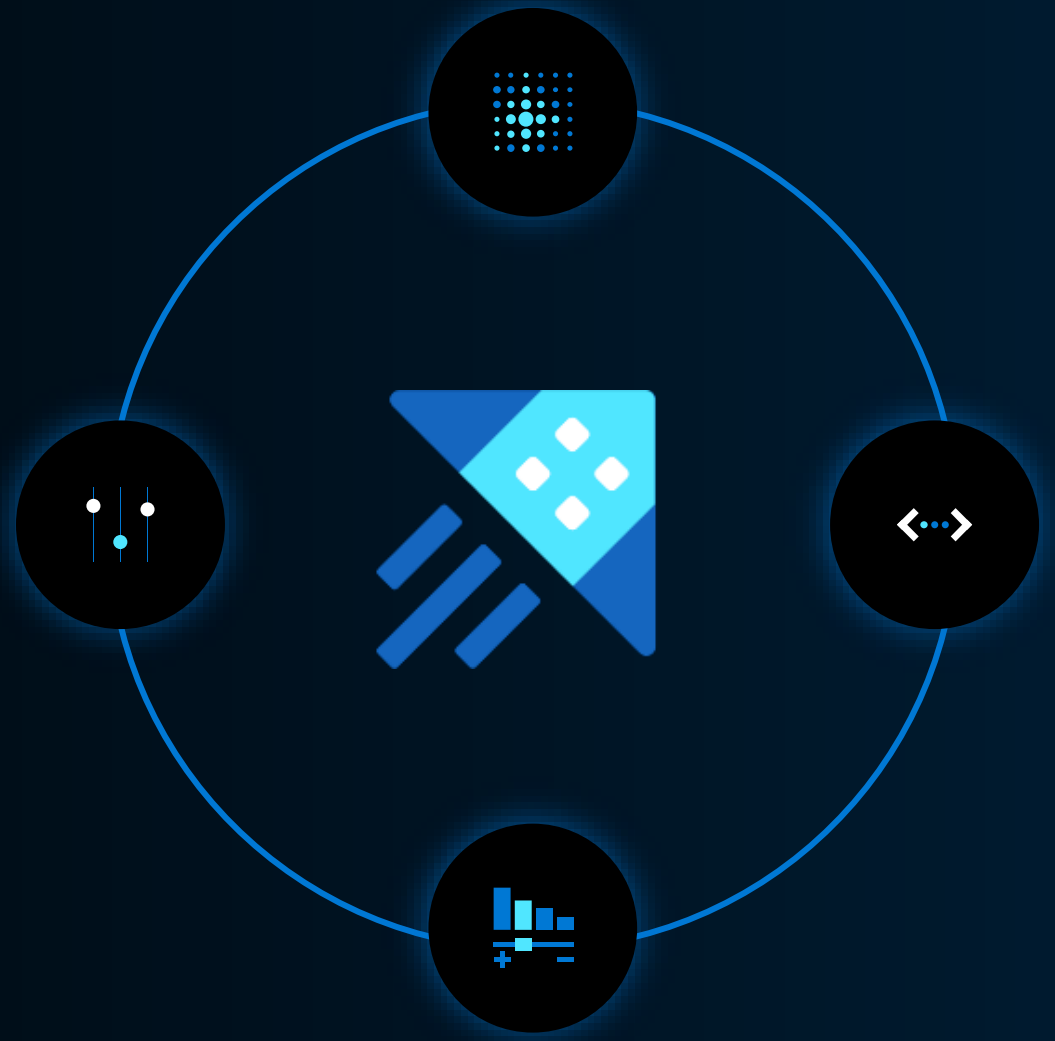
Microsoft

ADX for IoT Analytics

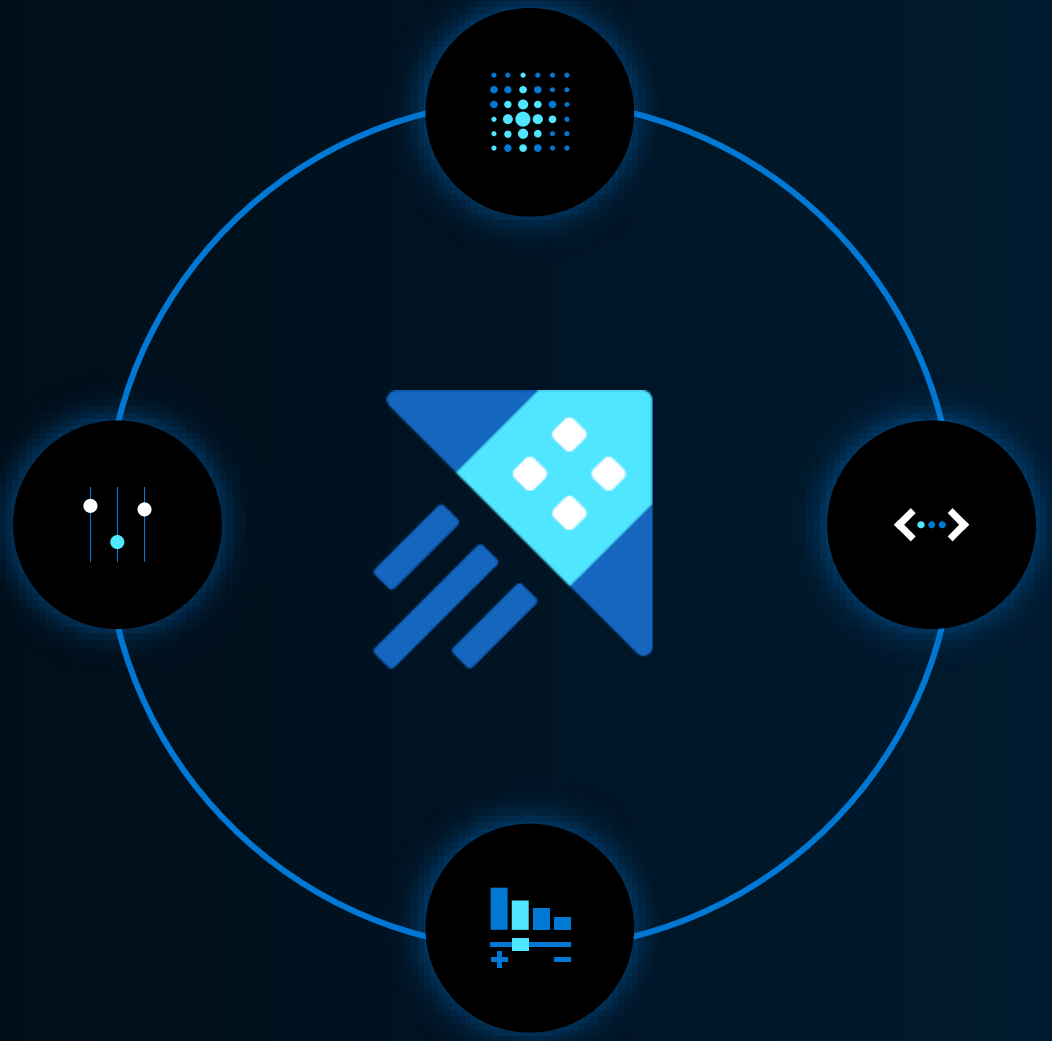


Agenda

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

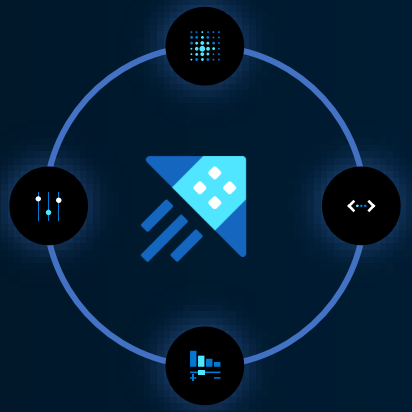
Intro

Module 1



Objectives

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

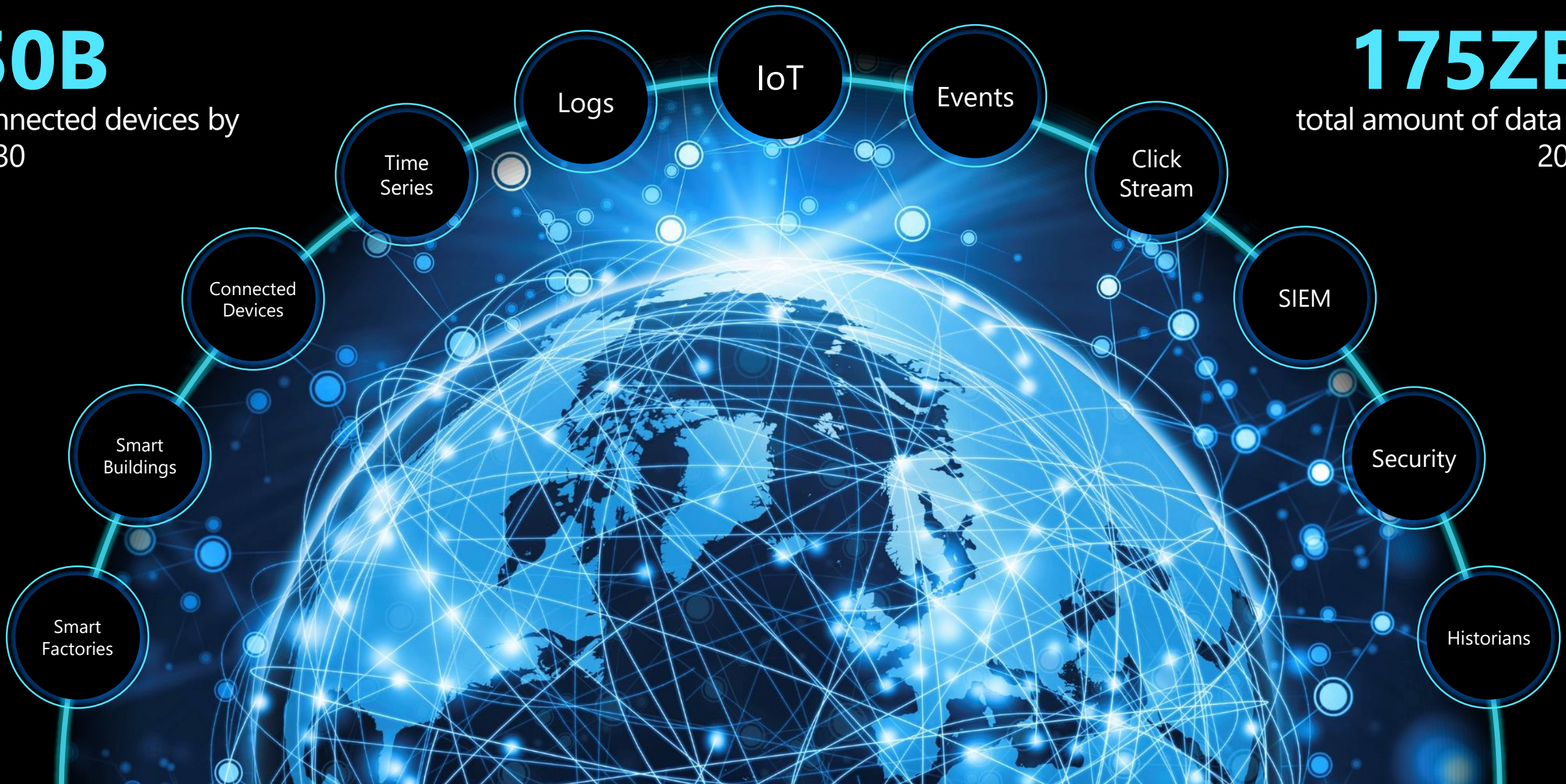


1. Observational analytics for telemetry workloads
2. What is ADX
3. Innovation & Proven technology
6. Pattern for real-time analytics

Telemetry - A key data for Digital Transformation

50B
connected devices by 2030

175ZB
total amount of data by 2025



Observational Analytics

- Frequently changing business questions
- Constantly changing schema
- Near real time visibility required
- Analytics systems costs are often prohibitive
- Looking for unpredictable phenomena

