I want my model to be deployed !

(another story of MLOps)

Paul PETON

Senior Data Consultant
Microsoft AI MVP

www.DataPlatformGeeks.com | www.SQLServerGeeks.com

@TheDataGeeks | @SQLServerGeeks | admin@DataPlatformGeeks.com

Paul Péton

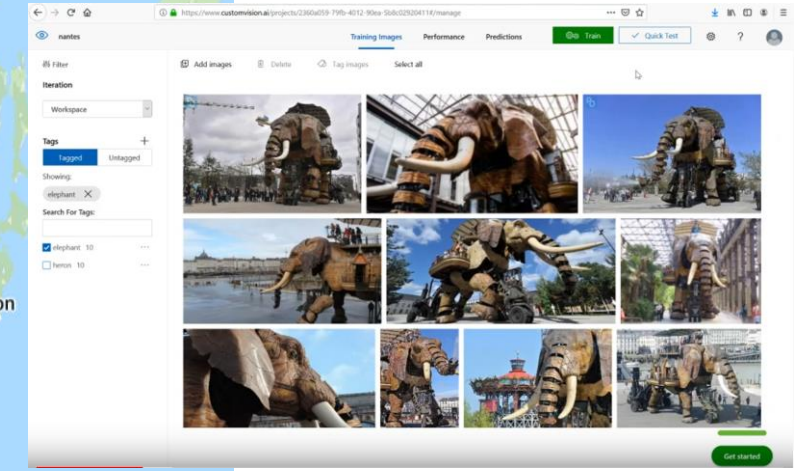
- Microsoft Artificial Intelligence MVP since 2018
- Meetup organizer : Club Power BI @Nantes, France (YouTube)
- Meetup speaker : Azure Nantes, France (YouTube)
- Used to be Data Miner *(an old buzzword…)*
- Senior Data Consultant @AZEO

Twitter : @paulpeton

https://www.linkedin.com/in/paul-peton-datascience

https://github.com/methodidacte/

Nantes, FRANCE



L'expertise Cloud AZEO

Nous vous donnons les clés du Cloud Microsoft

AZEO est reconnu comme l'un des influenceurs majeurs du Cloud Microsoft en France. Certifié Cloud Solution Provider par Microsoft, nous sommes suivis par l'entité services de l'éditeur comme VIP Partner Microsoft Services.

Cette reconnaissance est légitimée par nos nombreuses expériences, interventions communautaires et de par nos certifications Gold sur l'ensemble des compétences Cloud Microsoft.

AZEO, votre partenaire Cloud d'excellence :

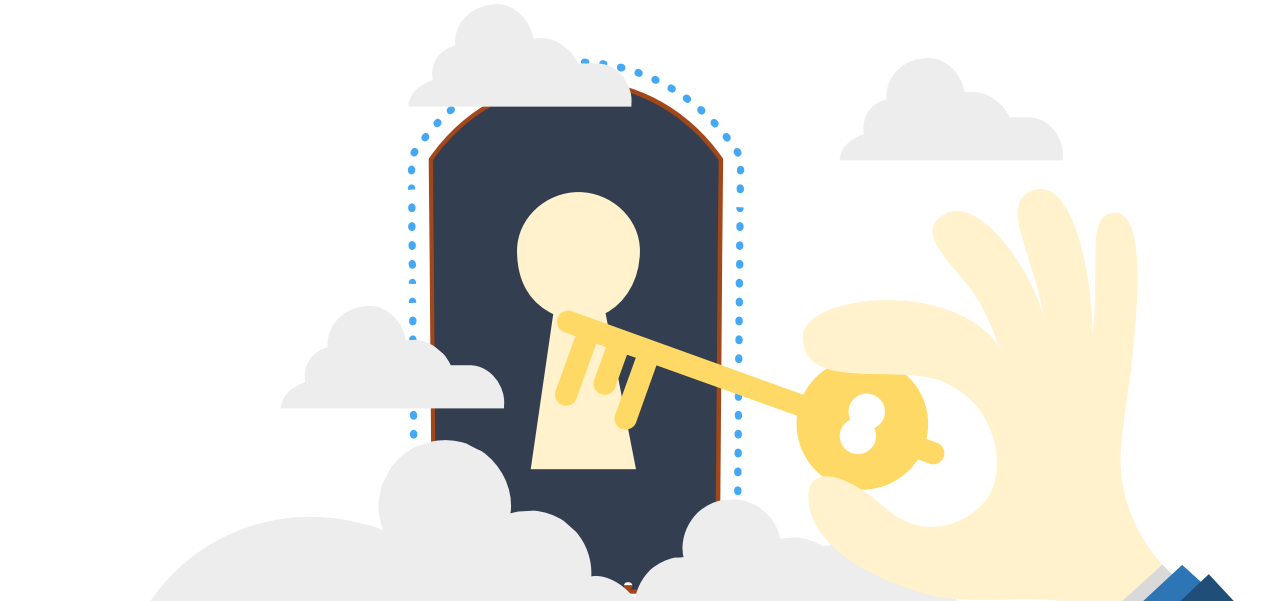
+230 Collaborateurs
dont 190 collaborateurs salariés
50 % en Ile-De-France
50 % en réseau national

4 Implantations
Boulogne-Billancourt, Bordeaux, Nantes, Toulouse

Stratégie Cloud First

Expertise Azure 360°

+32 %
des effectifs AZEO certifiés Azure



Microsoft Partner

Microsoft

- Gold Productivité cloud
- Gold Plateforme cloud
- Gold Plate-forme de données
- Gold DevOps
- Gold Gestion de la mobilité d'entreprise

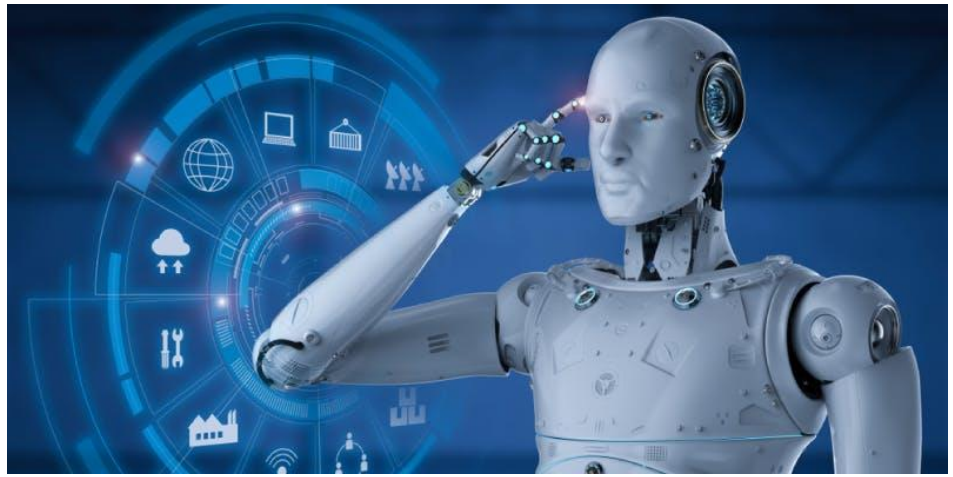
Agenda

- Quickly : what is Machine Learning ?
- Azure Machine Learning “new” Studio
- The MLOps approach
- Your new best friend : the Python SDK
  - 3 notebooks are available on my GitHub
- Examples of Azure Architecture
- Conclusion and go further

Artificial Intelligence versus Machine Learning

« If it's written in Python, it's…

Machine Learning»



« If it's written in Power Point, it's…

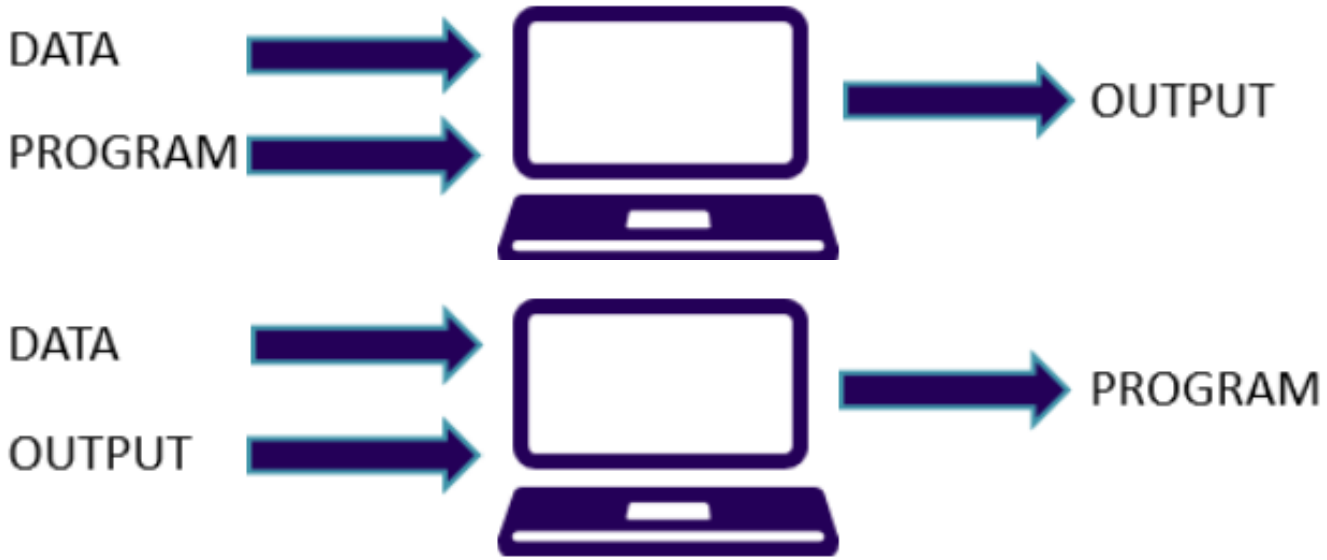
Artificial Intelligence»

Disclaimer :

this presentation is mainly about **industrialization** and less about *choice* or *optimization* of Machine Learning models.

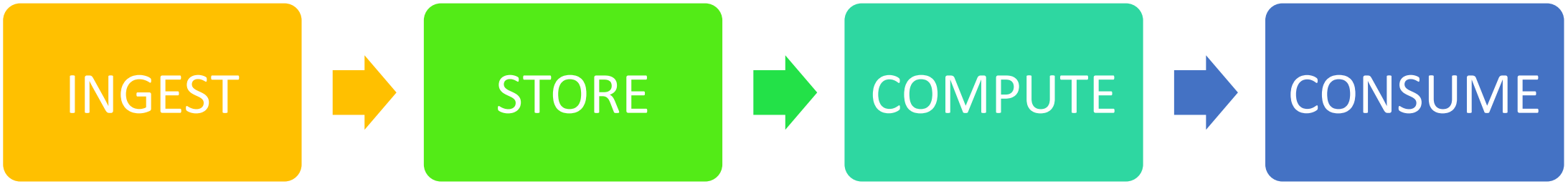
Supervised Machine Learning

Field of study that gives computers the ability to learn without being explicitly programmed (Arthur Samuel, 1959)



Have a look at this DPG webinar.

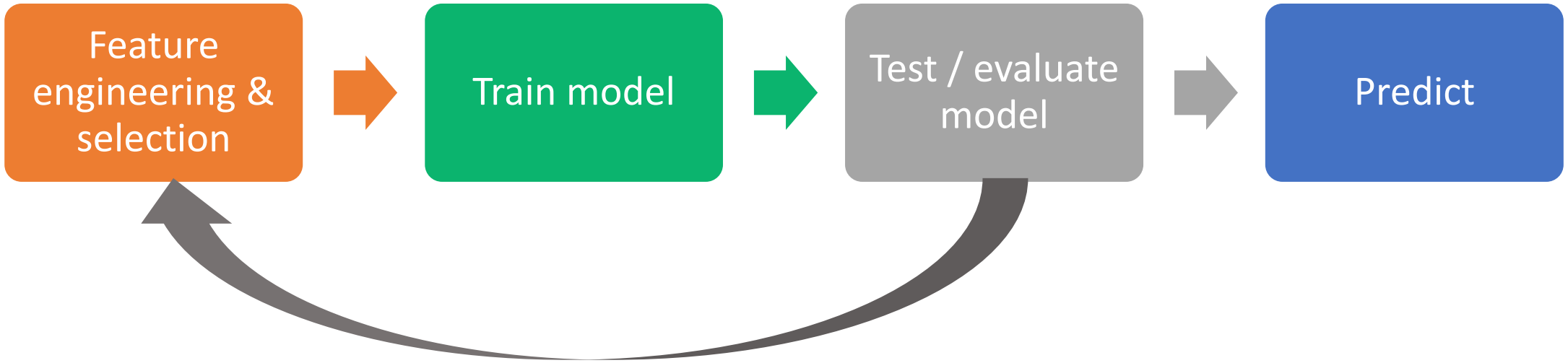
Data project : 4 common steps



Without the « consume » step(reporting, predictive models, apps…),
your data project is only **cost**.

Focus on the « compute » step in ML

Iterations on data preparation and training steps



We evaluate the model with **metrics**.

The « Predict » step is supported by a **deployment**.

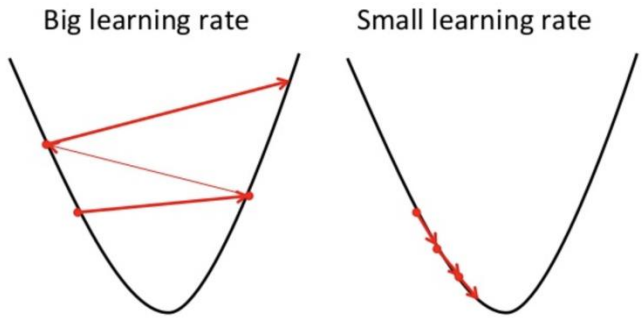
Parameters, hyperparameters, pickle…

The simplest model you have ever seen :

$$y = f(X) = aX + b$$

[a,b] are the **parameters** of the model.

Example of **hyperparameters** : learning rate



We will save the parameters values in a « **pickle** » file.

Others formats : ONNX, H5, RDS...

Why industrialize ?

