Deploying ML models to IoT Edge devices
-- using ONNX and AzureML

Henry Zeng
Principal Program Manager, AI Platform

Session: Deploying ML models to IoT Edge devices

The rate of innovation in hardware for the intelligent edge has led to *challenges in building AI enabled applications and services*. This session will show you how to *efficiently build and deploy machine learning* solutions for the vastly complex ecosystem of hybrid and disconnected architectures. You will *see real world applications* and learn how to *apply these patterns* to your own solutions.

What does the E2E ML lifecycle look like?

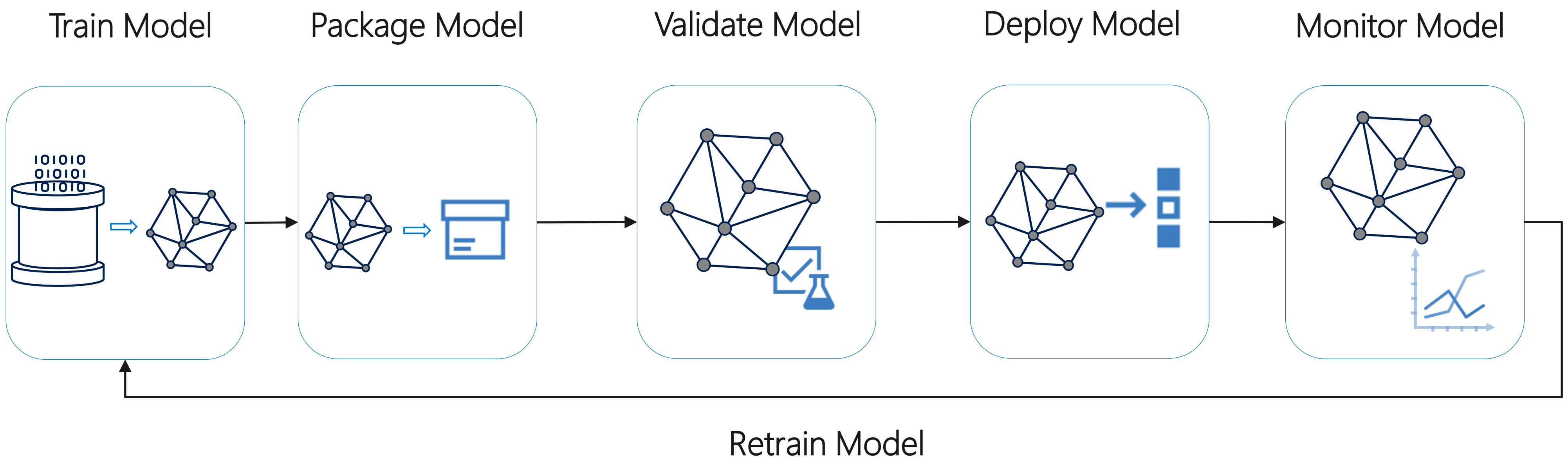
Develop and train model with reusable ML pipelines

Package model using containers to capture runtime dependencies for inference

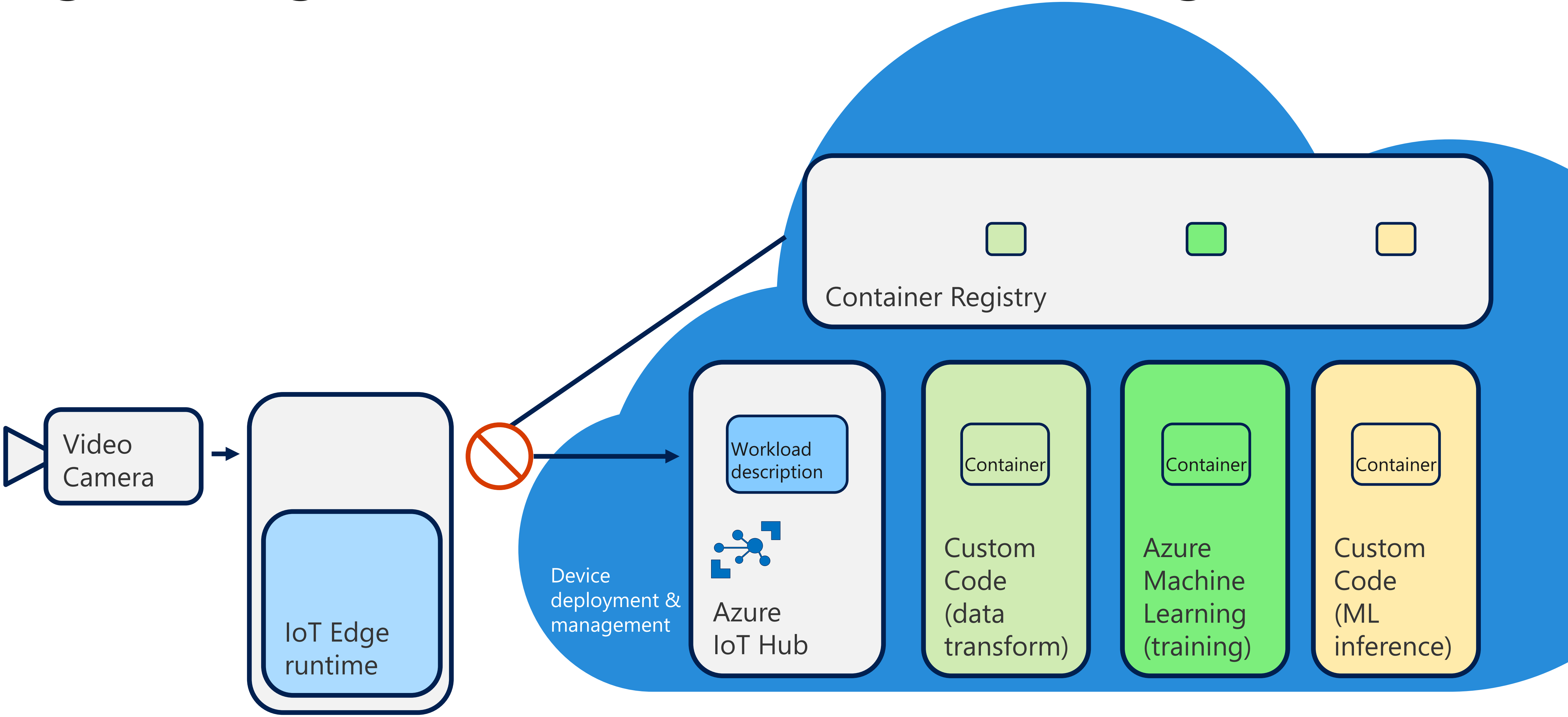
Validate model behavior—functionally, in terms of responsiveness, in terms of regulatory compliance

