Microsoft

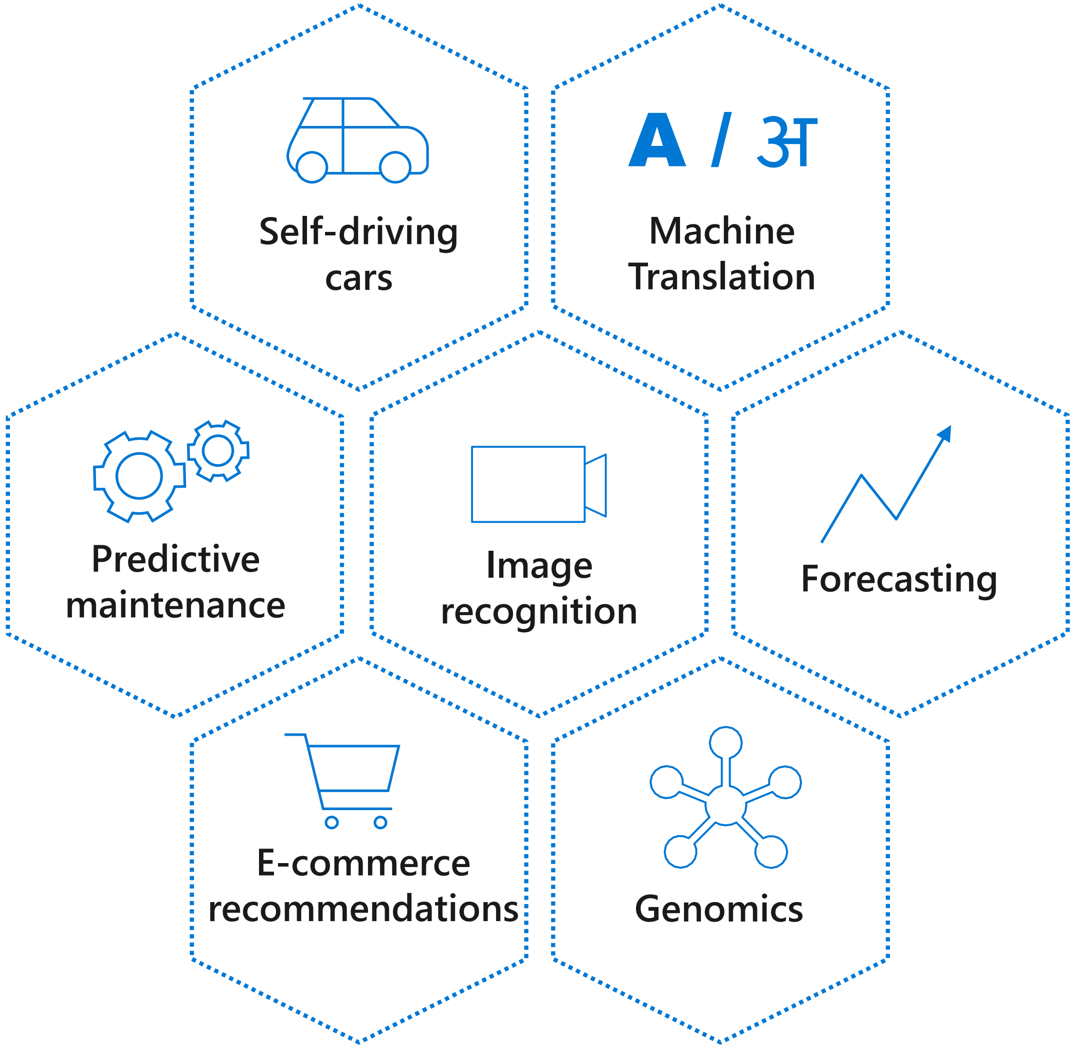
全生命周期的机器学习平台

王芷
创新技术架构师
wazhi@microsoft.com

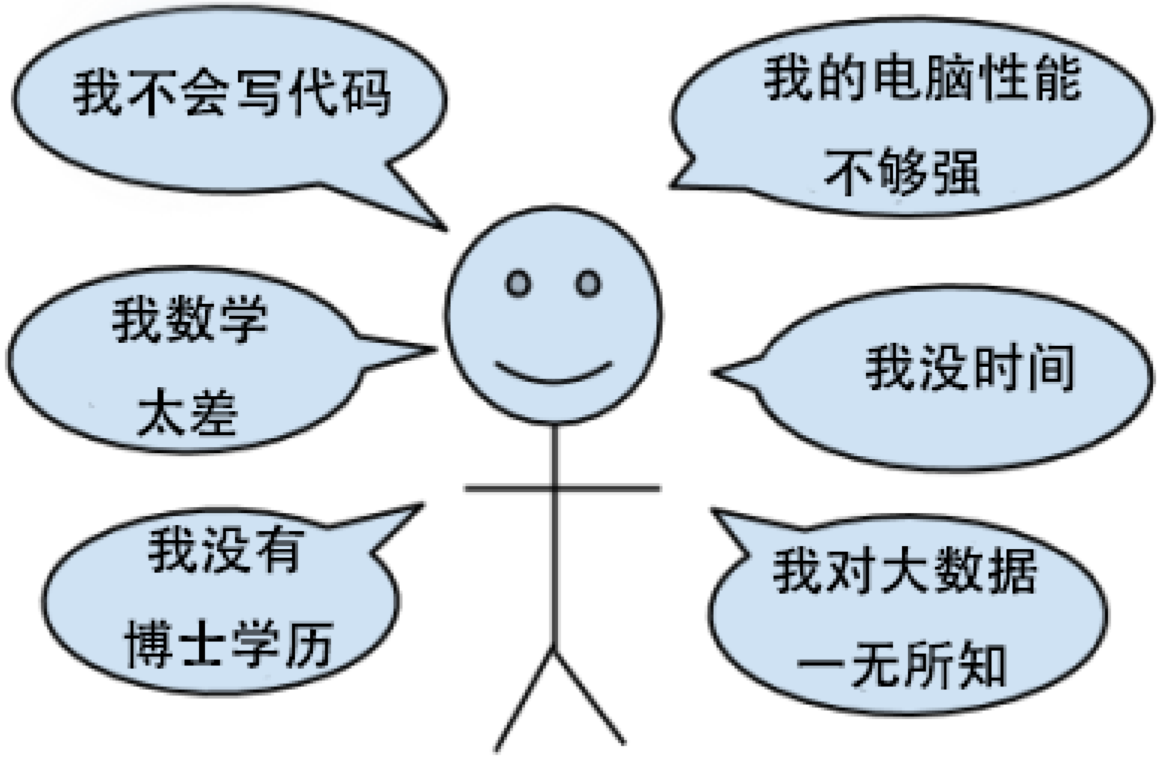
敲黑板

1. 端到端ML平台
2. 提高ML生产力
3. 总有一款适合你
4. 课后作业少不了

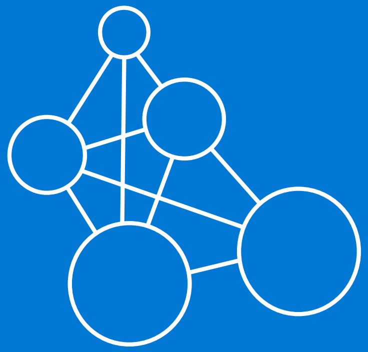
无所不能，无处不在的机器学习



我太难了...