- Check versioning
- Have backups for backtracking
- Planning the execution of treatments
- Monitor services
- Integrating organizational security
- Deploy on light terminals (edge)

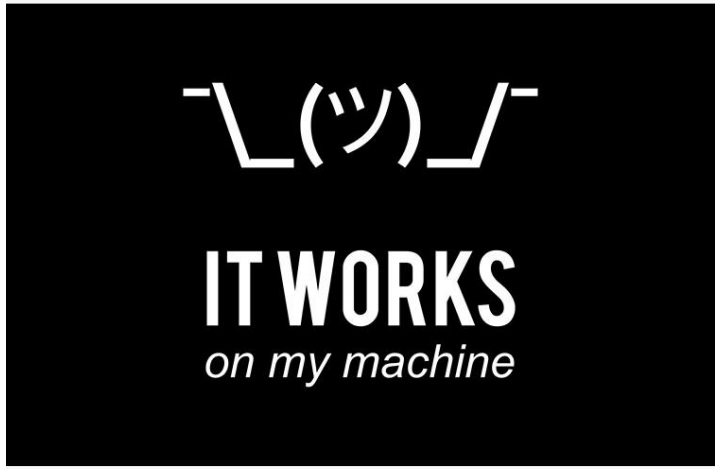
And why is it so difficult ?

“It works on my machine”

Says the Data Scientist

While training the model on a laptop

- with a (not so significant ?) part of the datas
- a whole night long
- with some unknown packages
- with dependencies of others project
- Without a distributed compute context



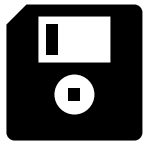
What do we need for production ?

A scheduler to plan

- Data cleansing
- Training / re-training of the model
- The forecast calculation (in batch mode)

A storage system to archive models

- Per algorithm, per version, per training dataset
- In a serialized (not proprietary) binary format



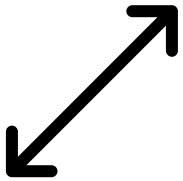
A tool for exposing the model

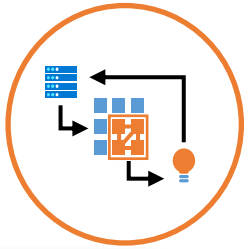
- Via REST API
- Secure access

Serving

Resources that can be deployed at scale

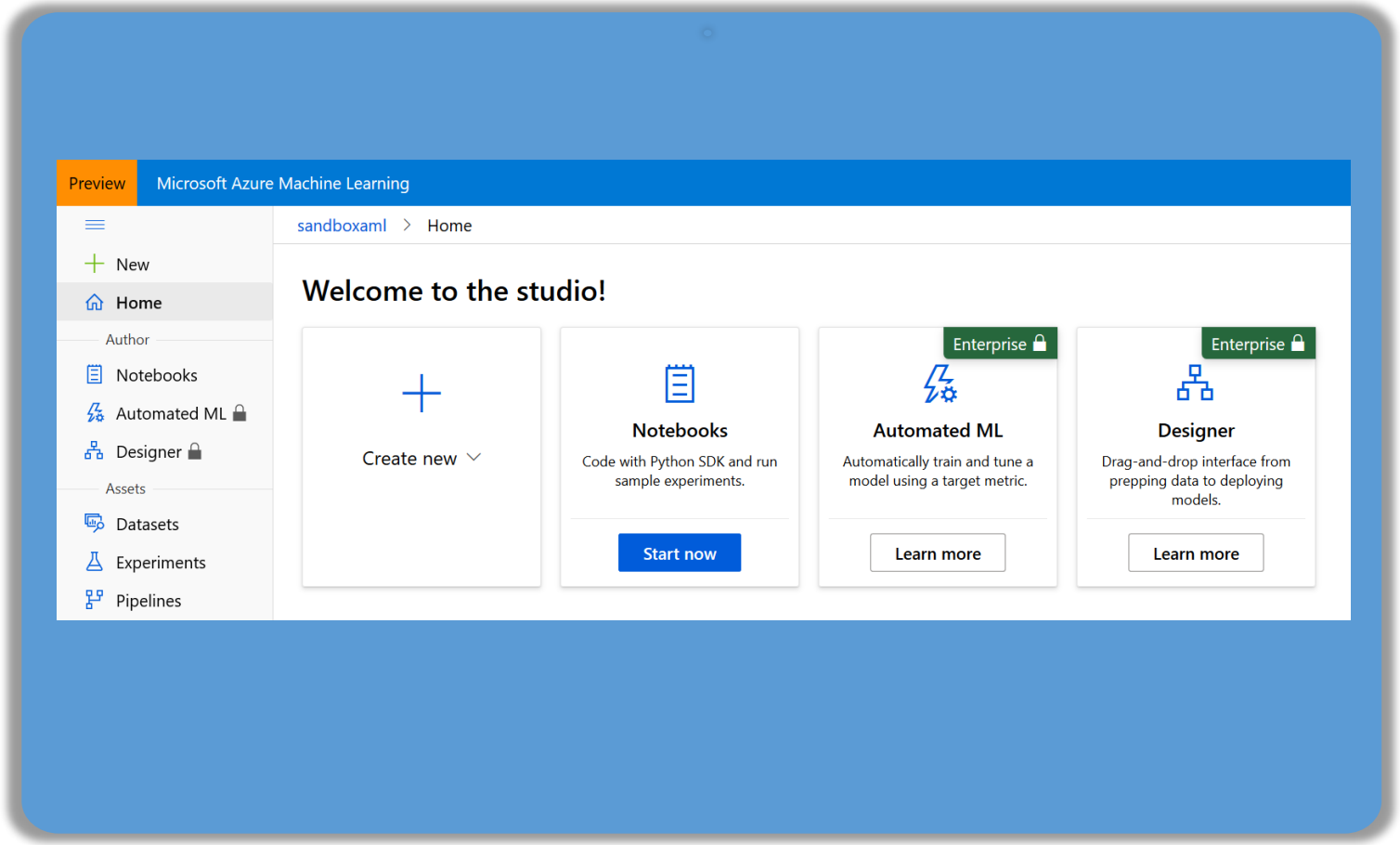
- With the help of the containers
- In the Cloud





Azure Machine Learning

The “new” Studio



Azure Machine Learning service

Set of Azure Cloud Services

Python SDK & R

That enables you to:

- ✓ Prepare Data
- ✓ Build Models
- ✓ Train Models
- ✓ Manage Models
- ✓ Track Experiments
- ✓ Deploy Models

Azure Machine Learning components

Experience
SDK, Notebooks, Drag-n-drop, Wizard

MLOps
Reproducible, Automatable, GitHub, CLI, REST

Datasets
Profiling, Drift, Labeling

Training
Experiments, Runs

Model Registry
Models, Images

Inferencing
Batch, Realtime

Compute
Jobs, Clusters, Instances

Azure IoT Edge
Security, Mgmt., Deployment

Cloud
CPU, GPU, FPGA

Edge
CPU, GPU, NPU

D E M O

Use case

Diabetes

Author: Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

Modelise the Y value by the others (regression)

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

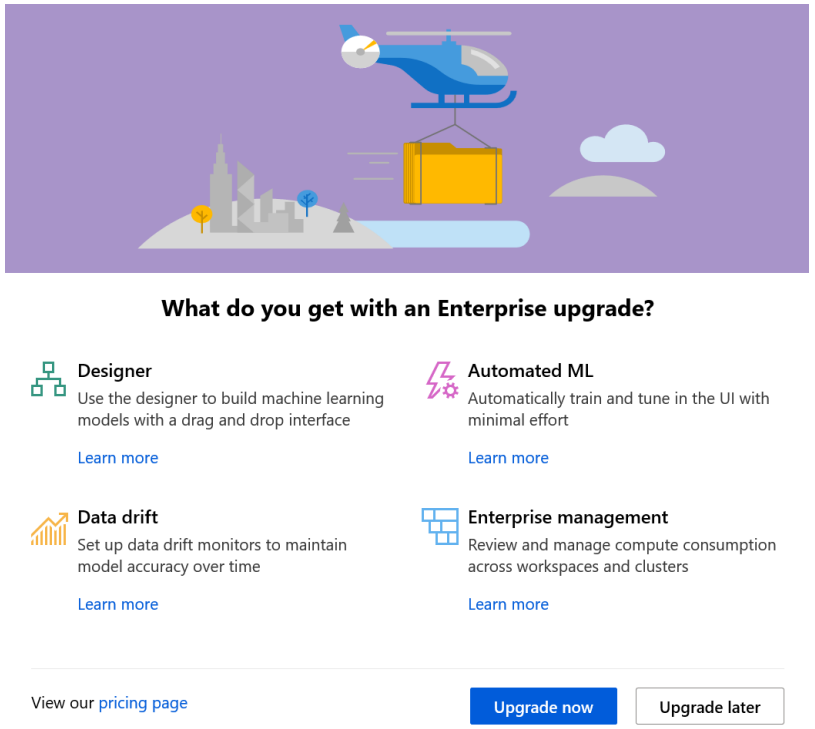
How to find the best model ?

What about Data Scientist's job ?

- 80% of the job is prepapring data
- 20% is complaining about the first 80%

Best model with Automated ML !

- As script (included in the Python SDK)
- With a UI (enterprise feature)



Automated ML with the Python SDK

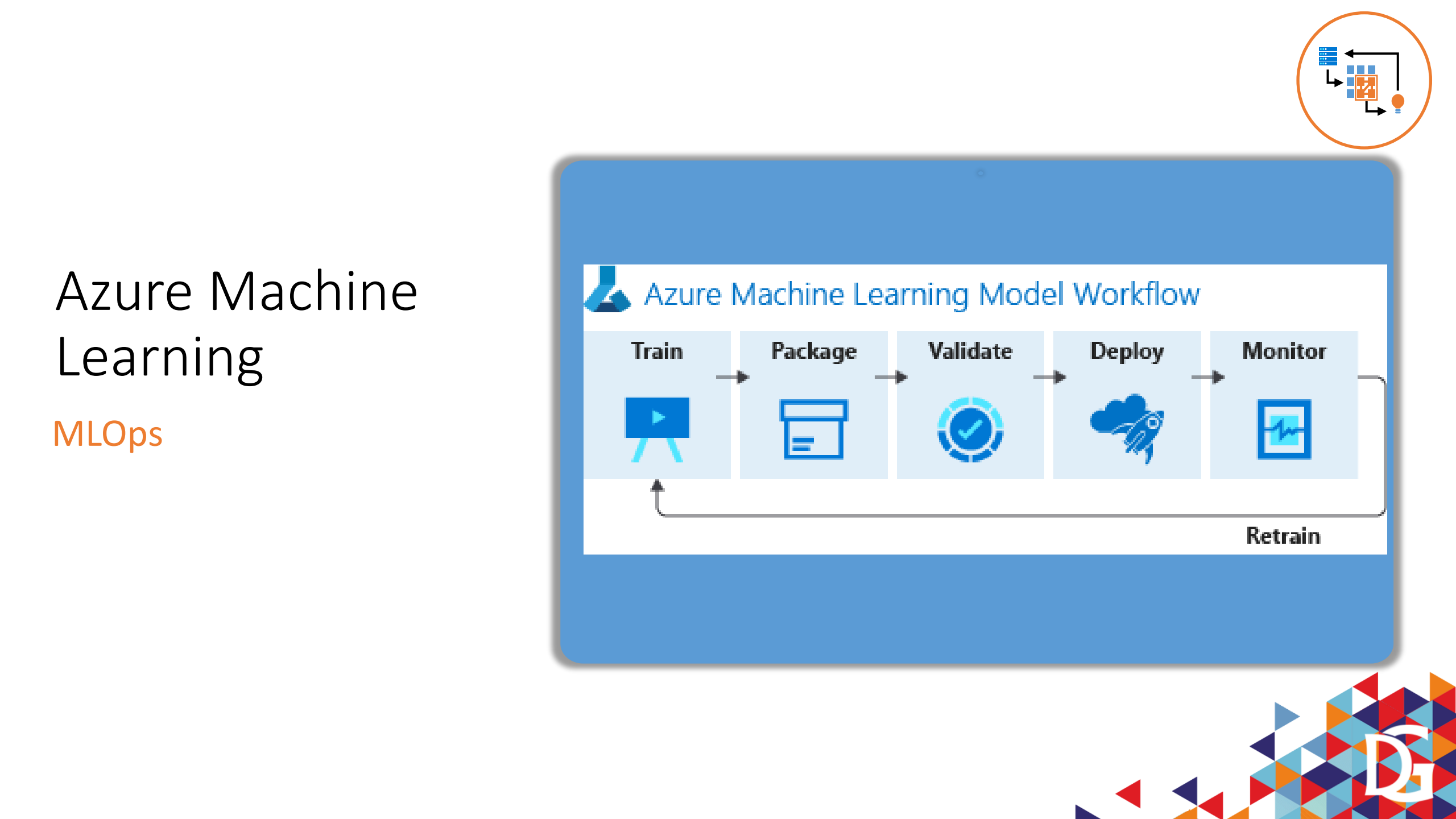
```
import logging
from azureml.train.automl import AutoMLConfig

# Define settings
automl_settings = {
    "iteration_timeout_minutes" : 10,
    "iterations" : 2,
    "primary_metric" : 'spearman_correlation',
    "preprocess" : True,
    "verbosity" : logging.INFO,
    "n_cross_validations": 5
}
```

Automated ML with the Python SDK

```
automl_config = AutoMLConfig(task = 'regression',
# could be classification or forecast for time series
                             debug_log = 'automated_ml_errors.log',
                             path = train_model_folder,
                             compute_target = aml_compute,
                             run_configuration = aml_run_config,
                             data_script = train_model_folder+"/get_data.py",
                             **automl_settings)

print("AutoML config created.")
```

Automated ML with the Python SDK

```
from azureml.train.automl.runtime import AutoMLStep

trainWithAutomlStep = AutoMLStep(
    name='AutoML_Regression',
    automl_config=automl_config,
    inputs=[output_split_train_x, output_split_train_y],
    allow_reuse=True,
    hash_paths=[os.path.realpath(train_model_folder)])

from azureml.core.experiment import Experiment
experiment = Experiment(ws, "taxi-experiment")
local_run = experiment.submit(automl_config, show_output=True)
```