Deploy model—to cloud and edge, for use in real-time/streaming/batch processing

Monitor model behavior and business value, know **when to replace/deprecate a stale model**



Edge intelligence enabled with Azure IoT Edge



AI for harvesting produce

- Current approach:
  - Low frequency sampling
  - Manual, labor intensive inspection
  - Using average as gut instinct
- AI for improved yield:
  - AI on video streams to estimate growth
  - Harvesting to maximize production

Reality of building ML application for the Edge

Challenge 1: Model training data needs to be specific for the edge environments

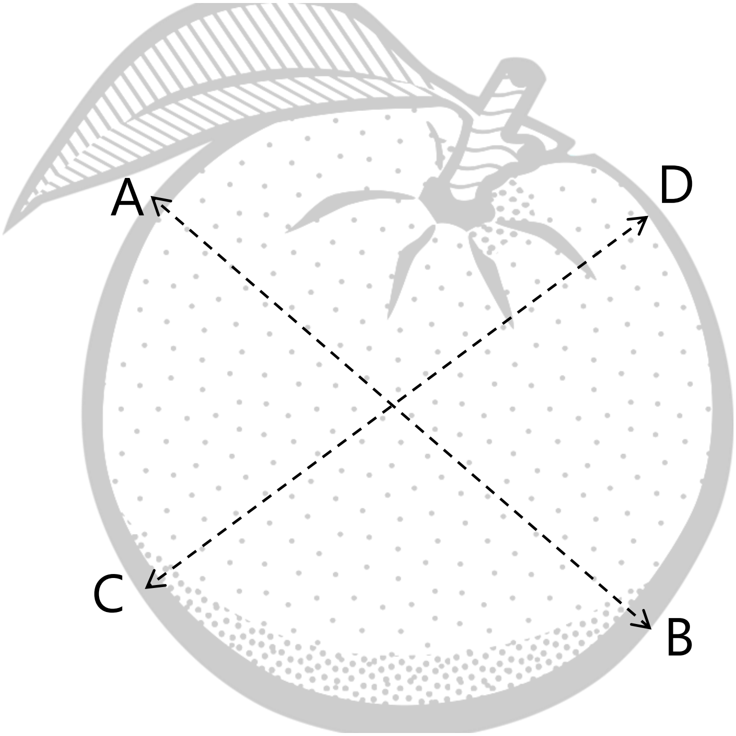
Challenge 2: SW architecture options for edge devices: throughput vs. accuracy considerations, preprocessing

Challenge 3: HW spec and configuration is fragmented for edge devices: ML execution stack

Challenge 4: Processing pipeline on edge devices to optimize all the available resources compute, storage, power & connectivity

Training for the IoT Edge

Model to detect produce maturity



Estimate maturity based on lateral measurements

Collect images for training the model

Approach

Based on this accepted approach in the field, our solution will estimate size in three steps

1. Detect key points on the orange
2. Estimate distance from camera
3. Estimate weight based on normalized distances among key points

Continuous retraining

ML model to suggest new frames to be used for (re)training

Data scientist to confirm data to be used for training

BUT conditions are different across farms

Factors that influence images captured

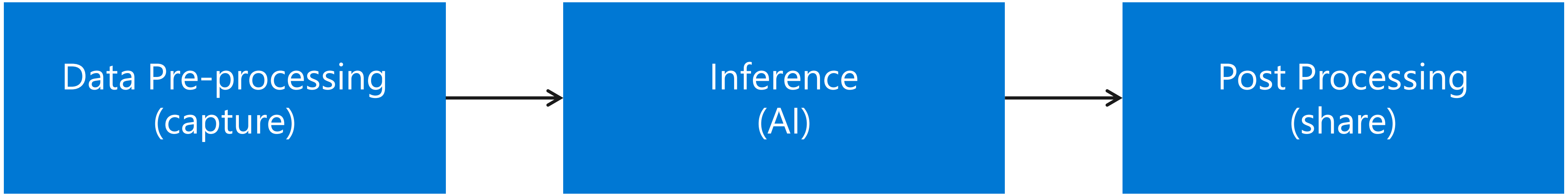
- sunlight
- geo-location
- seasonal variances
- air quality, etc.

What we did

- collect images locally
- re-train for local conditions
- tuned ML model to detect produce
- data scientist confirms images to be used for retraining

