Azure Machine Learning

Hands-on Azure ML

Saison 01 épisode 01

Jeudi 10 juin 2021

Description de l'atelier

- *A l'opposé de l'approche « it works on my laptob », il est fondamental de pratiquer la Data Science en interaction avec l'architecture data déployée en amont (stockage, ingestion) et en aval (automatisation, exposition). Le service Azure Machine Learning tient maintenant une place prépondérante, en tant que portail « unifiant » toutes les ressources nécessaires : jeux de données, puissance de calcul, environnements d'exécution, points de terminaison.*
- *La démarche adoptée consistera à passer **exclusivement par du code** pour manipuler toutes les briques du portail Azure ML studio.*

Prérequis

Disposer d’une souscription Azure

sur votre compte MSDN personnel

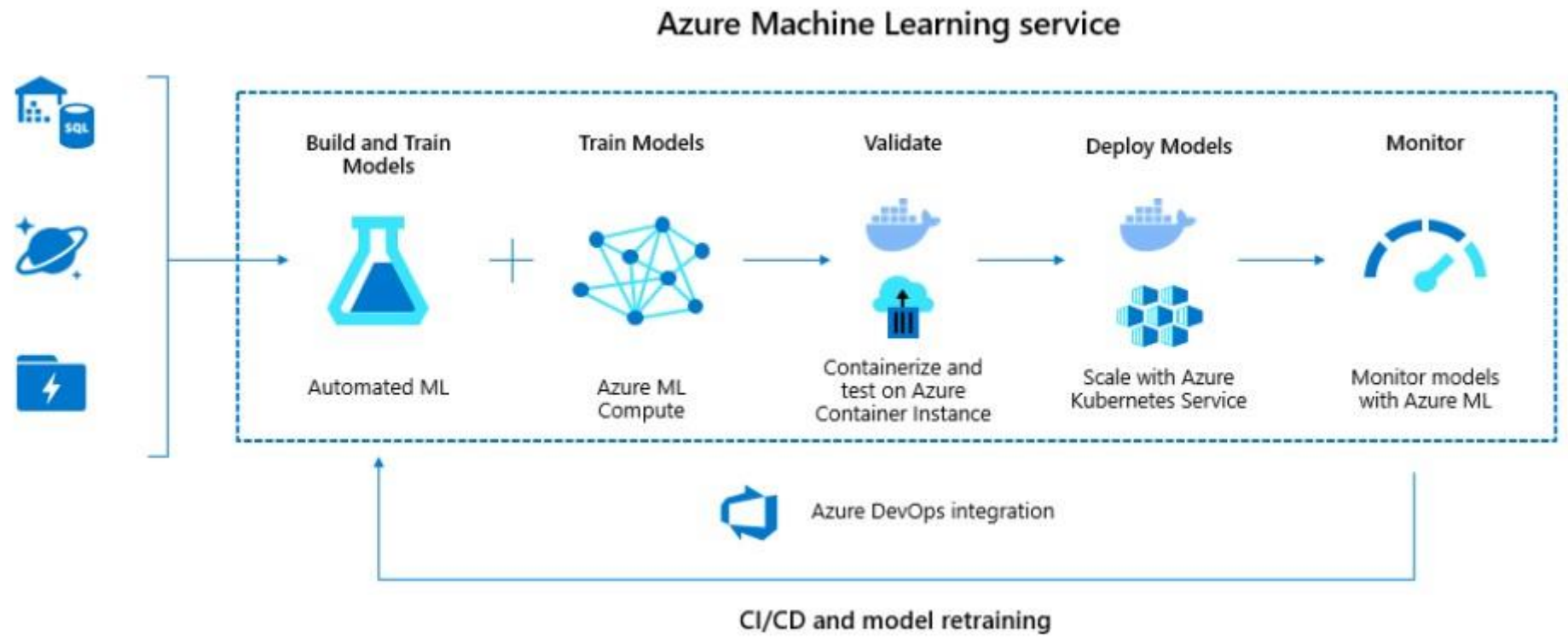
Avoir créé la ressource Azure ML en amont

voir le pas à pas fourni

Avoir des bases de développement Python ou R

Connaître les principes de bases du Machine Learning supervisé (train/test, classification/régression, métriques d’évaluation)

