Deploying AI for Near Real-Time Manufacturing Decisions
(Masterclass)

Pallavi Kar
Application Engineer – Data Science & Enterprise Integration
Digital Transformation and IIoT

Customer Goal

By connecting machines in operation, you can use data, algorithms, and models to make better decisions, improve processes, reduce cost, and improve customer experience.
Predictive Maintenance

- Operating Conditions vary over time and location
- Component Life and Safety
Transocean performed CPM of a BOP using an adaptive physics-based modeling approach with Simscape
Case Study: Transocean

**Objective:** Reduce BOP downtime

**Solution:**
- Simulink model of BOP and Control System
- Simulate 100s future scenarios - degradation trends and anomalies
- Pi Servers to collect data
- Preprocess data to avoid noise and outliers
- Train Models on future scenarios to predict in advance

**Outcome:** Robust condition and performance monitoring of BOP reduced downtime
Example Problem: Develop and operationalize a digital twin and a machine learning model to predict failures in industrial pumps

Current system requires Operator to manually monitor operational metrics for anomalies. Their expertise is required to detect and take preventative action.

**Process Engineer**
Develops models in MATLAB and Simulink

**System Architect**
Deploys and operationalizes model on Azure cloud

**Operator**
Makes operational decisions based on model output
Predictive Maintenance Workflow

1. Access and Explore Data
   - Files
   - Databases
   - Sensors

2. Preprocess Data
   - Working with Messy Data
   - Data Reduction/Transformation
   - Feature Extraction

3. Develop Predictive Models
   - Model Creation e.g. Machine Learning
   - Parameter Optimization
   - Model Validation

4. Integrate with Production Systems
   - Desktop Apps
   - Enterprise Scale Systems
   - Embedded Devices and Hardware

5. Visualize Results
   - 3rd party dashboards
   - Web apps
Backbone Infrastructure for Preventive, Predictive, Reactive, Actionable Analytics

- **Smart assets**
  - Data Ingestion
  - Local Communications

- **Edge systems**
  - Edge Management

- **OT Infrastructure**
  - Integration
  - Long-Range Communications

- **IT Systems**

---

**Value of data to decision making**

- **Model-Based Design with MATLAB & Simulink, code generation**
- **Edge Processing Model-Based Design, code generation**
- **Stream Processing with MATLAB Production Server**
- **Hadoop/Spark integration with MDCS, Compiler**

---

**Speed**

- Milliseconds
- Seconds
- Minutes
- Hours
- Days
- Months

**Scope**

- C/C++
- MODBUS TCP/IP
- MQTT
- docker
- kafka
- Kinesis
- Event Hub
- Azure
- Spark
- Hadoop
- Amazon Web Services
Steaming Analytics - Remaining Useful Life

Edge Device Publishing Data

Consume data and Update RUL
Project statement: Develop end-to-end predictive maintenance system

1. Monitor *flow, pressure*, and *current* of each pump so I always know their *operational state*

2. Need an *alert* when fault parameters drift outside an acceptable range so I can take *immediate corrective action*

3. Continuous estimate of each pump’s *remaining useful life (RUL)* so that I can *schedule maintenance or replace* the asset
We don’t have a large set of failure data, and it’s too costly to generate real failures in our plant for this project.

**Solution**: Use an accurate physics-based software model for the pump to develop synthetic training sets.
Need software for multidisciplinary problem across teams, plus integration w/ IT

**Solution**: Use MATLAB and integrate with OSS
Project constraints and solutions

We don’t have a large IT/hardware budget, and we need to see results before committing to a particular platform or technology.