SW Architecture for the edge AI Application

AI Application Pattern for IoT Edge



images
audio
temperature
humidity

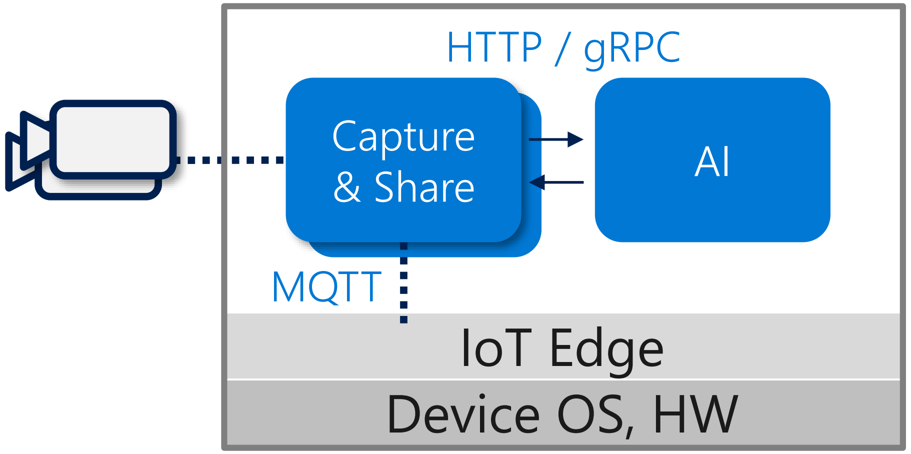
model
inference engine

telemetry
filtering
anonymizing

Architecture options for AI applications on edge

Separate containers

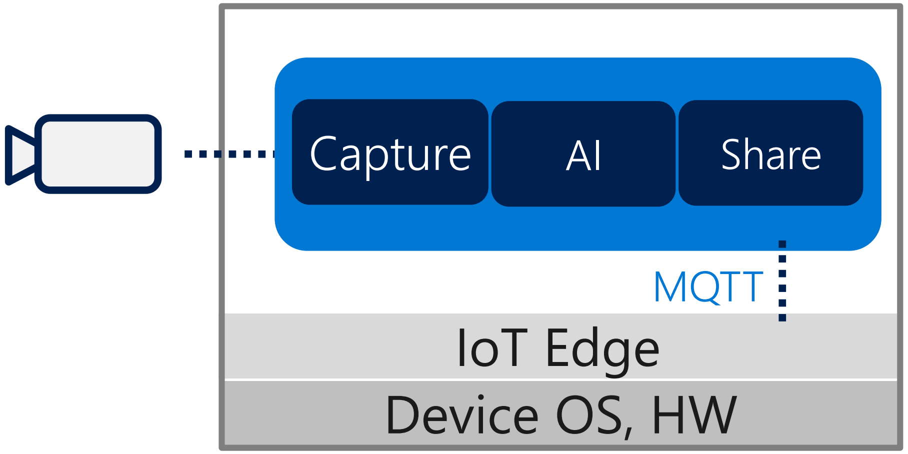
Non real-time applications



+	• Flexibility • Modularity • Reusability (Edge/Cloud)
-	• Performances

One modular container

Real-time applications

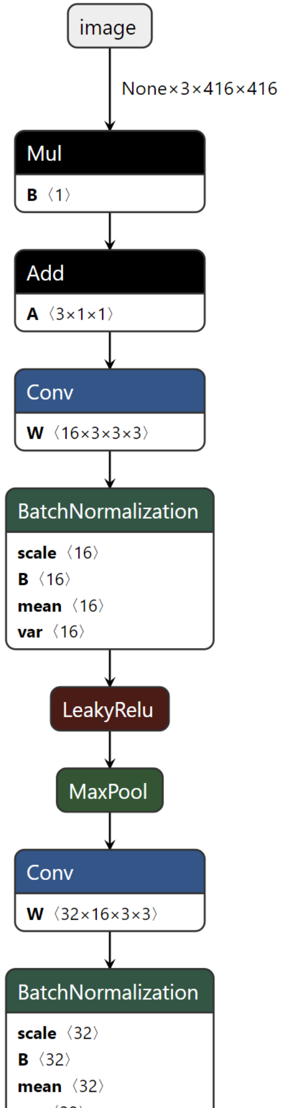


+	• Performances • Simplicity • Modularity
-	• Single source/share

Model packaging for the Edge

Components of efficient ML execution

Model graph



HW Specific Libs (and IR)*

- OpenVINO
- nGraph
- MKLDNN
- ROCm
- TVM IR
- ARM Compute Lib.
- ⋮
- TensorRT

Compute blocks* (implementation in silicon)

