



# Modern Industrial IoT Analytics on Azure

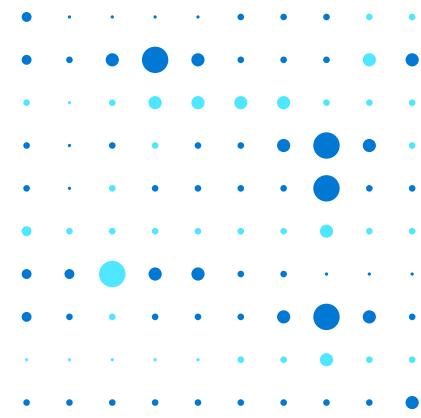
Customers Leverage Azure Databricks for Industrial IoT Analytics

February 2021



## Contents

Introduction .....	3
The Use Case — Wind Turbine Optimization .....	4
The Architecture — Ingest, Store, Prep, Train, Serve, Visualize .....	6
The Deployment .....	8
Data Ingest: Azure IoT Hub to Data Lake .....	8
Data Storage and Processing: Azure Databricks and Delta Lake .....	9
Machine Learning: Power Output and Remaining Life Optimization .....	12
Model Deployment and Hosting .....	18
Data Serving: Azure Data Explorer and Azure Synapse Analytics .....	20
Summary .....	24
What's Next? .....	25

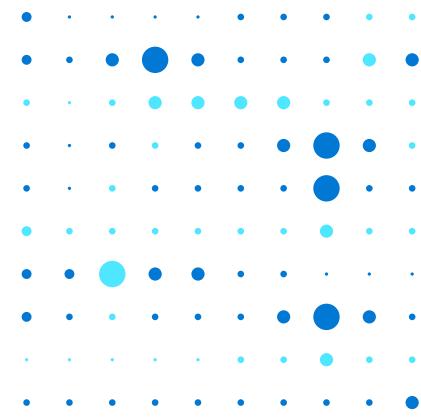


# Introduction

The Industrial Internet of Things (IIoT) has grown over the last few years from a grassroots technology stack being piloted predominantly in the oil and gas industry to wide-scale adoption and production use across the manufacturing, chemical, utilities, transportation and energy sectors. Traditional IoT systems like SCADA, Historians and even Hadoop do not provide the big data analytics capabilities needed by most organizations to predictively optimize their industrial assets due to the following factors.

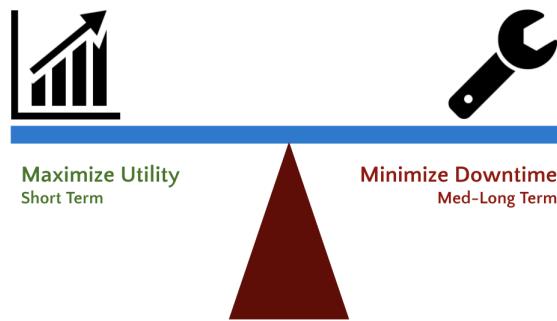
Challenge	Required Capability
Data volumes are significantly larger and more frequent	The ability to capture and store sub-second granular readings reliably and cost effectively from IoT devices streaming terabytes of data per day
Data processing needs are more complex	ACID-compliant data processing — time-based windows, aggregations, pivots, backfilling, shifting with the ability to easily reprocess old data
More user personas want access to the data	Data is an open format and easily shareable with operational engineers, data analysts, data engineers and data scientists without creating silos
Scalable ML is needed for decision-making	The ability to quickly and collaboratively train predictive models on granular, historic data to make intelligent asset optimization decisions
Cost-reduction demands are higher than ever	The ability to quickly and collaboratively train predictive models on granular, historic data to make intelligent asset optimization decisions

Organizations are turning to cloud platforms like Microsoft Azure to take advantage of the scalable, IIoT-enabling technologies they have to offer that make ingesting, processing, analyzing and serving time-series data sources like Historians and SCADA systems easy. This eBook discusses the end-to-end technology stack and the role Azure Databricks plays in the architecture for performing modern IIoT analytics.



## The Use Case – Wind Turbine Optimization

Most IIoT analytics projects are designed to maximize the short-term utilization of an industrial asset while minimizing its long-term maintenance costs. In this article, we focus on a hypothetical energy provider trying to optimize its wind turbines. The ultimate goal is to identify the set of optimal turbine operating parameters that maximizes each turbine's power output while minimizing its time to failure.

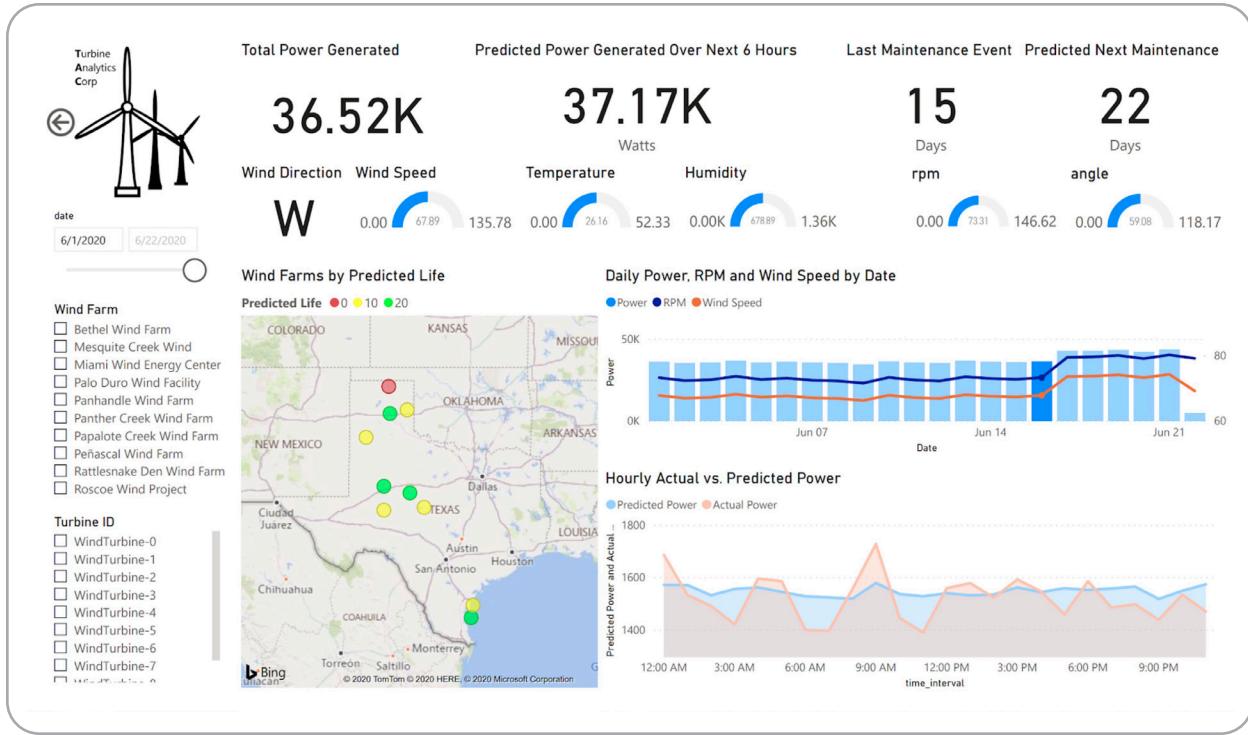


The goal of IIoT is to maximize utility in the short term while minimizing downtime over the long term.