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

Now also in Synapse

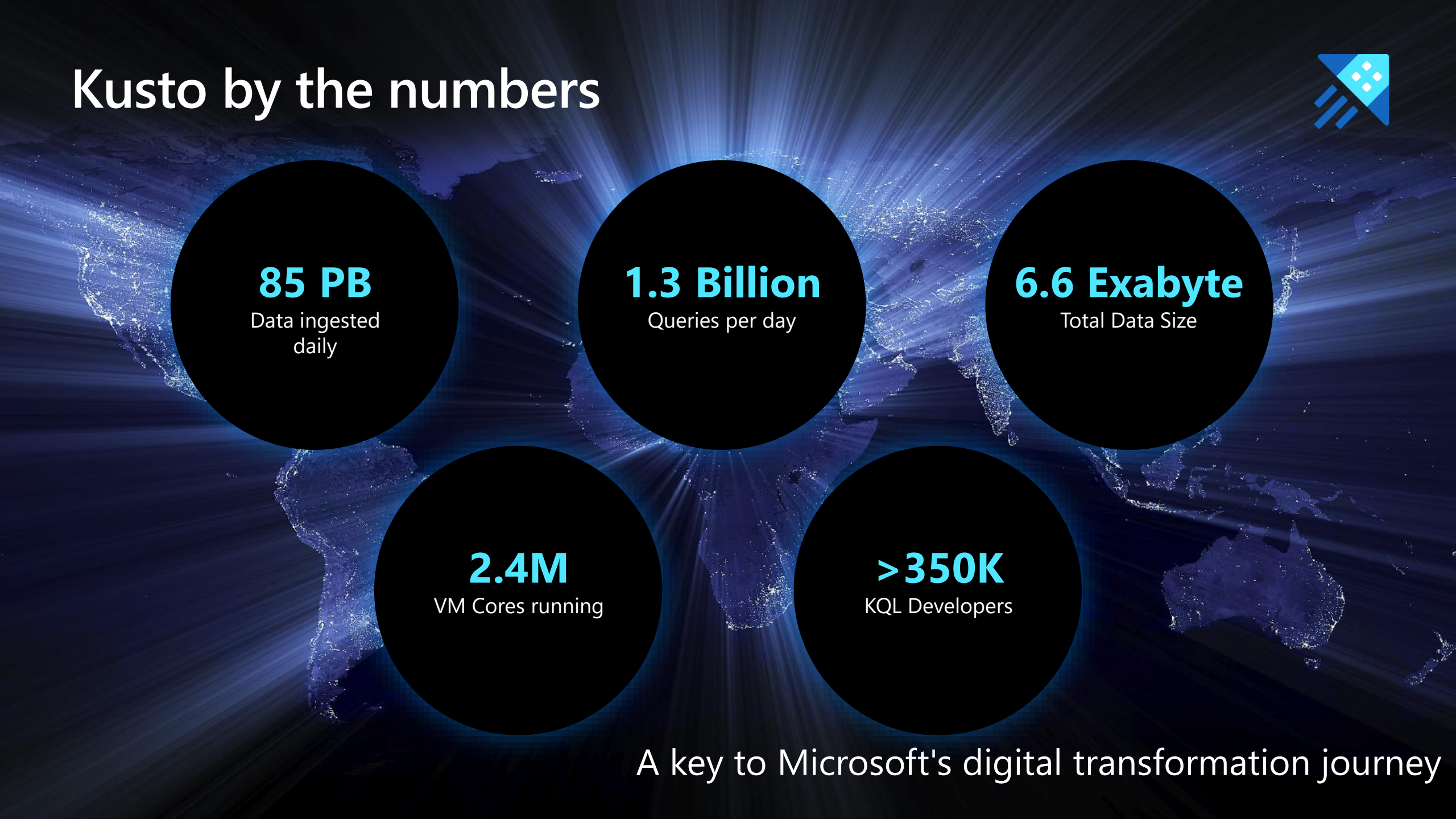
for time stamped data: logs, time series, telemetry

for free text, structured and semi-structured data

Near real-time data analytics at scale

for ad-hoc analytics

Kusto by the numbers



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

Windows, Skype, LinkedIn, Visual Studio, SQL, Bing, Microsoft Azure, Xbox, Power BI, Office

The platform for analytical solutions (SaaS)

Azure Monitor

Security Products

IoT

Gaming

Available as PaaS

esri, DocuSign, Check Point Software Technologies Ltd., Taboola, ZOOMD Know More, Do More., Ecolab, Bühler, agl, BASF We create chemistry, Grab

#PoweredByADX

Innovation Velocity

Analytical Power

- Engine v3
- Sequence Analytics (Scan)
- Geo Spatial

Data Management

- Materialized views
- Partitioning
- Parquet ingestion and query
- Soft delete
- Hot cache windows

Visualization

- Dashboard Import/export
- Dashboard Cross filter
- Better Kibana support

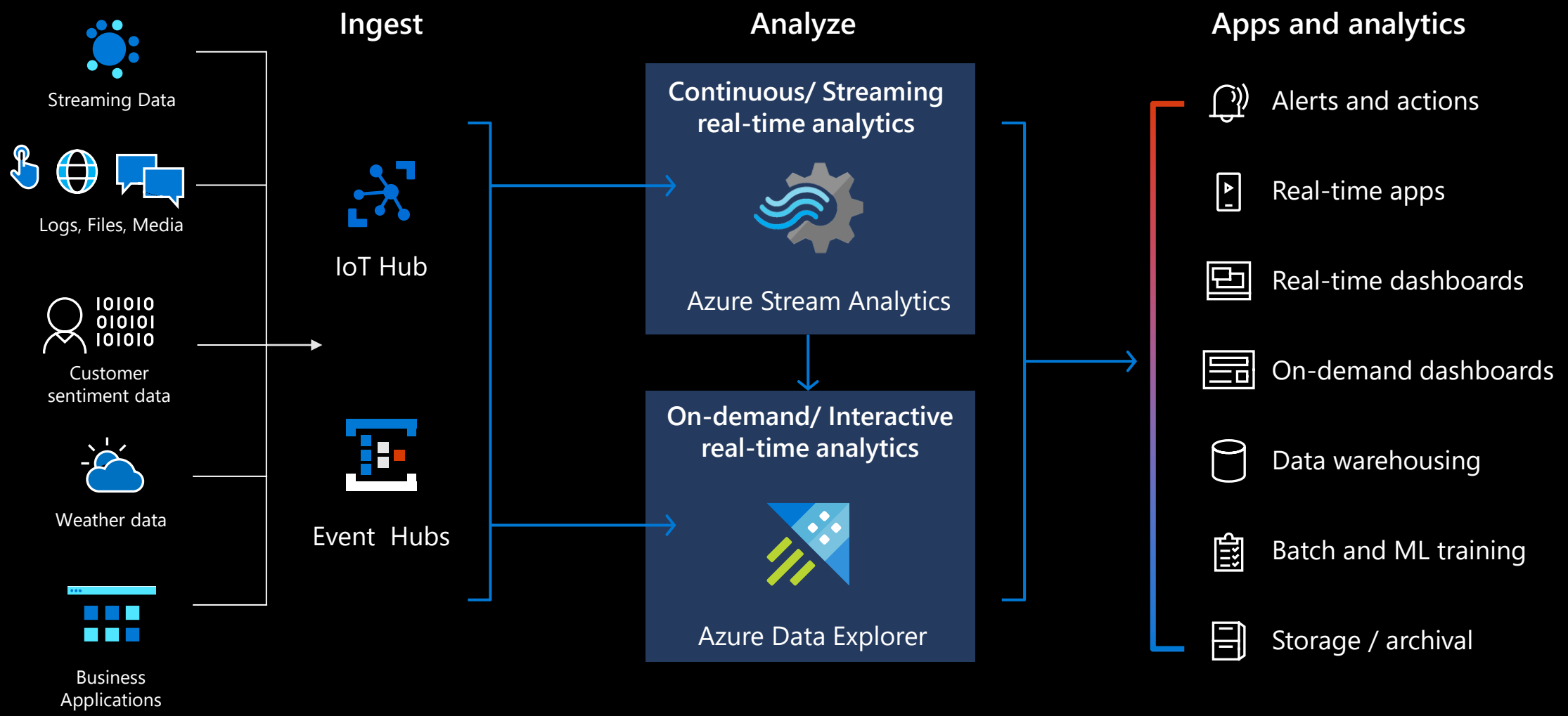
Operations

- Private endpoints
- Predictive Auto scale

Dev Experience

- Python & JavaScript

Azure powered pattern for real-time analytics



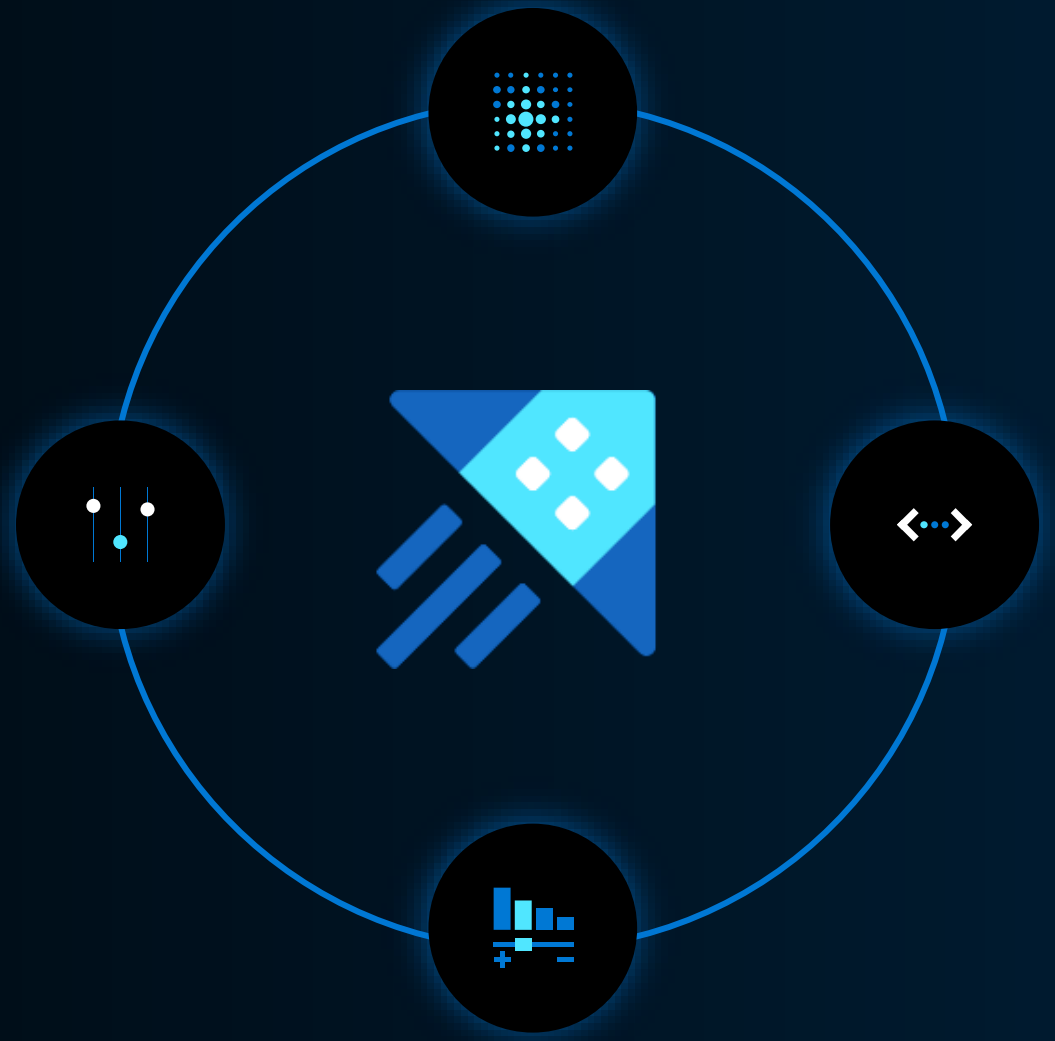
Demo*

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

* Estimate 5-15 minutes

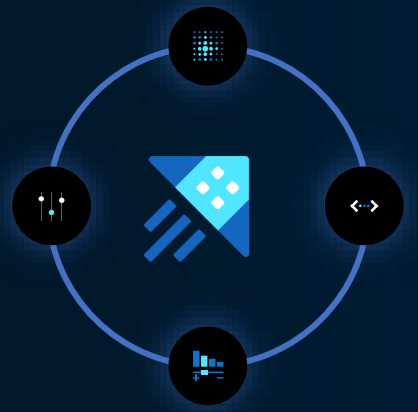
Overview

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
6. Web UI capabilities
7. Extensibility
8. Enterprise Ready

Common use cases across verticals

Retail

Web Analytics, IoT Analytics

Financial

Audit Logs

Oil/Gas & Energy

Industrial IoT, Historian, Time Series, Grid analytics

Security

Security Analytics, Threat Detection, SIEM

Healthcare

IoT device Analytics

Advertising

Personalized offers, campaign management

Media Entertainment

Content Delivery Network analytics, Viewer experience analytics, Live Event Analytics

Automotive

Manufacturing, Connected cars, Fleet management

Central Observability (Logs, Traces, Metrics) and Time Series

What is Azure Data Explorer?

- Fully managed
- High-performance
- Big data analytics platform
- Analyze high volumes of data in near real time
- End-to-end solution for data ingestion
- Query, visualization, and management
- Useful for log analytics, time series, IoT, and general-purpose exploratory

What makes ADX unique?

- Data velocity, variety, and volume
- User-friendly query language
- Advanced analytics
- Easy-to-use wizard
- Versatile data visualization
- Automatic ingest, process and export

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?