Azure Machine Learning

MLOps

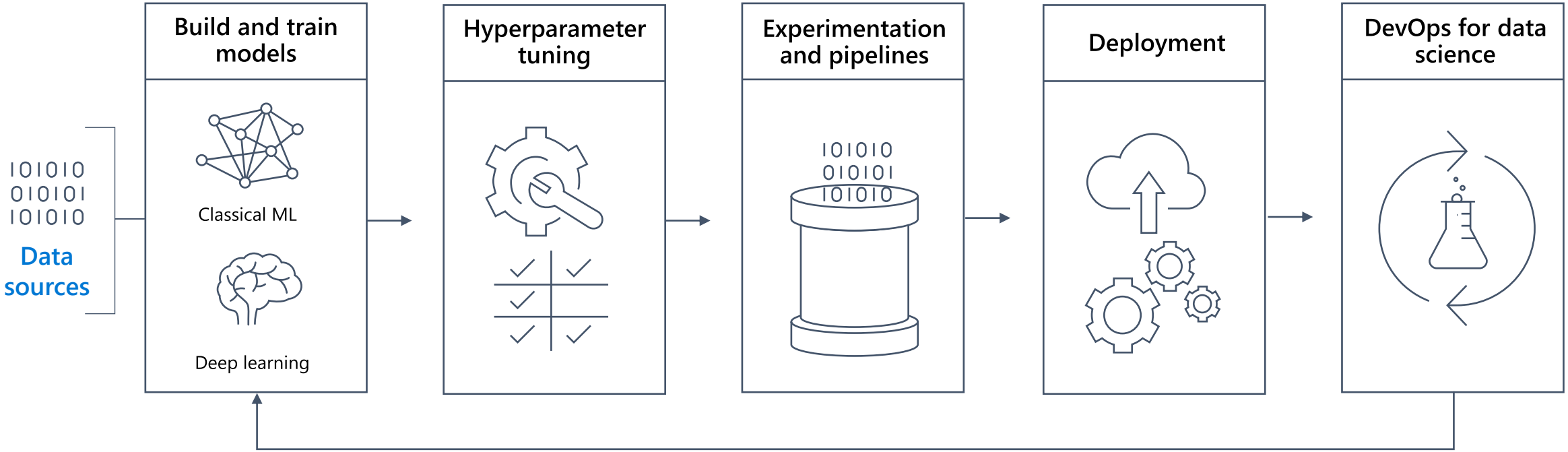
DevOps

Code reproducibility

Code testing

App deployment

Building blocks for a Data Science Project



DevOps

Code reproducibility

Code testing

App deployment

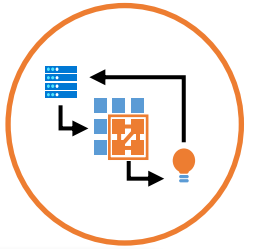
MLOps

Model reproducibility

Model validation

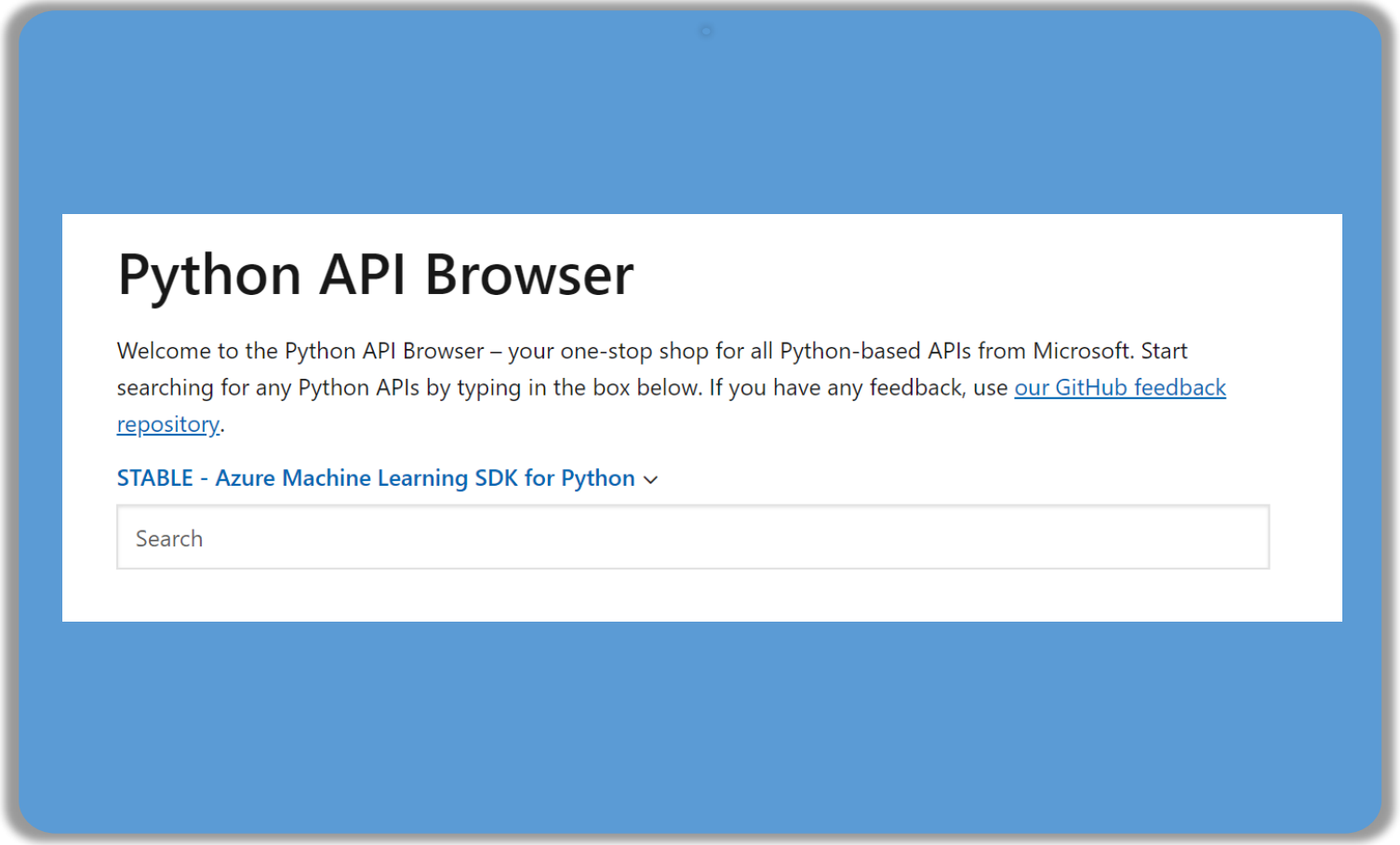
Model deployment

Model retraining



Azure Machine Learning

Python SDK



Azure Python SDK

Set of libraries that facilitate access to :

- Management components (Virtual Machine, Cluster, Image…)
- Runtime components (ServiceBus using HTTP, Batch, Monitor…)

Official GitHub repository :
https://github.com/Azure/azure-sdk-for-python
The full list of available packages and their latest version :
https://docs.microsoft.com/fr-fr/python/api/overview/azure/?view=azure-python

Installation :

```
!pip install --upgrade azureml-sdk
```

Or clone th GtiHub reporsitory :

```
git clone git://github.com/Azure/azure-sdk-for-python.git
cd azure-sdk-for-python
python setup.py install
```

Quick start tip (without Azure ML)

Home > rg-sandbox > New > Data Science Virtual Machine - Windows 2019

Data Science Virtual Machine - Windows 2019
Microsoft

Data Science Virtual Machine - Windows 2019 Save for later
Microsoft

Create | Start with a pre-set configuration

Overview | Plans

The **'Data Science Virtual Machine (DSVM)'** is a 'Windows Server 2019 with Containers' VM & includes popular tools for data exploration, analysis, modeling & development.

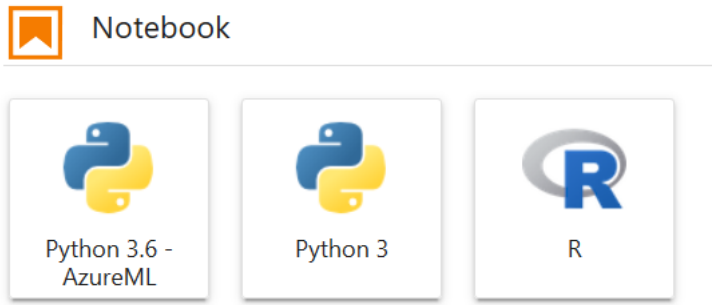
Highlights:

Anaconda Python
SQL Server 2019 Dev. Edition - With In-Database R and Python analytics
Microsoft Office 365 ProPlus BYOL - Shared Computer Activation
Julia
Jupyter notebooks
Visual Studio Community Ed. + Python, R & node.js tools
Power BI Desktop
Deep learning tools e.g. TensorFlow, Chainer
ML algorithm libraries e.g. xgboost, Vowpal Wabbit
Azure SDKs + libraries for various Azure Cloud offerings. Integration tools are included for:

1. Azure Machine Learning
2. Azure Data Factory
3. Stream Analytics
4. SQL Data Warehouse
5. Hadoop + Apache Spark (HDICluster)
6. Data Lake
7. Blob storage
8. ML & Data Science tutorials as Jupyter notebooks

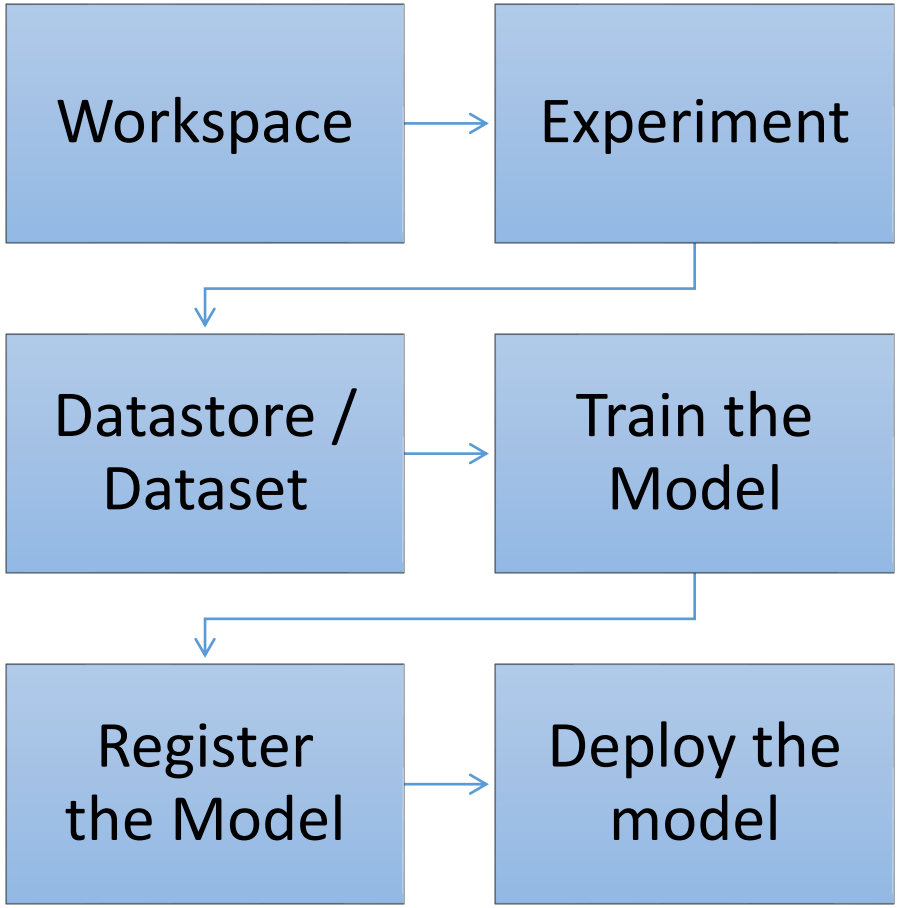
This image is pre-configured with Nvidia drivers, CUDA Toolkit, & cuDNN library for GPU workloads available if using NC class VM SKUs.

The Data Science VM is preconfigured with azureml and a Spark context.

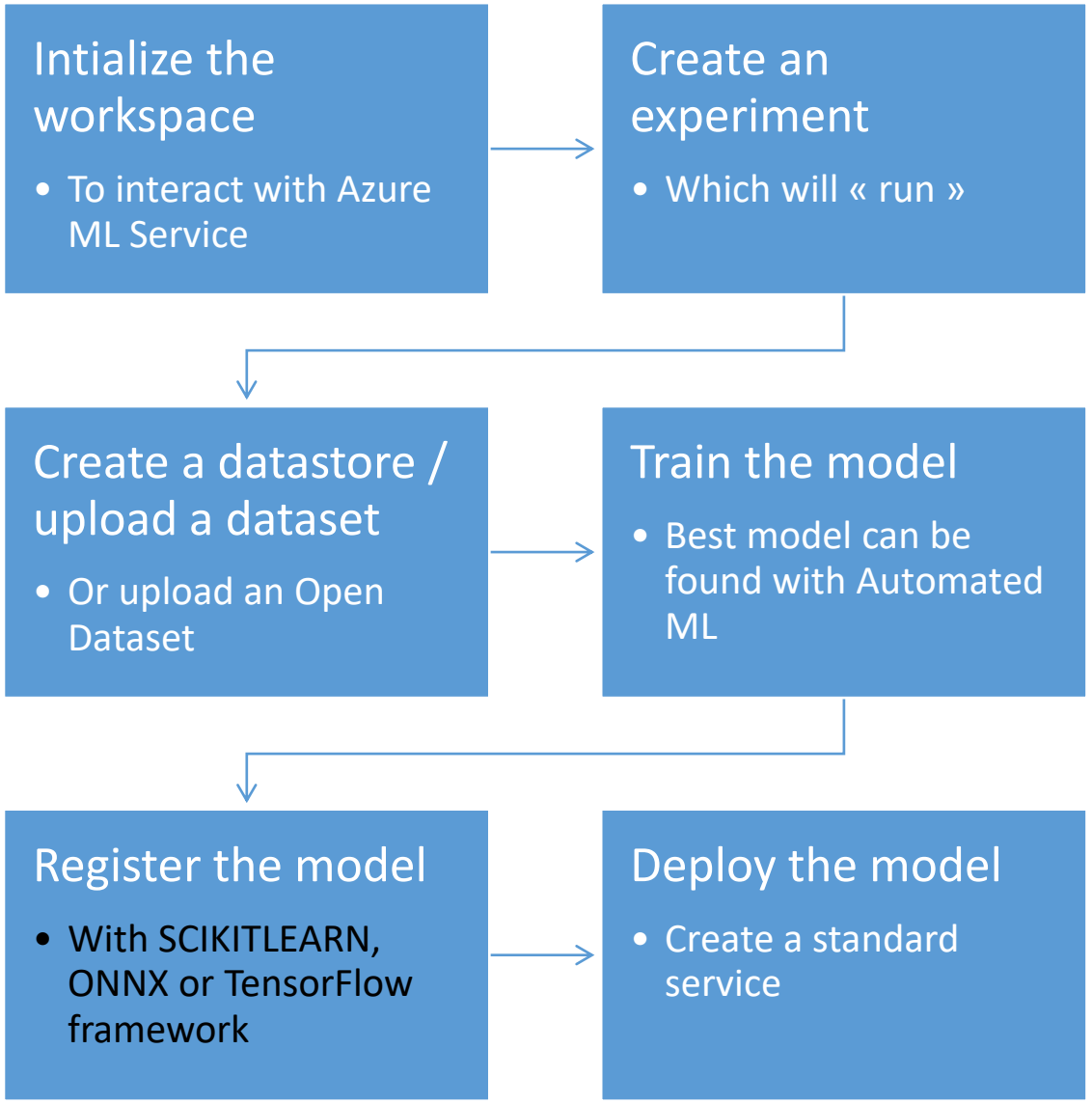


Main Objects

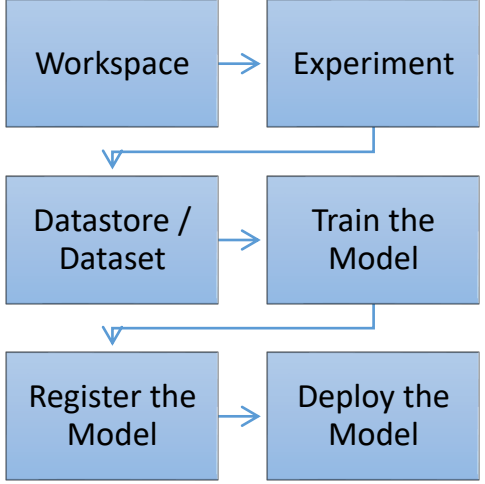
- Workspace
- Inside the Workspace
  - Datastore & Dataset
  - Compute target
  - Experiment
    - Run
  - Pipeline
  - Model
    - Environment
    - Estimator
  - Inference
  - Endpoint