- VPU
- CPU
- GPU
- DSP
- ⋮

* shown as examples, not comprehensive list

ONNX is the new open ecosystem for AI models

Create

Frameworks

MathWorks
Accelerating the pace of engineering and science

ML.NET

dmlc XGBoost

scikit learn

Services

Custom Vision, an Azure Cognitive Service

Automated ML

ONNX Model

Deploy

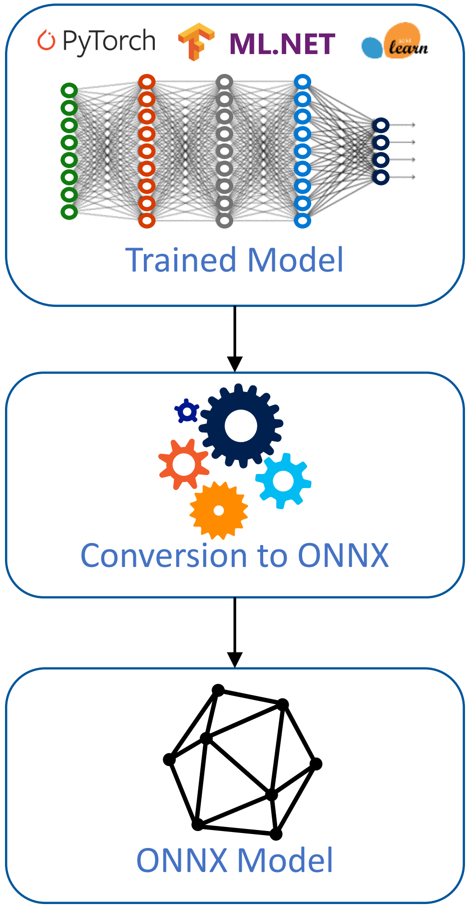
Azure

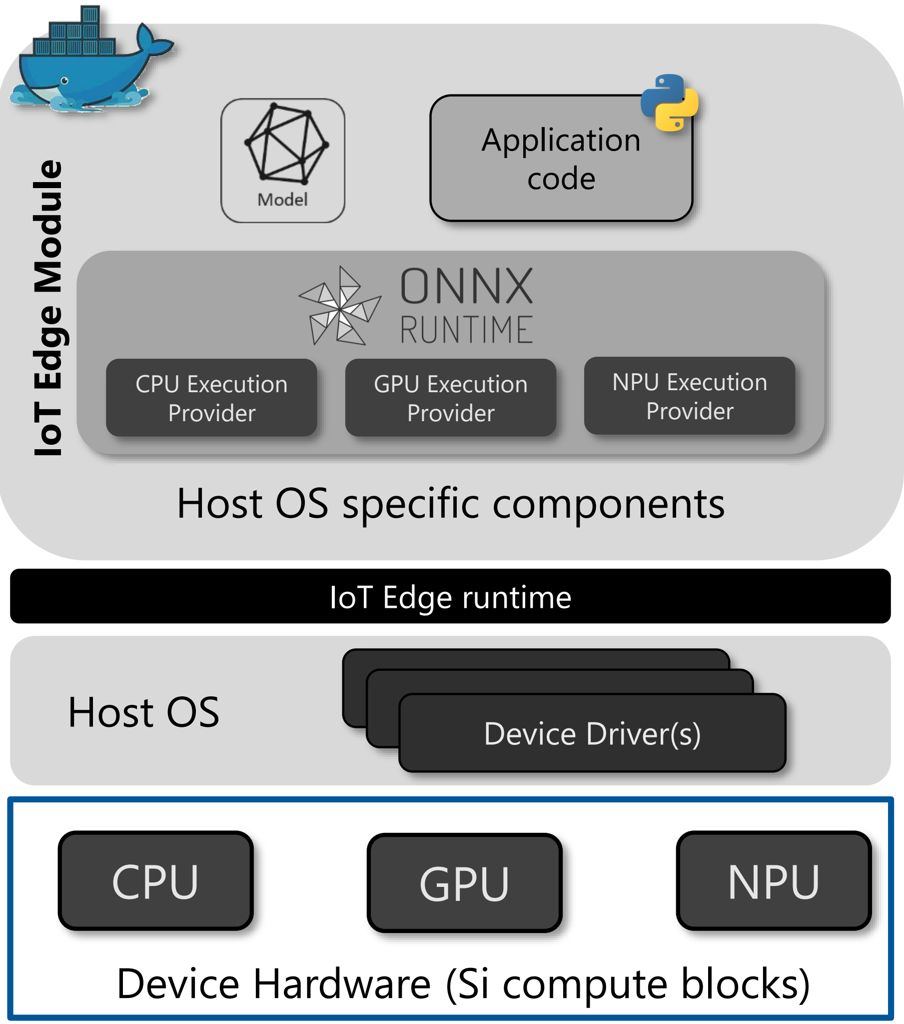
- Azure Machine Learning services
- Ubuntu VM
- Windows Server 2019 VM

Devices

- Windows devices
- IoT Devices
- Other devices (iOS, etc.)

ONNX – a common format for NN graph representation





Model Conversion to ONNX (examples)

```
from keras.models import load_model
import keras2onnx
import onnx

keras_model = load_model("model.h5")

onnx_model = keras2onnx.convert_keras(keras_model,
keras_model.name)

onnx.save_model(onnx_model, 'model.onnx')
```

```
import torch
import torch.onnx

model = torch.load("model.pt")

sample_input = torch.randn(1, 3, 224, 224)

torch.onnx.export(model, sample_input, "model.onnx")
```

```
python -m tf2onnx.convert
         --input frozen_model.pb
         --inputs input_batch:0, lengths:0
         --outputs top_k:1
         --fold_const
         --opset 8
         --output deepcc.onnx
```

```
import numpy as np
import chainer
from chainer import serializers
import onnx_chainer

serializers.load_npz("my.model", model)

sample_input = np.zeros((1, 3, 224, 224), dtype=np.float32)
chainer.config.train = False

onnx_chainer.export(model, sample_input, filename="my.onnx")
```

ONNX
Exporters & Converters

https://github.com/onnx/tutorials

Framework / Tool	Installation	Tutorial
Caffe	apple/coremltools and onnx/onnxmltools	Example
Caffe2	part of caffe2 package	Example
Chainer	chainer/onnx-chainer	Example
Cognitive Toolkit (CNTK)	built-in	Example
CoreML (Apple)	onnx/onnx-coreml and onnx/onnxmltools	Example
Keras	onnx/keras-onnx	Example
LibSVM	onnx/onnxmltools	Example
LightGBM	onnx/onnxmltools	Example
MATLAB	Deep Learning Toolbox	Example
ML.NET	built-in	Example
MXNet (Apache)	part of mxnet package docs github	Example
PyTorch	part of pytorch package	Example, exporting different ONNX opsets, Extending support
SciKit-Learn	onnx/sklearn-onnx	Example
SINGA (Apache) - Github (experimental)	built-in	Example
TensorFlow	onnx/tensorflow-onnx	Examples

ONNX RUNTIME

github.com/microsoft/onnxruntime

ONNX Runtime is a **high-performance inference engine** for machine learning models in the ONNX format

Flexible

Supports full ONNX-ML spec (v1.2-1.5)

