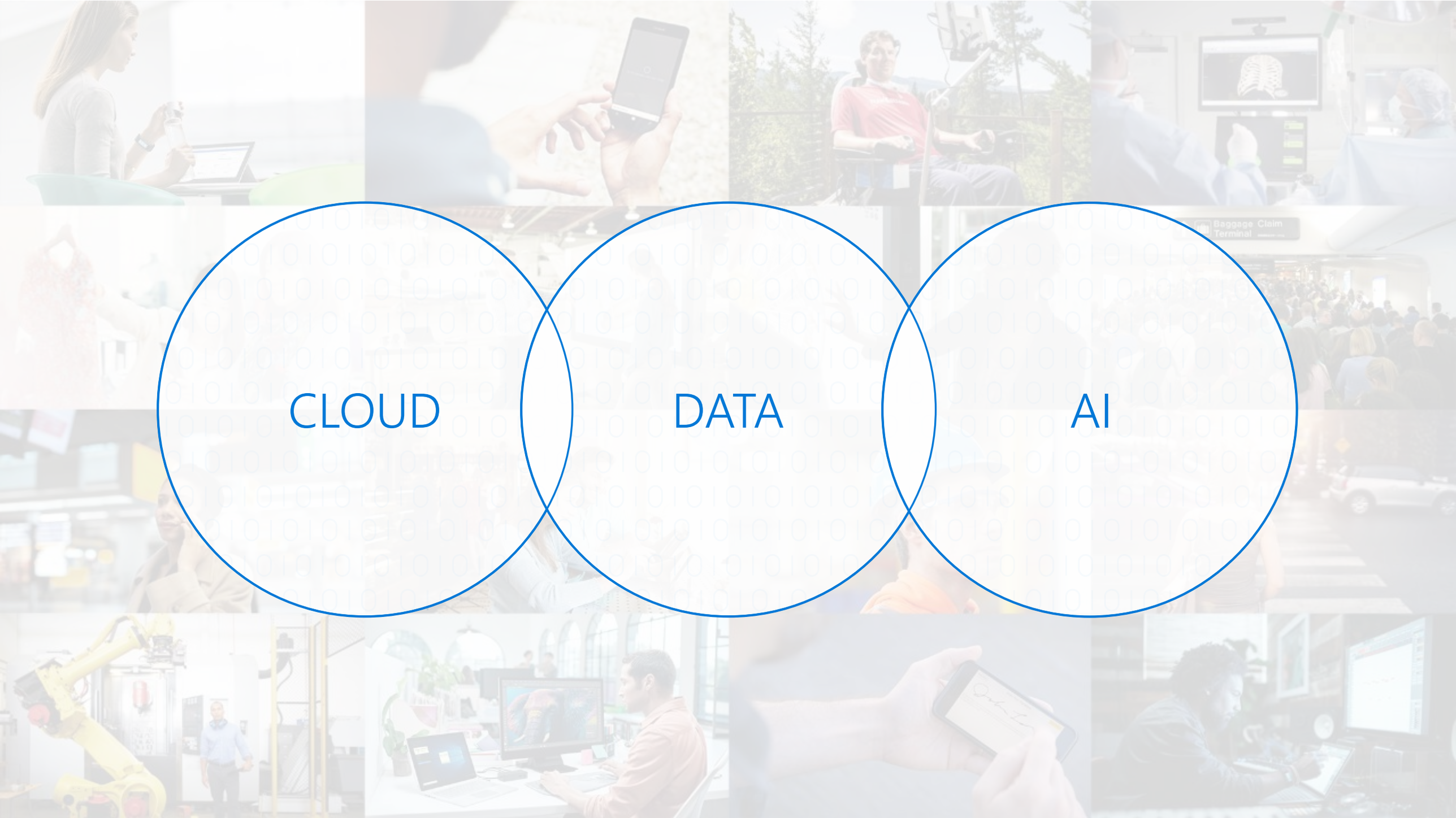
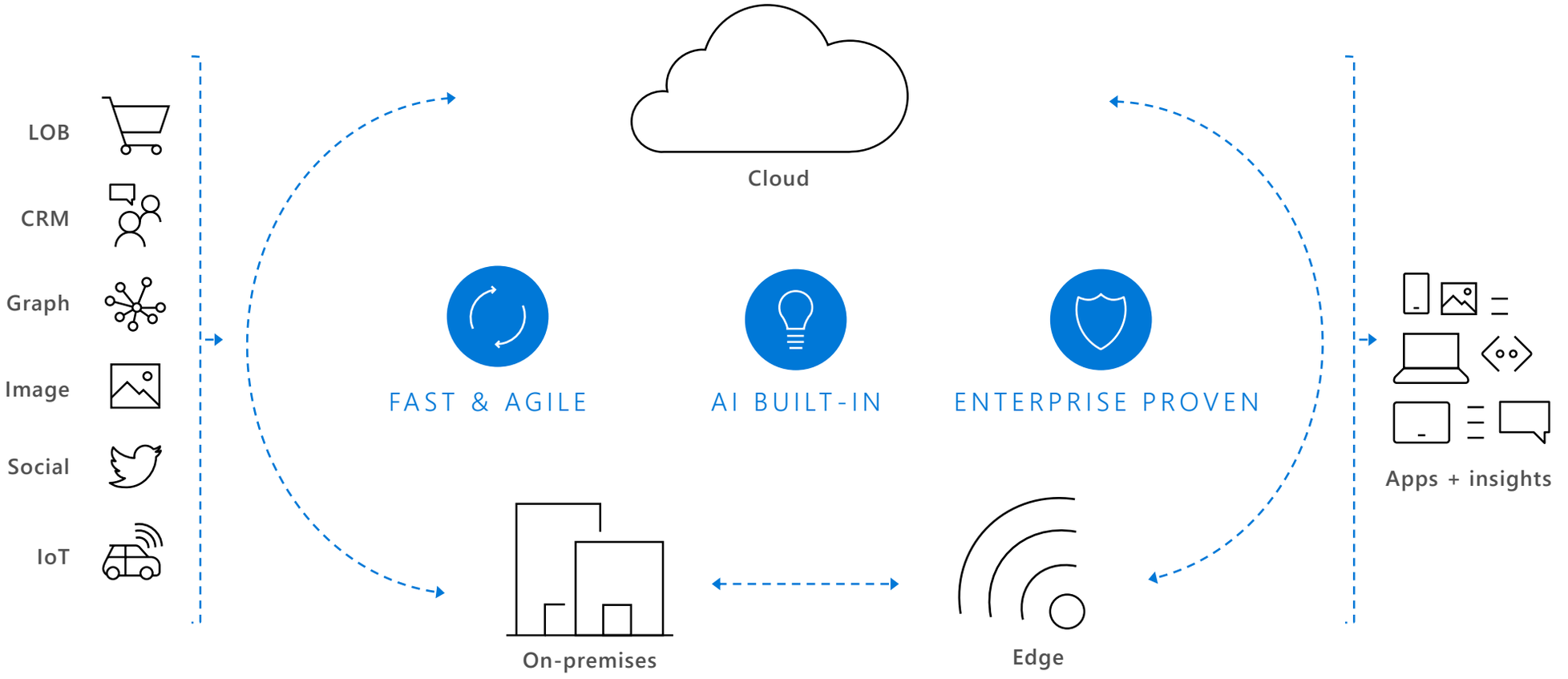
Microsoft

Azure Machine Learning

Kris Bock


CLOUD
DATA
AI

MICROSOFT AI PLATFORM



DATA SCIENCE & AI

KEY TRENDS

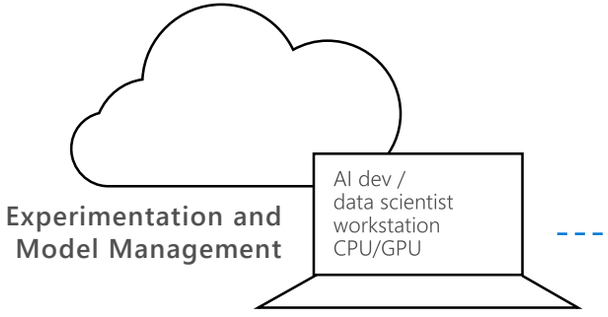
- Accelerating adoption of AI by developers (consuming models)
- Rise of hybrid training and scoring scenarios
- Push scoring/inference to the event (edge, cloud, on-prem)
- Moving high-end developers into deep learning as non-traditional path to DS / AI dev
- Growth of diverse hardware arms race across all form factors (CPU / GPU / FPGA / ASIC / device)
- Demonstrating success of transfer learning techniques while reducing dev complexity

CHALLENGES

- Data prep
- Model deployment & management
- Model lineage & auditing
- Explain-ability

