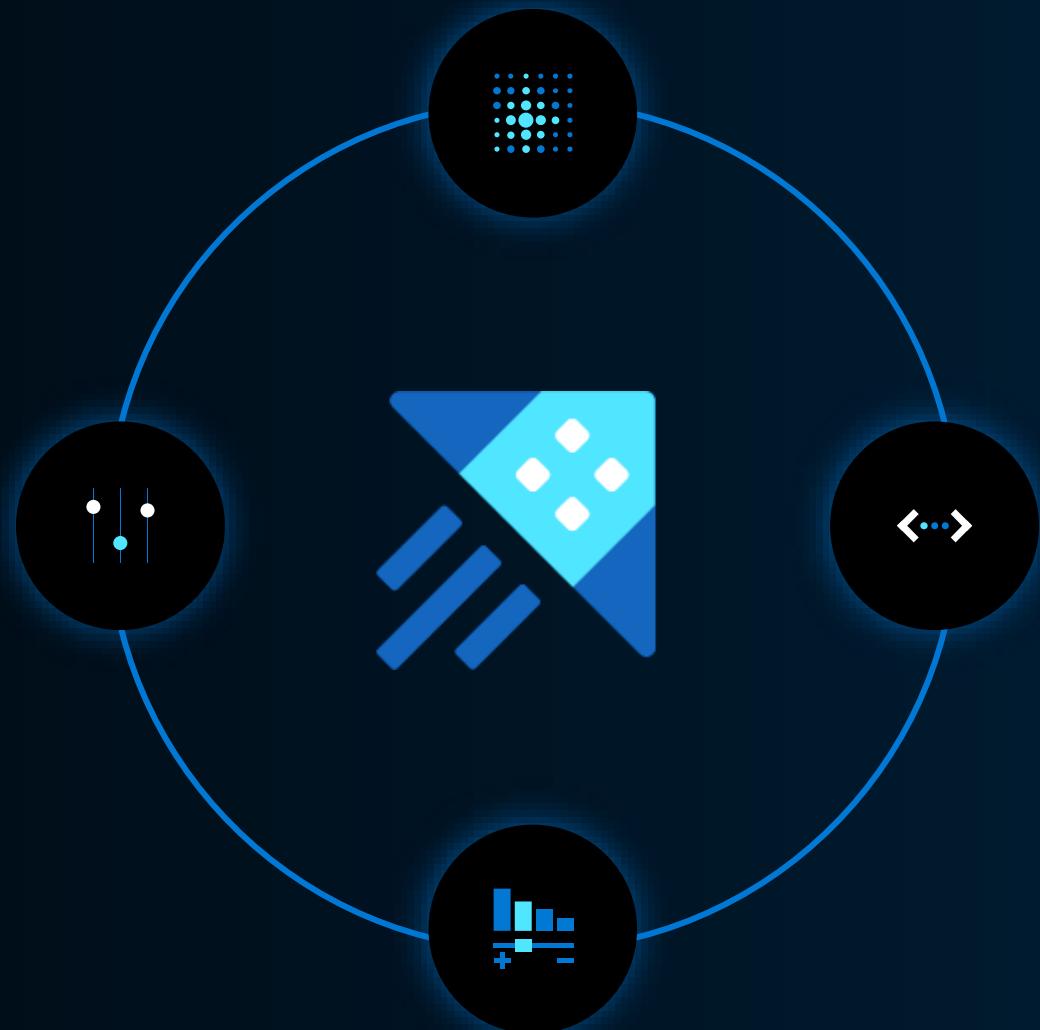


# ADX for IoT Analytics



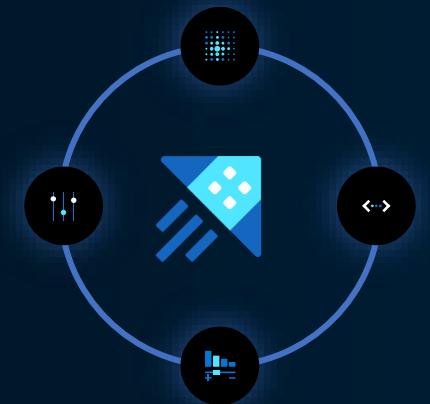
# Agenda



1. Intro
2. Overview
3. Architecture
4. Ingestion
5. ADX + IoT
6. Hands-on Lab
7. Azure Digital Twins
8. KQL for IoT
9. ML & Time series
10. Visualize Data
11. Ops & Management
12. Advanced topics

# Intro

## Module 1 Objectives



After completing this Learning, you will be able to understand:

1. Observational analytics for telemetry workloads
2. What is ADX
3. Innovation & Proven technology
6. Pattern for real-time analytics

# Telemetry - A key data for Digital Transformation

**50B**

connected devices by  
2030

**175ZB**

total amount of data by  
2025



# Kusto: Near real time analytics on observational data at petabyte scale

Analytics storage engine

Optimized

Native support

Low-latency

Human friendly query language

Now also in Synapse

for time stamped data: logs, time series, telemetry

for free text, structured and semi-structured data

Near real-time data analytics at scale

for ad-hoc analytics

# Kusto by the numbers



**135 PB**

Data ingested  
daily

**2.5 Billion**

Queries per day

**8.1 Exabyte**

Total Data Size

**2.4M**

VM Cores running

**>350K**

KQL Developers

A key to Microsoft's digital transformation journey

# Proven Technology



2015: Production

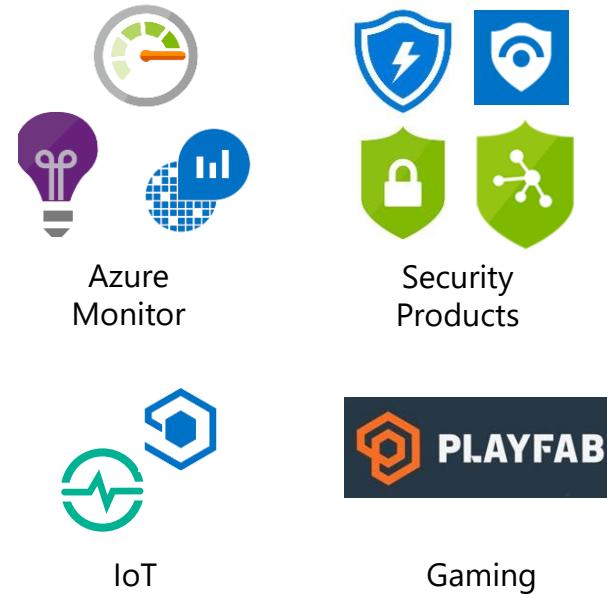
2019: Generally available

2021: Formally in Synapse

Battle tested for Microsoft internal workload



The platform for analytical solutions (SaaS)

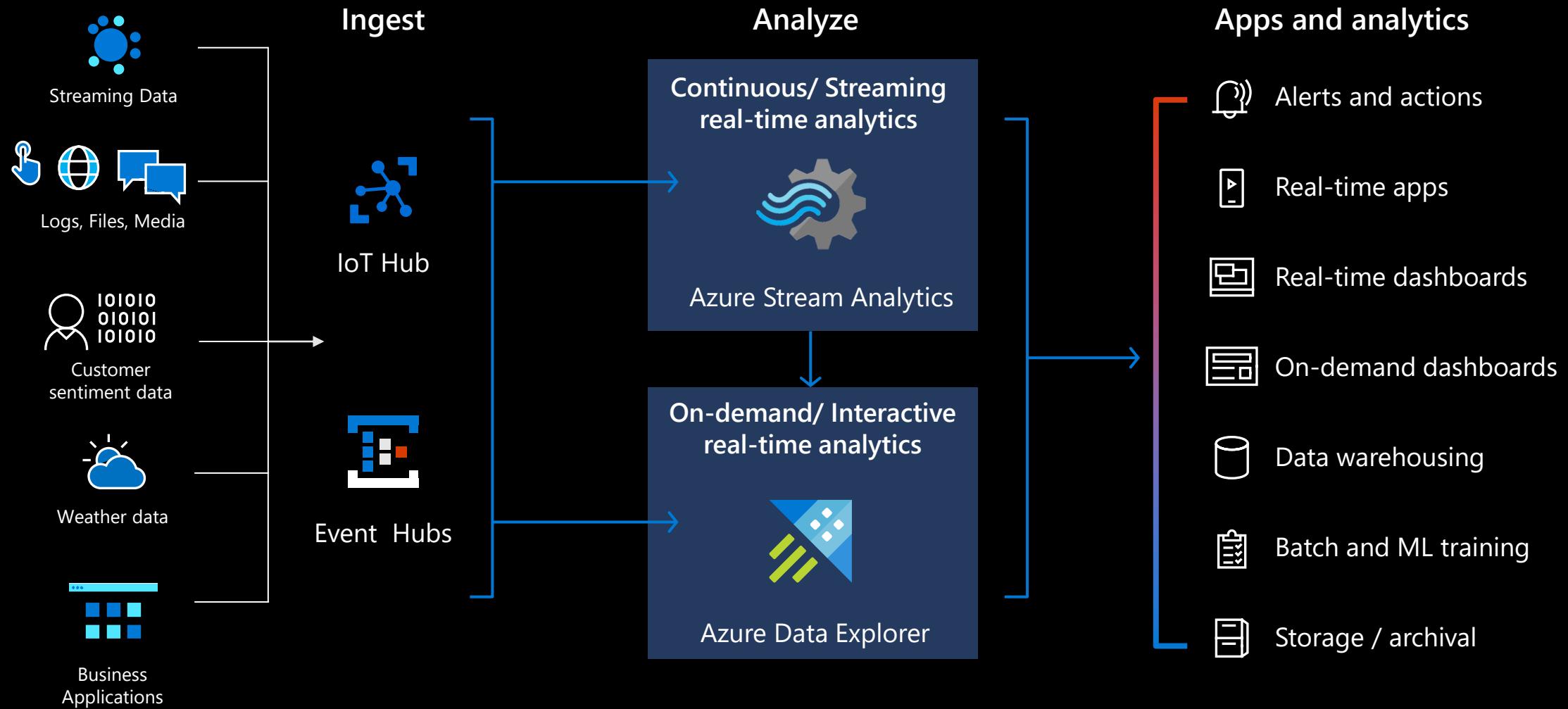


Available as PaaS



#PoweredByADX

# Azure powered pattern for real-time analytics

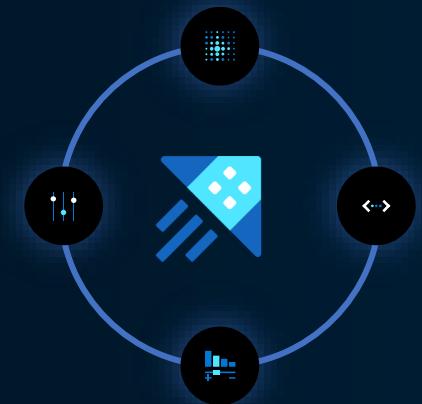


# Demo\*

- [Provision](#) ADX via Azure Portal.
- How to scale a cluster
- See insights blade
- Run basic [KQL](#)
- KQL cheat sheet ([PDF](#))

# Overview

## Module 1 Objectives



After completing this Learning, you will be able to understand:

1. Common use-cases
2. Deeper into what is ADX
3. Architecture
4. Batching vs streaming ingestion
5. Web UI capabilities
6. Extensibility
7. Enterprise Ready

# Major components of ADX



An ADX cluster does all the work to ingest, process, and query your data. The clusters are auto-scalable according to your needs. It stores the data on Azure Storage and caches some of this data on the cluster compute nodes to achieve optimal query performance.

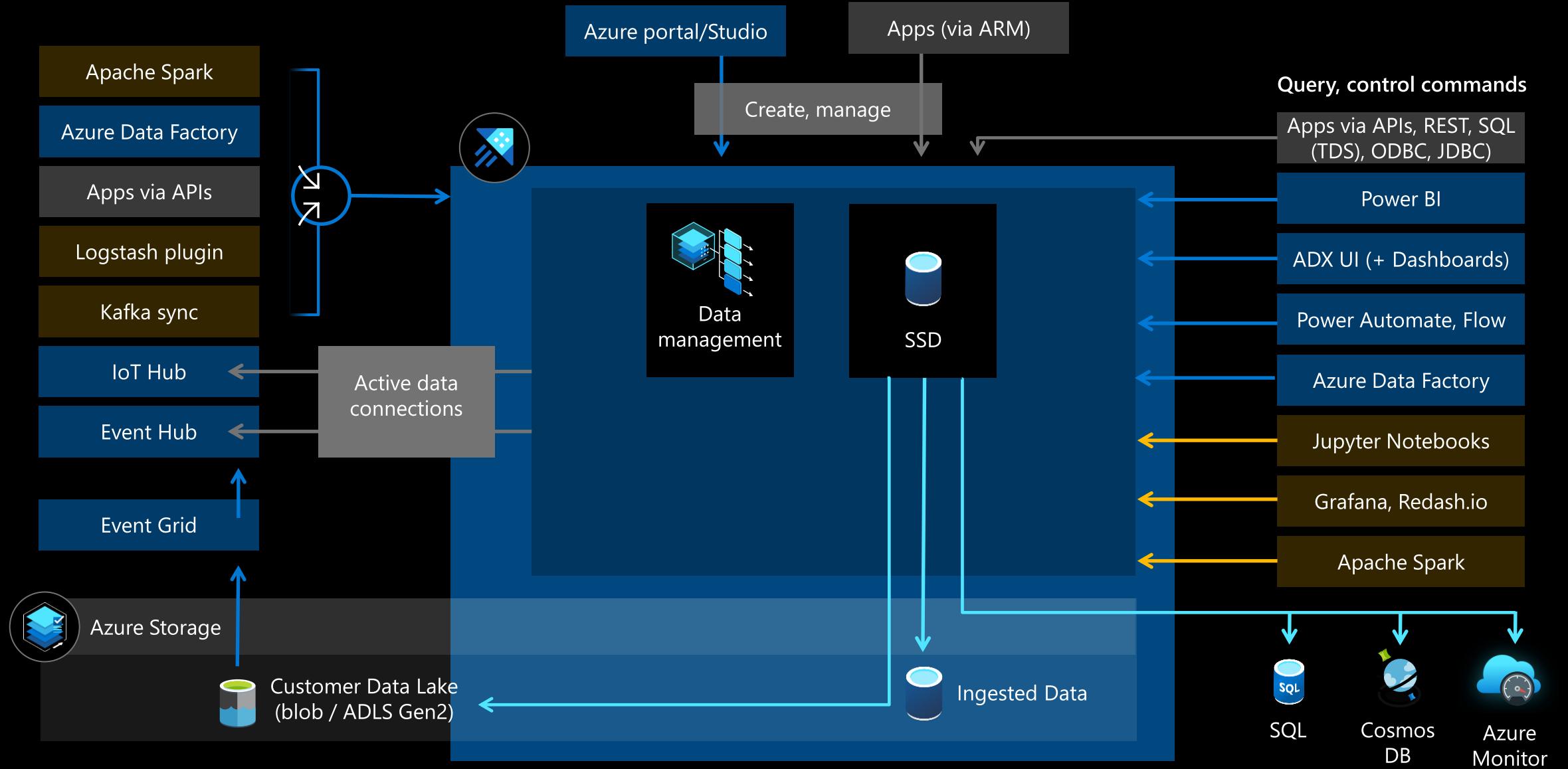
## What is an ADX cluster?

- Each ADX cluster can hold upto **10,000** databases and each database upto **10,000** tables.
- The data in each table is stored in **data shards** also called "**extents**".
- All data is **automatically indexed** and partitioned **based on the ingestion time**.
- There are no primary foreign key constraints or any other constraints, such as uniqueness.

The logical structure of a database is similar to many other relational databases. An ADX database can contain:

- **Tables:** Made up of a set of columns. Each column has one of nine different data types.
- **External tables:** Tables whose underlying storage is in other locations such as Azure Data Lake.

# Azure Data Explorer architecture



# Batching vs streaming ingestion



## Batching

- Optimized for high ingestion throughput
- Preferred method and most performant
- Data is batched according to properties
- Set [ingestion batching](#) policy on databases or tables
- Default max batching value is 5 minutes, 1000 items or total of 1 GB
- 4 GB data size limit for a batch ingestion command

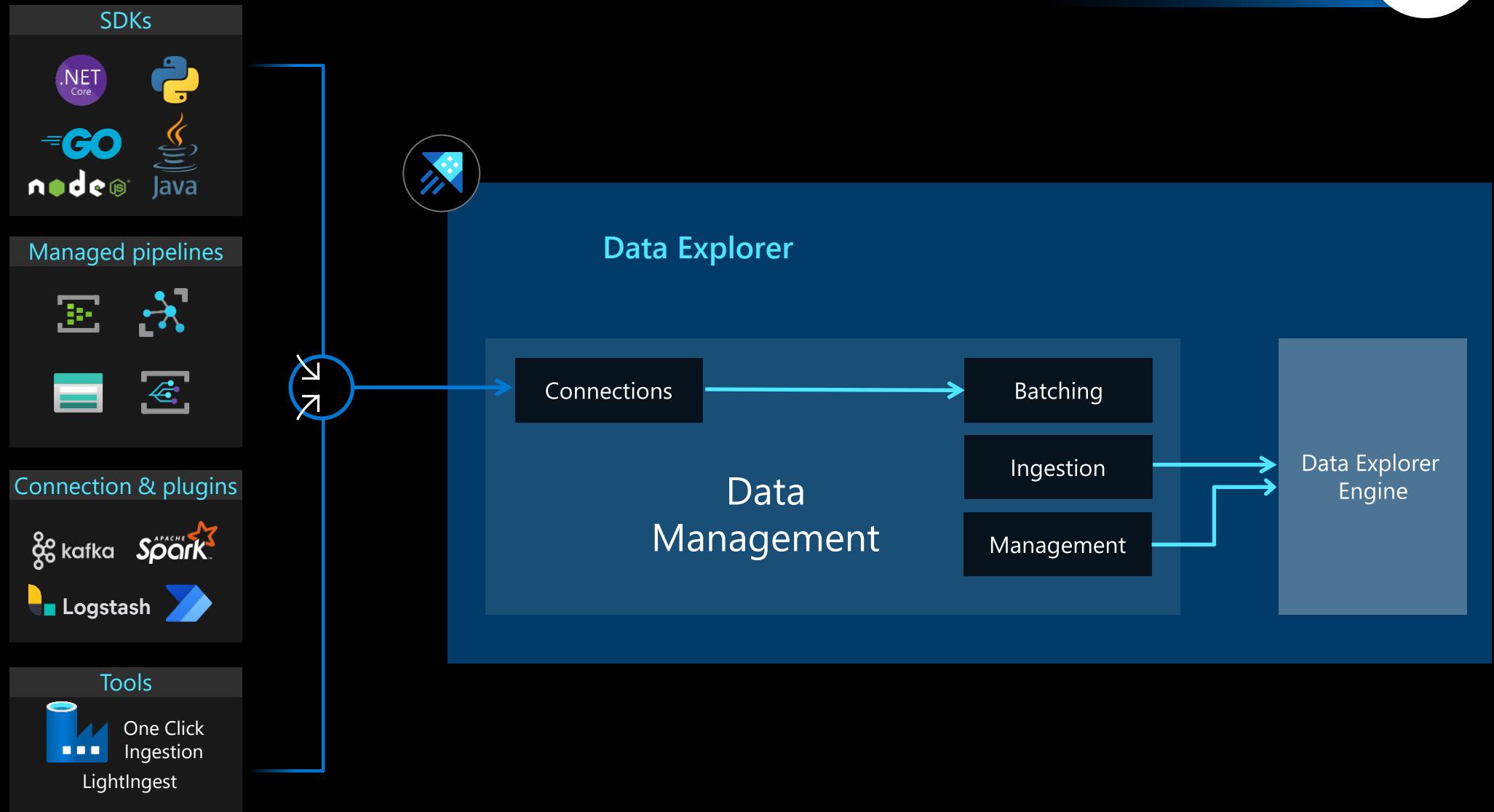
## Streaming

- Ongoing data ingestion from a streaming source
- Near real-time latency for small sets of data per table
- Initially ingested to row store
- Then moved to column store extents
- Streaming can be done using ADX client library or supported pipelines

**NOTE:** The recommendation is to ingest files between 100 MB and 1 GB.

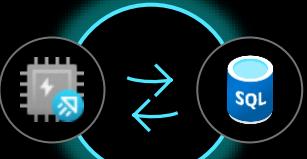
See more: [comparing-ingestion-methods-and-tools](#)

# Data Management and Ingestion



# Cross Queries



-  Between Data Explorer databases and clusters
-  Query SQL pool from Data Explorer
-  Query Azure Monitor from Data Explorer
-  Query Cosmos DB from Data Explorer
-  Query Data Lake from Data Explorer

# Data Sharing



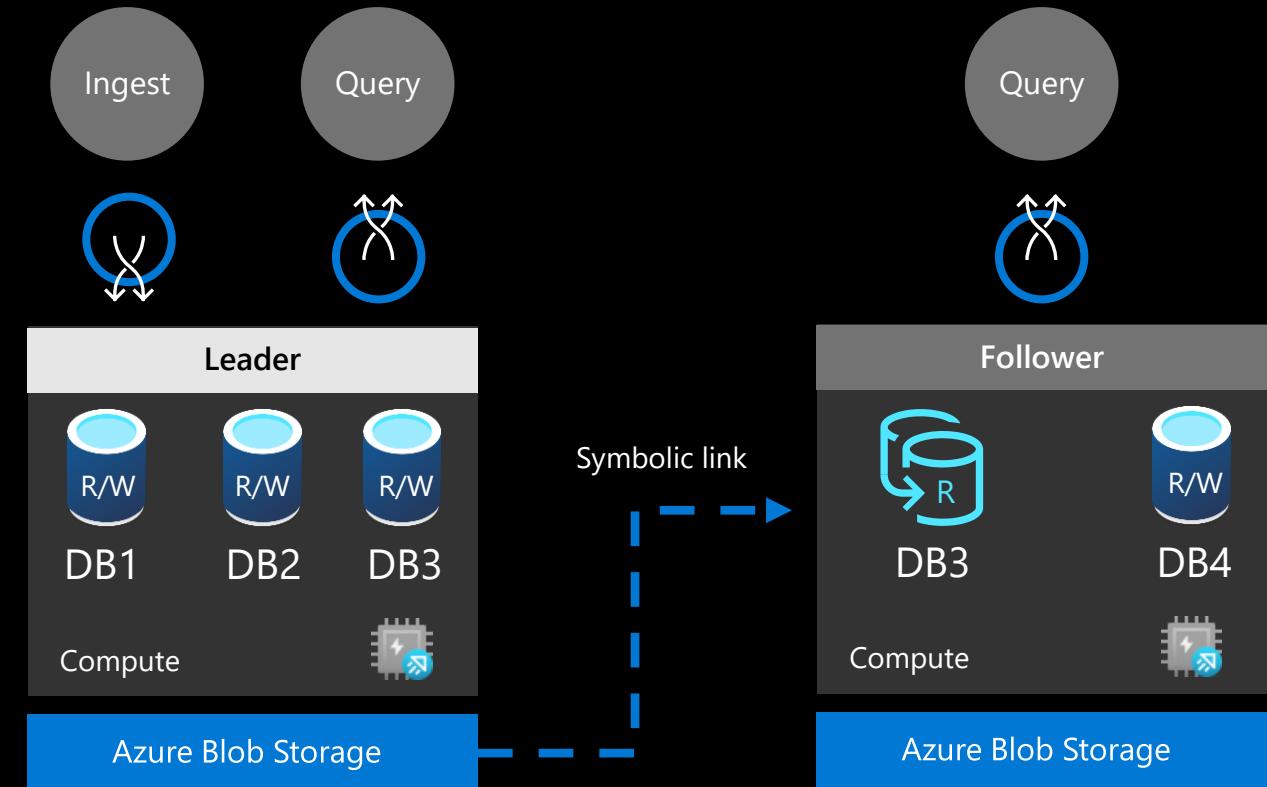
## Share data within the company

- Load balancing
- Data-as-a-service
- Hub and Spoke model
- Chargeback



## No Maintenance

- In-Place Data Sharing
- No data pipeline to maintain
- Near real-time updates



# Intuitive querying

## Simple and powerful

- Rich rational query language (filter, aggregate, join, calculated columns, and more)
- Built-in full-text search, time series, user analytics, geospatial, and machine learning operators
- Out-of-the box visualization
- Easy-to-use syntax + Microsoft IntelliSense
- Highly recognizable hierarchical schema entities

A screenshot of a code editor showing a Databricks SQL query:

```
1 GithubEvent
2 | where CreatedAt between(now() .. ago(1d))
3 | summarize count() by
```

The cursor is on the word "by". A dropdown menu shows the following suggestions:

- Actor
- CreatedAt
- Id
- Payload
- Public
- Repo
- Type
- abs(number)
- acos(number)
- ago(timespan)
- array\_concat(array, ...)
- array\_if(condition\_array, when\_true, when\_fals...

## Extensible

- In-line Python and R
- T-SQL



# Geospatial query

## 1. Geohash support

- Transformation from coordinates to geohashes and back
- Use-case: Summarization by geographical buckets, store locations based on a single column

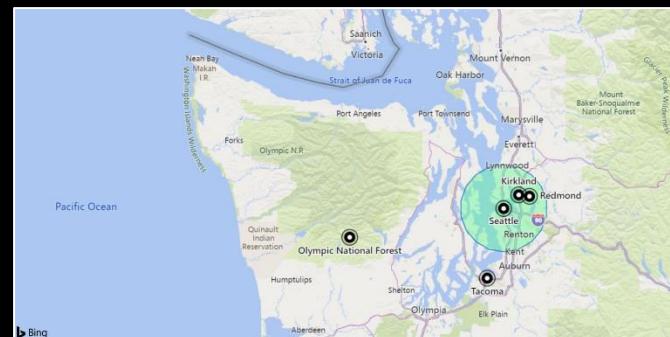


## 2. Distance

- Calculate the distance between two points

## 3. Contains

- Check whether a point is in a given circle
- Next: support for lines & polygons (i.e. check whether a point/line/polygon is in a polygon)



## 4. Next: Intersection

- Support for lines & polygons

Geospatial coordinates are interpreted as represented per the WGS-84 reference system.



# Advanced Analytics – Built-in & Extendible



## Out of the box

- Auto Clustering for Diagnosis and RCA
- Anomaly Detection
- Regression
- Forecasting
- Time Series Analysis library



## Spark Integration

- Native Spark connector for heavy duty model training
- Operationalize model into ADX for scoring



## Distributed Custom Code Execution

- Distributed Python and R execution
- Custom code is embedded in the KQL query



## Tools

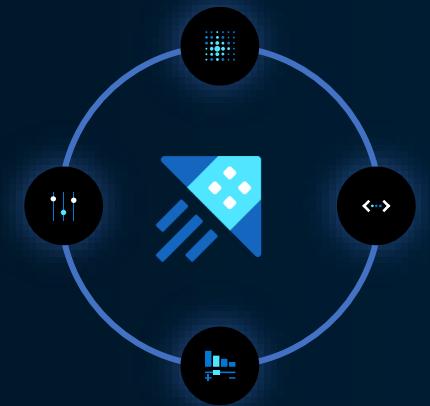
- Jupyter Integration with KQL Magic
- Python, Java SDKs

# Demo\*

- Transforming IoT Data with KQL

# Architecture

## Module 1 Objectives



After completing this Learning, you will be able to understand:

1. Deeper ADX architecture
2. Schema, Column Store, Extents, Row Store
3. Engine Service
4. How KQL works
6. Data Management Service

# Schema

Schema is  
Relational, Lightweight, Dynamic

Databases  
Authorization boundary  
Transaction boundary  
But not query boundary!