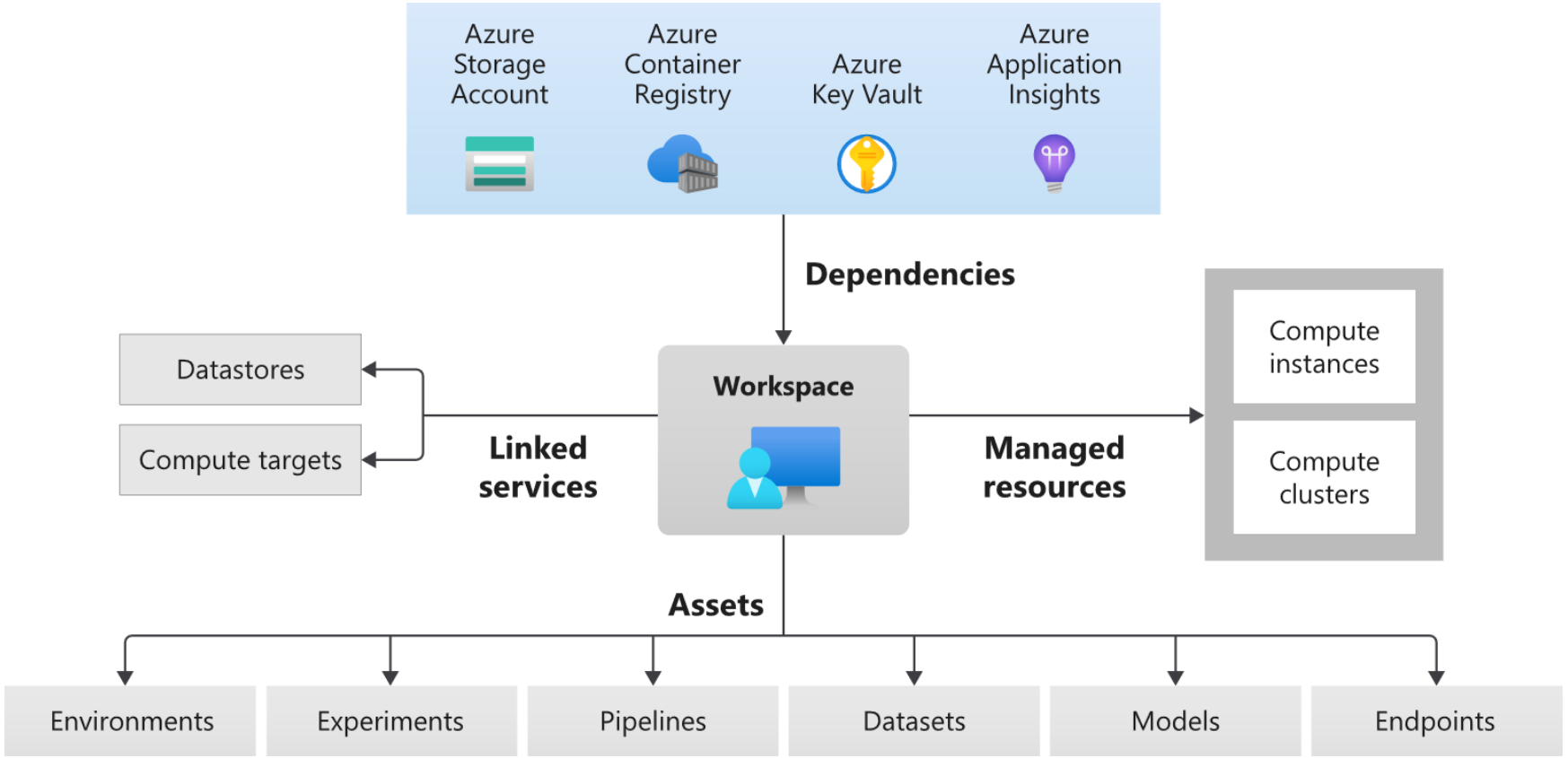
Data Science workflow:
tout tourne autour du modèle



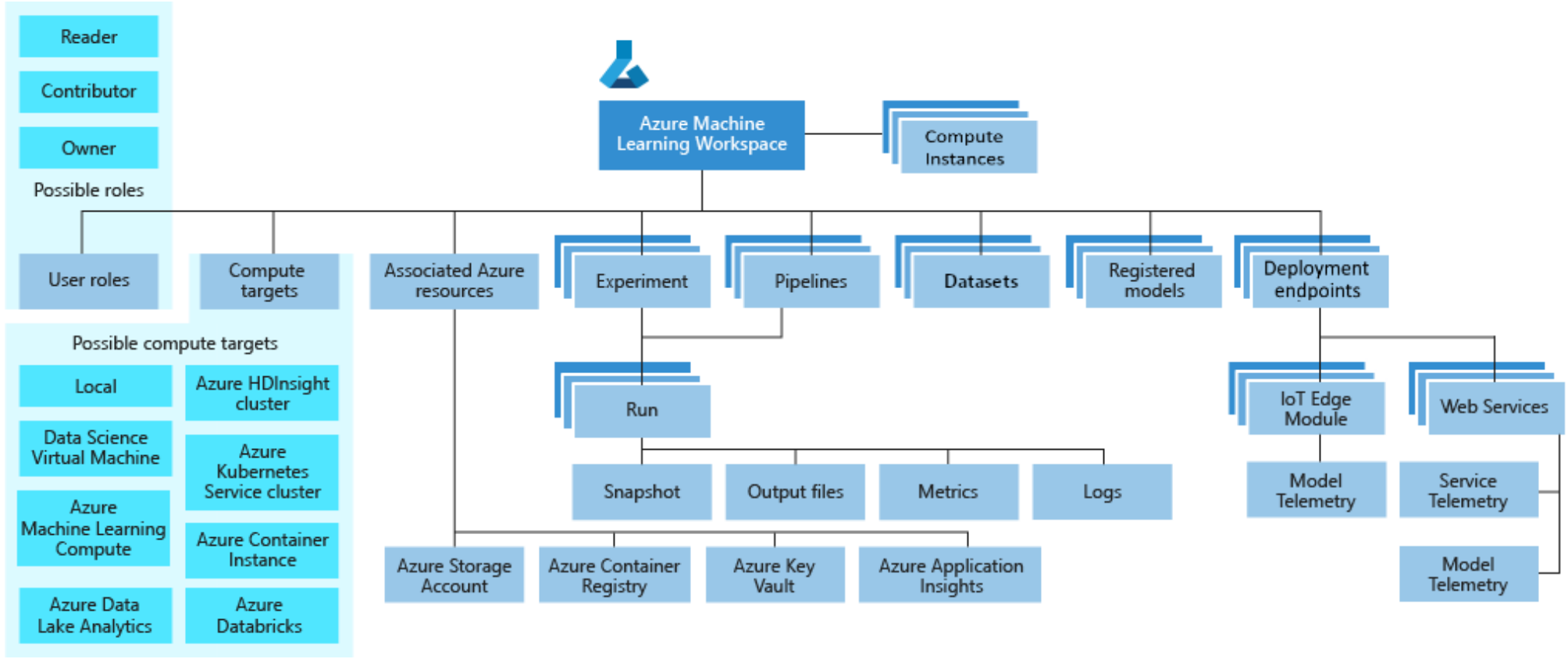
Episode n°1 : découvrir le SDK Python azureml-core