AZURE MACHINE LEARNING

AZURE MACHINE LEARNING SERVICES



Experimentation and Model Management

Notebooks
IDEs
Azure Machine Learning Workbench

TRAIN & DEPLOY OPTIONS

AZURE

- Spark
- SQL Server
- Virtual machines
- GPUs
- Container services

ON-PREMISES

- SQL Server
- Machine Learning Server

EDGE COMPUTING

- Azure IoT Edge

THE ML & AI PLATFORM

DEVELOPER TOOLS & SERVICES

AI Applications (1st & 3rd party) Cognitive Services Bot Framework

OPEN PLATFORM FOR DATA SCIENCE

Azure Machine Learning

Experimentation management, data prep, & collaboration

Machine Learning toolkits

CNTK Scikit-Learn
Tensorflow Other Libs.
ML Server PROSE

Model deployment & management

Docker
Cloud – Spark, SQL, other engines
ML Server – Spark, SQL, VMs
Edge

HYPERSCALE, ENTERPRISE-GRADE INFRASTRUCTURE

Software

Spark AI Batch Training DS VM SQL Server ACS

Storage management

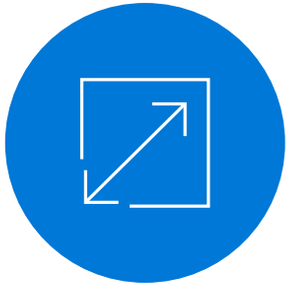
BLOB Cosmos DB SQL DB/DW ADLS

Hardware

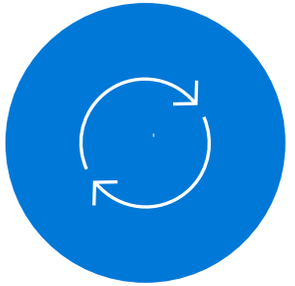
CPUs FPGA GPUs IoT

Bring AI everywhere

Benefit from the fastest AI developer cloud



Build, deploy, and manage at scale



Boost productivity with agile development

Begin building now with the tools and platforms you know

Build, deploy, and manage at scale

Build and deploy everywhere – cloud, on-premises, edge, and in-data

Deploy in minutes with data driven management and retraining of all your models

Prototype locally then scale up and out with VMs, Spark clusters, and GPUs

Real time, high through-put insights everywhere, including Excel integration

Manage models

Deployment and management of models as HTTP services

Container-based hosting of real time and batch processing

Management and monitoring through Azure
(e.g., AppInsights)

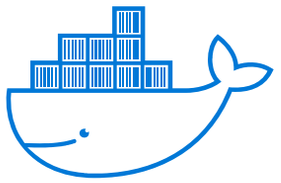
First class support for SparkML, Python, CNTK, TF, TLC, R,
extensible to support others (Caffe, MXnet)

Service authoring in Python and .NET Core

```
# -n app name
# -f scoring script file name
# -m dependencies, in this case it is the pickled model file
# -r type of model, in this case it is the scikit-learn model
$ az ml service create realtime -n irisapp -f iris_score.py -m model.pkl -r scikit-py
```

```
$ az ml service run realtime -n irisapp -d '{"input": "[[1.0, 2.2, 2.3, 2.7]]"}'
2
```

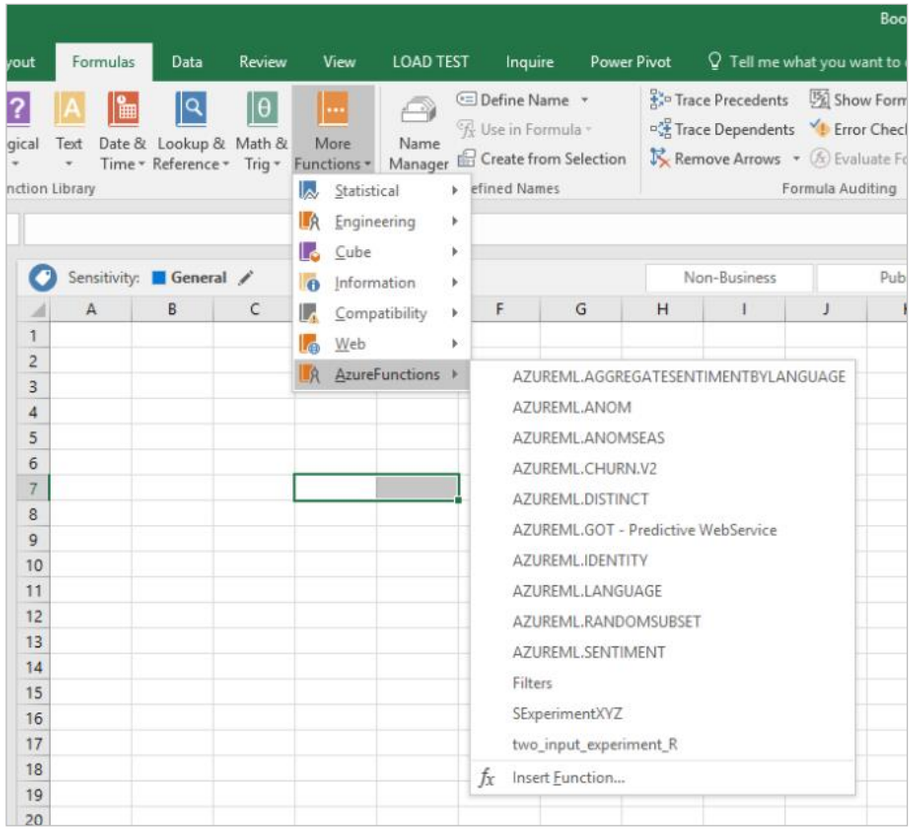
Deploy everywhere

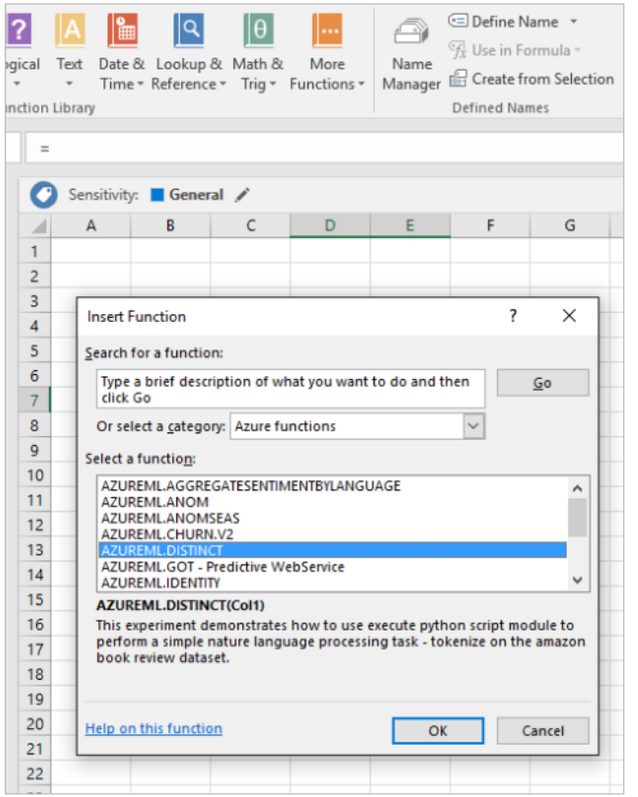


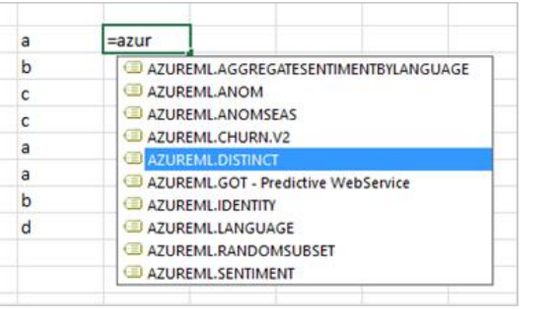
DOCKER

- Single node deployment (cloud/on-prem)
- Azure Container Service
- Azure IoT Edge
- Spark clusters

Consume Azure Machine Learning models in Excel







Boost productivity with agile development

Spend more time modeling and less time prepping with built-in intelligent data wrangling

Increase collaboration and sharing with notebooks and Git

Avoid losing anything with version control and reproducibility

Know the best performing models with metrics, lineage, run history, asset management, and more