**Solution:** Leverage cloud platform to quickly configure and provision the services needed to build the solution, while minimizing lock-in to a particular provider.
Physics of Triplex Pump

- Crankshaft drives three plungers
  - Each 120 degrees out of phase
  - One chamber always discharging
  - Three types of failures
Creating Multi-Domain Physical Models using Simscape

Monitor, Analyze, Predict, Control, Optimize

Pump Hardware
Acquire Real-Time Data for Updating Digital Twin

Monitor | Analyze | Predict | Control | Optimize

Pump Hardware

MODBUS TCPIP

Digital Twin

```matlab
m = modbus('tcpip', '192.168.2.1', 308)
m =

Modbus TCPIP with properties:

DeviceAddress: '192.168.2.1'
Port: 308
Status: 'open'
NumRetries: 1
Timeout: 10 (seconds)
ByteOrder: 'big-endian'
WordOrder: 'big-endian'
```
Use Simulink Design Optimizer to

- Monitor
- Analyze
- Predict
- Control
- Optimize

✓ Setup Experiments
✓ Parameterize
✓ Save Sessions
✓ Generate Code
Build digital twin and generate sensor data

Virtual Sensor Data

Diagnostics: On
- No Fault
- Blocked Inlet
- Seal Leak
- Worn Bearing
Simulate data with many failure conditions

Run parallel simulations

Access Data

```
ens = simulationEnsembleDatastore(location)

ens = simulationEnsembleDatastore with properties:

DataVariables: [25x1 string]
IndependentVariables: [0x0 string]
ConditionVariables: [0x0 string]
SelectedVariables: [25x1 string]
ReadSize: 1
NumMembers: 702
LastMemberRead: [0x0 string]
Files: [702x1 string]
```
Predictive Maintenance Workflow

1. Access and Explore Data
   - Files
   - Databases
   - Sensors

2. Preprocess Data
   - Working with Messy Data
   - Data Reduction/Transformation
   - Feature Extraction

3. Develop Predictive Models
   - Model Creation e.g. Machine Learning
   - Parameter Optimization
   - Model Validation

4. Integrate with Production Systems
   - Desktop Apps
   - Enterprise Scale Systems
   - Embedded Devices and Hardware

5. Visualize Results
   - 3rd party dashboards
   - Web apps
Represent signal information

Signal processing

```
[Spectrum, Frequencies] = pspectrum(data.Flow);
[pLow, pHigh] = bounds(Spectrum);
fPeak = Frequencies(Spectrum == pHigh);
qPeak2Peak = peak2peak(data.Flow);
qCrest = peak2rms(data.Flow);
qRMS = rms(data.Flow);
qMAD = mad(data.Flow);
```
Video showing App in action
Diagnostic Feature Designer App
Predictive Maintenance Toolbox R2019a

- Extract, visualize, and rank features from sensor data
- Use both statistical and dynamic modeling methods
- Work with out-of-memory data
- Explore and discover techniques without writing MATLAB code
Develop Predictive Models in MATLAB

Type of Fault (Classification)

Remaining Useful Life (Regression)

Process Engineer

Plant Operator
Develop Predictive Models in MATLAB

**Process Engineer**

<table>
<thead>
<tr>
<th>Time</th>
<th>1 LeakFault</th>
<th>2 BlockingFault</th>
<th>3 BearingFault</th>
<th>4 FaultType</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 sec</td>
<td>2.8472</td>
<td>-0.1477</td>
<td>1.8000</td>
<td>All</td>
</tr>
<tr>
<td>0.001 sec</td>
<td>-0.1498</td>
<td>-0.4207</td>
<td>1.3103</td>
<td>Bearing &amp; Blocking</td>
</tr>
<tr>
<td>0.002 sec</td>
<td>0.6511</td>
<td>1.6521</td>
<td>-0.5357</td>
<td>Leak</td>
</tr>
<tr>
<td>0.003 sec</td>
<td>0.1469</td>
<td>-0.2775</td>
<td>1.0074</td>
<td>All</td>
</tr>
<tr>
<td>0.004 sec</td>
<td>-0.6480</td>
<td>0.7065</td>
<td>-0.8876</td>
<td>Blocking</td>
</tr>
<tr>
<td>0.005 sec</td>
<td>-0.8165</td>
<td>-0.5434</td>
<td>-0.3079</td>
<td>Blocking</td>
</tr>
<tr>
<td>0.006 sec</td>
<td>-1.0061</td>
<td>1.2083</td>
<td>0.0661</td>
<td>Bearing</td>
</tr>
<tr>
<td>0.007 sec</td>
<td>1.0125</td>
<td>-1.9098</td>
<td>-0.7027</td>
<td>Leak &amp; Blocking</td>
</tr>
</tbody>
</table>