ML **most simple** workflow with Python SDK

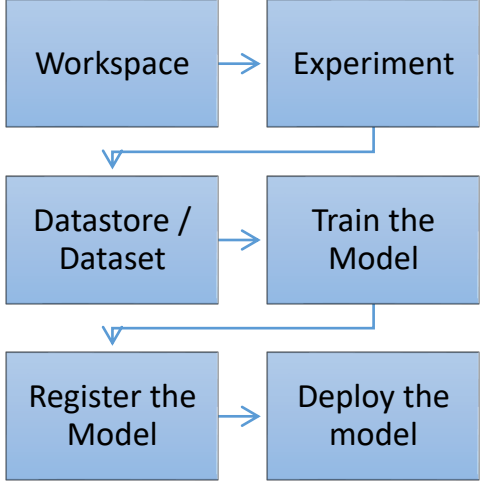


Intialize the workspace




```
import azureml.core
from azureml.core import Workspace

# check core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)

# load workspace configuration from the config.json file in the current folder.
ws = Workspace.from_config()
print(ws.name, ws.location, ws.resource_group, ws.location, sep='\t')
```

Create an experiment



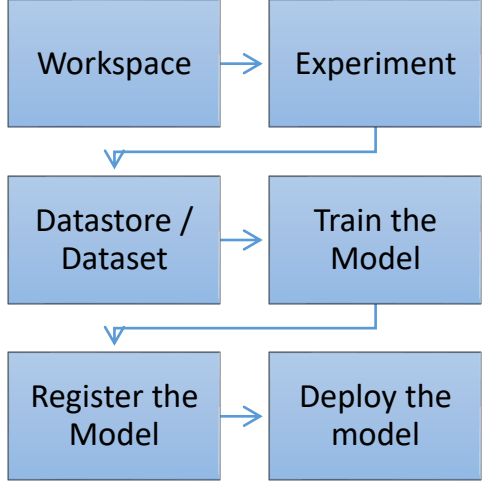

```
experiment_name = 'my_first_experiment'
```

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

```
exp
```

Name	Workspace	Report Page	Docs Page
diabetes_exp	sandboxaml	Link to Azure Machine Learning studio	Link to Documentation

Create a datastore



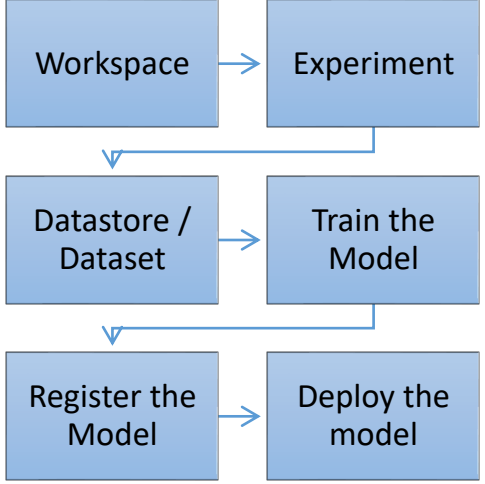

```
blob_datastore_name='MyBlobDatastore'
# Storage account name
account_name=os.getenv("BLOB_ACCOUNTNAME_62", "<my-account-name>")
# Name of Azure blob container
container_name=os.getenv("BLOB_CONTAINER_62", "<my-container-name>")
# Storage account key
account_key=os.getenv("BLOB_ACCOUNT_KEY_62", "<my-account-key>")

blob_datastore = Datastore.get(ws, blob_datastore_name) print("Found Blob
Datastore with name: %s" % blob_datastore_name)
```

Supported Azure Storage services

- Azure Blob Container
- Azure File Share
- Azure Data Lake
- Azure Data Lake Gen2
- Azure SQL Database
- Azure Database for PostgreSQL
- Azure Database for MySQL
- Databricks File System

We need an ODBC connector !

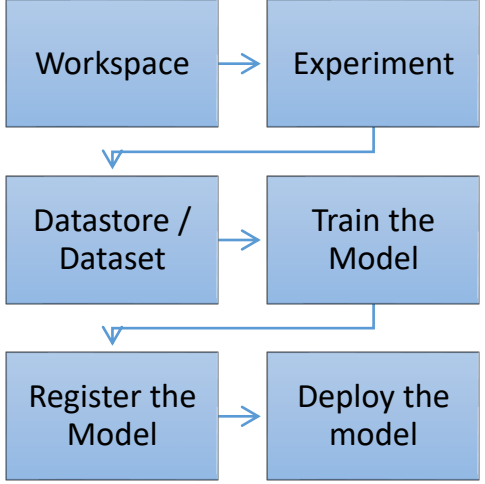
Upload data from datastore




```
from azureml.core import Dataset

dataset = Dataset.Tabular.from_delimited_files(path = [(datastore,
'train-dataset/tabular/iris.csv')])

# Convert the Dataset object to a (in-memory) pandas dataframe
df = dataset.to_pandas_dataframe()
```

We will speak about parallel compute later

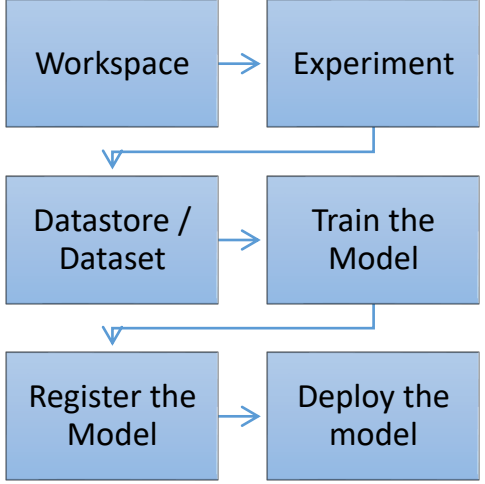
Train the model *(nothing from azureml !)*




```
# Separate train and test data
from sklearn.model_selection import train_test_split

X = diabetes.drop(target, axis=1)
y = diabetes["Y"].values.reshape(-1,1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.linear_model import LinearRegression

# Set up the model
model = LinearRegression()
# Use fit
model.fit(X_train, y_train)
```

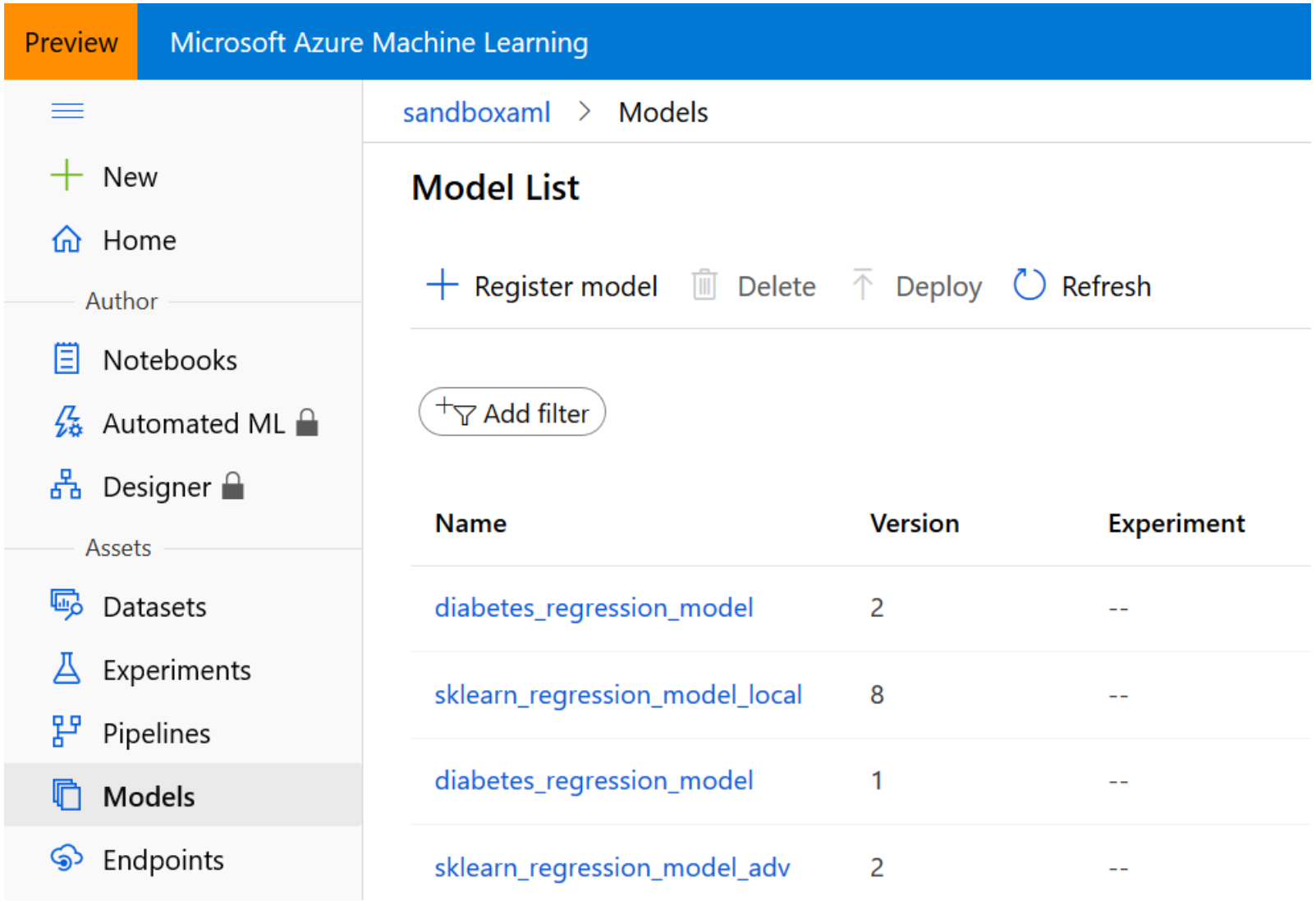
We train on the compute instance !

Register the model

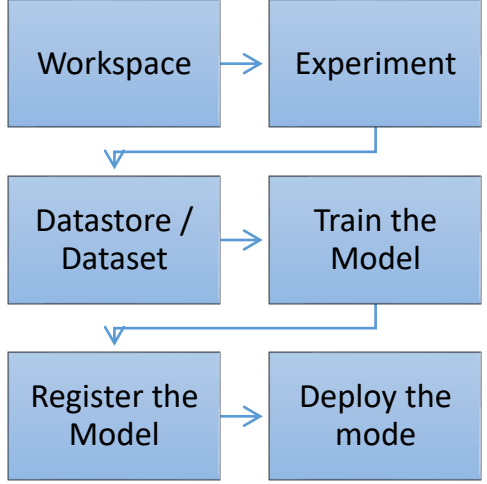



```
from azureml.core.model import Model

model = Model.register(model_path="diabetes_regression_model.pkl",
                       model_name="diabetes_regression_model",
                       tags={'area': "diabetes", 'type': "regression"},
                       description="Ridge reg to predict diabetes",
                       workspace=ws)
```

We can find the model in the studio

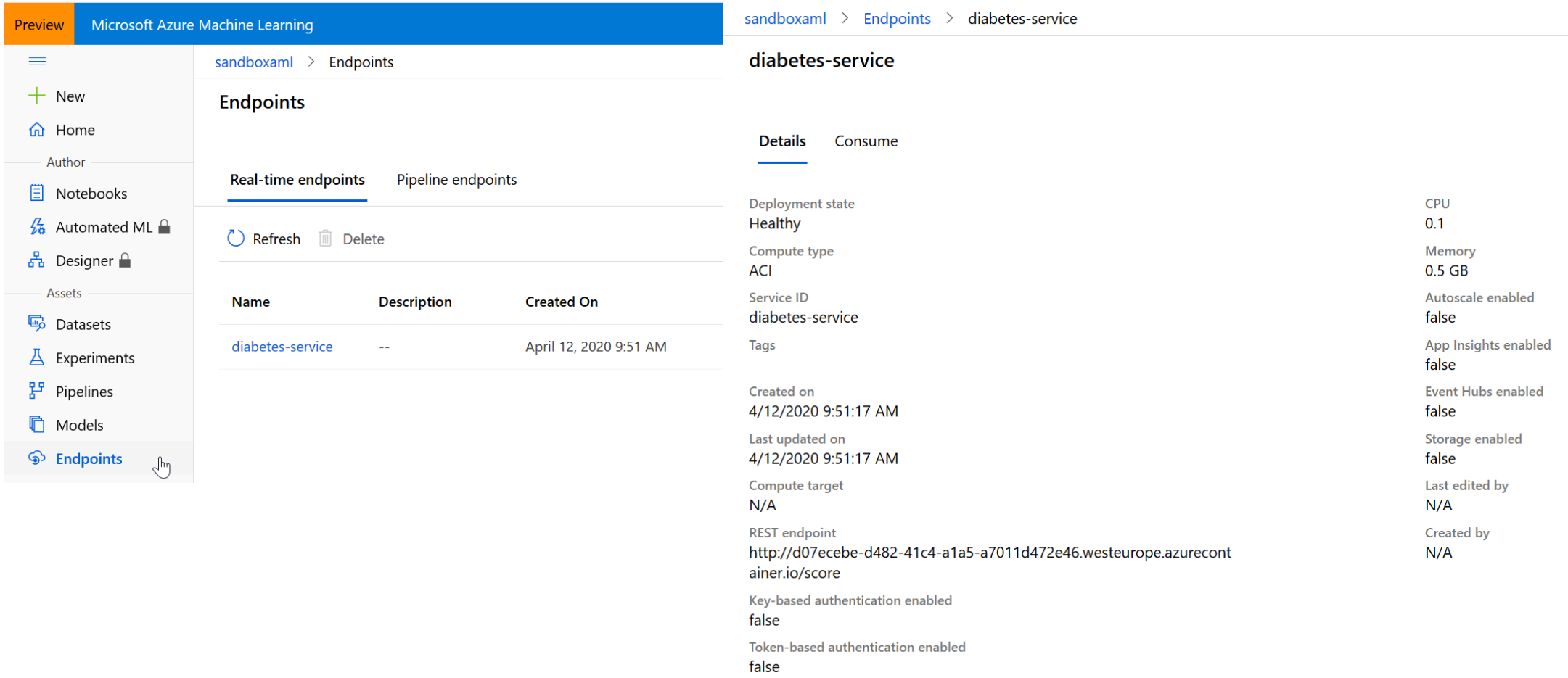


Deploy the model to the cloud (ACI)