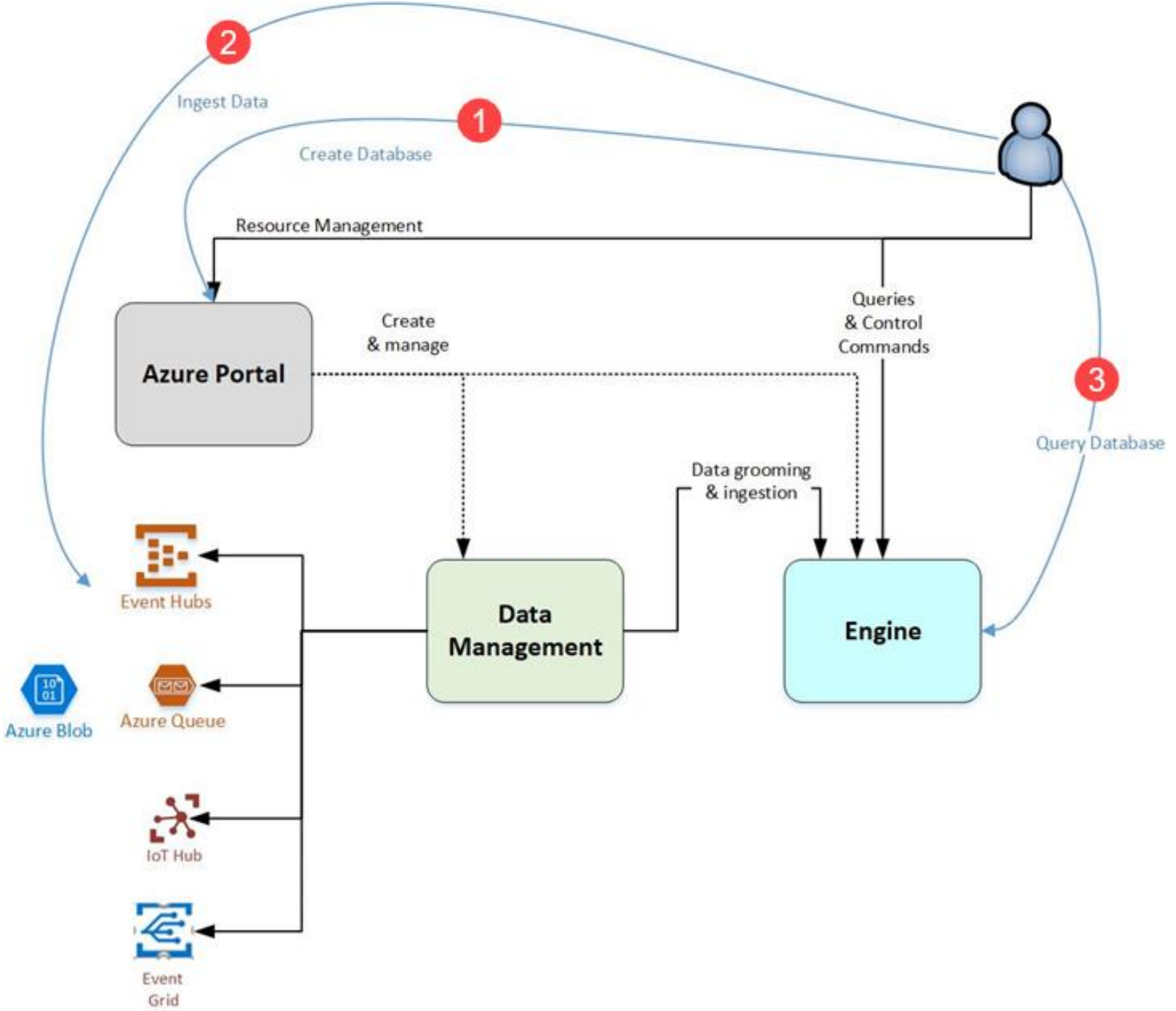
- Each ADX cluster can hold upto **10,000** databases and each database upto **10,000** tables.
- The data in each table is stored in data shards also called "**extents**".
- All data is **automatically indexed** and partitioned based on the ingestion time.
- There are no primary foreign key constraints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

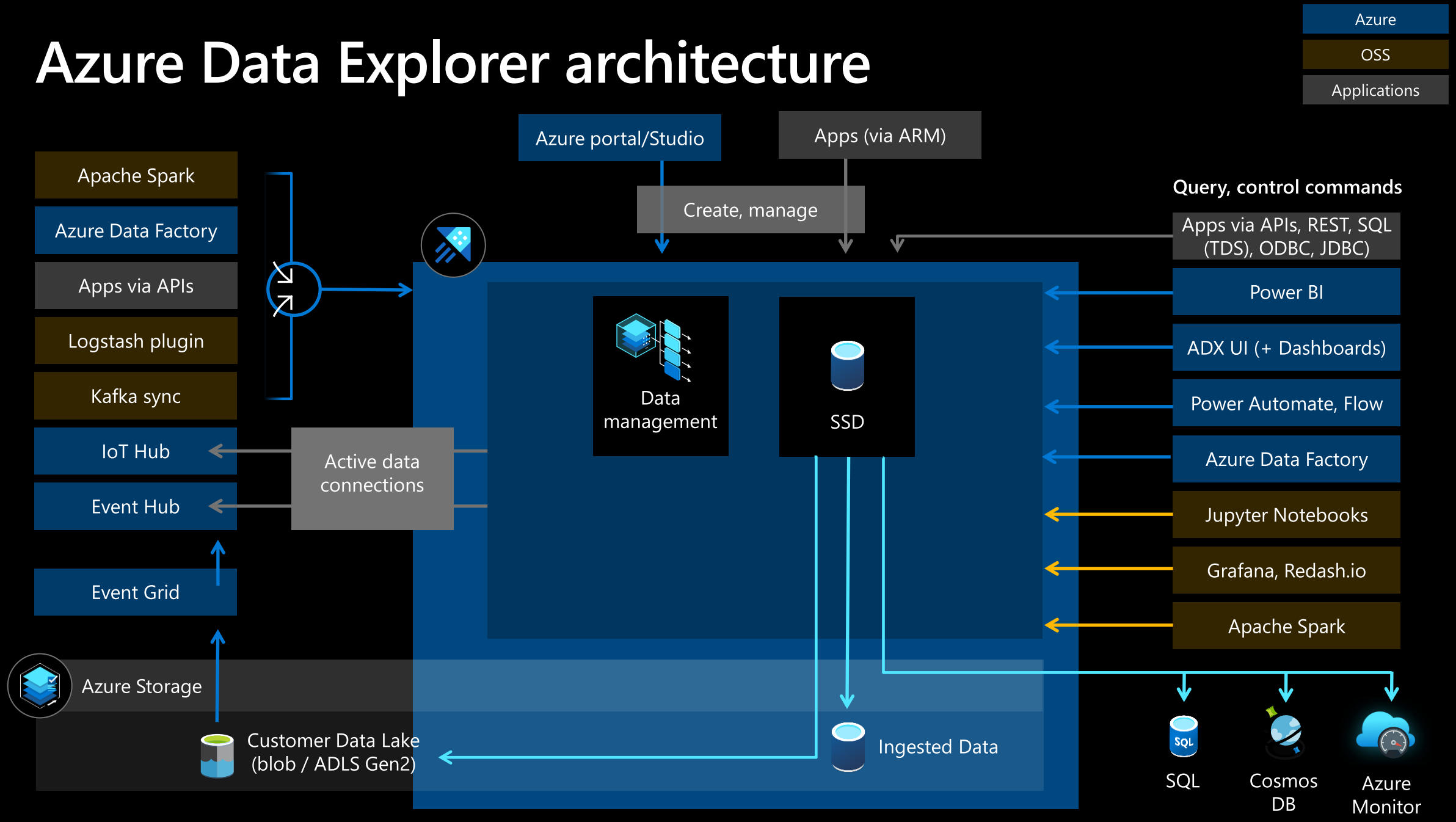
- **Tables**: Made up of a set of columns. Each column has one of nine different data types.
- **External tables:** Tables whose underlying storage is in other locations such as Azure Data Lake.

Working with ADX

1. Create Database
2. Ingest Data
3. Query Database
4. Visualize results

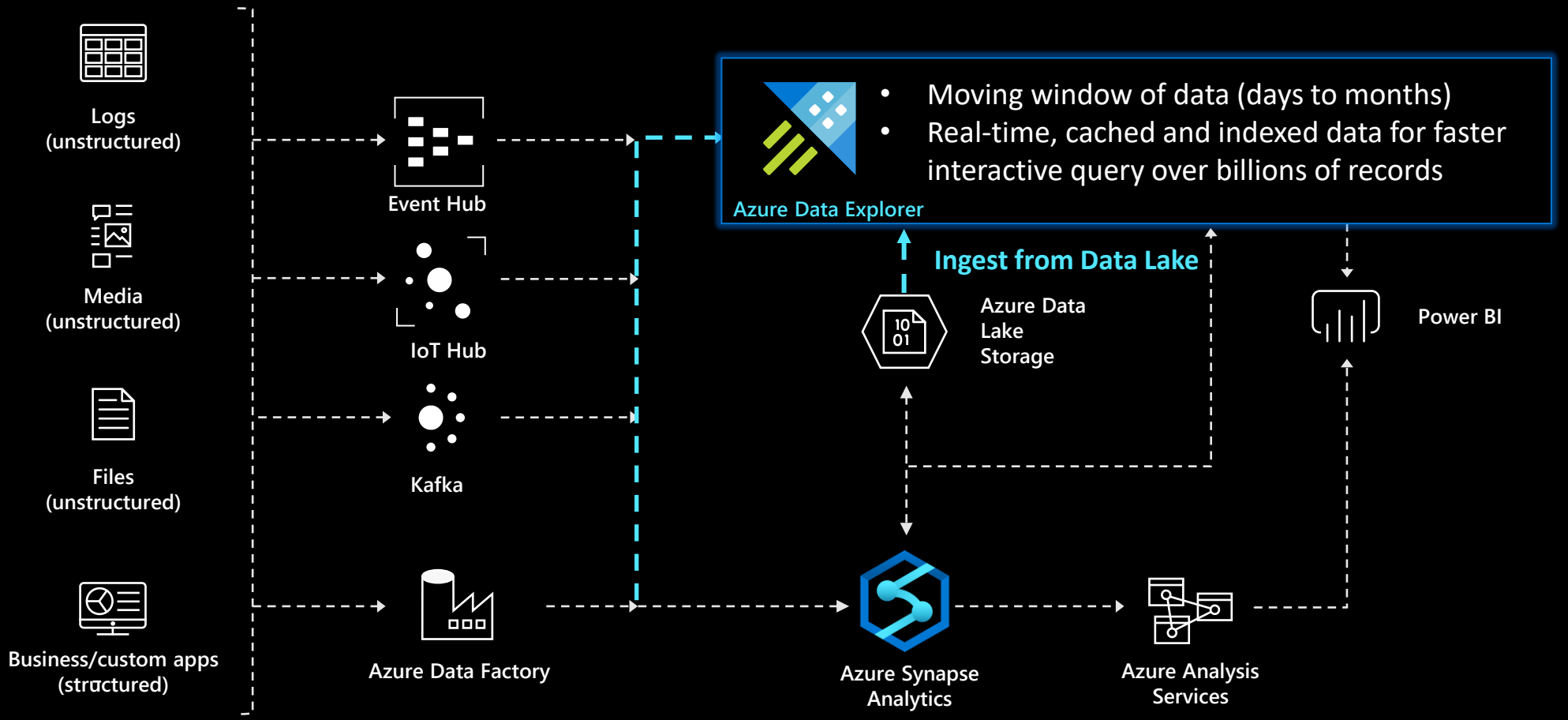


Azure Data Explorer architecture



Integration: Batch Ingestion

Ingest data from Data Lake or other sources in batch mode



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

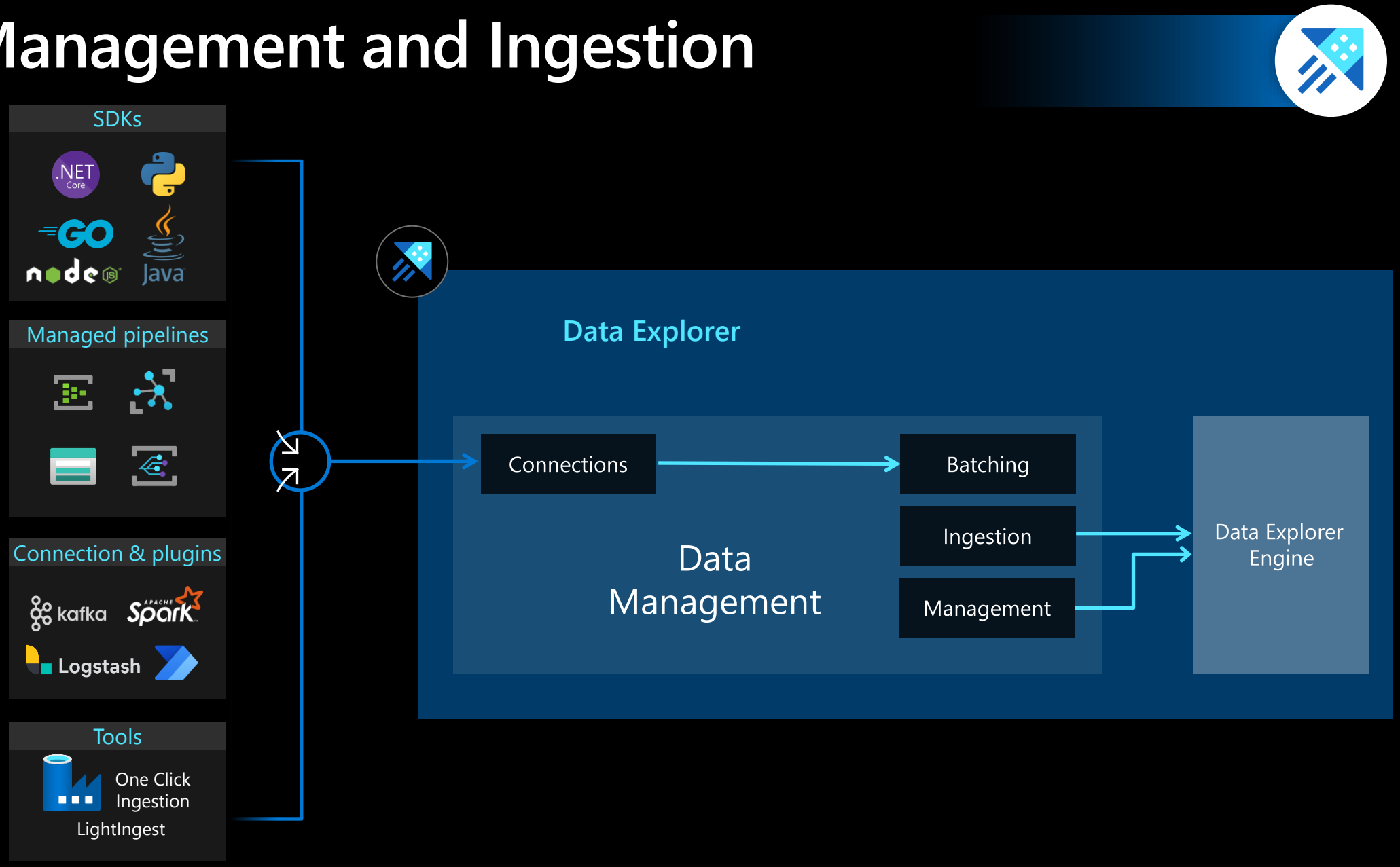
Streaming

- 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
- Steaming 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