**Label Faults**

```
tt = tall(ds);
tt = preprocessData(tt);
model = TreeBagger(50,tt,'Event');
```

Evaluating tall expression using the Spark Cluster:
- Pass 1 of 2: Completed in 11 sec
- Pass 2 of 2: Completed in 2.3333 min
Evaluation completed in 2.6167 min
Process Engineer

Develop Predictive Models

Develop Machine Learning Models
Estimate Remaining Useful Life

\[ S(t) = \phi + \theta(t) e^{(\beta(t)t + \epsilon(t) - \frac{\sigma}{2})} \]
Develop a Stream Processing Function

**Batch Processing:** Build and test model on simulated data

**Stream Processing:** Apply model to sensor data in near real-time
Develop a Stream Processing Function

Streaming Function

function new_state = streamingFunction(data, old_state)

Preprocess signals
[data, features] = preprocessData(data);

Predict faults
[Leak, Blocking, Bearing] = predictFaultValues(features);
FaultType = predictFault(features);
[RUL, Model] = predictUpdateRUL(data.Timestamp, data.Flow, 500);

Update state
new_state = updateState(data, old_state);

Write results
writeResults(Leak, Blocking, Bearing, FaultType, RUL, Model)
end
Prototype Predictive Maintenance Architecture on Azure

**Edge**
- Generate telemetry

**Production System**
- MATLAB Production Server
  - Worker processes
- Request Broker
- Apache Kafka
- State Persistence
- System Architect
- Storage Layer

**Analytics Development**
- MATLAB Compiler SDK
- Model
- Package & Deploy
- Debug
- MATLAB

**Business Decisions**
- kibana
- Presentation Layer
- End Users
- Algorithm Developers

**Analytics Development**
- MATLAB
- Debug
- Package & Deploy
- Model
- MATLAB

**Business Decisions**
- kibana
- Presentation Layer
- End Users
- Algorithm Developers
What does a streaming function look like?

```matlab
function pumpconsume(msg)
    % Persistence service cache name must be unique to this group ID.
    msg.metaInfo.cacheName = make_cache_name(msg.metaInfo.groupId);

    % Application publishes results to a Kafka topic. The topic name
    % uses the group ID to allow multiple simultaneous Kafka consumers --
    % test and production, for example.
    %
    % consume and <consumerFunction> must agree on the name of the metaInfo
    % field used to store the topic name.
    msg.metaInfo.resultTopic = strcat('pump_results_', ...
        matlab.lang.makeValidName(msg.metaInfo.groupId));

    % The name of the connection to the persistence service. Configured
    % (set) by MATLAB Production Server admin.
    msg.metaInfo.connection = 'Azure_Redis';

    % Variables in the data table that we publish to Kafka.
    msg.metaInfo.outputVariables = ...
        struct('Flow',NaN,'Pressure',NaN,'Current',NaN);

    % Emit more log messages.
    msg.metaInfo.Verbose = true;

    % Finally, consume (process) the data in this input message.
    mps.stream.consume(msg);
end
```
Test Stream Processing Function

Process Engineer

```python
results = runtests('predictFaults_tests')
```

Running predictFaults_tests
....
Done predictFaults_tests

```plaintext
results =
1x4 TestResult array with properties:

Name  
Passed  
Failed  
Incomplete  
Duration  
Details  
Totals:  4 Passed, 0 Failed, 0 Incomplete. 0.01614 seconds testing time.
```
Test and Debug Streaming Function
Integrate with Production Systems

Package Stream Processing Function

Archive information
classifyData

Additional files required for your archive to run
helperFunction.m  PredictiveModels...

