure the necessary packages are installed on the compute target before running the code.

When you need to specify other necessary packages, you can use a curated environment as reference for your own custom environments by modifying the Dockerfiles that back these curated environments.

Create and use custom environments

Completed100 XP

- 12 minutes

When you need to create your own environment in Azure Machine Learning to list all necessary packages, libraries, and dependencies to run your scripts, you can create **custom environments**.

You can define an environment from a Docker image, a Docker build context, and a conda specification with Docker image.

Create a custom environment from a Docker image

The easiest approach is likely to be to create an environment from a Docker image. Docker images can be hosted in a public registry like Docker Hub or privately stored in an Azure Container registry.

Many open-source frameworks are encapsulated in public images to be found on Docker Hub. For example, you can find a public Docker image that contains all necessary packages to train a deep learning model with PyTorch.

To create an environment from a Docker image, you can use the Python SDK:

PythonCopy
```
from azure.ai.ml.entities import Environment

env_docker_image = Environment(
    image="pytorch/pytorch:latest",
    name="public-docker-image-example",
    description="Environment created from a public Docker image.",
)
ml_client.environments.create_or_update(env_docker_image)
```

You can also use the Azure Machine Learning base images to create an environment (which are similar to the images used by curated environments):

PythonCopy
```
from azure.ai.ml.entities import Environment

env_docker_image = Environment(
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    name="aml-docker-image-example",
    description="Environment created from a Azure ML Docker image.",
)
ml_client.environments.create_or_update(env_docker_image)
```

Create a custom environment with a conda specification file

Though Docker images contain all necessary packages when working with a specific framework, it may be that you need to include other packages to run your code.

For example, you may want to train a model with PyTorch, and track the model with MLflow.

When you need to include other packages or libraries in your environment, you can add a conda specification file to a Docker image when creating the environment.

A conda specification file is a YAML file, which lists the packages that need to be installed using `conda` or `pip`. Such a YAML file may look like:

ymlCopy

```yaml
name: basic-env-cpu
channels:
  - conda-forge
dependencies:
  - python=3.7
  - scikit-learn
  - pandas
  - numpy
  - matplotlib
```

Tip

Review the conda documentation on how to **create an environment manually** for information on the standard format for conda files.

To create an environment from a base Docker image and a conda specification file, you can use the following code:

PythonCopy

```python
from azure.ai.ml.entities import Environment

env_docker_conda = Environment(
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    conda_file="./conda-env.yml",
    name="docker-image-plus-conda-example",
    description="Environment created from a Docker image plus Conda environment.",
)
ml_client.environments.create_or_update(env_docker_conda)
```

Note

Since all curated environments are prefixed with **AzureML-**, you can't create an environment with the same prefix.

Use an environment

Most commonly, you use environments when you want to run a script as a (**command**) **job**.

To specify which environment you want to use to run your script, you reference an environment using the `<curated-environment-name>:<version>` or `<curated-environment-name>@latest` syntax.

For example, the following code shows how to configure a command job with the Python SDK, which uses a curated environment including Scikit-Learn:

PythonCopy

```python
from azure.ai.ml import command

# configure job
```

```
job = command(
    code="./src",
    command="python train.py",
    environment="docker-image-plus-conda-example:1",
    compute="aml-cluster",
    display_name="train-custom-env",
    experiment_name="train-custom-env"
)

# submit job
returned_job = ml_client.create_or_update(job)
```

When you submit the job, the environment is built. The first time you use an environment, it can take 10-15 minutes to build the environment. You can review the logs of the environment build in the logs of the job.

When Azure Machine Learning builds a new environment, it's added to the list of custom environments in the workspace. The image of the environment is hosted in the Azure Container registry associated to the workspace. Whenever you use the same environment for another job (and another script), the environment is ready to go and doesn't need to be build again.

1.

A data scientist created a script that trains a machine learning model using the open-source library scikit-learn. The data scientist wants to quickly test whether the script can run on the existing compute cluster, what type of environment should the data scientist use?

Default

Curated

Correct. Curated environments are ideal to use for faster development time.

Custom

2.

A command job fails with the error message that a module isn't found. The data scientist used a curated environment and wants to add a specific Python package to create a custom environment and successfully run the job. Which file should be created before creating the custom environment that uses a curated environment as reference?

Training script

Docker image

Conda specification

Correct. You can list Python packages in a conda specification file.

Preprocess data and configure featurization

Completed 100 XP

- 6 minutes

Before you can run an automated machine learning (AutoML) experiment, you need to prepare your data. When you want to train a classification model, you'll only need to provide the training data.

After you've collected the data, you need to create a **data asset** in Azure Machine Learning. In order for AutoML to understand how to read the data, you need to create a **MLTable** data asset that includes the schema of the data.

You can create a MLTable data asset when your data is stored in a folder together with a MLTable file. When you have created the data asset, you can specify it as input with the following code:

PythonCopy

```
from azure.ai.ml.constants import AssetTypes
from azure.ai.ml import Input

my_training_data_input = Input(type=AssetTypes.MLTABLE, path="azureml:input-data-automl:1")
```

Tip

Learn more about **how to create a MLTable data asset in Azure Machine Learning**.

Once you've created the data asset, you can configure the AutoML experiment. Before AutoML trains a classification model, preprocessing transformations can be applied to your data.