Supporting cross-database and cross-cluster queries

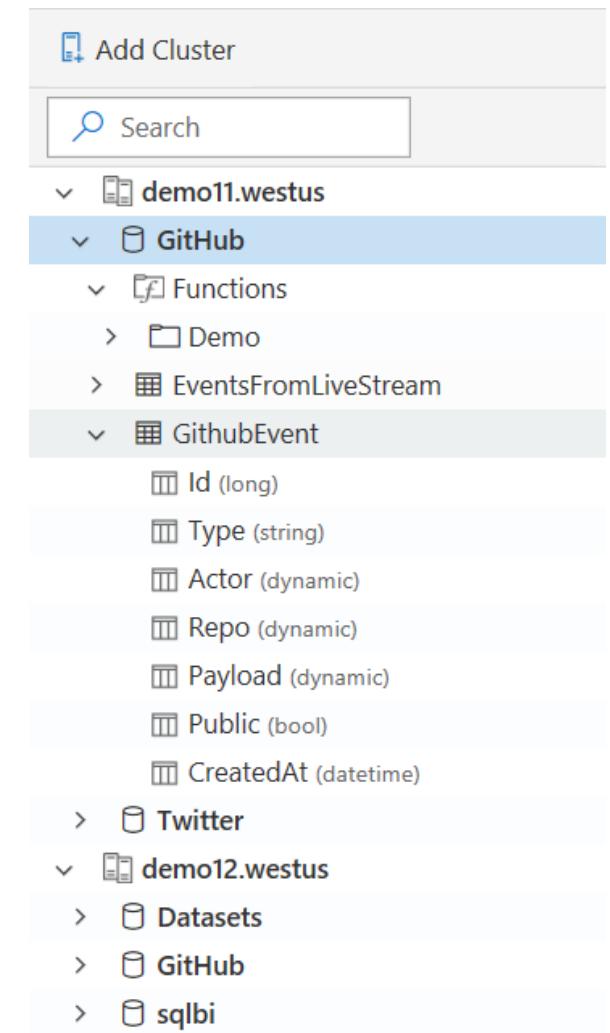
## Tables

Rectangular

## Columns

Supported types: Boolean, integer, real,  
date/time, timespan, string, dynamic (JSON)

## Stored functions

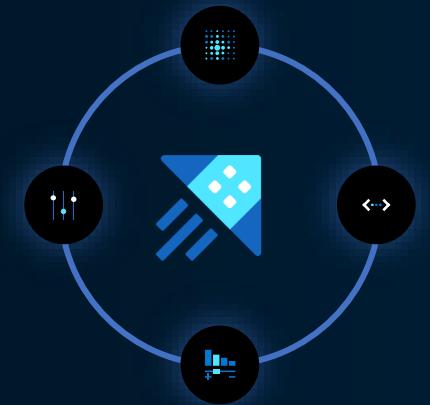


## Data Storage – Row Store

- Compression and indexing are very efficient at query time, but have a data freshness impact
  - One must batch records so that compression and indexing will be efficient
  - **Batching** adds an inherent latency
- Row Store (**Streaming** Ingestion)
  - Optimized for low latency – ingestion time is less than ~10 second
  - Data is held in-memory as uncompressed/unindexed
  - Once data accumulates, it is moved to columnar store

# Ingestion

## Module 1 Objectives



After completing this Learning, you will be able to understand:

1. Deeper ADX Ingestion
2. Queued
3. Streaming
4. Update Policy
6. Connectors
7. Light Ingest CLI

# Queued Ingestion

- Queued ingestion performs data batching and is optimized for high ingestion throughput.
- This is the preferred and most performant ingestion method.
- Data is batched according to ingestion properties.
- Small batches results in small extents. These extents will be later merged to larger extents, optimized
- for fast query results.
- **Batching** policy
  - Defined on cluster / database / table, inherited

```
.alter database Metrics policy ingestionbatching  
@'{ "MaximumBatchingTimeSpan": "00:00:30", "MaximumNumberOfItems": 500,  
"MaximumRawDataSizeMB": 1024}'
```

- Sealing batch by time / size / number of sources limits
- If not defined, default values are applied

# Streaming Ingestion

Home > New > Azure Data Explorer >

## Create an Azure Data Explorer Cluster

\* Basics   Scale   **Configurations**   Security   Network   Tags   Review + Create

### Configurations

Enable/disable the following Azure Data Explorer capabilities to optimize cluster costs and performance.

Streaming ingestion

On  Off

i Enabling streaming ingest can take a few minutes and requires service restart. Enabling streaming ingest consumes excess cluster resources. [Learn More](#)

Enable purge

On  Off

### Kusto

```
//create table  
.create table TestTable (TimeStamp:datetime, Name:string, Metric:int,  
Source:string)  
  
//define table policy  
.alter table TestTable policy streamingingestion enable  
  
//define database policy  
.alter database StreamingTestDb policy streamingingestion enable  
  
//control command  
.show table MyTable policy streamingingestion
```

# Data Formats and Mapping

## Data formats

- Text, Raw
- Delimiter separated values: CSV, TSV, TSVE, PSV, SCSV, SOH
- JSON (line-separated, multiline)
- Avro, Parquet, ORC
- Supports ZIP and GZIP compression

## Column mapping

- Rename, reorder, or skip fields
- CSV mapping maps field ordinals to columns
- JSON and Avro mappings map field paths to columns

```
.create table Logs ingestion csv mapping "CsvLogMapping"
'['
'  { "column" : "Timestamp", "DataType":"datetime", "Properties": {"Ordinal":"0"}},'
'  { "column" : "Text", "DataType":"string", "Properties": {"Ordinal":"1"}}
']'

.create-or-alter table Logs ingestion json mapping "JsonLogMapping"
'['
'  { "column" : "Timestamp", "Properties": {"Path":("$.timestamp")}},'
'  { "column" : "Text", "Properties": {"Path":("$.log_event")"}}
']'
```

# Update Policy

Used for additional processing over the ingested data or for distributing data between tables.

Invoked whenever new data is appended to the source table.

The query can invoke stored functions.

Scenarios

- Filter out data, deduplication
- Created calculated columns, changing schema
- Different retention policies for subset of the data

```
.alter table KustoErrorLogs policy update
'[{"Source": "KustoLogs", "Query": "ErrorLogs()"}]'

.create function ErrorLogs() {
    KustoLogs
    | where Level == 'Error'
}
```

# Ingest SDKs

- Ingest client libraries are used for performing queued ingestion.
- Data is uploaded to a blob and a message is posted to the Data Management service queue.
- The message contains reference to the blob, mandatory ingestion properties (Database, Table) and optional ingestion properties (Format, IngestionMapping etc.)

- SDKs

- .NET Standard
- Python
- Node.js
- Java
- REST API
- Go

```
var ingestUri = "https://ingest-<ClusterName>.<Region>.kusto.windows.net:443/";
var database = "<Database>";
var table = "<Table>";
var blobPath = "";

var ingestConnectionStringBuilder =
    new KustoConnectionStringBuilder(ingestUri)
{
    FederatedSecurity = true,
    InitialCatalog = database
}
    .WithAadApplicationKeyAuthentication(applicationClientId, applicationKey, tenantId);

using (var ingestClient = KustoIngestFactory.CreateQueuedIngestClient(ingestConnectionStringBuilder))
{
    var properties =
        new KustoQueuedIngestionProperties(database, table)
    {
        Format = DataSourceFormat.csv,
        IngestionMapping = tableMapping,
        IgnoreFirstRecord = true
    };

    ingestClient.IngestFromStorageAsync(blobPath, properties);
}
```

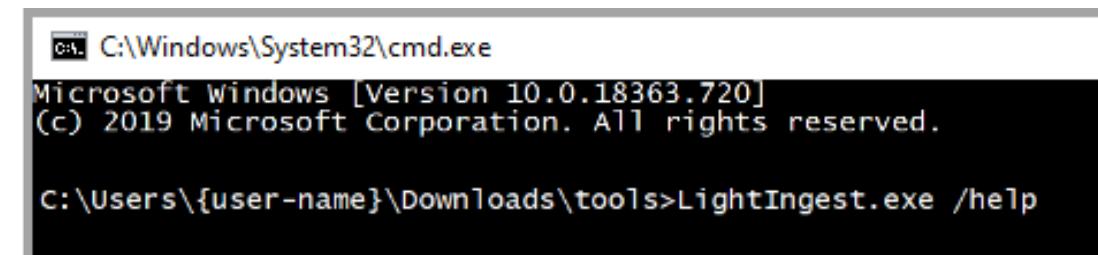
# Tools

## Light Ingest

- CLI Command tool to ingest data from Azure Storage into ADX
- Backfill

```
LightIngest "Data Source=https://ingest-demo.westus.kusto.windows.net;AAD Federated Security=True" -db:demo -table:Trips -source:"" -pattern:"*.csv.gz" -format:csv -limit:2 -ignoreFirst:true -cr:10.0 -dontWait:true
```

```
LightIngest "Data Source=https://ingest-demo.eastus.kusto.windows.net;AAD Federated Security=True" -db:demo -table:Trips -source:"https://demo.blob.core.windows.net/adx" -creationTimePattern:'historicalvalues'yyyyMMdd'.parquet'" -pattern:"*.parquet" -format:parquet -limit:2 -cr:10.0 -dontWait:true
```



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\{user-name}\Downloads\tools>LightIngest.exe /help
```

See More: [How to ingest data using CreationTime](#)

# Light Ingest CLI

```
Windows PowerShell
```

```
C:\kusto\tools>LightIngest.exe "Data Source=https://ingest-adxpm10774.eastus.kusto.windows.net;AAD Federated Security=True" -db:IoTAnalytics -table:SampleTable -source:"https://westuskustopublic.blob.core.windows.net/public/" -pattern:"SampleData-500-4394582f-668f-4d03-8bba-58f87a7e48a0.json" -format:json -limit:2 -cr:10.0 -dontWait:true -mappingRef:Mapping01
LightIngest invoked with the following arguments: Data Source=https://ingest-adxpm10774.eastus.kusto.windows.net;AAD Federated Security=True -db:IoTAnalytics -table:SampleTable -source:https://westuskustopublic.blob.core.windows.net/public/ -pattern:SampleData-500-4394582f-668f-4d03-8bba-58f87a7e48a0.json -format:json -limit:2 -cr:10.0 -dontWait:true -mappingRef:Mapping01

Please review the run parameters:

Connection string      : Data Source=https://ingest-adxpm10774.eastus.kusto.windows.net;AAD Federated Security=True
-database              : IoTAnalytics
-table                 : SampleTable

-sourcePath             : https://westuskustopublic.blob.core.windows.net/public/
-pattern               : SampleData-500-4394582f-668f-4d03-8bba-58f87a7e48a0.json
-creationTimePattern   :
-format                : json
-ignoreFirstRow         : False
-ingestionMappingRef    : Mapping01

-compression            : 10
-ingestTimeout (min)    : 60
-limit                 : 2
-dontWait               : True