Files packaged for redistribution
classifyData.cff  readme.txt

Include MATLAB function signature file
Add or create a function signature file to help clients use your MATLAB functions.
Package and Test to generate compiled archive
Compiled Package and Runtime requirements

Process Engineer
Starting MATLAB Production Server Dashboard
Deploying Streaming Function on Production System

Integrate with Production Systems

MATLAB Production Server

MATLAB Analytics

MATLAB Compiler SDK

MATLAB
Integrate Analytics with Production Systems

Production System

Edge
- Generate telemetry

Analytics Development
- MATLAB Compiler SDK
- MATLAB
  - Debug
  - Package & Deploy
  - Model

Business Decisions
- Presentation Layer
  - kibana

System Architect

- System Architect
- Integrate with Production Systems

Worker processes
- Request Broker

Connector
- Azure
- Apache Kafka

State Persistence
- Storage Layer
- elastic
MATLAB Production Server on Azure

Production System

Azure

Management Server

MATLAB Production Server(s) scaling group

Virtual Network

https management endpoint

Connectors for Streaming/Event Data

Connectors for Storage & Databases

State Persistence

Application Gateway Load Balancer

Enterprise Applications

Integrate with Production Systems

System Architect
MathWorks Cloud Reference Architecture

MathWorks Cloud
MathWorks Cloud provides you with instant access to MATLAB and other products and services you are licensed for hosted on MathWorks managed cloud infrastructure. With MATLAB Online™, you can use MATLAB in a web browser without installing, configuring, or managing any software. MathWorks Cloud also provides MATLAB Drive™, giving you the ability to store, access, and work with your files from anywhere. You can access MathWorks Cloud solutions anywhere across different devices, use them to teach and learn, and to incorporate MATLAB analytics for a variety of applications.

Learn more about hosted offerings.

Public Clouds
MPS License and Instance Settings
Serving REST Calls on Production Server
MathWorks Reference Architectures

- **matlab-aws-s3**: MATLAB interface for AWS S3.
  - Updated 26 days ago

- **matlab-azure-blob**: MATLAB interface for Windows Azure Blob Storage.
  - Updated on Feb 21

- **matlab-parquet**: MATLAB interface for Apache Parquet.
  - Updated on Dec 20, 2018

- **matlab-azure-data-lake**: MATLAB Interface for Azure Data Lake.
  - Updated on Feb 21

- **matlab-aws-common**: Code common to MATLAB interfaces. Code in this repository is used as a dependency for other projects such as matlab-aws-s3.
  - Updated on Feb 21

- **matlab-avro**: MATLAB interface for Apache Avro files.
  - Updated on Feb 9

---

**Databases**

- Cassandra
- MongoDB
- SQL Server

**Cloud Storage**

- Azure Blob
- Azure SQL
- Amazon S3

**Big Data / OT Platforms**

- Cloudera
- Hortonworks
- OSIsoft, PI System

**Streaming**

- AWS Kinesis
- Azure IoT Hub
- Kafka

**OT Platforms**

- OSIsoft, PI System

**Dashboards**

- Microsoft Power BI
- Tableau
- Qlik
- Spotfire
Review System Requirements

- **Requirements from the Process Engineer**
  - Every millisecond, each pump generates a time-stamped record of flow, pressure, and current
  - Model expects 1 sec. window of data per pump
  - Initially, 1’s – 10’s of devices, but quickly scale to 100’s

- **Requirements from the Operator**
  - Alerts when parameters drift outside the expected ranges
  - Continuous estimating of RUL for each pump
Integrate Analytics with Production Systems

Production System

- Apache Kafka
- State Persistence
- Storage Layer
- Worker processes
- Request Broker
- MATLAB Production Server