Understand scaling and normalization

AutoML applies scaling and normalization to numeric data automatically, helping prevent any large-scale features from dominating training. During an AutoML experiment, multiple scaling or normalization techniques will be applied.

Configure optional featurization

You can choose to have AutoML apply preprocessing transformations, such as:

- Missing value imputation to eliminate nulls in the training dataset.
- Categorical encoding to convert categorical features to numeric indicators.
- Dropping high-cardinality features, such as record IDs.
- Feature engineering (for example, deriving individual date parts from DateTime features)

By default, AutoML will perform featurization on your data. You can disable it if you don't want the data to be transformed.

If you do want to make use of the integrated featurization function, you can customize it. For example, you can specify which imputation method should be used for a specific feature.

After an AutoML experiment is completed, you'll be able to review which scaling and normalization methods have been applied. You'll also get notified if AutoML has detected any issues with the data, like whether there are missing values or class imbalance.

Run an Automated Machine Learning experiment

Completed100 XP

- 12 minutes

To run an automated machine learning (AutoML) experiment, you can configure and submit the job with the Python SDK.

The algorithms AutoML uses will depend on the task you specify. When you want to train a classification model, AutoML will choose from a list of classification algorithms:

- Logistic Regression
- Light Gradient Boosting Machine (GBM)
- Decision Tree

- Random Forest
- Naive Bayes
- Linear Support Vector Machine (SVM)
- XGBoost
- And others...

Tip

For a full list of supported algorithms, explore **the overview of supported algorithms**.

Restrict algorithm selection

By default, AutoML will randomly select from the full range of algorithms for the specified task. You can choose to block individual algorithms from being selected; which can be useful if you know that your data isn't suited to a particular type of algorithm. You also may want to block certain algorithms if you have to comply with a policy that restricts the type of machine learning algorithms you can use in your organization.

Configure an AutoML experiment

When you use the Python SDK (v2) to configure an AutoML experiment or job, you configure the experiment using the `automl` class. For classification, you'll use the `automl.classification` function as shown in the following example:

PythonCopy

```
from azure.ai.ml import automl

# configure the classification job
classification_job = automl.classification(
    compute="aml-cluster",
    experiment_name="auto-ml-class-dev",
    training_data=my_training_data_input,
    target_column_name="Diabetic",
    primary_metric="accuracy",
    n_cross_validations=5,
    enable_model_explainability=True
)
```

Note

AutoML needs a MLTable data asset as input. In the example, `my_training_data_input` refers to a MLTable data asset created in the Azure Machine Learning workspace.

Specify the primary metric

One of the most important settings you must specify is the **primary_metric**. The primary metric is the target performance metric for which the optimal model will be determined. Azure Machine Learning supports a set of named metrics for each type of task.

To retrieve the list of metrics available when you want to train a classification model, you can use the **ClassificationPrimaryMetrics** function as shown here:

PythonCopy

```
from azure.ai.ml.automl import ClassificationPrimaryMetrics

list(ClassificationPrimaryMetrics)
```

Tip

You can find a full list of primary metrics and their definitions in **evaluate automated machine learning experiment results**.

Set the limits

Training machine learning models will cost compute. To minimize costs and time spent on training, you can set limits to an AutoML experiment or job by using `set_limits()`.

There are several options to set limits to an AutoML experiment:

- `timeout_minutes`: Number of minutes after which the complete AutoML experiment is terminated.
- `trial_timeout_minutes`: Maximum number of minutes one trial can take.
- `max_trials`: Maximum number of trials, or models that will be trained.
- `enable_early_termination`: Whether to end the experiment if the score isn't improving in the short term.

PythonCopy

```
classification_job.set_limits(
    timeout_minutes=60,
    trial_timeout_minutes=20,
    max_trials=5,
    enable_early_termination=True,
)
```

To save time, you can also run multiple trials in parallel. When you use a compute cluster, you can have as many parallel trials as you have nodes. The maximum number of parallel trials is therefore related to the maximum number of nodes your compute cluster has. If you want to set the maximum number of parallel trials to be less than the maximum number of nodes, you can use `max_concurrent_trials`.

Set the training properties

AutoML will try various combinations of featurization and algorithms to train a machine learning model. If you already know that certain algorithms aren't well-suited for your data, you can exclude (or include) a subset of the available algorithms.

You can also choose whether you want to allow AutoML to use ensemble models.

Submit an AutoML experiment

You can submit an AutoML job with the following code:

PythonCopy

```
# submit the AutoML job
returned_job = ml_client.jobs.create_or_update(
    classification_job
)
```

You can monitor AutoML job runs in the Azure Machine Learning studio. To get a direct link to the AutoML job by running the following code:

PythonCopy

```
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

Set up AutoML training for tabular data with the Azure Machine Learning CLI and Python SDK

- Article
- 08/02/2023
- 40 contributors

Feedback

In this article

APPLIES TO: Azure CLI ml extension v2 (current) Python SDK azure-ai-ml v2 (current)

In this guide, learn how to set up an automated machine learning, AutoML, training job with the Azure Machine Learning Python SDK v2. Automated ML picks an algorithm and hyperparameters for you and generates a model ready for deployment. This guide provides details of the various options that you can use to configure automated ML experiments.

If you prefer a no-code experience, you can also Set up no-code AutoML training in the Azure Machine Learning studio.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin. Try the free or paid version of Azure Machine Learning.
- An Azure Machine Learning workspace. If you don't have one, you can use the steps in the Create resources to get started article.

- Python SDK
- Azure CLI

To use the **SDK** information, install the Azure Machine Learning SDK v2 for Python.

To install the SDK you can either,

- Create a compute instance, which already has installed the latest Azure Machine Learning Python SDK and is pre-configured for ML workflows. See Create an Azure Machine Learning compute instance for more information.
- Install the SDK on your local machine

Set up your workspace

To connect to a workspace, you need to provide a subscription, resource group and workspace name.

- Python SDK
- Azure CLI

The Workspace details are used in the `MLClient` from `azure.ai.ml` to get a handle to the required Azure Machine Learning workspace.

In the following example, the default Azure authentication is used along with the default workspace configuration or from any `config.json` file you might have copied into the folders structure. If no `config.json` is found, then you need to manually

introduce the subscription_id, resource_group and workspace when creating `MLClient`.

PythonCopy

```
from azure.identity import DefaultAzureCredential
from azure.ai.ml import MLClient

credential = DefaultAzureCredential()
ml_client = None
try:
    ml_client = MLClient.from_config(credential)
except Exception as ex:
    print(ex)
    # Enter details of your Azure Machine Learning workspace
    subscription_id = "<SUBSCRIPTION_ID>"
    resource_group = "<RESOURCE_GROUP>"
    workspace = "<AZUREML_WORKSPACE_NAME>"
    ml_client = MLClient(credential, subscription_id, resource_group, workspace)
```

Data source and format

In order to provide training data to AutoML in SDK v2 you need to upload it into the cloud through an **MLTable**.

Requirements for loading data into an MLTable:

- Data must be in tabular form.
- The value to predict, target column, must be in the data.

Training data must be accessible from the remote compute. Automated ML v2 (Python SDK and CLI/YAML) accepts MLTable data assets (v2), although for backwards compatibility it also supports v1 Tabular Datasets from v1 (a registered Tabular Dataset) through the same input dataset properties. However the recommendation is to use MLTable available in v2. In this example, we assume the data is stored at the local path, `./train_data/bank_marketing_train_data.csv`

- Python SDK
- Azure CLI

You can create an MLTable using the mltable Python SDK as in the following example:

PythonCopy

```
import mltable

paths = [
    {'file': './train_data/bank_marketing_train_data.csv'}
]
```

```
train_table = mltable.from_delimited_files(paths)
train_table.save('./train_data')
```

This code creates a new file, `./train_data/MLTable`, which contains the file format and loading instructions.

Now the `./train_data` folder has the MLTable definition file plus the data file, `bank_marketing_train_data.csv`.

For more information on MLTable, see the mltable how-to article

Training, validation, and test data

You can specify separate **training data and validation data sets**, however training data must be provided to the `training_data` parameter in the factory function of your automated ML job.

If you don't explicitly specify a `validation_data` or `n_cross_validation` parameter, automated ML applies default techniques to determine how validation is performed. This determination depends on the number of rows in the dataset assigned to your `training_data` parameter.

Expand table

Training data size	Validation technique
Larger than 20,000 rows	Train/validation data split is applied. The default is to take 10% of the initial training data set as the validation set. In turn, that validation set is used for metrics calculation.
Smaller than or equal to 20,000 rows	Cross-validation approach is applied. The default number of folds depends on the number of rows. **If the dataset is fewer than 1,000 rows**, 10 folds are used. **If the rows are equal to or between 1,000 and 20,000**, then three folds are used.

Compute to run experiment

Automated ML jobs with the Python SDK v2 (or CLI v2) are currently only supported on Azure Machine Learning remote compute (cluster or compute instance).

Learn more about creating compute with the Python SDKv2 (or CLIv2).

Configure your experiment settings

There are several options that you can use to configure your automated ML experiment. These configuration parameters are set in your task method. You can also set job training settings and exit criteria with the `training` and `limits` settings.

The following example shows the required parameters for a classification task that specifies accuracy as the primary metric and 5 cross-validation folds.

- Python SDK
- Azure CLI

PythonCopy

```
from azure.ai.ml.constants import AssetTypes
from azure.ai.ml import automl, Input

# note that this is a code snippet -- you might have to modify the variable values
to run it successfully

# make an Input object for the training data
my_training_data_input = Input(
    type=AssetTypes.MLTABLE, path="./data/training-mltable-folder"
)

# configure the classification job
classification_job = automl.classification(
    compute=my_compute_name,
    experiment_name=my_exp_name,
    training_data=my_training_data_input,
    target_column_name="y",
    primary_metric="accuracy",
    n_cross_validations=5,
    enable_model_explainability=True,
    tags={"my_custom_tag": "My custom value"}
)

# Limits are all optional
classification_job.set_limits(
    timeout_minutes=600,
    trial_timeout_minutes=20,
    max_trials=5,
    enable_early_termination=True,
)