Press [Ctrl+Q] to abort, press any other key or wait for 10 seconds to proceed
==> Starting...
ListBlobs: enumerating blobs under container 'https://westuskustopublic.blob.core.windows.net/public/' with prefix ''
ListBlobs: listed 1 blobs
==> Items discovered: [      1], filtered: [      1], posted for ingestion: [      1]
Done. Time elapsed: 00:00:01.7348310
Items discovered: [      1], filtered: [      1], posted for ingestion: [      1]
```

# Demo\*

- One-click ingestion CSV
- `.create table ingestion json mapping`
- `.ingest` into table from blob

# Knowledge check - Azure Data Explorer

## What best describes the ADX?

- a) Big data storage data optimized for batching non-interactive queries.
- b) Big data solution for data transformations.
- c) Big data analytics cloud platform optimized for high volume, interactive queries.

## What is a control command in ADX?

- a) Control commands can be used for maintenance and policy tasks.
- b) A query.
- c) A way to visualize data.

## Which of the following is an appropriate use case for ADX?

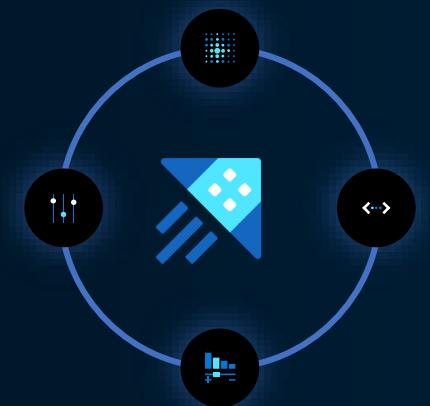
- a) Log and Text analytics
- b) In-memory batching processing of petabytes of information.
- c) Real-time analytics

## Which aspect of an extract, transform, and load (ETL) process makes it unsuitable for ADX.

- a) Long running tasks.
- b) Large volumes of data for ingestion.
- c) High query concurrency.

# IoT Analytics

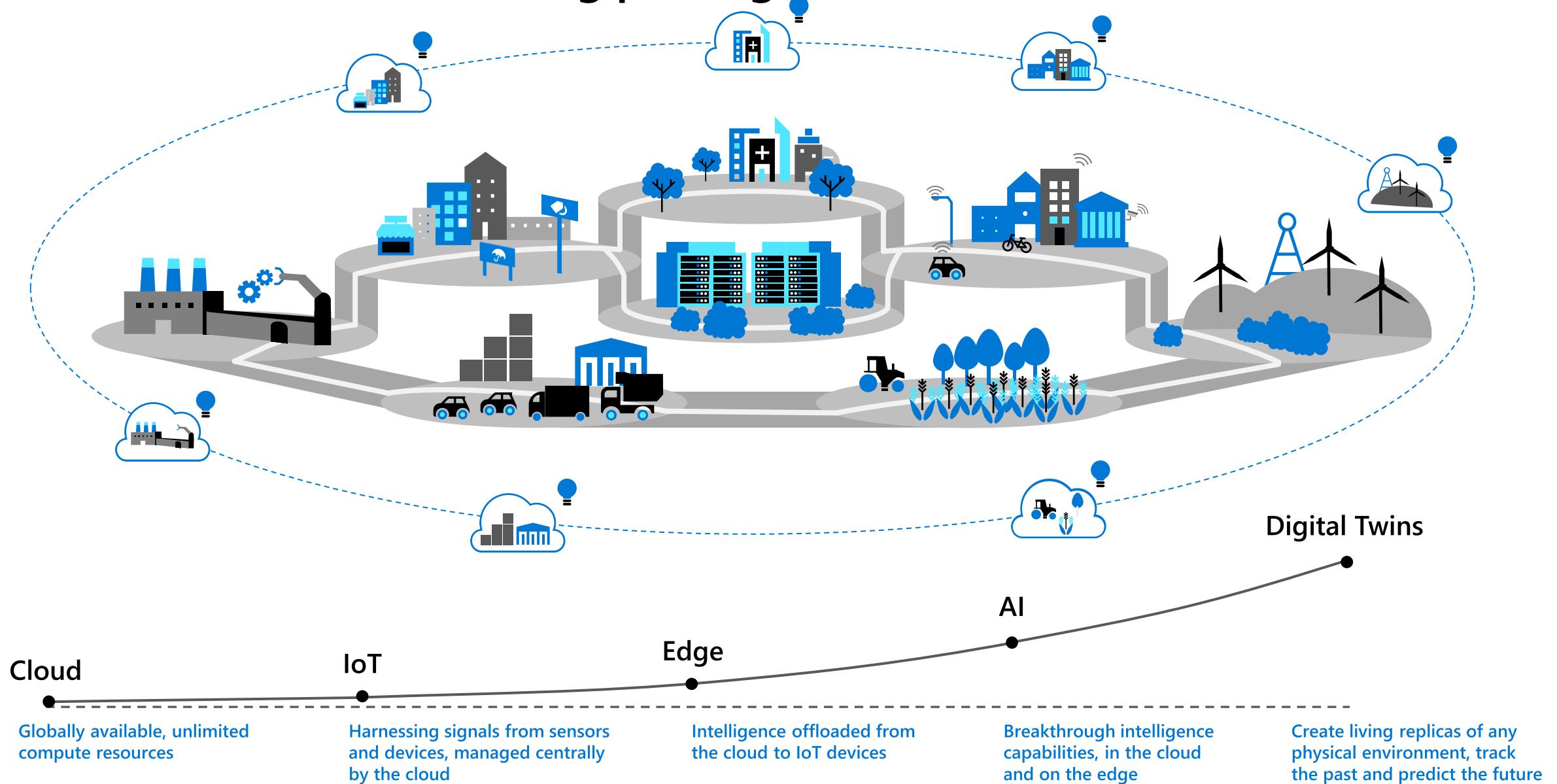
## Module 2 Objectives



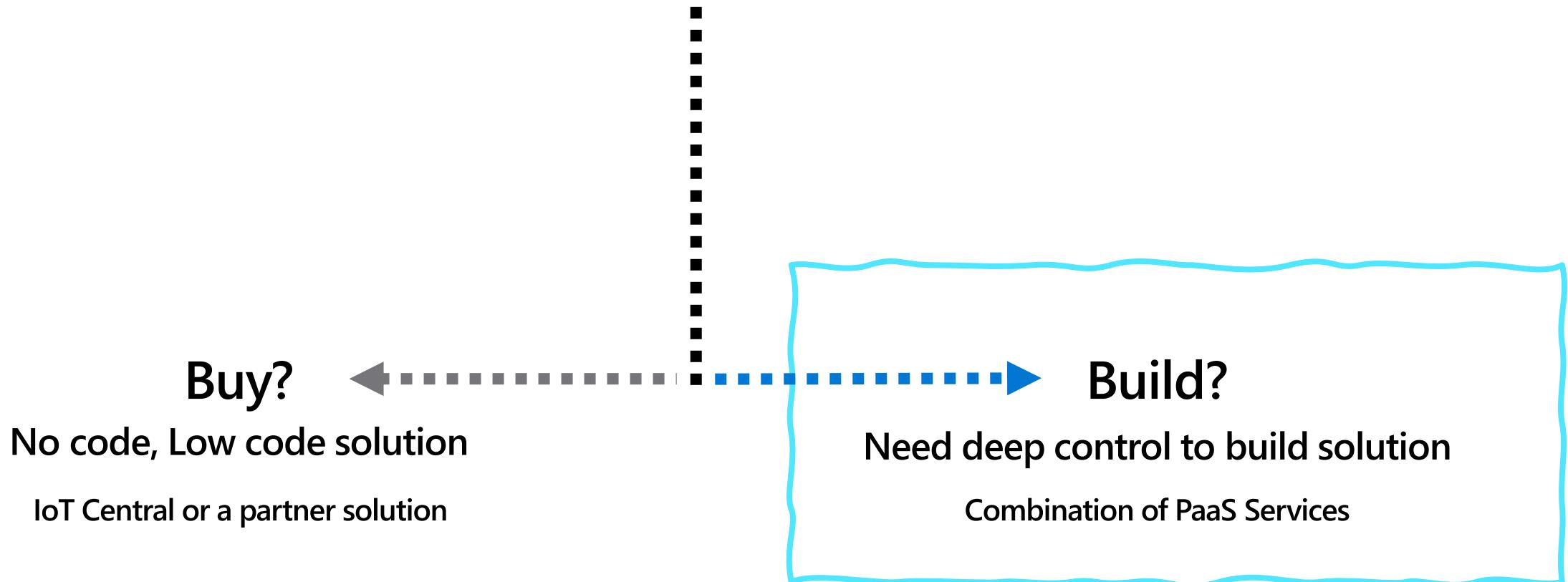
After completing this Learning, you will be able to understand:

1. Why IoT is essential
2. Build solution using Platform as a Service (PaaS) in Azure
3. IoT reference architecture
4. End-to-end demo for IoT analytics

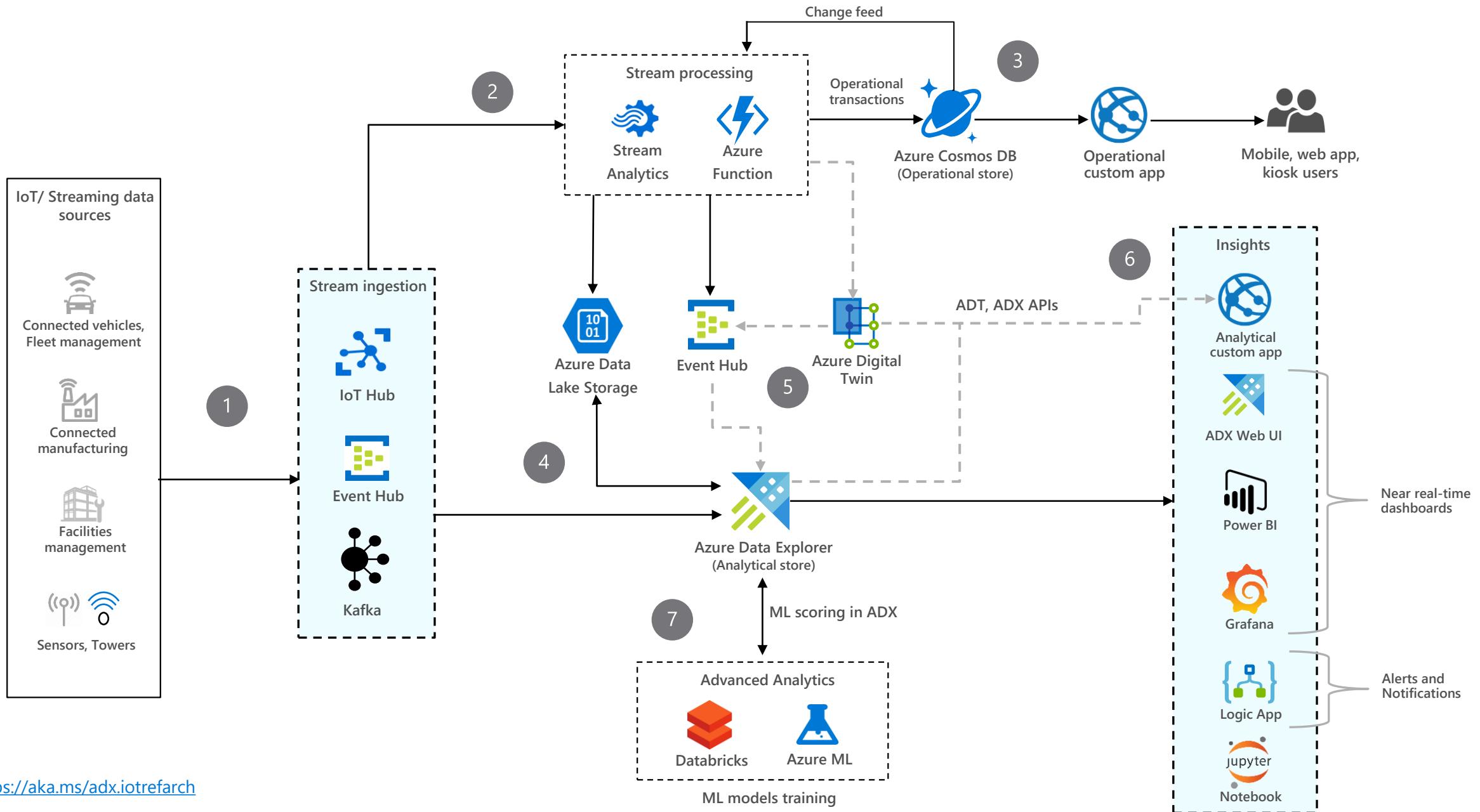
# IoT is essential to an evolving paradigm in our world



# Your options for IoT analytics solutions on Azure



# IoT Analytics reference architecture with Azure Data Explorer

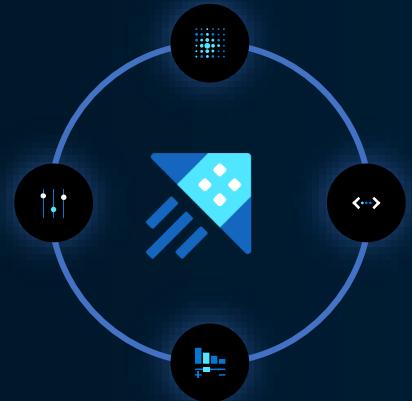


# Demo\*

- Thermostat data
- JSON
- Render
- Fill gaps
- Forecast
- Anomalies
- Dashboards
- Materialized Views
- External Tables

# Hands-on Lab: [aka.ms/adx.iot](https://aka.ms/adx.iot)

## Module 3 Objectives



After completing this Learning, you will be able to understand:

1. Deployment architecture
2. Resources of the lab
3. KQL sample queries for IoT analytics
4. Aggregations
5. How to fill missing data
6. Forecasting & anomaly detection
7. Query physical twin model
8. Materialized views

# Lab Architecture



IoT Central



Store Apps

Event Hubs



iotdata  
historicdata

Event Grid



historicdata

Digital Twins



office  
floor  
thermostat

Data Explorer



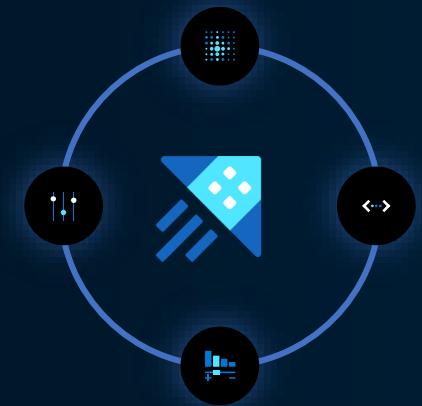
StageloTRawData  
Thermostats

Thermostat\_January2022



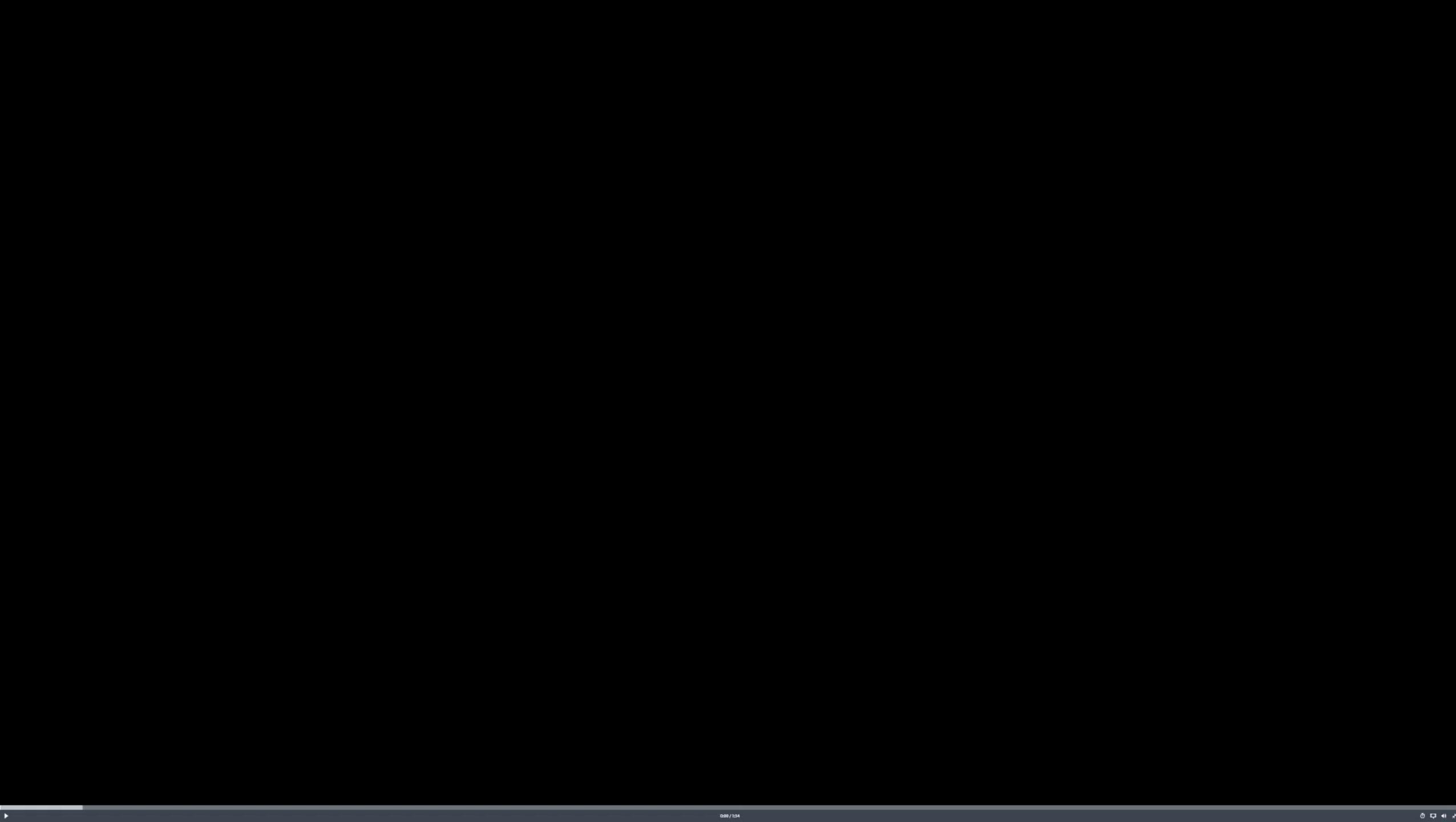
# Digital Twins

## Module 4 Objectives



After completing this Learning, you will be able to understand:

1. Azure Digital Twins
2. Using the twin graph
3. Data integration
4. Model the real world
5. Use-cases & examples



# Azure Digital Twins

- Digital replica of real-world things
- Create living replicas of your physical environment (places or people)
- Model your connected devices and run analytics on that data (business processes).



# Azure Digital Twins and time-series data

*How did the temperature on the fourth floor of the Contoso building respond to thermostat changes over the last 12 hours?*

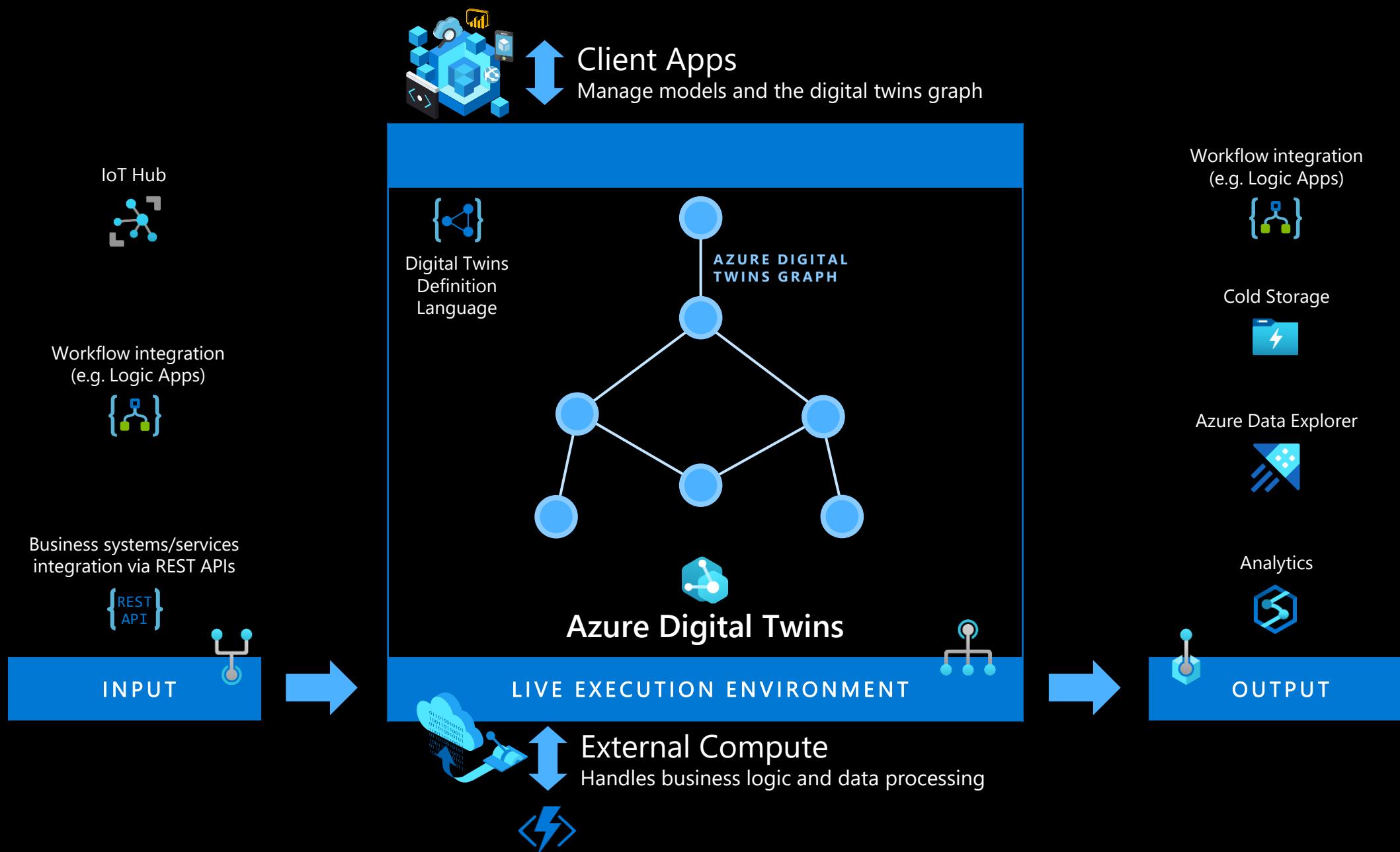
*How did power demand change for substations feeding the football arena during Saturday's game?*

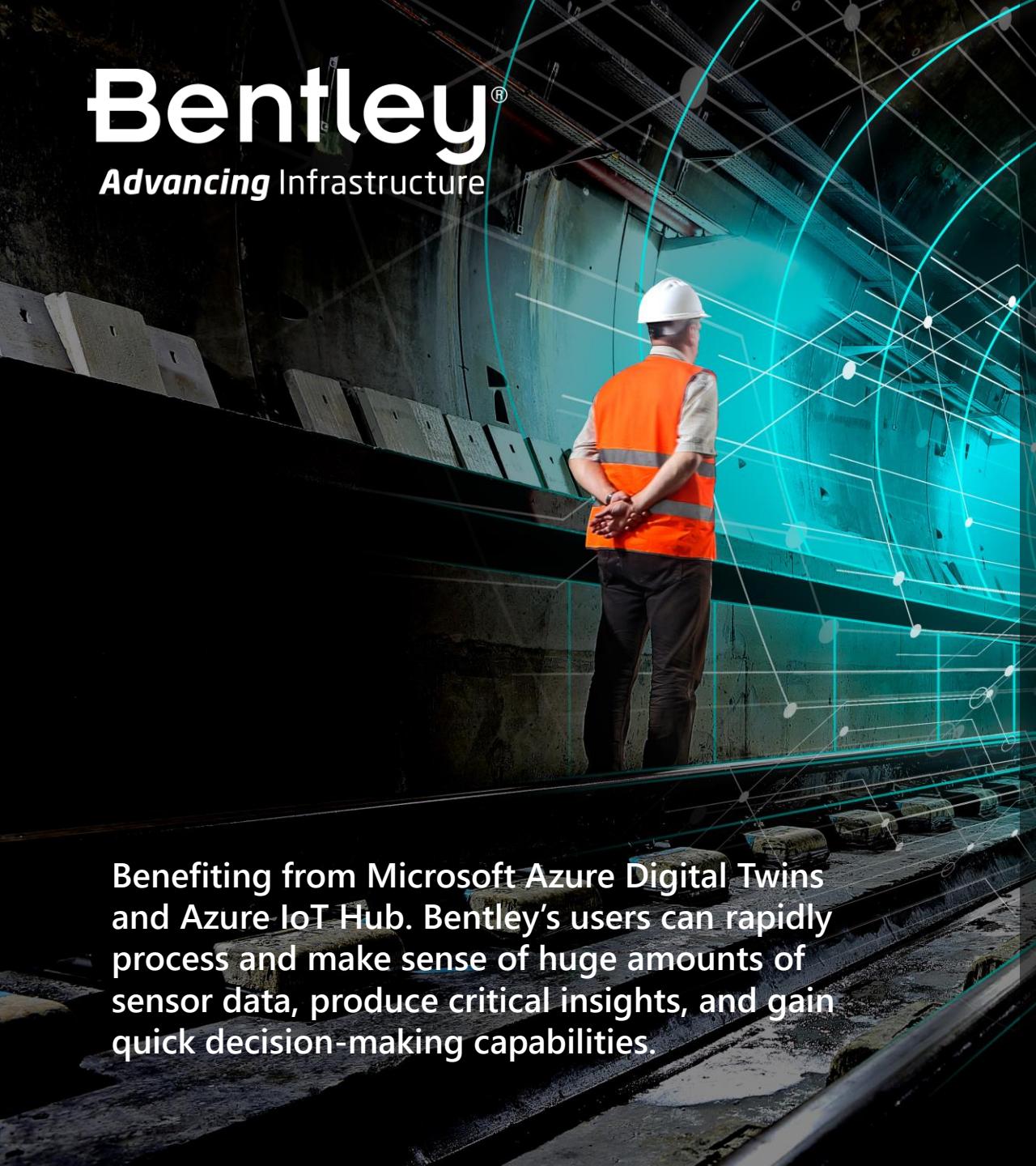
*How was the rate of production impacted on lines A and B in Factory 27 during the supply disruption last week?*

- twin graph data (Azure Digital Twins)
- historical time-series data (Azure Data Explorer)

Use the **twin graph** to *contextualize* **time series** data

# Create next generation IoT solutions that model the real world



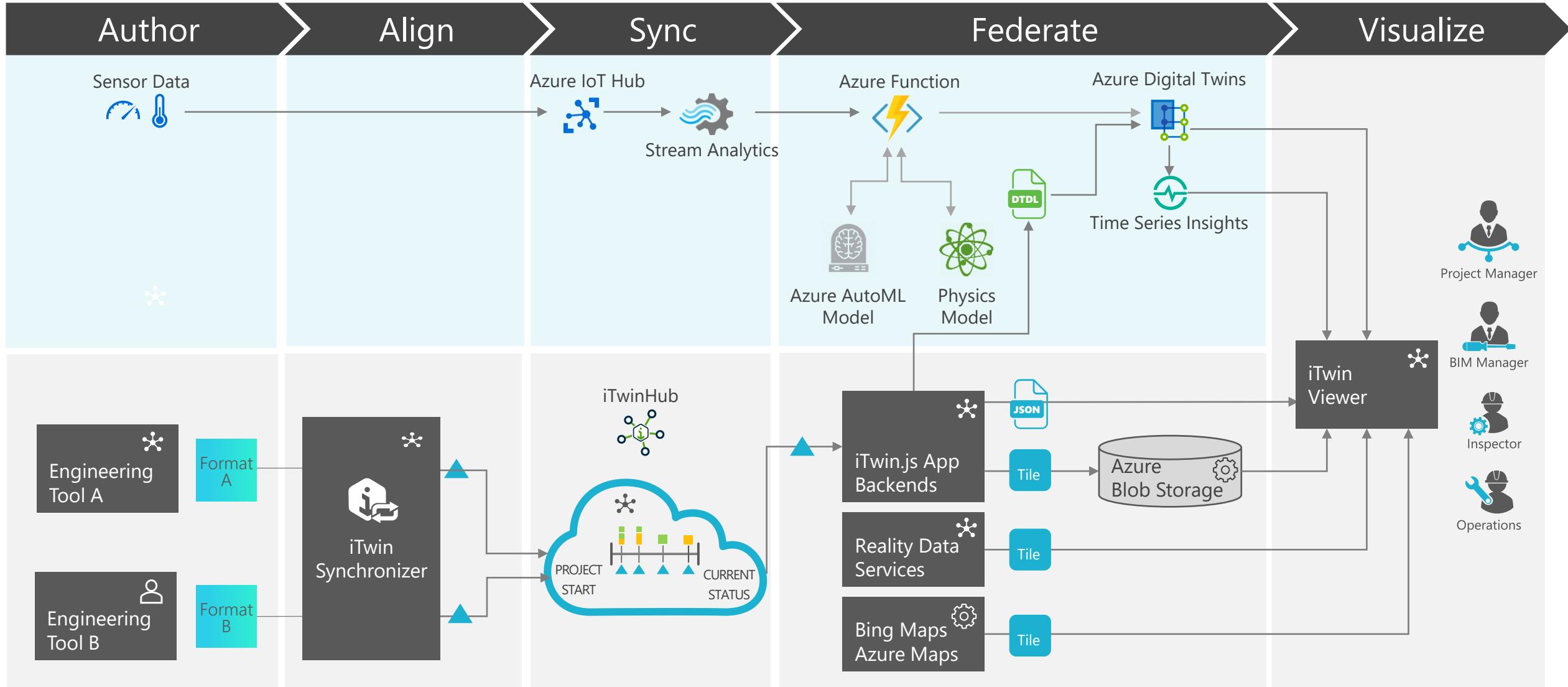


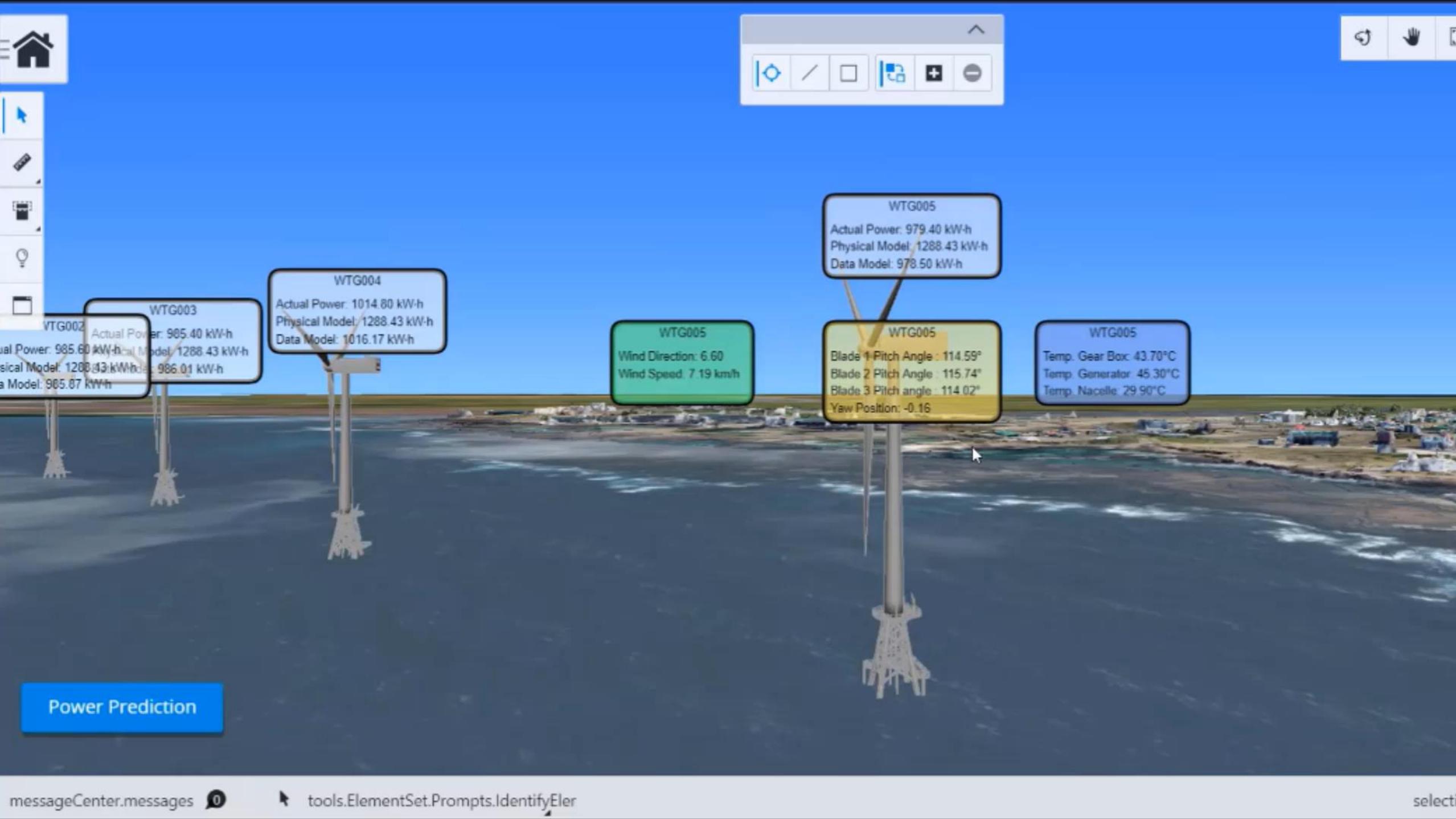
Benefiting from Microsoft Azure Digital Twins and Azure IoT Hub. Bentley's users can rapidly process and make sense of huge amounts of sensor data, produce critical insights, and gain quick decision-making capabilities.

## Enabling World-class O&M with Digital Twins

- Visualize, simulate, and optimize every aspect of critical infrastructure including energy efficiency, carbon footprint, safety, and disaster response
- Track current performance, predict future performance, and reduce maintenance costs
- Monitor energy consumption, air quality, space occupancy and temperature
- Simulate the impact of workplace incidents or extreme weather events

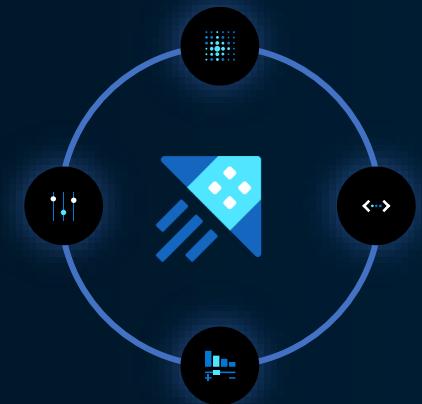
# Digital Twins - Viewer





# Kusto Query Language (KQL)

## Module 5 Objectives



After completing this Learning, you will be able to understand:

1. What is KQL
2. let, set and tabular statements
3. Common KQL for IoT
4. Materialized views
5. External tables
6. Azure Digital Twin query request
7. Shuffle
8. Best Practices

# Kusto Query Language (KQL) overview

- Kusto Query Language is a powerful tool to explore your data and discover patterns, identify anomalies and outliers, create statistical modeling, and more.
- The query uses schema entities that are organized in a hierarchy similar to SQL's: databases, tables, and columns.

# What is a Kusto query?

- A Kusto query is a read-only request to process data and return results.
- The request is stated in plain text, using a data-flow model that is easy to read, author, and automate.
- Kusto queries are made of one or more query statements.

# What is a query statement?

There are three kinds of user [query statements](#):

1. A [tabular expression statement](#)
2. A [let statement](#)
3. A [set statement](#)

All query statements are separated by a ; (semicolon), and only affect the query at hand.

**Note:**

For information about application query statements, see [Application query statements](#).

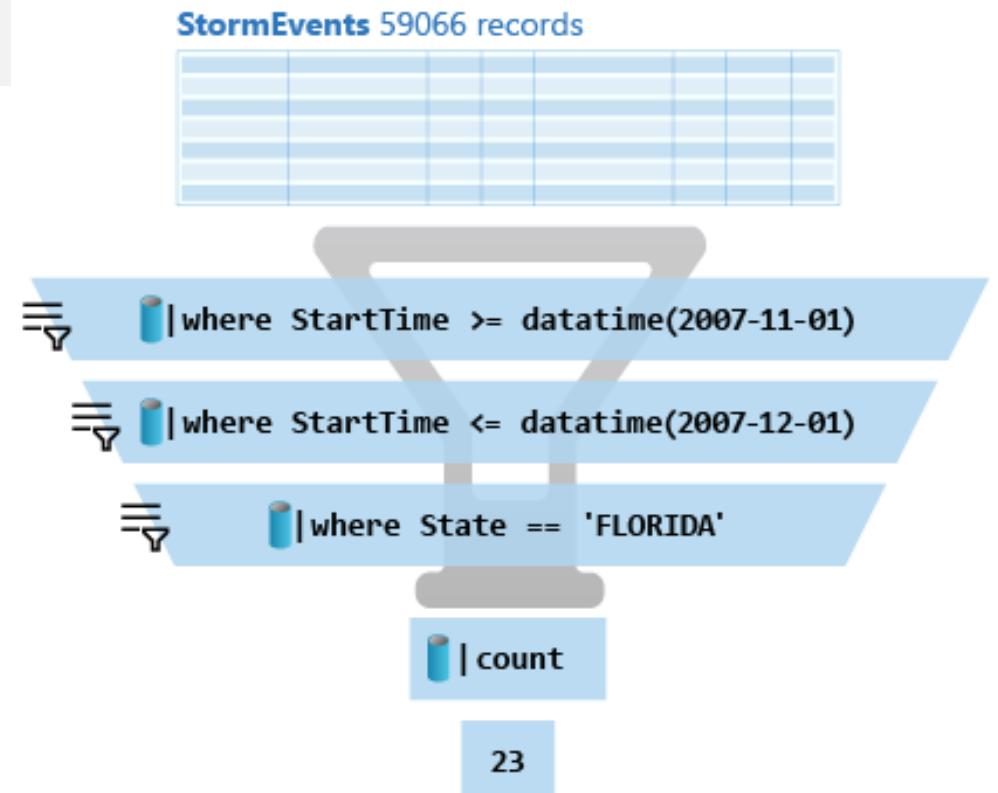
# What is a tabular statement?

Let's look at an example query:

**Kusto**

```
1 StormEvents  
2 | where StartTime between (datetime(2007-11-01) .. datetime(2007-12-01))  
3 | where State == "FLORIDA"  
4 | count
```

The following image shows a schematic representation of data being piped through this query:



# What is a let and set statement?

Use the `let` statement to set a variable name equal to an expression or a function, or to create [views](#).

Example:

## Kusto

```
1 .set MyTable <|
2   let text="Hello, World!";
3   print str=text
```

The `set` statement is used to set a query option for the duration of the query.

Example:

## Kusto

```
1 set querytrace;
2 StormEvents
3 | take 100
```

# SQL to KQL cheat sheet

- The primary language to interact with ADX is KQL (Kusto Query Language).
- You can use Kusto to translate SQL queries to KQL.
- Send an SQL query to Kusto, prefixing it with the verb '**EXPLAIN**'.

Example:

## Kusto

```
1 EXPLAIN  
2 SELECT COUNT_BIG(*) as C FROM StormEvents
```

## Query

```
StormEvents  
| summarize C=count()  
| project C
```

# Common KQL for IoT Analytics

- Function
- JSON
- Summarize
- Make-series
- Render
- Series\_fill\_linear
- Series\_decompose\_forecast
- Series\_decompose\_anomalies
- Materialized-view
- Arg-max
- External-table
- Invoke
- Project-away
- Evaluate
- Join

## Kusto (sample & function)

```
1 // cluster('igniteadxsource.eastus2').database('Occupancy')
2
3 StageIoTRawData
4 | sample 10
5
6 .create-or-alter function ExtractThermostatData {
7     StageIoTRawData
8         | where EventType has 'Thermostat' and Body has 'temp'
9         | project
10        EnqueuedTimeUTC,
11        SubEventType,
12        DeviceID,
13        BatteryLevel = tolong(Body.['BatteryLevel']),
14        Temp = toreal(Body.['temp']),
15        Humidity = toreal(Body.['humidity'])
16    }
17
18 Thermostats
19 | sample 10
```

## Kusto (sum, series & render)

```
21 Thermostats
22 | count
23
24 //What is the average temp every 1 min
25 Thermostats
26 | where EnqueuedTimeUTC > ago(7d)
27 | where DeviceId =='637086755190714287'
28 | summarize avg(Temp) by bin(EnqueuedTimeUTC,1m)
29 | render timechart
30
31
32 //Is there any missing data?
33 //make-series
34 //Create series of specified aggregated values along specified axis.
35 Thermostats
36 | where EnqueuedTimeUTC > ago(6h)
37 | where DeviceId =='637086755190714287'
38 | make-series AvgTemp=avg(Temp) on EnqueuedTimeUTC from ago(6h) to now() step 1m
39 | render timechart
```

## Kusto (series fill)

```
45 //How can I fill the missing values?  
46 //series_fill_linear()  
47 //Performs linear interpolation of missing values in a series.  
48 Thermostats  
49 | where EnqueuedTimeUTC > ago(6h)  
50 | where DeviceId == '637086755190714287'  
51 | make-series AvgTemp=avg(Temp) default=real(null) on EnqueuedTimeUTC  
52 from ago(6h) to now() step 1m  
53 | extend NoGapsTemp=series_fill_linear(AvgTemp)  
54 | project EnqueuedTimeUTC, NoGapsTemp  
55 | render timechart
```

## Kusto (forecast)

```
58 //What will be the temperature for the next one hour?  
59 Thermostats  
60 | where EnqueuedTimeUTC > ago(2d)  
61 | where DeviceId =='637086755190714287'  
62 | make-series AvgTemp=avg(Temp) default=null on EnqueuedTimeUTC  
63 from ago(2d) to now()+15m step 1m  
64 | extend NoGapsTemp=series_fill_linear(AvgTemp)  
65 | project EnqueuedTimeUTC, NoGapsTemp  
66 | extend forecast = series_decompose_forecast(NoGapsTemp, 15m)  
67 | render timechart with(title='Forecasting the next 15min by Time Series Decomposition')
```

## Kusto (anomalies)

```
69 //Are there any anomalies for this device?  
70 Thermostats  
71 | where EnqueuedTimeUTC > ago(3d)  
72 | where DeviceId =='637086755190714287'  
73 | make-series AvgTemp=avg(Temp) default=null on EnqueuedTimeUTC  
74 from ago(3d) to now() step 1m  
75 | extend NoGapsTemp=series_fill_linear(AvgTemp)  
76 | project EnqueuedTimeUTC, NoGapsTemp  
77 | extend anomalies = series_decompose_anomalies(NoGapsTemp,1)  
78 | render anomalychart with(anomalycolumns=anomalies)
```

## Kusto (expand)

```
80 //What the anomalies I should focus on across all devices?  
81 Thermostats  
82 | where EnqueuedTimeUTC > ago(3d)  
83 | make-series AvgTemp=avg(Temp) default=null on EnqueuedTimeUTC  
84 from ago(3d) to now() step 1m by DeviceId  
85 | extend NoGapsTemp=series_fill_linear(AvgTemp)  
86 | project EnqueuedTimeUTC, DeviceId, NoGapsTemp  
87 | extend anomalies = series_decompose_anomalies(NoGapsTemp,1)  
88 | mv-expand EnqueuedTimeUTC, anomalies, NoGapsTemp  
89 | where anomalies == 1
```

# Materialized Views

- Expose an aggregation query over a source table, or over [another materialized view](#).
- Always return an up-to-date result of the aggregation query (always fresh).
- Are more performant than running the aggregation directly over the source table.

Example:

## Kusto

```
1 .create materialized-view with (backfill=true, autoUpdateSchema=true )  
2 ArgMax on table T  
3 {  
4     T | summarize arg_max(Timestamp, *) by User  
5 }
```

**Note:** Materialized views have some [limitations](#), and don't work well for all scenarios. Review the [performance considerations](#) before working with the feature.

# Materialized Views for IoT

```
90 .show materialized-views
91
92
93 //Hourly_Average_Mview
94 Thermostats
95 | summarize avg_Battery_Level=avg(BatteryLevel), avg_Temp=avg(Temp),
96 avg_Humidity=avg(Humidity) by DeviceId,
97 bin(EnqueuedTimeUTC,1h)
98
100
101 //Current_Mview
102 Thermostats
103 | summarize (curr_Event_Time, curr_Battery_Level, curr_Temp, curr_Humidity) =
104 arg_max(EnqueuedTimeUTC, BatteryLevel, Temp, Humidity)
105 by DeviceId
```

# Con't

```
107 //Materialized views
108 Hourly_Average_Mview
109 | where EnqueuedTimeUTC > ago(1h)
110 | take 1000
111
112
113 Current_Mview
```

# External Tables

- An external table is a Kusto schema entity that references data stored outside the database.
- Similar to [tables](#), an external table has a well-defined schema (an ordered list of column name and data type pairs).

Azure Storage example:

## Kusto

```
1 .create external table ExternalTable (x:long, s:string)
2 kind=storage
3 dataformat=csv
4 (
5     h@'https://storageaccount.blob.core.windows.net/container1;secretKey'
6 )
```

Ref: [docs.microsoft.com/azure/data-explorer/kusto/query/schema-entities/externaltables](https://docs.microsoft.com/azure/data-explorer/kusto/query/schema-entities/externaltables)

## Con't

```
107 // .show external tables
108
109 external_table("ext_Thermostats")
110 | where EnqueuedTimeUTC between (datetime('03-01-2020 11:00 am') ..
111 datetime('03-01-2020 01:00 pm')) and DeviceId == '637086754472373714'
112 | project EnqueuedTimeUTC, BatteryLevel, Temp, Humidity
```

# Azure Digital Twin (ADT)

- Query the physical world (model representation) stored in ADT from ADX via a function.
- Join the ADT model to the device telemetry stored in ADX for a logical real-world output.

## Kusto