Analytics Development

- MATLAB Compiler SDK
- Debug
- Package & Deploy
- Model

Business Decisions

- Presentation Layer
- kibana
Connecting MATLAB Production Server to Kafka

- Connector feeds single **Kafka topic** to a MATLAB function

- **Publisher library** for MATLAB for writing to a results stream

- Connector Features:
  - Deploy as a micro-service with Docker
  - Drive everything through config
  - Group data into time windows and pass to MATLAB as a timetable
  - Use Kafka’s check-pointing (i.e. at-least-once)
Setting up the Kafka Connector

```bash
#!/usr/bin/env bash
BASE=$(pwd)
echo "BASE: ${BASE}"

MPS_HOME=${BASE}
KAFKA_CONNECTOR_DIR=${MPS_HOME}/.../kafka-connector/Software/Java
MPS_CLIENT=${KAFKA_CONNECTOR_DIR}/client/java/mps_client.jar
KAFKA_CONNECTOR=${KAFKA_CONNECTOR_DIR}/lib/com/mathworks/mps/kafka-connector/1.1.0/kafka-connector-1.1.0-jar-with-dependencies.jar
CLASSPATH=${MPS_CLIENT}:${KAFKA_CONNECTOR}

# CLASSPATH=${CLASSPATH}:${MPS_HOME}/lib/com/mathworks/mps/kafka-connector/1.1.0/kafka-connector-jar-with-dependencies.jar

export GROUP_ID=mpsstuff
export CONNECTOR_TOPIC=to-mps
export CONNECTOR_TOPIC_OUT=from-mps
export MPS_CONNECT=http://localhost:9910
export MPS_ARCHIVE=PumpFault
export MPS_FUNCTION=streamingFunction
export BOOTSTRAP_SERVERS=localhost:9092
export MPS_DISPATCH_FUNCTION=pumpconsume
# Skip or stop
export MATLAB_ERROR_ACTION=skip

echo "CLASSPATH == ${CLASSPATH}"

exec java -cp ${CLASSPATH} \
-Dlog4j.configuration=file:${MPS_HOME}/config/log4j.properties \
com.mathworks.mps.client.kafka.KafkaConnector $@
```
Kafka connector architecture

- **Consumer Thread Pool**
  - **Topic**: \( P_0, P_1, \ldots, P_n \)
  - **Committed Offsets**: \( C_0, C_1, \ldots, C_n \)
  - **Message State (Offsets, Timestamps, Watermarks)**
    - **Active Windows**: \( W_n, \ldots, W_1, W_0 \)
    - **Async Request Handler**: \( P_0, \ldots, P_1, \ldots, P_n \)
      - **Async HTTP to Server**: \( r_0, r_1, \ldots \)

- **Kafka**
  - **Subscribe**: \( \text{subscribe} \)
  - **Poll**: \( \text{poll} \)
  - **Add**: \( \text{add} \)
  - **Commit**: \( \text{commit} \)

- **Production Server Java Client**
  - **invoke**: \( \text{invoke} \)
  - **Networking Threads**

- **Integrate with Production Systems**
Streaming data is treated as an unbounded Timetable

### Input Stream

<table>
<thead>
<tr>
<th>Event Time</th>
<th>Pump Id</th>
<th>Flow</th>
<th>Pressure</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>18:01:10</td>
<td>Pump1</td>
<td>1975</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>18:10:30</td>
<td>Pump3</td>
<td>2000</td>
<td>109</td>
<td>115</td>
</tr>
<tr>
<td>18:05:20</td>
<td>Pump1</td>
<td>1980</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>18:10:45</td>
<td>Pump2</td>
<td>2100</td>
<td>110</td>
<td>100</td>
</tr>
<tr>
<td>18:30:10</td>
<td>Pump4</td>
<td>2000</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>18:35:20</td>
<td>Pump4</td>
<td>1960</td>
<td>103</td>
<td>105</td>
</tr>
<tr>
<td>18:20:40</td>
<td>Pump3</td>
<td>1970</td>
<td>112</td>
<td>104</td>
</tr>
<tr>
<td>18:39:30</td>
<td>Pump4</td>
<td>2100</td>
<td>105</td>
<td>110</td>
</tr>
<tr>
<td>18:30:00</td>
<td>Pump3</td>
<td>1980</td>
<td>110</td>
<td>113</td>
</tr>
<tr>
<td>18:30:50</td>
<td>Pump3</td>
<td>2000</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Output Stream