Supports both CPU and GPU

C#, C, and Python APIs

Cross Platform

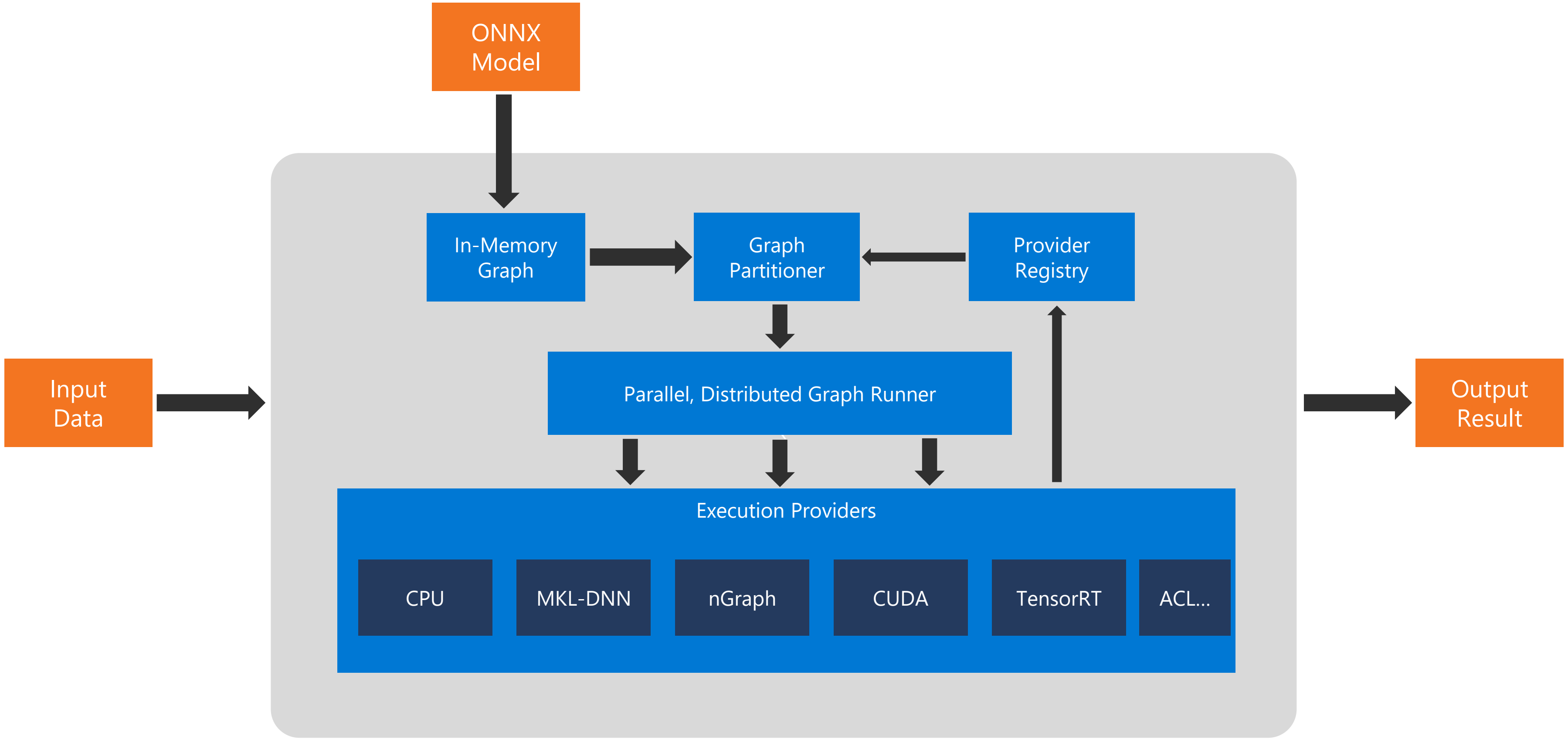
Works on
- Mac, Windows, Linux
- x86, x64, ARM

Also built-in to Windows 10 natively (WinML)

Extensible

Extensible architecture to plug-in optimizers and hardware accelerators

ONNX Runtime Architecture



Using ONNX Runtime – HW agnostic API

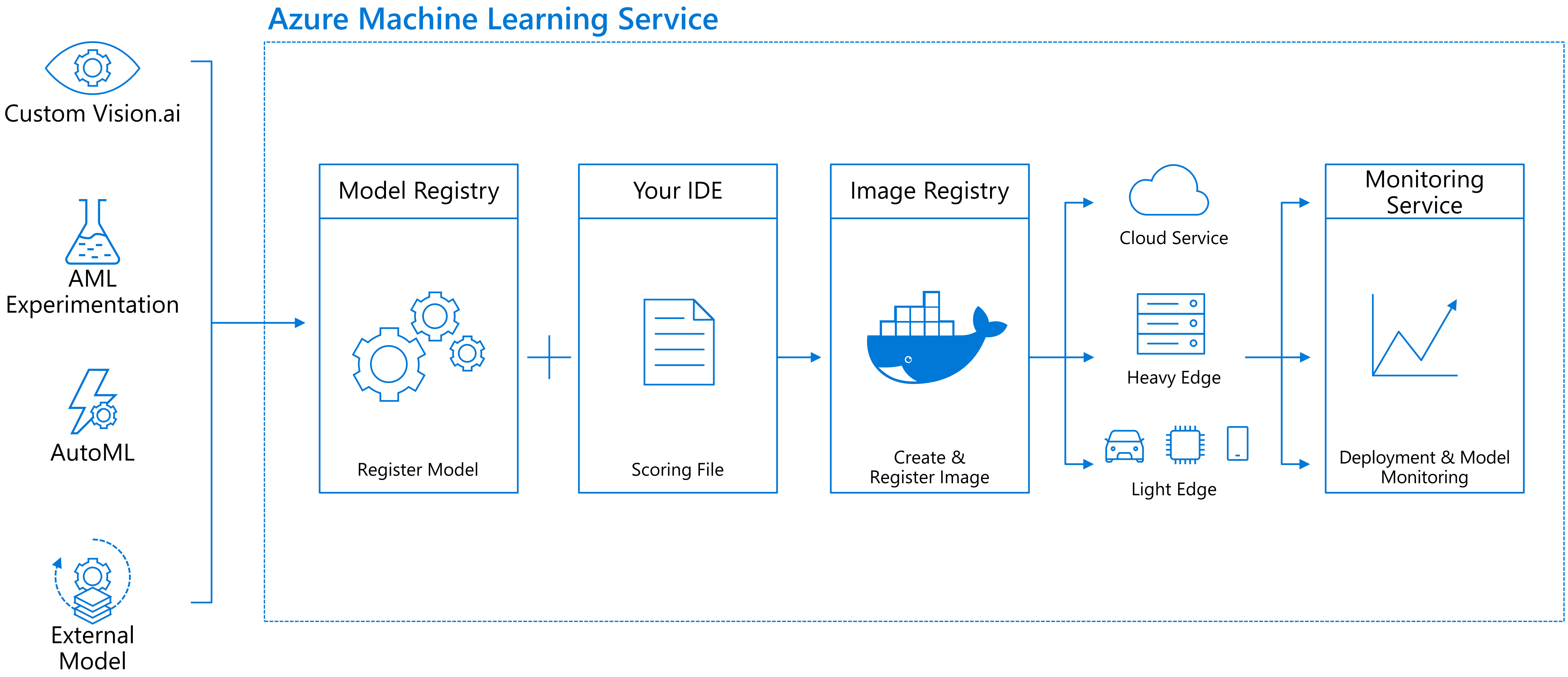
```python
import onnxruntime

session =
onnxruntime.InferenceSession("mymodel.onnx")

results = session.run([], {"input": input_data})
```

```csharp
using Microsoft.ML.OnnxRuntime;

var session = new InferenceSession("model.onnx");

var results = session.Run(input);
```