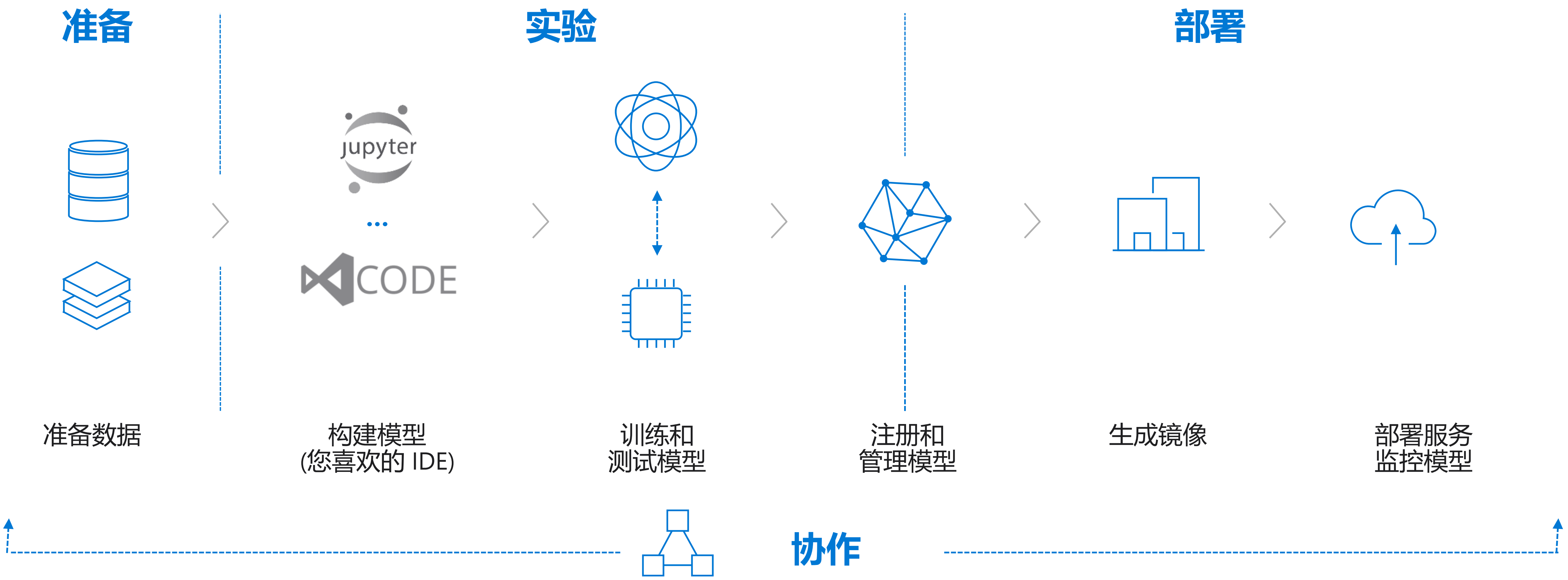


我没法开始学习“机器学习”，因为……

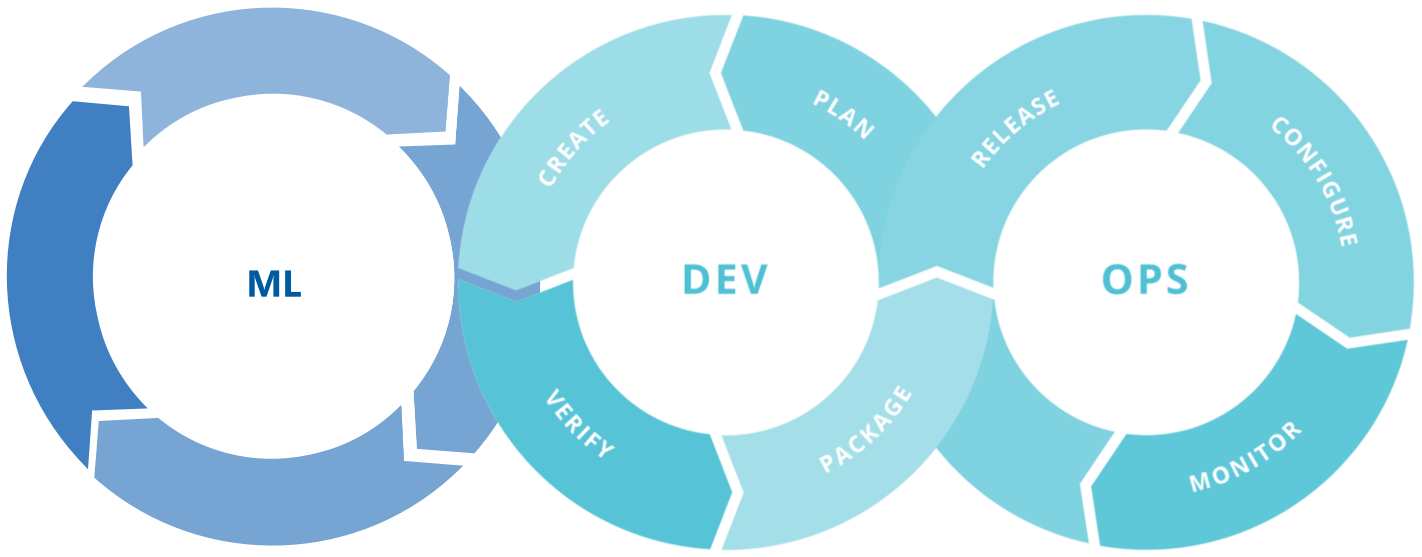


一个优秀的机器学习平台应该具备哪些素质？

支持端到端的机器学习全过程



MLOps = ML + DEV + OPS



Experiment
Data Acquisition
Business Understanding
Initial Modeling

Develop
Modeling + Testing
Continuous Integration
Continuous Deployment

Operate
Continuous Delivery
Data Feedback Loop
System + Model Monitoring

Azure Machine Learning Service
为机器学习而生

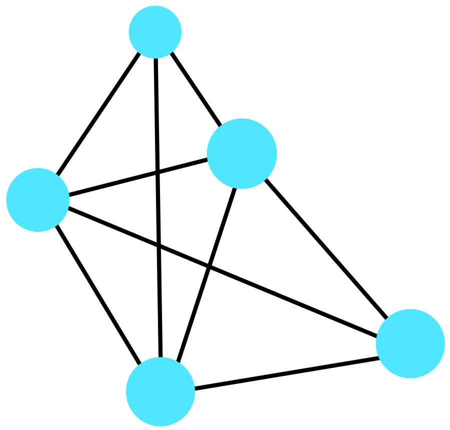
什么是 Azure 机器学习服务?

Azure 云服务 + Python Sdk

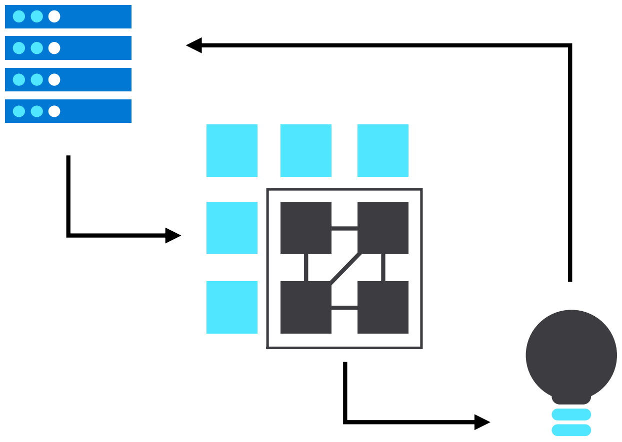
这使您能够:

- ✓ 准备数据
- ✓ 构建模型
- ✓ 训练模型
- ✓ 管理模型
- ✓ 跟踪实验
- ✓ 部署模型

Azure 机器学习服务



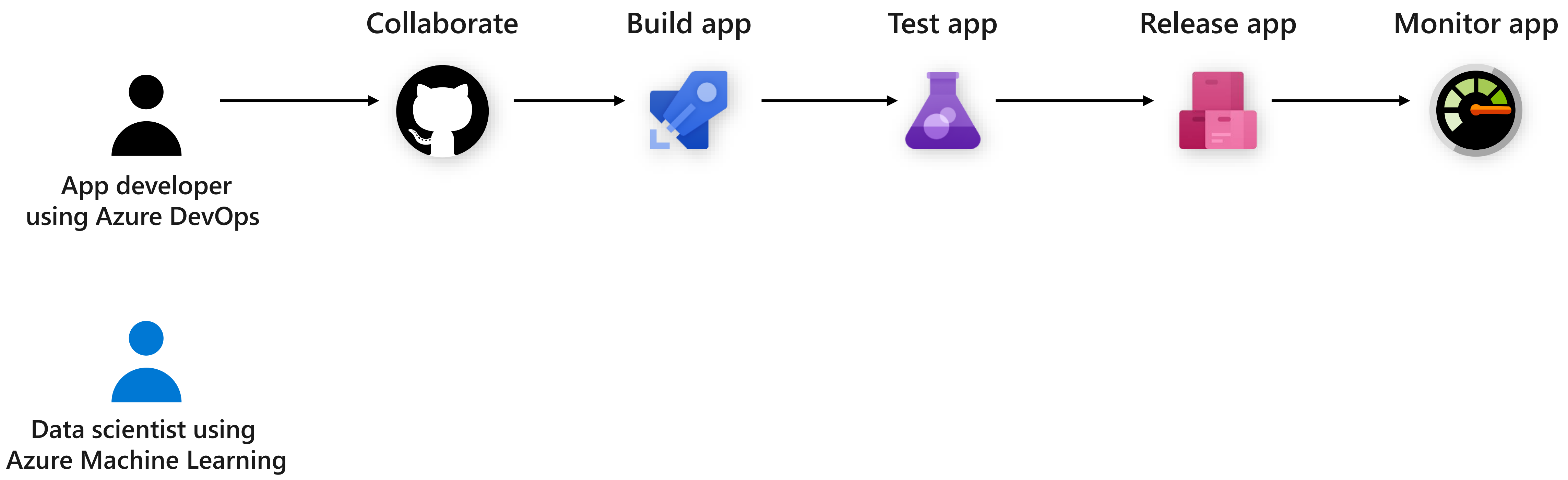
Simplified machine learning



End to end lifecycle management

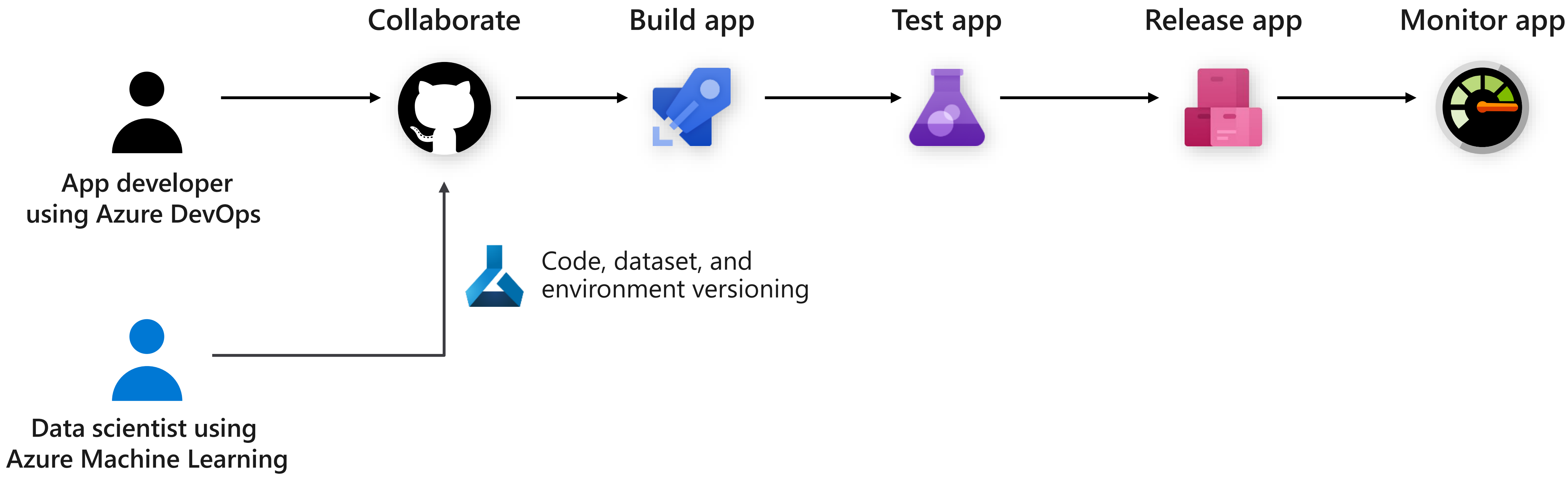
Open platform

MLOps Workflow



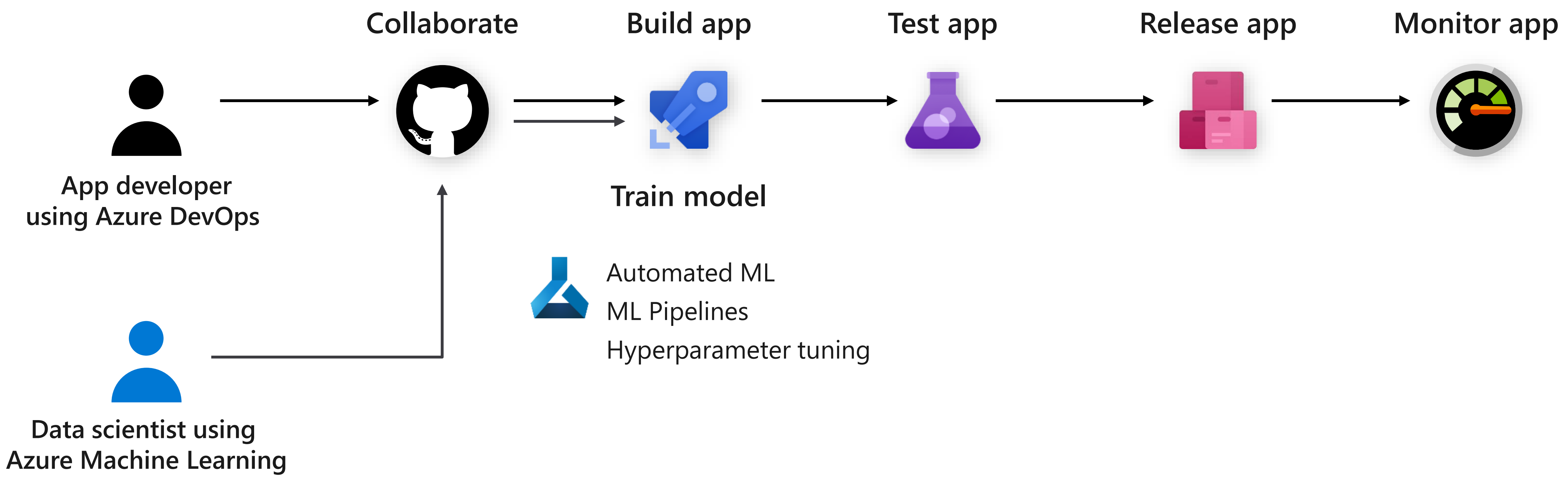
- ✗ Model reproducibility
- ✗ Model validation
- ✗ Model deployment
- ✗ Model retraining

MLOps Workflow



- ✓ Model reproducibility
- ✗ Model validation
- ✗ Model deployment
- ✗ Model retraining

MLOps Workflow



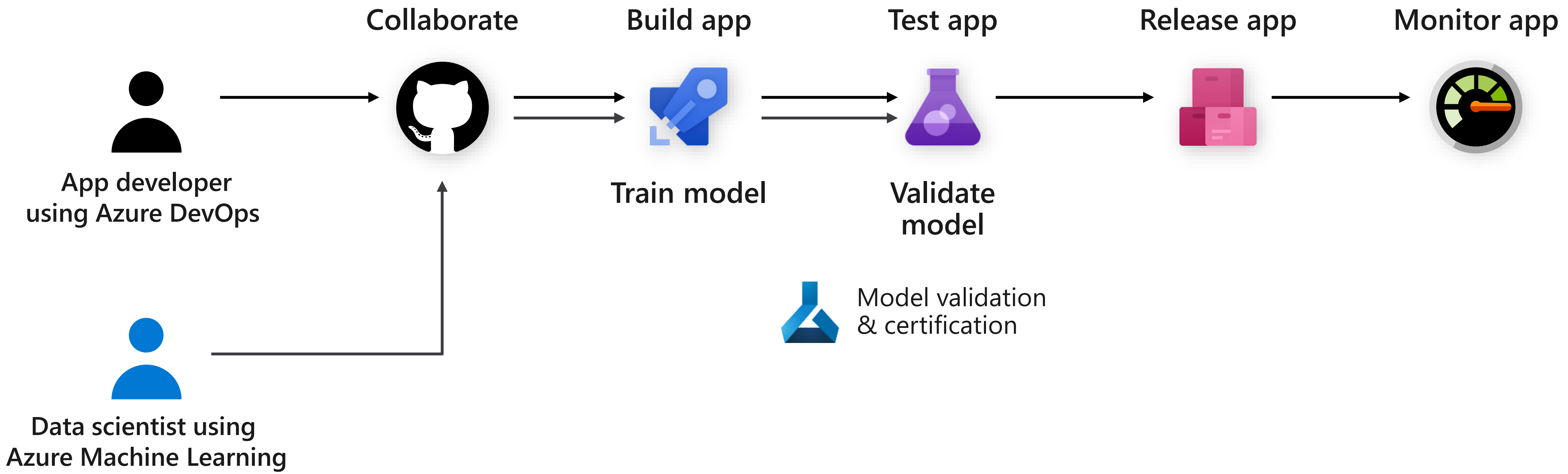
✓ Model reproducibility

✗ Model validation

✗ Model deployment

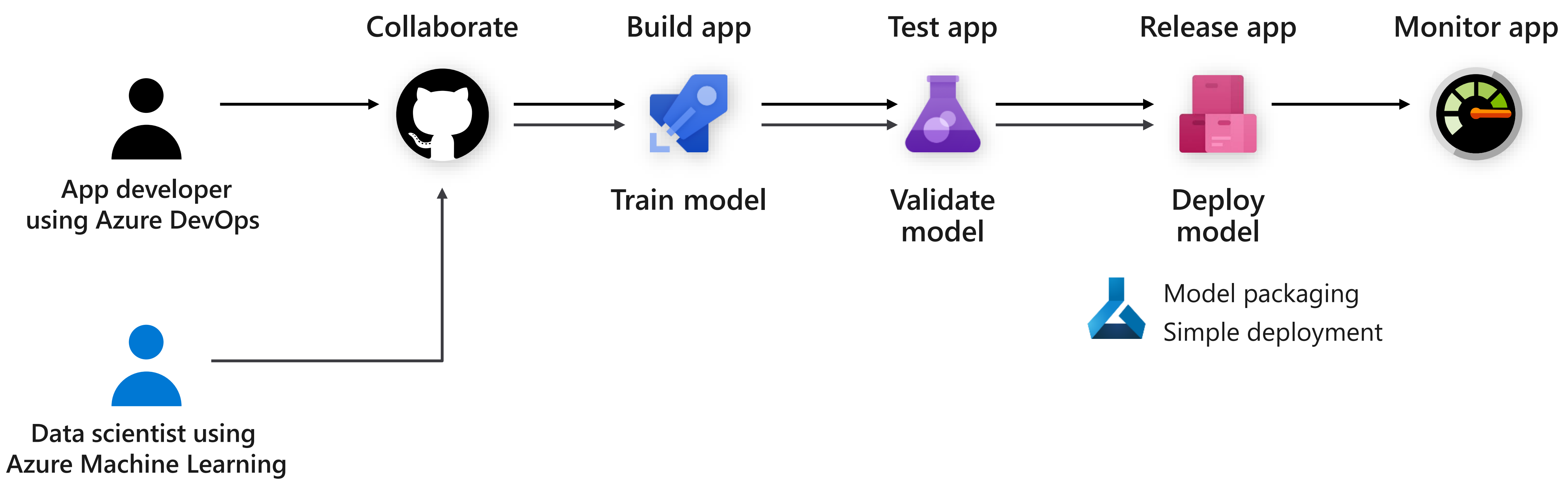
✗ Model retraining

MLOps Workflow



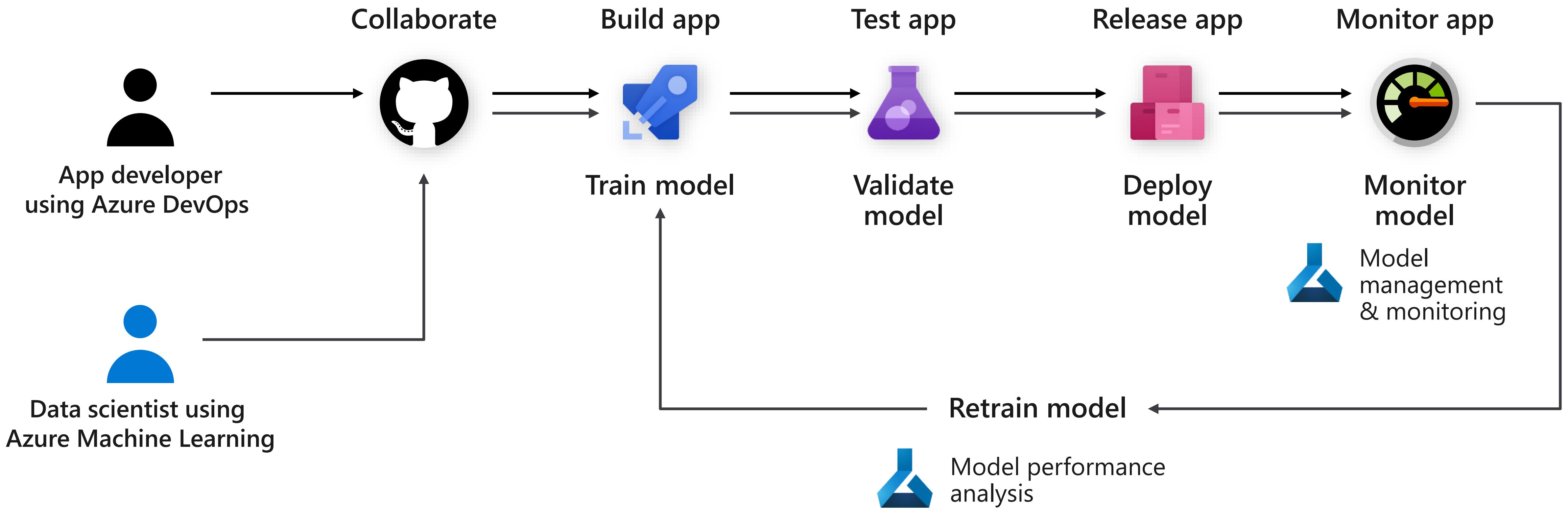
✓ Model reproducibility ✓ Model validation ✗ Model deployment ✗ Model retraining