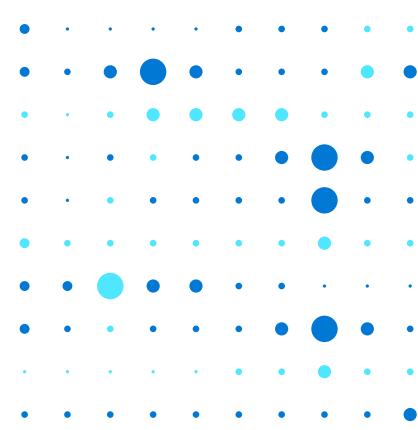
The final artifacts of this project are:

1. An automated data ingestion and processing pipeline that streams data to all end users
2. A predictive model that estimates the power output of each turbine given current weather and operating conditions
3. A predictive model that estimates the remaining life of each turbine given current weather and operating conditions
4. An optimization model that determines the optimal operating conditions to maximize power output and minimize maintenance costs thereby maximizing total profit
5. A real-time analytics dashboard for executives to visualize the current and future state of their wind farms, as shown on the next page:

## The Use Case – Wind Turbine Optimization

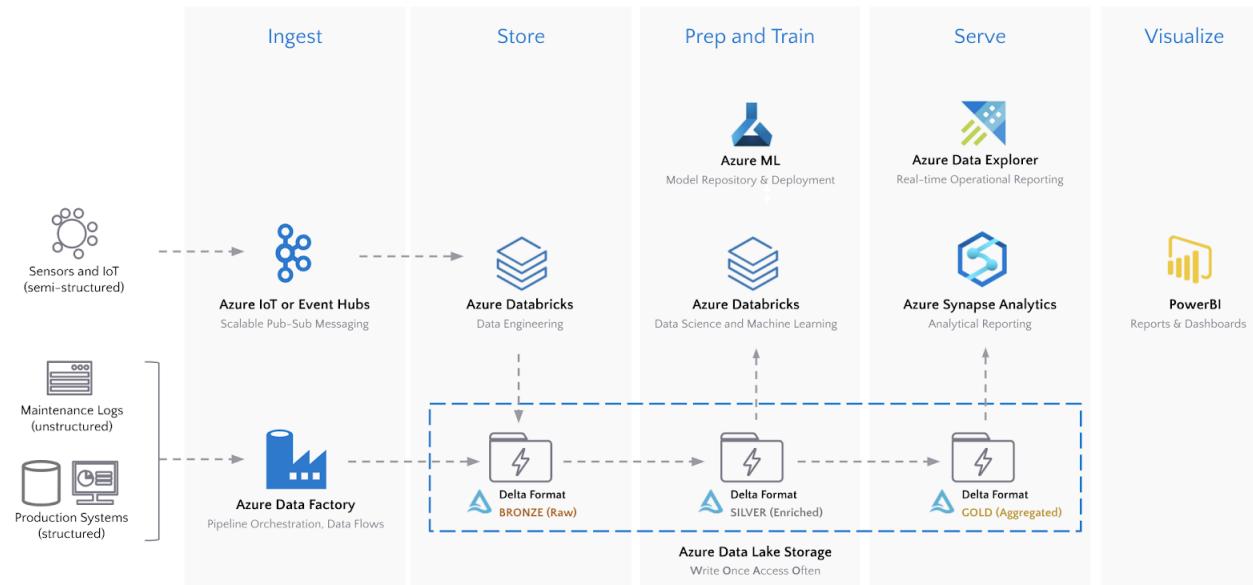


An IIoT analytics dashboard can help business executives visualize, for example, the current and future state of an industrial asset, such as a wind farm.



## The Architecture — Ingest, Store, Prep, Train, Serve, Visualize

The architecture below illustrates a modern, best-of-breed platform used by many organizations that leverages all that Azure has to offer for IIoT analytics.

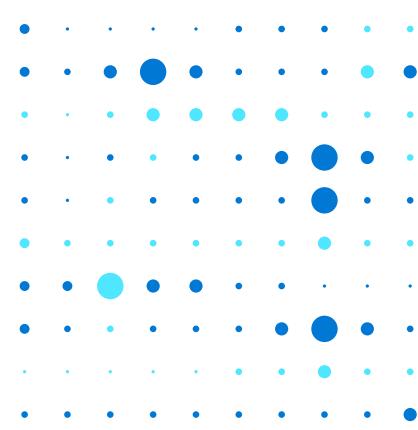


The IIoT data analytic architecture featuring the Azure Data Lake Storage and Delta Lake format offers data teams the optimal platform for handling time-series streaming data.

## The Architecture – Ingest, Store, Prep, Train, Serve, Visualize

A key component of this architecture is the Azure Data Lake Storage (ADLS), which enables the write-once, access-often analytics pattern in Azure. However, Data Lakes alone do not solve challenges that come with time-series streaming data. The Delta storage format provides a layer of resiliency and performance on all data sources stored in ADLS. Specifically for time-series data, Delta provides the following advantages over other storage formats on ADLS:

Required Capability	Other Formats on ADLS Gen 2	Delta Format on ADLS Gen 2
Unified batch and streaming	Data Lakes are often used in conjunction with a streaming store like CosmosDB, resulting in a complex architecture	ACID-compliant transactions enable data engineers to perform streaming ingest and historically batch loads into the same locations on ADLS
Schema enforcement and evolution	Data Lakes do not enforce schema, requiring all data to be pushed into a relational database for reliability	Schema is enforced by default. As new IoT devices are added to the data stream, schemas can be evolved safely so downstream applications don't fail.
Efficient upserts	Data Lakes do not support in-line updates and merges, requiring deletion and insertions of entire partitions to perform updates	MERGE commands are effective for situations handling delayed IoT readings, modified dimension tables used for real-time enrichment, or if data needs to be reprocessed
File compaction	Streaming time-series data into Data Lakes generates hundreds or even thousands of tiny files	Auto-compaction in Delta optimizes the file sizes to increase throughput and parallelism
Multi-dimensional clustering	Data Lakes provide push-down filtering on partitions only	Z-ORDERing time-series on fields like timestamp or sensor ID allows Databricks to filter and join on those columns up to 100x faster than simple partitioning techniques



# The Deployment

We use Azure's Raspberry PI IoT Simulator to simulate real-time machine-to-machine sensor readings and send them to Azure IoT Hub.

## Data Ingest: Azure IoT Hub to Data Lake

Our deployment has sensor readings for weather (wind speed and direction, temperature, humidity) and wind turbine telematics (angle and RPM) sent to an IoT cloud computing hub. Azure Databricks can natively stream data from IoT Hubs directly into a Delta table on ADLS and display the input vs. processing rates of the data.

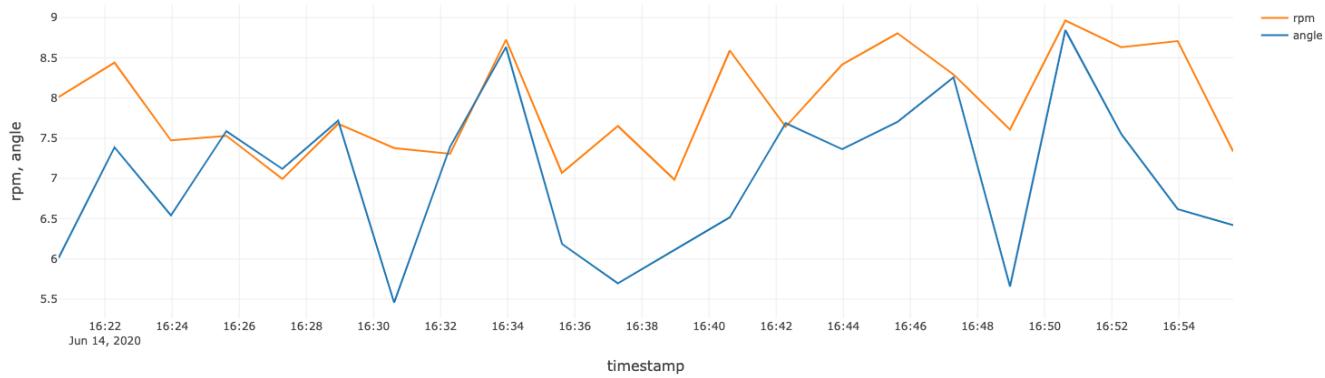
```
# Read directly from IoT Hubs using the EventHubs library for Azure Databricks
iot_stream = (
    spark.readStream.format("eventhubs") # Read from IoT Hubs directly
    .options(**ehConf)                 # Use the Event-Hub-enabled connect string
    .load()                           # Load the data
    .withColumn('reading', F.from_json(F.col('body').cast('string'), schema)) # Extract the payload
    .select('reading.*', F.to_date('reading.timestamp').alias('date'))       # Create a "date" field
)

# Split our IoT Hubs stream into separate streams and write them both into their own Delta locations
write_turbine_to_delta = (
    iot_stream.filter('temperature is null')                                # Filter out turbine telemetry data
    .select('date','timestamp','deviceId','rpm','angle') # Extract the fields of interest
    .writeStream.format('delta') # Write our stream to the Delta format
    .partitionBy('date')        # Partition our data by Date for performance
    .option("checkpointLocation", ROOT_PATH + "/bronze/cp/turbine") # Checkpoint
    .start(ROOT_PATH + "/bronze/data/turbine_raw")                  # Stream the data into an ADLS
Path
)
```