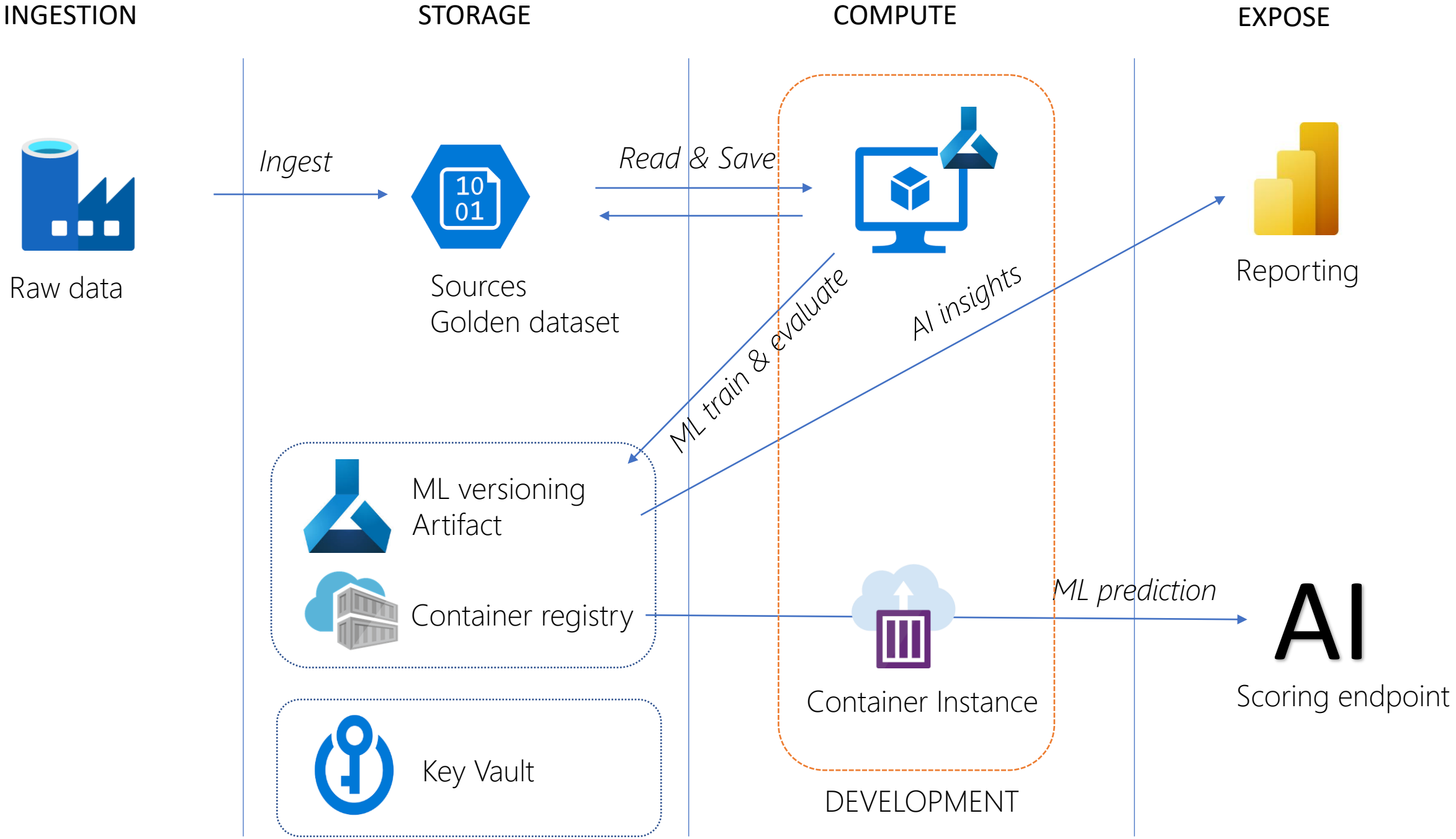
- Introduction : survol du portail Azure Machine Learning
- Atelier 1
  - Se connecter à l'espace de travail (workspace)
  - Créer un nouveau datastore
  - Créer des datasets
  - Charger un dataset en mémoire dans un pandas dataframe
- Atelier 2
  - Entrainer un modèle
  - Enregistrer, tagger et versionner le modèle
  - Déployer un web service local (scikit learn)
- Gouvernance : réfléchir au nommage des éléments
- Atelier 3
  - Définir une expérience et un environnement
  - Créer un cluster de calcul
  - Lancer une exécution (run) sur le cluster
  - Ajouter des logs (métriques) à l'exécution
- Des ressources pour pratiquer

