MLOps

Deploying real-time ML inference services

Machine Learning Operations

Damian Konrad Kowalczyk, PhD

Applied Scientist, Cloud + AI Division

Microsoft Corporation

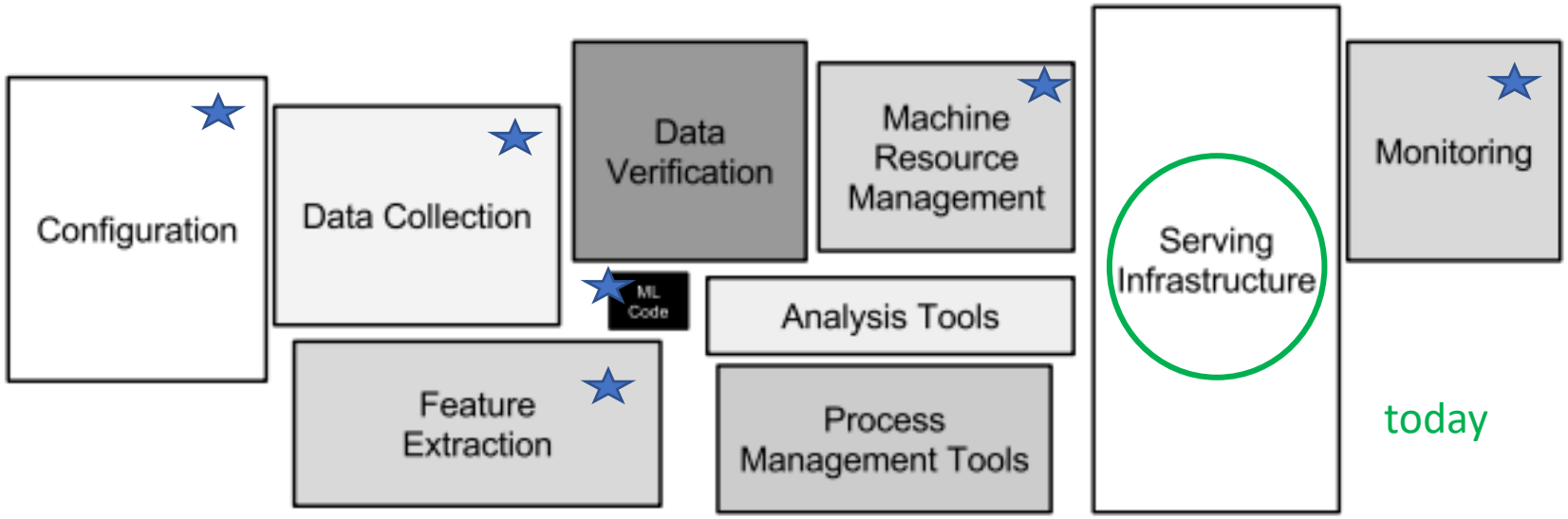
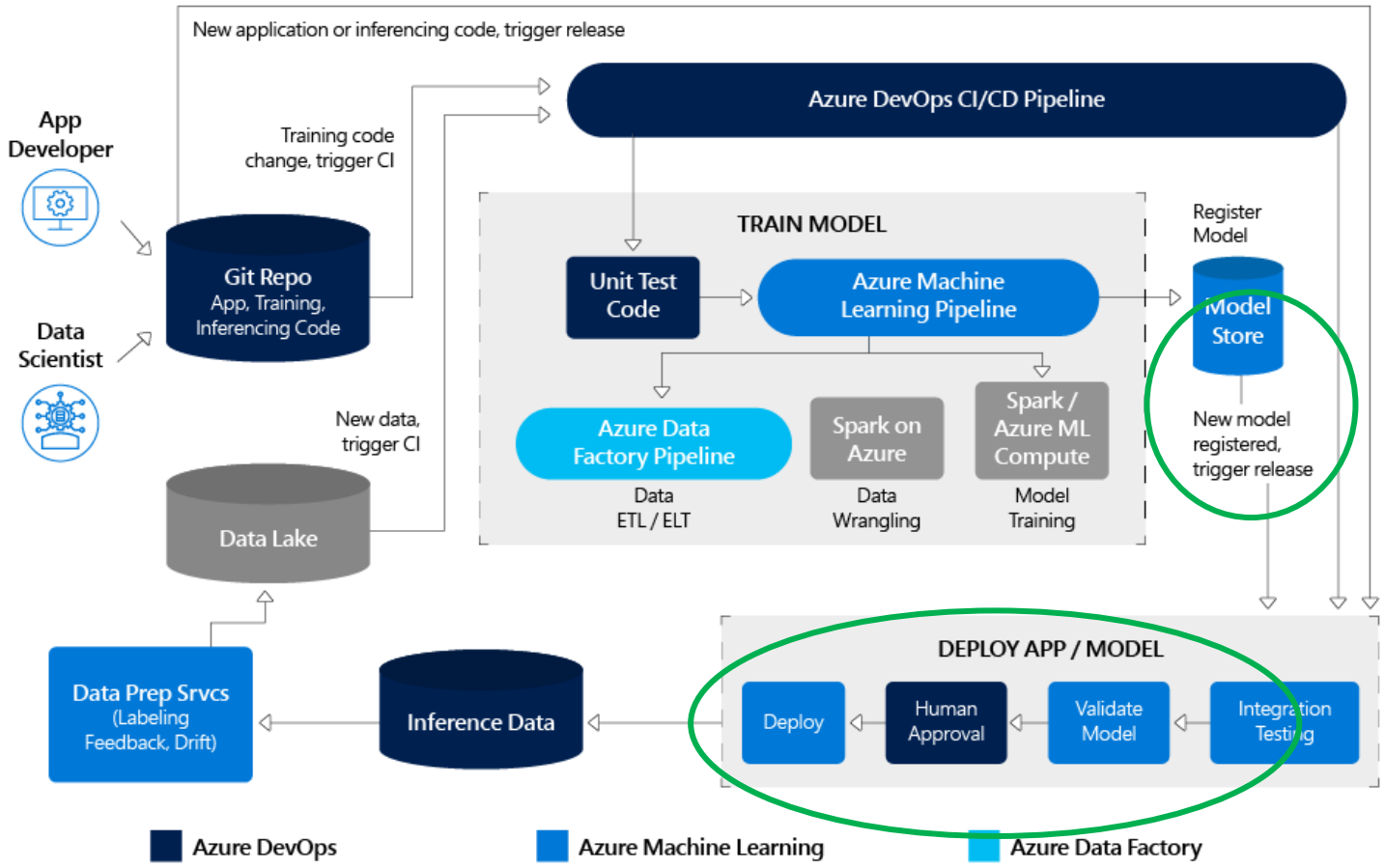