# Training properties are optional
classification_job.set_training(
    blocked_training_algorithms=["logistic_regression"],
    enable_onnx_compatible_models=True
)
```

Select your machine learning task type (ML problem)

Before you can submit your automated ML job, you need to determine the kind of machine learning problem you're solving. This problem determines which function your automated ML job uses and what model algorithms it applies.

Automated ML supports tabular data based tasks (classification, regression, forecasting), computer vision tasks (such as Image Classification and Object Detection), and natural language processing tasks (such as Text classification and Entity Recognition tasks). See our article on task types for more information. See our time series forecasting guide for more details on setting up forecasting jobs.

Supported algorithms

Automated machine learning tries different models and algorithms during the automation and tuning process. As a user, you don't need to specify the algorithm.

The task method determines the list of algorithms/models, to apply. Use the `allowed_training_algorithms` or `blocked_training_algorithms` parameters in the `training` configuration of the AutoML job to further modify iterations with the available models to include or exclude.

In the following list of links you can explore the supported algorithms per machine learning task listed below.

Expand table

Classification	**Regression**	**Time Series Forecasting**
Logistic Regression*	Elastic Net*	AutoARIMA
Light GBM*	Light GBM*	Prophet
Gradient Boosting*	Gradient Boosting*	Elastic Net
Decision Tree*	Decision Tree*	Light GBM
K Nearest Neighbors*	K Nearest Neighbors*	K Nearest Neighbors
Linear SVC*	LARS Lasso*	Decision Tree
Support Vector Classification (SVC)*	Stochastic Gradient Descent (SGD)*	Arimax
Random Forest*	Random Forest	LARS Lasso
Extremely Randomized Trees*	Extremely Randomized Trees*	Extremely Randomized Trees*
Xgboost*	Xgboost*	Random Forest

Classification	Regression	Time Series Forecasting
Naive Bayes*	Xgboost	TCNForecaster
Stochastic Gradient Descent (SGD)*	Stochastic Gradient Descent (SGD)	Gradient Boosting
		ExponentialSmoothing
		SeasonalNaive
		Average
		Naive
		SeasonalAverage

With additional algorithms below.

- Image Classification Multi-class Algorithms
- Image Classification Multi-label Algorithms
- Image Object Detection Algorithms
- NLP Text Classification Multi-label Algorithms
- NLP Text Named Entity Recognition (NER) Algorithms

Follow this link for example notebooks of each task type.

Primary metric

The `primary_metric` parameter determines the metric to be used during model training for optimization. The available metrics you can select is determined by the task type you choose.

Choosing a primary metric for automated ML to optimize depends on many factors. We recommend your primary consideration be to choose a metric that best represents your business needs. Then consider if the metric is suitable for your dataset profile (data size, range, class distribution, etc.). The following sections summarize the recommended primary metrics based on task type and business scenario.

Learn about the specific definitions of these metrics in Understand automated machine learning results.

Metrics for classification multi-class scenarios

These metrics apply for all classification scenarios, including tabular data, images/computer-vision and NLP-Text.

Threshold-dependent metrics, like `accuracy`, `recall_score_weighted`, `norm_macro_recall`, and `precision_score_weighted` may not optimize as well for datasets that are small, have large class skew (class imbalance), or when the expected metric value is very close to 0.0 or 1.0. In those cases, `AUC_weighted` can be a better choice for the primary metric. After automated ML completes, you can choose the winning model based on the metric best suited to your business needs.

Expand table

Metric	**Example use case(s)**
`accuracy`	Image classification, Sentiment analysis, Churn prediction
`AUC_weighted`	Fraud detection, Image classification, Anomaly detection/spam detection
`average_precision_score_weighted`	Sentiment analysis
`norm_macro_recall`	Churn prediction
`precision_score_weighted`	

Metrics for classification multi-label scenarios

- For Text classification, multi-label currently 'Accuracy' is the only primary metric supported.
- For Image classification multi-label, the primary metrics supported are defined in the ClassificationMultilabelPrimaryMetrics Enum

Metrics for NLP Text NER (Named Entity Recognition) scenarios

- For NLP Text NER (Named Entity Recognition) currently 'Accuracy' is the only primary metric supported.

Metrics for regression scenarios

`r2_score`, `normalized_mean_absolute_error` and `normalized_root_mean_squared_error` are all trying to minimize prediction errors. `r2_score` and `normalized_root_mean_squared_error` are both minimizing average squared errors while `normalized_mean_absolute_error` is minimizing the average absolute value of errors. Absolute value treats errors at all magnitudes alike and squared errors will have a much larger penalty for errors with larger absolute values. Depending on whether larger errors should be punished more or not, one can choose to optimize squared error or absolute error.

The main difference between `r2_score` and `normalized_root_mean_squared_error` is the way they're normalized and their meanings. `normalized_root_mean_squared_error` is root mean squared error normalized by range and can be interpreted as the average error magnitude for prediction. `r2_score` is mean squared error normalized by an estimate of variance of data. It's the proportion of variation that can be captured by the model.

Note

`r2_score` and `normalized_root_mean_squared_error` also behave similarly as primary metrics. If a fixed validation set is applied, these two metrics are optimizing the same target, mean squared error, and will be optimized by the same model. When only a training set is available and cross-validation is applied, they would be slightly different as the normalizer for `normalized_root_mean_squared_error` is fixed as the range of training set, but the normalizer for `r2_score` would vary for every fold as it's the variance for each fold.

If the rank, instead of the exact value is of interest, `spearman_correlation` can be a better choice as it measures the rank correlation between real values and predictions.

AutoML does not currently support any primary metrics that measure *relative* difference between predictions and observations. The metrics `r2_score`, `normalized_mean_absolute_error`, and `normalized_root_mean_squared_error` are all measures of absolute difference. For example, if a prediction differs from an observation by 10 units, these metrics compute the same value if the observation is 20 units or 20,000 units. In contrast, a percentage difference, which is a relative measure, gives errors of 50% and 0.05%, respectively! To optimize for relative difference, you can run AutoML with a supported primary metric and then select the model with the best `mean_absolute_percentage_error` or `root_mean_squared_log_error`. Note that these metrics are undefined when any observation values are zero, so they may not always be good choices.

Expand table

Metric	Example use case(s)
`spearman_correlation`	
`normalized_root_mean_squared_error`	Price prediction (house/product/tip), Review score prediction
`r2_score`	Airline delay, Salary estimation, Bug resolution time
`normalized_mean_absolute_error`	

Metrics for Time Series Forecasting scenarios

The recommendations are similar to those noted for regression scenarios.

Expand table

Metric	Example use case(s)
`normalized_root_mean_squared_error`	Price prediction (forecasting), Inventory optimization, Demand forecasting
`r2_score`	Price prediction (forecasting), Inventory optimization, Demand forecasting
`normalized_mean_absolute_error`	

Metrics for Image Object Detection scenarios

- For Image Object Detection, the primary metrics supported are defined in the ObjectDetectionPrimaryMetrics Enum

Metrics for Image Instance Segmentation scenarios

- For Image Instance Segmentation scenarios, the primary metrics supported are defined in the InstanceSegmentationPrimaryMetrics Enum

Data featurization

In every automated ML experiment, your data is automatically transformed to numbers and vectors of numbers and also scaled and normalized to help algorithms that are sensitive to features that are on different scales. These data transformations are called *featurization*.

Note

Automated machine learning featurization steps (feature normalization, handling missing data, converting text to numeric, etc.) become part of the underlying model. When using the model for predictions, the same featurization steps applied during training are applied to your input data automatically.

When configuring your automated ML jobs, you can enable/disable the `featurization` settings.

The following table shows the accepted settings for featurization.

Expand table

Featurization Configuration	Description
`"mode": 'auto'`	Indicates that as part of preprocessing, data guardrails and featurization steps are performed automatically. **Default setting**.
`"mode": 'off'`	Indicates featurization step shouldn't be done automatically.
`"mode":'custom'`	Indicates customized featurization step should be used.

The following code shows how custom featurization can be provided in this case for a regression job.

- Python SDK
- Azure CLI

PythonCopy