Les briques d'Azure Machine Learning



Terminologie utilisée




INGESTION
STORAGE
COMPUTE
EXPOSE
Raw data
Ingest
Sources
Golden dataset
Read & Save
ML train & evaluate
AI insights
Reporting
ML versioning
Artifact
Container registry
Key Vault
Container Instance
DEVELOPMENT
ML prediction
AI
Scoring endpoint

WHAT IS THE AZUREML PYTHON SDK ?

A set of libraries that facilitate access to :

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

Official GitHub repository :
https://github.com/Azure/azure-sdk-for-python

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
```

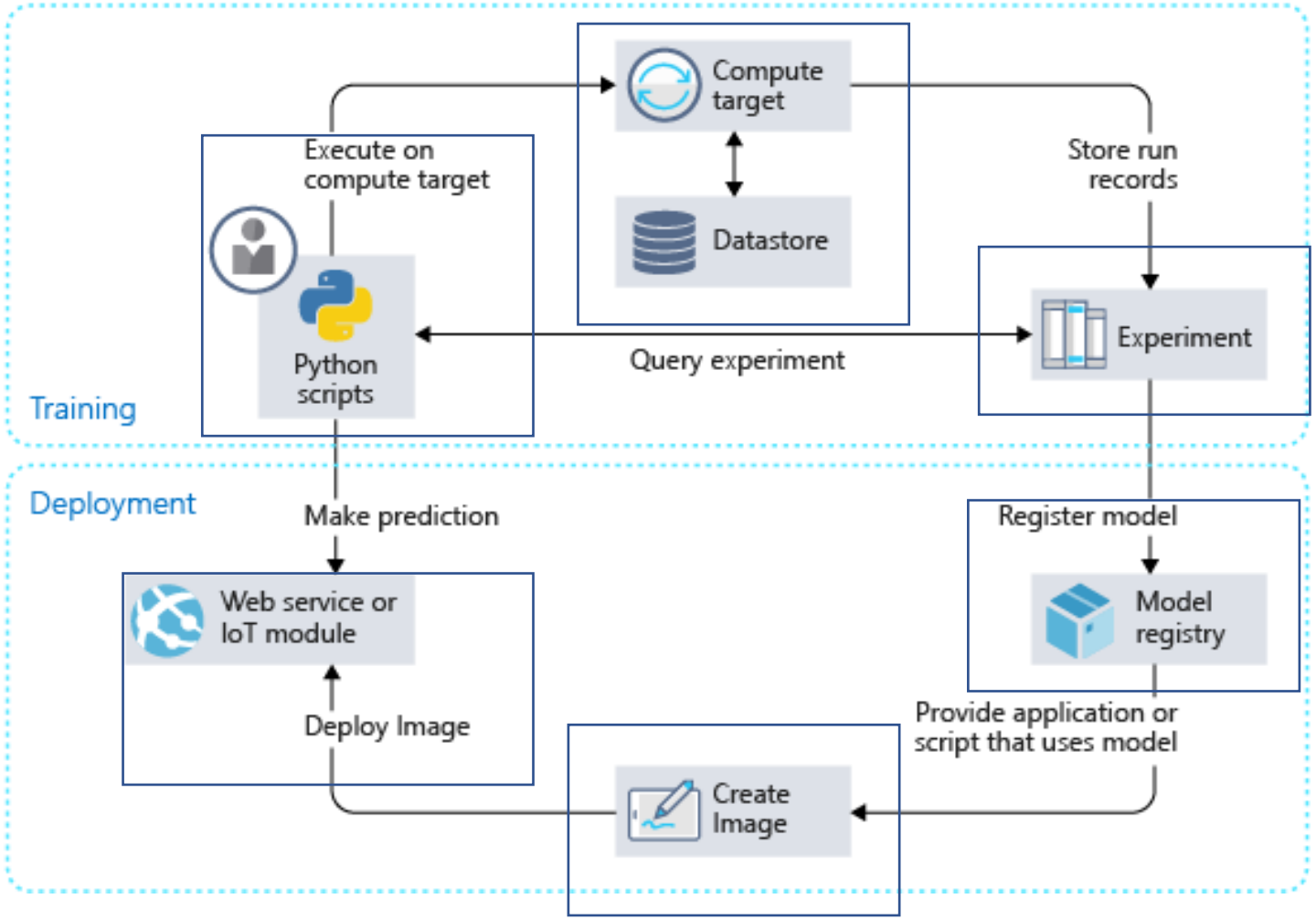
Or clone th GtiHub reporsitory :

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

THE MAIN OBJECTS

Inside the workspace (your Azure resource)

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



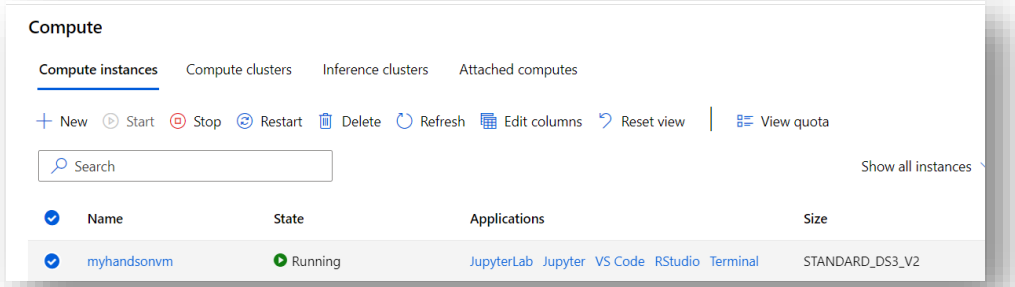
Lancer l'environnement de travail

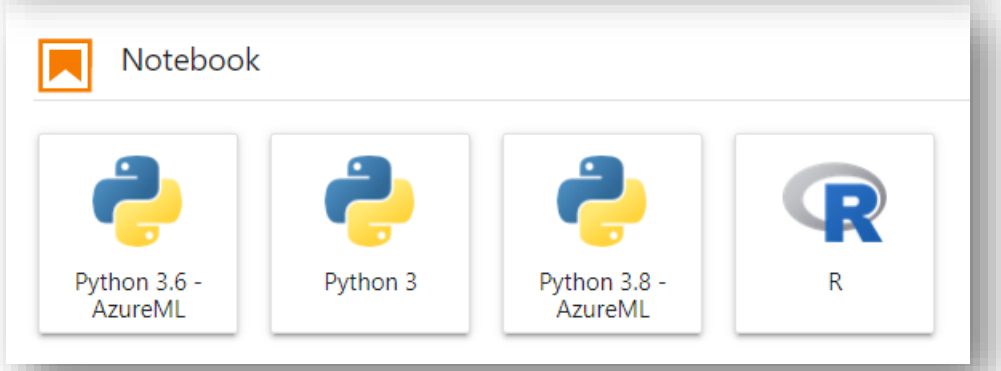
- Créer une instance de calcul (personnelle)
- Se connecter à JupyterLab
  - Lancer un nouveau notebook Python 3
- Installer les packages nécessaires

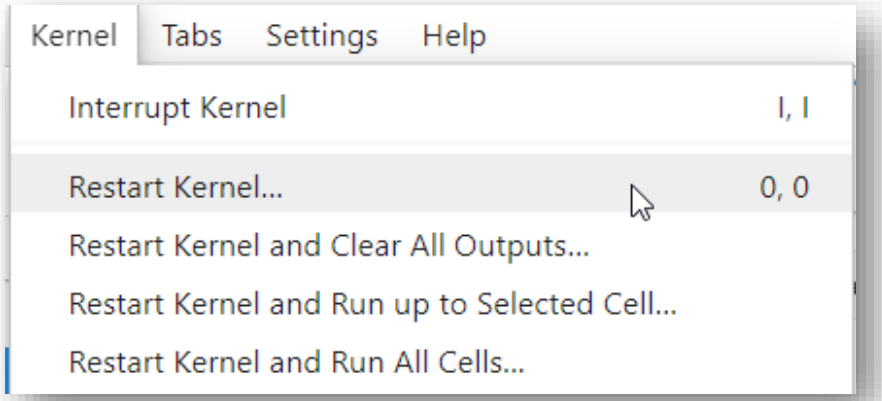
```
!pip install --upgrade azureml-core

# check core SDK version number
import azureml.core