```
1 .show function GetDevicesbyStore
2
3 .create-or-alter function with (folder = "Analytics/IoT", skipvalidation = "true")
4 GetDevicesbyStore(Office:string) {
5 let ADTendpoint = "https://digitaltwinpm10774.api.eus.digitaltwins.azure.net";
6 let ADTquery = strcat("SELECT T.$dtId as Office, F.$dtId as Floor, D.$dtId as
7 DeviceId FROM DIGITALTWINS T JOIN F RELATED T.officecontainsfloors JOIN D
8 RELATED F.floorcontainsdevices where T.$dtId='", Office, "'");
9 evaluate azure_digital_twins_query_request(ADTendpoint, ADTquery)
10 | project Office=toString(Office), Floor=toString(Floor),
11 DeviceId=toString(DeviceId)
12 }
```

# Con't

## Kusto

```
13 //Lets get the devices in the Dallas office
14 GetDevicesbyStore('Dallas')
15
16 //Now lets join that with ADX data and chart the results. We combine the floor
17 from ADT with the DeviceId to ad context
18 GetDevicesbyStore('Dallas')
19 | join kind=inner (Thermostats
20     | where EnqueuedTimeUTC >= ago(1h)
21     | summarize Temp=avg(Temp) by DeviceId, AggTime=bin(EnqueuedTimeUTC, 1m)
22 ) on DeviceId
23 | extend DeviceId=strcat(Floor, '-', DeviceId)
24 | project todouble(Temp), AggTime, DeviceId
25 | render timechart with (ycolumns = Temp, series=DeviceId)
```

# Join with shuffle to improve performance

It is better to use when the `shuffle` key (join key) has a high cardinality and the regular operator hits query limits. For example:

- Left-side has 15M records, cardinality of the join key is ~14M.
- Right-side has 150M records, cardinality of the join key is 10M
- **7x faster** with shuffle and **5x less memory**.
- Shuffle partitions data such that number of partitions is equal to the number of cluster nodes. If you observe query timeouts or query fails to complete, you can increase the partition count via **hint.num\_partitions = x**. Please increase the number of partitions gradually as too large number might lead to degraded performance.

## Kusto

```
1 customer
2 | join hint.strategy = shuffle
3 orders
4 on $left.c_custkey == $right.o_custkey
5 | summarize sum(c_acctbal) by c_nationkey
```

# KQL Query Best Practices: Make your query run faster

Action	Use	Don't use	Notes
String operators	User the <code>has</code> operator	Don't use <code>contains</code>	When looking for full tokens, <code>has</code> works better, since it doesn't look for substrings
Case-sensitive	Use <code>==</code> , <code>in</code> , <code>contains_cs</code>	Don't use <code>=~</code> , <code>in~</code>	Use case-sensitive operators when possible
Searching text	Look in a specific column	Don't use <code>*</code>	Does a full text search across all columns
New queries	Use <code>limit [small number]</code> or <code>count</code> at the end		Running unbound queries over unknown data sets may yield GBs of results to be returned to the client, resulting in a slow response and a busy cluster.

# KQL Query Best Practices: Make your query run faster

Action	Use	Don't use	Notes
Filtering	A table column	Don't filter on a calculated column.	
summarize	<code>hint.strategy=shuffle</code> when the group by keys are high cardinality		High cardinality is ideally above 1 million
join	Table with fewer rows first (left-side in query)		
Join across clusters	Run the query on the cluster where most of the data resides.	Don't filter on a calculated column	
Left is small and Right is large (joins)	Use <code>hint.strategy=broadcast</code>		Small means <= 100,000 records

See more: [aka.ms/adx/query.bp](http://aka.ms/adx/query.bp)

# Knowledge check – Kusto Query Language

**What is the input and output data format of each operator in a Kusto query?**

- a) String
- b) Tabular**
- c) JSON

**Which operator will return a specific number of arbitrary rows of data?**

- a) take**
- b) between
- c) contain

**Which operator can be used to insert new computed columns?**

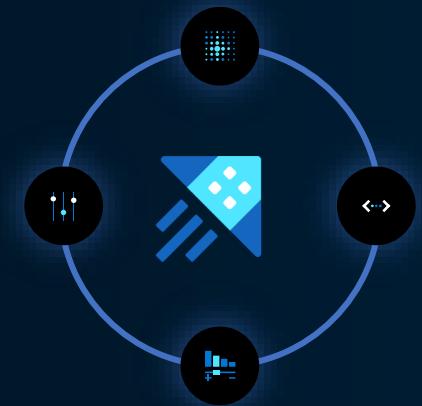
- a) print
- b) render
- c) project**

**Which query best answers the question: Which types of storms caused the most property damage in the state of Texas?**

- a)** StormEvents | where DamageProperty > 0 | where State == "TEXAS" | project State, EventType, DamageProperty | sort by DamageProperty
- b) StormEvents | where DamageCrops > 0 | where State == "TEXAS" | project State, EventType, DamageCrops | sort by DamageCrops
- c) StormEvents | where DamageProperty > 0 | where State == "FLORIDA" | where EventType contains "thunder" | project StartTime, EventType, DamageProperty

# ML & Timeseries

## Module 6 Objectives



After completing this Learning, you will be able to understand:

1. Time series analysis
2. Anomaly detection
3. Forecasting
4. Built-in machine learning
5. Python plugin
6. R-language plugin

# Time series analysis in ADX

- In ADX, data can be analyzed for various insights such as service health, physical production processes, and usage trends.
- Analysis is done on time series of selected metrics to find a deviation in the pattern compared to its typical baseline pattern.
- ADX contains native support for creation, manipulation, and analysis of multiple time series.
- Analyze thousands of time series in seconds, enabling near real-time monitoring solutions and workflows.

# Time series creation

- Create a large set of regular time series simply and intuitively using the `make-series` operator, and fill-in missing values as needed.
- The first step in time series analysis is to partition and transform the original telemetry table to a set of time series.
- The table usually contains a timestamp column, contextual dimensions, and optional metrics.

[[Click to run query](#)]

**Kusto**

```
1 demo_make_series1 | take 10
```

# Time series creation

- Since there are no metrics, we can only build a set of time series representing the **traffic count**, by OS.

[\[Click to run query\]](#)

## Kusto

```
1 let min_t = toscalar(demo_make_series1 | summarize min(TimeStamp));
2 let max_t = toscalar(demo_make_series1 | summarize max(TimeStamp));
3 demo_make_series1
4 | make-series num=count() default=0 on TimeStamp in range(min_t, max_t, 1h)
5 by OsVer
6 | render timechart
```

# Filtering

Filtering is a common practice in signal processing and useful for time series processing tasks (for example, smooth a noisy signal, change detection).

[\[Click to run query\]](#)

## Kusto

```
1 let min_t = toscalar(demo_make_series1 | summarize min(TimeStamp));
2 let max_t = toscalar(demo_make_series1 | summarize max(TimeStamp));
3 demo_make_series1
4 | make-series num=count() default=0 on TimeStamp in range(min_t, max_t, 1h)
5 by OsVer
6 | extend ma_num=series_fir(num, repeat(1, 5), true, true)
7 | render timechart
```

# Regression analysis

ADX supports segmented linear regression analysis to estimate the trend of the time series.

- Use [`series fit line\(\)`](#) to fit the best line to a time series for general trend detection.
- Use [`series fit 2lines\(\)`](#) to detect trend changes, relative to the baseline, that are useful in monitoring scenarios.

[[Click to run query](#)]

## Kusto

```
1  demo_series2
2  | extend series_fit_2lines(y), series_fit_line(y)
3  | render linechart with(xcolumn=x)
```

# Seasonality detection

- Many metrics follow seasonal (periodic) patterns.
- User traffic of cloud services usually contains daily and weekly patterns that are highest around the middle of the business day and lowest at night and over the weekend.
- Measurements such as temperature, pressure, or humidity may also show seasonal behavior.

[[Click to run query](#)]

## Kusto

```
1  demo_series3
2  | render timechart;
3
4  demo_series3
5  | project (periods, scores) = series_periods_detect(num, 0., 14d/2h, 2)
6  | mv-expand periods, scores
7  | extend days=2h*todouble(periods)/1d
```

# Element-wise functions

- Arithmetic and logical operations can be done on a time series.
- Using [series subtract\(\)](#) we can calculate a residual time series, that is, the difference between original raw metric and a smoothed one, and look for anomalies in the residual signal:

[\[Click to run query\]](#)

## Kusto

```
1 let min_t = toscalar(demo_make_series1 | summarize min(TimeStamp));
2 let max_t = toscalar(demo_make_series1 | summarize max(TimeStamp));
3 demo_make_series1
4 | make-series num=count() default=0 on TimeStamp in range(min_t, max_t, 1h) by
5 OsVer
6 | extend ma_num=series_fir(num, repeat(1, 5), true, true)
7 | extend residual_num=series_subtract(num, ma_num)
8 | where OsVer == "Windows 10"
9 | render timechart
```

# Time series at scale

- These functions can run at scale on thousands of time series in seconds for anomaly detection.
- These advanced capabilities combined with Azure Data Explorer fast performance supply a unique and powerful solution for time series analysis.

[[Click to run query](#)]

## Kusto

```
1 let min_t = toscalar(demo_many_series1 | summarize min(TIMESTAMP));
2 let max_t = toscalar(demo_many_series1 | summarize max(TIMESTAMP));
3 demo_many_series1
4 | make-series reads=avg(DataRead) on TIMESTAMP in range(min_t, max_t, 1h) by Loc,
5 Op, DB
6 | extend (rsquare, slope) = series_fit_line(reads)
7 | top 2 by slope asc
8 | project Loc, Op, DB, slope
```

# Anomaly detection and forecasting in ADX

- Contains **native** support for creation, manipulation, and analysis of multiple time series.
- The applicable time series functions are based on a robust well-known decomposition model, where each original time series is decomposed into seasonal, trend, and residual components.
- **Anomalies** are detected by outliers on the residual component, while **forecasting** is done by extrapolating the seasonal and trend components.
- Significantly enhances the basic **decomposition** model by automatic **seasonality** detection, robust **outlier** analysis, and **vectorized** implementation to process thousands of time series in seconds.

# Decomposition model

- Is applied to time series of metrics expected to manifest periodic and trend behavior, such as **service traffic**, component **heartbeats**, and **IoT** periodic measurements to forecast future metric values and detect anomalous ones.
- The **assumption** of this regression process is that **other than the previously known** seasonal and trend behavior, the time **series is randomly distributed**.
- Can detect anomalous values based on outlier analysis using only the residual portion.
- To create a decomposition model, use the function [series decompose\(\)](#). It takes a set of time series and automatically decomposes each time series to its seasonal, trend, residual, and baseline components.

# Decomposition model

For example, you can decompose traffic of an internal web service by using the following query:

[[Click to run query](#)]

## Kusto

```
1 let min_t = datetime(2017-01-05);
2 let max_t = datetime(2017-02-03 22:00);
3 let dt = 2h;
4 demo_make_series2
5 | make-series num=avg(num) on TimeStamp from min_t to max_t step dt by sid
6 | where sid == 'TS1'
7 | extend (baseline, seasonal, trend, residual) = series_decompose(num, -1,
8 | 'linefit')
9 | render timechart with(title='Web app. traffic of a month, decomposition',
10 ysplit=panels)
```

# Anomaly detection

- The function [`series\_decompose\_anomalies\(\)`](#) finds anomalous points on a set of time series.
- This function calls `series_decompose()` to build the decomposition model and then runs [`series\_outliers\(\)`](#) on the residual component.
- `series_outliers()` calculates anomaly scores for each point of the residual component using Tukey's fence test. Anomaly scores above 1.5 or below -1.5 indicate a mild anomaly rise or decline.
- Anomaly scores above 1.5 or below -1.5 indicate a **mild** anomaly rise or decline. Scores above 3.0 or below -3.0 indicate a **strong** anomaly.

# Anomaly detection

The following query allows you to detect anomalies in internal web service traffic:

[\[Click to run query\]](#)

## Kusto

```
1 let min_t = datetime(2017-01-05);
2 let max_t = datetime(2017-02-03 22:00);
3 let dt = 2h;
4 demo_make_series2
5 | make-series num=avg(num) on TimeStamp from min_t to max_t step dt by sid
6 | where sid == 'TS1'
7 | extend (anomalies, score, baseline) = series_decompose_anomalies(num, 1.5, -1,
8 | 'linefit')
9 | render anomalychart with(anomalycolumns=anomalies, title='Web app. traffic of a
10 month, anomalies')
```

# Forecasting

- The function [`series\_decompose\_forecast\(\)`](#) predicts future values of a set of time series.
- This function calls `series_decompose()` to build the decomposition model and then, for each time series, **extrapolates the baseline component into the future**.

[\[Click to run query\]](#)

## Kusto

```
1 let min_t = datetime(2017-01-05);
2 let max_t = datetime(2017-02-03 22:00);
3 let dt = 2h;
4 let horizon = 7d;
5 demo_make_series2
6 | make-series num=avg(num) on TimeStamp from min_t to max_t+horizon step dt
7 by sid
8 | where sid == 'TS1'
9 | extend forecast = series_decompose_forecast(num, toint(horizon/dt))
10 | render timechart with(title='Traffic of a month + forecasting next week')
```

# Scalability

- ADX allows for fast performance, which is critical for effective anomaly detection & forecasting over thousands of counters in near real-time.

[\[Click to run query\]](#)

## Kusto

```
1 let min_t = datetime(2017-01-05);
2 let max_t = datetime(2017-02-03 22:00);
3 let dt = 2h;
4 let horizon = 7d;
5 demo_make_series2
6 | make-series num=avg(num) on TimeStamp from min_t to max_t+horizon step dt
7 by sid
8 | extend offset=case(sid=='TS3', 4000000, sid=='TS2', 2000000, 0)
9 | extend num=series_add(num, offset)
10 | extend forecast = series_decompose_forecast(num, toint(horizon/dt))
11 | render timechart with(title='Traffic of a month + forecasting next week')
```

# Summary

- Each original time series is decomposed into seasonal, trend and residual components for detecting anomalies and/or forecasting.
- These functionalities can be used for near real-time monitoring scenarios, such as **fault detection, predictive maintenance, demand and load** forecasting.

# Machine Learning capability in ADX

- Since these diagnosis scenarios are common in Azure Data Explorer, machine learning **plugins** are available to **make the diagnosis phase easier** and shorten the duration of an RCA (Root Cause Analysis).
- ADX has three ML plugins: [autocluster](#), [basket](#), and [diffpatterns](#). All plugins implement clustering algorithms.
- The [autocluster](#) and [basket](#) plugins cluster a **single** record set.
- The [diffpatterns](#) plugin clusters the differences between **two** record sets.

# Clustering a single record set

- ADX allows for fast performance, which is critical for effective anomaly detection & forecasting over thousands of counters in near real-time.

[\[Click to run query\]](#)

## Kusto

```
1 let min_t = toscalar(demo_clustering1 | summarize min(PreciseTimeStamp));
2 let max_t = toscalar(demo_clustering1 | summarize max(PreciseTimeStamp));
3 demo_clustering1
4 | make-series num=count() on PreciseTimeStamp from min_t to max_t step 10m
5 | render timechart with(title="Service exceptions over a week, 10 minutes
6 resolution")
```

# Clustering a single record set

- The second spike in the example occurs on Tuesday afternoon.
- The following query is used to further diagnose and verify whether it's a sharp spike.
- The query redraws the chart around the spike in a higher resolution of eight hours in one-minute bins.

[\[Click to run query\]](#)

## Kusto

```
1 let min_t=datetime(2016-08-23 11:00);
2 demo_clustering1
3 | make-series num=count() on PreciseTimeStamp from min_t to min_t+8h step 1m
4 | render timechart with(title="Zoom on the 2nd spike, 1 minute resolution")
```

# Clustering a single record set

- In the example, you'll see a narrow two-minute spike from 15:00 to 15:02.

The following query, count the exceptions in this two-minute window:

[\[Click to run query\]](#)

## Kusto

```
1 let min_peak_t=datetime(2016-08-23 15:00);
2 let max_peak_t=datetime(2016-08-23 15:02);
3 demo_clustering1 | where PreciseTimeStamp between(min_peak_t..max_peak_t)
4 | count
5
6 //take a sample 20 of 972:
7 let min_peak_t=datetime(2016-08-23 15:00);
8 let max_peak_t=datetime(2016-08-23 15:02);
9 demo_clustering1 | where PreciseTimeStamp between(min_peak_t..max_peak_t)
10 | take 20
```

# Use [autocluster\(\)](#) for single record set clustering

- Instantly extract a short list of common segments.

[[Click to run query](#)]

## Kusto

```
1 let min_peak_t=datetime(2016-08-23 15:00);
2 let max_peak_t=datetime(2016-08-23 15:02);
3 demo_clustering1 | where PreciseTimeStamp between(min_peak_t..max_peak_t)
4 | evaluate autocluster()
```

- Autocluster uses a proprietary algorithm for mining multiple dimensions and extracting **interesting** segments.
- "Interesting" means that each segment has significant coverage of both the records set and the features set.

# Use basket() for single record set clustering

[[Click to run query](#)]

**Kusto**

```
1 let min_peak_t=datetime(2016-08-23 15:00);
2 let max_peak_t=datetime(2016-08-23 15:02);
3 demo_clustering1 | where PreciseTimeStamp between(min_peak_t..max_peak_t)
4 | evaluate basket()
```

- Basket implements the "Apriori" algorithm for item set mining.
- It extracts **all segments** whose coverage of the record set is **above a threshold** (default 5%).
- Both plugins limitation is that they cluster a single record set in an unsupervised manner with no labels.

# Clustering the difference between two records sets

- Diffpatterns takes **two record sets** and extracts the main segments that are different.
- One set is analyzed by autocluster and basket. The other set contains the reference record set, the baseline.

[\[Click to run query\]](#)

## Kusto

```
1 let min_peak_t=datetime(2016-08-23 15:00);
2 let max_peak_t=datetime(2016-08-23 15:02);
3 let min_baseline_t=datetime(2016-08-23 14:50);
4 let max_baseline_t=datetime(2016-08-23 14:58);
5 let splitime=(max_baseline_t+min_peak_t)/2.0;
6 demo_clustering1
7 | where (PreciseTimeStamp between(min_baseline_t..max_baseline_t)) or
8     (PreciseTimeStamp between(min_peak_t..max_peak_t))
9 | extend AB=iff(PreciseTimeStamp > splitime, 'Anomaly', 'Baseline')
10 | evaluate diffpatterns(AB, 'Anomaly', 'Baseline')
```

# Clustering the difference between two records sets

- We can see that the spike was because of exceptions from this specific segment, discovered by using the `diffpatterns` plugin.

[\[Click to run query\]](#)

## Kusto

```
1 let min_t = toscalar(demo_clustering1 | summarize min(PreciseTimeStamp));
2 let max_t = toscalar(demo_clustering1 | summarize max(PreciseTimeStamp));
3 demo_clustering1
4 | extend seg = iff(Region == "eau" and ScaleUnit == "su7" and DeploymentId ==
5 "b5d1d4df547d4a04ac15885617edba57"
6 and ServiceHost == "e7f60c5d-4944-42b3-922a-92e98a8e7dec", "Problem", "Normal")
7 | make-series num=count() on PreciseTimeStamp from min_t to max_t step 10m by seg
8 | render timechart
```

# Summary

- The `autocluster` and `basket` implement an unsupervised learning algorithm and are easy to use.
- Diffpatterns implements a supervised learning algorithm and, although more complex, it's more powerful for extracting `diffpatterns` segments for Root Cause Analysis.
- These plugins are used interactively in **ad-hoc** scenarios and in **automatic** near real-time monitoring services.
- In ADX, **time series anomaly detection is followed by a diagnosis process**, that is highly optimized to meet necessary performance standards.

# ADX Language plugin: Python

- Runs a user-defined function (UDF) using a Python script.
- The Python script gets tabular data as its input, and produces tabular output.
- The plugin's runtime is hosted in [sandboxes](#), running on the cluster's nodes.
- The plugin is **disabled** by default. Enable or disable the plugin in the Azure portal, in your cluster's [Configuration tab](#).
- The Python sandbox image is based on *Anaconda 5.2.0* distribution with the *Python 3.8* engine. See the list of [Anaconda packages](#).
- The Python image also contains common ML packages: tensorflow, keras, torch, hdbSCAN, xgboost, and other useful packages.
- The plugin imports *numpy* (as np) & *pandas* (as pd) by default. You can import other modules as needed.

# Python Examples

## Kusto

```
1 range x from 1 to 360 step 1
2 | evaluate python(
3 typeof(*, fx:double), //Output schema: append a new fx column to original table
4 ```
5 result = df
6 n = df.shape[0]
7 g = kargs["gain"]
8 f = kargs["cycles"]
9 result["fx"] = g * np.sin(df["x"]/n*2*np.pi*f)
10 ```
11 , pack('gain', 100, 'cycles', 4) //dictionary of parameters
12 ) | render linechart
```

# Python plugin

- Runs a user-defined function (UDF) using a Python script.
- The Python script gets tabular data as its input and produces tabular output.
- The plugin's runtime is hosted in [sandboxes](#), running on the cluster's nodes.
- The plugin is **disabled by default**.
- Contains common ML packages: [tensorflow](#), [keras](#), [torch](#), [hdbscan](#), [xgboost](#), and others.
- The plugin imports numpy (as np) & pandas (as pd) by default.

The screenshot shows the 'kustodocs | Configurations' page in the Azure Data Explorer interface. The left sidebar lists various cluster management options like Overview, Activity log, and Permissions. The main area is titled 'Configurations' and contains settings for 'Streaming ingestion', 'Python language extension', and 'R language extension'. The 'Python language extension' and 'R language extension' checkboxes are both set to 'On', indicated by a blue dot in the radio button. Below each setting, there is a warning message: 'PAY ATTENTION: \* Enabling Python plugin process can take a few minutes. During that time your cluster will be suspended.' and 'PAY ATTENTION: \* Enabling R plugin process can take a few minutes. During that time your cluster will be suspended.' respectively. The 'Save' button at the top right is also highlighted with a red box.

# Python Unsupervised Learning

```
1 //Create a custom UDF to run K-Means clustering using Python plugin
2 .create-or-alter function with (folder = "Python") kmeans_sf_OccupDetc(tbl:(*),
3 k:int, features:dynamic, cluster_col:string) {
4     let kwargs=pack('k', k, 'features', features, 'cluster_col', cluster_col);
5     let code='from sklearn.cluster import KMeans\n'
6         '\n'
7         'k = kargs["k"]\n'
8         'features = kargs["features"]\n'
9         'cluster_col = kargs["cluster_col"]\n'
10        '\n'
11        'km = KMeans(n_clusters=k)\n'
12        'df1 = df[features]\n'
13        'km.fit(df1)\n'
14        'result = df\n'
15        'result[cluster_col] = km.labels_';
16
17     | evaluate python(typeof(*), code, kwargs)
18 }
```

# Con't

```
20 // Invoke the custom UDF
21 Thermostats
22 | where EnqueuedTimeUTC > ago(7d)
23 | extend cluster_id=double(null)
24 | project EnqueuedTimeUTC, DeviceId, Temp, Humidity, cluster_id
25 | invoke kmeans_sf_OccupDetc(3, pack_array("Temp", "Humidity"), "cluster_id")
26 | sample 10
```

# Python Supervised Model

```
1 // Custom UDF to score based on pre-trained model
2 .create-or-alter function with (folder = "Python", skipvalidation = "true")
3 classify_sf_OccupDetc(
4     samples:(*),
5     models_tbl:(name:string,timestamp:datetime,model:string),
6     model_name:string,
7     features_cols:dynamic,
8     pred_col:string)
9 {
10     let model_str = toscalar(
11         ML_Models
12         | where name == model_name
13         | top 1 by timestamp desc
14         | project model
15     );
16     let kwargs = pack('smodel', model_str, 'features_cols', features_cols,
17     'pred_col', pred_col);
```

# Con't

```
18 let code ='import pickle\n'
19 'import binascii\n'
20 '\n'
21 'smodel = kargs["smodel"]\n'
22 'features_cols = kargs["features_cols"]\n'
23 'pred_col = kargs["pred_col"]\n'
24 'bmodel = binascii.unhexlify(smodel)\n'
25 'clf1 = pickle.loads(bmodel)\n'
26 'df1 = df[features_cols]\n'
27 'predictions = clf1.predict(df1)\n'
28 '\n'
29 'result = df\n'
30 'result[pred_col] = pd.DataFrame(predictions, columns=[pred_col])'
31 ;
32 samples | evaluate python(typeof(*), code, kwargs)
33 }
```

## Con't

```
35 //Based on the Temp and Humidity - Is the room occupied?  
36 Thermostats  
37 | where EnqueuedTimeUTC > ago(15m)  
38 | extend pred_Occupancy=bool(0)  
39 | extend CO2=0, HumidityRatio=0  
40 | invoke classify_sf_OccupDetc(ML_Models, 'Occupancy', pack_array('Temp',  
41 'Humidity', 'BatteryLevel', 'CO2', 'HumidityRatio'), 'pred_Occupancy')  
42 | project-away SubEventType
```

# Externaldata operator

## Kusto

```
1 let script = externaldata(script:string)
2 [h'https://kustoscriptsamples.blob.core.windows.net/samples/python/sample_script.py']
3 with(format = raw);
4 range x from 1 to 360 step 1
5 | evaluate python(
6     typeof(*, fx:double),
7     toscalar(script),
8     pack('gain', 100, 'cycles', 4))
9 | render linechart
```

# Install packages for the Python plugin

- Create a blob container to host the packages, preferably in the same place as your cluster.
- Alter the cluster's [callout policy](#) to allow access to that location.

For example, to enable access to a blob located in <https://artifcatswestus.blob.core.windows.net/public>

## Kusto

```
1 .show cluster policy callout
2
3 .alter-merge cluster policy
4 callout @'[ { "CalloutType": "sandbox_artifacts", "CalloutUriRegex": "artifcatswestus\
5 \.blob\\.core\\.windows\\.net/public/", "CanCall": true } ]'
```

**Note:** This change requires [AllDatabasesAdmin](#) permissions.

# Example

Install the [Faker](#) package that generates fake data.

## Kusto

```
1 range ID from 1 to 3 step 1
2 | extend Name=''
3 | evaluate python(typeof(*), ````if 1:
4     from sandbox_utils import Zipackage
5     Zipackage.install("Faker.zip")
6     from faker import Faker
7     fake = Faker()
8     result = df
9     for i in range(df.shape[0]):
10        result.loc[i, "Name"] = fake.name()
11    ```,
12    external_artifacts=pack('faker.zip',
13    'https://artifactswestus.blob.core.windows.net/public/Faker-8.11.0.zip?sas'))
```

# ADX Language plugin: R (Preview)

- Runs a user-defined function (UDF) using an R script.
- The script gets tabular data as its input, and produces tabular output.
- The plugin's runtime is hosted in [sandboxes](#), running on the cluster's nodes.
- The plugin is **disabled** by default. Enable or disable the plugin in the Azure portal, in your cluster's [Configuration tab](#).
- The R sandbox image is based on *R 3.4.4 for Windows*, and includes packages from [Anaconda's R Essentials bundle](#).
- The R sandbox limits access to the network.
- The R code **can't dynamically install additional packages** that aren't included in the image. If you need specific packages, open a **New support request** in the Azure portal.

# R Examples

## Kusto

```
1 range x from 1 to 360 step 1
2 | evaluate r(
3 typeof(*, fx:double), // Output schema: append a new fx column to original table
4 'result <- df\n'        // The R decorated script
5 'n <- nrow(df)\n'
6 'g <- kargs$gain\n'
7 'f <- kargs$cycles\n'
8 'result$fx <- g * sin(df$x / n * 2 * pi * f)'
9 , pack('gain', 100, 'cycles', 4) // dictionary of parameters
10 )
11 | render linechart
```

# Performance Tips

- Use filters on the source data set using the Kusto Query Language, when possible.
- To make a calculation on a subset of the source columns, project only those columns before invoking the plugin.

For example:

## Kusto