## The Deployment

Delta allows our IoT data to be queried within seconds of it being captured in IoT Hub.

```
%sql  
-- We can query the data directly from storage immediately as it streams into Delta  
SELECT * FROM delta.`/tmp/iiot/bronze/data/turbine_raw` WHERE deviceid = 'WindTurbine-1'
```



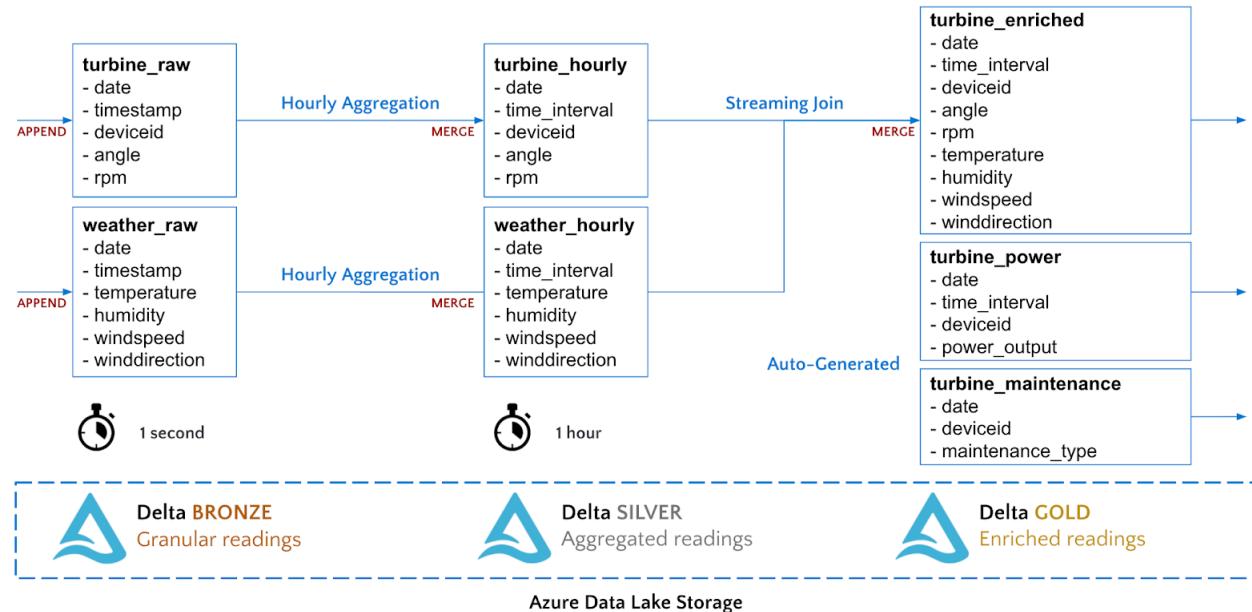
With the Delta storage format, IIoT data can be queried within seconds of capture, for use with downstream analytics, such as a time-series data visualization.

We can now build a downstream pipeline that enriches and aggregates our IIoT applications data for data analytics.

## Data Storage and Processing: Azure Databricks and Delta Lake

Delta supports a multi-hop pipeline approach to data engineering, where data quality and aggregation increases as it streams through the pipeline. Our time-series data will flow through the following Bronze, Silver and Gold data levels.

## The Deployment



Delta Lake supports a multi-hop pipeline approach to data engineering, where data quality and aggregation increases as it streams through the pipeline.

Our pipeline from Bronze to Silver will simply aggregate our turbine sensor data to one-hour intervals. We will perform a streaming MERGE command to upsert the aggregated records into our Silver Delta tables.

```
# Create functions to merge turbine and weather data into their target Delta tables
def merge_records(incremental, target_path):
    incremental.createOrReplaceTempView("incremental")

    # MERGE consists of a target table, a source table (incremental),
    # a join key to identify matches (deviceid, time_interval), and operations to perform
    # (UPDATE, INSERT, DELETE) when a match occurs or not
    incremental._jdf.sparkSession().sql("""
        MERGE INTO turbine_hourly t
        USING incremental i
        ON i.date=t.date AND i.deviceId = t.deviceid AND i.time_interval = t.time_interval
        WHEN MATCHED THEN UPDATE SET *
        WHEN NOT MATCHED THEN INSERT *
    """)

# Perform the streaming merge into our data stream
turbine_stream = (
    spark.readStream.format('delta').table('turbine_raw') # Read data as a stream from our Bronze Delta
    .groupBy('deviceId','date',F.window('timestamp','1 hour')) # Aggregate readings to hourly intervals
    .agg({"rpm":"avg","angle":"avg"})
    .writeStream
    .foreachBatch(merge_records)                                # Pass each micro-batch to a function
    .outputMode("update")                                     # Merge works with update mod
    .start()
)
```

## The Deployment

Our pipeline from Silver to Gold will join the two streams together into a single table for hourly weather and turbine measurements.

```
# Read streams from Delta Silver tables
turbine_hourly = spark.readStream.format('delta').option("ignoreChanges", True).table("turbine_hourly")
weather_hourly = spark.readStream.format('delta').option("ignoreChanges", True).table("weather_hourly")

# Perform a streaming join to enrich the data
turbine_enriched = turbine_hourly.join(weather_hourly, ['date', 'time_interval'])

# Perform a streaming merge into our Gold data stream
merge_gold_stream = (
    turbine_enriched.writeStream
    .foreachBatch(merge_records)
    .start()
)
```

We can query our Gold Delta table immediately.

The screenshot shows a Jupyter Notebook cell titled 'Cmd\_21'. The cell contains a SQL query: '%sql SELECT \* FROM turbine\_enriched WHERE deviceid='WindTurbine-1''. Below the query, it shows '(3) Spark Jobs' and a table with two rows of data. The table has columns: deviceid, date, time\_interval, angle, rpm, temperature, humidity, windspeed, and winddirection. The data is as follows:

deviceid	date	time_interval	angle	rpm	temperature	humidity	windspeed	winddirection
WindTurbine-1	2020-06-14	2020-06-14T16:00:00.000+0000	7.1520467	7.983291	26.130173	70.060844	7.006084	W
WindTurbine-1	2020-06-14	2020-06-14T17:00:00.000+0000	7.3464413	8.395933	26.089893	70.747955	7.0747952	NW

With Delta Lake, you can query your enriched, AI-ready data immediately, for use with IIoT data science predictive models to optimize asset utilization.

The notebook also contains a cell that will generate historical hourly power readings and daily maintenance logs that will be used for model training. Running that cell will:

1. Backfill historical readings for one year in the `turbine_enriched` table
2. Generate historical power readings for each turbine in the `power_output` table
3. Generate historical maintenance logs for each turbine in the `turbine_maintenance` table

We now have enriched, artificial intelligence (AI)-ready data in a performant, reliable format on Azure Data Lake that can be fed into our data science modeling to optimize asset utilization.

## The Deployment

```
%sql
-- Query all 3 tables together
CREATE OR REPLACE VIEW gold_readings AS
SELECT r.*,
       p.power,
       m.maintenance AS maintenance
FROM turbine_enriched r
JOIN turbine_power p ON (r.date=p.date AND r.time_interval=p.time_interval AND r.deviceid=p.deviceid)
LEFT JOIN turbine_maintenance m ON (r.date=m.date AND r.deviceid=m.deviceid);

SELECT * FROM gold_readings
```

date	time_interval	deviceid	rpm	angle	temperature	humidity	windspeed	winddirection	power	maintenance
2020-06-12	2020-06-12T05:00:00.000+0000	WindTurbine-8	6.66773192141555	5.374035574520114	23.79682872841723	61.74833192837229	6.174833192837228	E	124.62010486492946	true
2020-06-12	2020-06-12T06:00:00.000+0000	WindTurbine-8	6.368534099393156	5.132889145951731	22.729005454913175	58.97752970570339	5.897752970570339	N	113.68700051216648	true
2020-06-12	2020-06-12T07:00:00.000+0000	WindTurbine-8	6.27760260488803	5.059600493667568	22.404475130921323	58.13543404685028	5.813543404685028	E	110.46367583934948	true
2020-06-12	2020-06-12T08:00:00.000+0000	WindTurbine-8	9.12659422889947	7.355820935541138	32.572395275892696	84.51929028661552	8.451929028661551	SW	233.47981228981843	true
2020-06-12	2020-06-12T09:00:00.000+0000	WindTurbine-8	7.436996407447541	5.994044711467739	26.54229831777958	68.87231342346766	6.887231342346765	SW	155.0340151201512	true
2020-06-12	2020-06-12T10:00:00.000+0000	WindTurbine-8	7.4902038161716185	6.03692863521874	26.73219311370233	69.3650549994116	6.93650549994116	S	157.2603081067385	true
2020-06-12	2020-06-12T11:00:00.000+0000	WindTurbine-8	7.19824097875406	5.801613047030012	25.690189451537517	66.66124985225359	6.66124985225359	SE	145.23943691974017	true
2020-06-12	2020-06-12T12:00:00.000+0000	WindTurbine-8	6.7642942185052775	5.451862521627735	24.14145512790678	62.642573154790355	6.2642573154790355	NE	128.2557456603227	true
2020-06-12	2020-06-12T13:00:00.000+0000	WindTurbine-8	6.210520028020707	5.512200280207074	24.14145512790678	62.642573154790355	6.2642573154790355	E	121.42000762626209	true

Showing the first 1000 rows.