print(f'Azure ML SDK Version: {azureml.core.VERSION}')
```

- Redémarrer le kernel
- Par la suite, stabiliser la version du SDK :

```
!pip install azureml-core==1.30.0
```







Atelier n°1

Connexion à l’espace de travail et aux données

Atelier 1 : vos objectifs

- Depuis le portail Azure, ajouter un **conteneur** nommé « data » dans le compte de stockage
  - Uploader les deux fichiers CSV « german_credit » dans ce conteneur
- Depuis le studio Azure ML, créer un nouveau **datastore** nommé « mydsstore »
  - Basé sur le conteneur data
  - En utilisant le secret du compte de stockage
- Depuis le studio Azure ML, créer un nouveau **dataset**
  - A partir d'un fichier https://datahub.io/machine-learning/credit-g/r/credit-g.csv
- Depuis un notebook, se connecter à l'espace de travail (**workspace**)
  - De manière interactive
    - Soit avec le fichier config.json : `from_config()`
    - Soit en donnant les trois informations : `subscription_id, resource_group, workspace_name`
    - *Nous ne passerons pas pour l'instant par un principal de service*
- Charger un dataset en mémoire dans un pandas **dataframe**
  - Compter le nombre de lignes du dataset
  - Mettre à jour le dataset et compter le nombre de lignes du dataset

Workspace
(méthode 1)