```
from azure.ai.ml.automl import ColumnTransformer

transformer_params = {
    "imputer": [
        ColumnTransformer(fields=["CACH"], parameters={"strategy":
"most_frequent"}),
        ColumnTransformer(fields=["PRP"], parameters={"strategy":
"most_frequent"}),
    ],
}
regression_job.set_featurization(
    mode="custom",
    transformer_params=transformer_params,
    blocked_transformers=["LabelEncoding"],
    column_name_and_types={"CHMIN": "Categorical"},
)
```

Exit criteria

There are a few options you can define in the `set_limits()` function to end your experiment prior to job completion.

Expand table

Criteria	description
No criteria	If you don't define any exit parameters the experiment continues until no further progress is made on your primary metric.
`timeout`	Defines how long, in minutes, your experiment should continue to run. If not specified, the default job's total timeout is 6 days (8,640 minutes). To specify a timeout less than or equal to 1 hour (60 minutes), make sure your dataset's size isn't greater than 10,000,000 (rows times column) or an error results.

Criteria	description
	This timeout includes setup, featurization and training runs but doesn't include the ensembling and model explainability runs at the end of the process since those actions need to happen once all the trials (children jobs) are done.
`trial_timeout_minutes`	Maximum time in minutes that each trial (child job) can run for before it terminates. If not specified, a value of 1 month or 43200 minutes is used
`enable_early_termination`	Whether to end the job if the score is not improving in the short term
`max_trials`	The maximum number of trials/runs each with a different combination of algorithm and hyper-parameters to try during an AutoML job. If not specified, the default is 1000 trials. If using `enable_early_termination` the number of trials used can be smaller.
`max_concurrent_trials`	Represents the maximum number of trials (children jobs) that would be executed in parallel. It's a good practice to match this number with the number of nodes your cluster

Run experiment

Note

If you run an experiment with the same configuration settings and primary metric multiple times, you'll likely see variation in each experiments final metrics score and generated models. The algorithms automated ML employs have inherent randomness that can cause slight variation in the models output by the experiment and the recommended model's final metrics score, like accuracy. You'll likely also see results with the same model name, but different hyper-parameters used.

Warning

If you have set rules in firewall and/or Network Security Group over your workspace, verify that required permissions are given to inbound and outbound network traffic as defined in **Configure inbound and outbound network traffic**.

Submit the experiment to run and generate a model. With the `MLClient` created in the prerequisites, you can run the following command in the workspace.

- Python SDK
- Azure CLI

PythonCopy

```
# Submit the AutoML job
```

```
returned_job = ml_client.jobs.create_or_update(
    classification_job
)  # submit the job to the backend

print(f"Created job: {returned_job}")