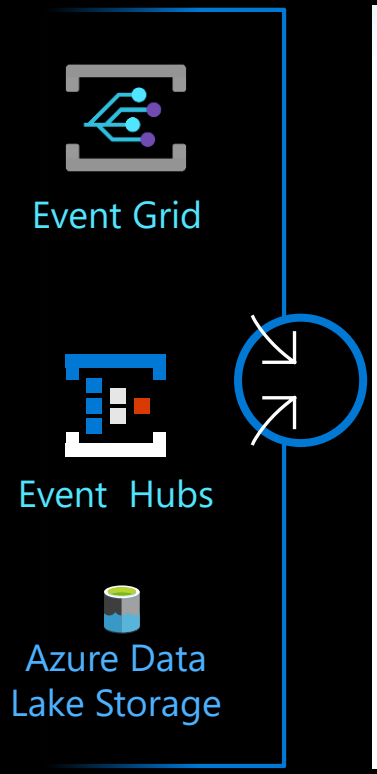
Data Management and Ingestion



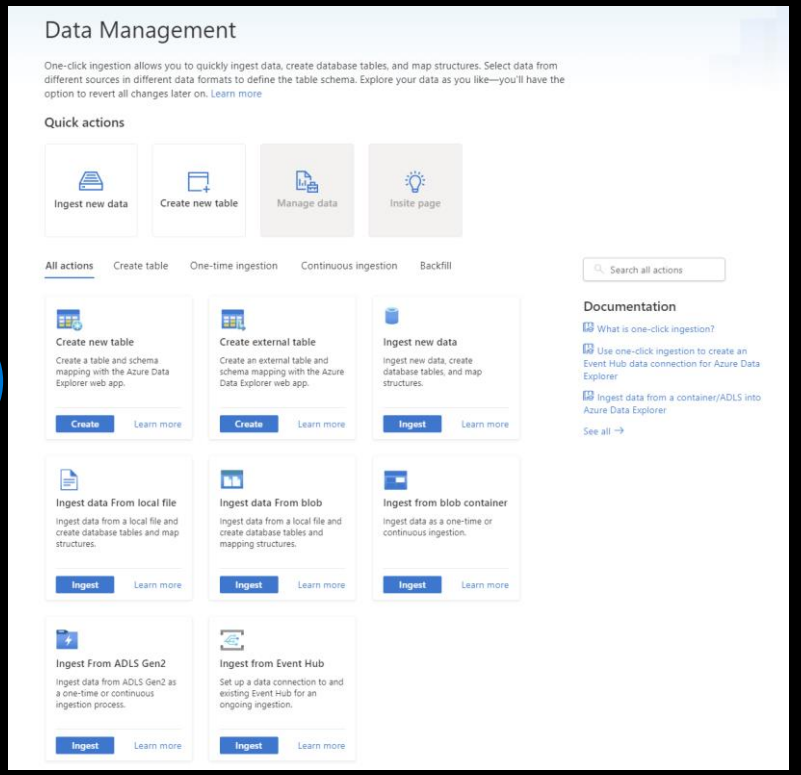
ADX Web UI One-Click (wizard)

Friction free on-boarding of your Azure data sources to Data Explorer

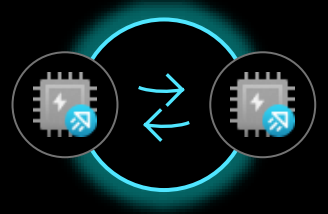
1. Get data from Azure Data Lake, Blob, Event Hub, or local file
2. Infer the schema automatically from source
3. Create table and mapping automatically from source
4. Generate custom code to start ADX project with one of the supported SDKs
5. Manage data-policies (e.g. retention, batching policies, streaming)
6. Get insight on your data management
7. Ingest one-time data or create connection for continuous data ingestion
   - Continuous (EventHub/EventGrid), one time or backfill (LightIngest)



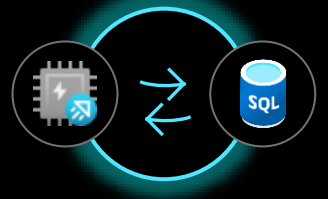
Data Explorer



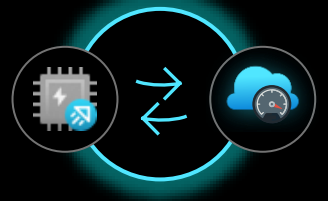
Cross Queries



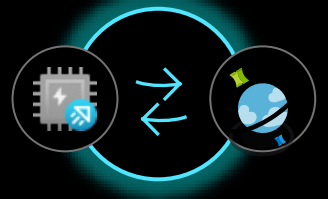
Between Data Explorer databases and clusters



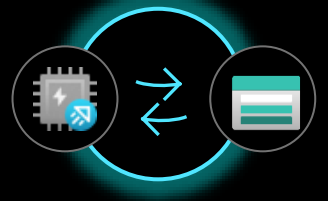
Query SQL pool from Data Explorer



Query Azure Monitor from Data Explorer



Query Cosmos DB from Data Explorer



Query Data Lake from Data Explorer

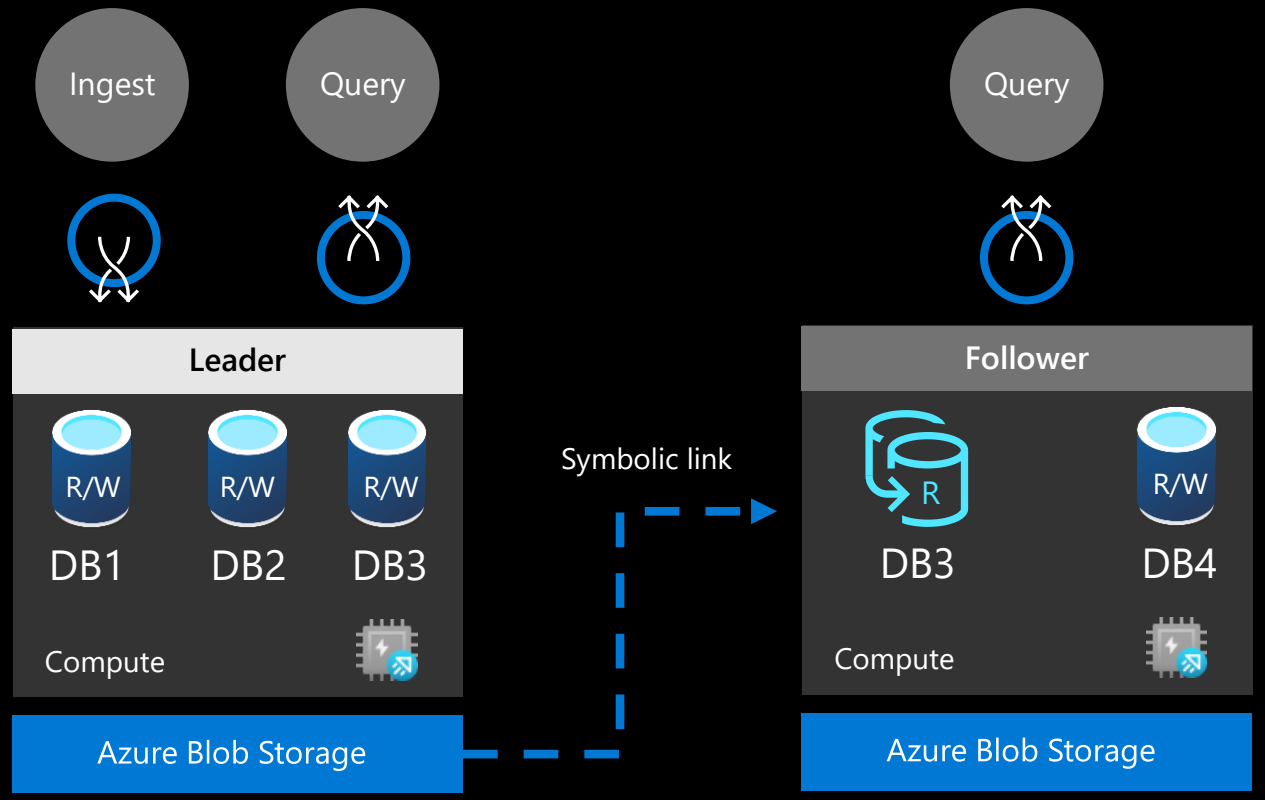
Data Sharing

Share data within the company

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

No Maintenance

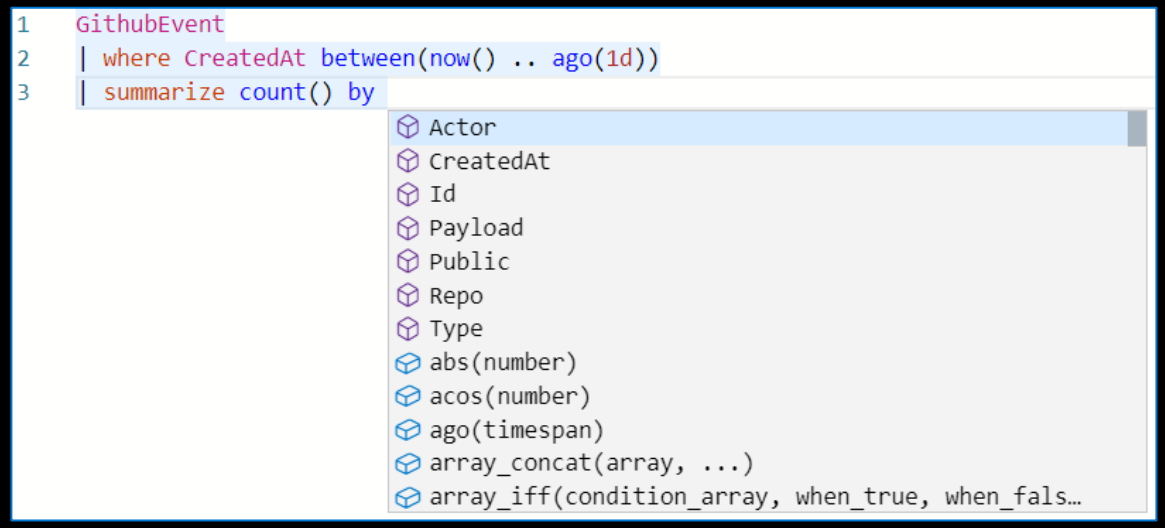
- In-Place Data Sharing
- No data pipeline to maintain
- Near real-time updates



