Quickstart: Get started with Azure Machine Learning

Article • 10/04/2023

APPLIES TO: Python SDK azure-ai-ml **v2 (current)**

This tutorial is an introduction to some of the most used features of the Azure Machine Learning service. In it, you will create, register and deploy a model. This tutorial will help you become familiar with the core concepts of Azure Machine Learning and their most common usage.

You'll learn how to run a training job on a scalable compute resource, then deploy it, and finally test the deployment.

You'll create a training script to handle the data preparation, train and register a model. Once you train the model, you'll *deploy* it as an *endpoint*, then call the endpoint for *inferencing*.

The steps you'll take are:

- ✓ Set up a handle to your Azure Machine Learning workspace
- ✓ Create your training script
- ✓ Create a scalable compute resource, a compute cluster
- ✓ Create and run a command job that will run the training script on the compute cluster, configured with the appropriate job environment
- ✓ View the output of your training script
- ✓ Deploy the newly-trained model as an endpoint
- ✓ Call the Azure Machine Learning endpoint for inferencing

Watch this video for an overview of the steps in this quickstart.
https://www.microsoft.com/en-us/videoplayer/embed/RW14vFs?postJsllMsg=true

Prerequisites

1. To use Azure Machine Learning, you'll first need a workspace. If you don't have one, complete Create resources you need to get started to create a workspace and learn more about using it.

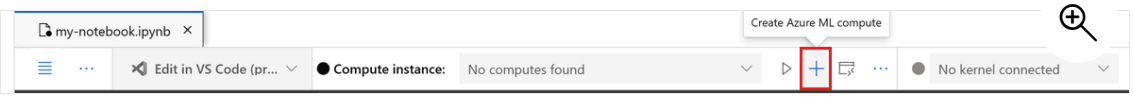
2. Sign in to studio and select your workspace if it's not already open.

3. Open or create a notebook in your workspace:

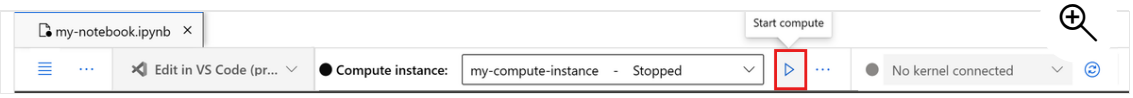
- Create a new notebook, if you want to copy/paste code into cells.
- Or, open **tutorials/get-started-notebooks/quickstart.ipynb** from the **Samples** section of studio. Then select **Clone** to add the notebook to your **Files**. (See where to find **Samples**.)

Set your kernel

1. On the top bar above your opened notebook, create a compute instance if you don't already have one.



2. If the compute instance is stopped, select **Start compute** and wait until it is running.



3. Make sure that the kernel, found on the top right, is `Python 3.10 - SDK v2`. If not, use the dropdown to select this kernel.

4. If you see a banner that says you need to be authenticated, select **Authenticate**.

> **Important**
>
> The rest of this tutorial contains cells of the tutorial notebook. Copy/paste them into your new notebook, or switch to the notebook now if you cloned it.