MLOps Workflow



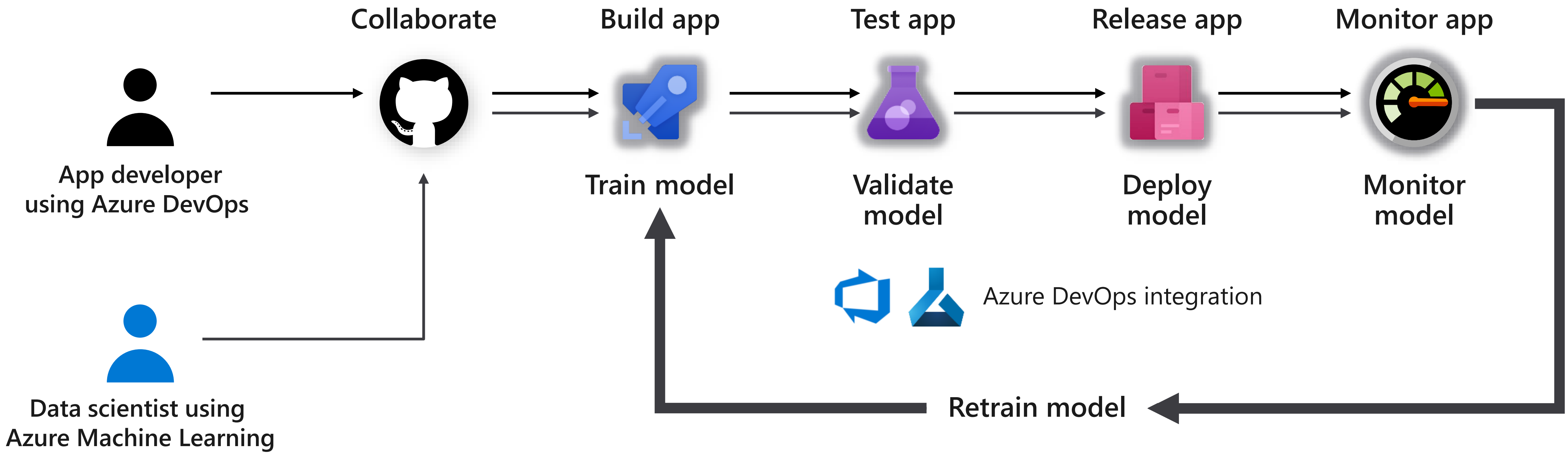
✓ Model reproducibility ✓ Model validation ✓ Model deployment ✗ Model retraining

MLOps with Azure Machine Learning



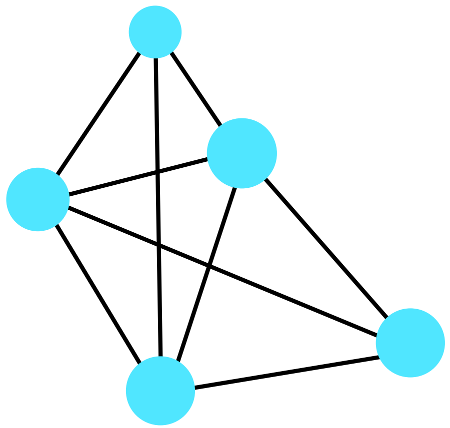
- ✓ Model reproducibility
- ✓ Model validation
- ✓ Model deployment
- ✓ Model retraining

MLOps with Azure Machine Learning

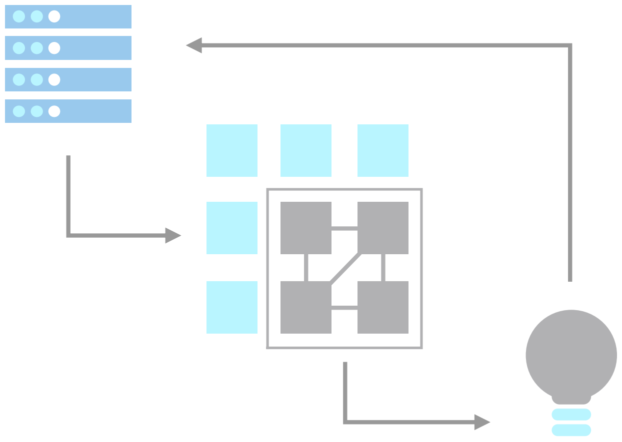


- ✓ Model reproducibility
- ✓ Model validation
- ✓ Model deployment
- ✓ Model retraining

提高生产力——简化的机器学习平台



Simplified machine learning

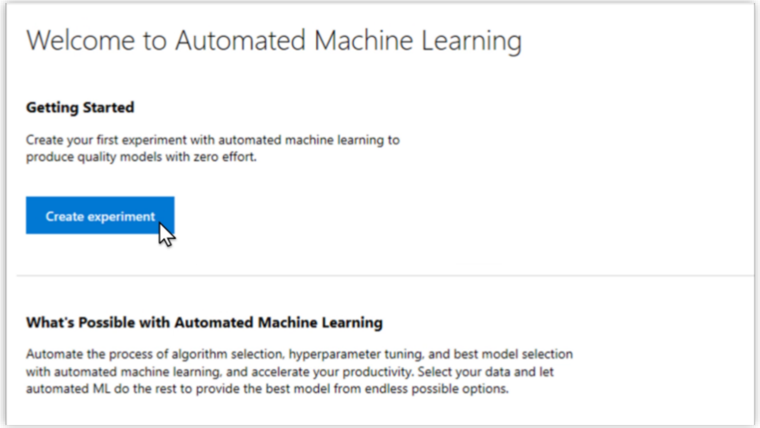


End to end lifecycle management

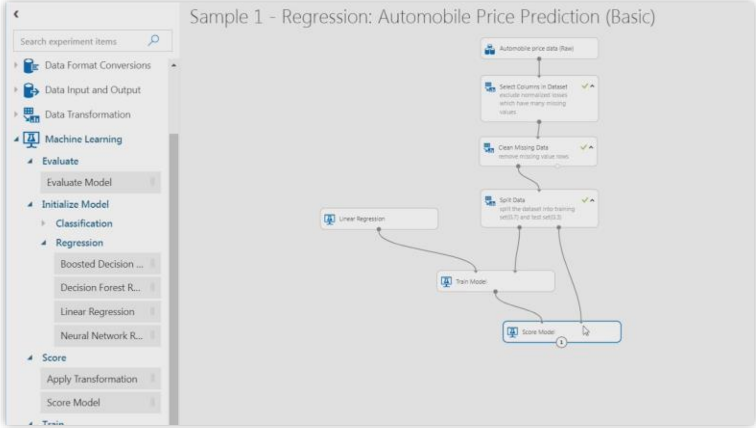
Open platform

Simplify machine learning for any skill level

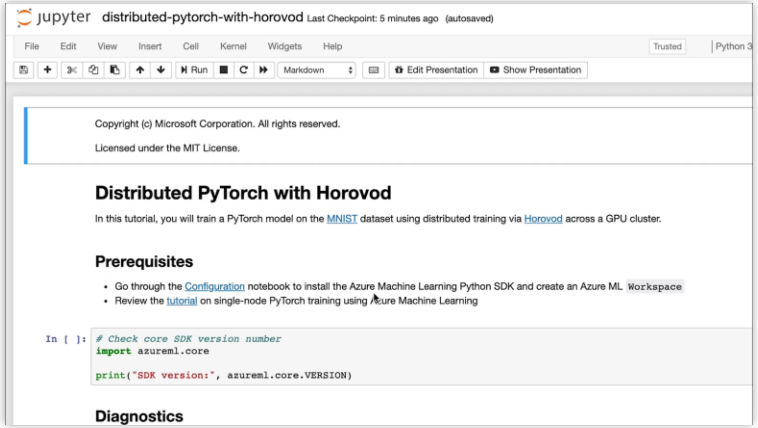
New capabilities in Azure Machine Learning service



Automated
machine learning UI



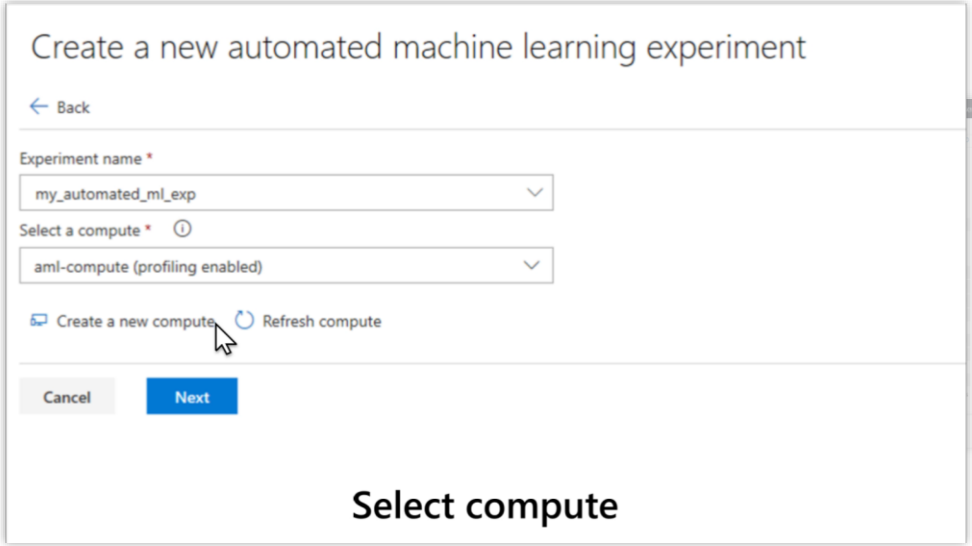
Visual interface



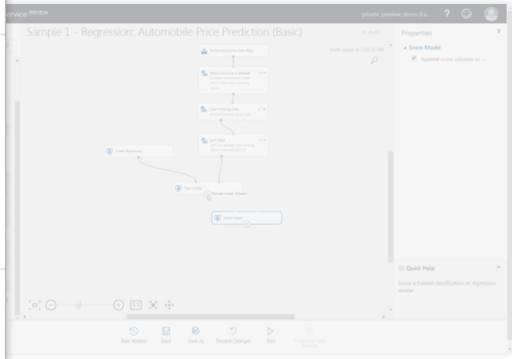
Machine learning notebooks

Simplify machine learning for any skill level

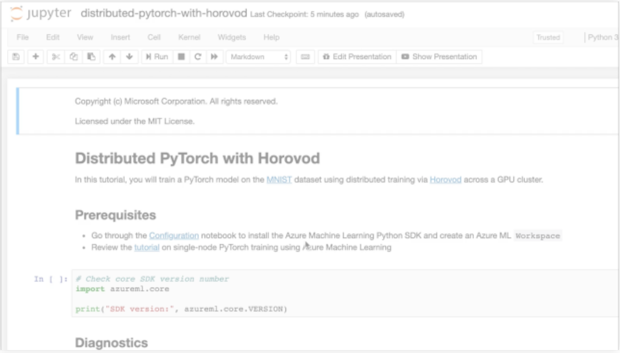
New capabilities in Azure Machine Learning service