# Get a URL for the status of the job
returned_job.services["Studio"].endpoint
```

Multiple child runs on clusters

Automated ML experiment child runs can be performed on a cluster that is already running another experiment. However, the timing depends on how many nodes the cluster has, and if those nodes are available to run a different experiment.

Each node in the cluster acts as an individual virtual machine (VM) that can accomplish a single training run; for automated ML this means a child run. If all the nodes are busy, a new experiment is queued. But if there are free nodes, the new experiment will run automated ML child runs in parallel in the available nodes/VMs.

To help manage child runs and when they can be performed, we recommend you create a dedicated cluster per experiment, and match the number of `max_concurrent_iterations` of your experiment to the number of nodes in the cluster. This way, you use all the nodes of the cluster at the same time with the number of concurrent child runs/iterations you want.

Configure `max_concurrent_iterations` in the `limits` configuration. If it is not configured, then by default only one concurrent child run/iteration is allowed per experiment. In case of compute instance, `max_concurrent_trials` can be set to be the same as number of cores on the compute instance VM.

Explore models and metrics

Automated ML offers options for you to monitor and evaluate your training results.

- For definitions and examples of the performance charts and metrics provided for each run, see Evaluate automated machine learning experiment results.
- To get a featurization summary and understand what features were added to a particular model, see Featurization transparency.

From Azure Machine Learning UI at the model's page you can also view the hyper-parameters used when training a particular model and also view and customize the internal model's training code used.

Register and deploy models

After you test a model and confirm you want to use it in production, you can register it for later use.

Tip

For registered models, one-click deployment is available via the **Azure Machine Learning studio**. See **how to deploy registered models from the studio**.

AutoML in pipelines

To leverage AutoML in your MLOps workflows, you can add AutoML Job steps to your Azure Machine Learning Pipelines. This allows you to automate your entire workflow by hooking up your data prep scripts to AutoML and then registering and validating the resulting best model.

Below is a sample pipeline with an AutoML classification component and a command component that shows the resulting AutoML output. Note how the inputs (training & validation data) and the outputs (best model) are referenced in different steps.

- Python SDK
- Azure CLI

PythonCopy

```
# Define pipeline
@pipeline(
    description="AutoML Classification Pipeline",
    )
def automl_classification(
    classification_train_data,
    classification_validation_data
):
    # define the automl classification task with automl function
    classification_node = classification(
        training_data=classification_train_data,
        validation_data=classification_validation_data,
        target_column_name="y",
        primary_metric="accuracy",
        # currently need to specify outputs "mlflow_model" explictly to reference
it in following nodes
        outputs={"best_model": Output(type="mlflow_model")},
    )
    # set limits and training
    classification_node.set_limits(max_trials=1)
    classification_node.set_training(
        enable_stack_ensemble=False,
        enable_vote_ensemble=False
    )
```

```
    command_func = command(
        inputs=dict(
            automl_output=Input(type="mlflow_model")
        ),
        command="ls ${{inputs.automl_output}}",
        environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu:latest"
    )
    show_output =
command_func(automl_output=classification_node.outputs.best_model)


pipeline_job = automl_classification(
    classification_train_data=Input(path="./training-mltable-folder/",
type="mltable"),
    classification_validation_data=Input(path="./validation-mltable-folder/",
type="mltable"),
)

# set pipeline level compute
pipeline_job.settings.default_compute = compute_name

# submit the pipeline job
returned_pipeline_job = ml_client.jobs.create_or_update(
    pipeline_job,
    experiment_name=experiment_name
)
returned_pipeline_job

# ...
# Note that this is a snippet from the bankmarketing example you can find in our
examples repo -> https://github.com/Azure/azureml-
examples/tree/main/sdk/python/jobs/pipelines/1h_automl_in_pipeline/automl-
classification-bankmarketing-in-pipeline
```

For more examples on how to include AutoML in your pipelines, please check out our examples repo.

AutoML at scale: distributed training

For large data scenarios, AutoML supports distributed training for a limited set of models:

Expand table

Distributed algorithm	Supported tasks	Data size limit (approximate)
LightGBM	Classification, regression	1TB
TCNForecaster	Forecasting	200GB

Distributed training algorithms automatically partition and distribute your data across multiple compute nodes for model training.

Note

Cross-validation, ensemble models, ONNX support, and code generation are not currently supported in the distributed training mode. Also, AutoML may make choices such as restricting available featurizers and sub-sampling data used for validation, explainability and model evaluation.

Distributed training for classification and regression

To use distributed training for classification or regression, you need to set the `training_mode` and `max_nodes` properties of the job object.

Expand table

Property	Description
training_mode	Indicates training mode; `distributed` or `non_distributed`. Defaults to `non_distributed`.
max_nodes	The number of nodes to use for training by each AutoML trial. This setting must be greater than or equal to 4.

The following code sample shows an example of these settings for a classification job:

- Python SDK
- Azure CLI

PythonCopy

```
from azure.ai.ml.constants import TabularTrainingMode

# Set the training mode to distributed
classification_job.set_training(
    allowed_training_algorithms=["LightGBM"],
    training_mode=TabularTrainingMode.DISTRIBUTED
)

# Distribute training across 4 nodes for each trial
classification_job.set_limits(
    max_nodes=4,
    # other limit settings
)
```

Note

Distributed training for classification and regression tasks does not currently support multiple concurrent trials. Model trials execute sequentially with each trial using `max_nodes` nodes. The `max_concurrent_trials` limit setting is currently ignored.

Distributed training for forecasting

To learn how distributed training works for forecasting tasks, see our forecasting at scale article. To use distributed training for forecasting, you need to set the `training_mode`, `enable_dnn_training`, `max_nodes`, and optionally the `max_concurrent_trials` properties of the job object.

Expand table

Property	Description
training_mode	Indicates training mode; `distributed` or `non_distributed`. Defaults to `non_distributed`.
enable_dnn_training	Flag to enable deep neural network models.
max_concurrent_trials	This is the maximum number of trial models to train in parallel. Defaults to 1.
max_nodes	The total number of nodes to use for training. This setting must be greater than or equal to 2. For forecasting tasks, each trial model is trained using max(2,floor(max_nodes/max_concurrent_trials)) nodes.

The following code sample shows an example of these settings for a forecasting job:

- Python SDK
- Azure CLI

PythonCopy