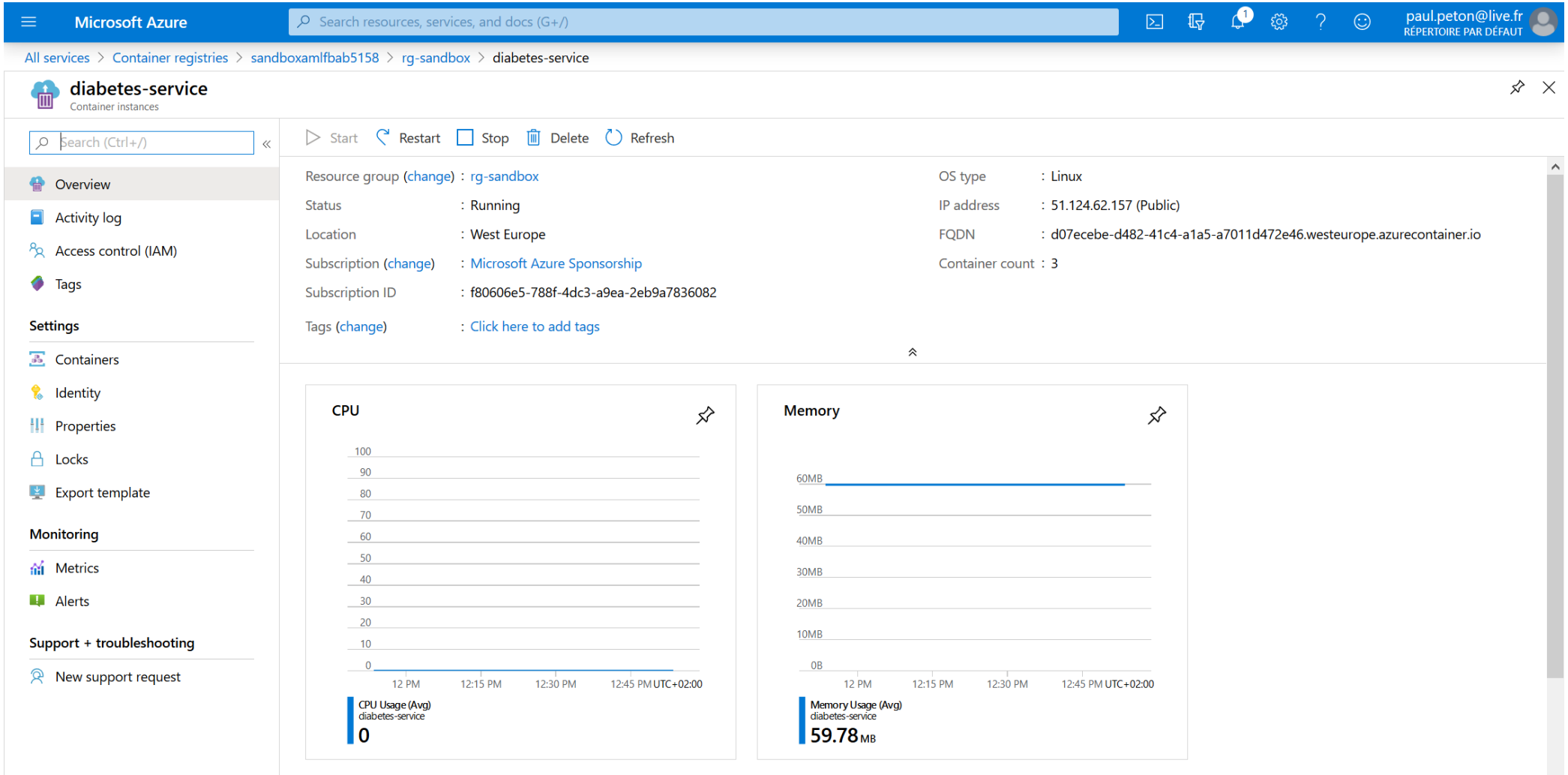



```
from azureml.core import Webservice
from azureml.exceptions import WebserviceException

service_name = 'diabetes-service'

# Remove any existing service under the same name.
try:
    Webservice(ws, service_name).delete()
except WebserviceException:
    pass

service = Model.deploy(ws, service_name, [model])
service.wait_for_deployment(show_output=True)
```

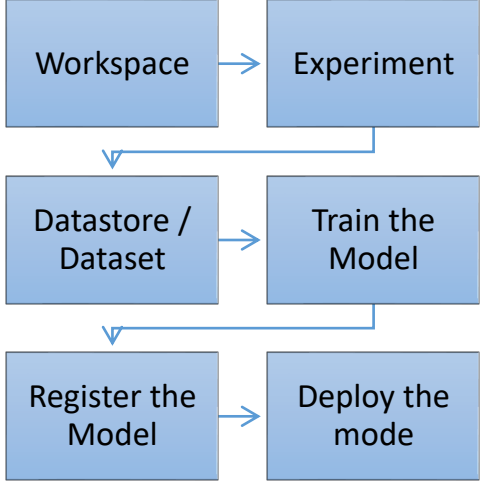
We can see the endpoint in the studio



We can find the container on the Azure portal



Test the model

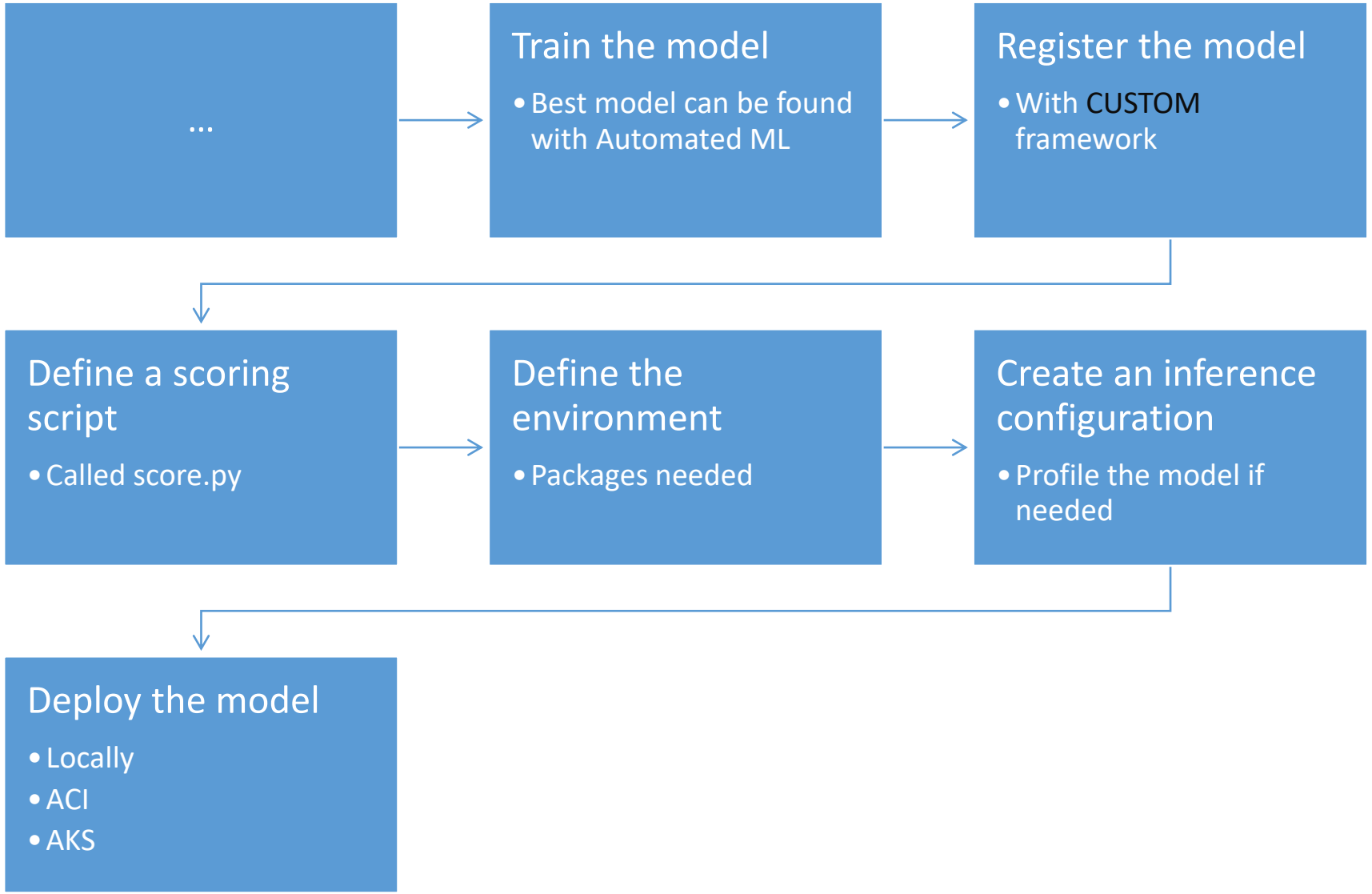



```
import json

input_payload = json.dumps({
    'data': [
        [59, 2, 32.1, 101.0, 157, 93.2, 38.0, 4.0, 4.8598, 87]
    ],
    'method': 'predict'
# If you have a classification model, you can get probabilities by changing
this to 'predict_proba'.
})

output = service.run(input_payload)
print(output)
```

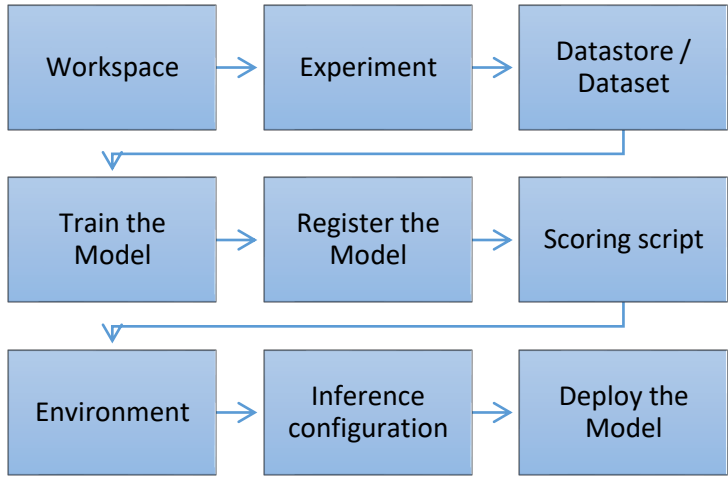
Where to deploy ?

- In a local Docker environment
- To the “cloud” : Azure Container Instance
  - For developer scenario
- To Azure Kubernetes Service
  - For production scenario
- To Azure Function
  - With the Docker Image
  - Serverless
- To Azure Webapp
  - With the Docker Image
  - Cheaper than AKS

ML **custom** workflow with Python SDK



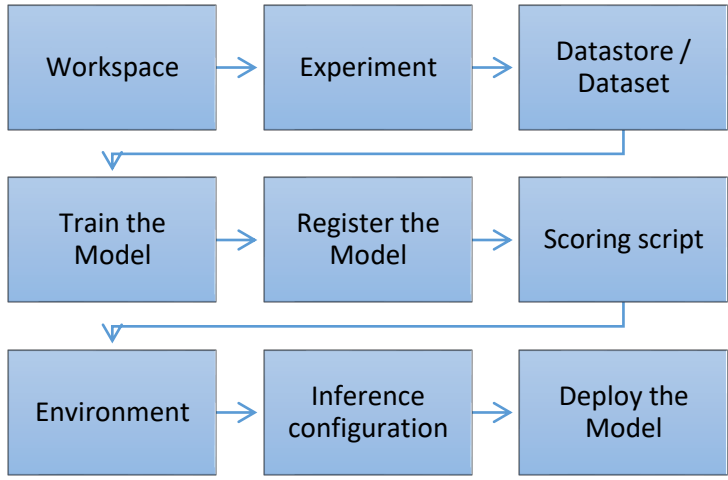
Scoring script



Code Python 3.6 - AzureML

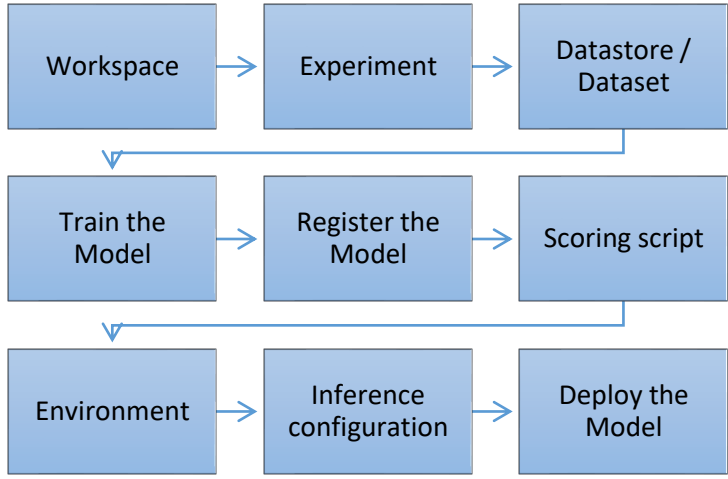
```
[ ]: %%writefile score.py

import joblib
import numpy as np
import os

from inference_schema.schema_decorators import input_schema, output_schema
from inference_schema.parameter_types.numpy_parameter_type import NumpyParameterType


# The init() method is called once, when the web service starts up.
#
# Typically you would deserialize the model file, as shown here using joblib,
# and store it in a global variable so your run() method can access it later.
def init():
    global model

    # The AZUREML_MODEL_DIR environment variable indicates
    # a directory containing the model file you registered.
    model_filename = 'diabetes_regression_model.pkl'
    model_path = os.path.join(os.environ['AZUREML_MODEL_DIR'], model_filename)

    model = joblib.load(model_path)


# The run() method is called each time a request is made to the scoring API.
#
# Shown here are the optional input_schema and output_schema decorators
# from the inference-schema pip package. Using these decorators on your
# run() method parses and validates the incoming payload against
# the example input you provide here. This will also generate a Swagger
# API document for your web service.
@input_schema('data', NumpyParameterType(np.array([[59, 2, 32.1, 101.0, 157, 93.2, 38.0, 4.0, 4.8598, 87]])))
@output_schema(NumpyParameterType(np.array([151.000])))
def run(data):
    # Use the model object loaded by init().
    result = model.predict(data)

    # You can return any JSON-serializable object.
    return result.tolist()
```