Our data engineering pipeline is complete! Data is now flowing from IoT Hubs to Bronze (raw) to Silver (aggregated) to Gold (enriched). It is time to perform some analytics on our data.

## Machine Learning: Power Output and Remaining Life Optimization

Optimizing the utility, lifetime and operational efficiency of industrial assets like wind turbines has numerous revenue and cost benefits. The real-world challenge we explore in this article is maximizing the revenue of a wind turbine while minimizing the opportunity cost of downtime, thereby maximizing our net profit.

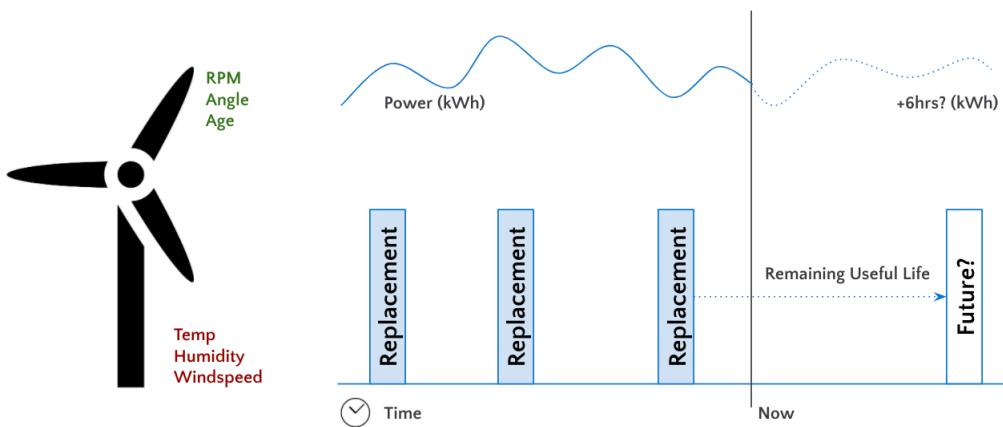
$$\text{Net profit} = \text{Power generation revenue} - \text{Cost of added strain on equipment}$$

If we push a turbine to a higher RPM, it will generate more energy and therefore more revenue. However, the added strain on the turbine will cause it to fail more often, introducing cost.

To solve this optimization problem, we will create two models:

1. Predict the power generated by a turbine given a set of operating conditions
2. Predict the remaining life of a turbine given a set of operating conditions

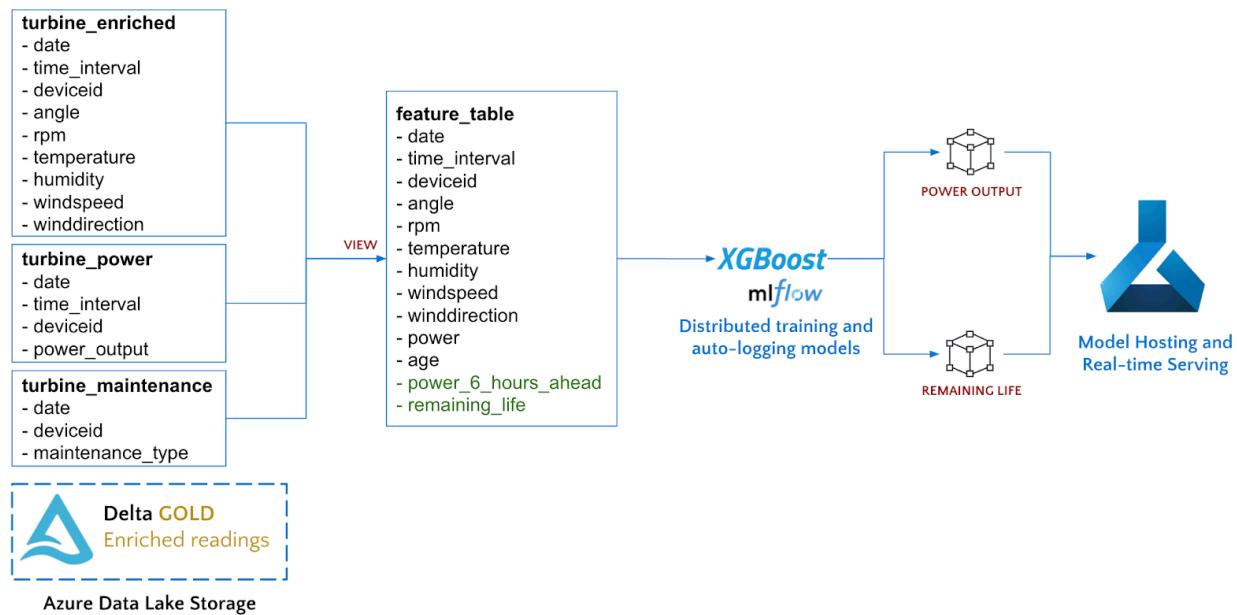
## The Deployment



Using Azure Databricks IIoT data analytics to predict the remaining life of a wind turbine.

We can then produce a profit curve to identify the optimal operating conditions that maximize power revenue while minimizing costs.

Using Azure Databricks with our Gold Delta tables, we will perform feature engineering to extract the fields of interest, train the two models, and finally deploy the models to Azure Machine Learning for hosting.



The Azure Databricks machine learning model lifecycle for an IIoT data analytics use case.

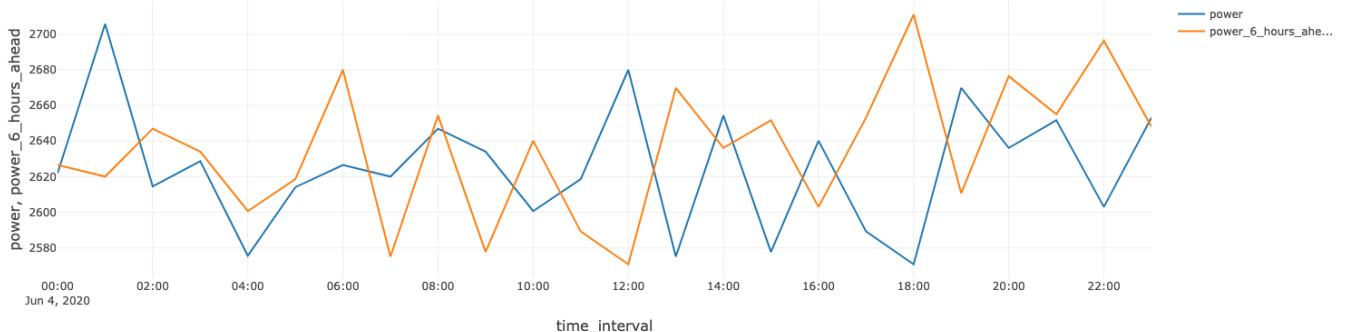
## The Deployment

To calculate the remaining useful lifetime of each wind turbine, we can use our maintenance records that indicate when each asset is replaced.

```
%sql
-- Calculate the age of each turbine and the remaining life in days
CREATE OR REPLACE VIEW turbine_age AS
WITH reading_dates AS (SELECT distinct date, deviceid FROM turbine_power),
maintenance_dates AS (
    SELECT d.* , datediff(nm.date, d.date) as datediff_next, datediff(d.date, lm.date) as datediff_last
    FROM reading_dates d LEFT JOIN turbine_maintenance nm ON (d.deviceid=nm.deviceid AND
d.date<=nm.date)
    LEFT JOIN turbine_maintenance lm ON (d.deviceid=lm.deviceid AND d.date>=lm.date ))
SELECT date, deviceid, min(datediff_last) AS age, min(datediff_next) AS remaining_life
FROM maintenance_dates
GROUP BY deviceid, date;
```