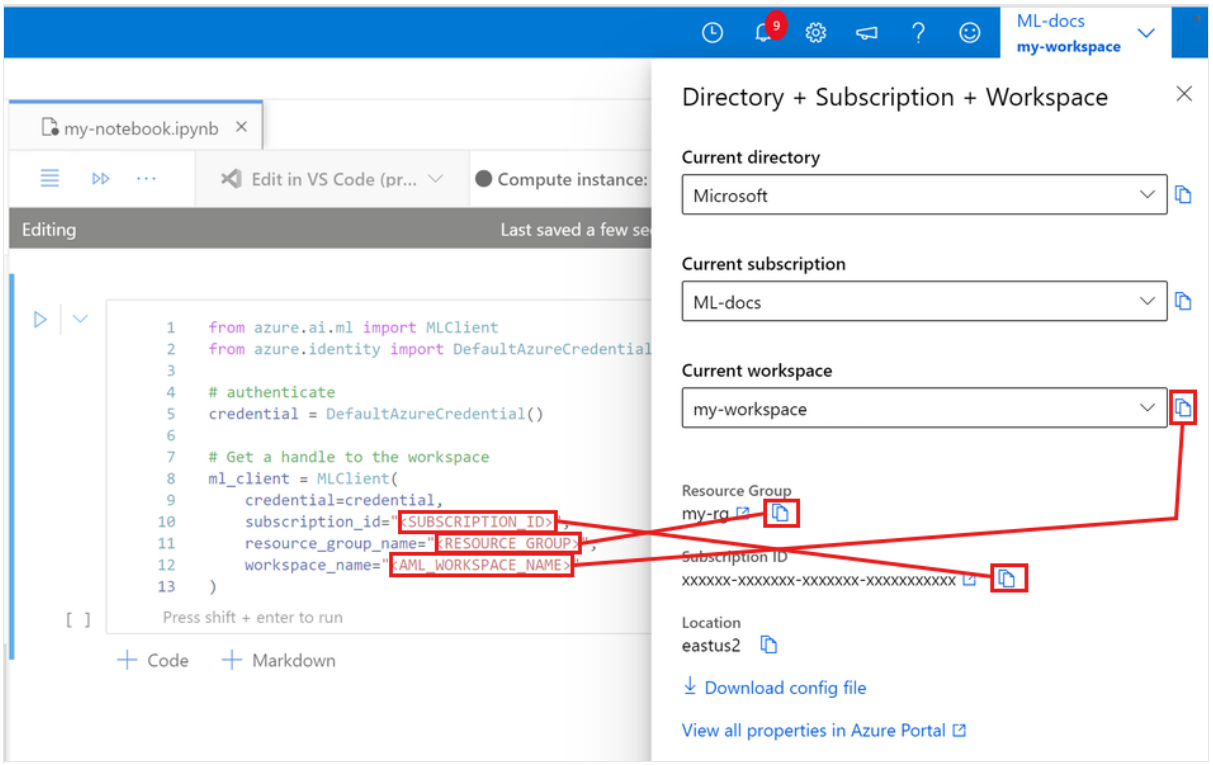
Create handle to workspace

Before we dive in the code, you need a way to reference your workspace. The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning.

You'll create `ml_client` for a handle to the workspace. You'll then use `ml_client` to manage resources and jobs.

In the next cell, enter your Subscription ID, Resource Group name and Workspace name. To find these values:

1. In the upper right Azure Machine Learning studio toolbar, select your workspace name.
2. Copy the value for workspace, resource group and subscription ID into the code.
3. You'll need to copy one value, close the area and paste, then come back for the next one.



Python

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

# authenticate
credential = DefaultAzureCredential()

# Get a handle to the workspace
ml_client = MLClient(
    credential=credential,
    subscription_id="<SUBSCRIPTION_ID>",
    resource_group_name="<RESOURCE_GROUP>",
    workspace_name="<AML_WORKSPACE_NAME>",
)
```

> **Note**

Creating MLClient will not connect to the workspace. The client initialization is lazy, it will wait for the first time it needs to make a call (in this notebook, that will happen in the cell that creates the compute cluster).

Create training script

Let's start by creating the training script - the *main.py* Python file.

First create a source folder for the script:

Python

```
import os

train_src_dir = "./src"
os.makedirs(train_src_dir, exist_ok=True)
```

This script handles the preprocessing of the data, splitting it into test and train data. It then consumes this data to train a tree based model and return the output model.

MLFlow will be used to log the parameters and metrics during our pipeline run.

The cell below uses IPython magic to write the training script into the directory you just created.

Python

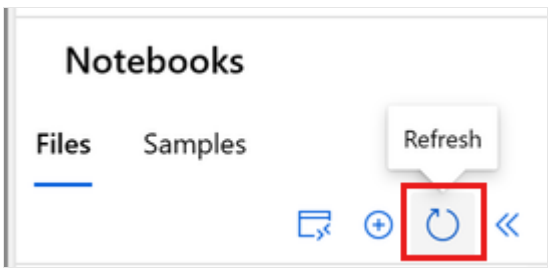
```
%%writefile {train_src_dir}/main.py
import os
import argparse
import pandas as pd
import mlflow
import mlflow.sklearn
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

def main():
    """Main function of the script."""