Environment definition




```
# Set up the (compute target) environnement and save in a YAML file
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies

env = Environment("diabetes_env")
env.docker.enabled = True
# Two methods : Conda or pip
env.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn',
                                                                          'pandas',
                                                                          'numpy',
                                                                          'matplotlib'
                                                                         ])

env.python.conda_dependencies.add_pip_package("inference-schema[numpy-support]")
env.python.conda_dependencies.save_to_file(".", "diabetes_env.yml")
```

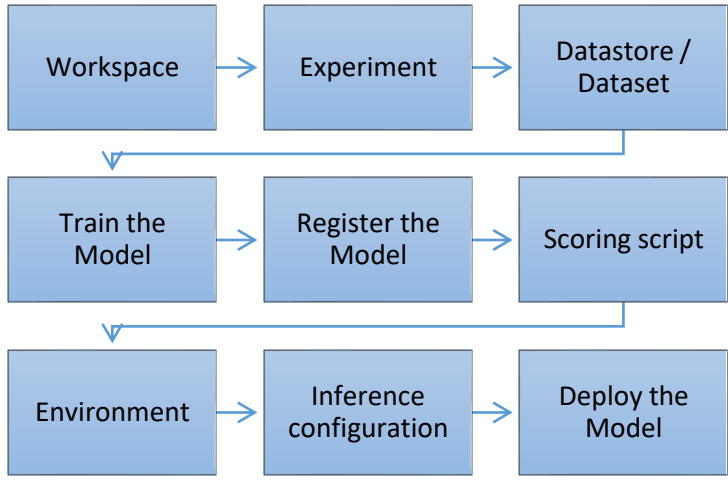
Create the inference configuration




```
from azureml.core.model import InferenceConfig
```

```
inference_config = InferenceConfig(entry_script='score.py',
                                   environment=env)
```

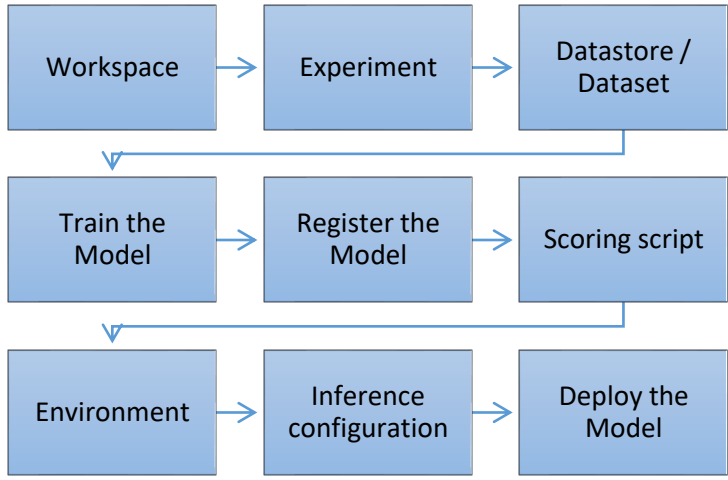
Deploy the model (1/2)




```
from azureml.core import Webservice
from azureml.core.webservice import AciWebservice
from azureml.exceptions import WebserviceException

service_name = 'diabetes-custom-service'

# Remove any existing service under the same name.
try:
    Webservice(ws, service_name).delete()
except WebserviceException:
    pass
aci_config = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1)
```

Find the best deploy configuration with the profiling object

Deploy the model (2/2)




```
service = Model.deploy(workspace=ws,
                       name=service_name,
                       models=[model],
                       inference_config=inference_config,
                       deployment_config=aci_config)

service.wait_for_deployment(show_output=True)
```

Local scenario (Docker Web Service)

Deploy Model as a Local Docker Web Service

Make sure you have Docker installed and running.

Note that the service creation can take few minutes.

NOTE:

The Docker image runs as a Linux container. If you are running Docker for Windows, you need to ensure the Linux Engine is running:

```
# PowerShell command to switch to Linux engine
& 'C:\Program Files\Docker\Docker\DockerCli.exe' -SwitchLinuxEngine
```

Entrée [14]:

```
from azureml.core.webservice import LocalWebservice

# This is optional, if not provided Docker will choose a random unused port.
deployment_config = LocalWebservice.deploy_configuration(port=6789)
deployment_config
```

Out[14]: <azureml.core.webservice.local.LocalWebserviceDeploymentConfiguration at 0x7f214a2bcf60>

Entrée [15]:

```
local_service = Model.deploy(ws, "test", [model], inference_config, deployment_config)

local_service.wait_for_deployment()
```

Local scenario (Docker Web Service)

Entrée [13]:

```
inference_config = InferenceConfig(entry_script='score.py', environment=environment)
# if cpu and memory_in_gb parameters are not provided
# the model will be profiled on default configuration of
# 3.5CPU and 15GB memory
profile = Model.profile(workspace = ws,
            profile_name = 'profile-%s' % datetime.now().strftime('%m%d%Y-%H%M%S'),
            models = [model],
            inference_config = inference_config,
            input_data=None,
            input_dataset=sample_request_data_diabetes,
            cpu=1.0,
            memory_in_gb=0.5,
            description=None)

profile.wait_for_completion(True)
details = profile.get_details()
```

```
Runnin
g.....................................................................................................
...
Succeeded
```

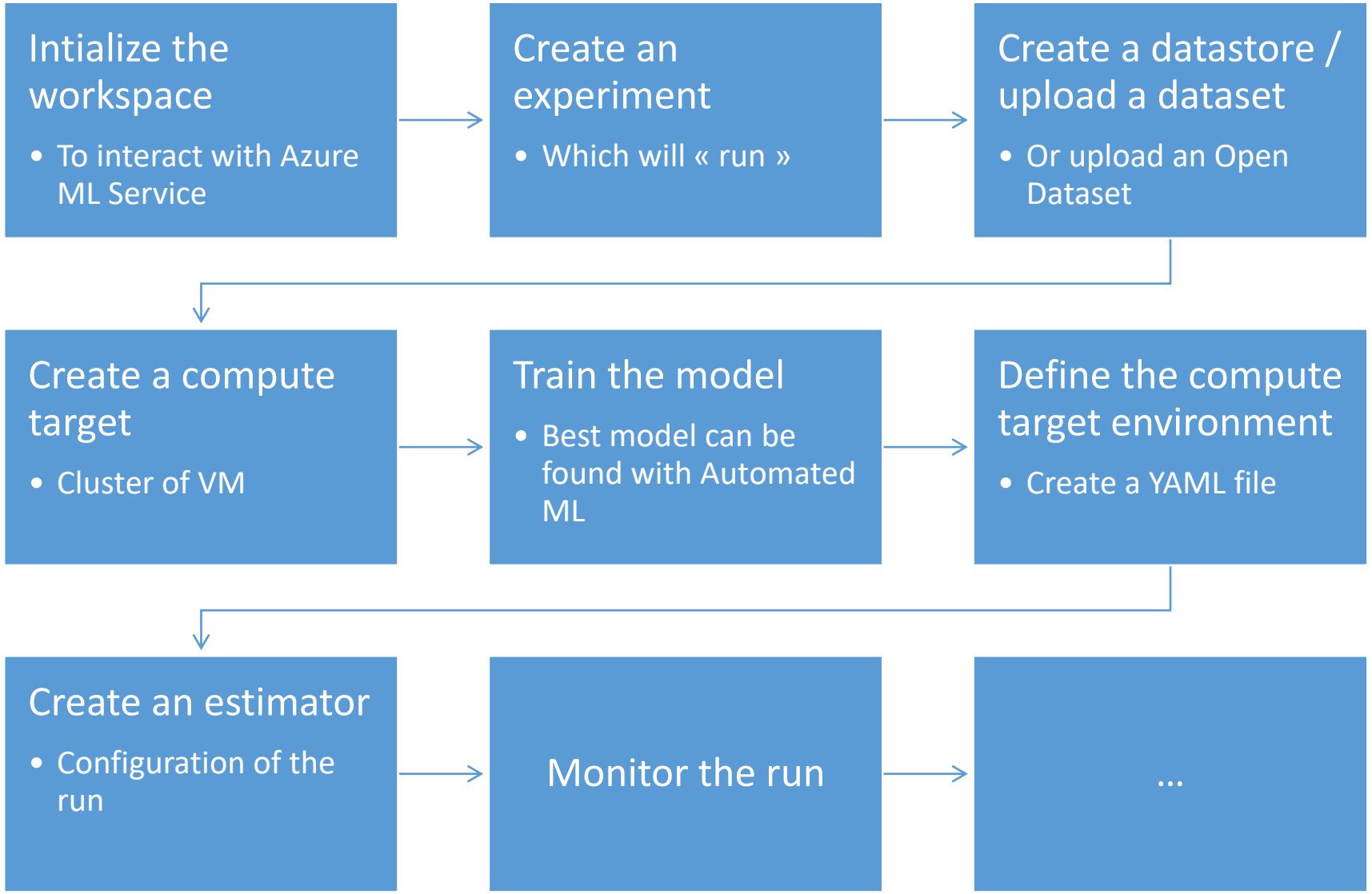
Entrée [15]:
```
local_service = Model.deploy(ws, "test", [model], inference_config, deployment_config)

local_service.wait_for_deployment()
```

```
Downloading model sklearn_regression_model_local:9 to /tmp/azureml_k8ox8npm/sklearn_regression_model_local/9
Generating Docker build context.
Package creation Succeeded
Logging into Docker registry sandboxamlfbab5158.azurecr.io
Logging into Docker registry sandboxamlfbab5158.azurecr.io
Building Docker image from Dockerfile...
Step 1/5 : FROM sandboxamlfbab5158.azurecr.io/azureml/azureml_ac4d351d5f766e7bbeffaa4abbcb62d6
 ---> 0130837010dc
Step 2/5 : COPY azureml-app /var/azureml-app
 ---> 3af0166199af
Step 3/5 : RUN mkdir -p '/var/azureml-app' && echo eyJhY2NvdW50Q29udGV4dCI6eyJzdWJzY3JpcHRpb25JZCI6ImY4MDYwNmU1LTc4
OGYtNGRjMy1hOWVhLTJlYjlhNzgzNjA4MiIsInJlc291cmNlR3JvdXBOYW1lIjoicmctc2FuZGJveCIsImFjY291bnROYW1lIjoic2FuZGJveGFtbCI
sIndvcmtzcGFjZUlkIjoiMjlkZWIxZWUtYzUxNS00MzFiLTg0OWItNTRkYjU2YWQxMDI0In0sIm1vZGVscyI6e30sIm1vZGVsc0luZm8iOnt9fQ== |
base64 --decode > /var/azureml-app/model_config_map.json
 ---> Running in 1e51f9df2768
 ---> 46c4a74804b6
Step 4/5 : RUN mv '/var/azureml-app/tmp4_wakvz9.py' /var/azureml-app/main.py
 ---> Running in 530c403dee51
 ---> 217c98f0d7a3
Step 5/5 : CMD ["runsvdir","/var/runit"]
 ---> Running in c63062d5fd5d
 ---> 5617ed35a23a
Successfully built 5617ed35a23a
Successfully tagged test:latest
Container (name:focused_sanderson, id:71c5ca5bc6cbdfd3f75f08fe1ad6e82d0b32c90fe353cf00fd20ff8cd9b74ec9) cannot be k
illed.
Container has been successfully cleaned up.
Image sha256:b00be1abc9ba9ff2c4d1f10ba39d5ff72b63065bc71f2345b4e07956d6089c49 successfully removed.
Starting Docker container...
Docker container running.
Checking container health...
Local webservice is running at http://localhost:6789
```

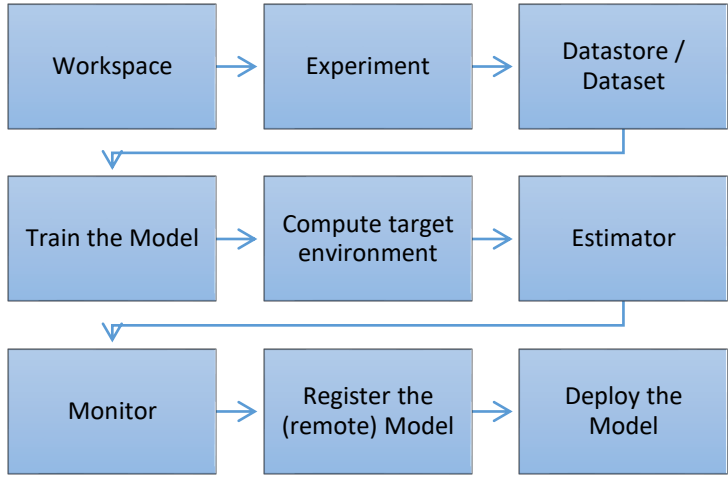
Entrée [16]:
```
print('Local service port: {}'.format(local_service.port))
```

```
Local service port: 6789
```