…… also available for C

Deployment & Execution on the Edge

Deploy Azure ML models at scale



score.py for IoT Edge

Empty function required to satisfy AML-SDK checks

```python
def run(msg):
    # this is a dummy function required to satisfy AML-SDK requirements.
    return msg
```

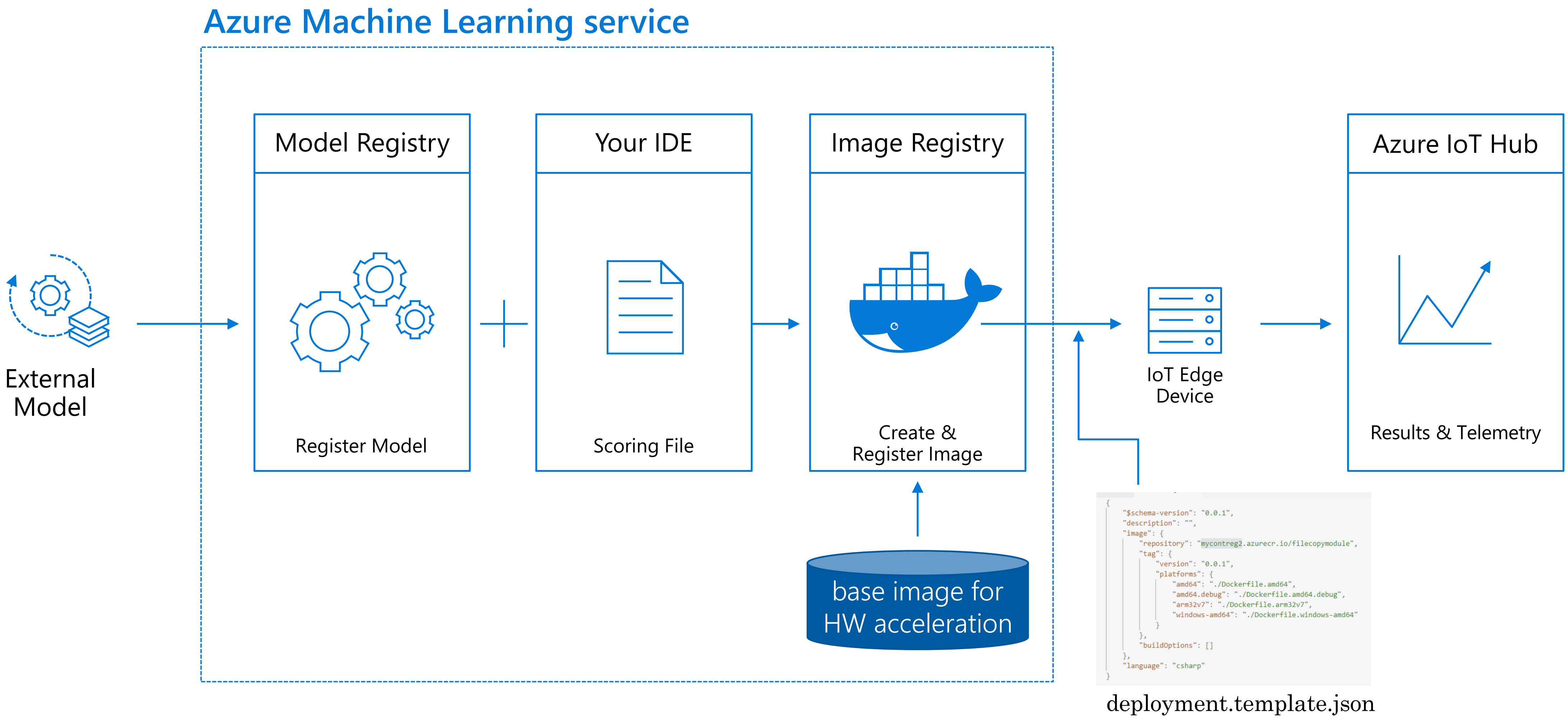
Sample notebook [here]()

ENTRYPOINT for container on IoT Edge device

Loop to read video frame from /device/video0
Run inference on frame
Send result/telemetry to IoT Hub/cloud services

```python
def init():
    # Choose HTTP, AMQP or MQTT as transport protocol.  Currently only MQTT is supported.
    PROTOCOL = IoTHubTransportProvider.MQTT
    DEVICE = 0 # when device is /dev...
    LABEL_FILE = "labels.txt"
    MODEL_FILE = "Model.onnx"
    global MESSAGE_TIMEOUT # setting for IoT H...
    MESSAGE_TIMEOUT = 1000
    LOCAL_DISPLAY = "OFF" # flag for local display on/off, default OFF


    # Create the IoT Hub Manager to send message to IoT Hub
    print("trying to make IOT Hub manager")

    hub_manager = HubManager(PROTOCOL)

    if not hub_manager:
        print("Took too long to make hub_manager, exiting program.")
        print("Try restarting IotEdge or this module.")
        sys.exit(1)

    # Get Labels from labels file
    labels_file = open(LABEL_FILE)
    labels_string = labels_file.re...
    labels = labels_string.split('...
    labels_file.close()
    label_lookup = {}
    for i, val in enumerate(labels...
        label_lookup[val] = i

    # get model path from within the contai...
    model_path=Model.get_model_path(MODEL_F...

    # Loading ONNX model

    print("loading model to ONNX Runtime...
    start_time = time.time()
    ort_session = rt.InferenceSession(m...th)
    print("loaded after", time.time()-...time,"s")

    # start reading frames from video ...dpoint

    cap = cv2.VideoCapture(DEVICE)

    while cap.isOpened():
        _, _ = cap.read()
        ret, img_frame = cap.read()
        if not ret:
            print('no video RESETTING FRAMES TO 0 TO RUN IN LOOP')
```