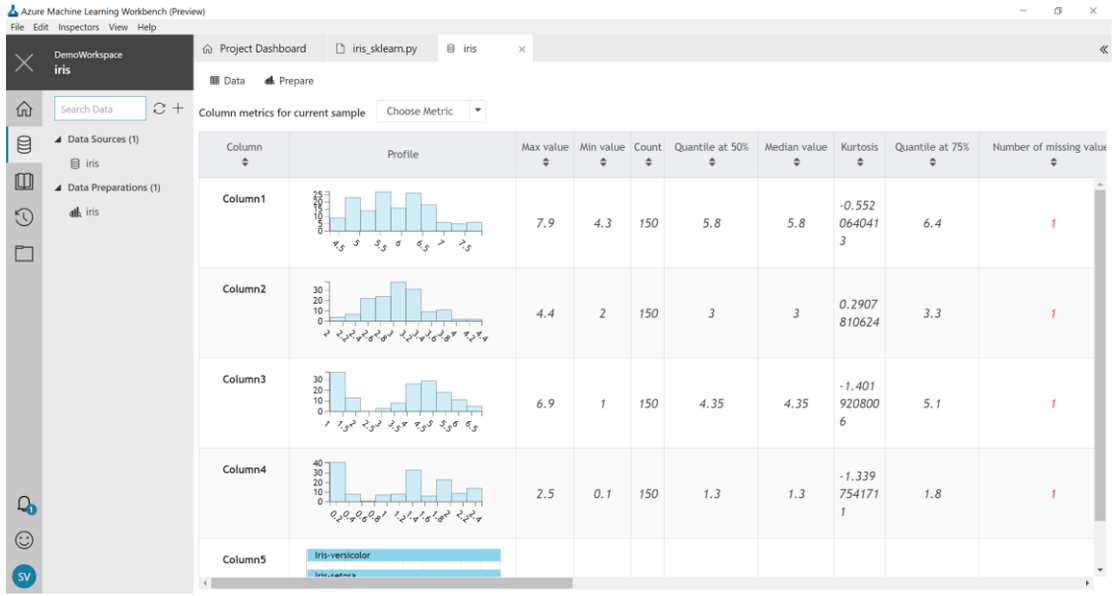
Prep data

Sample, understand, and prep data rapidly

Leverage PROSE SDK and more for intelligent, data prep by example

Extend/customize transform and featurize through Python

Generate Python and PySpark for execution at scale



Experiment

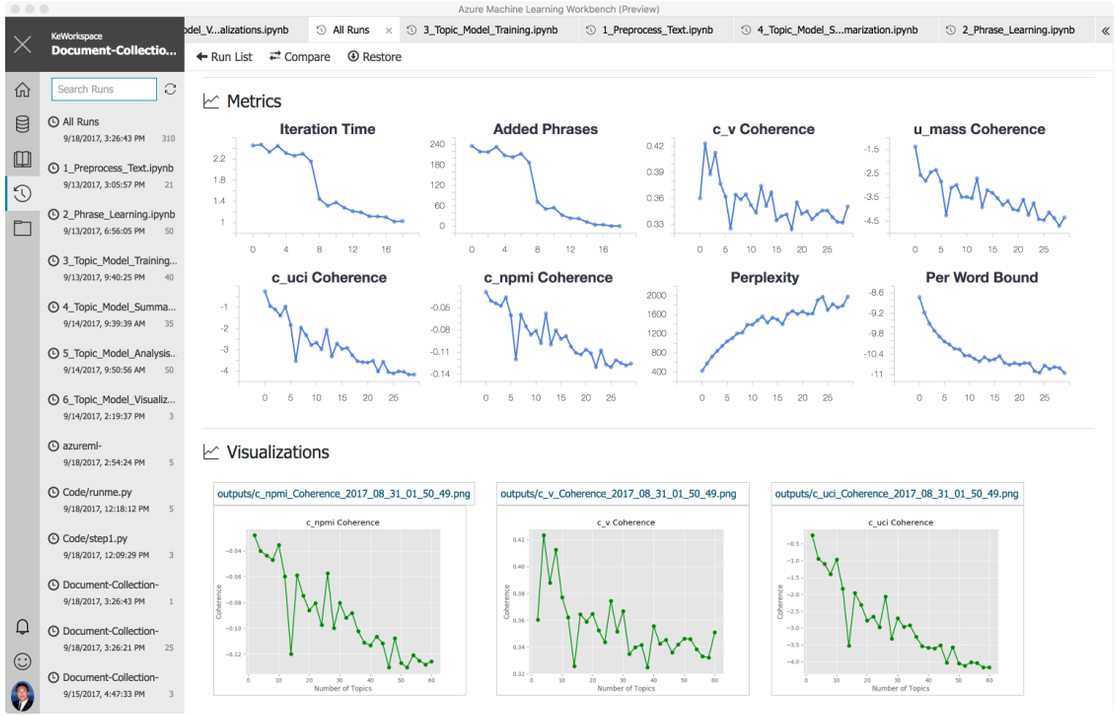
Manage job for local and cloud experiments

Find support for Spark + Python + R (roadmap)

Execute jobs locally, on remote VMs (scale up),
on Spark clusters (scale out), or SQL on-premises

Create with Git-backed experimentation tracking of code,
config, parameters, and data

Discover and compare with detailed historical metadata



Begin building now with the tools and platforms you know

Choose between visual drag-and-drop or code-first authoring

Use your favorite IDE

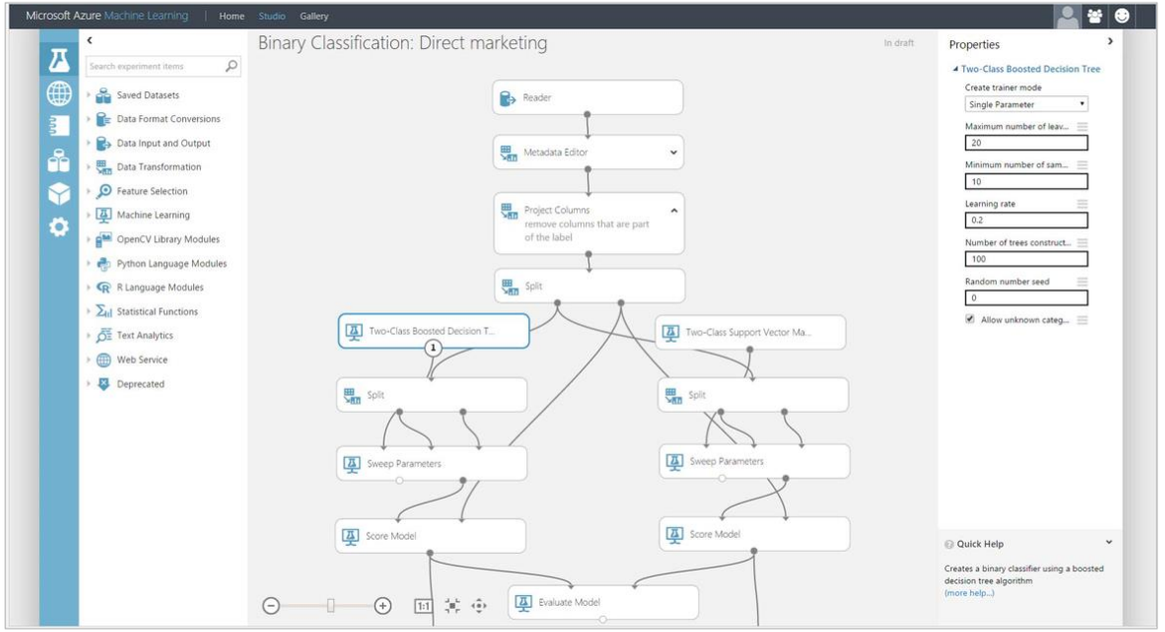
Call Azure Machine Learning services directly in VS Code*

Build on any framework or library with the most popular languages

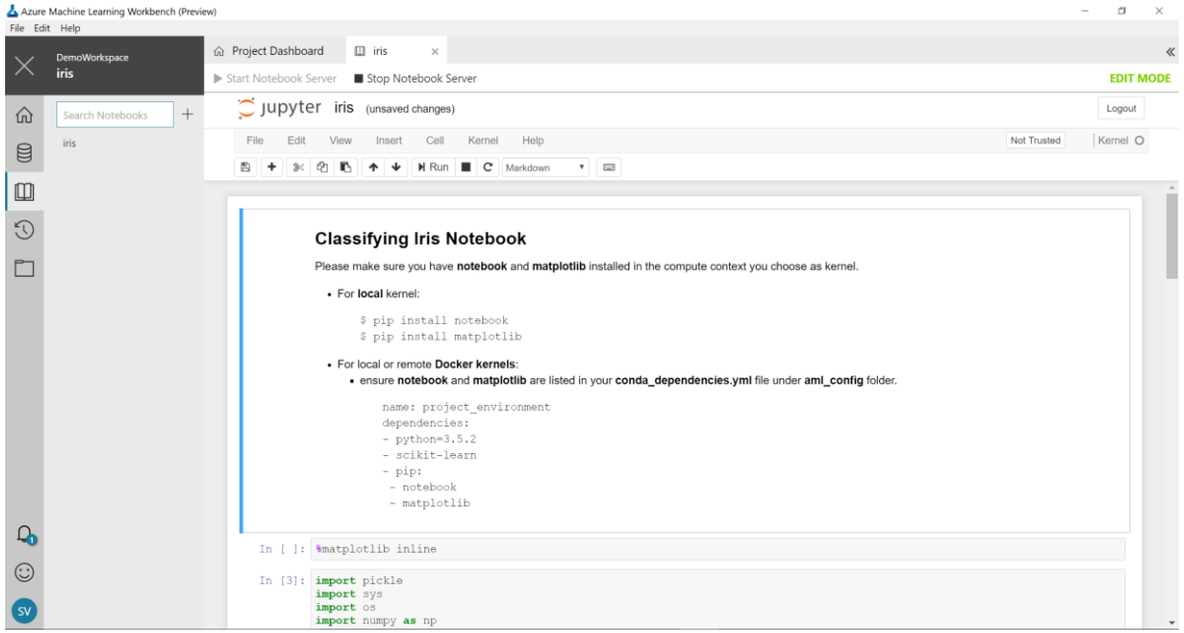
Train quicker and easier with industry-leading Spark and GPUs