```
1 .show operations
2 | where StartedOn > ago(1d)
3 | project d_seconds = Duration / 1s
4 | evaluate hint.distribution = per_node r(
5     typeof(*, d2:double),
6     'result <- df\n'
7     'result$d2 <- df$d_seconds\n'
8 )
9 | summarize avg = avg(d2)
```

# Usage Tips

- Use the [external data operator](#) to obtain the content of a script that you've stored in an external location, such as Azure blob storage or a public GitHub repository.

For example:

## Kusto

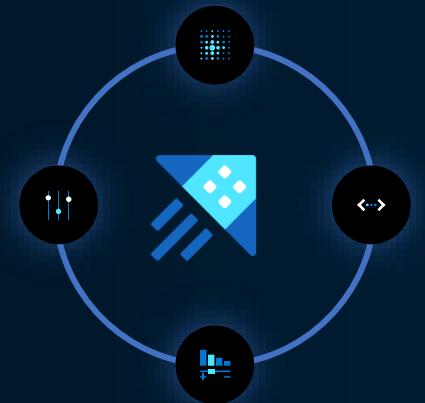
```
1 let script = externaldata(script:string)
2 [h'https://kustoscriptsamples.blob.core.windows.net/samples/R/sample_script.r']
3 with(format = raw);
4 range x from 1 to 360 step 1
5 | evaluate r(
6     typeof(*, fx:double),
7     toscalar(script),
8     pack('gain', 100, 'cycles', 4))
9 | render linechart
```

# Demo\*

- Python
- ML scoring
- External artifact
- R
- External data

# Visualize Data

## Module 7 Objectives



After completing this Learning, you will be able to understand:

1. How to visualize data
2. KQL render
3. Built-in Dashboards
4. Power BI

# ADX: How to visualize data

- Built-in Dashboards
- Azure Workbooks
- Native ADX connectors
  - Power BI
  - Grafana
  - Kibana (K2)
  - Redash
- ODBC/JDBC connectors
  - Power BI (ODBC)
  - Tableau (ODBC)
  - Qlik (ODBC)
  - Sisense (JDBC)

# Visualize Data: [Render](#) (KQL)

- |  |   |   |
|--|---|---|
| 1. <a href="#"><u>anomalychart</u></a> | 6. <a href="#"><u>ladderchart*</u></a>  | 11. <a href="#"><u>stackedareachart</u></a> |
| 2. <a href="#"><u>areachart</u></a>    | 7. <a href="#"><u>linechart</u></a>     | 12. <a href="#"><u>table</u></a>            |
| 3. <a href="#"><u>barchart</u></a>     | 8. <a href="#"><u>piechart</u></a>      | 13. <a href="#"><u>timechart</u></a>        |
| 4. <a href="#"><u>card</u></a>         | 9. <a href="#"><u>pivotchart*</u></a>   | 14. <a href="#"><u>timepivot*</u></a>       |
| 5. <a href="#"><u>columnchart</u></a>  | 10. <a href="#"><u>scatterchart</u></a> |   |

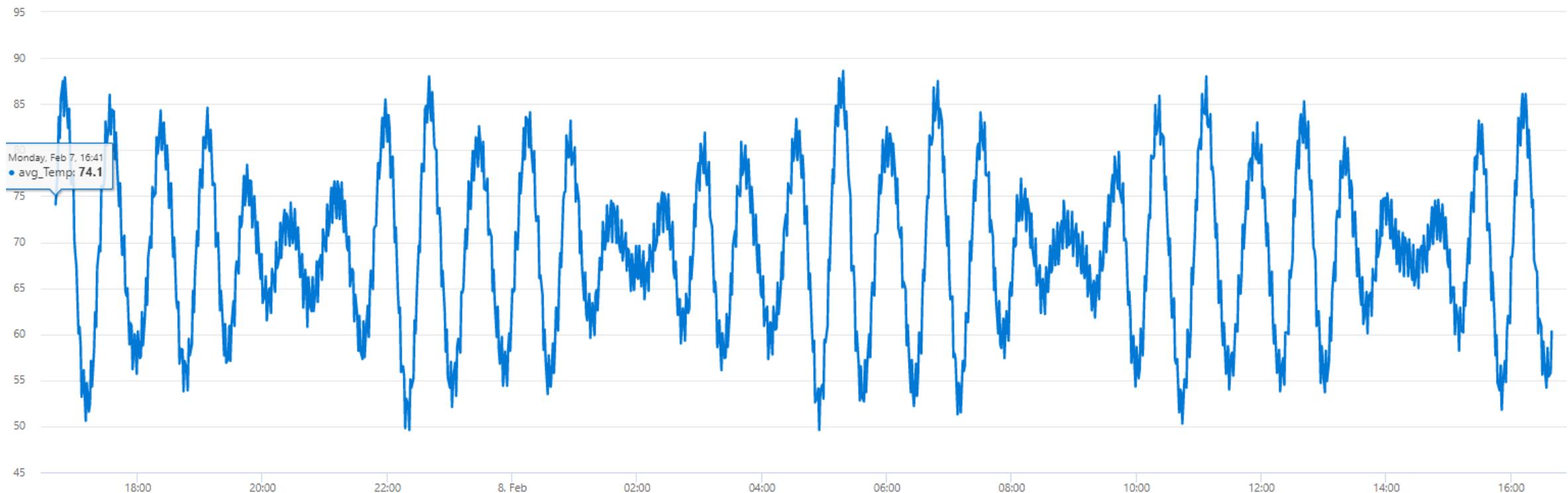
## Sample:

```
range x from 0.0 to 2*pi() step 0.01 | extend y=sin(x) | render linechart
```

# render operator example:

timechart

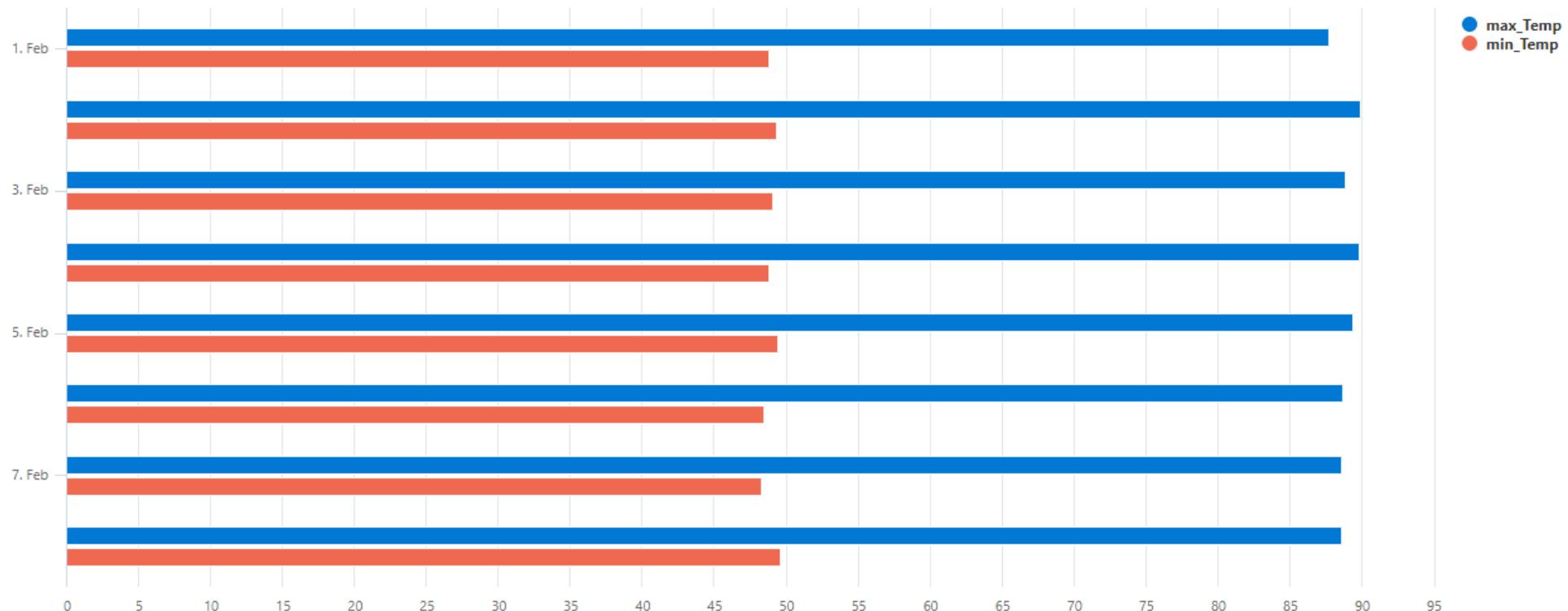
```
109 Thermostats  
110 | where EnqueuedTimeUTC >= ago(7d)  
111 | summarize avg(Temp) by bin(EnqueuedTimeUTC, 1m)  
112 | render timechart
```



# render operator example:

barchart

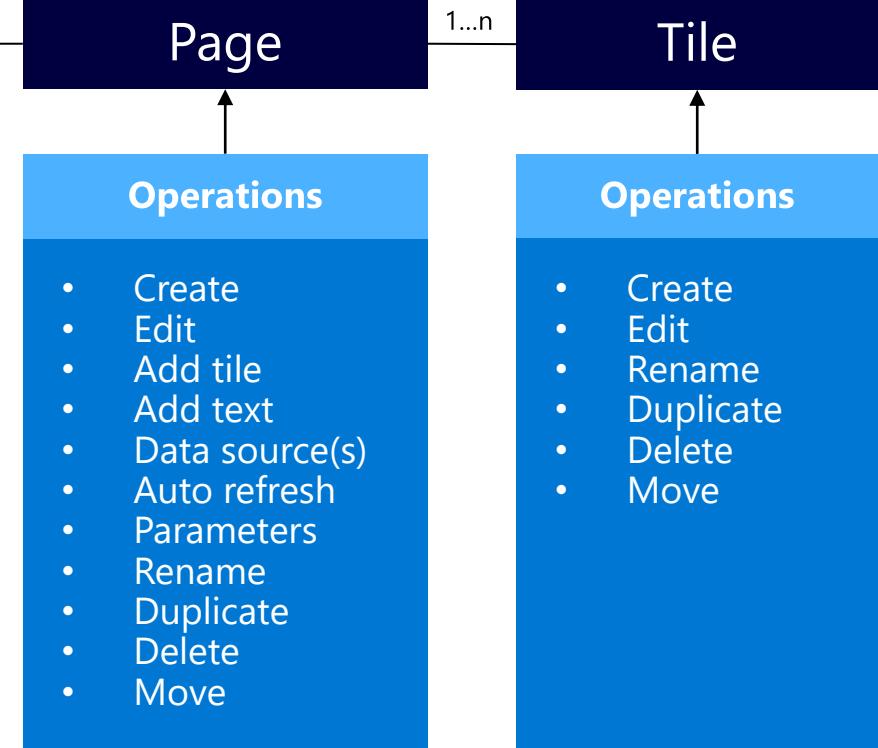
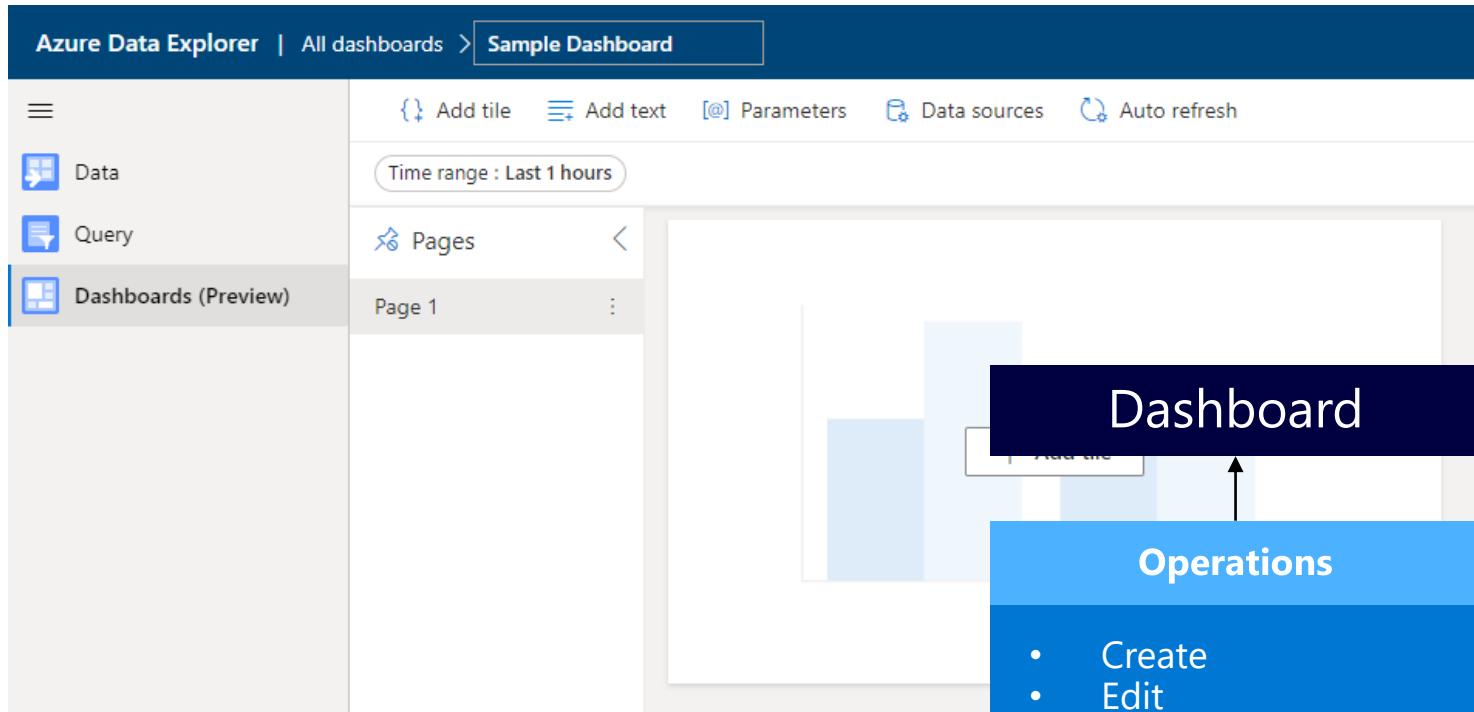
```
109 Thermostats  
110 | where EnqueuedTimeUTC >= ago(7d)  
111 | summarize min(Temp), max(Temp) by bin(EnqueuedTimeUTC,1d)  
112 | render barchart
```



# ADX Dashboards (built-in)



# Concepts



# Creating Dashboards

## 1. Write the query, render the chart and pin it to an existing or a new Dashboard

A screenshot of the Azure Data Explorer interface. On the left, a code editor displays a Kusto Query Language (KQL) script:

```
1 let ['startTime']=datetime('2022-02-01T16:41:32Z');
2 let ['endTime']=datetime('2022-02-08T16:41:32Z');
3 let ['Device_Id']='637085868243706792';
4 // Please enter your KQL query (Example):
5 // TableName | take 10
6 Thermostats
7 | where EnqueuedTimeUTC between (startTime .. endTime)
8 | where DeviceId == Device_Id
9 | summarize min(Temp), max(Temp) by bin(EnqueuedTimeUTC,1d)
10 | render barchart
```

To the right of the code editor is a context menu with several options:

- Link to clipboard
- Link, query to clipboard
- Link, query, results to clipboard
- Pin to dashboard** (this option is highlighted with a red box)
- Query to Power BI
- Open in Excel

## 2. Vanilla approach: Create a new Dashboard

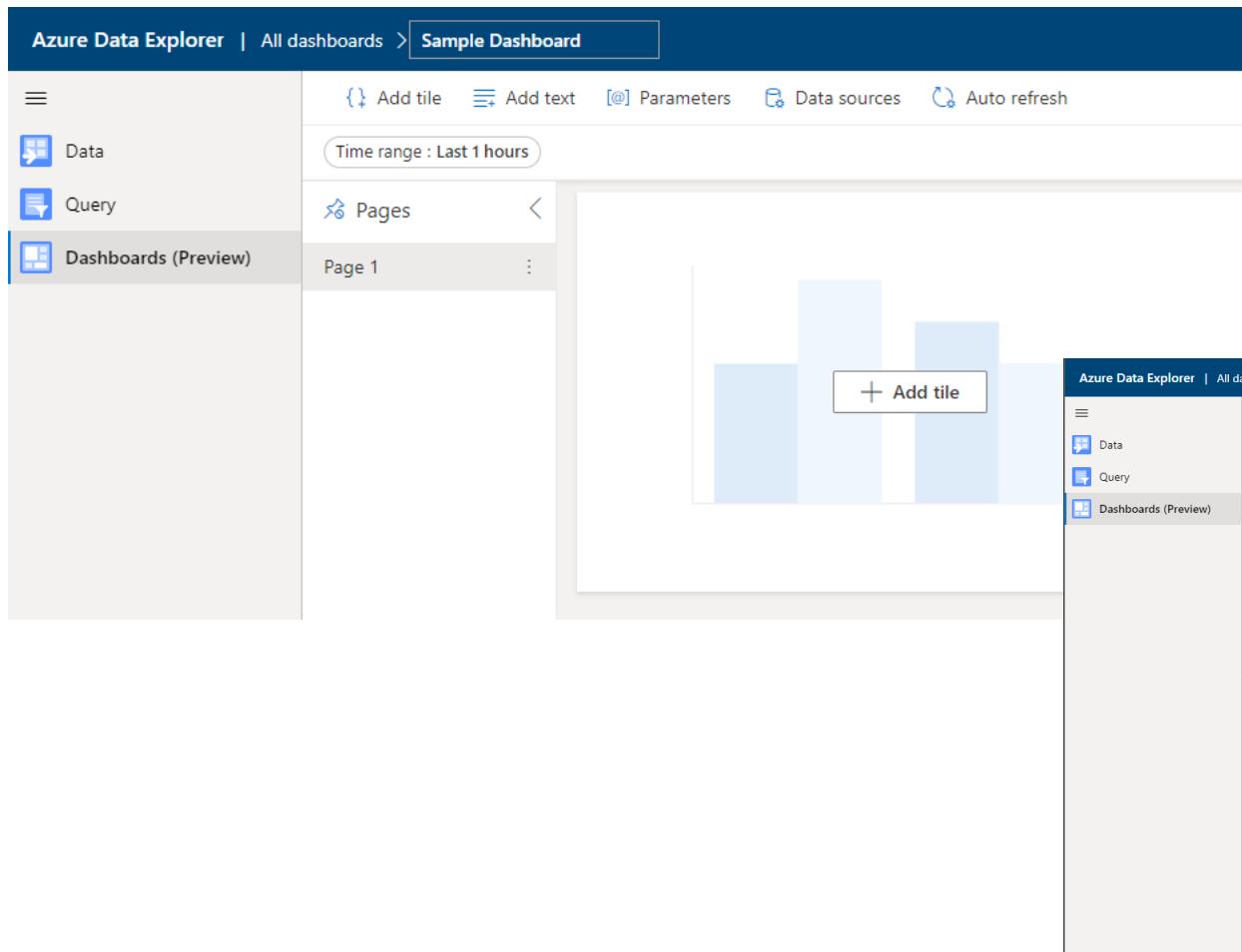
A screenshot of the Azure Data Explorer interface, specifically the 'Dashboards (Preview)' section. A 'New dashboard' dialog box is open in the foreground, prompting for a 'Dashboard name'. The input field contains 'SampleDashboard'.

The background shows a list of existing dashboards:

- All (15)
- Recent (10)
- Favorites (0)
- ADX Cluster Profile
- CDN Logs Demo
- GitHub Analytics
- Healthcare Dashboard
- IoT Anomaly Hunting
- IoT Demo 01
- IoT Thermostat Dashboard
- Kusto Partner

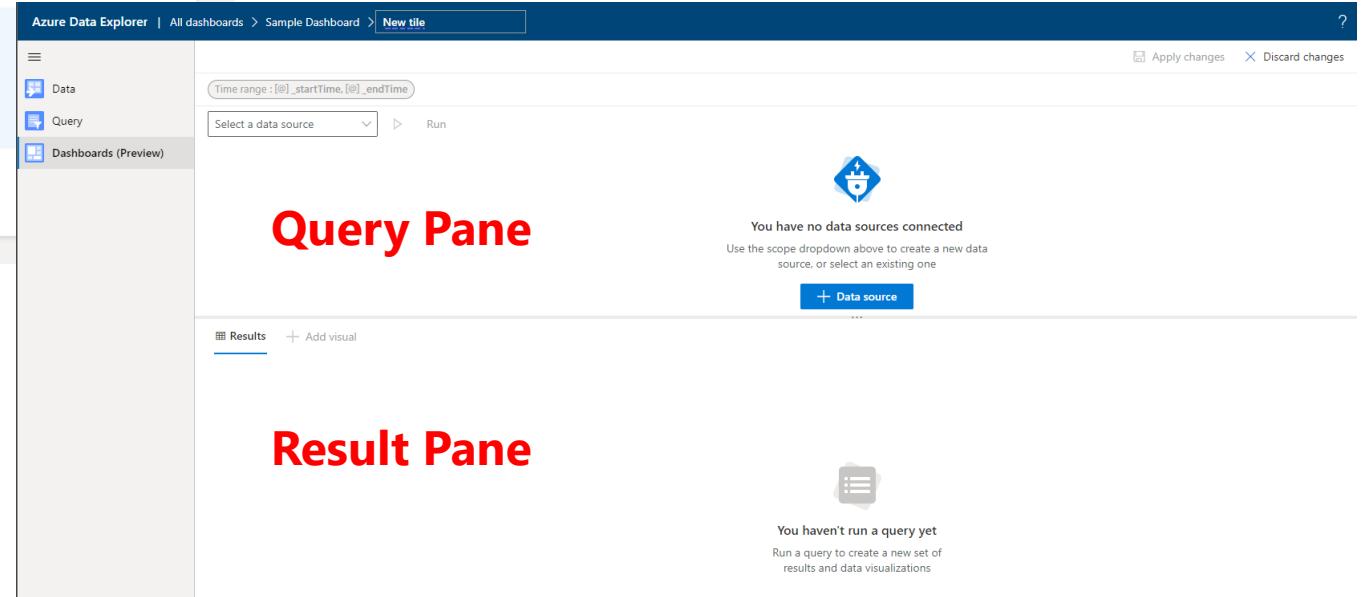
# Creating pages and tiles

1. 1<sup>st</sup> Page is automatically created. Add a new tile.



The screenshot shows the Azure Data Explorer interface for creating dashboards. On the left, there's a sidebar with options for Data, Query, and Dashboards (Preview). The main area is titled "Sample Dashboard" and shows "Page 1". A prominent "Add tile" button is located in the center of the dashboard area. The top navigation bar includes "Add tile", "Add text", "Parameters", "Data sources", and "Auto refresh". A time range selector shows "Last 1 hours".

2. On adding a new tile, familiar KQL editor experience is provided



The screenshot shows the KQL editor interface. It has two main panes: the "Query Pane" on the left and the "Result Pane" on the right. The Query Pane contains a message stating "You have no data sources connected" with a "Data source" button. The Result Pane contains a message stating "You haven't run a query yet" with a "Run a query" button. The top navigation bar for this pane includes "Apply changes" and "Discard changes".

# Adding data sources

1. Add data source. Add your cluster's URI. You can find it on the Overview page in Azure Portal

Create new data source

Data source name

Cluster URI \*

Enter cluster URI

Connect

Database \*

Select a database

Apply changes

You have no data sources connected.

Use the scope dropdown above to create a new data source, or select an existing one.

+ Data source

You haven't run a query yet.

Run a query to create a new set of results and data visualizations

Apply Cancel

2. Connect to your cluster to retrieve the databases

Create new data source

Data source name

Thermostat

Cluster URI \*

https://igniteadxsource.eastus2.kusto.windows.net/

Connect

Database \*

Select a database

Apply Cancel

3. Select the database that you want to query

Create new data source

Data source name

Thermostat

Cluster URI \*

https://igniteadxsource.eastus2.kusto.windows.net/

Connect

Database \*

Occupancy

Occupancy

Apply Cancel

# Creating a chart

1. Write the query, execute it and click on "Add Visual"

Thermostat ▾ ▶ Run

```
1 Thermostats
2 | where EnqueuedTimeUTC >= ago(10d)
3 | summarize percentile(Temp, 90) by bin(EnqueuedTimeUTC, 1d)
4
5
```

Results + Add visual

EnqueuedTimeUTC	percentile_Temp_90
2022-01-29 00:00:00.0000	80.216666666666669
2022-01-30 00:00:00.0000	79.849665532879825
2022-01-31 00:00:00.0000	79.932700472908
2022-02-01 00:00:00.0000	80.266666917067312
2022-02-02 00:00:00.0000	80.417558983666069
2022-02-03 00:00:00.0000	80.279343931930143
2022-02-04 00:00:00.0000	80.272895580687788
2022-02-05 00:00:00.0000	80.376923076923077
2022-02-06 00:00:00.0000	80.270208333333329
2022-02-07 00:00:00.0000	80.364573804573809
2022-02-08 00:00:00.0000	80.304166666666666

2. Choose your preferred chart type. Generally, ADX will infer x & y axes from the table. Use formatting options (conditional, legends, titles, etc.)



# Parameters

## 1. Add parameters to your query

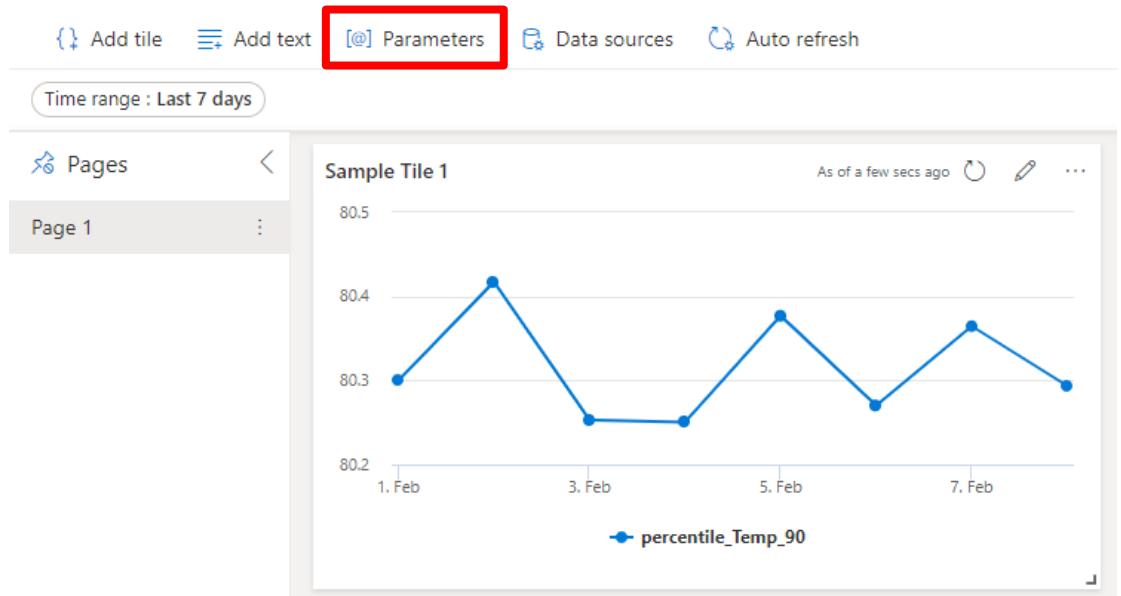
Time range : Last 1 hours

Thermostat

Run

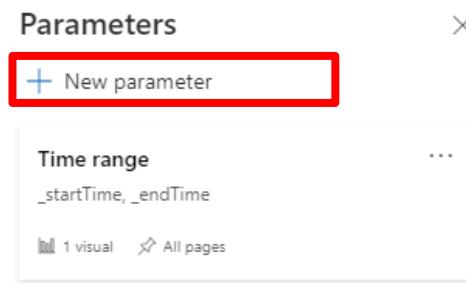
```
1 Thermostats
2 | where EnqueuedTimeUTC between (_startTime.._endTime)
3 | summarize percentile(Temp, 90) by bin(EnqueuedTimeUTC, 1d)
4
5
```

## 2. Add more parameters for your page (applies to tiles that have appropriately parametrized queries)



# Parameters (cont'd...)

1. Define new parameter



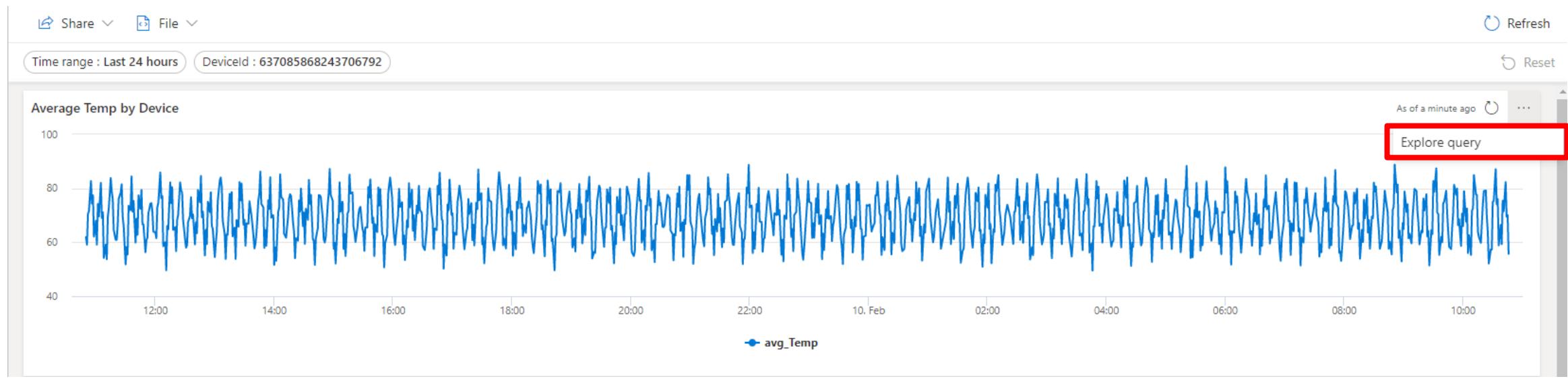
2. Enter values for the new parameter

Label  
Parameter type \*  
SingleSelection  
Variable name \*  
Data type \*  
string  
Show on Pages  
Select all  
Source  
 Fixed values  
 Query  
Value \*      Label  
+ Add value  
Select all  
 Add empty "Select all" value  
Adds an empty value to your possible options. This will need to be handled as a special case in consuming queries.  
Show sample  
Default value \*  
No default value  
Done      Cancel

3. Write a query to create a list of values for the parameter

Add a query  
Time range : [@] \_startTime,[@] \_endTime  
Thermostat      Run  
1 // Please enter your KQL query (Example):  
2 // <table name>  
3 // | where <datetime column> between ('\_startTime') .. ('\_endTime') // Time range filtering  
4 // | take 100  
5  
Results  
You haven't run a query yet  
Done      Cancel

# Switching between reporting and exploration