Automated machine learning UI



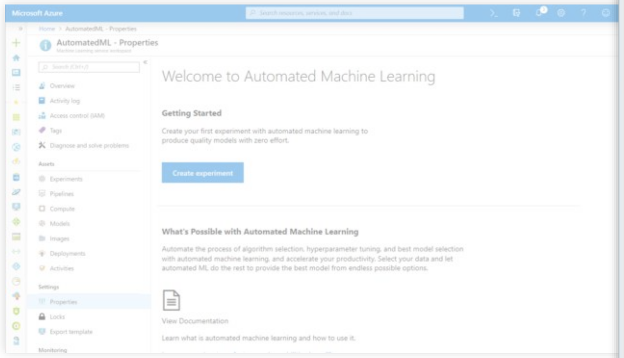
Visual interface



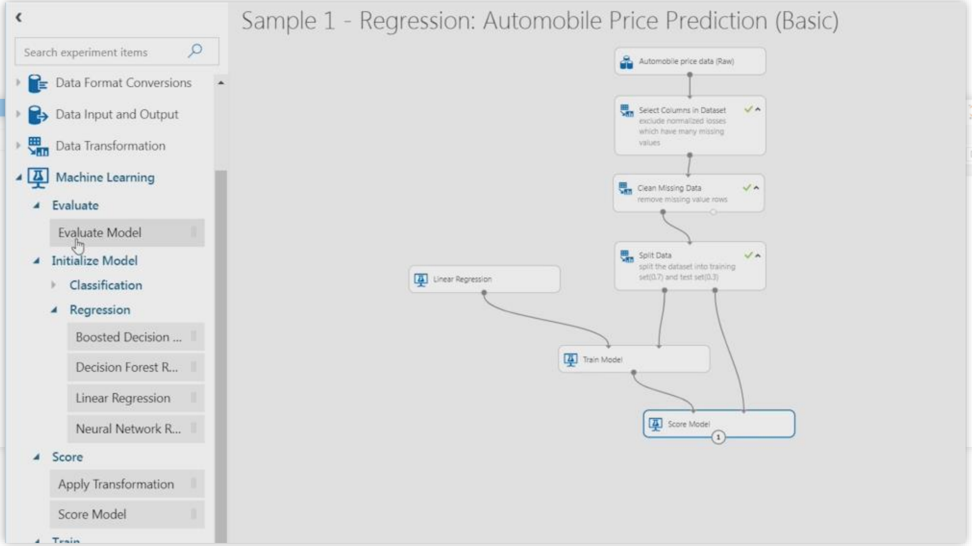
Machine learning notebooks

Simplify machine learning for any skill level

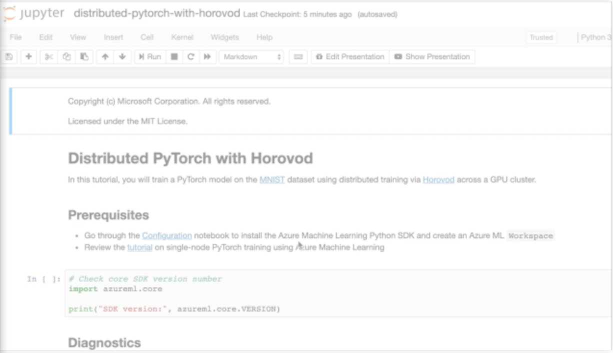
New capabilities in Azure Machine Learning service



Automated
machine learning UI



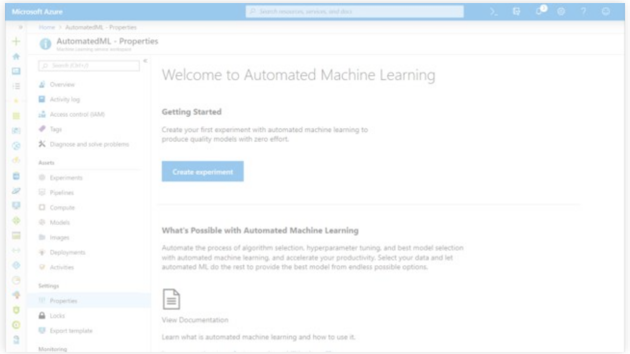
Visual interface



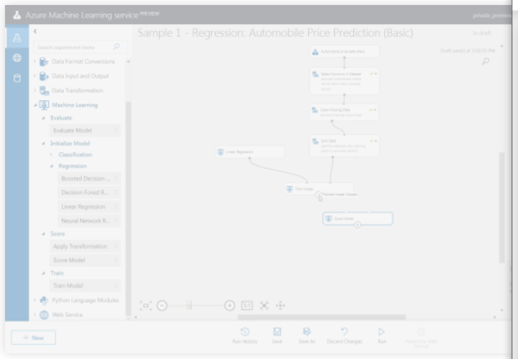
Machine learning notebooks

Simplify machine learning for any skill level

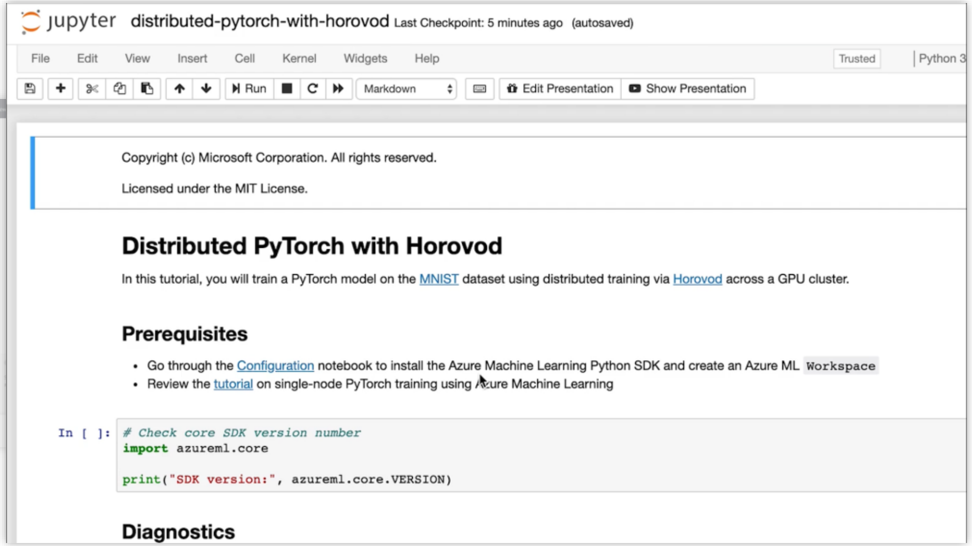
New capabilities in Azure Machine Learning service