    # input and output arguments
    parser = argparse.ArgumentParser()
    parser.add_argument("--data", type=str, help="path to input
data")
    parser.add_argument("--test_train_ratio", type=float, re-
quired=False, default=0.25)
    parser.add_argument("--n_estimators", required=False, de-
```

```
fault=100, type=int)
    parser.add_argument("--learning_rate", required=False, de-
fault=0.1, type=float)
    parser.add_argument("--registered_model_name", type=str,
help="model name")
    args = parser.parse_args()

    # Start Logging
    mlflow.start_run()

    # enable autologging
    mlflow.sklearn.autolog()

    ####################
    #<prepare the data>
    ####################
    print(" ".join(f"{k}={v}" for k, v in vars(args).items()))

    print("input data:", args.data)

    credit_df = pd.read_csv(args.data, header=1, index_col=0)

    mlflow.log_metric("num_samples", credit_df.shape[0])
    mlflow.log_metric("num_features", credit_df.shape[1] - 1)

    train_df, test_df = train_test_split(
        credit_df,
        test_size=args.test_train_ratio,
    )
    ####################
    #</prepare the data>
    ####################

    ##################
    #<train the model>
    ##################
    # Extracting the label column
    y_train = train_df.pop("default payment next month")

    # convert the dataframe values to array
    X_train = train_df.values

    # Extracting the label column
    y_test = test_df.pop("default payment next month")

    # convert the dataframe values to array
    X_test = test_df.values

    print(f"Training with data of shape {X_train.shape}")

    clf = GradientBoostingClassifier(
        n_estimators=args.n_estimators, learning_rate=args.learn-
ing_rate
    )
```

```
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    print(classification_report(y_test, y_pred))
    ###################
    #</train the model>
    ###################

    ##########################
    #<save and register model>
    ##########################
    # Registering the model to the workspace
    print("Registering the model via MLFlow")
    mlflow.sklearn.log_model(
        sk_model=clf,
        registered_model_name=args.registered_model_name,
        artifact_path=args.registered_model_name,
    )

    # Saving the model to a file
    mlflow.sklearn.save_model(
        sk_model=clf,
        path=os.path.join(args.registered_model_name,
"trained_model"),
    )
    ###########################
    #</save and register model>
    ###########################

    # Stop Logging
    mlflow.end_run()

if __name__ == "__main__":
    main()
```

As you can see in this script, once the model is trained, the model file is saved and registered to the workspace. Now you can use the registered model in inferencing endpoints.

You might need to select **Refresh** to see the new folder and script in your **Files**.



Create a compute cluster, a scalable way to run a training job

> **Note**
>
> To try serverless compute (preview), skip this step and proceed to configure the command.

You already have a compute instance, which you're using to run the notebook. Now you'll add a second type of compute, a **compute cluster** that you'll use to run your training job. While a compute instance is a single node machine, a compute cluster can be single or multi-node machines with Linux or Windows OS, or a specific compute fabric like Spark.

You'll provision a Linux compute cluster. See the full list on VM sizes and prices .

For this example, you only need a basic cluster, so you'll use a Standard_DS3_v2 model with 2 vCPU cores, 7-GB RAM.

Python

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

# Name assigned to the compute cluster
cpu_compute_target = "cpu-cluster"

try:
    # let's see if the compute target already exists
    cpu_cluster = ml_client.compute.get(cpu_compute_target)
    print(
        f"You already have a cluster named {cpu_compute_target},
we'll reuse it as is."
    )

except Exception:
    print("Creating a new cpu compute target...")

    # Let's create the Azure Machine Learning compute object with
the intended parameters
    # if you run into an out of quota error, change the size to a
comparable VM that is available.\
    # Learn more on https://azure.microsoft.com/en-us/pricing
/details/machine-learning/.
    cpu_cluster = AmlCompute(
        name=cpu_compute_target,
        # Azure Machine Learning Compute is the on-demand VM ser-
```

```
vice
        type="amlcompute",
        # VM Family
        size="STANDARD_DS3_V2",
        # Minimum running nodes when there is no job running
        min_instances=0,
        # Nodes in cluster
        max_instances=4,
        # How many seconds will the node running after the job ter-
mination
        idle_time_before_scale_down=180,
        # Dedicated or LowPriority. The latter is cheaper but there
is a chance of job termination
        tier="Dedicated",
    )
    print(
        f"AMLCompute with name {cpu_cluster.name} will be created,
with compute size {cpu_cluster.size}"
    )
    # Now, we pass the object to MLClient's create_or_update method
    cpu_cluster = ml_client.compute.begin_create_or_up-
date(cpu_cluster)
```

Configure the command

Now that you have a script that can perform the desired tasks, and a compute cluster to run the script, you'll use a general purpose **command** that can run command line actions. This command line action can directly call system commands or run a script.