```python
from azure.ai.ml.constants import TabularTrainingMode

# Set the training mode to distributed
forecasting_job.set_training(
    enable_dnn_training=True,
    allowed_training_algorithms=["TCNForecaster"],
    training_mode=TabularTrainingMode.DISTRIBUTED
)

# Distribute training across 4 nodes
# Train 2 trial models in parallel => 2 nodes per trial
forecasting_job.set_limits(
    max_concurrent_trials=2,
    max_nodes=4,
    # other limit settings
)
```

See previous sections on configuration and job submission for samples of full configuration code.

Evaluate and compare models

Completed 100 XP

- 3 minutes

When an automated machine learning (AutoML) experiment has completed, you'll want to review the models that have been trained and decide which one performed best.

In the Azure Machine Learning studio, you can select an AutoML experiment to explore its details.

On the **Overview** page of the AutoML experiment run, you can review the input data asset and the summary of the best model. To explore all models that have been trained, you can select the **Models** tab:

musing_cloud_7wwysnx

Overview Data guardrails

Refresh Edit and sub

Search

Showing 1-5 of 5 models

Algorithm name

VotingEnsemble

Explore preprocessing steps

When you've enabled featurization for your AutoML experiment, data guardrails will automatically be applied too. The three data guardrails that are supported for classification models are:

- Class balancing detection.
- Missing feature values imputation.
- High cardinality feature detection.

Each of these data guardrails will show one of three possible states:

- **Passed**: No problems were detected and no action is required.
- **Done**: Changes were applied to your data. You should review the changes AutoML has made to your data.
- **Alerted**: An issue was detected but couldn't be fixed. You should review the data to fix the issue.

Next to data guardrails, AutoML can apply scaling and normalization techniques to each model that is trained. You can review the technique applied in the list of models under **Algorithm name**.

For example, the algorithm name of a model listed may be `MaxAbsScaler, LightGBM`. `MaxAbsScaler` refers to a scaling technique where each feature is scaled by its maximum absolute value. `LightGBM` refers to the classification algorithm used to train the model.

Retrieve the best run and its model

When you're reviewing the models in AutoML, you can easily identify the best run based on the primary metric you specified. In the Azure Machine Learning studio, the models are automatically sorted to show the best performing model at the top.

In the **Models** tab of the AutoML experiment, you can **edit the columns** if you want to show other metrics in the same overview. By creating a more comprehensive overview that includes various metrics, it may be easier to compare models.

To explore a model even further, you can generate explanations for each model that has been trained. When configuring an AutoML experiment, you can specify that explanations should be generated for the best performing model. If however, you're interested in the interpretability of another model, you can select the model in the overview and select **Explain model**.

Note

Explaining a model is an approximation to the model's interpretability. Specifically, explanations will estimate the relative importance of features on the target feature (what the model is trained to predict). Learn more about **model interpretability**.

Tip

Learn more about **how to evaluate AutoML runs**.

Next unit: Exercise - Find the best classification model with Automated Machine Learning

1.

A data scientist wants to use automated machine learning to find the model with the best AUC_weighted metric. Which parameter of the classification function should be configured?

```
task='AUC_weighted'
```

```
target_column_name='AUC_weighted'
```

```
primary_metric='AUC_weighted'
```

Correct. Set the primary metric to the performance score for which you want to optimize the model.

2.

A data scientist has preprocessed the training data and wants to use automated machine learning to quickly iterate through various algorithms. The data shouldn't be changed. What should be the featurization mode to train a model without letting automated machine learning make changes to the data?

```
auto
```

```
custom
```

`off`

Correct. If you don't want the data to be preprocessed at all, set featurization to be off.

Configure MLflow for model tracking in notebooks

Completed 100 XP

- 6 minutes

As a data scientist, you'll want to develop a model in a notebook as it allows you to quickly test and run code.

Anytime you train a model, you want the results to be reproducible. By tracking and logging your work, you can review your work at any time and decide what the best approach is to train a model.

MLflow is an open-source library for tracking and managing your machine learning experiments. In particular, **MLflow Tracking** is a component of MLflow that logs everything about the model you're training, such as **parameters**, **metrics**, and **artifacts**.

To use MLflow in notebooks in the Azure Machine Learning workspace, you'll need to install the necessary libraries and set Azure Machine Learning as the tracking store. When you've configured MLflow, you can start to use MLflow when training models in notebooks.

Configure MLflow in notebooks

You can create and edit notebooks within Azure Machine Learning or on a local device.

Use Azure Machine Learning notebooks

Within the Azure Machine Learning workspace, you can create notebooks and connect the notebooks to an Azure Machine Learning managed **compute instance**.

When you're running a notebook on a compute instance, MLflow is already configured, and ready to be used.

To verify that the necessary packages are installed, you can run the following code:

PythonCopy