*More IDEs coming soon

Build as you like



VISUAL DRAG-AND-DROP



CODE-FIRST

Use what you want

Use your favorite IDE

Leverage all types of data

USE ANY FRAMEWORK OR LIBRARY

TensorFlow Microsoft CNTK Apache Spark ML

USE ANY TOOL

CODE Visual Studio jupyter PyCharm

USE THE MOST POPULAR INNOVATIONS

git Apache Spark

docker python R

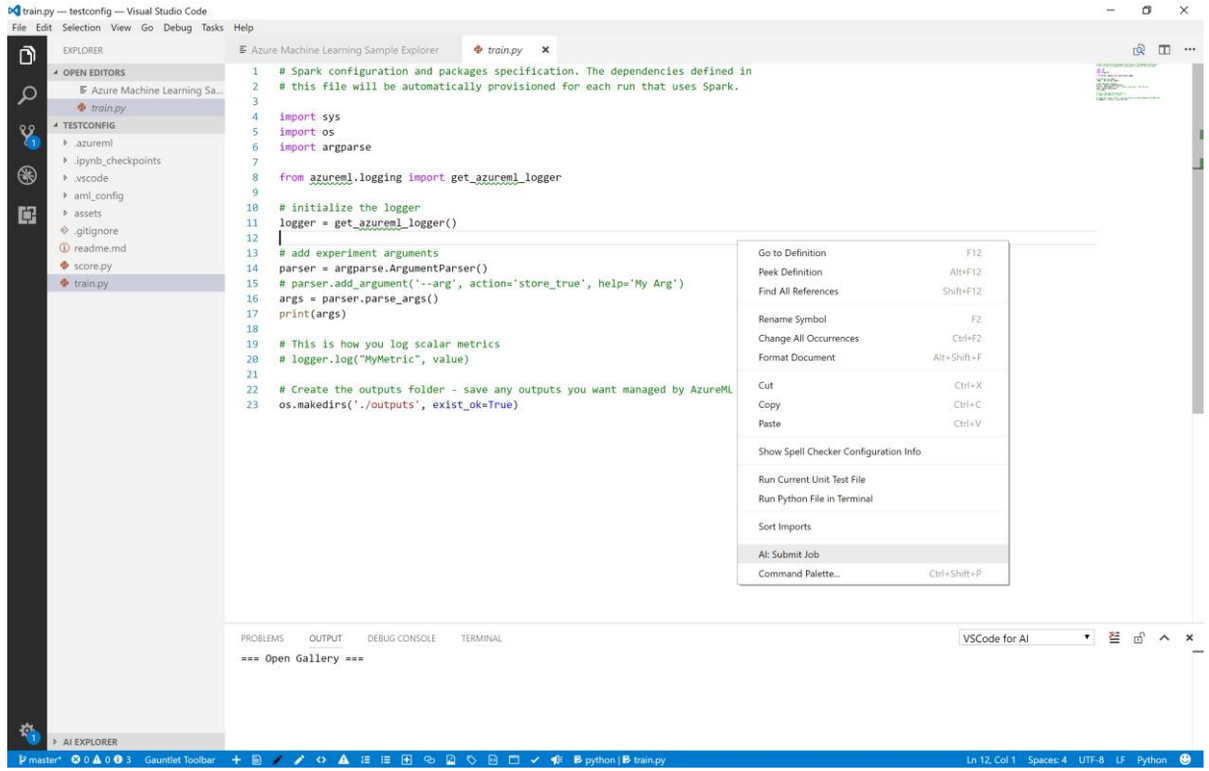
Integrated into your IDE

Visual Studio Code extension (more IDEs and notebooks on roadmap)

Begin building with your IDE - no extra tool needed

Integrated rich authoring for machine learning and deep learning

Call Azure Machine Learning services straight from your IDE or notebook



CODE

packages

Azure ML Packages Preview

Python packages for computer vision, Forecasting and Text Analytics

Easily build and deploy models on Azure Machine Learning

Provides high level APIs for data preparation, augmentation, training, evaluating and deployment.

Model experimentation and comparison, run history, model management and deployment through Azure ML

Computer Vision Simplified with Azure ML Packages

Before → Code example – Keras

Training a small CNN, screenshots taken from the file "cifar10_cnn.py" in the Keras github repository.

```
# The data, split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_da
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_clas
y_test = keras.utils.to_categorical(y_test, num_classe

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
else:
    print('Using real-time data augmentation.')
    # This will do preprocessing and realtime data augmentation:
    datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images

    # Compute quantities required for feature-wise normalization
    # (std, mean, and principal components if ZCA whitening is applied).
    datagen.fit(x_train)

    # Fit the model on the batches generated by datagen.flow().
    model.fit_generator(datagen.flow(x_train, y_train,
                                     batch_size=batch_size),
                        epochs=epochs,
                        validation_data=(x_test, y_test),
                        workers=4)

# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)

# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

After → Code example – Azure ML Package for Computer Vision

Transfer learning example using the AML Package for Computer Vision.

```
# create a dataset from a directory with subfolders
dataset = Dataset.create_from_dir(dataset_name, dataset_location)

# split the full dataset into a train and test set
train_set_orig, test_set = dataset.split(train_size=.8, stratify='label')

# Image augmentation
rotate_and_flip = augmenters.Sequential([augmenters.Affine(rotate=(-45, 45)),
                                         augmenters.Fliplr(.5)])
crop = augmenters.Sequential([augmenters.Crop(percent=(0, .1))])
train_set = augment_dataset(train_set_orig, [rotate_and_flip, crop])

# create the model
model = TransferLearningModel(num_classes = num_classes,
                              base_model_name = 'ResNet18_ImageNet_CNTK')