```
<< ▶ Run ⏪ Recall Scope: @igniteadxsource.eastus2/Occupancy
1 let ['startTime']=datetime('2022-02-09T10:49:14Z');
2 let ['endTime']=datetime('2022-02-10T10:49:14Z');
3 let ['Device_Id']='637085868243706792';
4 // Please enter your KQL query (Example):
5 // TableName | take 10
6 Thermostats
7 | where EnqueuedTimeUTC between (startTime .. endTime)
8 | where DeviceId == Device_Id
9 | summarize avg(Temp) by bin(EnqueuedTimeUTC,1m)
10 | render timechart
```

# Cross-filter

Cross-filters allow you to select a value in one visual and all dashboard visuals, such as line or scatter charts, are filtered to only show related data.

Using cross-filters achieves the same result as selecting the equivalent value for the parameter in the parameter list at the top of the dashboard.

Time range : Last 14 days Event Type : All Companies : @\_companies Repository name : All

Run

Visual formatting **Interactions**

EventsFromLiveStream  
| where CreatedAt between (startTime..endTime)  
| where Type in(\_eventType) or isempty(\_eventType)  
| extend RepoName = tostring(Repo.name)  
| where RepoName has\_any (\_repoName) or isempty(\_repoName)  
| summarize count() by Type

**Cross filter** **On**

Type (string)

Event Type (string)

Reset

Results **Visual** Expand preview Done (1.789 s) 15 records

Type distribution As of a minute ago

PushEvent 55.0 %

PullRequestReviewEvent 2.4 %

DeleteEvent 3.6 %

WatchEvent 4.8 %

IssueCommentEvent 5.2 %

PullRequestEvent 8.2 %

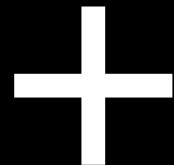
CreateEvent 13.4 %

GollumEvent  
PublicEvent  
MemberEvent  
CommitCommentEvent

1/4 ▲ ▼

Event Type	Percentage
PushEvent	55.0 %
PullRequestEvent	8.2 %
CreateEvent	13.4 %
IssueCommentEvent	5.2 %
WatchEvent	4.8 %
DeleteEvent	3.6 %
PullRequestReviewEvent	2.4 %

# ADX + Power BI



# 3 ways of connecting Power BI to Azure Data Explorer

## Direct Query

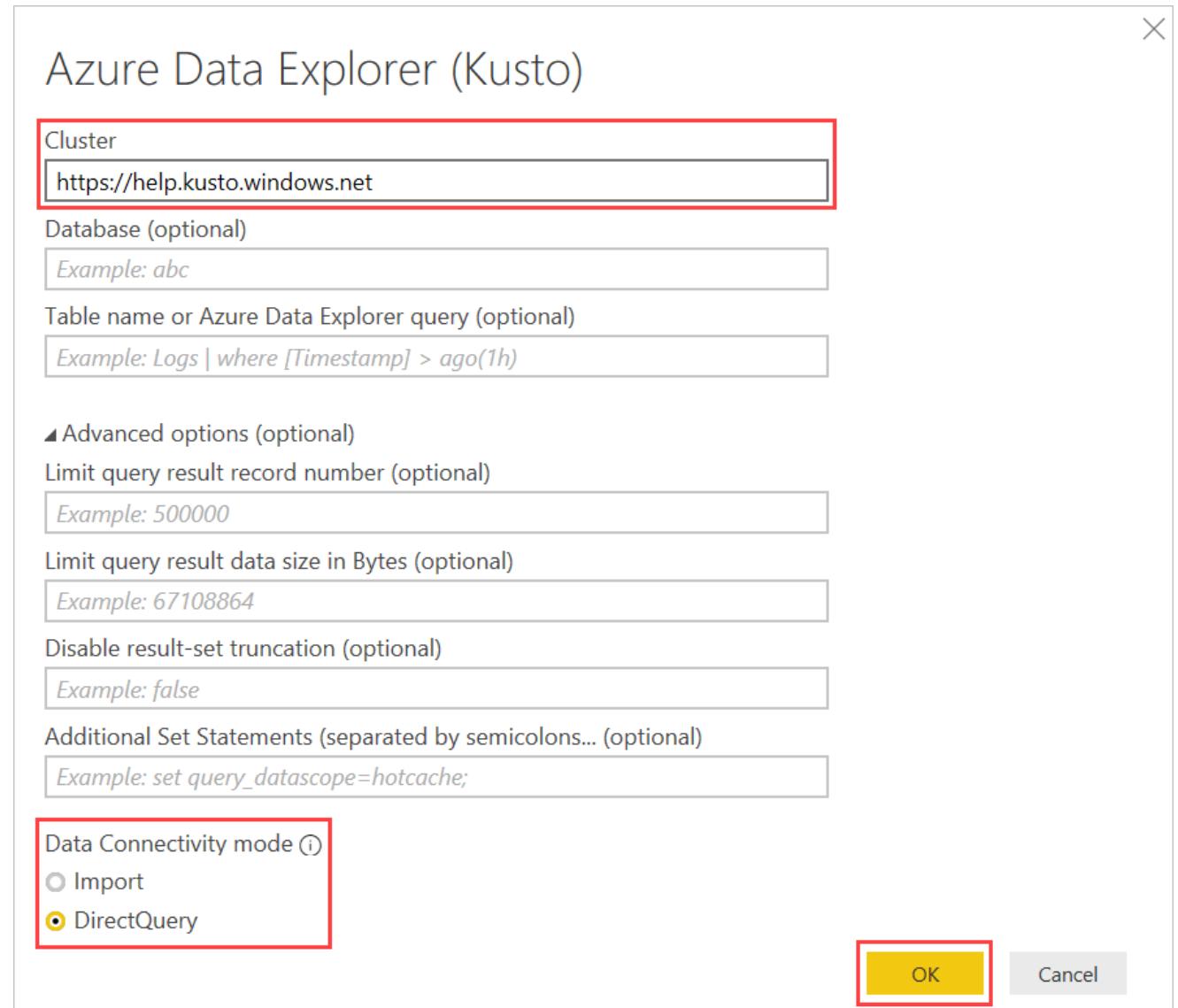
- Use for **large**, frequently updated datasets
- For creating Fact tables as they are large and change often
- Need near real-time data

## Imported Query

- Use for interaction with **smaller** data sets
- For creating Dimension tables as they're small and don't change often
- Don't need near real-time data.
- Data is already aggregated, or perform aggregation in Kusto

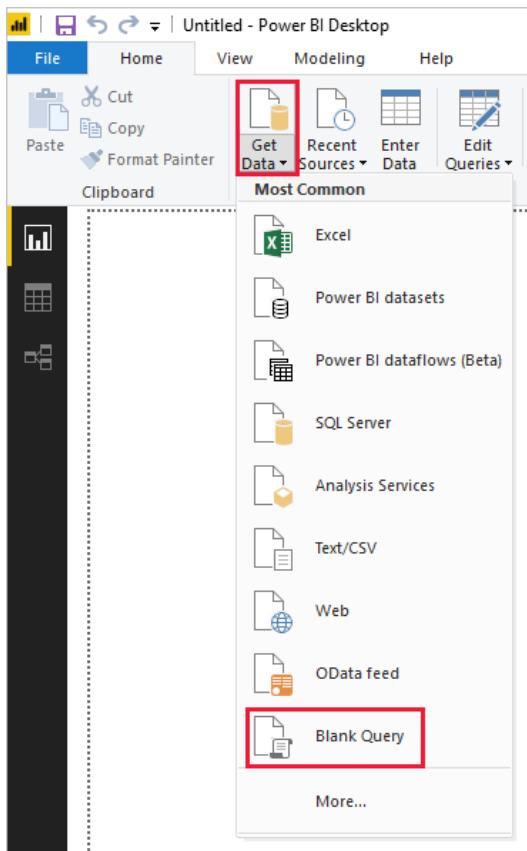
## SQL Query

- In Import mode, you can provide SQL query that the ADX-PBI Connector will translate into KQL



# Imported query

1. Use “Blank Query” to Get Data



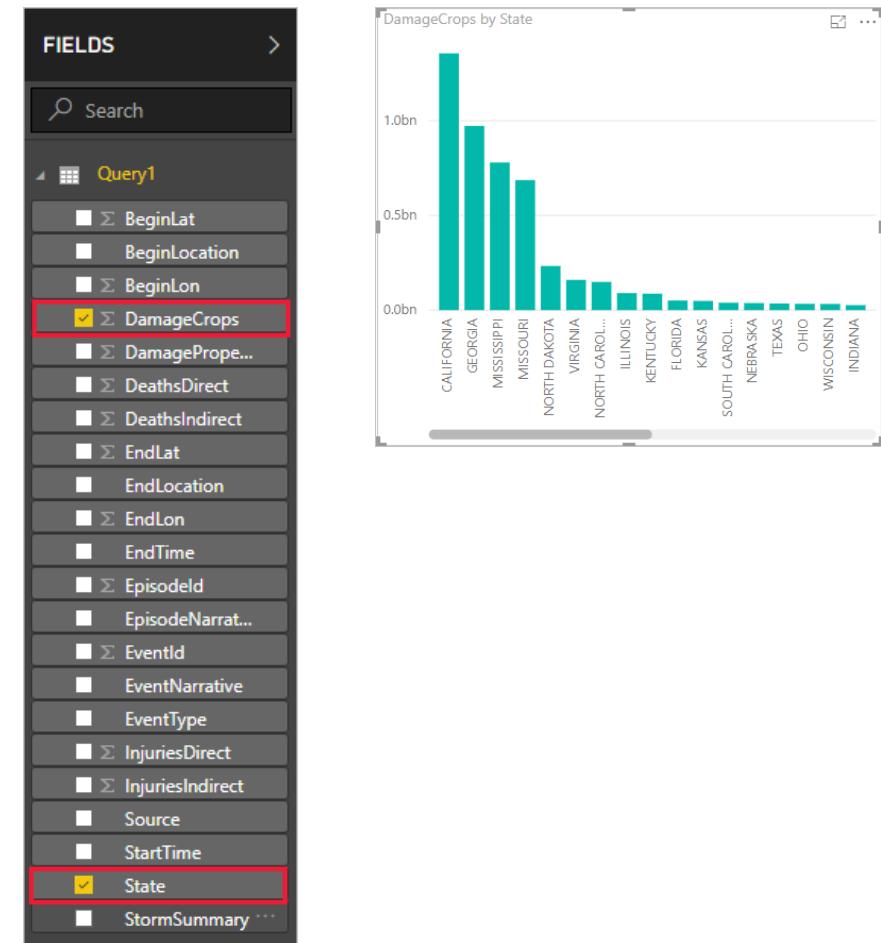
2. Paste KQL as shown below with the required parameters

The screenshot shows the 'Advanced Editor' window with a title bar 'Query1'. The main area contains a Kusto Query script:

```
let KustoQuery =  
    let Source = Json.Document(Web.Contents("https://help.kusto.windows.net:443/..."));  
    TypeMap = #table(  
        { "DataType", "Type" },  
        {  
            { "Double", Double.Type },  
            { "Int64", Int64.Type },  
            { "Int32", Int32.Type },  
            { "Int16", Int16.Type },  
            { "UInt64", Number.Type },  
            { "UInt32", Number.Type },  
            { "UInt16", Number.Type },  
            { "Byte", Byte.Type },  
            { "Single", Single.Type },  
            { "Decimal", Decimal.Type },  
            { "TimeSpan", Duration.Type },  
            { "DateTime", DateTimeZone.Type },  
            { "String", Text.Type },  
            { "Boolean", Logical.Type },  
            { "SByte", Logical.Type },  
            { "Guid", Text.Type }  
        }  
    );  
    Source  
);
```

Below the code, a note says 'No syntax errors have been detected.' At the bottom right are 'Done' and 'Cancel' buttons.

3. With the populated field, create Power BI visual



# Best practices for using Power BI with ADX

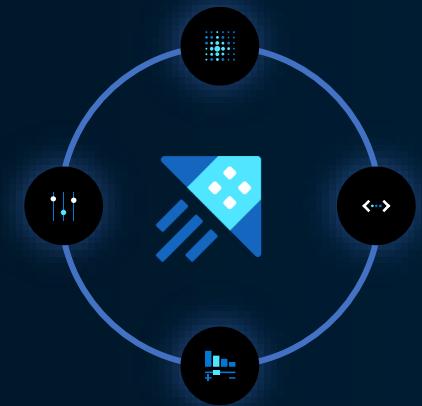
- **Travel light** - Bring only the data that you need for your reports to Power BI. For deep interactive analysis, use the Azure Data Explorer Web UI that is optimized for ad-hoc exploration with the KQL.
- **Composite model** - Use composite model to combine aggregated data for top-level dashboards with filtered operational raw data. You can clearly define when to use raw data and when to use an aggregated view.
- **Parallelism** - improve the performance of dashboard rendering by increasing the parallelism of the end-to-end flow as follows:
  - Increase the number of concurrent connections in DirectQuery in Power BI.
  - Use weak consistency to improve parallelism. This may have an impact on the freshness of the data.
- **Effective slicers** – Use sync slicers to prevent reports from loading data before you're ready. After you structure the data set, place all visuals, and mark all the slicers, you can select the sync slicer to load only the data needed.
- **Use filters** - Use as many Power BI filters to focus the Azure Data Explorer search on the relevant data shards.
- **Efficient visuals** – Select the most performant visuals for your data.

# Demo\*

- ADX Dashboard: [IoT Demo 01](#)
- Power BI: [Thermostat](#)
- Grafana: [IoT Demo Dashboard](#)

# Ops & Management

## Module 8 Objectives



After completing this Learning, you will be able to understand:

1. ADX Operations
2. Management
3. BCDR
4. Monitor
5. Troubleshoot

# ADX Ops

## Management

- Cluster [advisor](#), SKUs, auto stop, scaling, principals, networking and extensions.
- DB permissions, config using KQL, principals, and policies
- Data purge, delete or duplicates.

## Business continuity (BC) and Disaster recovery (DR)

- Mitigate disruptive events
- High availability of ADX
- DR configurations

## Monitor

- Metrics, diagnostic logs, commands, queries, and tables
- Resource health & insights

## Troubleshoot

- Creating a cluster, connecting, databases & tables

# Cluster Types & SKUs

## Cluster Types

- Prod: Two nodes for engine and data-management, backed by SLA.
- Dev/Test (no SLA): Single node, lowest cost, lacks redundancy.

## Compute

- Compute optimized: High CPU to SSD ratio. Suited for a high rate of queries over small-mid data size.
- Heavy compute: Higher CPU to SSD ratio on AMD-chips.

## Storage optimized

- Larger (1-4 TBs) per node.
- Suited for large volume of data with less intensive compute.
- Optional premium storage (managed disk) attached for hot storage.

## Isolated compute

- For workloads that require server instance-level isolation.

# Auto-stop

- Inactivity is defined by having no ingestion/queries, and an average CPU below 1% of capacity.
- Restart the cluster via Azure Portal, [REST API](#), [CLI](#) or [PowerShell](#).
- Will receive several days of recommendations to stop the cluster from Advisor.
- If inactive, small amount or no data, then stopped after 5 days.
- If inactive, data ingested, then stopped after 10 days.

Create an Azure Data Explorer Cluster

Configurations

Enable/disable the following Azure Data Explorer capabilities to optimize cluster costs and performance.

Streaming ingestion:  On  Off

Enable purge:  On  Off

Auto-Stop cluster:  On  Off

**REST example**

Update the cluster using the following operation:

HTTP

```
PATCH https://management.azure.com/subscriptions/12345678-1234-1234-1234-123456789098/resourceGroups/kustorgtest/providers/Microsoft.Kusto/clusters/kustocluster?api-version=2020-08-01
```

Request body to disable Auto-Stop

```
JSON
{
  "properties": {
    "enableAutoStop": false
  }
}
```

Request body to enable Auto-Stop

```
JSON
{
  "properties": {
    "enableAutoStop": true
  }
}
```

**tobeautostopped | Activity log**

Activity log

Overview

Access control (IAM)

Diagnose and solve problems

Permissions

Query

Settings

Search (Ctrl+ /)

Activity

Refresh

Diagnostics settings

Download as CSV

Logs

Pin current filters

Management Group : None

Subscription : My\_Subscription

Event severity : All

Resource group : oren

Resource : tobeautostopped

Add Filter

Timespan : Last 2 weeks

Operation name

Status

Time

Time stamp

Subscription

Stop Clusters

Updated

a month ago

Thu Sep 16 ...

My\_Subscription

# Scaling

- A static size can lead to under-utilization or over-utilization
- Scale a cluster with changing demand

## Horizontal (in & out)

- Manual (default) static capacity.
- Optimized autoscale (**recommended**).
  - **Predictive** scaling based on your usage pattern.
  - Will scale in-time to handle the load.
  - Takes **6 days** of data to take affect.
  - If under-utilized will scale in and out if over-utilized.
- Custom autoscale is dynamic based on metrics you specify.

## Vertical (up & down).

- Scale the SKU.

The screenshot shows the Azure Data Explorer cluster configuration interface. At the top, there are buttons for Save (highlighted with a red box), Discard, Refresh, Configure, Run history, JSON, Notify, and Diagnostics logs. Below this, a section titled "Choose how to scale your resource" offers three options: Manual scale (radio button off), Optimized autoscale (radio button on, highlighted with a red box), and Custom autoscale (radio button off). The Optimized autoscale section includes fields for Minimum instance count (2) and Maximum instance count (10), both with green checkmarks. A note below states: "This Azure Data Explorer cluster will be scaled intelligently between the minimum and maximum instance counts, based on usage patterns." In the "Select compute specifications" section, a table lists various SKU options with their details:

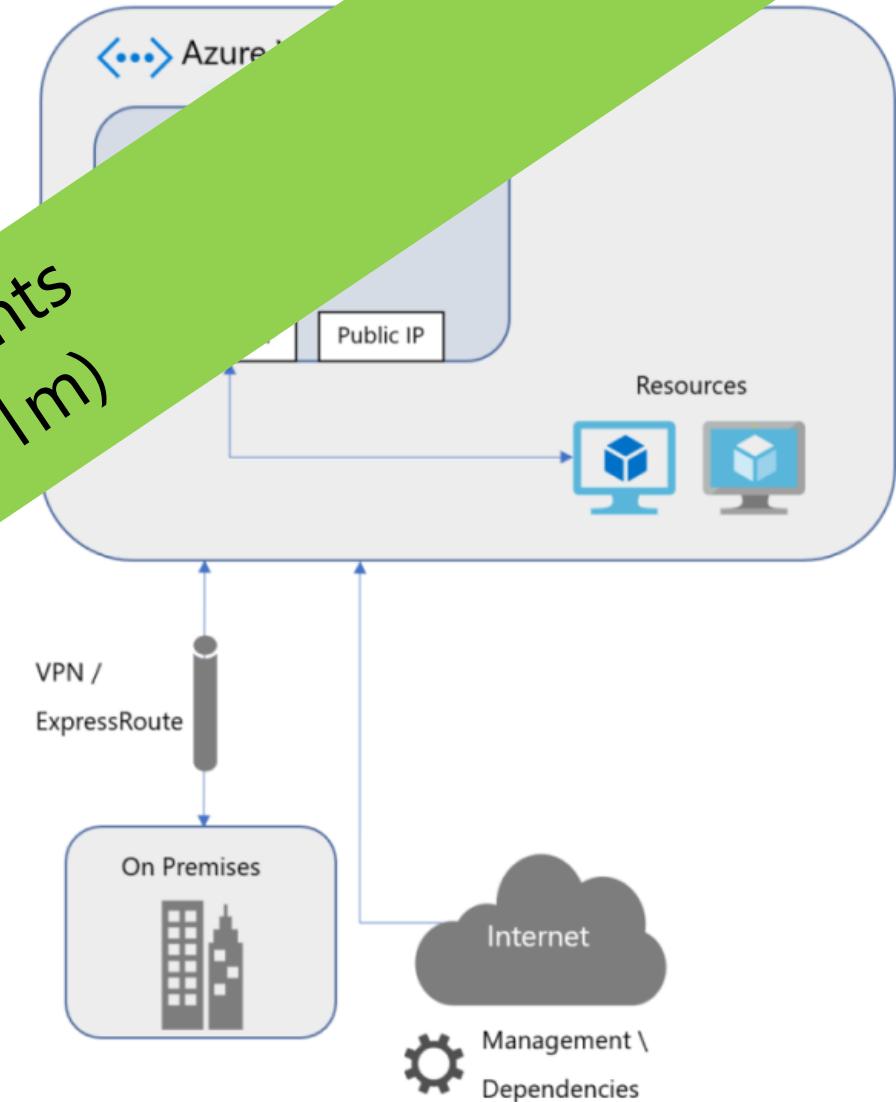
SKU	vCPUs	GB Cache	GB Ram	Compute \$/h	Azure Data Explorer \$/h
D11_V2	2	75	14	\$0.185/h	\$0.11/h
D12_V2	4	150	28	\$0.371/h	\$0.22/h
D13_V2	8	307	56	\$0.742/h	\$0.44/h
D14_V2	16	614	112	\$1.482/h	\$0.88/h
L4	4	650	32	\$1.652/h	\$0.374/h
L5	8	1300	64	\$3.302/h	\$0.748/h
L6	16	2600	128	\$6.602/h	\$1.496/h
L7	32	5200	256	\$13.202/h	\$2.992/h
L8	64	10400	512	\$26.402/h	\$5.984/h

Below the table, a note says: "This SKU supports cluster's..."

## V-Net Deployment

- Intent is to isolate the network traffic and limit the attack surface.
- Enforce [Network Security Group](#) (NSG) rules on traffic.
- Connect your on-premises network to Azure via VPN or ExpressRoute.
- Secure your data connection using [TLS](#) and [event grid](#)) with service endpoints.

Recommend Private Endpoints  
[youtu.be/yUL-CR6\\_h9g](https://youtu.be/yUL-CR6_h9g) (11m)



**Important:** NSG rules block access to public IP addresses outside the VNet. Subnet size must be specified, since it can't be changed after ADX is deployed.

# Principals, Identity and Access Control

- Cluster and Database permissions can be managed several ways.
- Programmatically using C#, Python & ARM template.
- Role-based access control (RBAC) to cluster users.
- You can configure access controls to databases via Azure portal, CLI, PowerShell or KQL.

## Kusto

```
1 .add database Test users ('aaduser=imikeoein@fabrikam.com') 'Test user (AAD)'
2
3 .add database Test admins ('aadgroup=SGDisplayName;fabrikam.com') 'Test group
4 @fabrikam.com (AAD)'
5
6 .add database Test viewers ('aadapp=4c7e82bd-6adb-46c3-b413-
7 fdd44834c69b;9752a91d-8e15-44e2-aa72-e9f8e12c3ec5') 'Test app on another
8 tenant (AAD)'
```

**See more:** <https://docs.microsoft.com/azure/data-explorer/kusto/management/access-control/role-based-authorization>

# Configure a database via KQL [script](#)

The script can only run control commands that start with the following verbs, each separated by **exactly** one line break:

- `.create`
- `.create-or-alter`
- `.create-merge`
- `.alter`
- `.alter-merge`

## Main methods

1. [Upload script \(simple\)](#): you create a script as a blob in an Azure storage account, provide its URL and shared access signatures (SaS) directly.
2. [Inline](#) (advanced), you provide your Kusto Query Language script **inline**, and a storage account is created during the deployment.

# Database and table policies

## Usage scenarios

- Set a **retention** policy with a 10-day [soft-delete](#) period.
- Set a **database** hot cache policy for latest 5 days of data.
- Set a **table** hot cache policy for latest 5 days of data.
- Add an Azure Active Directory (AAD) application (**database principle**) as admin for a database.

Using Python or C#: <https://docs.microsoft.com/azure/data-explorer/database-table-policies-csharp>

# Data Purge & Soft-Delete

## Purge

- Deletion through `.purge` protects personal data and shouldn't be used in other scenarios.
- Not designed for frequent deletes, or massive quantities of data.
- Invoked in two ways & disabled by default.

Use case	Considerations	Method
Delete all data from a table		<a href="#">Use .clear table</a>
Routinely delete old data	Routinely delete old data	<a href="#">Use a retention policy</a>
Bulk delete specific data by extents	Only use if you are an expert user	<a href="#">Use .drop extents</a>
Delete records based on their content	<ul style="list-style-type: none"><li>- Storage artifacts that contain the deleted records aren't necessarily deleted</li><li>- Deleted records <b>can't</b> be recovered (regardless of any retention or recoverability settings)</li><li>- Quick way to delete records</li></ul>	<a href="#">Use soft delete</a>
Delete records based on their content	<ul style="list-style-type: none"><li>- Storage artifacts that contain the deleted records are deleted</li><li>- Deleted records <b>can't</b> be recovered (regardless of any retention or recoverability settings)</li><li>- Requires significant system resources and time to complete</li></ul>	<a href="#">Use purge</a>

# Duplicates

- Malfunctioning devices resend local cached data that cause duplication in the analytical database.
- Monitor the percentage of duplicate data using KQL.

## Kusto