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



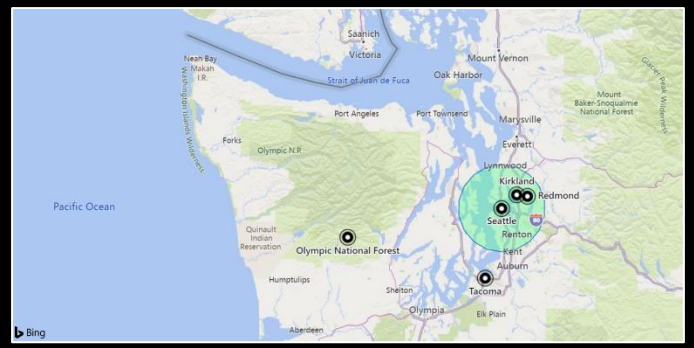
Extensible

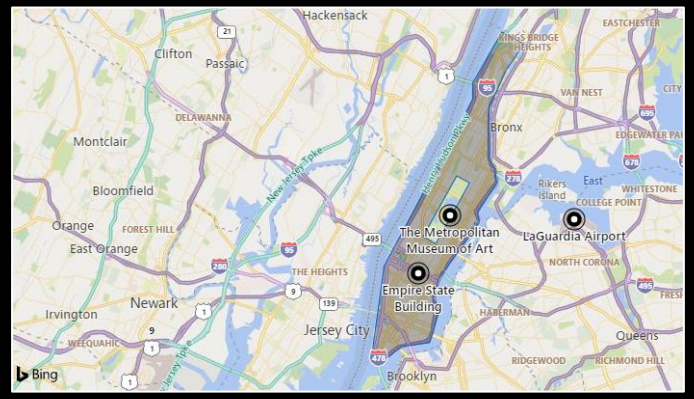
- In-line Python and R
- T-SQL

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

Distributed Custom Code Execution

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

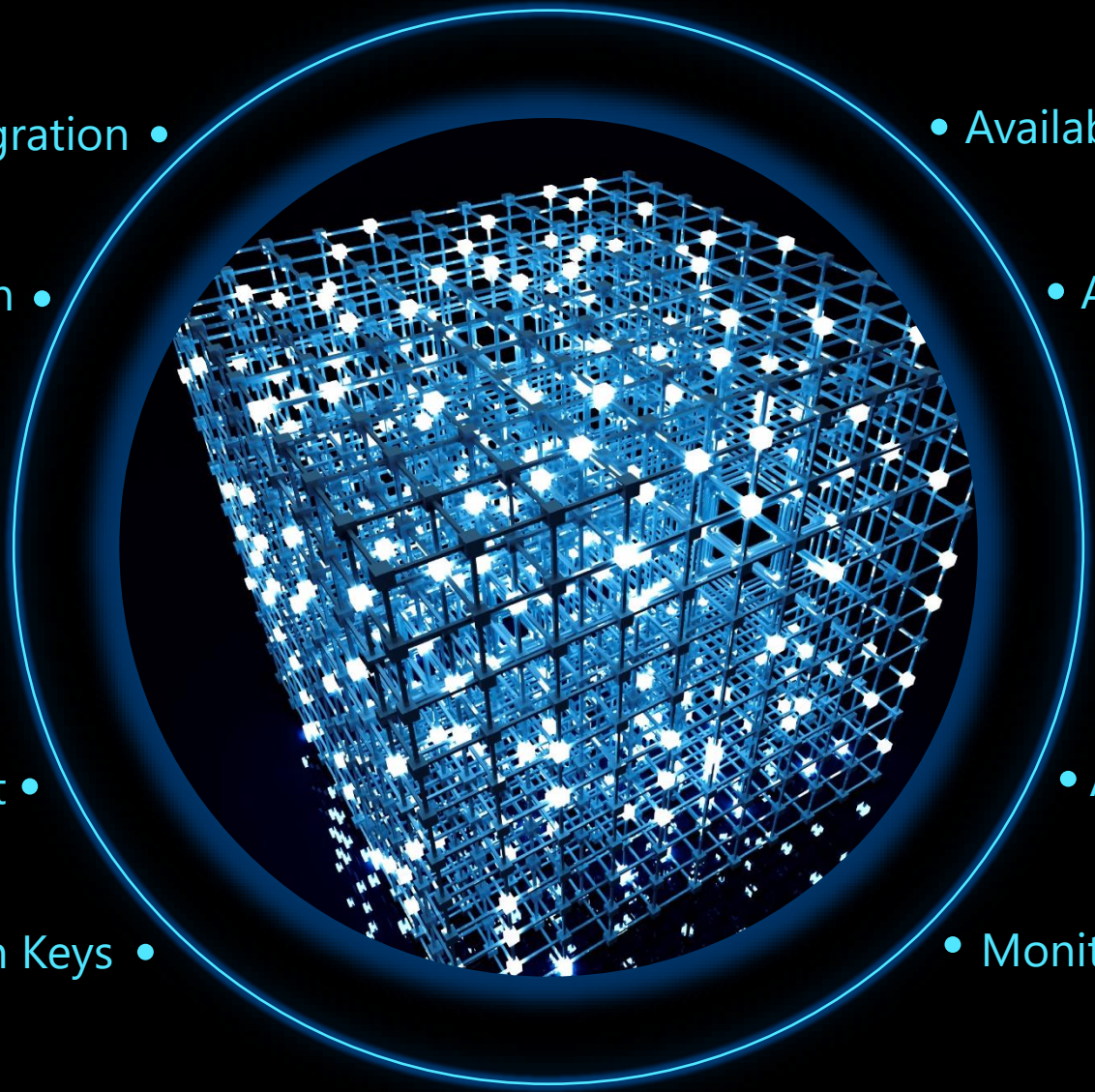
Spark Integration

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

Tools

- Jupyter Integration with KQL Magic
- Python, Java SDKs

Enterprise Ready – Mission Critical



- Azure Active Directory Integration
- Role based authorization
- Network Perimeter Security
- Encryption with customer managed keys/"BYOK"
- Build Azure Policy Support
- Bring Your Own Keys
- Availability Zones
- Auto Scale-Up/In
- Globally available
- CI/CD Integration
- Automated provisioning
- Monitoring

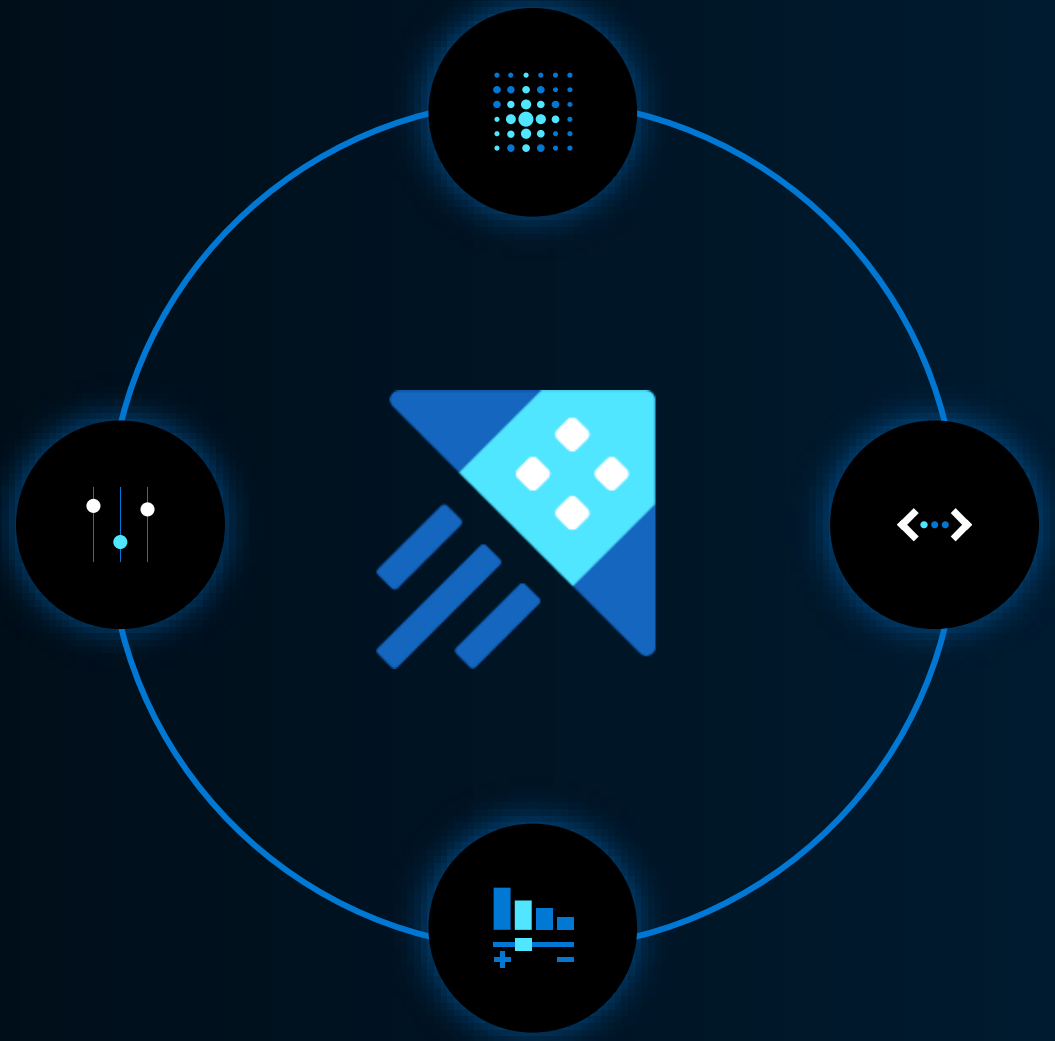
Demo*

- Transforming IoT Data with KQL

* Estimate 10-15 minutes

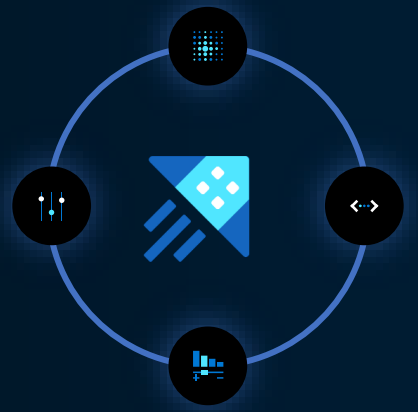
Architecture

Module 1



Objectives

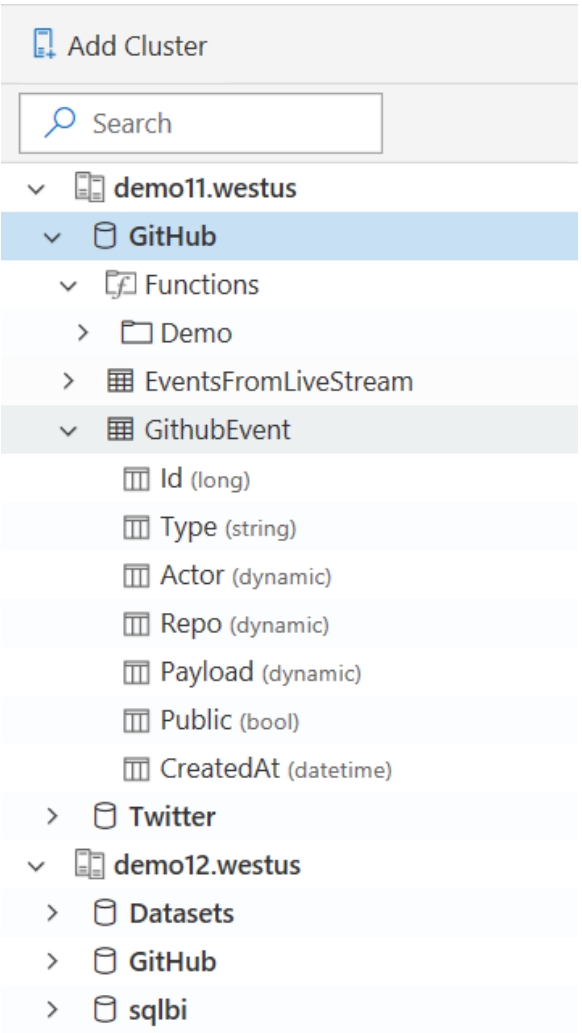
After completing this Learning, you will be able to understand:



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

Schema

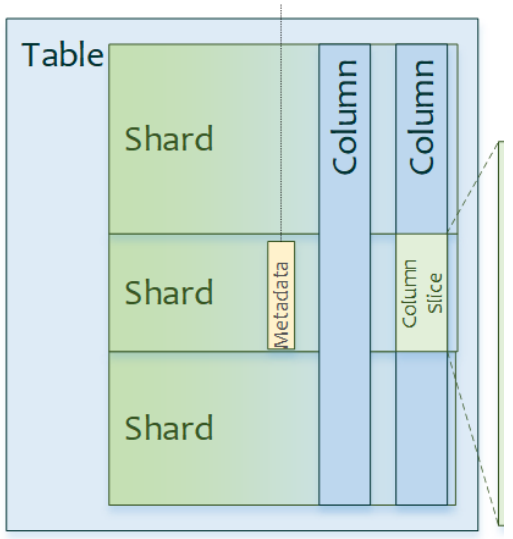
- Schema is
  - Relational, Lightweight, Dynamic
- Databases
  - Authorization boundary
  - Transaction boundary
  - But not query boundary!
    - Supporting cross-database and cross-cluster queries
- Tables
  - Rectangular
- Columns
  - Supported types: Boolean, integer, real, date/time, timespan, string, dynamic (JSON)
- Stored functions



Data Storage – Column Store

- Sharded columnar store
- Data is organized in immutable Extents
  - Append-only, no updates, deletes only for GDPR
- All columns are compressed
- All columns are indexed
  - Indexes can be trivial (e.g., a date/time index is just min/max values)
  - Indexes can be very complex (e.g., a full substring index for string columns allowing complex and efficient match operations)

Data Storage – Extents



- The data for each table is held in multiple extents
  - Extents are immutable and autonomous
  - All operations work on complete extents

- Data is append-only
  - Data deleted by dropping a whole data shard (bulk delete)
  - Purge support: Re-create existing shards by query + ingestion

- Data sharding scheme
  - During ingestion, data is sharded by time-of-arrival & size