```
pip show mlflow
pip show azureml-mlflow
```

The `mlflow` package is the open-source library. The `azureml-mlflow` package contains the integration code of Azure Machine Learning with MLflow.

Use MLflow on a local device

When you prefer working in notebooks on a local device, you can also make use of MLflow. You'll need to configure MLflow by completing the following steps:

1. Install the `mlflow` and `azureml-mlflow` package.

   PythonCopy

   ```
   pip install mlflow
   pip install azureml-mlflow
   ```

2. Navigate to the Azure Machine Learning studio.
3. Select the name of the workspace you're working on in the top right corner of the studio.
4. Select **View all properties in Azure portal**. A new tab will open to take you to the Azure Machine Learning service in the Azure portal.
5. Copy the value of the **MLflow tracking URI**.

6. Use the following code in your local notebook to configure MLflow to point to the Azure Machine Learning workspace, and set it to the workspace tracking URI.

   PythonCopy

   ```
   mlflow.set_tracking_uri = "MLFLOW-TRACKING-URI"
   ```

Tip

Learn about alternative approaches to **set up the tracking environment when working on a local device**. For example, you can also use the Azure Machine Learning SDK v2 for Python, together with the workspace configuration file, to set the tracking URI.

When you've configured MLflow to track your model's results and store it in your Azure Machine Learning workspace, you're ready to experiment in a notebook.

Train and track models in notebooks

Completed 100 XP

- 10 minutes

As a data scientist, you use notebooks to experiment and train models. To group model training results, you'll use **experiments**. To track model metrics with MLflow when training a model in a notebook, you can use MLflow's logging capabilities.

Create an MLflow experiment

You can create a MLflow experiment, which allows you to group runs. If you don't create an experiment, MLflow will assume the default experiment with name `Default`.

To create an experiment, run the following command in a notebook:

PythonCopy

```
import mlflow

mlflow.set_experiment(experiment_name="heart-condition-classifier")
```

Log results with MLflow

Now, you're ready to train your model. To start a run tracked by MLflow, you'll use `start_run()`. Next, to track the model, you can:

- Enable **autologging**.
- Use **custom logging**.

Enable autologging

MLflow supports automatic logging for popular machine learning libraries. If you're using a library that is supported by autolog, then MLflow tells the framework you're using to log all the metrics, parameters, artifacts, and models that the framework considers relevant.

You can turn on autologging by using the `autolog` method for the framework you're using. For example, to enable autologging for XGBoost models you can use `mlflow.xgboost.autolog()`.

Tip

Find a list of **all supported frameworks for autologging in the official MLflow documentation**.

A notebook cell that trains and tracks a classification model using autologging may be similar to the following code example:

PythonCopy

```
from xgboost import XGBClassifier

with mlflow.start_run():
    mlflow.xgboost.autolog()

    model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
    model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)
```

As soon as `mlflow.xgboost.autolog()` is called, MLflow will start a run within an experiment in Azure Machine Learning to start tracking the experiment's run.

When the job has completed, you can review all logged metrics in the studio.

lemon_fox_4ts8znd9 Completed

Overview | Metrics | Images | Child jobs | Outputs + logs | Code

Refresh | Connect to compute | Resubmit | Register model | Cancel | Delete

Properties

Status
Completed

Created on
Nov 11, 2022 2:33 PM

Start time
Nov 11, 2022 2:33 PM

Duration
7.093s

Compute duration
7.093s

Compute target
None

Name
6bd70f61-2dc1-4234-bdac-d91fa8a8e8d7

Script name
--

Created by

Experiment
heart-condition-classifier

Arguments
None

Registered models
None

See all properties
Raw JSON

Use custom logging

Additionally, you can manually log your model with MLflow. Manually logging models is helpful when you want to log supplementary or custom information that isn't logged through autologging.

Note

You can choose to only use custom logging, or use custom logging in combination with autologging.

Common functions used with custom logging are:

- `mlflow.log_param()`: Logs a single key-value parameter. Use this function for an input parameter you want to log.
- `mlflow.log_metric()`: Logs a single key-value metric. Value must be a number. Use this function for any output you want to store with the run.
- `mlflow.log_artifact()`: Logs a file. Use this function for any plot you want to log, save as image file first.
- `mlflow.log_model()`: Logs a model. Use this function to create an MLflow model, which may include a custom signature, environment, and input examples.

Tip

Learn more about how to track models with MLflow by exploring the **official MLflow documentation**, or the **Azure Machine Learning documentation**

To use custom logging in a notebook, start a run and log any metric you want:

PythonCopy

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

with mlflow.start_run():
    model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
    model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    mlflow.log_metric("accuracy", accuracy)
```

Custom logging gives you more flexibility, but also creates more work as you'll have to define any parameter, metric, or artifact you want to log.

When the job has completed, you can review all logged metrics in the studio.

magenta_fig_5s723kbg Completed

Overview Metrics Images Child jobs Outputs + logs

Refresh Connect to compute Resubmit Register m

Properties

Status
Completed

Created on
Nov 11, 2022 2:40 PM

Start time
Nov 11, 2022 2:40 PM

Duration
0.7740s

Compute duration
0.7740s

Compute target
None

Name
262ad53e-e440-4294-aa3c-1d3a204e063e

Script name
--

Created by

Experiment
heart-cond

Arguments
None

Registered m
None

See all prope

Raw JSO