<table>
<thead>
<tr>
<th>Time window</th>
<th>Pump Id</th>
<th>Bearing Friction</th>
</tr>
</thead>
<tbody>
<tr>
<td>18:00:00</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>18:10:00</td>
<td>Pump1</td>
<td>5</td>
</tr>
<tr>
<td>18:10:00</td>
<td>Pump3</td>
<td>...</td>
</tr>
<tr>
<td>18:10:00</td>
<td>Pump4</td>
<td>...</td>
</tr>
<tr>
<td>18:20:00</td>
<td>Pump2</td>
<td>7</td>
</tr>
<tr>
<td>18:20:00</td>
<td>Pump3</td>
<td>3</td>
</tr>
<tr>
<td>18:20:00</td>
<td>Pump4</td>
<td>...</td>
</tr>
<tr>
<td>18:30:00</td>
<td>Pump1</td>
<td>...</td>
</tr>
<tr>
<td>18:30:00</td>
<td>Pump3</td>
<td>4</td>
</tr>
<tr>
<td>18:30:00</td>
<td>Pump4</td>
<td>...</td>
</tr>
<tr>
<td>18:40:00</td>
<td>Pump5</td>
<td>...</td>
</tr>
<tr>
<td>18:40:00</td>
<td>Pump3</td>
<td>5</td>
</tr>
<tr>
<td>18:40:00</td>
<td>Pump4</td>
<td>8</td>
</tr>
</tbody>
</table>
Messaging adapter for Production Server

- Bridges streaming data and Production Server Async Java Client
- Batches incoming messages and sends them via HTTP request/response
  - Time windows, event time processing, and out-of-order data
- Uses Asynchronous pipeline model with back-pressure
  - Kafka consumers are automatically paused when server is busy
- Supports sequential (stateful) and unordered (stateless) processing
  - Provide unique stream ID/topic/partition info for persistence layer
- Pass data as MATLAB timetables
- Partition aware – enables full exploitation of partition-based parallelism
Creating persistence

Integrate with Production Systems
Attaching persistence

Integrate with Production Systems
Debug your streaming function on live data

**Production System**

- **MATLAB Production Server**
  - Worker processes
  - Broker

- **Connector**
- **Apache Kafka**
- **State Persistence**
- **Storage Layer**

**Analytics Development**

- **MATLAB Compiler SDK**
- **Model**
- **Package & Deploy**
- **Debug**
- **MATLAB**

**Business Decisions**

- **kibana**
- **Presentation Layer**
Debug a Stream Processing Function in MATLAB
Running Kafka with MPS
Complete your application

Integrate with Production Systems

System Architect

4

Edge

Generate telemetry

Production System

Production System

Analytics Development

Edge

Generate telemetry

Production System

Analytics Development

Business Decisions

Complete your application
Complete Your Application

Visualize Results

Plant Operator
Build Standalone UI based applications in MATLAB
MATLAB, Simulink and Cloud Reference Architectures provide “Integrated AI Development and Deployment Workflow” for Cross Functional Teams

- Successfully use Digital Twins to generate faults and train models
- Fast prototyping of physical and AI models with MATLAB
- Easy integration with OSS
- Cloud reference architectures for enabling faster IT setup
- Customize dashboard for Operator’s needs
Resources to learn and get started

- GitHub: MathWorks Reference Architectures
- Working with Enterprise IT Systems
- Data Analytics with MATLAB
- Simulink
MATLAB EXPO 2019

Thank You