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

Data management Service & Ingestion

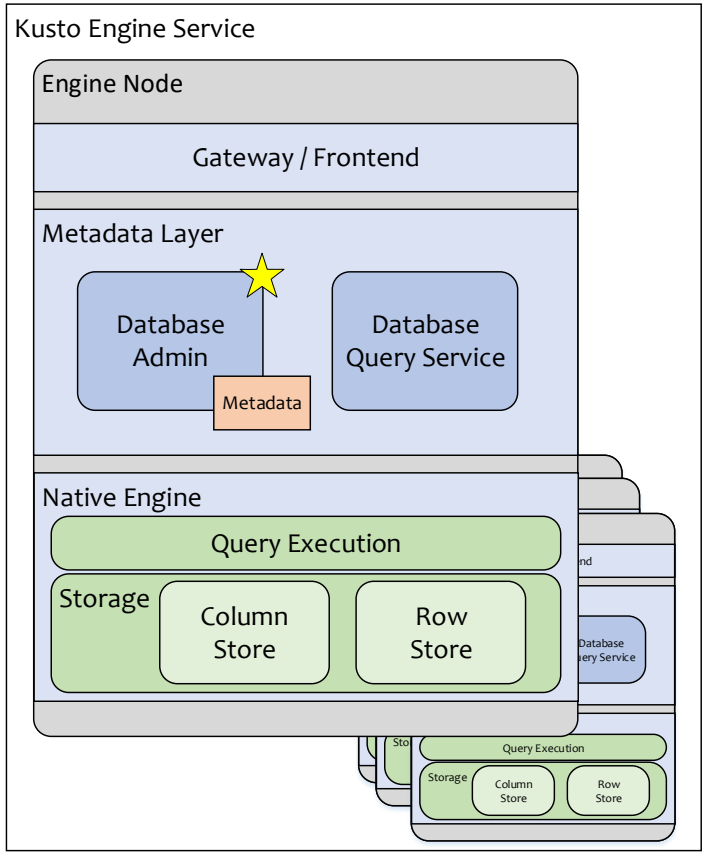
- Data Formats:
  - CSV, TSV, SOHSV, SCSV, JSON, AVRO, PARQUET, ORC, TXT

- Parsing, transposing, optional sorting, indexing, and compression
  - Ingestion rate - ~150-200MB/s per node (D14 node, CSV)

- Data management (DM) cluster
  - The ingestion service
  - Used to “adapt” any data source to Kusto
  - EventHub, IoT Hub, EventGrid, …
  - Ingestions are batched by size and/or time
  - Integration with Row-Store
  - Triggers periodic data management tasks (retention, merge extents commands, etc.)

Engine Service

- Single node type
- All nodes are symmetrical
  - Any node can take over when a node is down
- A single node is assigned to be the Admin node per database
  - Manages the database metadata (all metadata writes)
  - Creates the query plan based on the metadata (*usually)
  - Query plan is distributed to all nodes
- All nodes participate in query execution – Tail nodes
  - Extents are assigned to nodes using consistent hashing
  - Scale in/out reshuffles extents between nodes
  - Shuffling is minimized



How does the Kusto Query Language work?

- KQL is an expressive, intuitive, and highly productive query language.
- It offers a smooth transition from **simple** one-liners to **complex** data processing scripts, and supports querying structured, semi-structured, and unstructured (text **search**) data.
- There is a wide variety of query language **operators** and **functions** (aggregation, filtering, time series functions, geospatial functions, joins, unions, and more) in the language.
- KQL supports cross-cluster and cross-database queries, and is feature rich from a parsing (JSON, XML etc.) perspective.
- In addition, the language **natively** supports advanced analytics.

Query

Standard query pipeline:

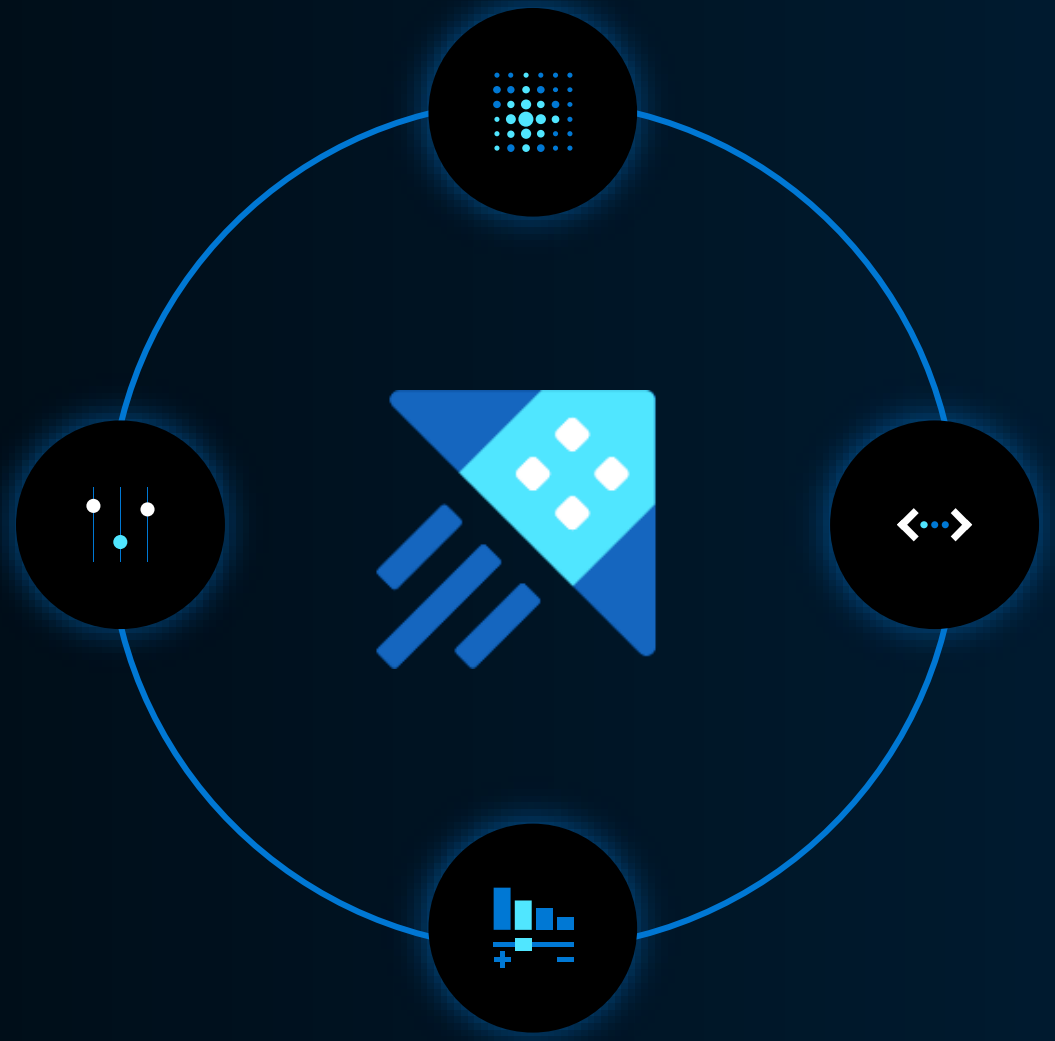
- Parsing
- Semantic analysis
- Optimizations
  - Prefiltering
  - Rewrite rules
- Execution
  - Relational operators
  - Storage

The Data Management Service

- Queued data ingestion
  - Batching
  - Executing ingest commands, utilizing engine capacity
  - Progress monitoring & handle transient failures
  - Status reporting
  - Data connections
- Extents merge
- Purge scheduling & cleanup
- Data retention

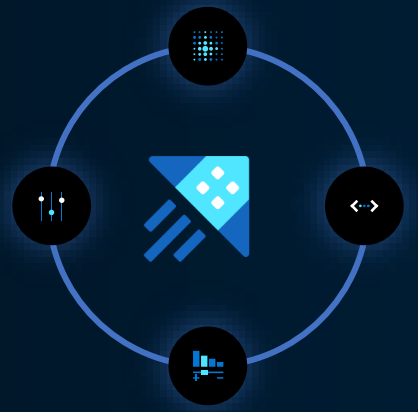
Ingestion

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

- Streaming ingestion allows near real-time latency for small sets of data per table.
- Data is ingestion to row store. In the background, data is later moved to column store extents.
- Streaming ingestion mode is enabled via the portal and applied on tables where policy is set.

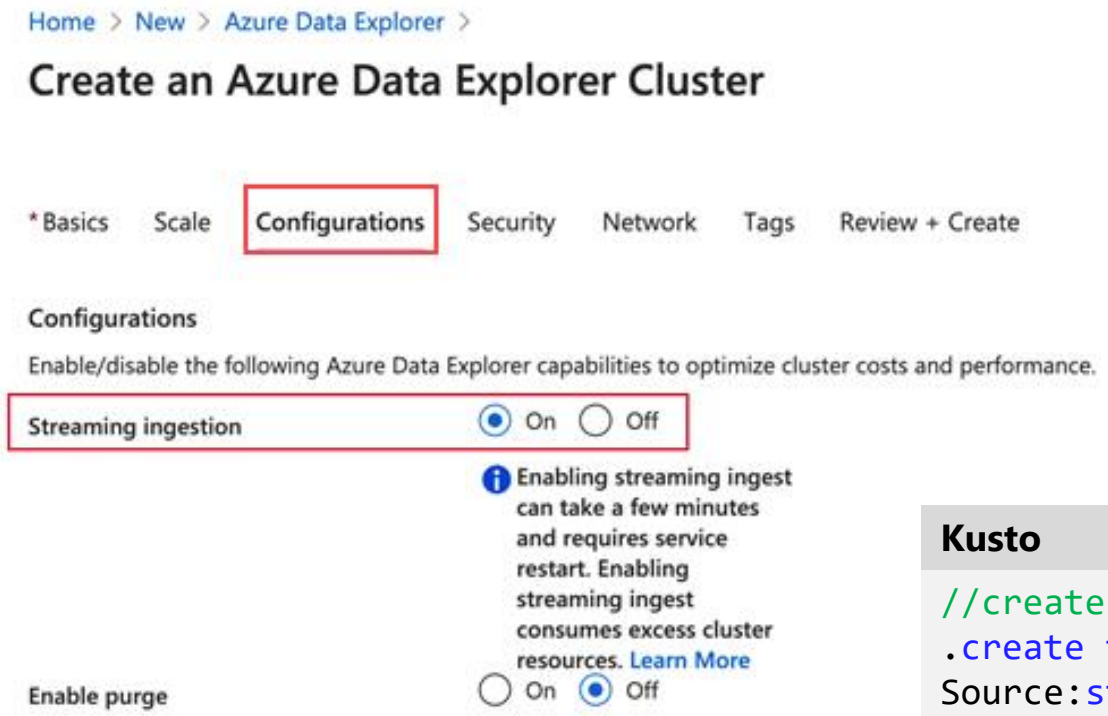
- Streaming ingestion types
  - Custom ingestion using one of the ADX client libraries
  - Using Event Hub as a data source
- Streaming ingestion policy
  - Enable on cluster / define on database / table, inherited

```
.alter database StreamingTestDb policy streamingingestion enable
```

```
.alter table TestTable policy streamingingestion enable
```

  - Number of row stores should be scaled up as the data rate increases
  - If not defined, data **batching** is applied