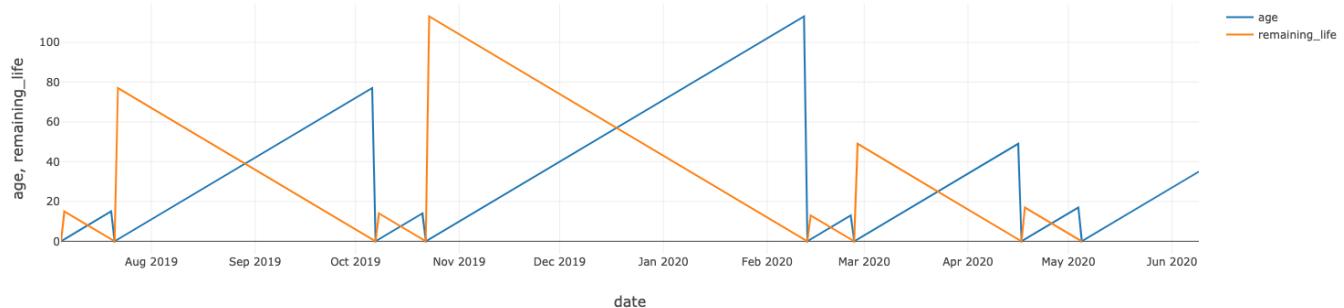
To predict power output at a six-hour time horizon, we calculate time series shifts using Spark window functions.

```
CREATE OR REPLACE VIEW feature_table AS
SELECT r.* , age, remaining_life,
-- Calculate the power 6 hours ahead using Spark Windowing and build a feature_table to feed into
our ML models
LEAD(power, 6, power) OVER (PARTITION BY r.deviceid ORDER BY time_interval) as power_6_hours_ahead
FROM gold_readings r
JOIN turbine_age a ON (r.date=a.date AND r.deviceid=a.deviceid)
WHERE r.date < CURRENT_DATE(); -- Only train on historical data
```



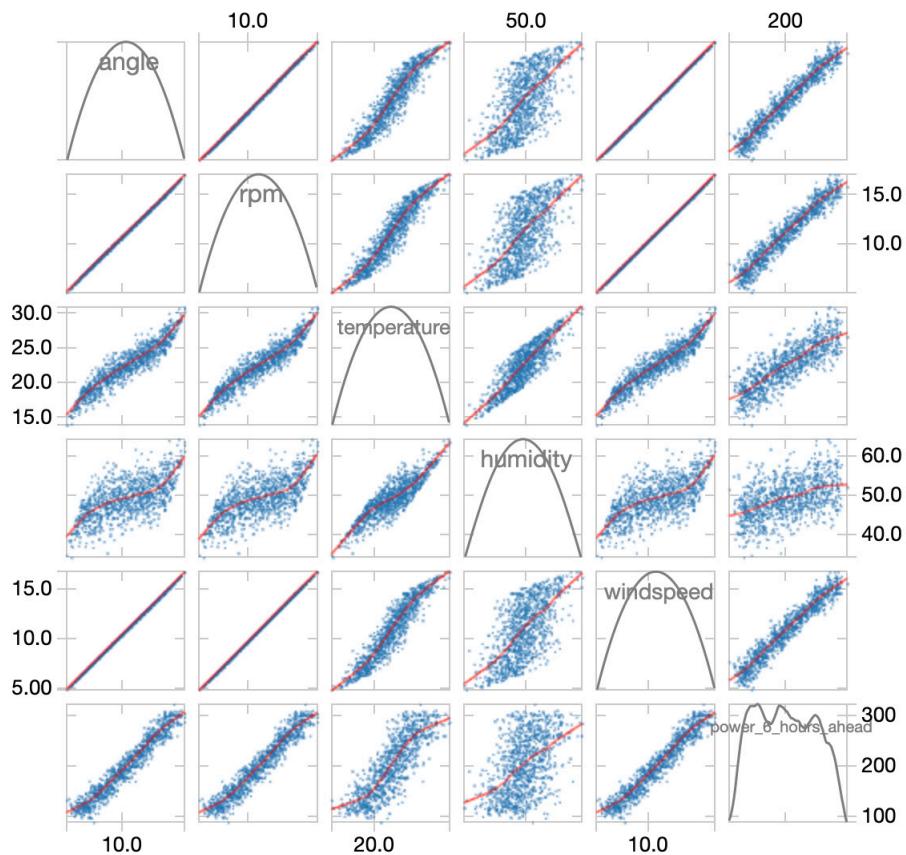
With Azure Databricks, you can calculate time-series shifts using Spark window functions to predict, for example, the power output of a wind farm at a six-hour time horizon.

## The Deployment



With Azure Databricks' IIoT data analytics, you can calculate, for example, the remaining useful lifetime of a wind turbine, using maintenance records that indicate when each asset has been replaced.

There are strong correlations between both Turbine operating parameters (RPM and Angle) as well as weather conditions and the power generated six hours from now.



With Azure Databricks' IIoT data analytics, you can uncover, for example, the strong correlations between both turbine operating parameters (RPM and Angle) as well as weather conditions and future power generated.

## The Deployment

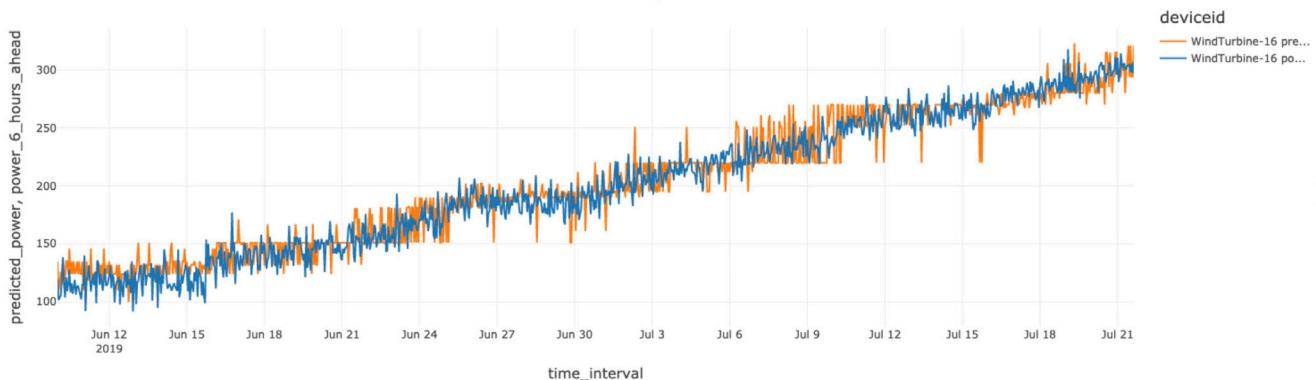
We can now train an XGBoost Regressor model to use our feature columns (weather, sensor and power readings) to predict our label (power reading six hours ahead). We can train a model for each wind turbine in parallel using a Pandas UDF, which distributes our XGBoost model training code to all the available nodes in the Azure Databricks cluster.

```
# Create a Spark Dataframe that contains the features and labels we need
feature_cols = ['angle','rpm','temperature','humidity','windspeed','power','age']
label_col = 'power_6_hours_ahead'

# Read in our feature table and select the columns of interest
feature_df = spark.table('feature_table')

# Create a Pandas UDF to train a XGBoost Regressor on each turbine's data
@pandas_udf(feature_df.schema, PandasUDFType.GROUPED_MAP)
def train_power_model(readings_pd):
    mlflow.xgboost.autolog() # Auto-Log the XGB parameters, metrics, model and artifacts
    with mlflow.start_run():
        # Train an XGBRegressor on the data for this Turbine
        alg = xgb.XGBRegressor()
        train_dmatrix = xgb.DMatrix(data=readings_pd[feature_cols].astype('float'),label=readings_pd[label_col])
        model = xgb.train(dtrain=train_dmatrix, evals=[(train_dmatrix, 'train')])
    return readings_pd

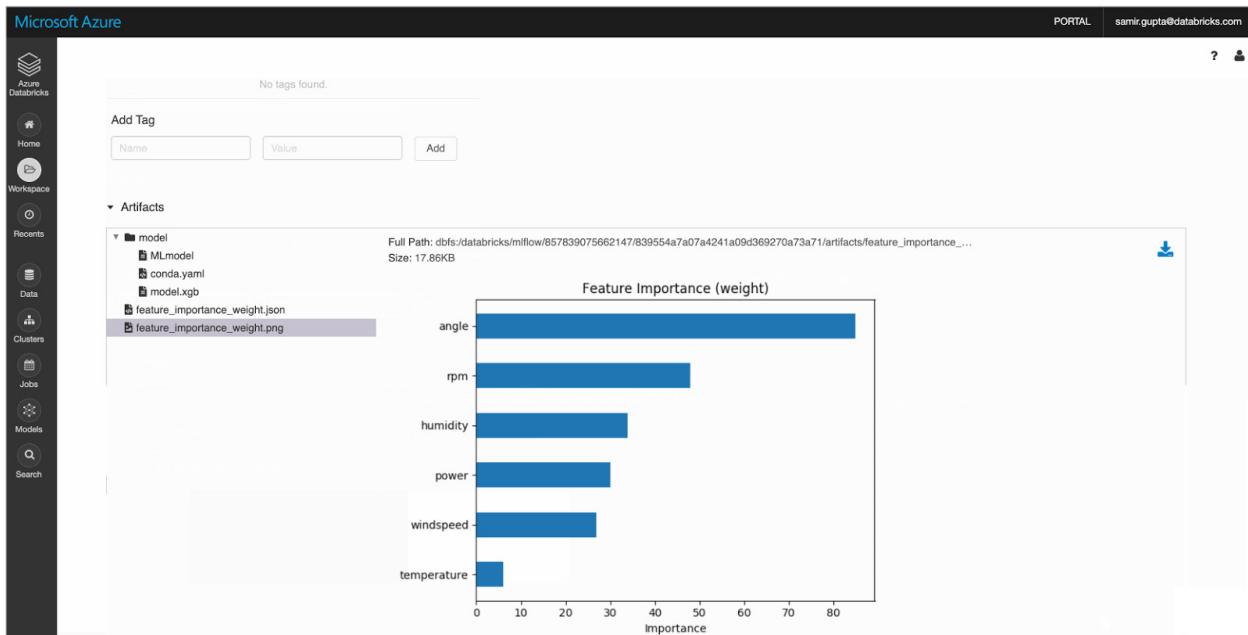
# Run the Pandas UDF against our feature dataset
power_predictions = feature_df.groupBy('deviceid').apply(train_power_model)
```