```
# load workspace configuration from the config.json file in the current folder.
try:
    ws = Workspace.from_config()
    print('Workspace connection succeeded')
except:
    print('Workspace not found’)

ws.get_details()
```

Workspace
(méthode 2)

```
# load workspace configuration from the config.json file in the current folder.
try:
    ws = Workspace(subscription_id = subscription_id, resource_group =
resource_group, workspace_name = workspace_name)
    ws.write_config()
    print('Library configuration succeeded')
except:
    print('Workspace not found’)


ws.get_details()
```

Sources de données supportées

Déclaration d'une source de données

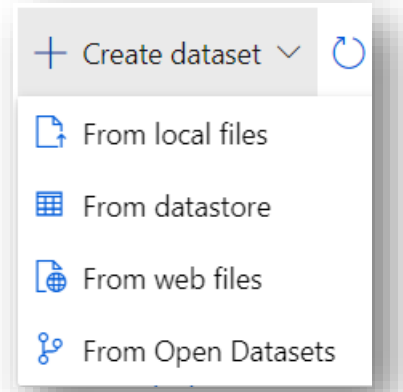
- ❑ Azure Blob Container
- ❑ Azure File Share
- ❑ Azure Data Lake
- ❑ Azure Data Lake Gen2

- ❑ Azure SQL Database
- ❑ Azure Database for PostgreSQL
- ❑ Azure Database for MySQL

- ❑ Databricks File System

Création d'un jeu de données depuis l'interface

- ❑ Depuis un datastore déclaré
  - ❑ Tabular dataset
  - ❑ File dataset
- ❑ Fichiers locaux
  - ❑ Upload file
  - ❑ Upload folder
- ❑ URL Web
  - ❑ http://example.com/files/examples.csv
- ❑ Open Datasets
  - ❑ Open Datasets | Microsoft Azure



Datastore
(Blob Storage)

```
# default datastore
from azureml.core import Datastore
Datastore.get_default(ws)

# create data store (Blob storage)
Datastore.register_azure_blob_container(workspace,
                datastore_name,
                container_name,
                account_name,
                sas_token=None,
                account_key=None,
                protocol=None,
                endpoint=None,
                overwrite=False,
                create_if_not_exists=False,
                skip_validation=False,
                blob_cache_timeout=None,
                grant_workspace_access=False,
                subscription_id=None,
                resource_group=None)
```

Datastore
(Data Lake Store gen2)

```
from azureml.core import Datastore


# create data store (ADLS gen2)
Datastore.register_azure_data_lake_gen2(workspace,
                datastore_name,
                filesystem,
                account_name,
                tenant_id=None,
                client_id=None,
                client_secret=None,
                resource_url=None,
                authority_url=None,
                protocol=None,
                endpoint=None,
                overwrite=False,
                subscription_id=None,
                resource_group=None,
                grant_workspace_access=False
)
```

Utiliser Azure Key Vault

```
from azureml.core import Keyvault

keyvault = ws.get_default_keyvault()

#On préfèrera saisir les informations manuellements dans le Key Vault plutôt que par le code
keyvault.set_secret(name="sp-authentication-client-secret", value = my_secret)

#Attention, il est possible de voir en clair les secrets obtenus avec la méthode get_secret
keyvault.get_secret(name="sp-authentication-client-secret")
```

Dataset

(depuis un dataset déjà créé sur le portail)

```
from azureml.core import Dataset
```

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

#Obtenir une version particulière (par défaut, version='latest')
dataset = Dataset.get_by_name(ws, name=dataset_name, version=1)
df = dataset.to_pandas_dataframe()
```

Dataset

(depuis un fichier hébergé sur un datastore)

```
from azureml.core import Dataset

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

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

Atelier n°2

Entrainement de modèle et déploiement de service web

Atelier 2 : vos objectifs

- Entrainer un modèle d'apprentissage avec scikit learn
  - Choisir un autre « toy dataset »
  - Choisir un autre algorithme, adapter les hyperparamètres
- Enregistrer, tagger et versionner le modèle
  - Au format Pickle avec la librairie joblib
  - Avec model.register
    - Ajouter des tags
  - Tester le modèle sur de nouvelles valeurs
- Restaurer un modèle d'une version précédente
  - Contrôler le numéro de version du modèle
- Déployer un web service local (scikit learn)
  - Interroger le web service

Entrainer un modèle avec scikit learn