Deploy from Azure ML to IoT Edge



End-to-End Pipeline for AI on the Edge

Re-cap the scenario

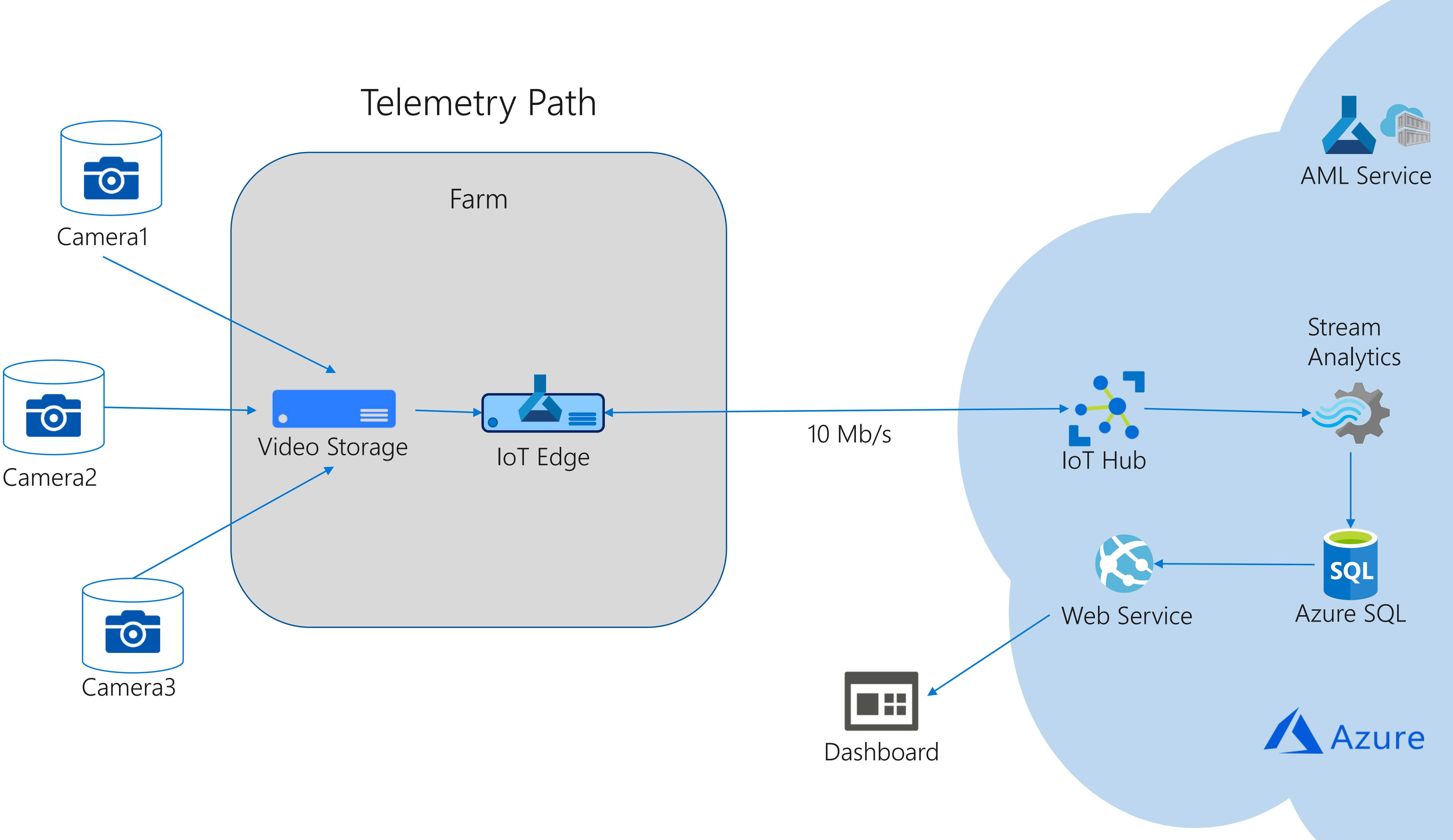
Scenario

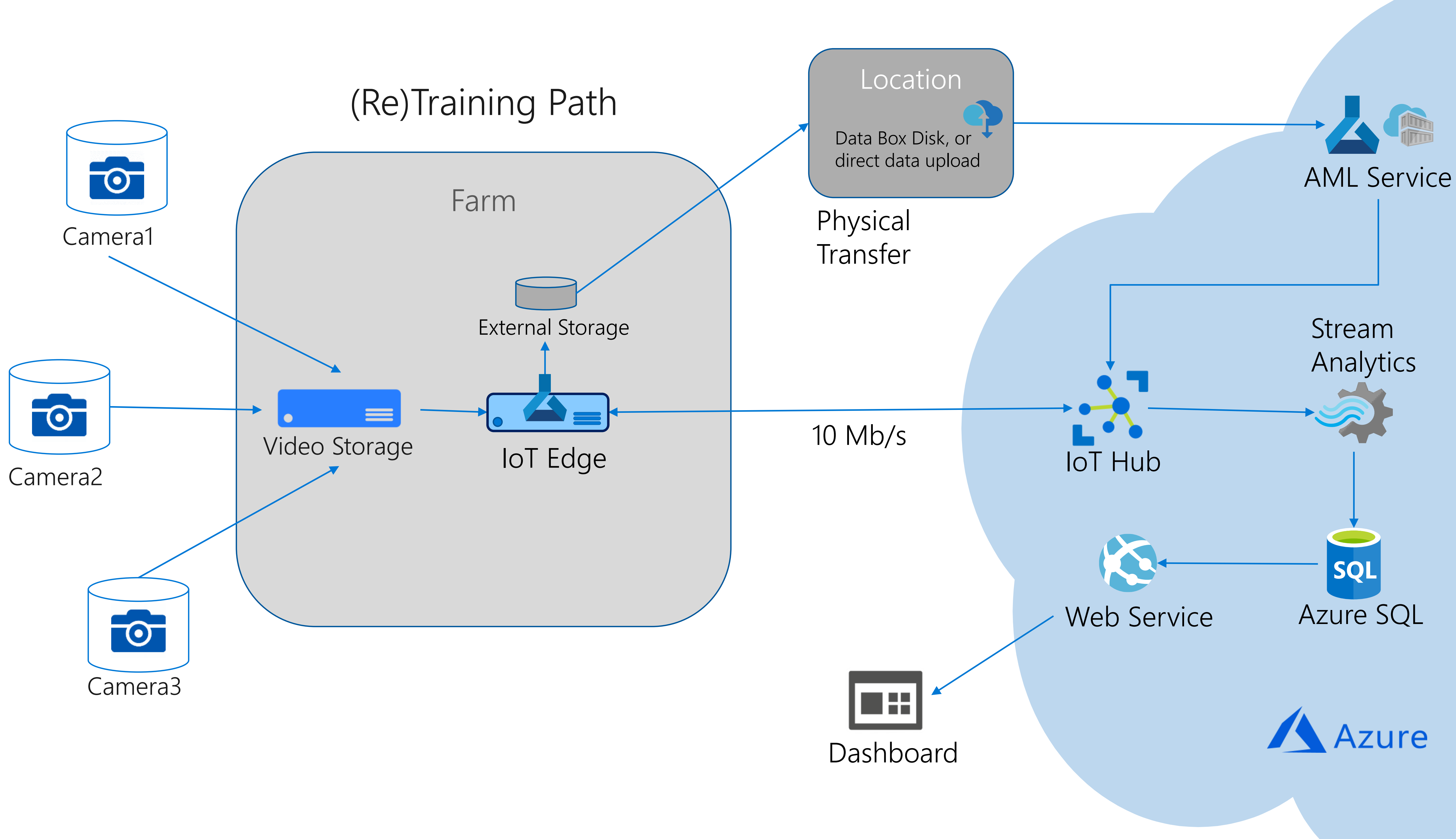
- Produce harvesting – use image classification to estimate harvesting oranges