With Azure Databricks' IIoT data analytics, you can predict, for example, the power output of a specific wind turbine and display the results in a time-series visualization.

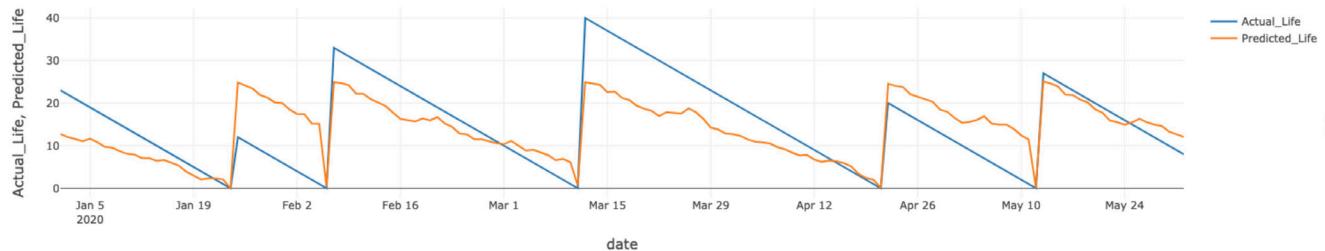
Azure Databricks will automatically track each model training run with a hosted MLflow experiment. For XGBoost Regression, MLflow will track any parameters passed into the params argument, the RMSE metric, the turbine this model was trained on, and the resulting model itself. For example, the RMSE for predicting power on deviceid WindTurbine-18 is 45.79.

## The Deployment



With Azure Databricks' IIoT data analytics, you can automatically track each model training run with a hosted MLflow experiment. For XGBoost Regression, MLflow will track any parameters passed into the params argument, to predict, for example, the power output of a wind turbine by specific deviceid.

We can train a similar model for the remaining life of the wind turbine. The actual vs. predicted for one of the turbines is shown below.



With Azure Databricks' IIoT data analytics, you can predict, for example, the remaining lifespan of a wind turbine and generate a time-series visualization comparing the prediction against the actuality.

## The Deployment

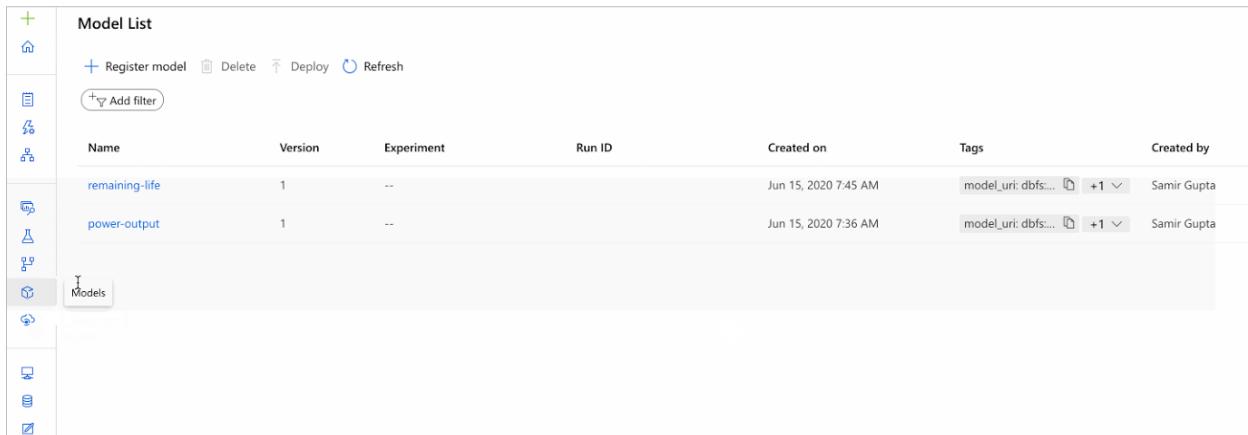
### Model Deployment and Hosting

Azure Databricks is integrated with Azure Machine Learning for model deployment and scoring. Using the Azure ML APIs directly inside of Databricks, we can automatically deploy an image for each model to be hosted in a fast, scalable container service (ACI or AKS) by Azure ML.

```
# Create a model image inside of AzureML
model_image, azure_model = mlflow.azureml.build_image(model_uri=path,
                                                       workspace=workspace,
                                                       model_name=model,
                                                       image_name=model,
                                                       description="XGBoost model to predict power output"
                                                       synchronous=False)

# Deploy a web service to host the model as a REST API
dev_webservice_deployment_config = AciWebservice.deploy_configuration()
dev_webservice = Webservice.deploy_from_image(name=dev_webservice_name,
                                              image=model_image,
                                              workspace=workspace)
```

Once the model is deployed, it will show up inside the Azure ML studio, and we can make REST API calls to score data interactively.



The screenshot shows the Azure ML studio interface with the 'Models' section selected. The 'Model List' page displays two registered models:

Name	Version	Experiment	Run ID	Created on	Tags	Created by
remaining-life	1	--		Jun 15, 2020 7:45 AM	model_uri: dbfs... +1	Samir Gupta
power-output	1	--		Jun 15, 2020 7:36 AM	model_uri: dbfs... +1	Samir Gupta

Azure Databricks is integrated with Azure Machine Learning for model deployment and scoring. Once the model is deployed, it will show up inside the Azure ML studio, and you can make REST API calls to score data interactively.