Automated
machine learning UI



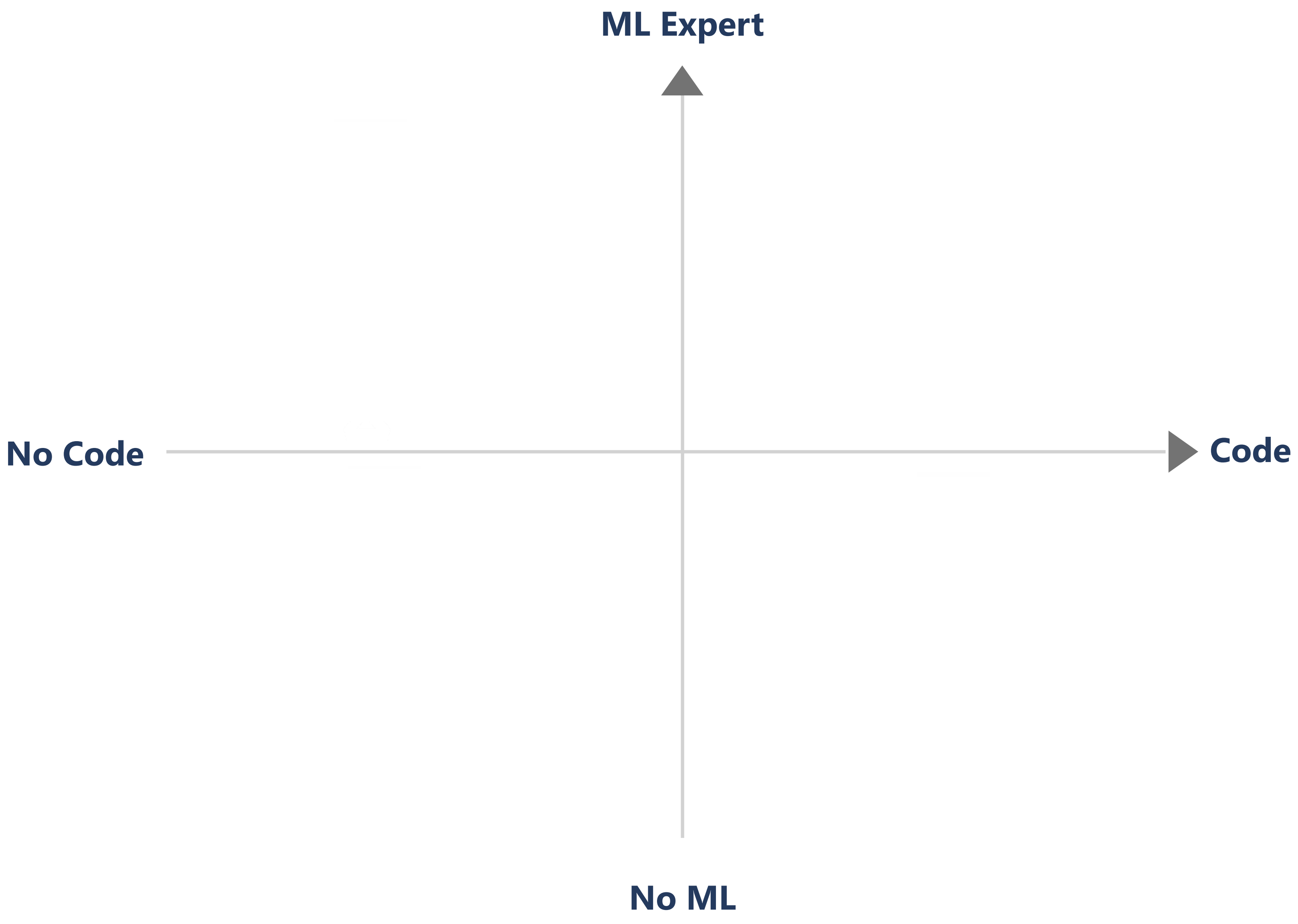
Visual interface



Machine learning notebooks

总有一款适合你

More Personas Building Machine Learning Models



Samantha

using Visual Interface

PAIN POINTS

Familiar with ML algorithms and benefits, but doesn't like or know how to code

Require scale for training and deployment jobs

Bringing models into production can be difficult

VALUE PROP

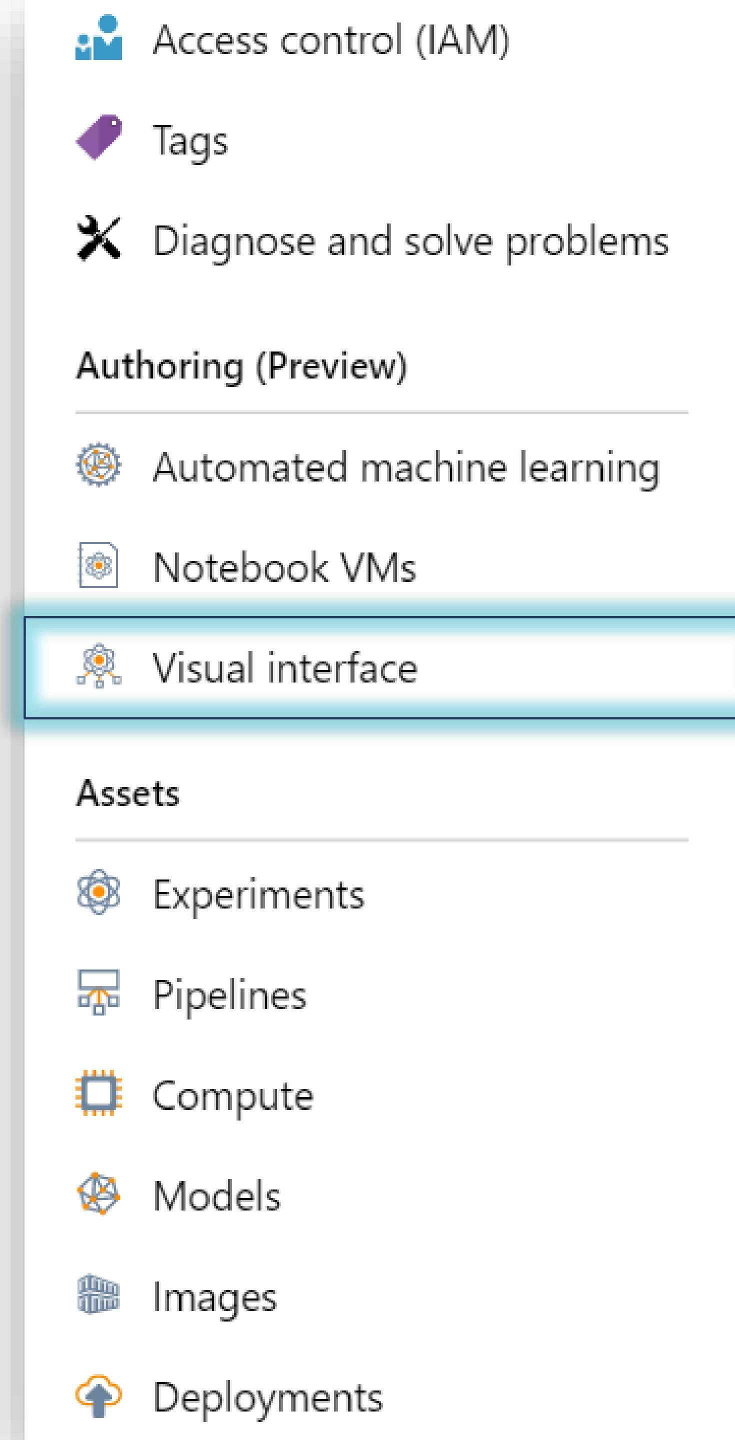
Simple drag and drop UI with functionality of powerful ML framework

Standard Modules and samples makes it easy to get started and customize the experience

Support for custom R and Python modules is required

Easily share outcomes with other personas in the organization

Seamless integration with model registry, deployments and MLOps



Dean

using AutoML UI

PAIN POINTS

Familiar with the benefit of ML, but doesn't know how to get started

Would like to predict something in the data to provide business value

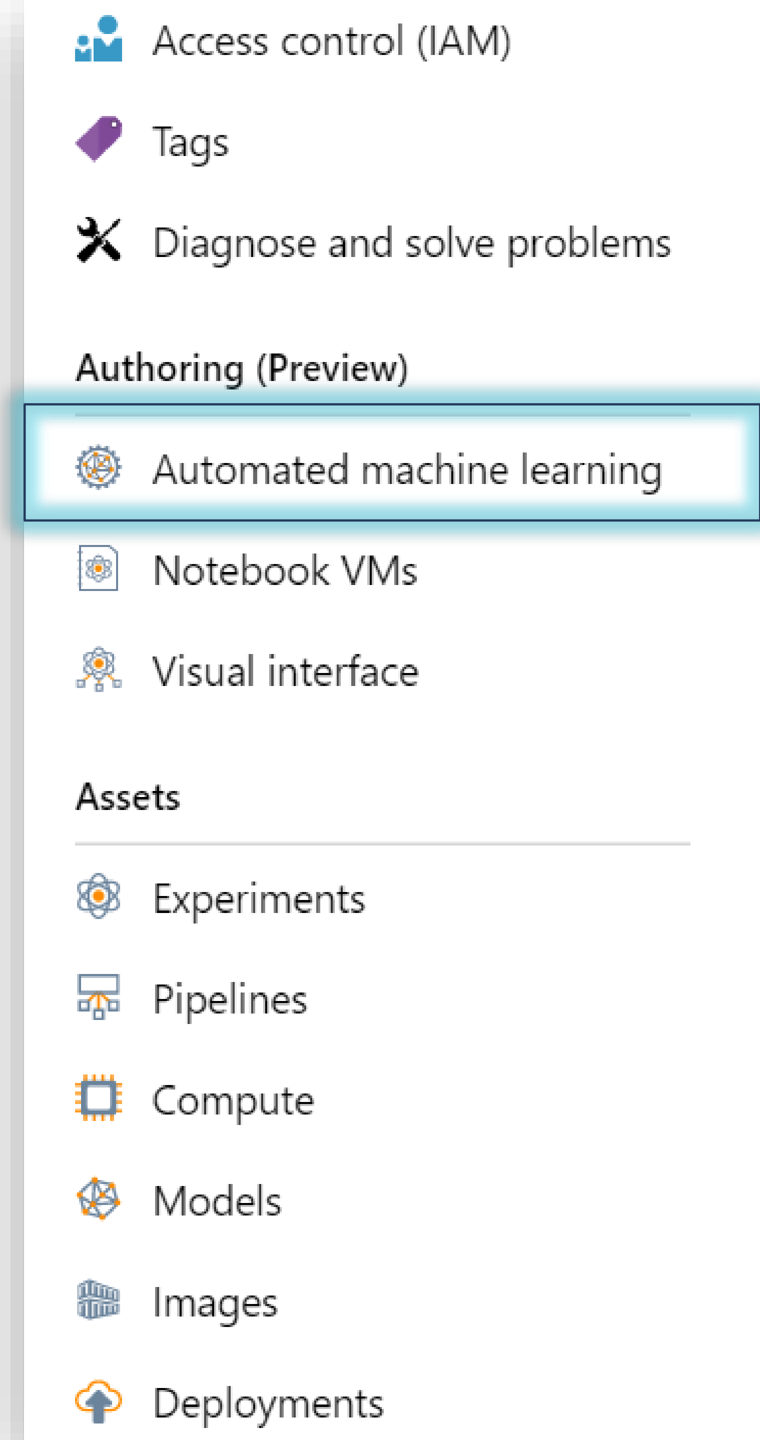
Familiar with the data available in the organization as he has used it for building reports and insights

VALUE PROP

Use the same data you are already familiar with from other areas

Easily communicate results and insights with the data science team to get approval on approach

Seamless integration with model registry, deployments and MLOps



Daniel
using Notebooks VMs

PAIN POINTS

Familiar with the Jupyter notebook workflow; time consuming to learn new workflows

Limited by their local compute

Experimentation is messy, untracked, and hard to follow for others

Difficult to collaborate with other data scientists and get models into production