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

source: [Hidden Technical Debt in Machine Learning Systems (nips.cc)]

science meets engineering: MLOps on Azure



Breaking the wall between data scientists and app developers with Azure DevOps | Azure Blog and Updates | Microsoft Azure

Today

- Introduction & compute targets for inference
- Deploy a model as a real-time inferencing service.
- Consume a real-time inferencing service.
- Troubleshoot service deployment

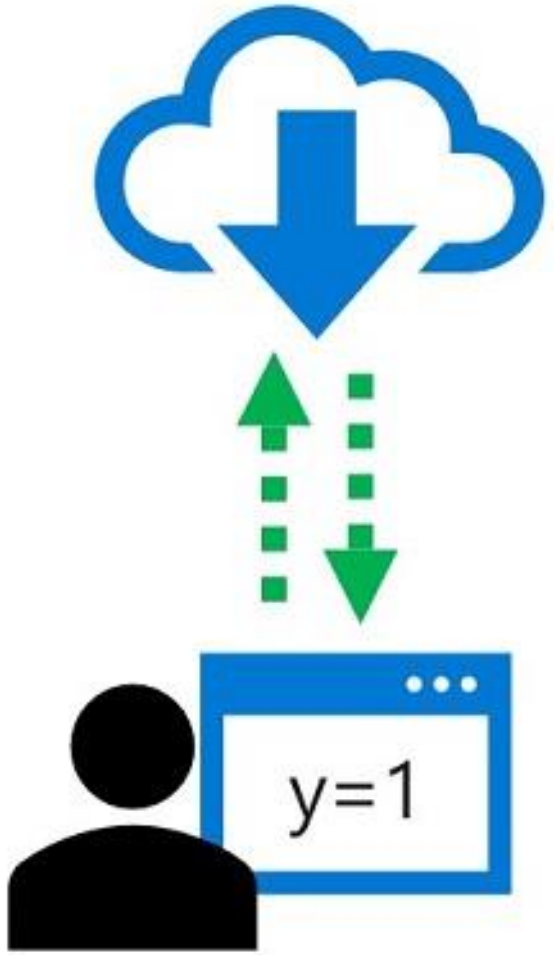
Part of: Build and operate machine learning solutions with Azure Machine Learning

extra: predicting virality of incoming social media posts (case study)

Introduction

In machine learning, *inferencing* refers to the use of a trained model to predict labels for new data on which the model has not been trained. Often, the model is deployed as part of a service that enables applications to request immediate, or *real-time*, predictions for individual, or small numbers of data observations.

In Azure Machine Learning, you can create real-time inferencing solutions by deploying a model as a service, hosted in a containerized platform, such as Azure Kubernetes Services (AKS).



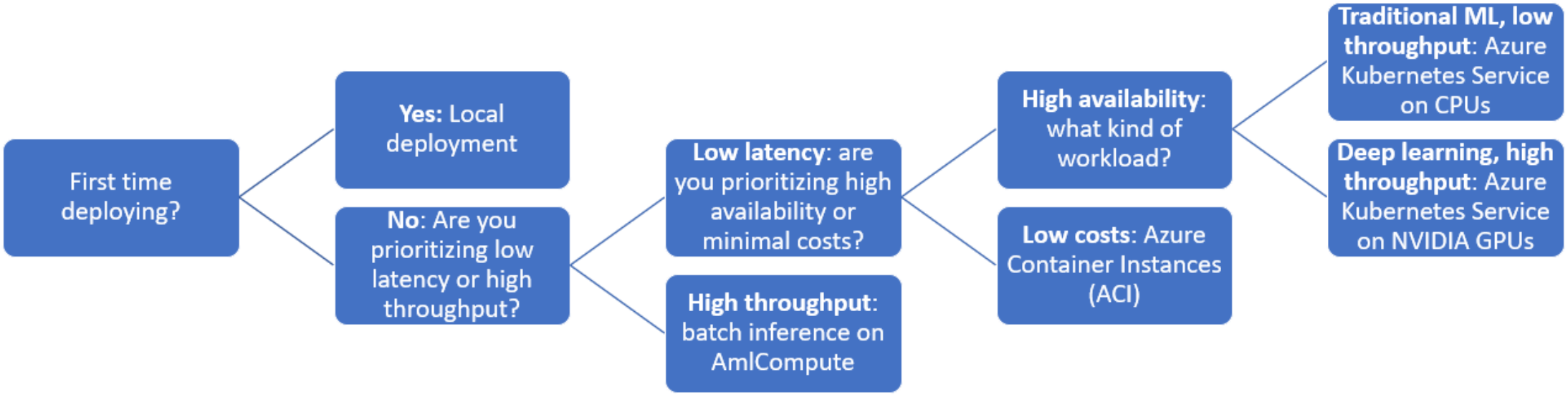
Compute target for model deployment

You can deploy a model as a real-time web service to several kinds of compute target, including:

- local compute
- Azure Machine Learning compute instance,
- Azure Container Instance (ACI),
- Azure Kubernetes Service (AKS) cluster,
- Azure Function (preview)
- Internet of Things (IoT) module