## The Deployment

```
# Construct a payload to send with the request
payload = {
    'angle':12,
    'rpm':10,
    'temperature':25,
    'humidity':50,
    'windspeed':10,
    'power':200,
    'age':10
}

def score_data(uri, payload):
    rest_payload = json.dumps({"data": [list(payload.values())]})
    response = requests.post(uri, data=rest_payload, headers={"Content-Type": "application/json"})
    return json.loads(response.text)

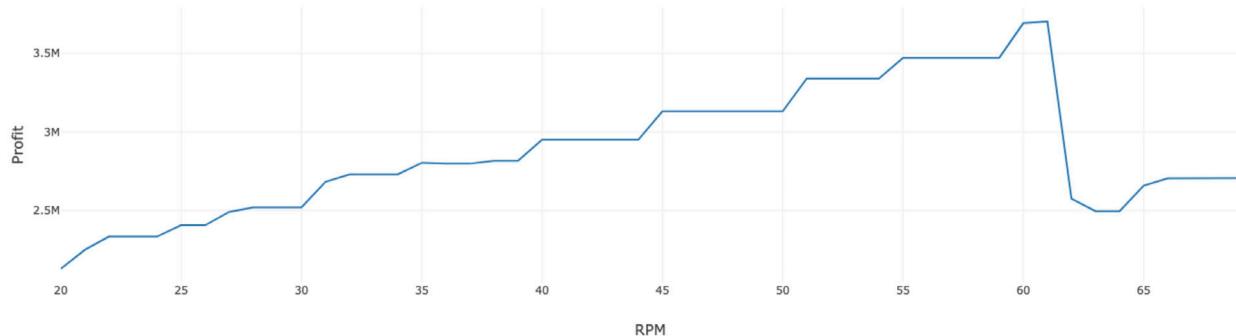
print(f'Predicted power (in kwh) from model: {score_data(power_uri, payload)}')
print(f'Predicted remaining life (in days) from model: {score_data(life_uri, payload)})'
```

Now that both the power optimization and the RUL models are deployed as prediction services, we can utilize both in optimizing net profit from each wind turbine.

Assuming \$1 per kWh, annual revenue can simply be calculated by multiplying the expected hourly power by 24 hours and 365 days.

The annual cost can be calculated by multiplying the daily revenue by the number of times the turbine needs to be maintained in a year (365 days / remaining life).

We can iteratively score various operating parameters simply by making multiple calls to our models hosted in Azure ML. By visualizing the expected profit cost for various operating parameters, we can identify the optimal RPM to maximize profit.



With Azure Databricks' IIoT data analytics, you can iteratively score various operating parameters by calling the models hosted in Azure ML. The resulting visual analytic, for example, of the expected profit cost for various operating parameters can help to identify the optimal RPM to maximize profit.

## The Deployment

# Data Serving: Azure Data Explorer and Azure Synapse Analytics

### Operational Reporting in ADX

Azure Data Explorer (ADX) provides real-time operational analytics on streaming time-series data. Data can be streamed directly into ADX from IoT Hub, or pushed from Azure Databricks using the [Kusto Spark Connector](#) from Microsoft as shown below.

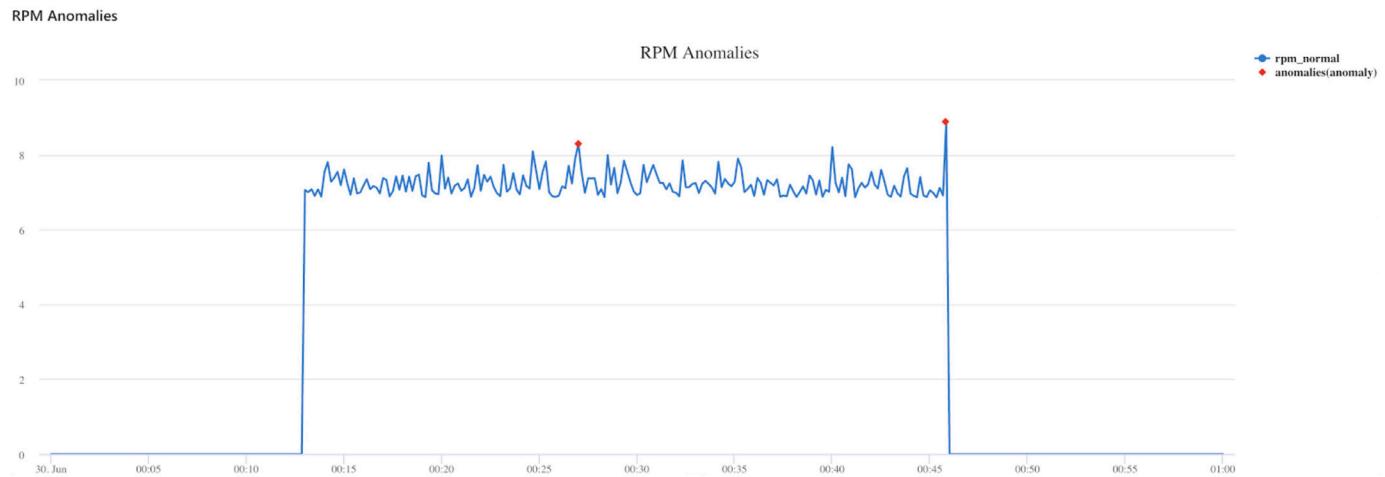
```
stream_to_adx = (
    spark.readStream.format('delta').option('ignoreChanges',True).table('turbine_enriched')
    .writeStream.format("com.microsoft.kusto.spark.datasink.KustoSinkProvider")
    .option("kustoCluster",kustoOptions["kustoCluster"])
    .option("kustoDatabase",kustoOptions["kustoDatabase"])
    .option("kustoTable", kustoOptions["kustoTable"])
    .option("kustoAadAppId",kustoOptions["kustoAadAppId"])
    .option("kustoAadAppSecret",kustoOptions["kustoAadAppSecret"])
    .option("kustoAadAuthorityID",kustoOptions["kustoAadAuthorityID"]))
)
```

Power BI can then be connected to the Kusto table to create a true, real-time, operational dashboard for turbine engineers.

ADX also contains native time-series analysis functions such as forecasting and anomaly detection. For example, the Kusto code below finds anomalous points for RPM readings in the data stream.

```
turbine_raw
| where rpm > 0
| make-series rpm_normal = avg(rpm) default=0 on todatetime(timestamp) in range(datetime(2020-06-30 00:00:00), datetime(2020-06-30 01:00:00), 10s)
| extend anomalies = series_decompose_anomalies(rpm_normal, 0.5)
| render anomalychart with(anomalycolumns=anomalies, title="RPM Anomalies")
```

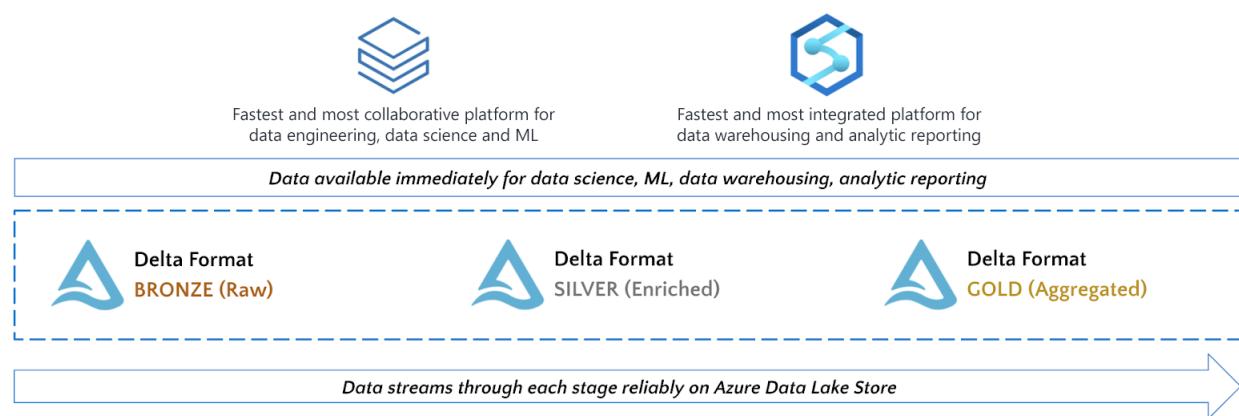
## The Deployment



With Azure Databricks' IIoT data analytics, data can be streamed directly into ADX from IoT Hub, or pushed from Azure Databricks, to generate real-time operational analytics on streaming time-series data.

## Analytical Reporting in ASA

Azure Synapse Analytics (ASA) is the next-generation data warehouse from Azure that natively leverages ADLS Gen 2 and integrates with Azure Databricks to enable seamless data sharing between these services.



While leveraging Azure Databricks and Azure Synapse, use the best tool for the job given your team's requirements.

## The Deployment

While leveraging the capabilities of Synapse and Azure Databricks, the recommended approach is to use the best tool for the job given your team's requirements and the user personas accessing the data. For example, data engineers that need the performance benefits of Delta and data scientists that need a collaborative, rich and flexible workspace will gravitate toward Azure Databricks. Analysts that need a low-code or data warehouse-based SQL environment to ingest, process and visualize data will gravitate toward Synapse.