Streaming Ingestion



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);
}
```

Connectors and Plugins

- ADX Connector for Apache Spark
  - Open-source project: https://github.com/Azure/azure-kusto-spark
  - The connector for Spark implements data source and data sink for moving data across ADX and Spark clusters to use both of their capabilities.

- ADX Kafka Sink
  - Opensource project: https://github.com/Azure/kafka-sink-azure-kusto
  - Micro-batching and ingesting data using Java Kusto Ingest SDK

- Logstash
  - The Logstash output plugin communicates with Azure Data Explorer and sends the data to the service.

- FluentBit
  - Opensource project: https://docs.fluentbit.io/manual/pipeline/outputs/azure_blob
  - Send logs to Blob and use Event Grid to ingest to Data Explorer

- Telegraf: https://github.com/influxdata/telegraf/tree/master/plugins/outputs/azure_data_explorer

Connectors and Plugins

- Azure Data Flow connector
  - The Microsoft Flow Azure Kusto connector allows users to run Kusto queries and commands automatically as part of a scheduled or triggered task

- Azure Data Factory (ADF)\Synapse Pipelines
  - A cloud-based data service that allows you to integrate different data stores and perform activities on the data. Supported activities for ADX: Copy, Lookup, Command
  - Copy in bulk from a database template
  - Mapping Data Flow

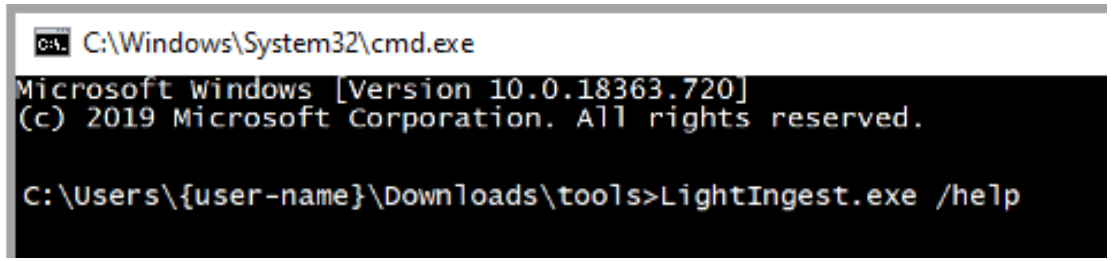
Tools

Light Ingest

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

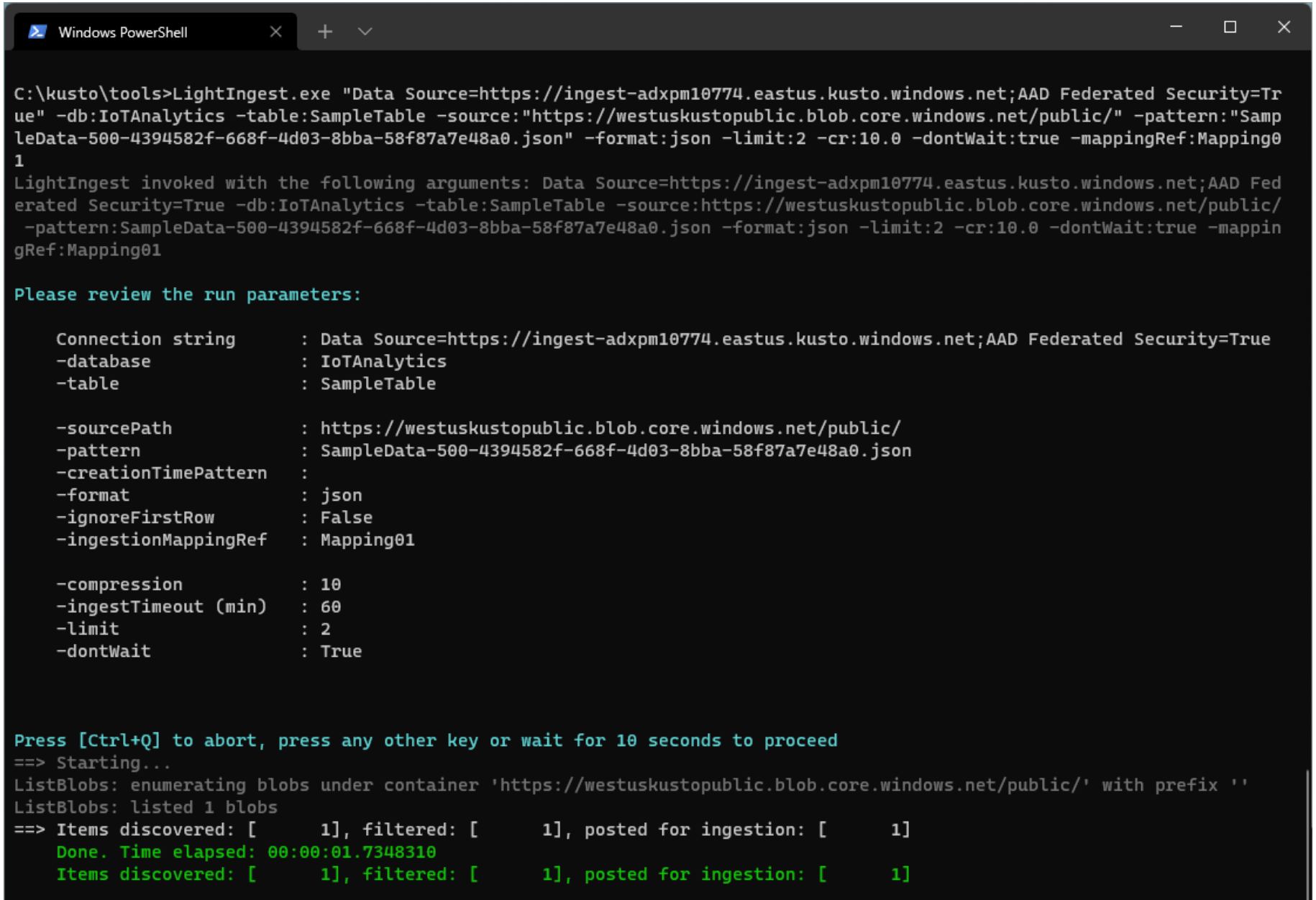
```
LightIngest "Data Source=https://ingest-demo.westus.kusto.windows.net;AAD Federated
Security=True" -db:demo -table:Trips -source:"" -pattern:"*.csv.gz" -format:csv -limit:2
-ignoreFirst:true -cr:10.0 -dontWait:true