ML « **big data** » workflow with Python SDK



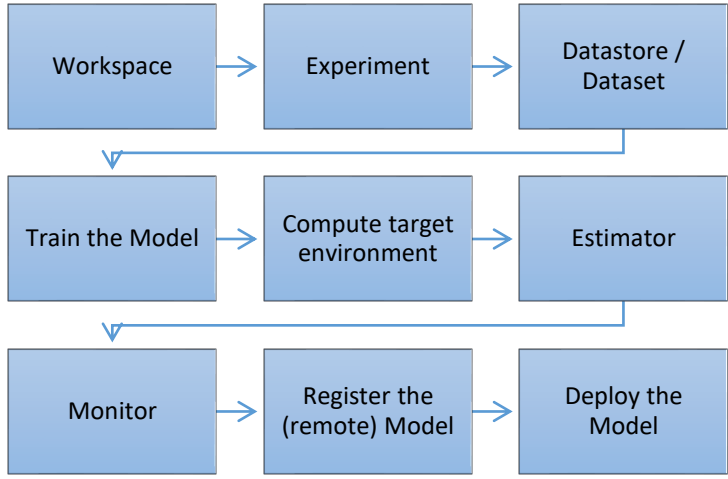
Create a remote compute target




```
# Compute cluster creation.
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

cpu_cluster_name = "myComputeCluster"
# Verify that cluster does not exist already
try:
    cpu_cluster = ComputeTarget(workspace=ws, name=cpu_cluster_name)
    print(" Cluster already exists")
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D2_V2',
                                                           min_nodes=0, max_nodes=4)
    cpu_cluster = ComputeTarget.create(ws, cpu_cluster_name, compute_config)

cpu_cluster.wait_for_completion(show_output=True, min_node_count=0, timeout_in_minutes=10)
```

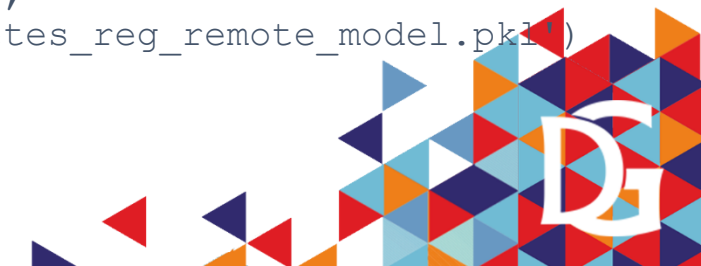
min_nodes=0 *The target will be deleted when no jobs are running on it.*

Train on a remote target cluster

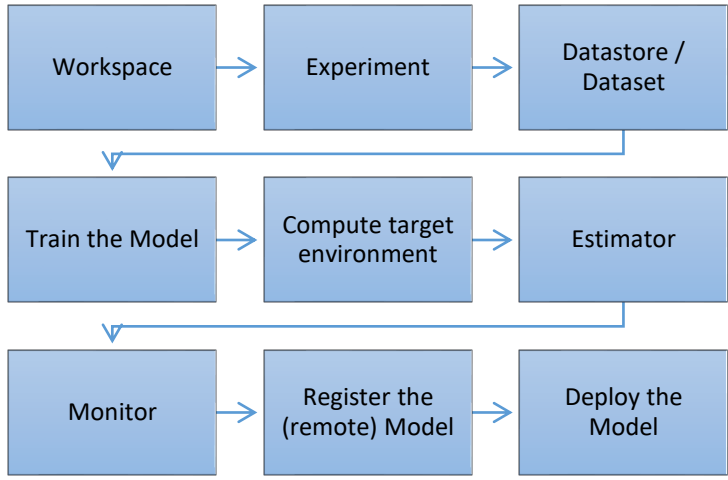



```
[ ]: %%writefile $script_folder/train.py

import argparse
import os
import numpy as np
import glob

from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.externals import joblib
import pickle

from azureml.core import Run
from azureml.core import Dataset
from utils import load_data

from azureml.core import Workspace

ws = Workspace(subscription_id, resource_group, workspace_name)

parser = argparse.ArgumentParser()
parser.add_argument('--regularization', type=float, dest='reg', default=0.5, help='regularization strength')
args = parser.parse_args()

# load train and test set into numpy arrays
diabetes_dataset = Dataset.get_by_name(ws, name='diabetes')
diabetes = diabetes_dataset.to_pandas_dataframe().drop("Path", axis=1)
target = "Y"
X = diabetes.drop(target, axis=1)
y = diabetes["Y"].values.reshape(-1,1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# get hold of the current run
run = Run.get_context()

print('Train a Ridge regression model with regularization strength of', args.reg)
model = Ridge(alpha=args.reg, solver="auto", random_state=42)
model.fit(X_train, y_train)

print('Predict the test set')
y_hat = model.predict(X_test)

# calculate score on the prediction
score = model.score(X_test, y_test)
print('Score is ', score)

run.log('regularization strength', np.float(args.reg))
run.log('score', np.float(score))

os.makedirs('outputs', exist_ok=True)
# note file saved in the outputs folder is automatically uploaded into experiment record
joblib.dump(value=model, filename='outputs/diabetes_reg_remote_model.pkl')
```

```
parser = argparse.ArgumentParser()

parser.add_argument('--regularization',
type=float, dest='reg', default=0.5,
help='regularization strength')

args = parser.parse_args()


model = Ridge(alpha=args.reg, solver="auto",
random_state=42)


# note file saved in the outputs folder is
automatically uploaded into experiment record

joblib.dump(value=model,
filename='outputs/diabetes_reg_remote_model.pkl')
```



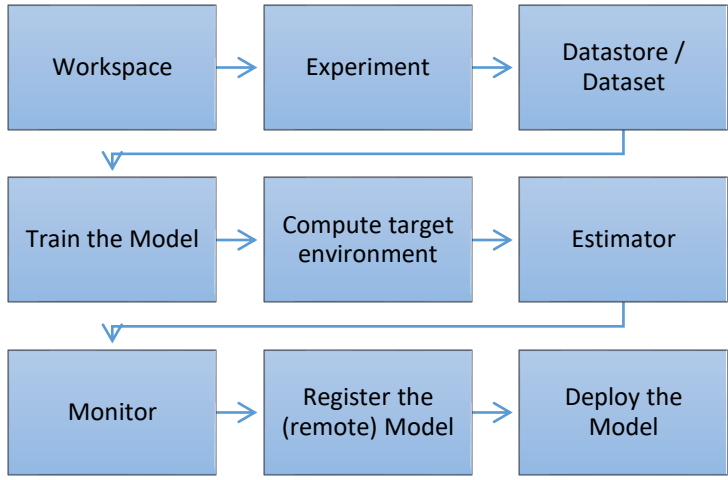
Create an estimator




```
from azureml.train.estimator import Estimator

script_params = {
'--regularization': 0.5
}

est = Estimator(source_directory=script_folder,
                script_params=script_params,
                compute_target=compute_target,
                environment_definition=env,
                entry_script='train.py')

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

Monitor a run

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

# specify show_output to True for a verbose log
run.wait_for_completion(show_output=True)
```



Jupyter widget

Jupyter widget

Watch the progress of the run with a Jupyter widget. Like the run submission, the widget is asynchronous and provides live updates every 10-15 seconds until the job completes.

Entrée [13]:

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

Run Properties

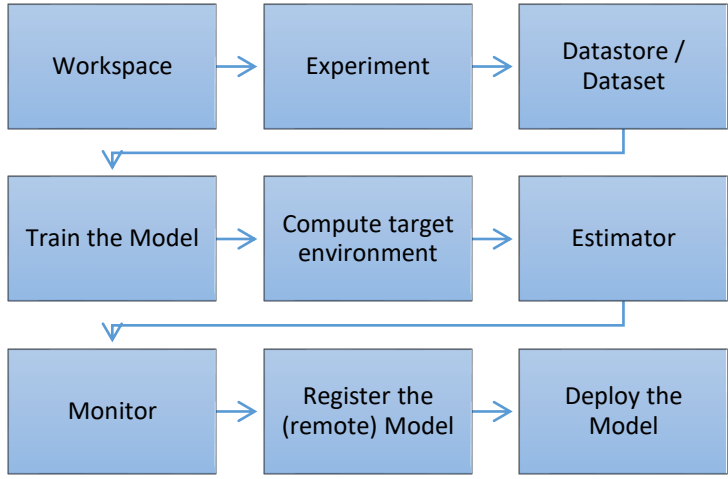
Status	Preparing
Start Time	14/04/2020 07:56:04
Duration	0:00:42
Run Id	sklearn-mnist_1586843745_876530d6
Arguments	N/A

Output Logs azureml-logs/20_image_build_log.txt ☑ Auto-switch

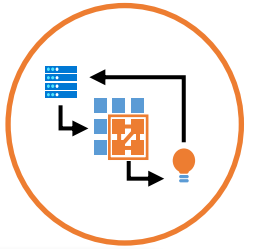
```
 ---> 06556a281e2a
Step 3/15 : RUN mkdir -p $HOME/.cache
 ---> Running in 7a8a53c6937a
Removing intermediate container 7a8a53c6937a
 ---> 38a6055afb2e
Step 4/15 : WORKDIR /
 ---> Running in d7fcad727a99
Removing intermediate container d7fcad727a99
 ---> 779b4dbd7050
Step 5/15 : COPY azureml-environment-setup/99brokenproxy /etc/apt/apt.conf.d/
```

Click here to see the run in Azure Machine Learning studio

Register the (remote) model

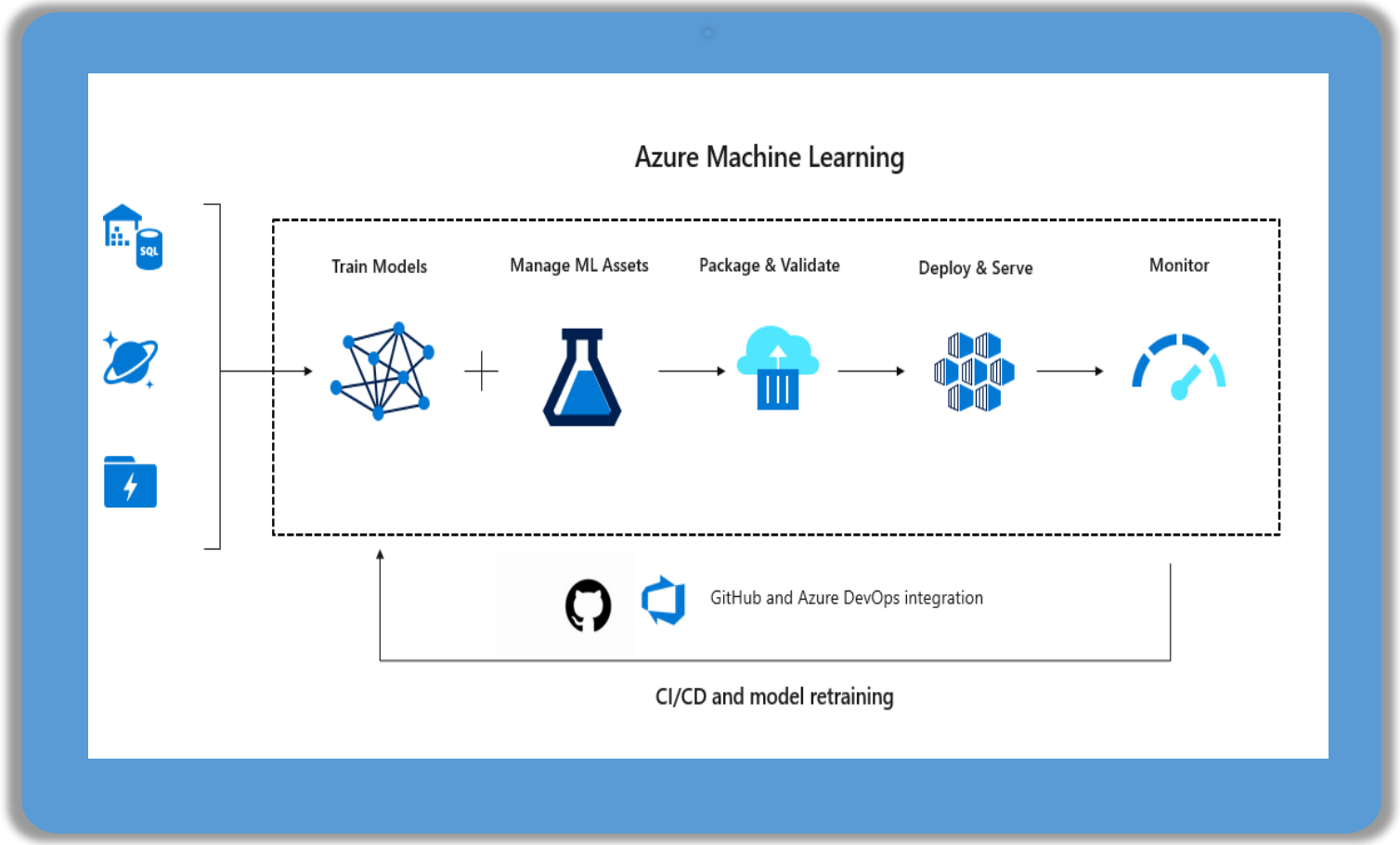



```
print(run.get_file_names())

# register model
model = run.register_model(model_name='sklearn_mnist',
model_path='outputs/sklearn_mnist_model.pkl')

print(model.name, model.id, model.version, sep='\t')
```