# train the model
model.train(num_epochs=45, mb_size=16)

# return the accuracy
ce = ClassificationEvaluation(model, test_set, minibatch_size=16)
acc = ce.compute_accuracy()
```

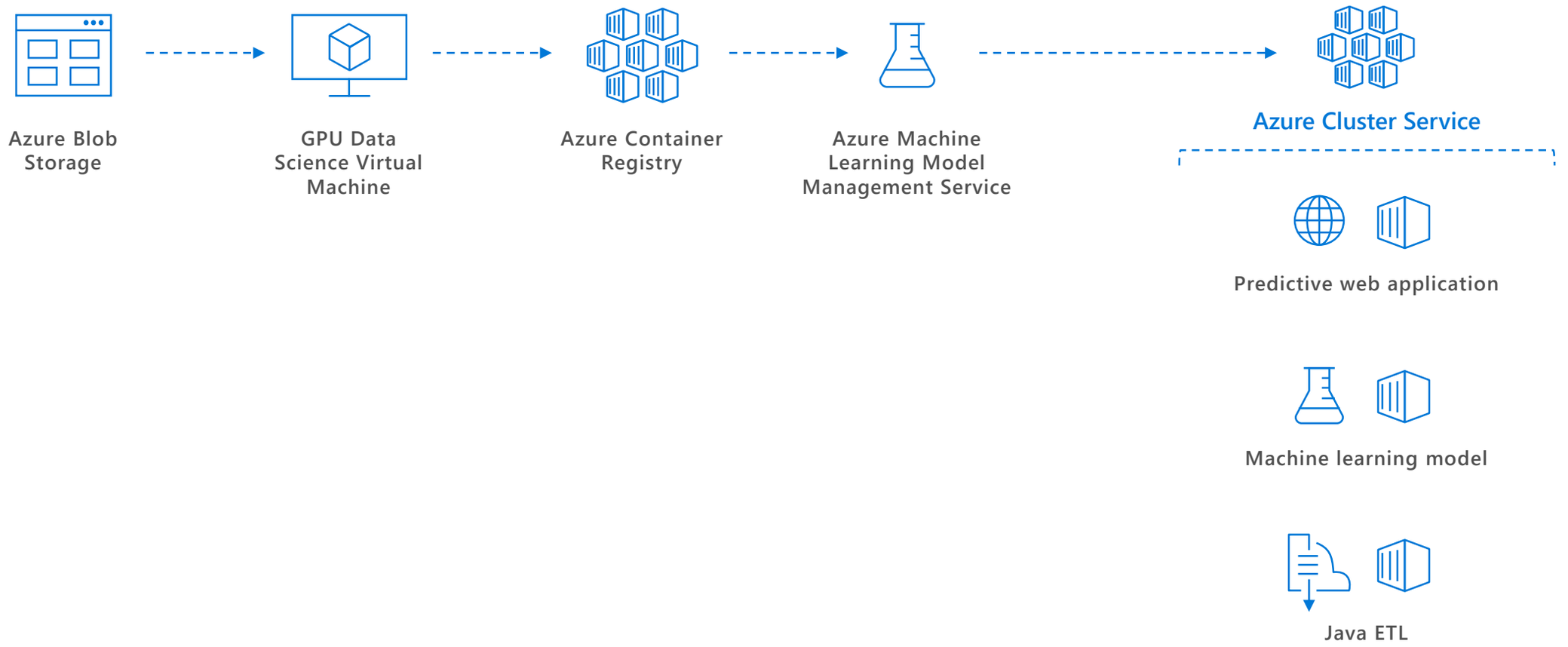
packages

https://aka.ms/aml-packages

SAMPLE PATTERNS

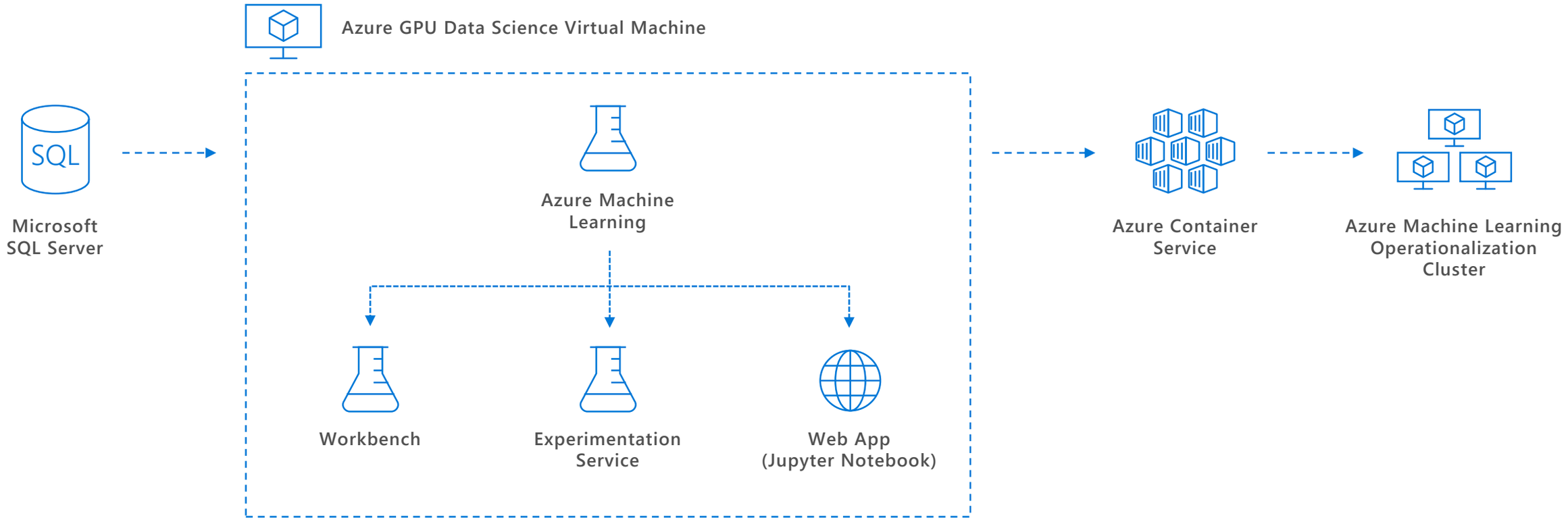
Enhanced image classification

Transfer learning, convolutional neural networks and gradient-boosting decision tree learning algorithms have redefined image classification



Improved text prediction

Deep learning and natural language processing boosts search efficacy and tagging accuracy



CUSTOMER EVIDENCE

Drone-based electric grid inspector powered by deep learning

Challenge

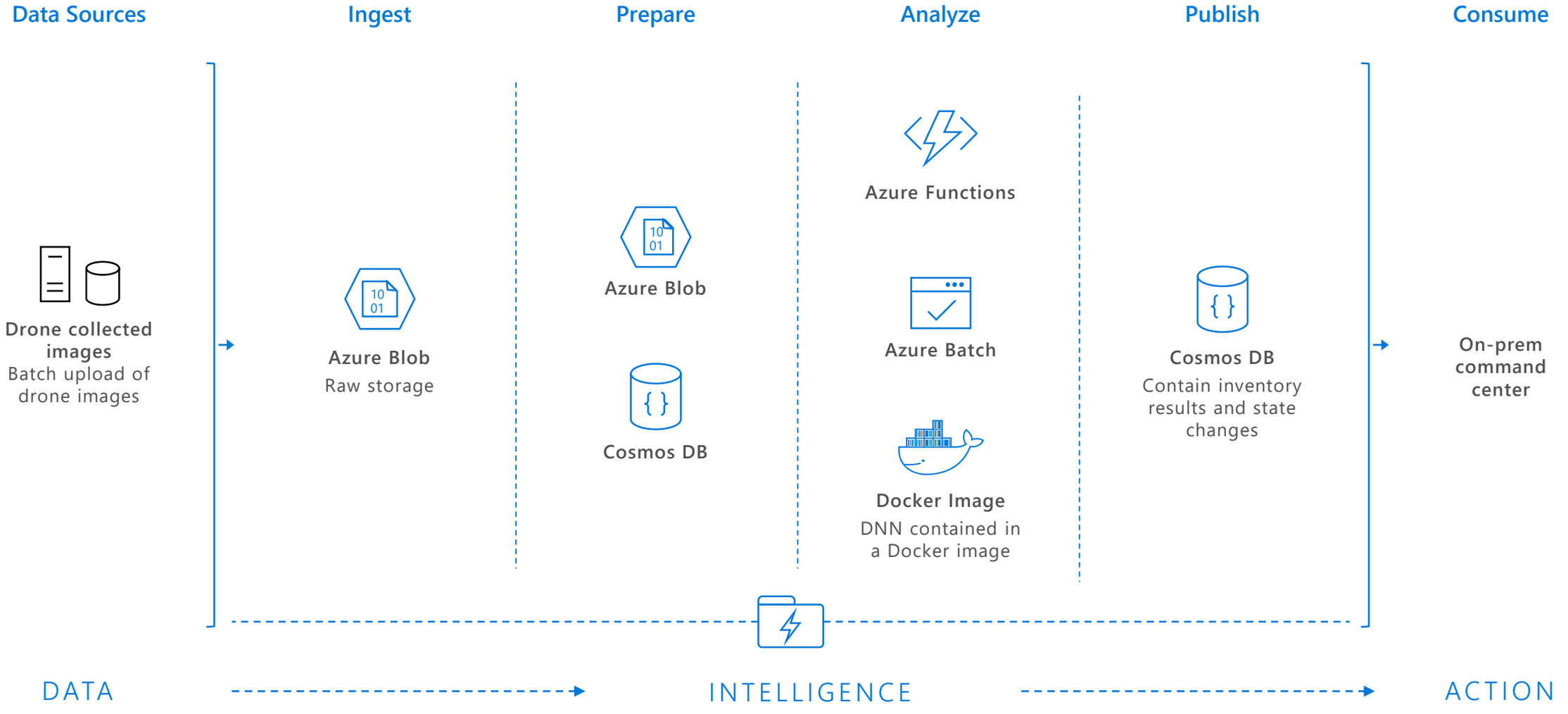
- Traditional power line inspection services are costly
- Demand for low cost image scoring and support for multiple concurrent customers
- Needed powerful AI to execute on a drone solution

Solution

- Deep learning to analyze multiple streaming data feeds
- Azure GPUs support Single Shot multibox detectors
- Reliable, consistent, and highly elastic scalability with Azure Batch Shipyards

eSmart SYSTEMS

eSmart architecture



Increased match accuracy with image analysis

Challenge

- Assist buyer search by providing accurate options of similar clothing items from catalogue
- Need for improved smart-image matching capability based on color, pattern, neck style, etc.

Solution

- Training data created using Bing and domain-specific images
- Used transfer learning to leverage pre-trained ImageNet deep neural network
- Accurate list of most similar clothing items using similarity metrics for apparel
- Match accuracy of 74%

brillio

Pricing

Experimentation service (Public preview)

- First 2 seats: Free
- Seats 3 and above: $50 per seat per month

Model Management service (Public preview)

Managed models	Deployed models	Cores	Price (monthly)
20	2	4	Free
100	10	16	$50
1,000	100	120	$375
10,000	1,000	800	$2,500

Azure Machine Learning Studio and Production Web (GA)

- API pricing https://azure.microsoft.com/en-us/pricing/details/machine-learning/

All services can be used independently

Try it for free

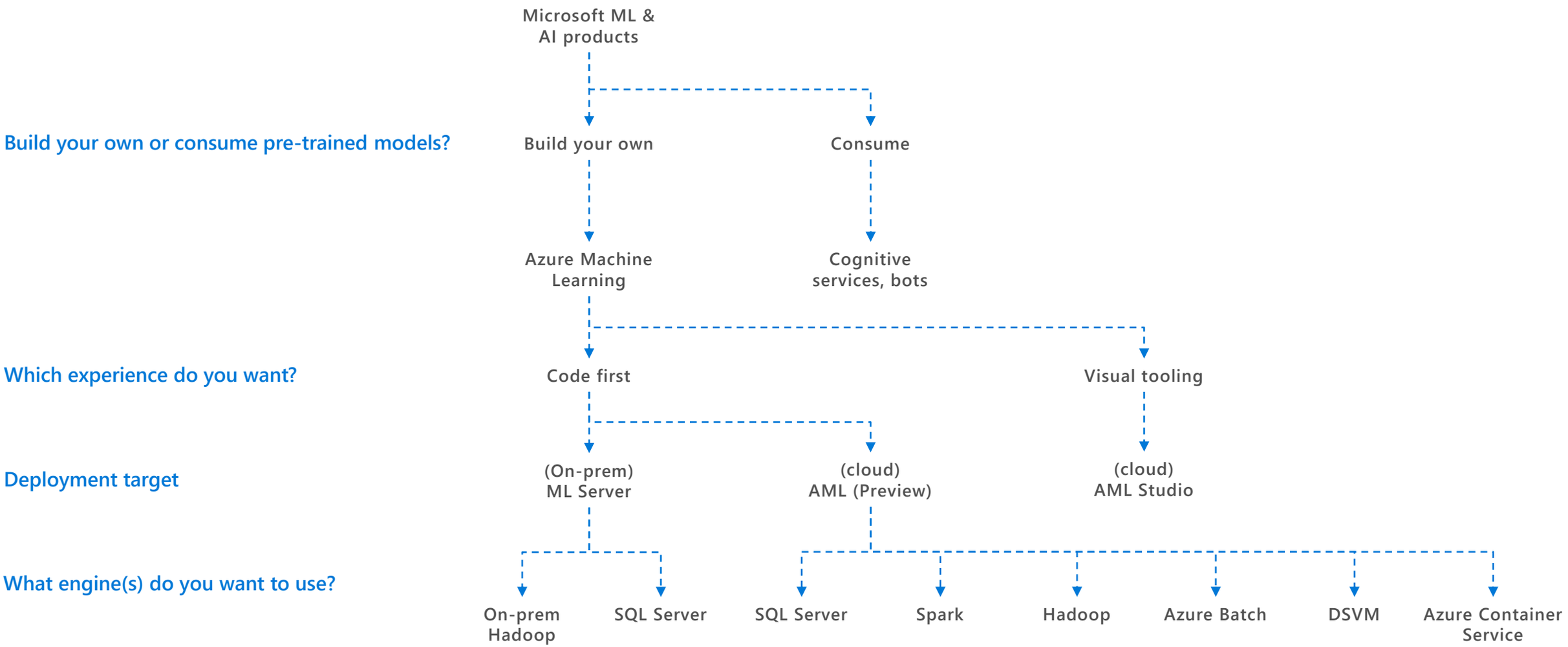
Learn more

Start now

APPENDIX

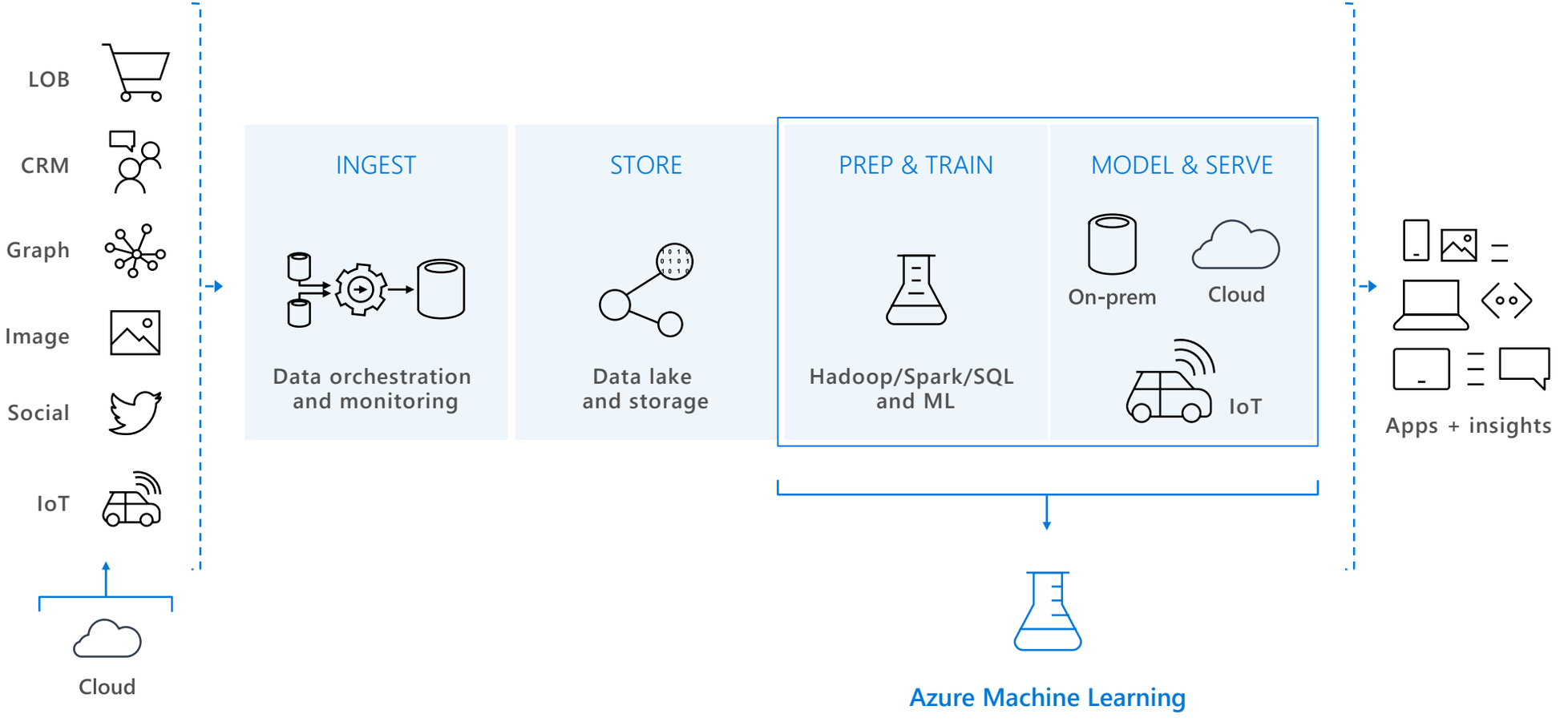
Machine Learning & AI Portfolio

When to use what?



EVOLUTION OF THE DATA ESTATE

Increasing data volumes. New data sources and types. Open Source languages.



The growing data and AI ecosystem

Customers

nielsen
hp
JPMorgan Chase & Co.
Standard Bank
citi
eSmart Systems
OTIS
Hendrick Motorsports
Rolls Royce
Portland Trail Blazers
Nestlé
Expedia