Here, you'll create input variables to specify the input data, split ratio, learning rate and registered model name. The command script will:

- Use the compute cluster to run the command.
- Use an *environment* that defines software and runtime libraries needed for the training script. Azure Machine Learning provides many curated or ready-made environments, which are useful for common training and inference scenarios. You'll use one of those environments here. In the Train a model tutorial, you'll learn how to create a custom environment.
- Configure the command line action itself - `python main.py` in this case. The inputs/outputs are accessible in the command via the `${{ ... }}` notation.
- In this sample, we access the data from a file on the internet.

> **Note**
>
> To use serverless compute (preview), delete `compute="cpu-cluster"` in this

code. Serverless is the simplest way to run jobs on AzureML.

Python

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

registered_model_name = "credit_defaults_model"

job = command(
    inputs=dict(
        data=Input(
            type="uri_file",
            path="https://azuremlexamples.blob.core.windows.net
/datasets/credit_card/default_of_credit_card_clients.csv",
        ),
        test_train_ratio=0.2,
        learning_rate=0.25,
        registered_model_name=registered_model_name,
    ),
    code="./src/",  # location of source code
    command="python main.py --data ${{inputs.data}}
--test_train_ratio ${{inputs.test_train_ratio}} --learning_rate
${{inputs.learning_rate}} --registered_model_name ${{inputs.regis-
tered_model_name}}",
    environment="AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest",
    compute="cpu-cluster", #delete this line to use serverless com-
pute
    display_name="credit_default_prediction",
)
```

Submit the job

It's now time to submit the job to run in Azure Machine Learning. This time you'll use `create_or_update` on `ml_client`.

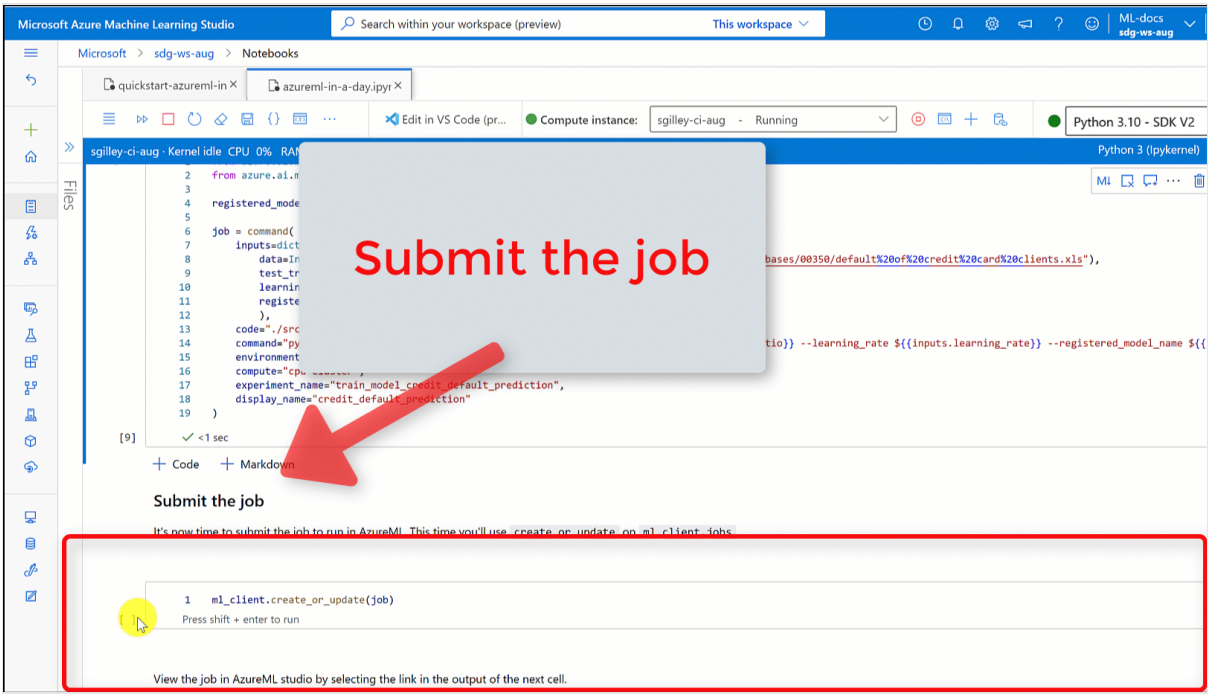
Python