The Synapse streaming connector for Azure Databricks allows us to stream the Gold turbine readings directly into a Synapse SQL Pool for reporting.

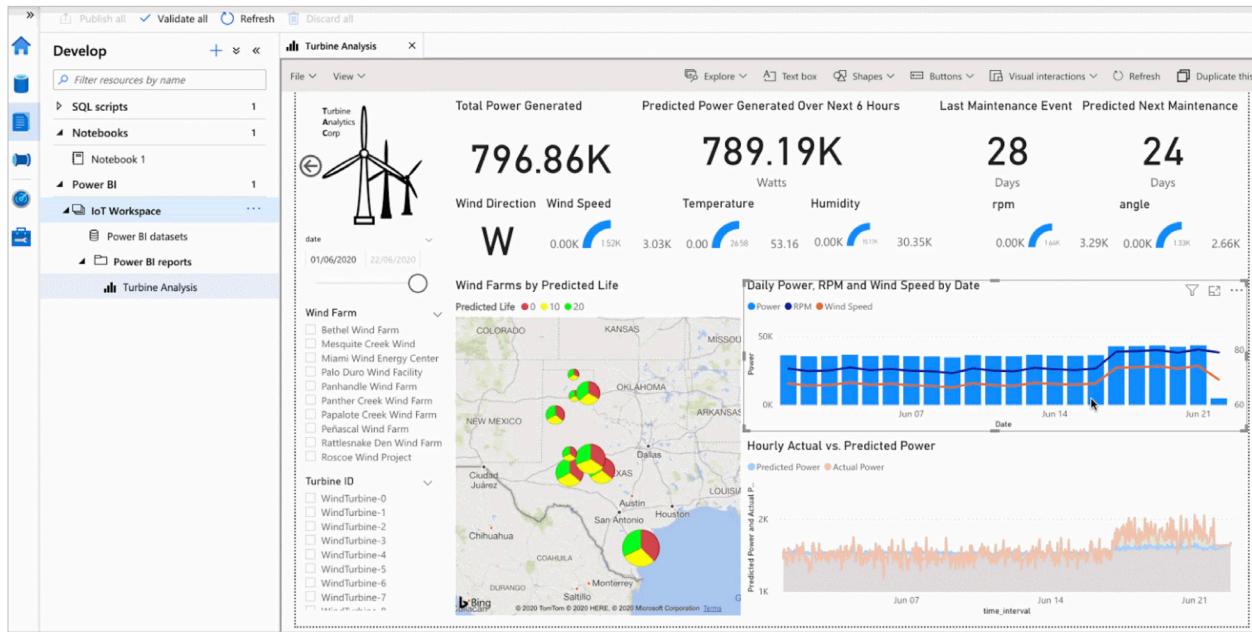
```
spark.conf.set("spark.databricks.sqldw.writeSemantics", "copy") # Use COPY INTO for faster loads

write_to_synapse = (
    # Read in Gold turbine readings
    spark.readStream.format('delta').option('ignoreChanges',True).table('turbine_enriched')
        .writeStream.format("com.databricks.spark.sqldw") # Write to Synapse
        .option("url",dbutils.secrets.get("iot","synapse_cs")) # SQL Pool JDBC (SQL Auth)
        .option("tempDir", SYNAPSE_PATH) # Temporary ADLS path
        .option("forwardSparkAzureStorageCredentials", "true")
        .option("dbTable", "turbine_enriched") # Table in Synapse to write to
        .option("checkpointLocation", CHECKPOINT_PATH+"synapse")
        .start()
)
```

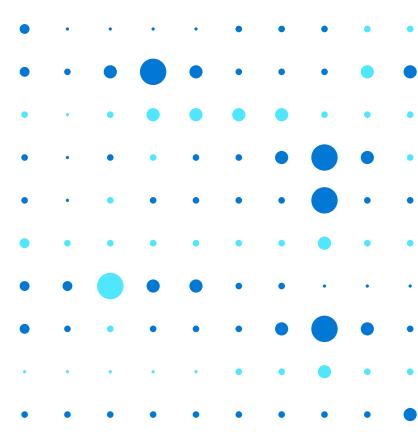
Alternatively, Azure Data Factory can be used to read data from the Delta format and write it into Synapse SQL Pools. More documentation can be found [here](#).

## The Deployment

Now that the data is clean, processed and available to data analysts for reporting, we can build a live Power BI dashboard against the data as shown below.



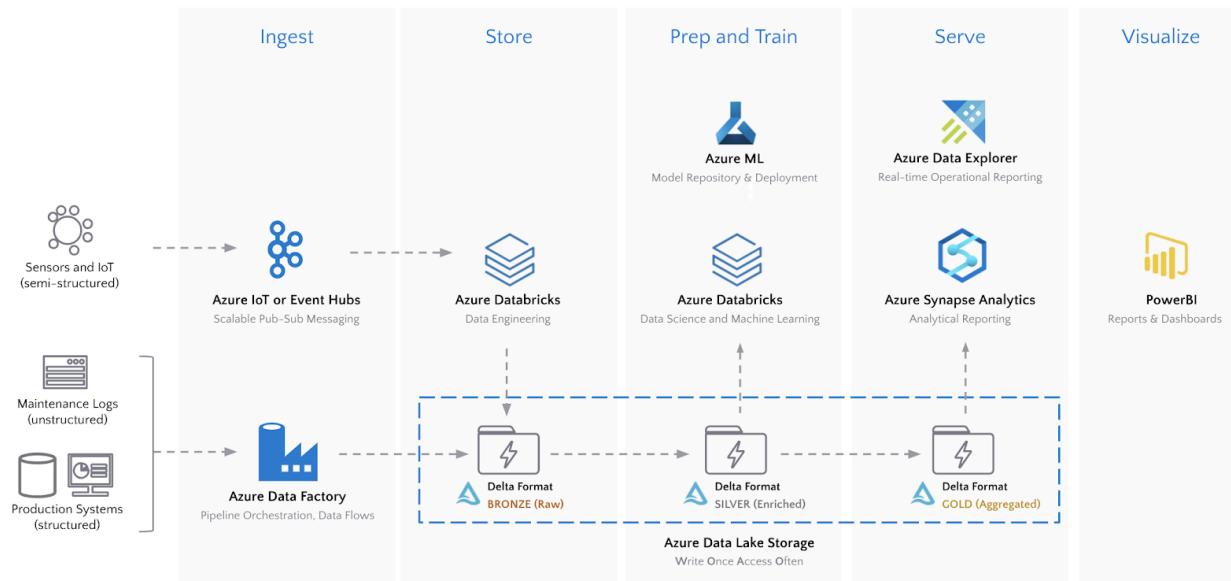
With Azure Databricks' IIoT data analytics, you can use the output to generate powerful real-time BI dashboards.



## Summary

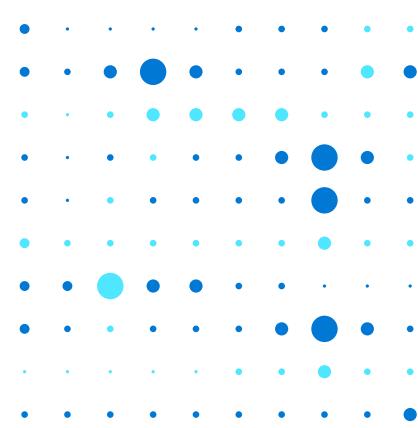
To summarize, we have successfully:

- Ingested real-time IIoT data from field devices into Azure
- Performed complex time-series processing on Data Lake directly
- Trained and deployed ML models to optimize the utilization of our wind turbine assets
- Served the data to engineers for operational reporting and data analysts for analytical reporting



The IIoT data analytic architecture featuring the Azure Data Lake Storage and Delta storage format offers data teams the optimal platform for handling time-series streaming data.

The key big data technology that ties everything together is Delta Lake. Delta on ADLS provides reliable streaming data pipelines and highly performant data science and analytics queries on massive volumes of time-series data. Lastly, it enables organizations to truly adopt a Lakehouse pattern by bringing best-of-breed Azure tools to a write-once, access-often data store.



## What's Next?

Try out the [data engineering notebook](#) and [machine learning notebook](#), learn more about Azure Databricks with this [three-part training series](#) and see how to create modern data architectures on Azure by attending [this webinar](#).

# Invent with purpose