```
target = "target_name"
X = df.drop(target, axis=1)
Y = df[[target]].reshape(1,-1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 5)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)

from sklearn.model_selection import GridSearchCV
parameters = {"n_neighbors": [2,3,4,5]}
model = GridSearchCV(KNeighborsClassifier(), parameters)
```

Enregistrer, tagger et versionner le modèle

```
model_name = 'german_credit_log_model'
model_path = 'models/' + model_name + '.pkl'

import joblib
joblib.dump(model, model_path)

from azureml.core.model import Model


amlmodel = Model.register(workspace=ws,
                          model_name=model_name,
                          model_path=model_path,
                          model_framework=Model.Framework.SCIKITLEARN,        # Framework used to create the model.
                          model_framework_version=sklearn.__version__,        # Version of scikit-learn used to create the model.
                          tags={'dataset': "german credit", 'type': "logistic regression"},
                          description="Logisitc regression model for credit risk"
                          )
```

Restaurer un modèle (azureml) d’une précédente version précédente

```
amlmodel = Model(ws, model_name, version)


print('Model name:', amlmodel.name)
print('Model version:', amlmodel.version)
print('Model path:', amlmodel.get_model_path)


model_path = amlmodel.get_model_path
```

« Unpikle » un modèle et test sur de nouvelles valeurs

```
model = joblib.load(model_path)

raw_data = json.dumps({
    'data': [
        ['0<=X<200',24,'existing paid','radio/tv',5951,'<100','1<=X<4',2,'female div/dep/mar','none',2,'real estate',22,'none','own',1,'skilled',1,'none','yes']
    ]
    ,'method': 'predict'   # If you have a classification model, you can get probabilities by changing this to 'predict_proba'.
})

data = json.loads(raw_data)['data']
method = json.loads(raw_data)['method']
result = model.predict(data) if method=="predict" else model.predict_proba(data)

print(result.tolist())
```

Déployer un web service (scikit learn)

```
# Use a default environment
# nécessite d'avoir précisé le framework lors de l'enregistrement du modèle
# Default environments are only provided for these frameworks: ['Onnx', 'ScikitLearn', 'TensorFlow']."

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

service_name = 'german-credit-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)
```

Interroger le web service

```
import json


input_payload = json.dumps({
    "data": [
        ["0<=X<200",24,"existing paid","radio/tv",5951,"<100","1<=X<4",2,"female div/dep/mar","none",2,"real estate",22,"none","own",1,"skilled",1,"none","yes"],
    "method": "predict"  # For classification model, get probabilities with 'predict_proba'.
})


output = service.run(input_payload)


print(output)
```

Atelier n°3

Logs d’exécution au sein d’une expérience

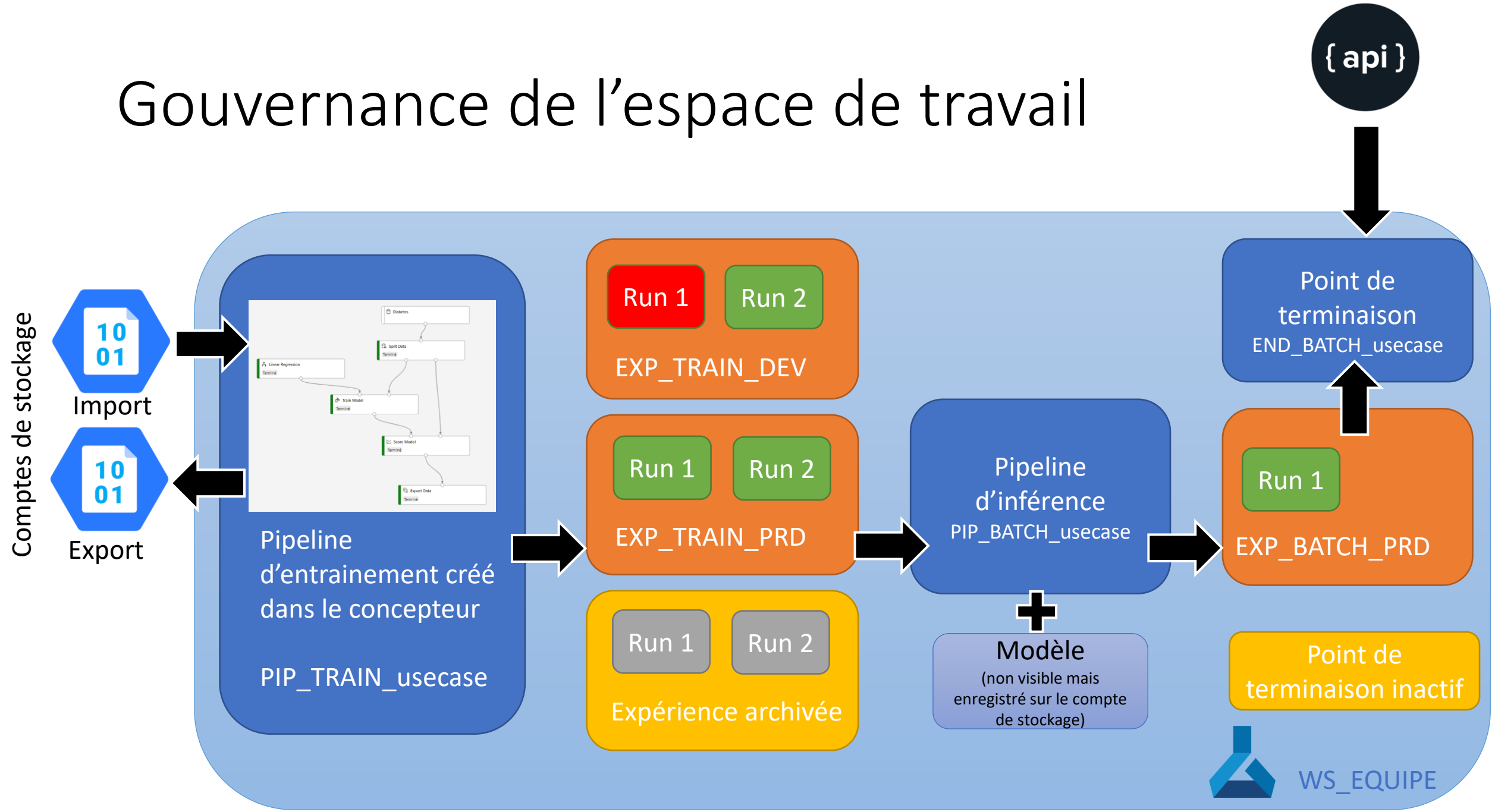
Atelier 3 : vos objectifs

- Créer une **expérience**
- Définir une expérience et un **environnement**
  - Par un objet environnement
  - Par un fichier YAML de configuration
- Créer une cible de calcul (**compute target**)
- Lancer une exécution (**run**) sur le cluster
- Enregistrer le modèle
- Ajouter des logs (métriques) à l'exécution

Créer une expérience