```
ml_client.create_or_update(job)
```

View job output and wait for job completion

View the job in Azure Machine Learning studio by selecting the link in the output of the previous cell.

The output of this job will look like this in the Azure Machine Learning studio. Explore the tabs for various details like metrics, outputs etc. Once completed, the job will register a model in your workspace as a result of training.



> **Important**
>
> Wait until the status of the job is complete before returning to this notebook to continue. The job will take 2 to 3 minutes to run. It could take longer (up to 10 minutes) if the compute cluster has been scaled down to zero nodes and custom environment is still building.

Deploy the model as an online endpoint

Now deploy your machine learning model as a web service in the Azure cloud, an online endpoint.

To deploy a machine learning service, you'll use the model you registered.

Create a new online endpoint

Now that you have a registered model, it's time to create your online endpoint. The endpoint name needs to be unique in the entire Azure region. For this tutorial, you'll create a unique name using UUID .

Python

```
import uuid

# Creating a unique name for the endpoint
online_endpoint_name = "credit-endpoint-" + str(uuid.uuid4())[:8]
```

Create the endpoint:

Python

```
# Expect the endpoint creation to take a few minutes
from azure.ai.ml.entities import (
    ManagedOnlineEndpoint,
    ManagedOnlineDeployment,
    Model,
    Environment,
)

# create an online endpoint
endpoint = ManagedOnlineEndpoint(
    name=online_endpoint_name,
    description="this is an online endpoint",
    auth_mode="key",
    tags={
        "training_dataset": "credit_defaults",
        "model_type": "sklearn.GradientBoostingClassifier",
    },
)

endpoint = ml_client.online_endpoints.begin_create_or_update(end-
point).result()

print(f"Endpoint {endpoint.name} provisioning state: {endpoint.pro-
visioning_state}")
```

> **Note**
>
> Expect the endpoint creation to take a few minutes.

Once the endpoint has been created, you can retrieve it as below:

Python

```
endpoint = ml_client.online_endpoints.get(name=online_end-
point_name)

print(
    f'Endpoint "{endpoint.name}" with provisioning state "{end-
```

```
point.provisioning_state}" is retrieved'
)
```

Deploy the model to the endpoint

Once the endpoint is created, deploy the model with the entry script. Each endpoint can have multiple deployments. Direct traffic to these deployments can be specified using rules. Here you'll create a single deployment that handles 100% of the incoming traffic. We have chosen a color name for the deployment, for example, *blue*, *green*, *red* deployments, which is arbitrary.

You can check the **Models** page on Azure Machine Learning studio, to identify the latest version of your registered model. Alternatively, the code below will retrieve the latest version number for you to use.

Python

```
# Let's pick the latest version of the model
latest_model_version = max(
    [int(m.version) for m in ml_client.models.list(name=regis-
tered_model_name)]
)
print(f'Latest model is version "{latest_model_version}" ')
```

Deploy the latest version of the model.

Python

```
# picking the model to deploy. Here we use the latest version of
our registered model
model = ml_client.models.get(name=registered_model_name, ver-
sion=latest_model_version)

# Expect this deployment to take approximately 6 to 8 minutes.
# create an online deployment.
# if you run into an out of quota error, change the instance_type
to a comparable VM that is available.\
# Learn more on https://azure.microsoft.com/en-us/pricing/details
/machine-learning/.

blue_deployment = ManagedOnlineDeployment(
    name="blue",
    endpoint_name=online_endpoint_name,
    model=model,
    instance_type="Standard_DS3_v2",
    instance_count=1,
)
```

```
blue_deployment = ml_client.begin_create_or_update(blue_deploy-
ment).result()
```

> **Note**
>
> Expect this deployment to take approximately 6 to 8 minutes.

When the deployment is done, you're ready to test it.

Test with a sample query

Once the model is deployed to the endpoint, you can run inference with it.

Create a sample request file following the design expected in the run method in the score script.

Python

```
deploy_dir = "./deploy"
os.makedirs(deploy_dir, exist_ok=True)
```

Python

```
%%writefile {deploy_dir}/sample-request.json
{
  "input_data": {
    "columns":
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22],
    "index": [0, 1],
    "data": [
            [20000,2,2,1,24,2,2,-1,-1,-
2,-2,3913,3102,689,0,0,0,0,689,0,0,0,0],
            [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 10, 9, 8, 7, 6, 5, 4,
3, 2, 1, 10, 9, 8]
        ]
  }
}
```

Python