VALUE PROP

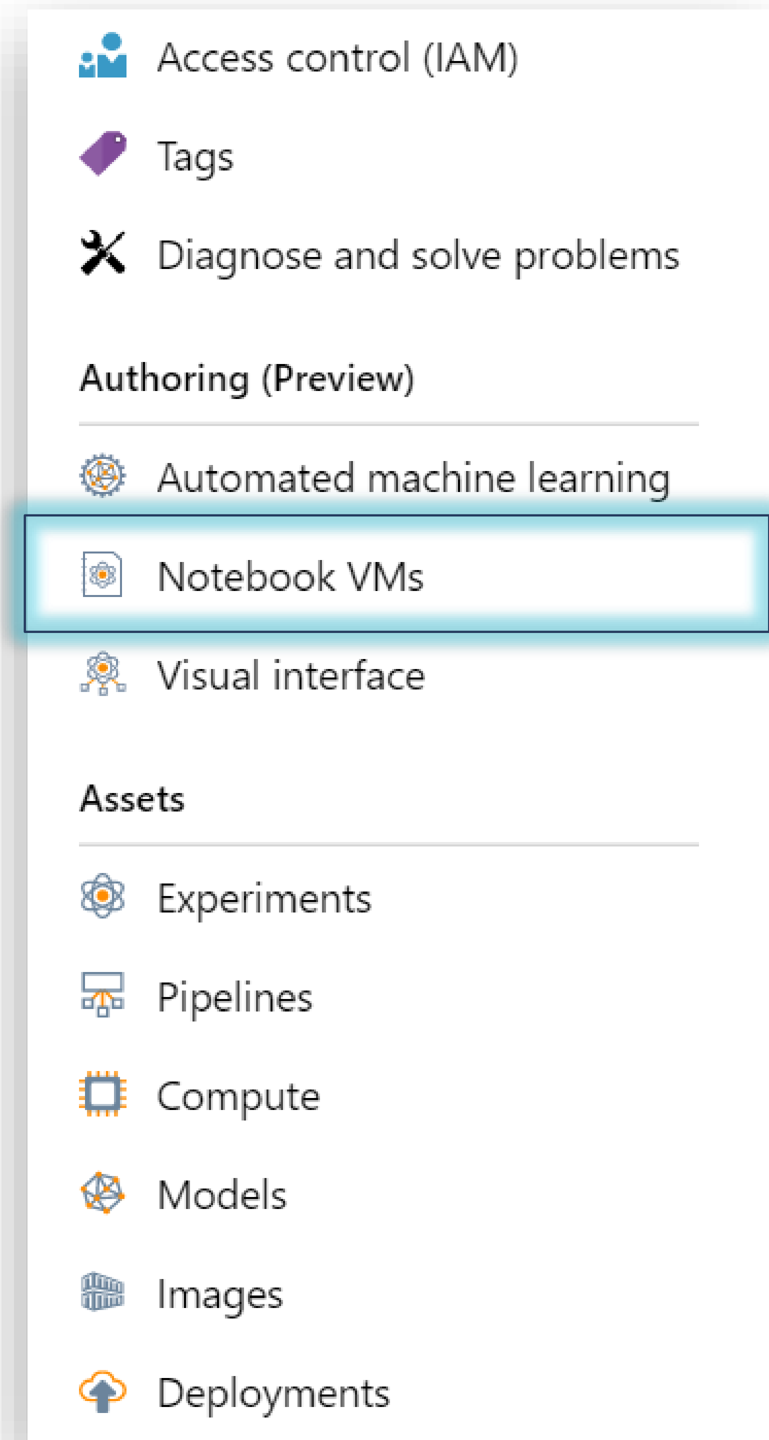
Provide familiar workflow

Provision VMs as large as needed for the task

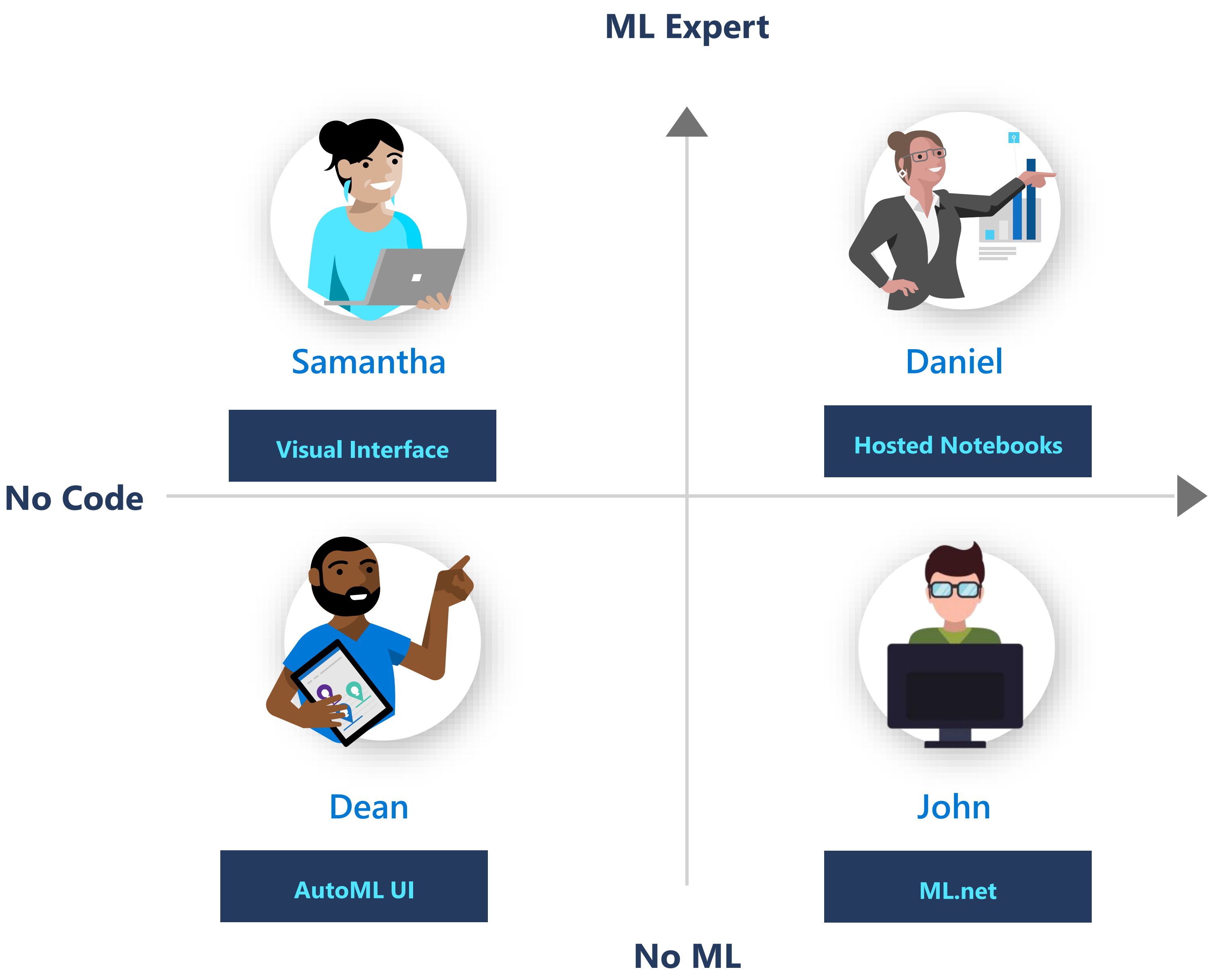
Easily track experiments, metrics, etc.

Easily store, access, share their code throughout the organization

Seamless integration with model registry, deployments and MLOps

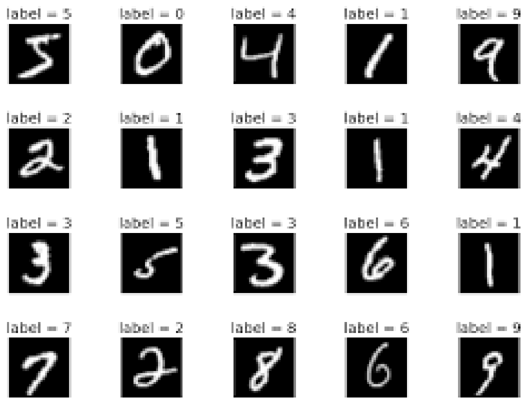


More Personas Building Machine Learning Models



Demo

Setup for Code Example



This tutorial trains a simple logistic regression using the MNIST dataset and scikit-learn with Azure Machine Learning service.

MNIST is a dataset consisting of 70,000 grayscale images.

- Each image is a handwritten digit of 28x28 pixels, representing a number from 0 to 9.
- The goal is to create a multi-class classifier to identify the digit a given image represents.

Step 1 – Create a workspace

```
from azureml.core import Workspace
ws = Workspace.create(name='myworkspace',
                      subscription_id='<azure-subscription-id>',
                      resource_group='myresourcegroup',
                      create_resource_group=True,
                      location='eastus2' # or other supported Azure region
                     )

# see workspace details
ws.get_details()
```

Step 2 – Create an Experiment

Create an experiment to track the runs in the workspace. A workspace can have multiple experiments

```
experiment_name = ‘my-experiment-1'

from azureml.core import Experiment
exp = Experiment(workspace=ws, name=experiment_name)
```

Step 3 – Create remote compute target

```
# choose a name for your cluster, specify min and max nodes
compute_name = os.environ.get("BATCHAI_CLUSTER_NAME", "cpucluster")
compute_min_nodes = os.environ.get("BATCHAI_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("BATCHAI_CLUSTER_MAX_NODES", 4)

# This example uses CPU VM. For using GPU VM, set SKU to STANDARD_NC6
vm_size = os.environ.get("BATCHAI_CLUSTER_SKU", "STANDARD_D2_V2")

provisioning_config = AmlCompute.provisioning_configuration(
                              vm_size = vm_size,
                              min_nodes = compute_min_nodes,
                              max_nodes = compute_max_nodes)
# create the cluster
print(‘ creating a new compute target... ')
compute_target = ComputeTarget.create(ws, compute_name, provisioning_config)

# You can poll for a minimum number of nodes and for a specific timeout.
# if no min node count is provided it will use the scale settings for the cluster
compute_target.wait_for_completion(show_output=True,
                                   min_node_count=None, timeout_in_minutes=20)
```

Zero is the default. If min is zero then the cluster is automatically deleted when no jobs are running on it.

Step 4 – Upload data to the cloud

First load the compressed files into numpy arrays. Note the '*load_data*' is a custom function that simply parses the compressed files into numpy arrays.

```
# note that while loading, we are shrinking the intensity values (X) from 0-255 to 0-1 so that the model converge faster.
X_train = load_data('./data/train-images.gz', False) / 255.0
y_train = load_data('./data/train-labels.gz', True).reshape(-1)

X_test = load_data('./data/test-images.gz', False) / 255.0
y_test = load_data('./data/test-labels.gz', True).reshape(-1)
```

Now make the data accessible remotely by uploading that data from your local machine into Azure so it can be accessed for remote training. The files are uploaded into a directory named mnist at the root of the datastore.

```
ds = ws.get_default_datastore()
print(ds.datastore_type, ds.account_name, ds.container_name)

ds.upload(src_dir='./data', target_path='mnist', overwrite=True, show_progress=True)
```

We now have everything you need to start training a model.