Walmart
Emirates
ECOLAB

System integrators

NEWSIGNATURE
BLUE GRANITE Business Insight. Delivered.
HCL
KPMG
NEAL ANALYTICS
Cognizant
accenture High performance. Delivered.
TATA
TATA CONSULTANCY SERVICES

ISVs

Publicis Worldwide
brillio
OSIsoft.
THOMSON REUTERS
acxiom
Ken Sci
CrowdFlower
Affine Data ▸ Insights ▸ Transformation
Schneider Electric
Milliman
Adobe
Johnson Controls
ABB
Epic

Training partners

dataT.be
Presciient
Project Botticelli
Lander Analytics Navigating Data Science
endjin
KARVY
datascienceDojo
slalom
∞ Numbers
U2U
Hitachi Solutions

FY2017 summary

Accelerating growth through adoption, engagement, and revenue

Our technology enables

- the developer to easily integrate AI in their apps
- the emerging data scientist to get started easily
- the professional data scientist to leverage innovation of their choice from an increasingly diverse ecosystem

Our roadmap is focused on

- End-to-end management of the data science lifecycle
- Simplifying model deployment and management, in the cloud, on premises, in the edge, and into data engines
- Enabling the latest in hardware innovation, delivered at enterprise-grade hyperscale around the world
- Developing increasingly targeted content (templates, industry solutions, and models) to reduce development time

Our technology is uniquely enhanced by

- Decades of industry leading research at Microsoft Research
- Lessons and code from running some of the biggest first party workloads on the planet (Bing, Xbox, Skype, Windows, Office)

MICROSOFT MACHINE LEARNING FOR APACHE SPARK

Apache Spark

A fault-tolerant distributed computing framework

Generalizes, optimizes, and combines Map-Reduce and SQL style-operations

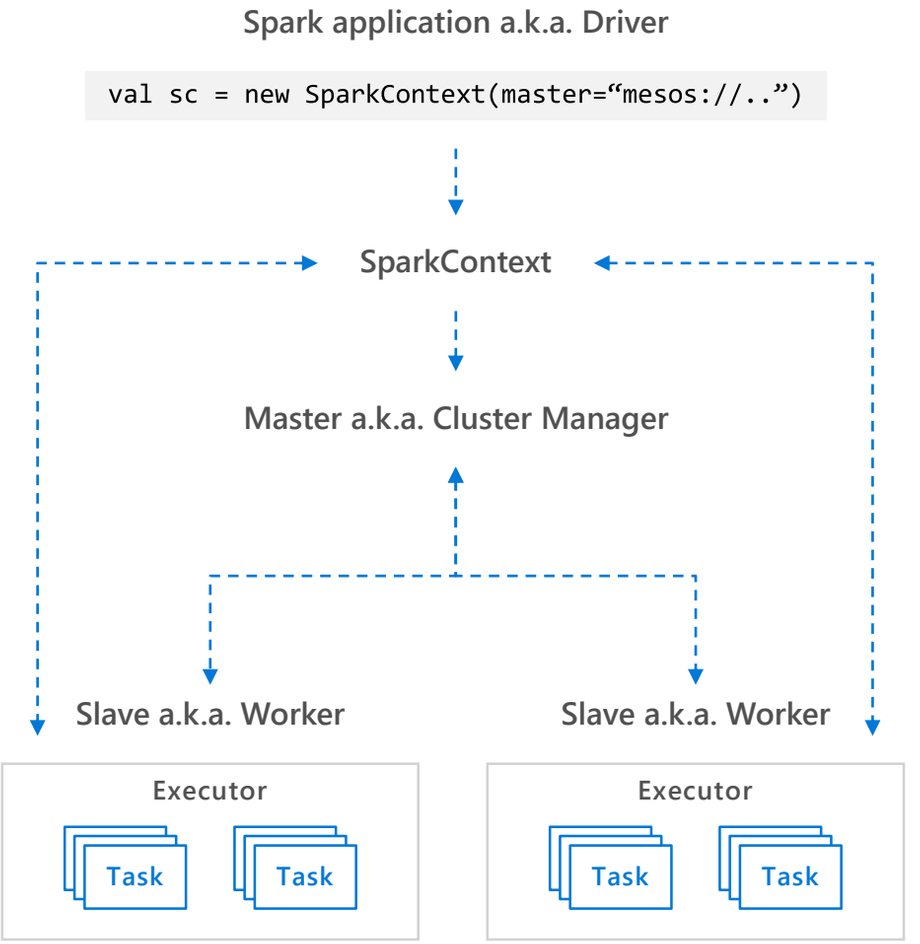
Scales to thousands of machines

Manages parallelism and resources using functional API

Possesses built-in Scala but has bindings in Python and R

Provides access to a flourishing community

Offers multiple libraries that include machine learning, streaming, and graph analysis



Integration of deep learning into Spark

- Create a library to easily develop learning applications on Spark
- Expand Microsoft's presence in the open source community
- Combine Spark with a flexible and hardware independent deep learning framework: Cognitive Toolkit (CNTK)
- Provide ways to easily run and operationalize models at a wide range of scales
- ML programming model improvements
  - Improved default featurization
  - Parallel model evaluation
  - Model comparison & summarization

Released @ Spark Summit 2017

Microsoft ML for Spark

Example

Pre-trained CNN to classify images in the CIFAR-10 dataset