```
# test the blue deployment with some sample data
ml_client.online_endpoints.invoke(
    endpoint_name=online_endpoint_name,
    request_file="./deploy/sample-request.json",
```

```
    deployment_name="blue",
)
```

Clean up resources

If you're not going to use the endpoint, delete it to stop using the resource. Make sure no other deployments are using an endpoint before you delete it.

> **Note**
>
> Expect the complete deletion to take approximately 20 minutes.

Python

```
ml_client.online_endpoints.begin_delete(name=online_endpoint_name)
```

Stop compute instance

If you're not going to use it now, stop the compute instance:

1. In the studio, in the left navigation area, select **Compute**.
2. In the top tabs, select **Compute instances**
3. Select the compute instance in the list.
4. On the top toolbar, select **Stop**.

Delete all resources

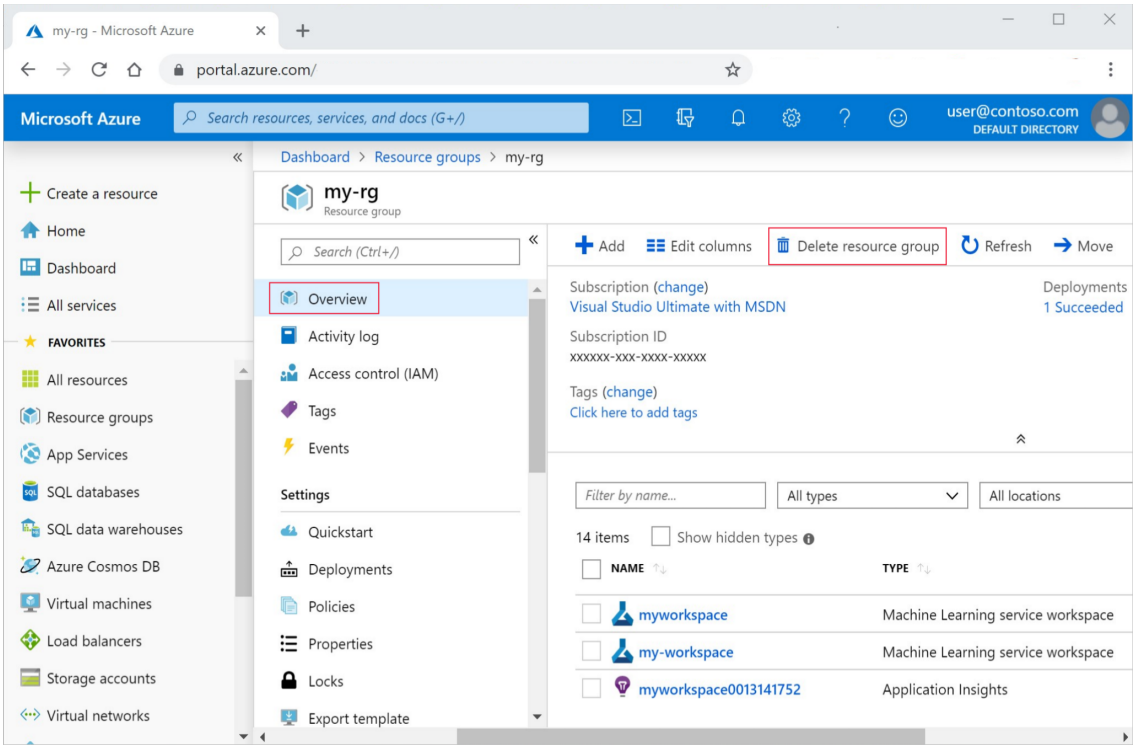
> **Important**
>
> The resources that you created can be used as prerequisites to other Azure Machine Learning tutorials and how-to articles.

If you don't plan to use any of the resources that you created, delete them so you don't incur any charges:

1. In the Azure portal, select **Resource groups** on the far left.

2. From the list, select the resource group that you created.

3. Select **Delete resource group**.



4. Enter the resource group name. Then select **Delete**.

Next steps

Now that you have an idea of what's involved in training and deploying a model, learn more about the process in these tutorials:

Tutorial	Description
Upload, access and explore your data in Azure Machine Learning	Store large data in the cloud and retrieve it from notebooks and scripts
Model development on a cloud workstation	Start prototyping and developing machine learning models
Train a model in Azure Machine Learning	Dive in to the details of training a model
Deploy a model as an online endpoint	Dive in to the details of deploying a model
Create production machine learning pipelines	Split a complete machine learning task into a multistep workflow.