Step 5 – Train a local model

Train a simple logistic regression model using scikit-learn locally. This should take a minute or two.

```
%%time from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Next, make predictions using the test set and calculate the accuracy
y_hat = clf.predict(X_test)
print(np.average(y_hat == y_test))
```

You should see the local model accuracy displayed. [It should be a number like 0.915]

Step 6 – Train model on remote cluster

To submit a training job to a remote you have to perform the following tasks:

- 6.1: Create a directory
- 6.2: Create a training script
- 6.3: Create an estimator object
- 6.4: Submit the job

Step 6.1 – Create a directory

Create a directory to deliver the required code from your computer to the remote resource.

```
import os
script_folder = './sklearn-mnist' os.makedirs(script_folder, exist_ok=True)
```

Step 6.2 – Create a Training Script (1/2)

```
%%writefile $script_folder/train.py
# load train and test set into numpy arrays
# Note: we scale the pixel intensity values to 0-1 (by dividing it with 255.0) so the model can
# converge faster.
# ‘data_folder’ variable holds the location of the data files (from datastore)
Reg = 0.8 # regularization rate of the logistic regression model.
X_train = load_data(os.path.join(data_folder, 'train-images.gz'), False) / 255.0
X_test  = load_data(os.path.join(data_folder, 'test-images.gz'), False) / 255.0
y_train = load_data(os.path.join(data_folder, 'train-labels.gz'), True).reshape(-1)
y_test  = load_data(os.path.join(data_folder, 'test-labels.gz'), True).reshape(-1)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape, sep = '\n’)

# get hold of the current run
run = Run.get_context()
#Train a logistic regression model with regularizaion rate of’ ‘reg’
clf = LogisticRegression(C=1.0/reg, random_state=42)
clf.fit(X_train, y_train)
```

Step 6.2 – Create a Training Script (2/2)

```
print('Predict the test set’)
y_hat = clf.predict(X_test)

# calculate accuracy on the prediction
acc = np.average(y_hat == y_test)
print('Accuracy is', acc)


run.log('regularization rate', np.float(args.reg))
run.log('accuracy', np.float(acc)) os.makedirs('outputs', exist_ok=True)


# The training script saves the model into a directory named ‘outputs’. Note files saved in the
# outputs folder are automatically uploaded into experiment record. Anything written in this
# directory is automatically uploaded into the workspace.
joblib.dump(value=clf, filename='outputs/sklearn_mnist_model.pkl')
```

Step 6.3 – Create an Estimator

An estimator object is used to submit the run.

```
from azureml.train.estimator import Estimator

script_params = { '--data-folder': ds.as_mount(), '--regularization': 0.8 }

est = Estimator(source_directory=script_folder,
                script_params=script_params,
                compute_target=compute_target,
                entry_script='train.py’,
                conda_packages=['scikit-learn'])
```

The directory that contains the scripts. All the files in this directory are uploaded into the cluster nodes for execution

Name of estimator

Python Packages needed for training

Training Script Name

Compute target (Batch AI in this case)

Parameters required from the training script

Step 6.4 – Submit the job to the cluster for training

```
run = exp.submit(config=est)
run
```

What happens after you submit the job?

Image creation

A Docker image is created matching the Python environment specified by the estimator. The image is uploaded to the workspace. Image creation and uploading takes about 5 minutes.

This happens once for each Python environment since the container is cached for subsequent runs. During image creation, logs are streamed to the run history. You can monitor the image creation progress using these logs.

Scaling

If the remote cluster requires more nodes to execute the run than currently available, additional nodes are added automatically. Scaling typically takes about 5 minutes.

Running

In this stage, the necessary scripts and files are sent to the compute target, then data stores are mounted/copied, then the entry_script is run. While the job is running, stdout and the ./logs directory are streamed to the run history. You can monitor the run's progress using these logs.

Post-Processing

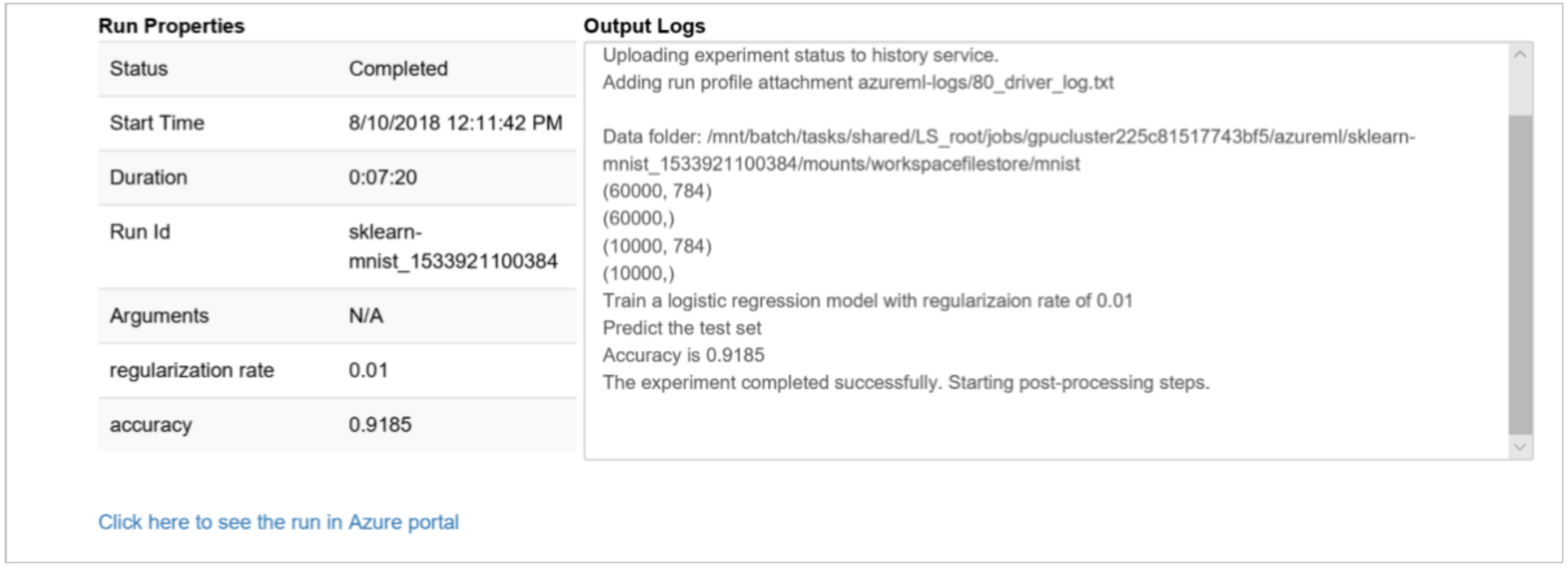
The ./outputs directory of the run is copied over to the run history in your workspace so you can access these results.

Step 7 – Monitor a run

You can watch the progress of the run with a Jupyter widget. The widget is asynchronous and provides live updates every 10-15 seconds until the job completes.

```
from azureml.widgets import RunDetails
RunDetails(run).show()
```

Here is a still snapshot of the widget shown at the end of training:



Step 8 – See the results

As model training and monitoring happen in the background. Wait until the model has completed training before running more code. Use *wait_for_completion* to show when the model training is complete

Specify 'True' for a verbose log

```
run.wait_for_completion(show_output=False)

# now there is a trained model on the remote cluster
print(run.get_metrics())
```

Displays the accuracy of the model. You should see an output that looks like this.

```
{'regularization rate': 0.8, 'accuracy': 0.9204}
```

Step 9 – Register the model

Recall that the last step in the training script is:

```
joblib.dump(value=clf, filename='outputs/sklearn_mnist_model.pkl')
```

This wrote the file 'outputs/sklearn_mnist_model.pkl' in a directory named 'outputs' in the VM of the cluster where the job is executed.

- outputs is a special directory in that all content in this directory is automatically uploaded to your workspace.
- This content appears in the run record in the experiment under your workspace.
- Hence, the model file is now also available in your workspace.

```
# register the model in the workspace
model = run.register_model (
                             model_name='sklearn_mnist’,
                             model_path='outputs/sklearn_mnist_model.pkl’)
```

The model is now available to query, examine, or deploy

Step 9 – Deploy the Model

Deploy the model registered in the previous slide, to Azure Container Instance (ACI) as a Web Service

There are 4 steps involved in model deployment

Step 9.1 – Create scoring script

Step 9.2 – Create environment file

Step 9.3 – Create configuration file

Step 9.4 – Deploy to ACI!

Step 9.1 – Create the scoring script

Create the scoring script, called score.py, used by the web service call to show how to use the model. It requires two functions – init() and run (input data)

The init() function, typically loads the model into a global object. This function is run only once when the Docker container is started.

```
from azureml.core.model import Model

def init():
      global model
      # retreive the path to the model file using the model name
      model_path = Model.get_model_path('sklearn_mnist’)
      model = joblib.load(model_path)

def run(raw_data):
      data = np.array(json.loads(raw_data)['data’])
      # make prediction
      y_hat = model.predict(data) return json.dumps(y_hat.tolist())
```

The run(input_data) function uses the model to predict a value based on the input data. Inputs and outputs to the run typically use JSON for serialization and de-serialization, but other formats are supported

Step 9.2 – Create environment file

Create an environment file, called *myenv.yml*, that specifies all of the script's package dependencies. This file is used to ensure that all of those dependencies are installed in the Docker image. This example needs scikit-learn and azureml-sdk.

```
from azureml.core.conda_dependencies import CondaDependencies

myenv = CondaDependencies()
myenv.add_conda_package("scikit-learn")

with open("myenv.yml","w") as f:
        f.write(myenv.serialize_to_string())
```

Step 9.3 – Create configuration file

Create a deployment configuration file and specify the number of CPUs and gigabyte of RAM needed for the ACI container. Here we will use the defaults (1 core and 1 gigabyte of RAM)

```
from azureml.core.webservice import AciWebservice

aciconfig = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1,
                                               tags={"data": "MNIST", "method" : "sklearn"},
                                               description='Predict MNIST with sklearn')
```

Step 9.4 – Deploy the model to ACI

Build an image using:
- The scoring file (score.py)
- The environment file (myenv.yml)
- The model file

Register that image under the workspace and send the image to the ACI container.

```
%%time
from azureml.core.webservice import Webservice
from azureml.core.image import ContainerImage

# configure the image
image_config = ContainerImage.image_configuration(
                                        execution_script ="score.py",
                                        runtime ="python",
                                        conda_file ="myenv.yml")

service = Webservice.deploy_from_model(workspace=ws, name='sklearn-mnist-svc’,
                                        deployment_config=aciconfig, models=[model],
                                        image_config=image_config)

service.wait_for_deployment(show_output=True)
```

Start up a container in ACI using the image

Step 10 – Test the deployed model using the HTTP end point

Test the deployed model by sending images to be classified to the HTTP endpoint

```
import requests
import json

# send a random row from the test set to score
random_index = np.random.randint(0, len(X_test)-1)
input_data = "{\"data\": [" + str(list(X_test[random_index])) + "]}"

headers = {'Content-Type':'application/json’}

resp = requests.post(service.scoring_uri, input_data, headers=headers)

print("POST to url", service.scoring_uri)
#print("input data:", input_data)
print("label:", y_test[random_index])
print("prediction:", resp.text)
```

Send the data to the HTTP end-point for scoring

课后作业

免费试用:
http://aka.ms/amlfree

产品文档:
http://aka.ms/azureml-docs

示例代码:
https://github.com/Azure/MachineLearningNotebooks
https://github.com/Microsoft/MLOps

Microsoft