```
...
import mmlspark as mml
# Initialize CNTKModel and define input and output columns
cntkModel = mml.CNTKModel().setInputCol("images").setOutputCol("output").setModelLocation(modelFile)
# Train on dataset with internal spark pipeline
scoredImages = cntkModel.transform(imagesWithLabels)
...
```

See other sample notebooks as well as the MMLSpark documentation for Scala and PySpark.

Get started with Microsoft Machine Learning for Spark

- To start with GitHub, access the GitHub repo:
  https://github.com/Azure/mmlspark

- To start with Docker, use Docker image:
  `docker run -it -p 8888:8888 -e ACCEPT_EULA=yes    microsoft/mmlspark`

- To view example Jupyter notebooks, navigate to:
  http://localhost:8888

MICROSOFT COGNITIVE TOOLKIT

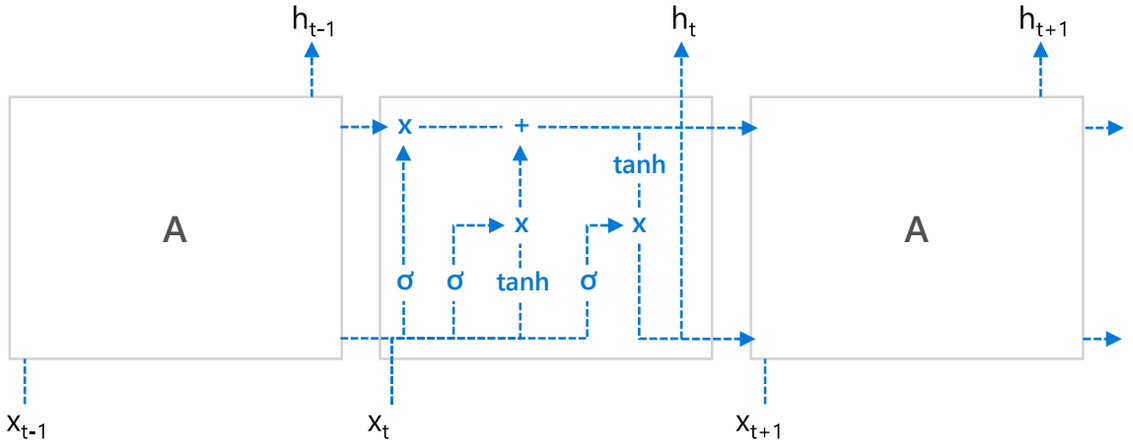
Cognitive Toolkit (CNTK)

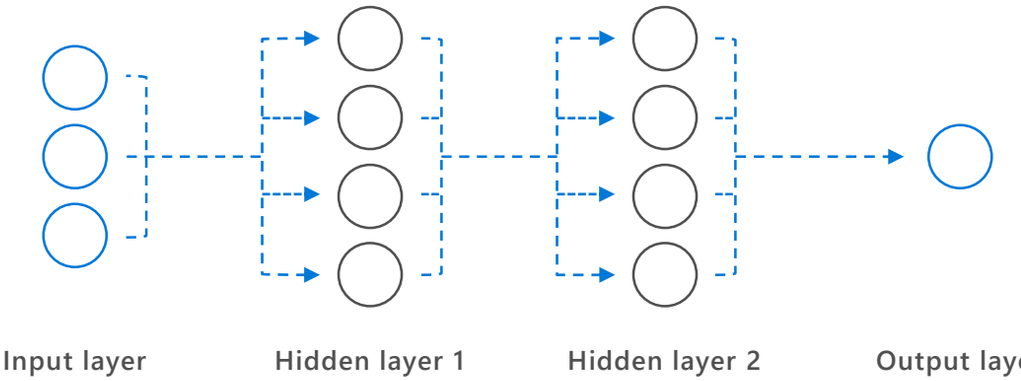
Microsoft's deep learning library

Supports and expresses a huge variety of neural networks and other deep architectures

LSTMS, ConvNets, reinforcement learning, Gans

Differentiates automatically and trains the net when users implement the forward direction of the network





CNTK x Spark

Step 1: CNTK Java Bindings

CNTK is written in C++ but has bindings in Python, Brainscript, and C#

Use the Simple Wrapper and Interface Generator (SWIG) to expose CNTK's Evaluation library to Java

Note: All Java bindings are machine generated so they require very little maintenance (available in CNTK's new release)