Azure Machine Learning uses *containers* as a deployment mechanism, packaging the model and the code to use it as an image.

choosing your next compute target



1. Register a trained model (recap)

```
from azureml.core import Model

classification_model = Model.register(workspace=ws,
                       model_name='classification_model',
                       model_path='model.pkl', # local path
                       description='A classification model')
```

```
run.register_model( model_name='classification_model',
                    model_path='outputs/model.pkl', # run outputs path
                    description='A classification model')
```

2. Define inference configuration

```
import json
import joblib
import numpy as np
from azureml.core.model import Model

# Called when the service is loaded
def init():
    global model
    # Get the path to the registered model file and load it
    model_path = Model.get_model_path('classification_model')
    model = joblib.load(model_path)

# Called when a request is received
def run(raw_data):
    # Get the input data as a numpy array
    data = np.array(json.loads(raw_data)['data'])
    # Get a prediction from the model
    predictions = model.predict(data)
    # Return the predictions as any JSON serializable format
    return predictions.tolist()
```

The model will be deployed as a service that consist of:

1. A script to load the model and return predictions for submitted data.
2. An environment in which the script will be run.

```
from azureml.core.conda_dependencies import CondaDependencies

# Add the dependencies for your model
myenv = CondaDependencies()
myenv.add_conda_package("scikit-learn")

# Save the environment config as a .yml file
env_file = 'service_files/env.yml'
with open(env_file,"w") as f:
    f.write(myenv.serialize_to_string())
print("Saved dependency info in", env_file)
```

2. Define inference configuration

After creating the entry script and environment configuration file, you can combine them in an **InferenceConfig** for the service

```
from azureml.core.model import InferenceConfig

classifier_inference_config = InferenceConfig(runtime= "python",
                                              source_directory = 'service_files',
                                              entry_script="score.py",
                                              conda_file="env.yml")
```

3. Define a deployment configuration

Define + create compute target for inference

```
from azureml.core.compute import ComputeTarget, AksCompute

cluster_name = 'aks-cluster'
compute_config = AksCompute.provisioning_configuration(location='eastus')
production_cluster = ComputeTarget.create(ws, cluster_name, compute_config)
production_cluster.wait_for_completion(show_output=True)
```

Create deployment configuration

```
from azureml.core.webservice import AksWebservice

classifier_deploy_config = AksWebservice.deploy_configuration(cpu_cores = 1,
                                                              memory_gb = 1)
```

4. Deploy the model

With all the configuration prepared, you can deploy the model as an inference service

```
from azureml.core.model import Model

model = ws.models['classification_model']
service = Model.deploy(workspace=ws,
                       name = 'classifier-service',
                       models = [model],
                       inference_config = classifier_inference_config,
                       deployment_config = classifier_deploy_config,
                       deployment_target = production_cluster)
service.wait_for_deployment(show_output = True)
```

Consume a real-time inferencing service

After deploying a real-time service, you can consume it from client applications to predict labels for new data cases.

```
{
  "data":[
      [0.1,2.3,4.1,2.0], // 1st case
      [0.2,1.8,3.9,2.1],  // 2nd case,
      ...
  ]
}
```

```
import json

# An array of new data cases
x_new = [[0.1,2.3,4.1,2.0],
         [0.2,1.8,3.9,2.1]]

# Convert the array to a serializable list in a JSON document
json_data = json.dumps({"data": x_new})

# Call the web service, passing the input data
response = service.run(input_data = json_data)

# Get the predictions
predictions = json.loads(response)

# Print the predicted class for each case.
for i in range(len(x_new)):
    print (x_new[i], predictions[i])
```

Troubleshoot service deployment

1. Check the service state

```
from azureml.core.webservice import AksWebservice

# Get the deployed service
service = AksWebservice(name='classifier-service', workspace=ws)

# Check its state
print(service.state)
```

2. review service logs

```
print(service.get_logs())
```

3. deploy to a local container

```
from azureml.core.webservice import LocalWebservice

deployment_config = LocalWebservice.deploy_configuration(port=8890)
service = Model.deploy(ws, 'test-svc', [model], inference_config, deployment_config)
```