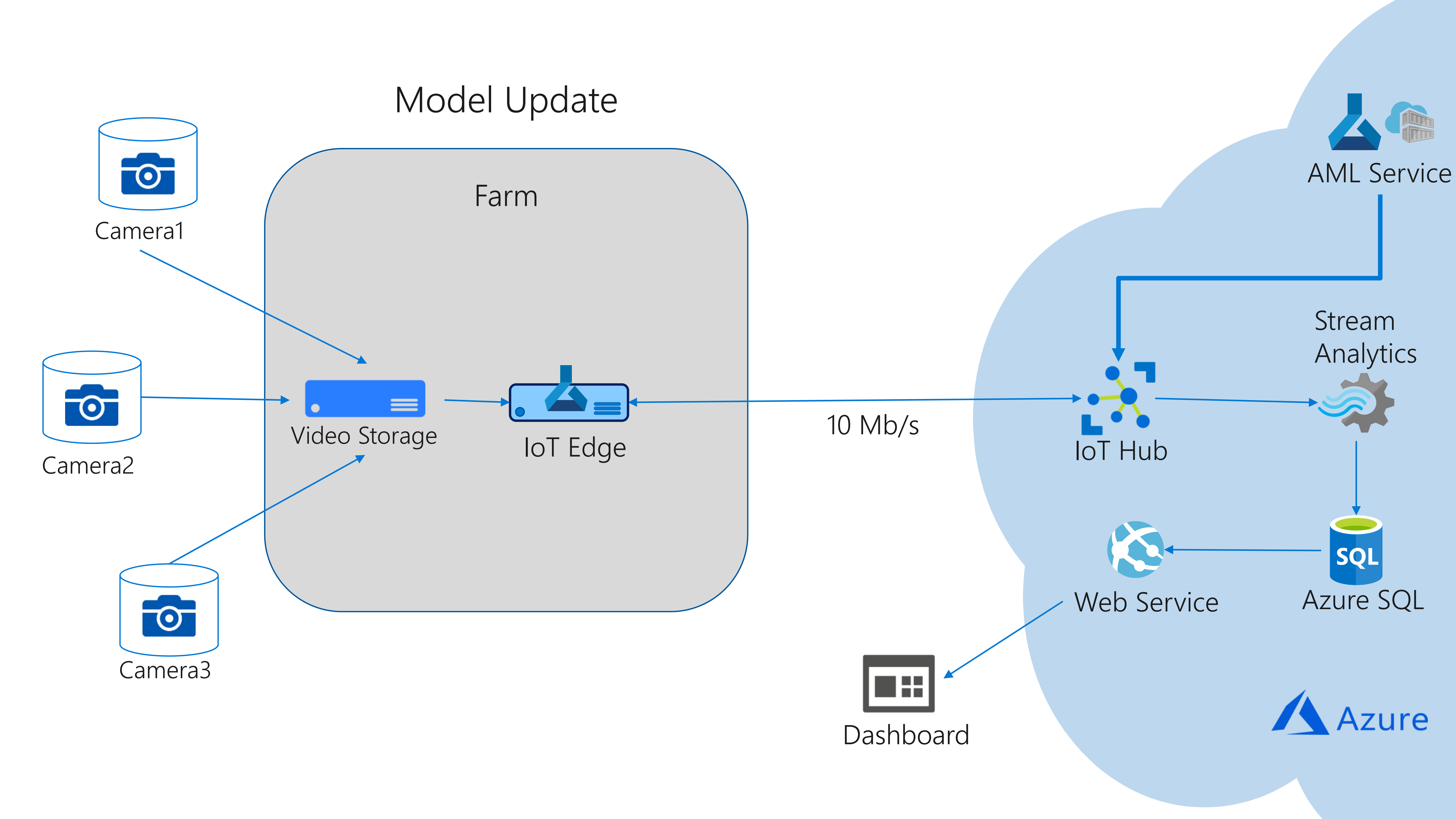
Goals

- Run AI model on farm (edge) to produce orange weights that are transferred to the cloud for analysis
- Include process for collecting training data to allow the model to improve over time
- Build the processes to improve collaboration between the Data Science and IoT dev ops teams

Telemetry Path




(Re)Training Path
Camera1
Camera2
Camera3
Farm
External Storage
Video Storage
IoT Edge
Location
Data Box Disk, or
direct data upload
Physical
Transfer
10 Mb/s
AML Service
IoT Hub
Stream
Analytics
Web Service
Azure SQL
SQL
Dashboard
Azure


Model Update
Camera1
Camera2
Camera3
Farm
Video Storage
IoT Edge
10 Mb/s
AML Service
Stream Analytics
IoT Hub
SQL
Web Service
Azure SQL
Dashboard
Azure

THANK YOU

Sample notebook for single container implementation: here