```
experiment_name = 'my_experience_name'

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

#Attention, il ne sera pas possible de supprimer une expérience quand celle-ci contiendra des exécutions (runs)
#Elle pourra en revanche être archivée.
experiment.archive()
experiment.archived_time
experiment.reactivate()

experiment.delete(ws, experiment.id)
#Message: Only empty Experiments can be deleted. This experiment contains run(s)
runs = experiment.get_runs(include_children=True)
for r in runs:
    print(r.id)
```

Gouvernance de l’espace de travail



Create a (remote) target compute

```
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

cpu_cluster_name = 'my_compute_cluster'

# Verify that cluster does not exist already
try:
    compute_target = ComputeTarget(workspace=ws, name=cpu_cluster_name)
    print("Cluster already exists")
except ComputeTargetException:
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_DS3_V2',
                                                           min_nodes=0, max_nodes=4,
                                                           tags={"Type": "CPU", "Priority":"Dedicated"},
                                                           idle_seconds_before_scaledown=1800) #Timeout for scaling down)
 compute_target = ComputeTarget.create(ws, cpu_cluster_name, compute_config)

compute_target.wait_for_completion(show_output=True, min_node_count=0, timeout_in_minutes=30)
```

Set up the (compute target) environnement

```
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies


env = Environment('compute_env')

env.docker.enabled = True
env.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn',
                                                          'pandas',
                                                          'numpy',
                                                          'joblib',
                                                          'matplotlib'
                                                         ])
env.python.conda_dependencies.add_pip_package('inference-schema[numpy-support]')

env.python.conda_dependencies.save_to_file('.', 'compute_env.yml')
```

Configuration d’exécution : ScriptRunConfig

```
from azureml.core import ScriptRunConfig

src = ScriptRunConfig(source_directory='scripts',
                      script='train.py',
                      arguments=['--param1', <value>, '- param2', <value>],
                      environment=env)
```

Exécution

```
from azureml.core import Run

run = exp.submit(src)

from azureml.widgets import RunDetails

RunDetails(run).show()

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

run_logs = run.get_details_with_logs()
run_logs['submittedBy']
run_logs['startTimeUtc']
```

Enregistrer le modèle depuis une exécution

```
print(run.get_file_names())

# register model
aml_model = run.register_model(model_name='my_model_name', model_path='outputs/my_model.pkl')
```

Ajouter des logs (métriques) à l'exécution

```
from azureml.core import Run
run = Run.get_context()
run.log('metric-name', metric_value)

metrics = run.get_metrics()
# metrics is of type Dict[str, List[float]] mapping metric names
# to a list of the values for that metric in the given run.

metrics.get('metric-name')
# list of metrics in the order they were recorded
```

Aller plus loin

- Enregistrer le modèle avec un exemple de données
- Utiliser les décorateurs dans le script de scoring

- Travailler avec une data preparation enregistrée en pickle
- Utiliser les pipelines et les steps

- Utiliser Mlflow
  - Pour les métriques
  - Pour les artefacts

- Utiliser une image Docker existante

Des ressources GitHub pour pratiquer

- Notebooks utilisés pendant le Hands-On

- https://azure.github.io/azureml-cheatsheets/

- https://github.com/methodidacte/azureml
- https://github.com/retkowsky
- https://github.com/Azure/MachineLearningNotebooks
- https://github.com/microsoft/MLOpsPython