```
Function modelFunc = Function.load(new File("resnet20_cifar10_python.dnn"), device);
Variable outputVar = modelFunc.getOutputs().get(0);
Variable inputVar = modelFunc.getArguments().get(0);
```

CNTK x Spark

Step 2: Spark Transformers

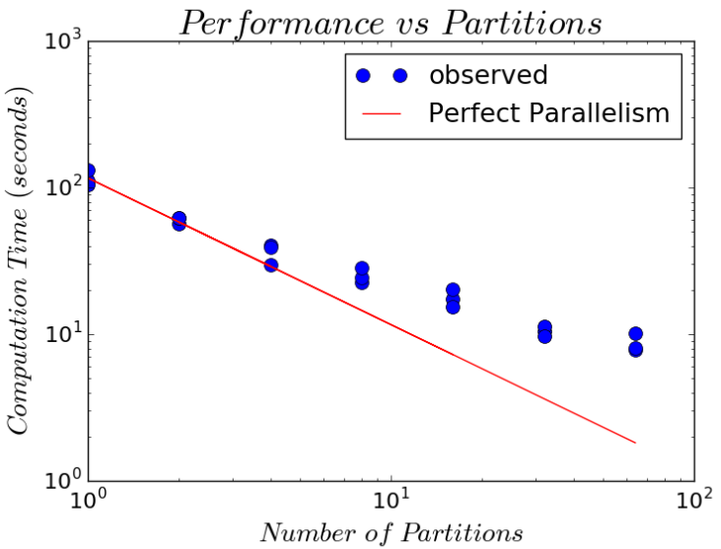
Use new CNTK Java Bindings since Spark is built on Scala (inter-op with Java)

Execute custom Scala maps on each of the nodes in the cluster

Distribute automatically, load, and run the CNTK model on all spark executors

Scale performance with machines as each machine maps to a small portion of the total dataset

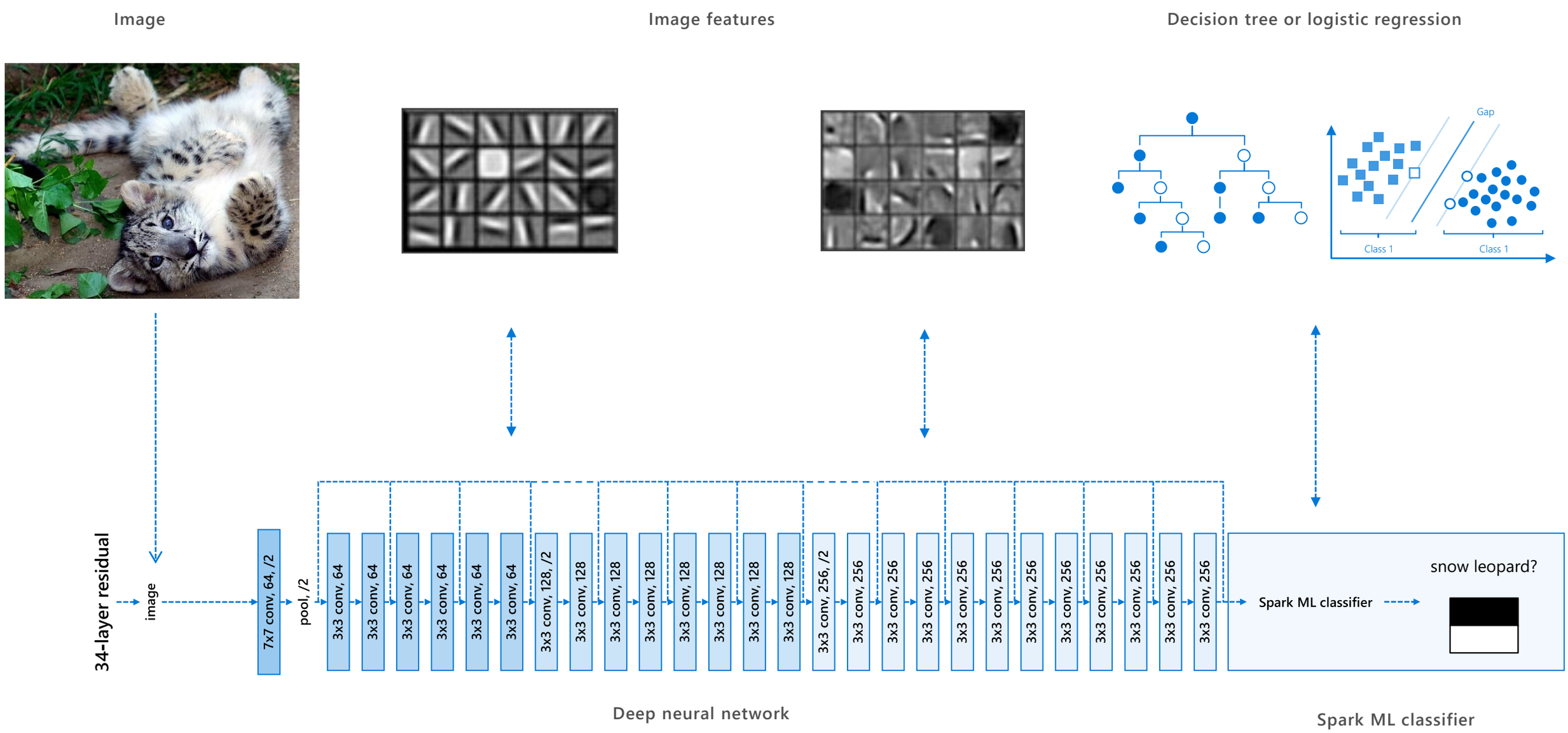
```
val model = new CNTKModel()
  .setModel(session, modelPath)
  .setInputCol(inputCol)
  .setOutputCol(outputCol)
  .setOutputNodeName("z")
```



Free PySpark bindings

- Scala is the core code; Python is the language of data science
- Spark has exposed bindings to Python in the PySpark package
- Automatically expose the CNTK Spark integration to python by instantly generating the Python bindings for all of your work
- Working with Databricks to contribute this tool to Spark's core and merge your work with SparkDL

Computer vision and classification on Spark

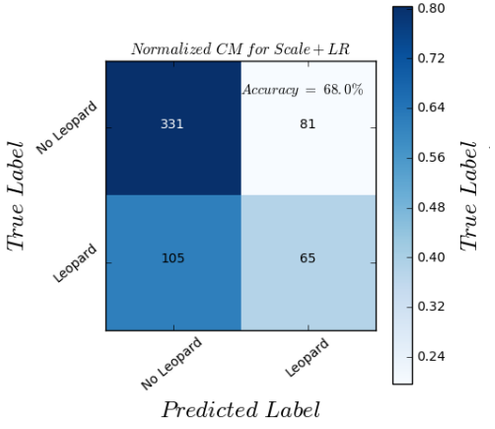


Deep featurization

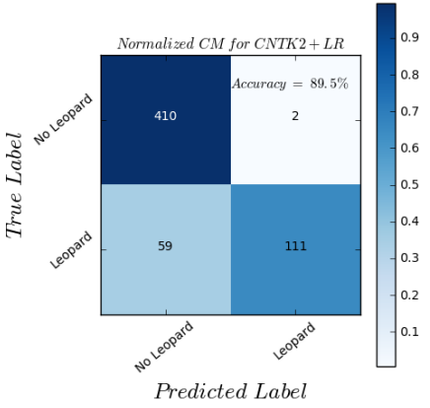
WITHOUT

WITH

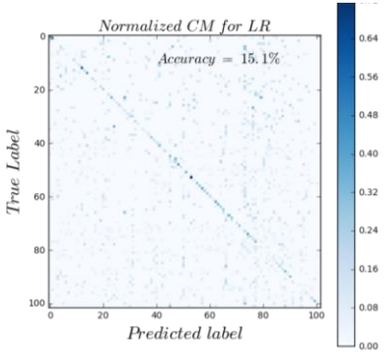
68% accuracy



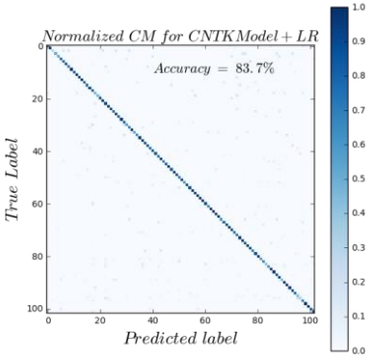
89% accuracy



15.1% accuracy



83.7% accuracy



AZURE MACHINE LEARNING OPERATIONALIZATION

Why operationalization?

- Models need to be made accessible
- Model deployment is challenging
- Need for web services as multi-platform integration points

Supported API patterns

BATCH OPERATIONS

- Intermittent or continuous
- Single node per job (for now)
- Mixed mode or batch only

REAL-TIME OPERATIONS

- High scale request-response pattern

Operationalization CLI

Environment and tech stack