```
1 let _sample = 0.01; // 1% sampling
2 let _data =
3 DeviceEventsAll
4 | where EventDateTime between (datetime('10/1/18') .. datetime('10/10/18'));
5 let _totalRecords = toscalar(_data | count);
6 _data
7 | where rand() <= _sample
8 | summarize recordsCount=count() by hash(DeviceId) + hash(EventId) + hash(StationId)
9 | summarize duplicateRecords = countif(recordsCount > 1)
10 | extend duplicate_percentage = (duplicateRecords / _sample) / _totalRecords
```

See more: <https://docs.microsoft.com/azure/data-explorer/dealing-with-duplicates>

# Business Continuity and Disaster Recovery (BCDR)

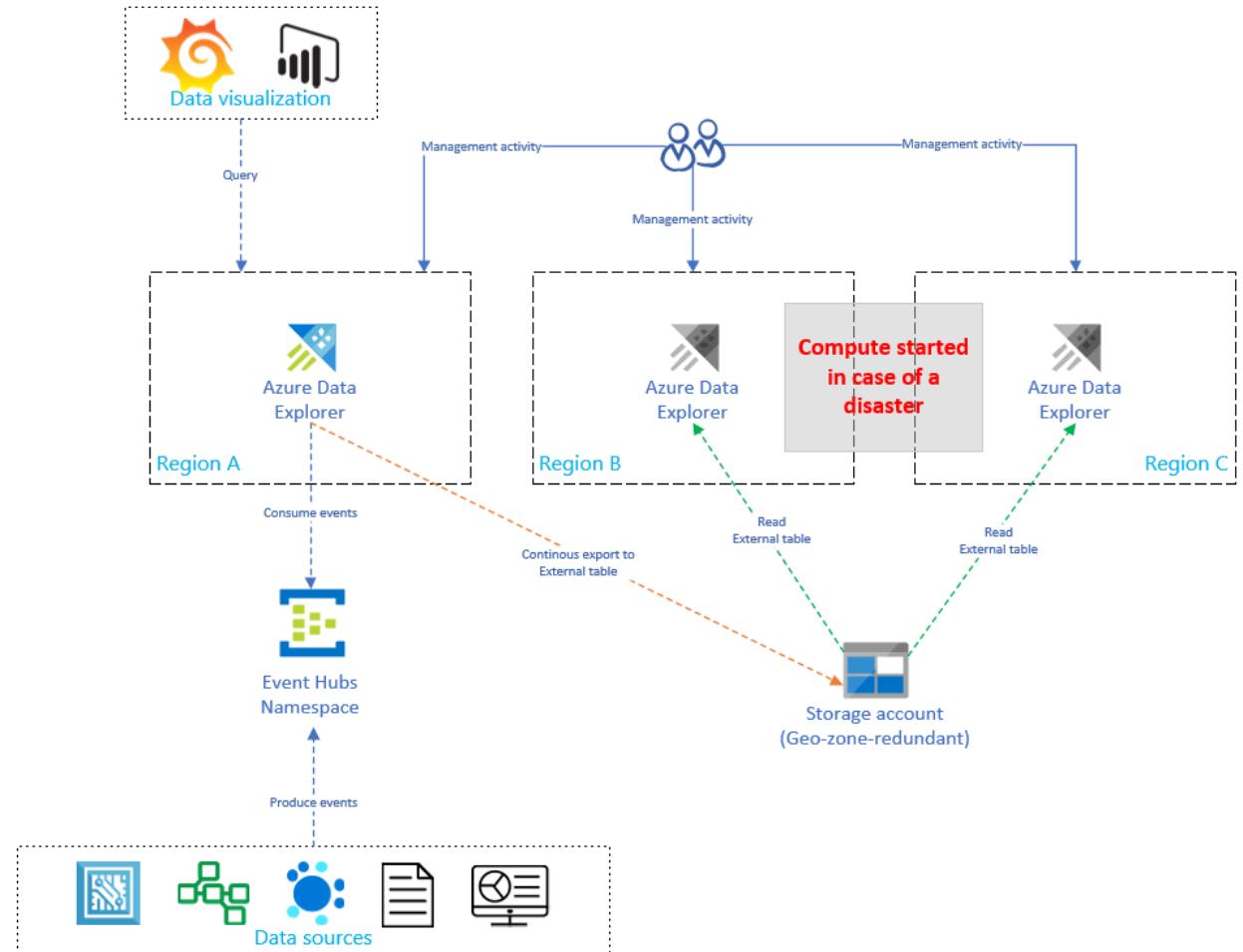
BCDR enables your business to continue operating in the face of a disruption and mitigate events.

- Human errors: Prevent by enabling delete lock capability. Recover tables using `.undo drop table`, must first enable **recoverability** property in the retention policy.
- High availability: Storage defaults to Locally Redundant Storage (LRS) in a datacenter. At an additional cost, use Zone Redundant Storage (ZRS) for across Azure regional availability zones. At provisioning time, select availability zones to distribute nodes for intra-region resiliency. Optional, leader-follower capability for read-only access.
- Availability zones: Protect from partial region failure. Zone failure is intra-region. Selection is only supported at time of cluster creation and can't be modified later.
- Datacenter outage: same as regional outage.
- **Regional Outage**: use multiple clusters across paired regions. Consider multiple DR configurations based on your RTO & RPO.

# BCDR Solutions

## On-demand data recovery configuration

- Cluster in Region A performs [continuous export](#) to a GRS storage account.
- Create secondary replicas with access to data using [external tables](#).
- Start/Stop replicas using [Power Automate](#), CLI, PowerShell or REST API.
- Implement a highly available application service
- Perform dynamic or static routing using DNS-based [Traffic Manager](#) or [Front Door](#) based routing.



# Troubleshooting

## **Creating a Cluster**

- Adequate permission
- Validation errors
- [Azure service health](#)
- Contact [support](#)

## **Connecting to ADX**

- Connection string
- Adequate permissions
- Cluster exists
- [Azure service health](#)
- Contact [support](#)

## **Creating or deleting a database or table**

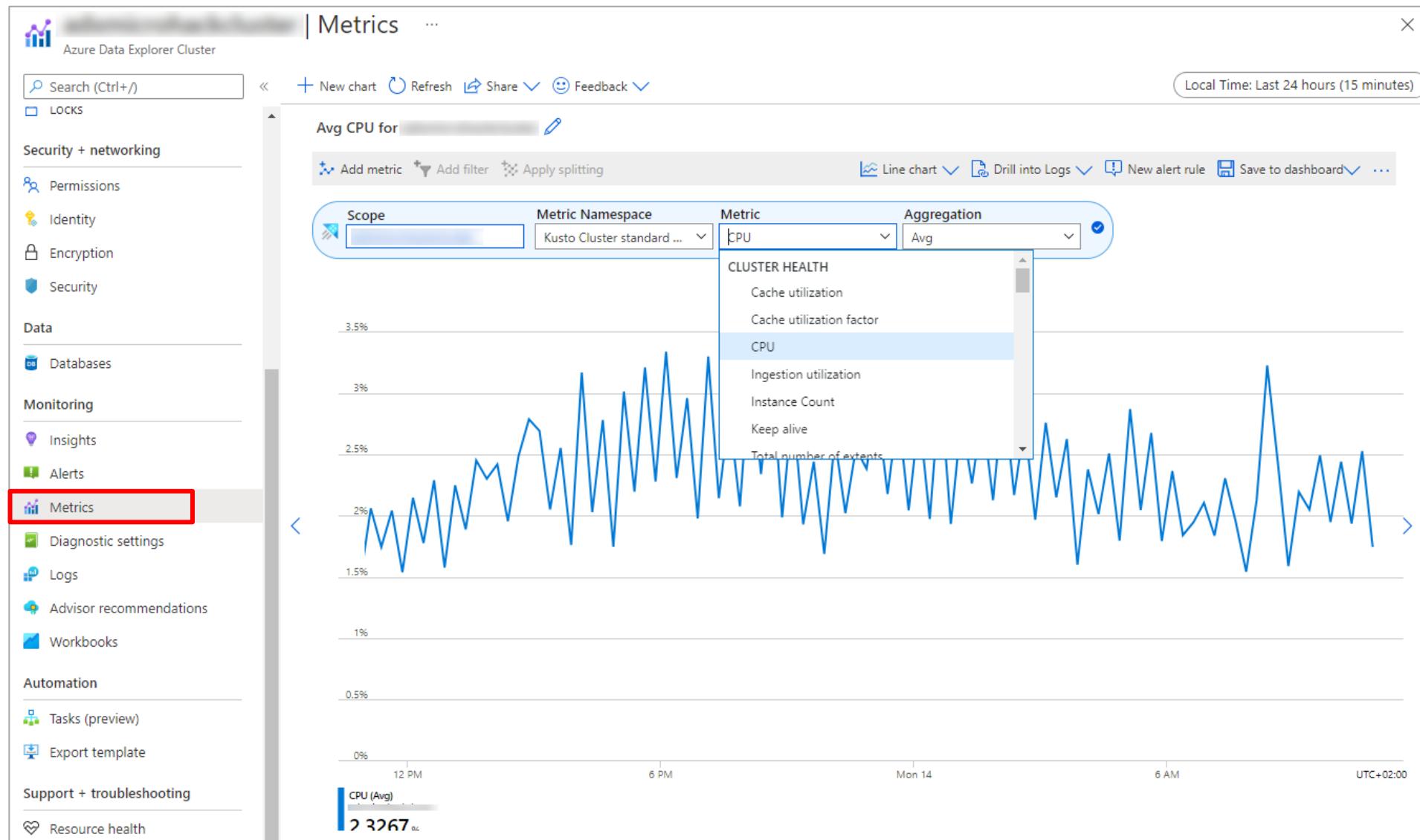
- Adequate permissions
- Database name
- Retention and caching within range
- Already exists

# System information

Commands available to Database Admins and Database Monitors roles.

- .show queries** - displays information on completed and running queries.
- .show commands** - displays information on completed commands and their resources utilization.
- .show commands-and-queries** - displays information on completed commands and queries, and their resources utilization.
- .show journal** - displays history of the metadata operations.
- .show operations** - displays administrative operations both running and completed, since Admin node was last elected.
- .show ingestion failures** - displays information on failures encountered during data ingestion to the cluster.
- .show table data statistics** - displays table data statistics per column.

# Azure monitor metrics



# Azure monitor diagnostic logs

The screenshot shows the 'Diagnostic settings' blade for the 'Azure Data Explorer Cluster' resource. The left sidebar contains navigation links for Security + networking, Data, Monitoring, and Metrics. The 'Metrics' section is currently selected and highlighted with a red box. The main content area displays diagnostic settings configuration options, including a search bar, refresh and feedback buttons, and a detailed description of what diagnostic settings are used for. A table lists existing diagnostic settings, which currently show 'No diagnostic settings defined'. A blue '+ Add diagnostic setting' button is available to begin configuration. Below this, instructions for adding a diagnostic setting are provided, along with a list of data types that can be collected.

Diagnostic settings

Azure Data Explorer Cluster

Search (Ctrl+ /)

Refresh Feedback

Diagnostic settings are used to configure streaming export of platform logs and metrics for a resource to the destination of your choice. You may create up to five different diagnostic settings to send different logs and metrics to independent destinations. [Learn more about diagnostic settings](#)

Diagnostic settings

Name	Storage account	Event hub	Log Analytics workspace	Partner solution	Edit setting
No diagnostic settings defined					

+ Add diagnostic setting

Click 'Add Diagnostic setting' above to configure the collection of the following data:

- SucceededIngestion
- FailedIngestion
- IngestionBatching
- Command
- Query
- TableUsageStatistics
- TableDetails
- Journal
- AllMetrics

Databases

Insights

Alerts

Metrics

Diagnostic settings

Logs

Advisor recommendations

Workbooks

# Azure monitor diagnostic logs

Home > Azure Data Explorer Clusters > [Cluster Name] >

## Diagnostic setting

Save Discard Delete Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name \* My-logs

**Logs**

Categories

- SucceededIngestion
- FailedIngestion
- IngestionBatching
- Command
- Query
- TableUsageStatistics
- TableDetails
- Journal

Metrics

- AllMetrics

**Destination details**

Send to Log Analytics workspace

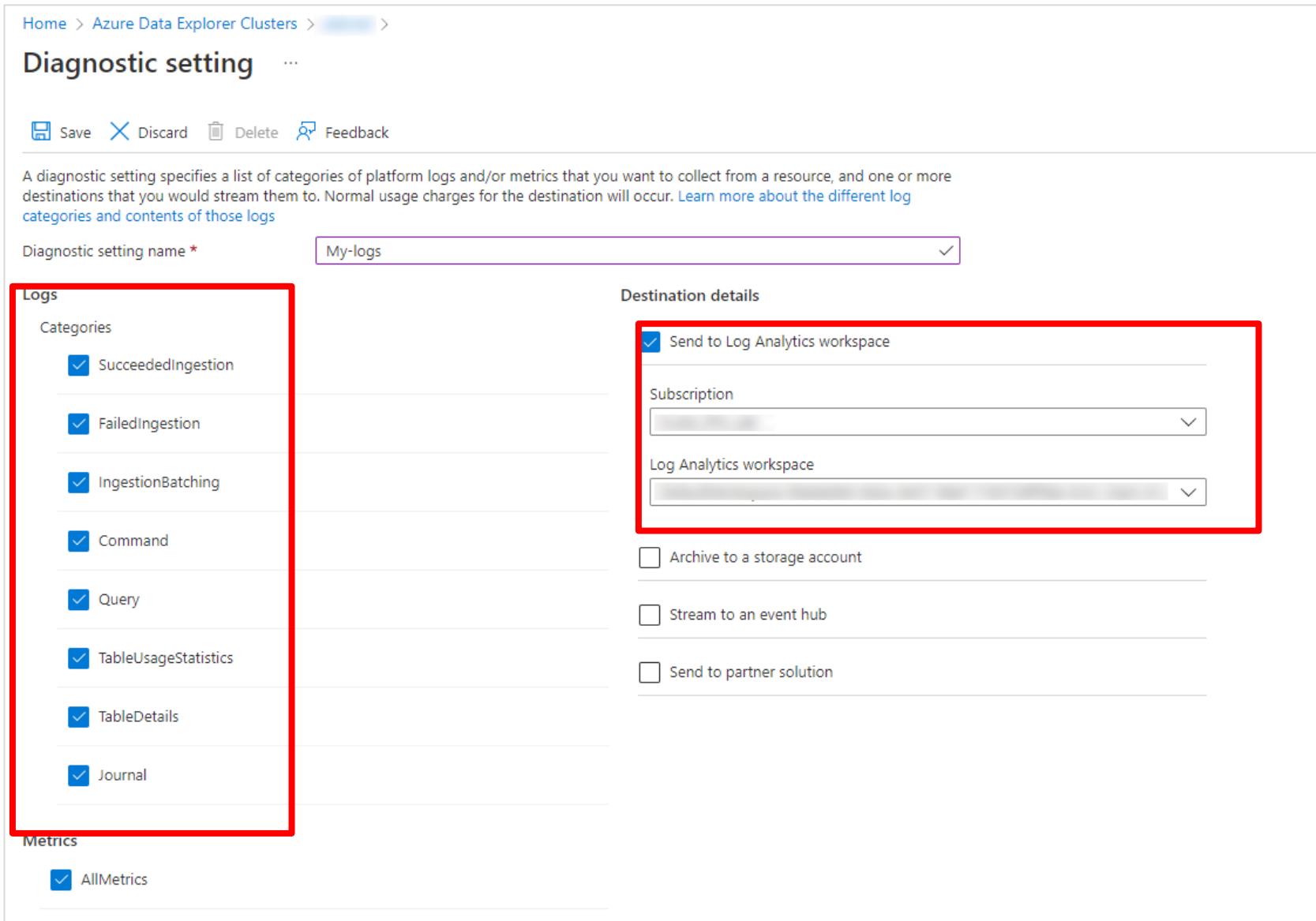
Subscription: [Subscription Name]

Log Analytics workspace: [Log Analytics Workspace Name]

Archive to a storage account

Stream to an event hub

Send to partner solution



# Metrics vs Logs

Attribute	Metrics	Logs
Data	Numerical values only	Text or numeric data
Benefits	Analyze with time charts. Lightweight and capable of near-real time scenarios.	Analyze with KQL. Ideal for deep analysis and identifying root cause.
Structure	Standard set of properties including sample time, resource being monitored, a numeric value. Some metrics include multiple dimensions for further definition (for example, the database name).	Unique set of properties depending on the log type.
Granularity	Cluster or database level (no table-level details)	Including table-level details (for example, failed ingestions per table)
Collection	Collected at regular intervals, out-of-the box	Need to be enabled. May be collected sporadically as events trigger a record to be created.
View in Azure portal	Metrics blade	Logs blade

# Resource health

The screenshot shows the 'Resource health' blade for an 'Azure Data Explorer Cluster'. The left sidebar lists navigation options: Identity, Encryption, Security, Data (Databases), Monitoring (Insights, Alerts, Metrics, Diagnostic settings, Logs, Advisor recommendations, Workbooks), Automation (Tasks (preview), Export template), and Support + troubleshooting (Resource health, New Support Request). The 'Resource health' option is highlighted with a red box. The main content area displays the title 'Resource health' with a 'Learn more' link. It includes a button to 'Add resource health alert' and another to 'Diagnose and solve problems'. A message states: 'Resource health watches your resource and tells you if it's running as expected.' Below this, a section titled 'Available' indicates there are no known problems. A 'What actions can you take?' section lists monitoring documentation. The 'Health history' table shows resource health events over the last 4 weeks, all marked as 'Available'.

Resource health | ...

Add resource health alert Diagnose and solve problems

Resource health watches your resource and tells you if it's running as expected. [Learn more](#)

Available

There aren't any known problems affecting this Azure Data Explorer cluster.

What actions can you take?

1. To know how to monitor health of your Azure Data Explorer cluster, check the [Monitor Azure Data Explorer documentation](#)

Health history

Resource health events over the last 4 weeks

Date	Description
02/14/2022	Available
02/13/2022	Available
02/12/2022	Available
02/11/2022	Available
02/10/2022	Available
02/09/2022	Available
02/08/2022	Available
02/07/2022	Available

# Azure Monitor insights – at scale (multiple resources)

From the Azure Monitor page, you can select 'Azure Data Explorer Clusters'. This opens the "at-scale" view, which allows monitoring of multiple clusters.

The screenshot shows the Azure Monitor interface for managing multiple Azure Data Explorer Clusters. The left sidebar lists various monitoring categories, with 'Azure Data Explorer Clusters' highlighted by a red box. The main area displays a summary for 7 clusters, with a larger view for '7 / 37'. The 'Ingestion Performance' tab is selected. The interface includes a search bar, a top navigation bar with 'Subscriptions', 'Cluster', and 'Time range' filters, and a detailed table showing ingestion metrics for each cluster. The table includes columns for Subscription, Ingestion latency, Succeeded ingestion results, Failed ingestion results, Ingestion volume, and Events processed. Specific clusters shown include ADX-Prod-001 through 005 and ADX-Dev-001, 002, all under the Contoso-Prod subscription. A second group under Fabrikam-Dev contains two clusters.

Subscription	Ingestion latency	Succeeded ingestion results	Failed ingestion results	Ingestion volume	Events processed
Contoso-Prod (5)	15.28mins	3	2	900MB	-
Contoso-Prod (5)	0s	0	2	300MB	-
Contoso-Prod (5)	0s	0	3	300MB	-
Contoso-Prod (5)	41.13s	44	2	500MB	71
Contoso-Prod (5)	21.07mins	8.92K	16	80GB	475.95 K
Fabrikam-Dev (2)	0s	0	-	60MB	-
Fabrikam-Dev (2)	0s	0	10	400MB	-

# Azure Monitor insights

ADX-prod-001 | Insights (preview) Activity log

Azure Data Explorer Cluster

Search (Ctrl+ /) Workbooks Customize Time range: Last 7 days Overview Activity log Access control (IAM) Tags Diagnose and solve problems Permissions Query Settings Scale up Scale out Configurations Identity Encryption Security Properties Locks Data Databases Monitoring Insights (preview) Alerts Metrics Diagnostic settings Logs Advisor recommendations Workbooks Automation Tasks (preview) Export template Support + troubleshooting Resource health

Overview Key Metrics Usage Tables Cache Ingestion Cluster Boundedness Materialized Views

Keep Alive (Avg) CPU (Avg) Ingestion Utilization (Avg) Ingestion Latency (Avg) Cache Utilization (Avg) Succeeded Ingestions (#) Failed Ingestions (#)

1 0.0747 % 0 % 0.2 s 16 % 168 5

Show Metrics Help

Resource health (not affected by the time range parameter)

Availability state	Occurred time	Reason chronicity	Reported time	Summary
Available	11/11/2020, 6:45:52 AM	Persistent	12/31/2020, 7:23:37 PM	There aren't any known problems affecting this Azure Data Explorer cluster.

Resource health page >

Advisor recommendations (not affected by the time range parameter)

Description	Category	LastUpdated
(PREVIEW) Right-size Azure Data Explorer clusters for optimal cost	Cost	12/31/2020, 1:44:24 PM

Advisor recommendations page >

Top resource consumers

This is the top consumers' usage distribution within the CPU and memory consumption of the cluster. To see more details, go to the [Usage](#) tab.

Principal	Application	% of Used Cluster Memory	% of Used Cluster CPU
user001@contoso.com	KusWeb	66.515%	69%
user002@contoso.com	Kusto.Explorer	33.485%	6%
user003@contoso.com	Flow	0%	14%
AAD app id=11111111-bbbb-2222-4444-aaaaaaaaaa	Grafana-ADX	0%	11%

Unique user count

Over a sliding timeline window. Not affected by the time range parameter

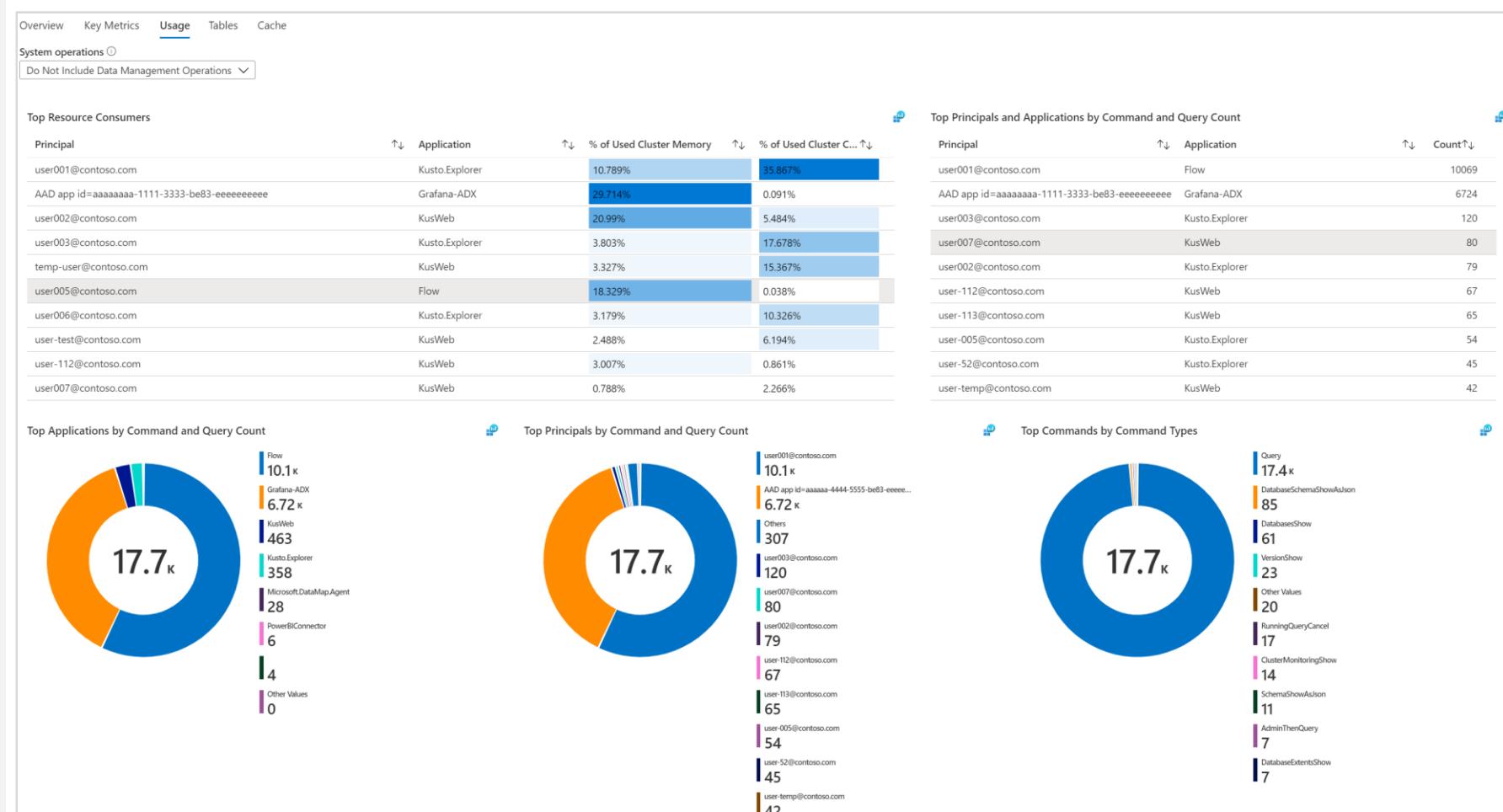
Daily (Avg) Weekly (Avg) Monthly (Avg)

2.714 4.857 4.857

# Azure Monitor insights

The **Usage tab** allows users to deep dive into the performance of the cluster's commands and queries. On this tab, you can:

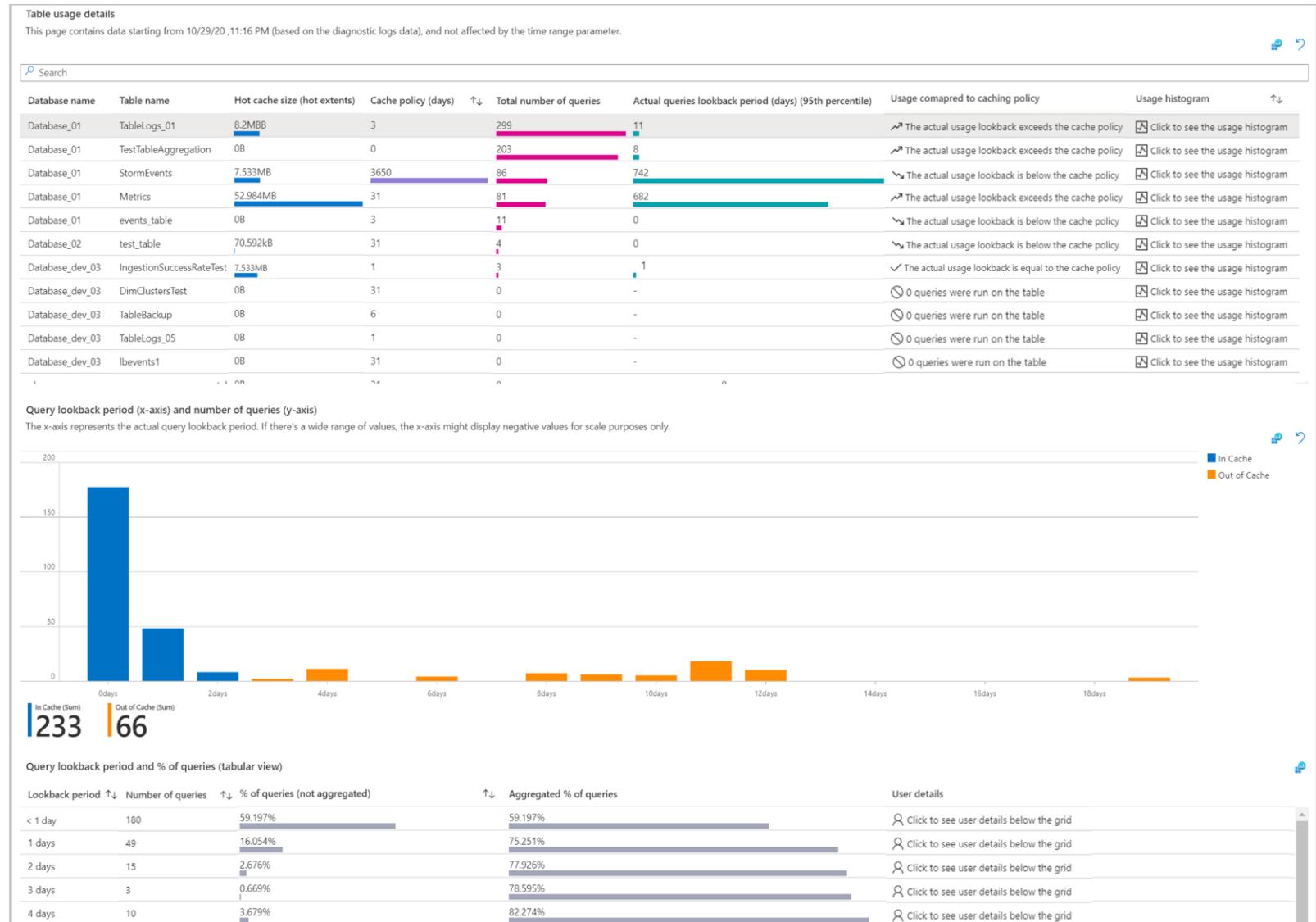
- See which workload groups, users, and applications are sending the most queries or consuming the most CPU and memory.
- Identify users and applications by failed queries.
- Identify recent changes in the number of queries, compared to the historical daily average
- Identify trends and peaks in the number of queries, memory, and CPU consumption by workload group, user, application, and command type.



# Azure Monitor insights

The **Cache tab** allows you to analyze their actual queries' lookback window patterns and compare them to the configured cache policy (for each table).

You can identify tables used by the most queries and tables that are not queried at all and adapt the cache policy accordingly.

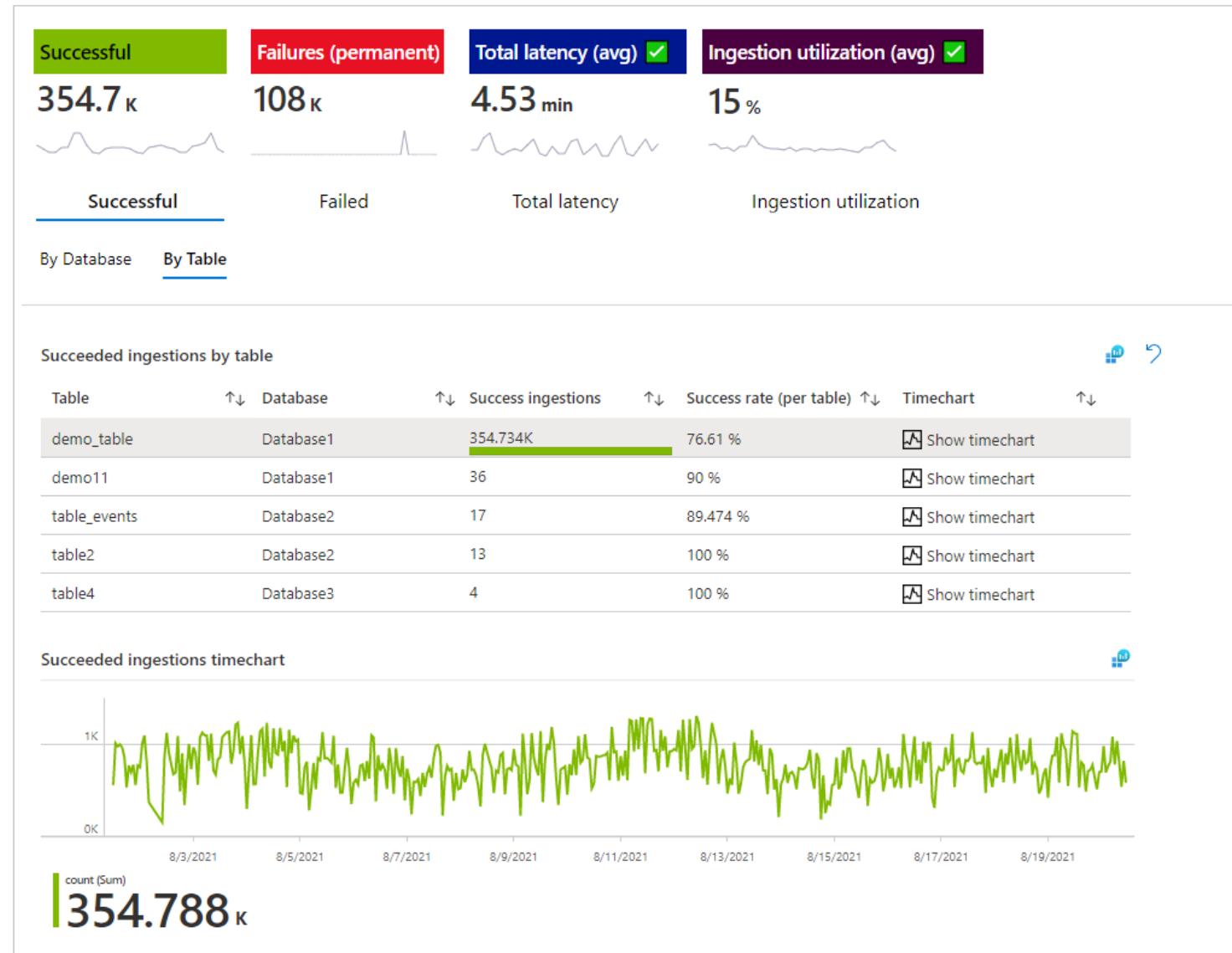


# Azure Monitor insights

The **Ingestion tab** allows you to monitor the ingestion process.

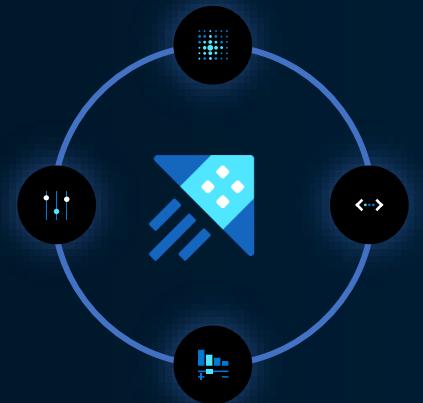
Here are some questions you can get answers to:

1. What is the result of my ingestion attempts? How many ingestions have succeeded or failed? (by database or table granularity)
2. Are there any tables that may be missing data due to ingestion errors? What exactly are the error details?
3. What was the amount of data processed by the ingestion pipeline?
4. What is the latency of the ingestion process? Did the latency build up in ADX's pipeline or upstream of ADX?
5. For ingestion using Event Hub, Event Grid, or IoT Hub, how can I compare the number of events arriving at ADX with the number of events sent for ingestion?



# Advanced

## Module 8 Objectives



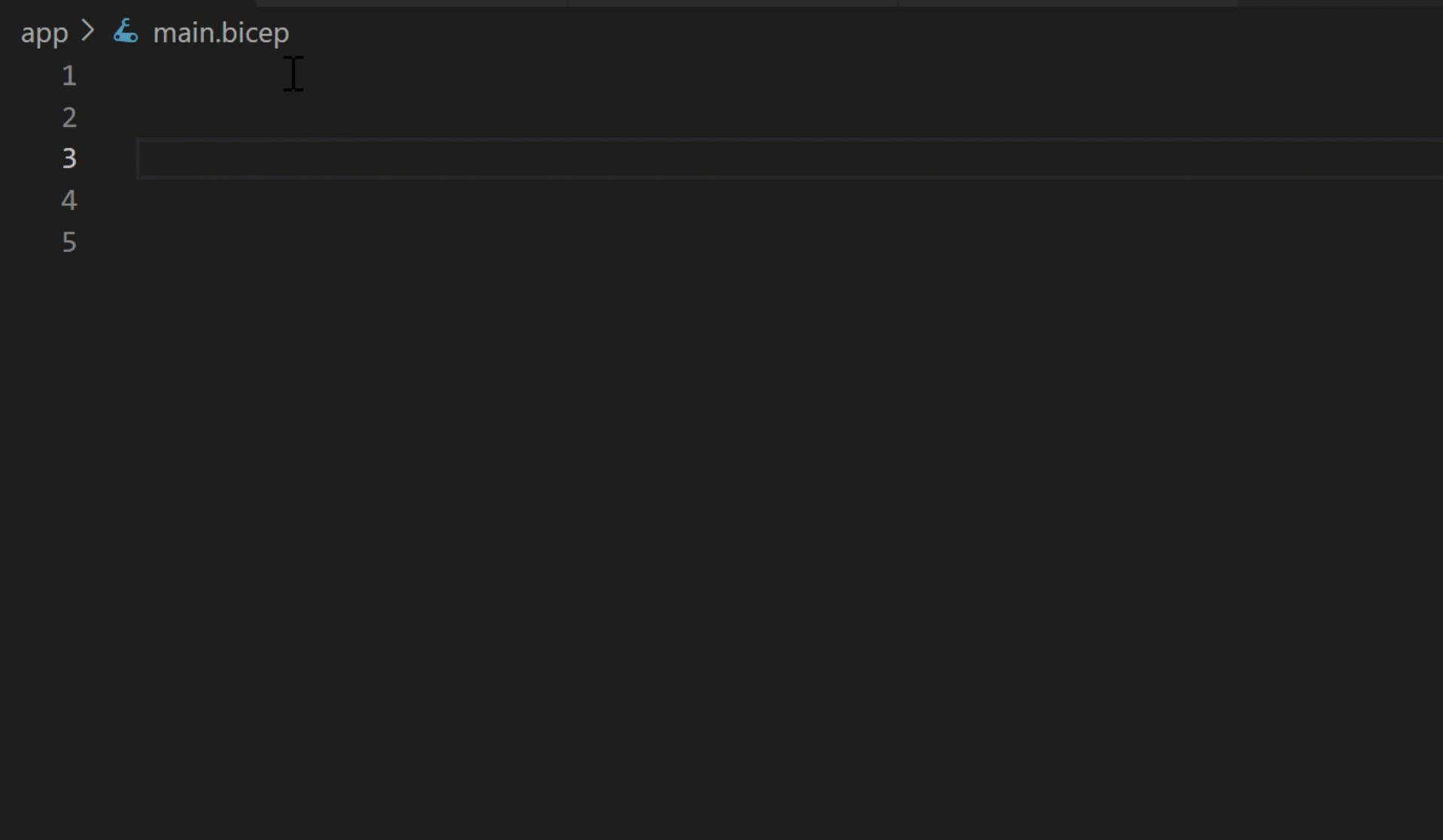
After completing this Learning, you will be able to understand:

1. Bicep
2. Management Commands
3. Policies
4. Caching
5. Partitioning
6. REST APIs
7. SDKs
8. Optimize for high-concurrency

# What is Bicep?

- Next generation ARM template. Bicep is a **trans-piled** language. Expressive, yet also compact with notion of modules.
- Available for every service in Azure because its native binding to ARM.
- In a Bicep file, you **define the infrastructure** you want to deploy to Azure, and then use that file throughout the development lifecycle to repeatedly deploy your infrastructure.
- Simple syntax, takes out a lot of noise of the ARM template JSON.
- Bicep & ARM [templates](#) & property values are available for **Microsoft.Kusto** clusters.
- Workshop hands-on lab **bicep modules** available in GitHub repo <https://aka.ms/adx.iot> > modules.

# Authoring Bicep using [VS Code Extension](#)



A screenshot of the Visual Studio Code interface. The title bar shows "app > main.bicep". The code editor displays the following Bicep code:

```
1
2
3
4
5
```

The code editor has a dark theme with light-colored text. Line numbers 1 through 5 are visible on the left side. A cursor is positioned at the start of line 2. A small red rectangle highlights the area between line 3 and line 4.

# Management Commands (Control)

Retrieve information, query data, modify service.

## Plugin

- Show
- Enable
- Disable

## System

- Show queries
- Show commands
- Show journal
- Show operations
- Show failed ingestions

## Security

- Add
- Drop
- Set

## Policies

- Cache
- Sharding
- Update
- Retention

See more: <https://docs.microsoft.com/azure/data-explorer/kusto/management>

# Combining queries and control commands

Example of **AdminThenQuery**:

## Kusto

```
1 // 1. Using pipe: Count how many tables are in the database-in-scope:  
2 .show tables  
3 | count  
4  
5 // 2. Using semicolon: Count how many tables are in the database-in-scope:  
6 .show tables;  
7 $command_results  
8 | count  
9  
10 // 3. Using semicolon, and including a let statement:  
11 .show tables;  
12 let useless=(n:string){strcat(n,'-', 'useless')};  
13 $command_results | extend LastColumn=useless(TableName)
```

# Combining queries and control commands

Example of **AdminFromQuery**:

## Kusto

```
1 .set MyTable <|
2 let text="Hello, World!";
3 print str=text
```

Drop all drop-by [tags](#) from extents in table MyTable:

## Kusto

```
1 .ingest ... with @'{"tags":"[\"drop-by:2016-02-17\"]"}'
2
3 .drop extents <| .show table MyTable extents where tags has "drop-by:2016-02-17"
```

# Ingest from query

Execute a query or a control command and ingest the results of the query into a table.

Command	If table exists	If table doesn't exist
.set	Command fails	Creates table and ingests data
.append	Data is appended to the table	Command fails
.set-or-append	Data is appended to the table	Creates table and ingests data
.set-or-replace	Data replaces the data in the table	Creates table and ingests data

# Policies Types

- Auto delete
- **Cache**
- Callout
- Capacity
- Ingestion batching
- Ingestion time
- Managed Identity
- Merge
- **Partitioning**
- **Retention**
- Extent tags retention
- Restricted view access
- **Row level security**
- Row order
- Sandbox
- Sharding
- Streaming ingestion
- **Update**
- Query weak consistency

See more: <https://docs.microsoft.com/azure/data-explorer/kusto/management>

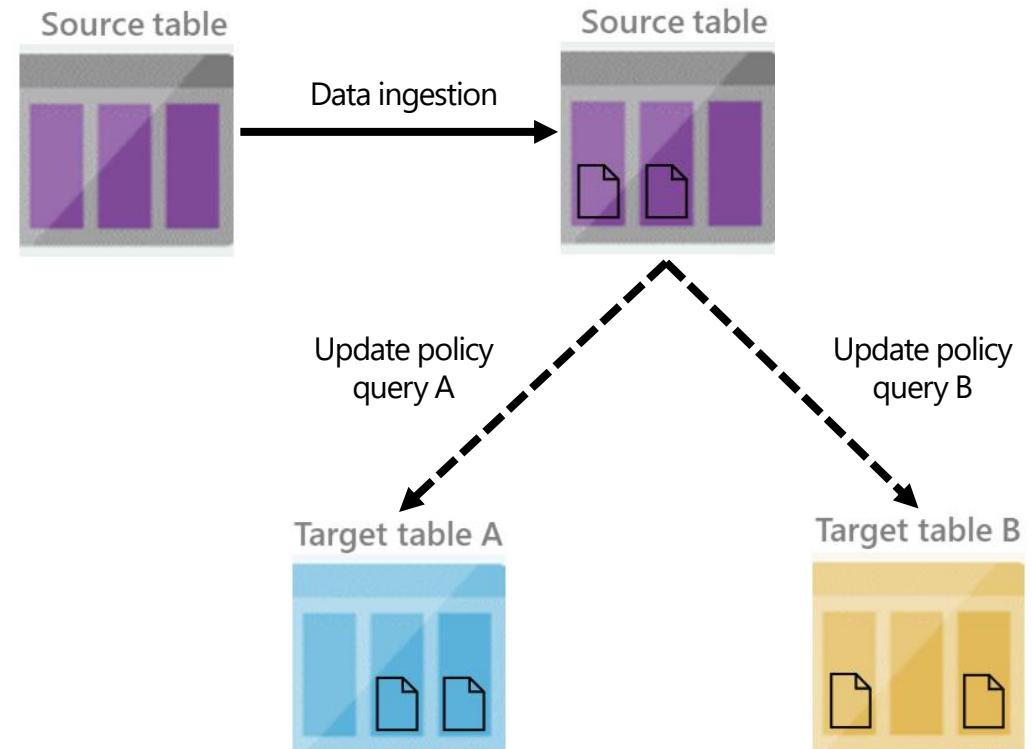
# Update policies

## Basics

- Think of it as “**lightweight ELT**”
- Attached to target table, execute function on new ingestion to source table
- A table can have **multiple** update policies associated with it and policies can be chained.
- Beware resource usage – test your code well.

## Usage scenarios

- **Splitting data** to multiple tables – e.g., receiving JSON events with different schemas
- **Filtering** data – e.g., some data is not of interest, but upstream filtering is difficult
- Data **transformation** – e.g., format changes, exploding JSON objects or arrays



See more: <https://y0nil.github.io/kusto.blog/blog-posts/update-policies.html>

# Usage Tips

- If update policies are used to perform data transformations and original data does not need to be preserved, set soft delete to 0sec and the update policy to be transactional.

Example:

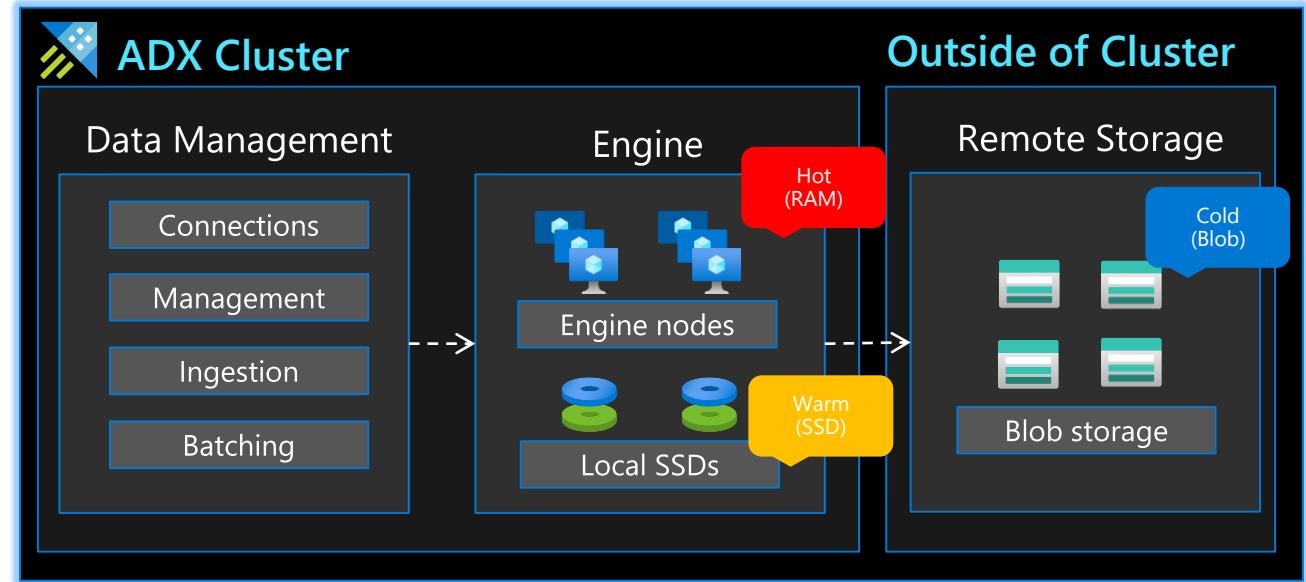
## Kusto

```
1 .alter-merge table SourceTable policy retention softdelete = 0sec
2 recoverability = disabled
3
4 .create function TransformData() {
5     SourceTable | extend Counter = Counter * 2 //your business logic
6 }
7
8 .alter table TargetTable policy update
9 @'[{ "Enabled": true, "Source": "SourceTable", "Query": "TransformData()", 
10 "IsTransactional": true, "PropagateIngestionProperties": true}]'
```

# Data tiering

## Basics

- ADX is a great tool for storing large volumes of data due to high compression ratios. However, storing all the data on fast media can be prohibitively expensive.
- ADX offers **tiering** mechanism with full transparency to the user – i.e., same query, data in different places.
- Data can sit in RAM of compute nodes, local SSDs, or cold storage (Azure Blob).
- The “where” is defined by caching policies. **Important** to remember – all data remains query-able.



# Data tiering

## Usage scenarios

- Users are typically interested in recent data – last hour or day - as they, for example, perform troubleshooting. However, data may be preserved for years for business/regulatory requirements.
- Caching policies dramatically lower cost by moving older than x-since ingestion to cold storage.
- Users can define additional time windows of hot data to aid with analysis.
- Queries can be scoped to target only hot data.

### Kusto

```
1 set query_datascope="hotcache"; MyTable | limit 10
2
3 .alter cluster policy caching hot = 30d
4
5 .alter cluster policy caching
6     hot = 30d,
7     hot_window = datetime(2021-01-01) .. datetime(2021-02-01)
```

# Retention Policy

## Basics

- Sometimes there is no need to keep old data.
- Provides a way to automatically delete data after x-since ingestion.
- The data is not deleted immediately but at some later point depending on the free resources in the cluster.
- Rows beyond retention policy are not returned in queries.

## Usage scenarios

- Often combined with caching policies
- RAM/SSDs -> Cold Storage -> Delete
  - RAM/SSDs -> Cold Storage -> Delete
  - Caching policy      Retention policy
- Recoverability=enabled allows to un-delete data for 14 days.

## Kusto

```
1 .alter-merge table MyTable policy retention softdelete = 365d recoverability = enabled
```

# Query results cache

## Basics

- Helps preserve cluster resources for frequently executed queries
- Improves responsiveness (user-perceived latencies)
- Queries must be the same, letter-to-letter, to be served from cache
- Caching period is configurable
- Not compatible with tables that have row-level security enabled
- The cache capacity is currently fixed at **1GB per node** and query results doesn't exceed **16MB**.

## Usage scenarios

- Populating dropdowns in dashboards
- Business reporting scenarios where staleness is acceptable
- Not a good fit for monitoring dashboards where reaction time to spike/dip is critical

## Kusto

```
1 set query_results_cache_max_age = time(5m);
2 MyTable
3 | distinct MyColumn //your expensive query
```

# Partitioning policy

## Basics

- Defines how [extents \(data shards\)](#) should be partitioned for a specific table or a [materialized view](#).
- By default, extents are partitioned by their ingestion time, and in most cases there's no need to set a partitioning policy.

## Usage scenarios

- Filters on mid/high cardinality string column, ie. TenantId or MetricId.
- Aggs or joins on high cardinality group-by string column, ie. At least **10M distinct** values.
- Filtering out-of-order data by datetime column.

**Caution:** Adds overhead. Not recommended, if compressed data per partition is less than 1GB. Review the recommendations for [materialized views partitioning policy](#).

**See more:** [.alter table partitioning policy](#)

# Row Level Security (RLS)

## Basics

- Evaluates a predicate based on AAD UPN or group membership and returns a restricted view of the underlying table.
- Access restriction logic sits in the database tier, rather than in application tier.
- Can't be enabled on a table with [continuous data export](#), referenced by a query of an [update policy](#), on which [restricted view access policy](#) is configured or another RLS policy is enabled.

## Usage scenarios

- Granting access to subset of data – e.g., we might want to restrict engineers to only app data they actively debug, or in a multitenant scenario to restrict tenants to see only their data in a shared table.
- Data Masking – e.g., PII data, only last four digits of credit card

# Row Level Security (RLS)

Example:

## Kusto

```
1 //Example of RLS statement
2 .create-or-alter function RLSFunction() {
3     let IsManager=current_principal_is_member_of("aadgroup=managers@company.com");
4     let AllData=MyTable | where IsManager;
5     let PartialData=MyTable | where not(IsManager) and UserColumn==current_principal();
6     union AllData, PartialData
7 }
8
9 //and applied to a table
10 .alter table MyTable policy row_level_security enable "RLSFunction()"
```

# REST API Overview

Primary means of communicating with any ADX service.

- Query data
- Query and modify metadata
- Ingest data
- Query the service health status
- Manage resources

## REST Operations Groups

- [\*\*Clusters\*\*](#): Creates or updates a cluster
- [\*\*Data Connections\*\*](#): Creates or updates a data connection
- [\*\*Databases\*\*](#): Creates or updates a database
- [\*\*Operations\*\*](#): Lists available operations

## MS-TDS

- ADX also supports the Microsoft SQL Server communication protocol (MS-TDS).
- Limited support for T-SQL queries
- Enables client tools such as LINQPad, sqlcmd, Tableau, Excel, and Power BI.
- KQL over TDS, ie. `sp_execute_kql` via C# [SqlCommand](#)
- ADX cluster as a SQL Server Linked Server via ODBC Driver for SQL Server 2017 or later.

See more: <https://docs.microsoft.com/azure/data-explorer/kusto/api/tds>

# Python SDK

- *Kusto Python Client* library lets you query ADX clusters using Python 2.x/3.x
- [Jupyter Notebooks](#) that are attached to Spark clusters.

PyPi packages:

- [azure-kusto-data](#)
- [azure-kusto-ingest](#)
- [azure-mgmt-kusto](#)

## Option 1: Install via PyPi

```
pip install azure-kusto-data  
pip install azure-kusto-ingest
```

See more: <https://docs.microsoft.com/azure/data-explorer/kusto/api/python/kusto-python-client-library>

# Python SDK

## IngestSample.py

```
1 from azure.kusto.data import KustoConnectionStringBuilder, DataFormat
2 from azure.kusto.ingest import QueuedIngestClient, IngestionProperties, FileDescriptor,
3 BlobDescriptor
4
5 ip = IngestionProperties(database="IoTAnalytics", table="iotingest",
6 data_format=DataFormat.CSV)
7 client =
8 QueuedIngestClient(KustoConnectionStringBuilder.with_interactive_login("https://adxiot.eastus.
9 kusto.windows.net"))
10
11 # ingest from file. ie. the raw (uncompressed) size of the data is (15 KB)
12 fd = FileDescriptor("iotingest.csv", 15360)
13 client.ingest_from_file(fd, ingestion_properties=ip)
14 client.ingest_from_file("iotingest.csv", ingestion_properties=ip)
15
16 # ingest from blob. ie. the raw (uncompressed) size of the data is (50 KB)
17 bd = BlobDescriptor("https://adlsg2.blob.core.windows.net/cntr/iotingest.csv.gz?sas", 51200)
18 client.ingest_from_blob(bd, ingestion_properties=ip)
```

See more: <https://github.com/Azure/azure-kusto-python/blob/master/azure-kusto-ingest/tests/sample.py>

# Optimize for high concurrency

Use cases include large-scale monitoring and alerting dashboards.

## Optimize data

- Schema design, partitioning, pre-aggregation and caching.

## Leader-follower

- [Azure Data Share](#)

## Optimize queries

- Query best practices, results cache, query consistency

## Set cluster policies

- [Request rate limit policy](#)

## Monitor ADX

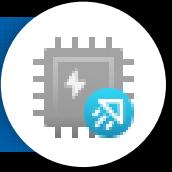
# Demo\*

- KQLMagic

# Connecting data & unlocking insights is key

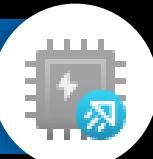


# Distributed Query Engine



- Storage and Compute isolated
- Auto compression, indexing, optimization
- Semi-structure (JSON) and unstructured data (free text) indexing
- Data is sharded, distributed, cached across nodes on local SSDs and persisted on cloud storage
- Ingestion time-based partitioning + custom partitioning
- Hot and cold data management
- Table level caching + results set caching
- Workload management
- Partitioning
- Materialized views
- External tables for Blob, Data Lake, and SQL

# Technical Use Cases

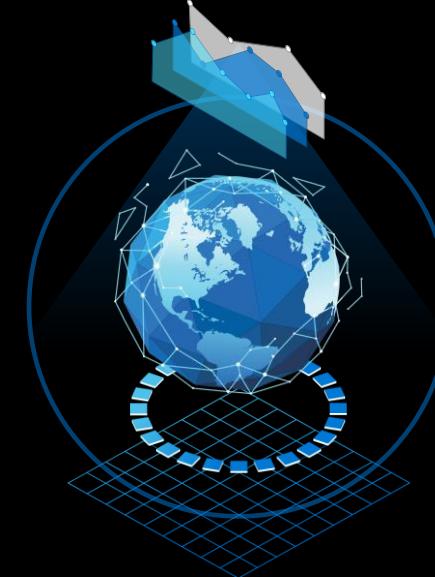


Near-real time big data analytics solutions



Near-real time logs analytics solutions for your observability, monitoring, and security data

- Replace infrastructure-based log search solutions
- Complement with your SIEM product
- Accelerate your AI Ops journey



IoT Analytics solution

- Connected devices
- Build cloud base historian



Analytical SaaS solutions

# Resources



- Documentation: [aka.ms/adx.docs](https://aka.ms/adx.docs)
- Reference Architecture: [aka.ms/adx.iotrefarch](https://aka.ms/adx.iotrefarch)
- Tech Blogs: [aka.ms/adx.techblogs](https://aka.ms/adx.techblogs)
- Free personal cluster: [aka.ms/adx.free](https://aka.ms/adx.free)
- Cost Estimator: [aka.ms/adx.cost](https://aka.ms/adx.cost) or [aka.ms/adx.cost.old](https://aka.ms/adx.cost.old)

Thank you

[aka.ms/adx.iot.eval](https://aka.ms/adx.iot.eval)