4. test the local deployment

```
print(service.run(input_data = json_data))
```

5. reload after local changes/fixes
   (no redeployment necessary locally)

```
service.reload()
print(service.run(input_data = json_data))
```

case study: predicting virality of incoming tweets (8000/s)

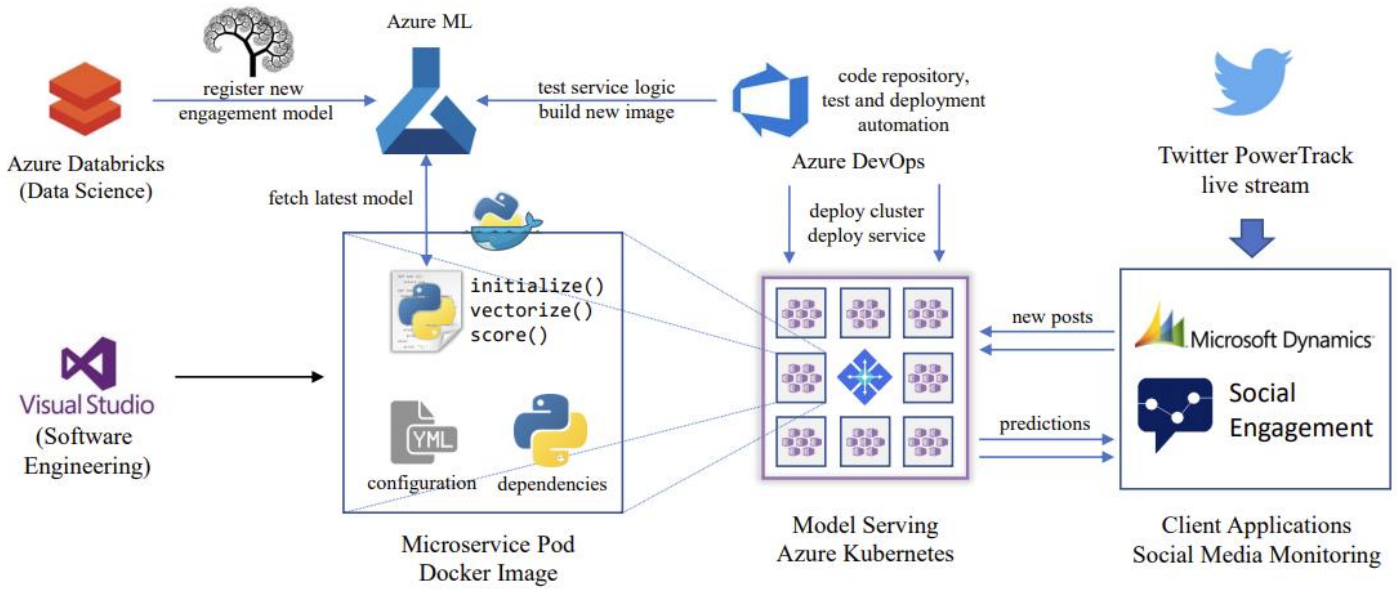


Figure 3.11: Topology of the model deployment, update and serving infrastructure, facilitating high-availability engagement analysis in production.

Social Engagement at Scale, Kowalczyk, D. K.

Exercises

- Knowledge check - Learn | Microsoft Docs
- Exercise #1 Deploy a model as a real-time service - Learn | Microsoft Docs
- Exercise #2 Deploy your own model as a real-time service

Resources

- Build and operate machine learning solutions with Azure Machine Learning

How-To Guides:

- How to deploy machine learning models - Azure Machine Learning | Microsoft Docs
- How to deploy models to Azure Container Instances - Azure Machine Learning | Microsoft Docs
- Deploy ML models to Kubernetes Service - Azure Machine Learning | Microsoft Docs

damk@dtu.dk

Damian Kowalczyk, Microsoft Corporation

demo