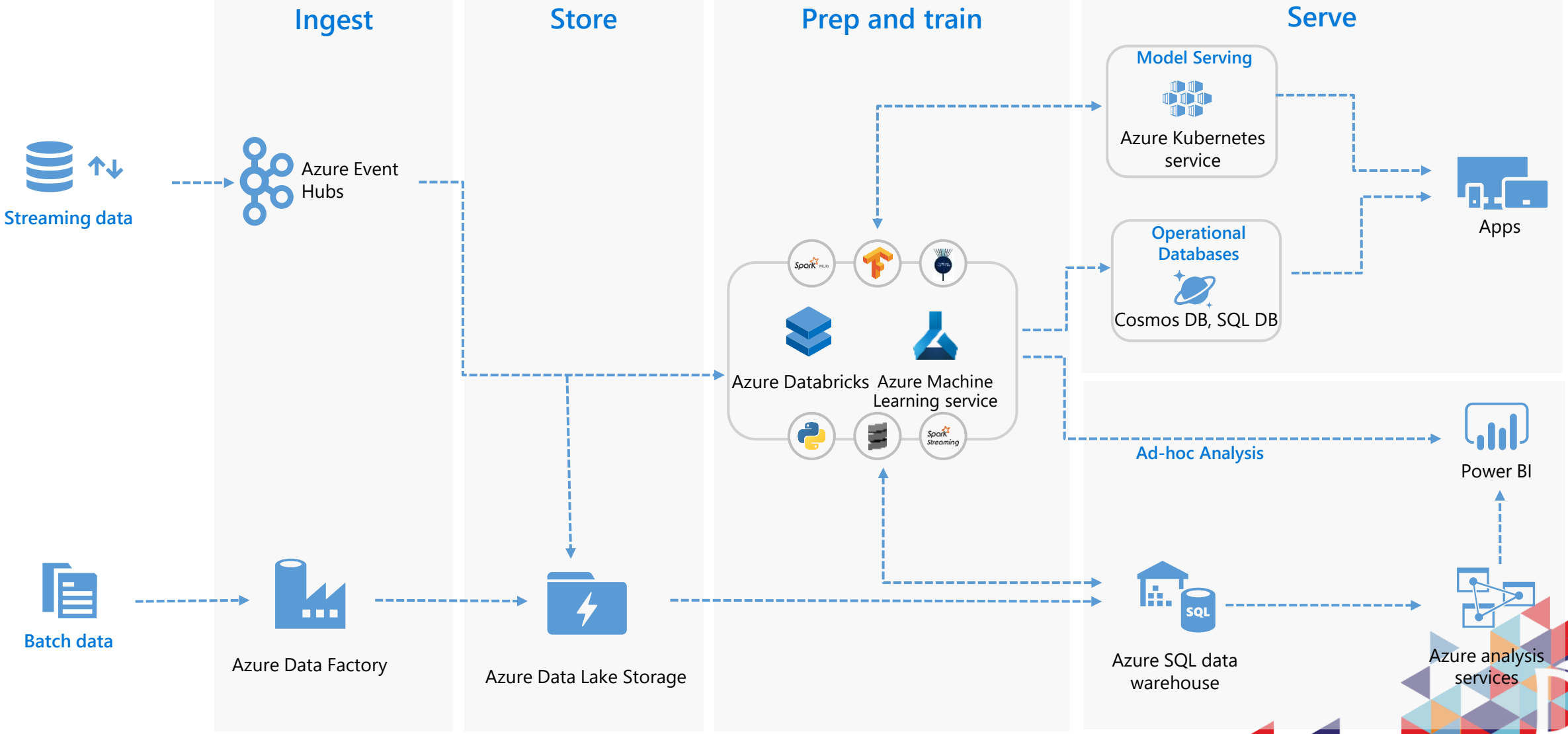


Azure Machine Learning

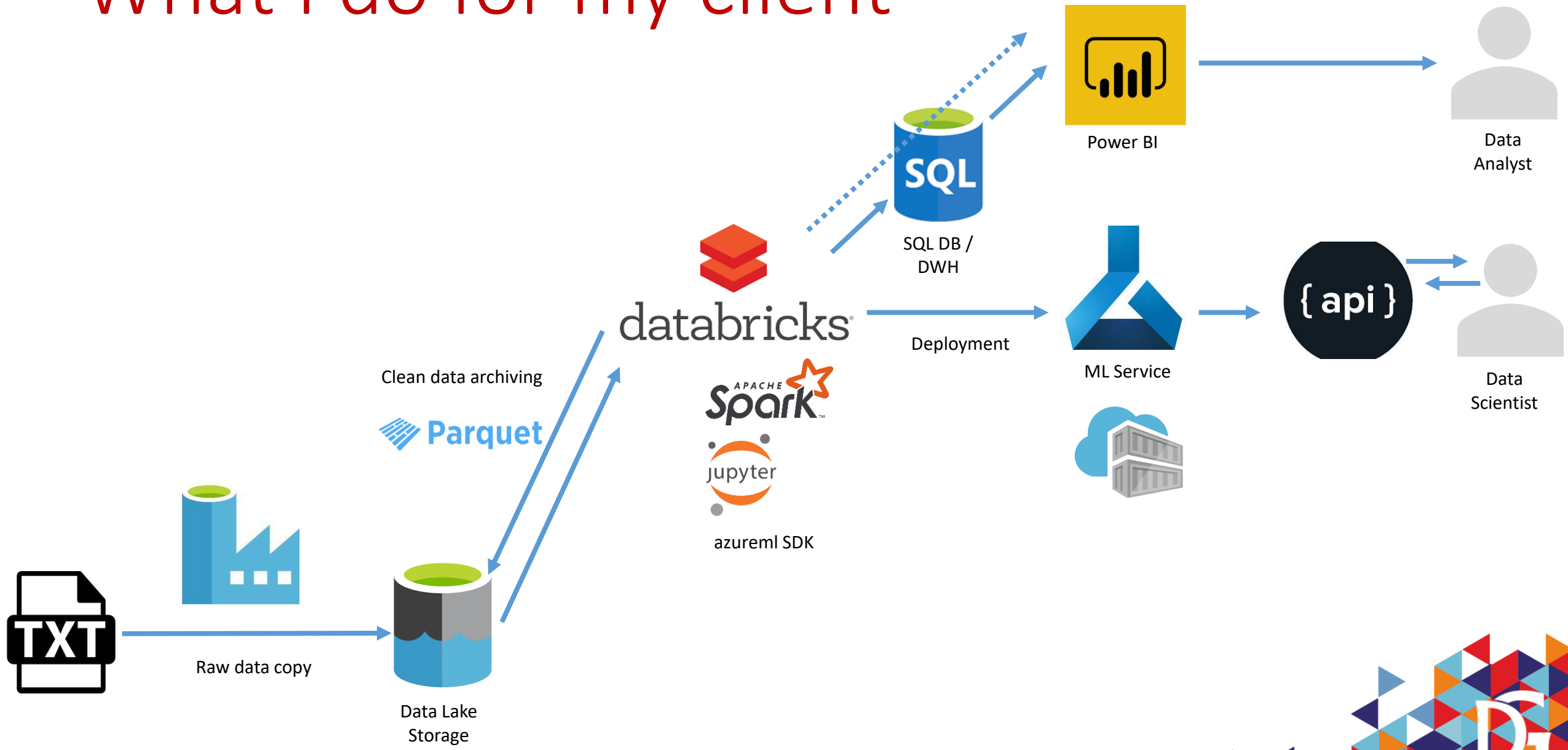
Industry leading MLOps



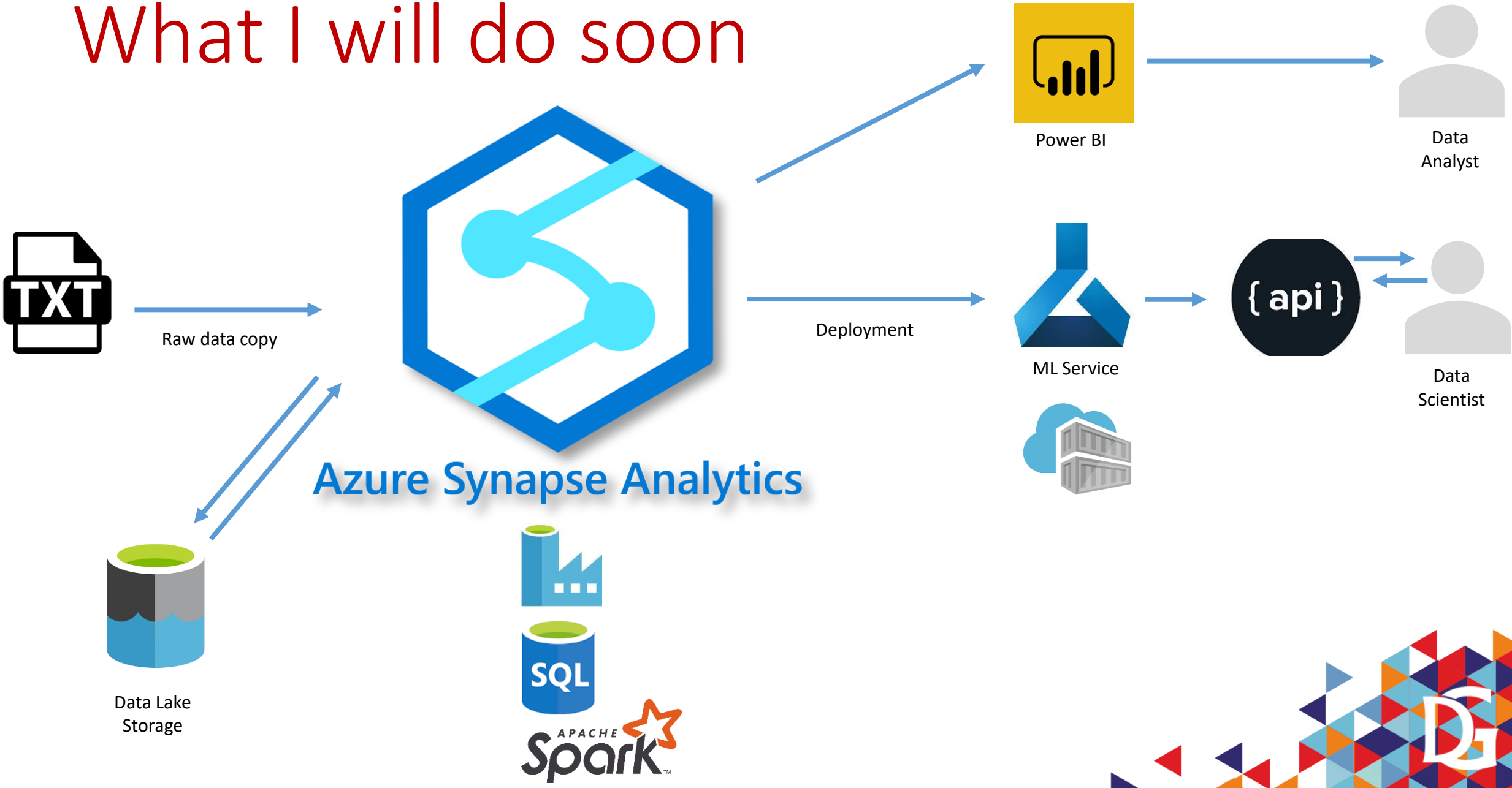
Recommended architecture e2e ML



What I do for my client



What I will do soon



My opinion about deploying with Azure ML

Strengths to build on :

- Notebooks are everywhere (except in the heart of code purists), locally and in the cloud.
- Services communicate (increasingly) with each other and are easily planned.
- The configuration and deployment of the containers is possible through the interface but can also be automated through the code.
- The story ends with visualizations in Power BI (JDBC connector to Databricks tables)

My opinion about deploying with Azure ML

Which would make it easier for us:

- A Data Preparation tool with predefined connectors
  - *Where is Azure Machine Learning Workbench ?*
- A better interface for designing ML pipelines, including code
  - *Could be the Studio*
- A simple DEV / UAT / PROD separation
  - *Have a look at Azure Devops (release pipeline)*
- A better mastery of Software Engineering by Data Scientists
  - *Learn Scala !*
- Or a better scalability of Python
  - *Try koalas, dask, modin…*

Summary

- Data Scientists can spend time on
  - exploring data
  - interpret models
- Data Engineers have a full toolbox to industrialize ML
- Are they really two distinct people ?

Now, only ~~sky~~ cloud is the limit !

- Fully automated deployments
- A/B testing scenario
- Integrate feedback and loop ?

And now, go further with…

- AutomatedML
- azureml-core on Databricks
- azureml for R
- API
- mlFlow
- InterpretML Community


Q&A

Webography

- https://github.com/Azure/MachineLearningNotebooks/
- https://medium.com/@paul.peton/choisir-un-environnement-technique-pour-la-data-science-e8816e1d5926
- https://www.blue-granite.com/blog/train-and-deploy-machine-learning-models-using-the-azureml-service
- https://docs.microsoft.com/en-us/azure/machine-learning/azure-machine-learning-release-notes
- https://stackoverflow.com/questions/tagged/azure-machine-learning-service
- https://docs.microsoft.com/en-us/python/api/overview/azure/ml/?view=azure-ml-py

Help Us Grow The Community

Our Facebook Group

www.facebook.com/groups/theSQLGeeks

Our LinkedIn group

https://www.linkedin.com/groups/6753546

On Mobile

http://www.dataplatformgeeks.com/dpg-mobile/

We are posting today's group pic in our LinkedIn & FB Group.

Share it on your wall and post comments. Let your friends know that you joined today's session and it was worth your time. Invite them to www.DataPlatformGeeks.com so that they too can benefit. We all grow together.

DataPlatformGeeks (DPG) Community

Join the fastest growing community of data & analytics professionals

Why Join DPG?

Attend all events hosted by DPG, including SQLMaestros Special Events

Get access to free videos, labs, magazines and host of learning resources

Download all events & conference material

Learn new skills. Sharpen existing skills

Be part of Asia's Largest Data/Analytics Community

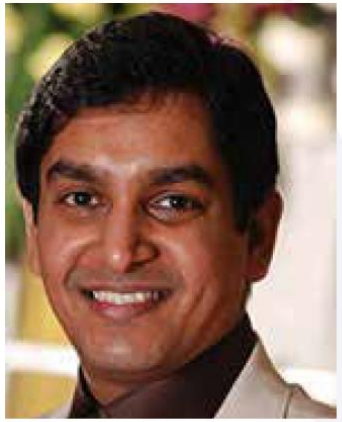
Opportunity to be a regional mentor, host DPG Meet-up's in your organization & speak at our events

Immense technical & professional development

www.DataPlatformGeeks.com

www.SQLServerGeeks.com

DPG Core Team



Amit Bansal
(Founder & President)

Manohar Punna
(Vice President)

Avanish Panchal
(Editor & Regional Mentor)

Vijay Mishra
(Regional Mentor)

Vijay Reddy
(Regional Mentor)

Sandip Pani
(Regional Mentor)

Prince Rastogi
(Regional Mentor)

Surbhi Agarwal
(Regional Mentor)

Sakharam Shinde
(Regional Mentor)

Debdutta Nath
(Regional Mentor)

Have Questions?

Join the largest SQL group on FB

www.facebook.com/groups/theSQLGeeks

Our LinkedIn group

https://www.linkedin.com/groups/6753546

On Mobile

http://www.dataplatformgeeks.com/dpg-mobile/

Thank You For Your Time!

@TheDataGeeks
@SQLServerGeeks

www.YouTube.com/SQLServerGeeks

www.DataPlatformGeeks.com
www.SQLServerGeeks.com

admin@DataPlatformGeeks.com

facebook.com/DataPlatformGeeks
facebook.com/SQLServerGeeks
facebook.com/groups/theSQLGeeks

Join The Community on Mobile
http://www.dataplatformgeeks.com/dpg-mobile/