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
```



See More: [How to ingest data using CreationTime]

Light Ingest CLI



Demo*

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

* Estimate 5-10 minutes

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.

Product Links

Product

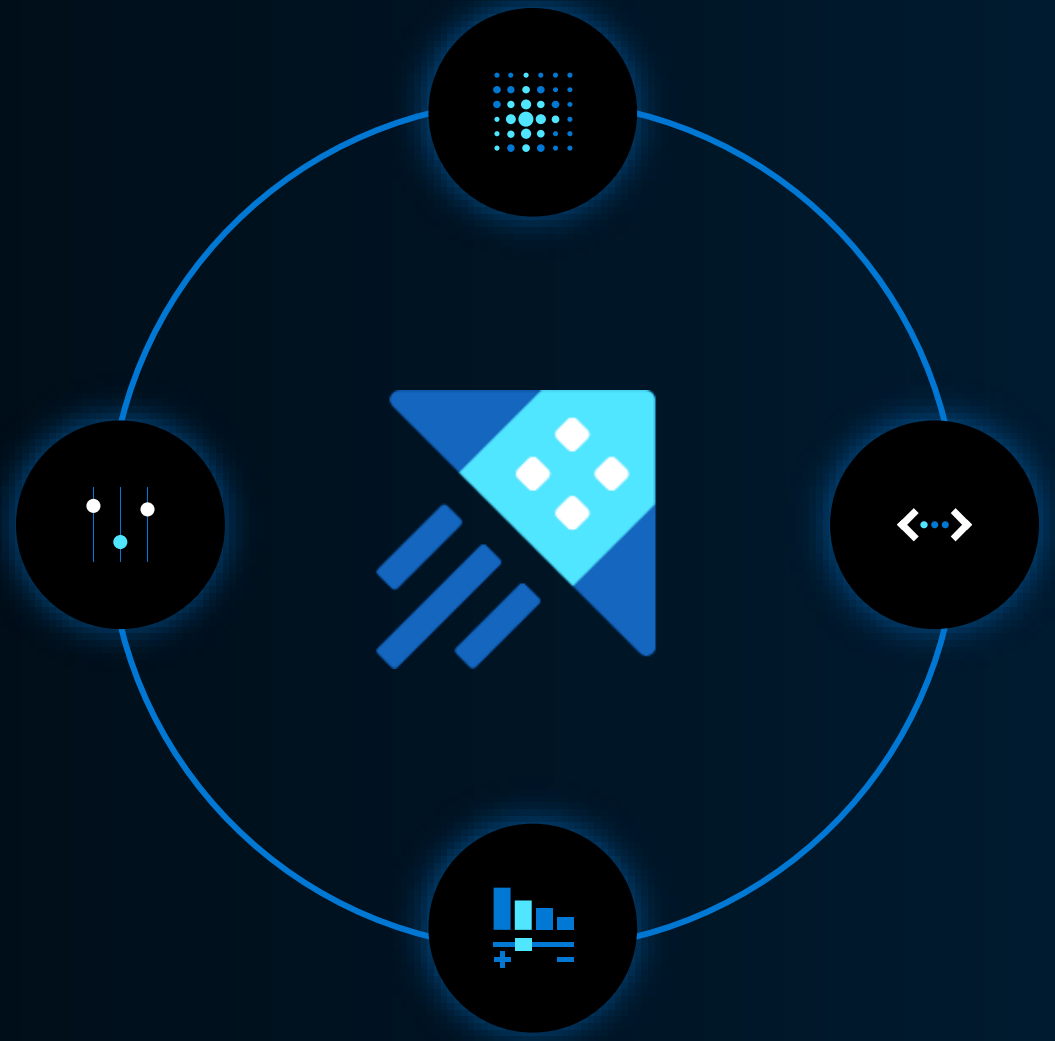
- Product Page: http://aka.ms/AzureDataExplorer
- Docs: https://aka.ms/adx.docs
- Cost Estimator: http://aka.ms/adx.cost
- Free Courses: KQL from Scratch, Azure data exploring, How to start with Azure Data Explorer (blog), Advanced KQL (blog)
- Lab: https://aka.ms/adx.lab
- Whitepaper: https://azure.microsoft.com/en-us/resources/azure-data-explorer/en-us/
- 101 blog: https://azure.microsoft.com/en-us/blog/azure-data-explorer-technology-101/
- Trial Cluster - http://aka.ms/adx.try
- Reference Architectures - https://aka.ms/adx.architectures

Social and Community

- Twitter: @AzDataExplorer
- Tech Community Blog: https://aka.ms/adx.blog
- Tech Community Forum: http://aka.ms/adx.techcommunity
- Stack overflow: https://aka.ms/adx.sof
- YouTube Channel: https://aka.ms/adx.youtube
- Make product suggestions: https://aka.ms/AzureDataExplorer.UserVoice

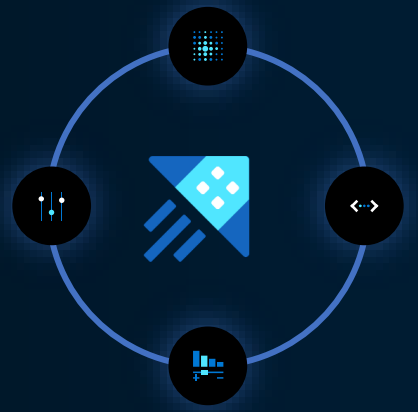
IoT Analytics

Module 2



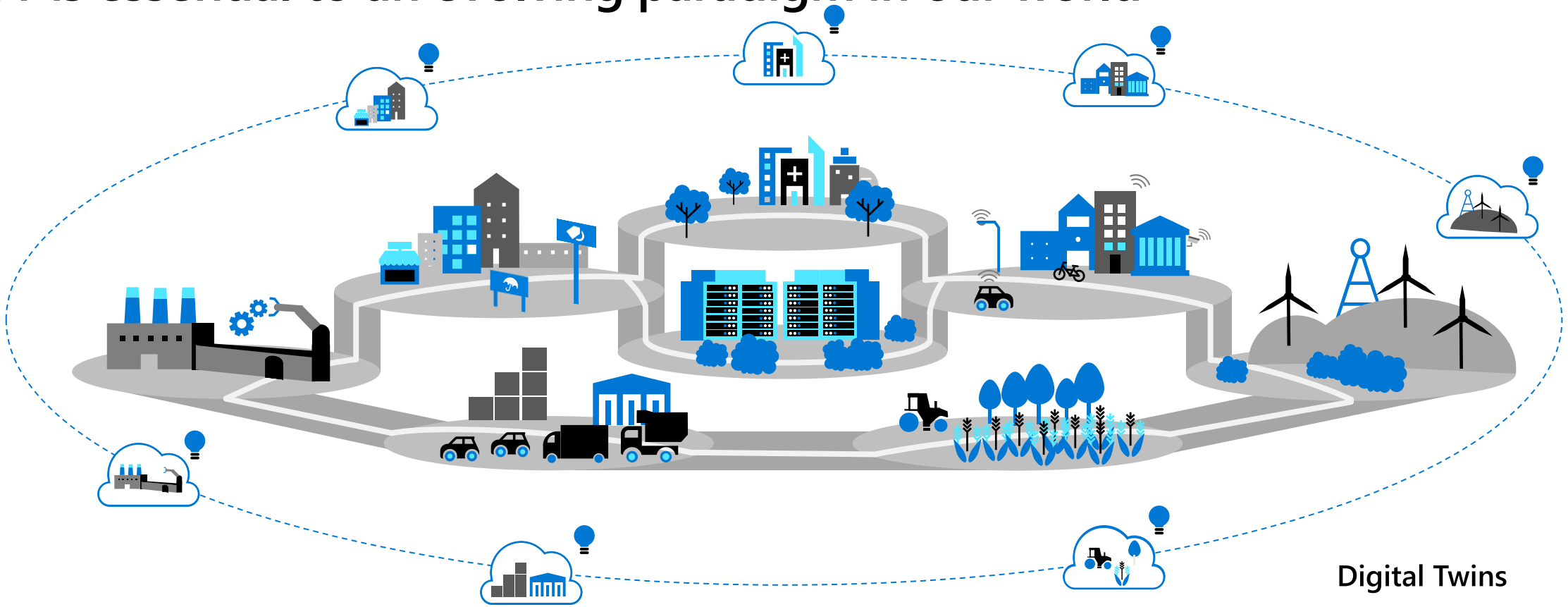
Objectives

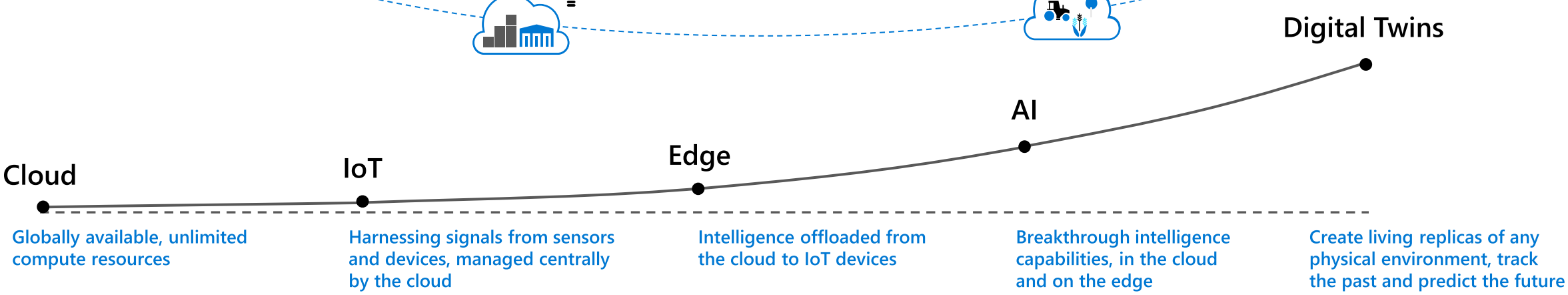
After completing this Learning, you will be able to understand:



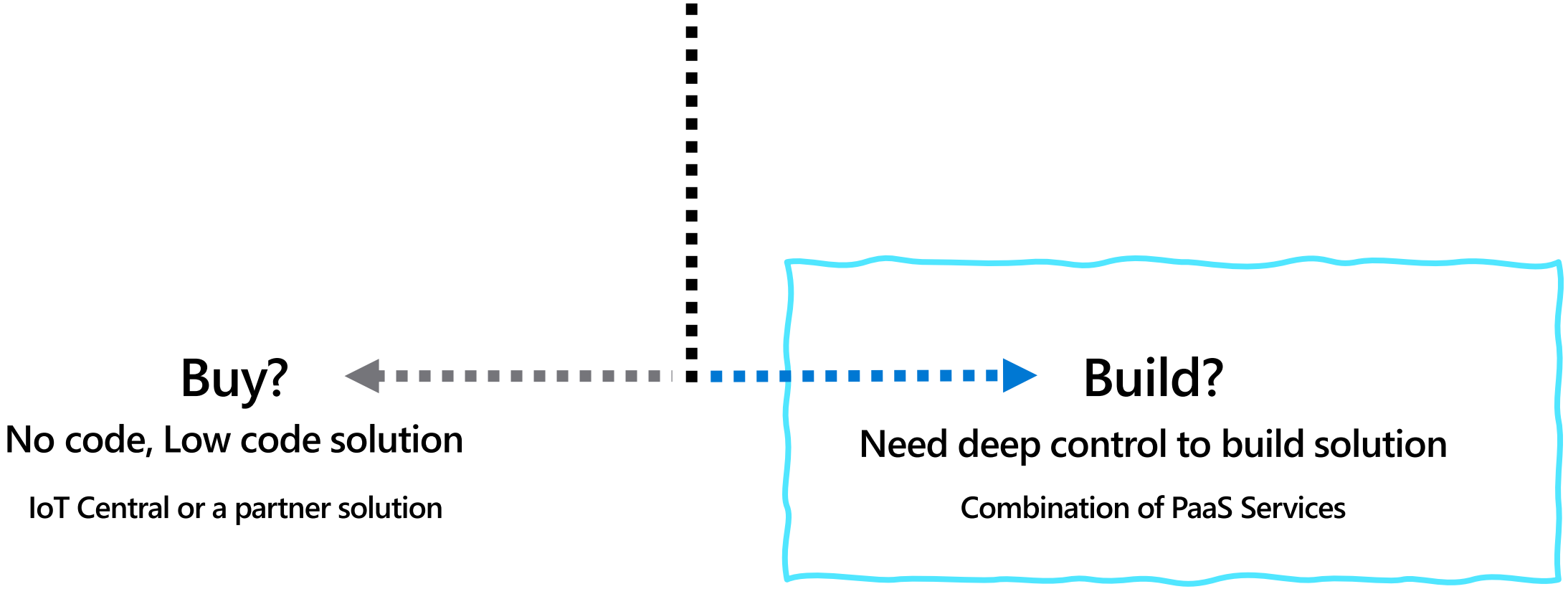
1. Why IoT is essential
2. Build solution using Platform as a Service (PaaS) in Azure
3. IoT reference architecture
4. End-to-end demo for IoT analytics

IoT is essential to an evolving paradigm in our world

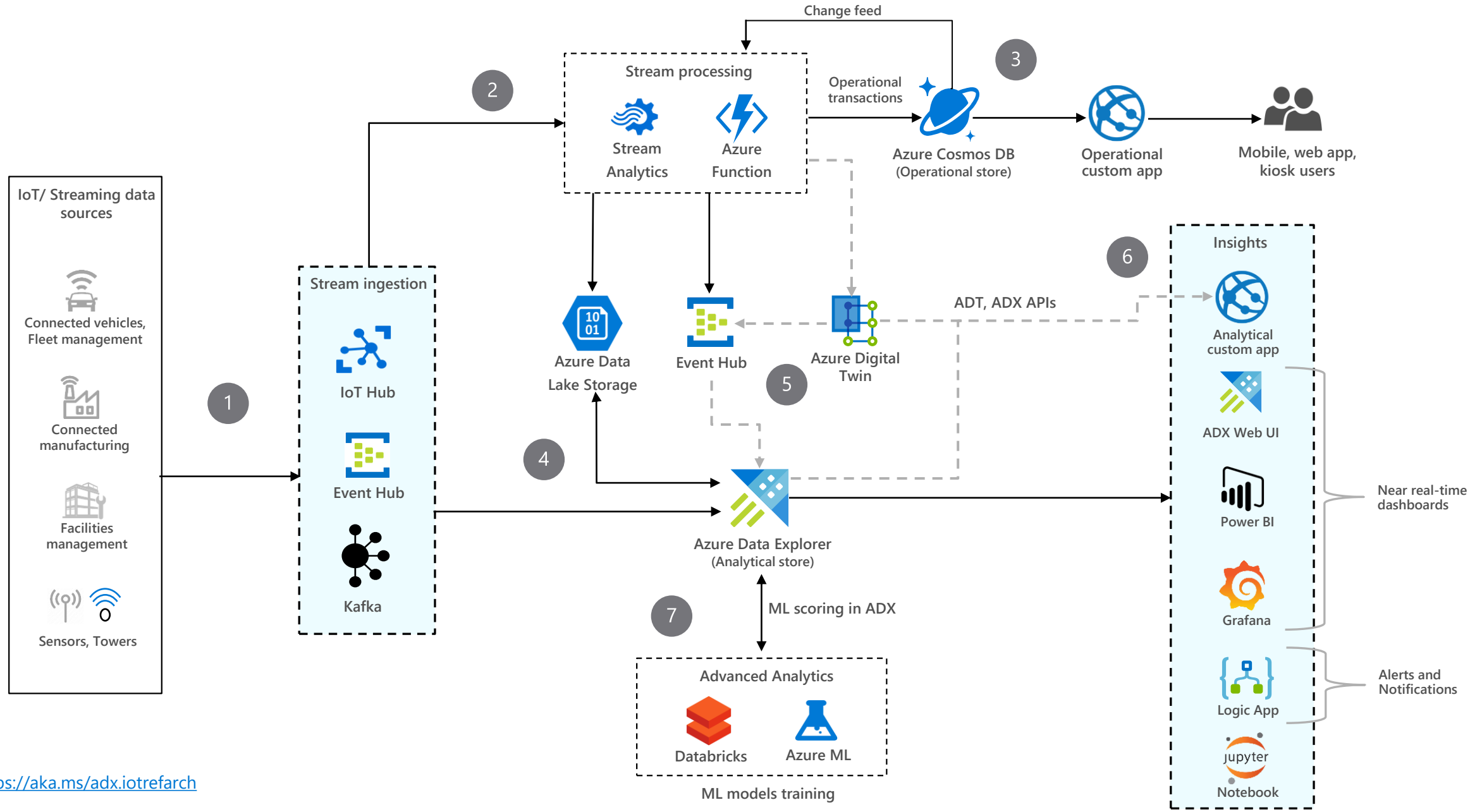




Your options for IoT analytics solutions on Azure



IoT Analytics reference architecture with Azure Data Explorer



https://aka.ms/adx.iotrefarch

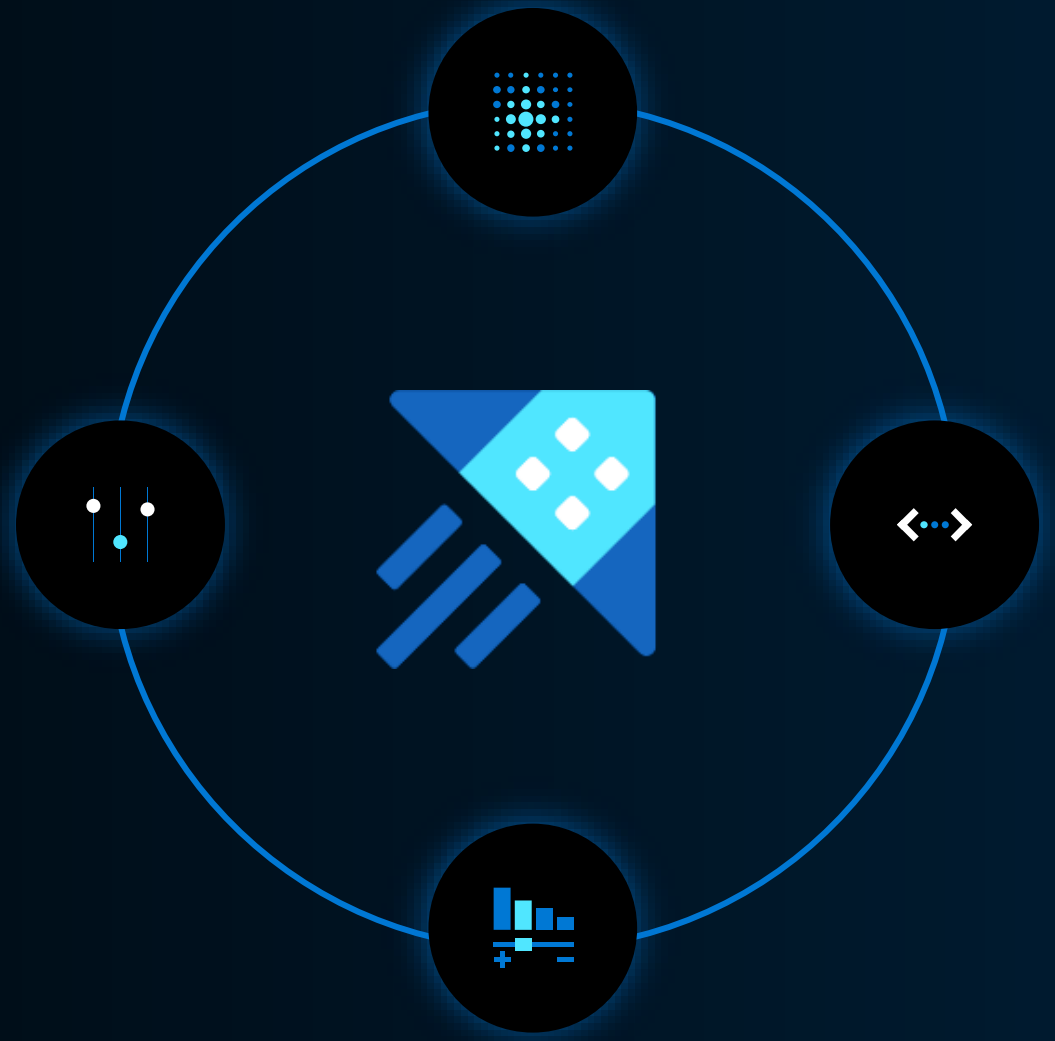
Demo*

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

* Estimate 20 minutes

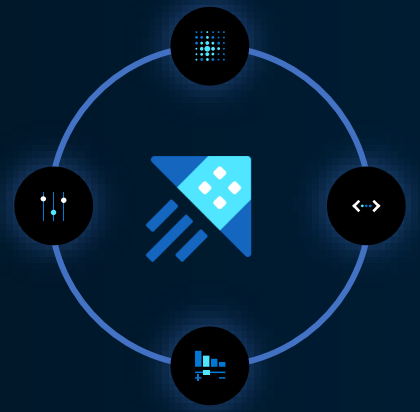
Hands-on Lab

Module 3: aka.ms/adx.iot



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

StageIoTRawData
Thermostats

Thermostat_January2022

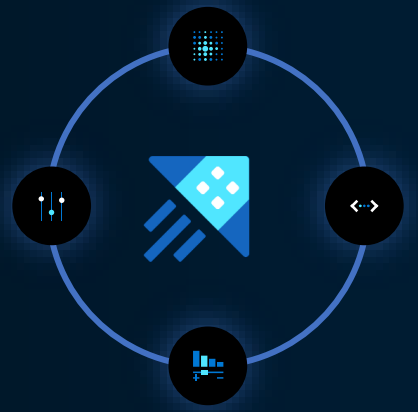
adxscript

Digital Twins

Module 4

Objectives

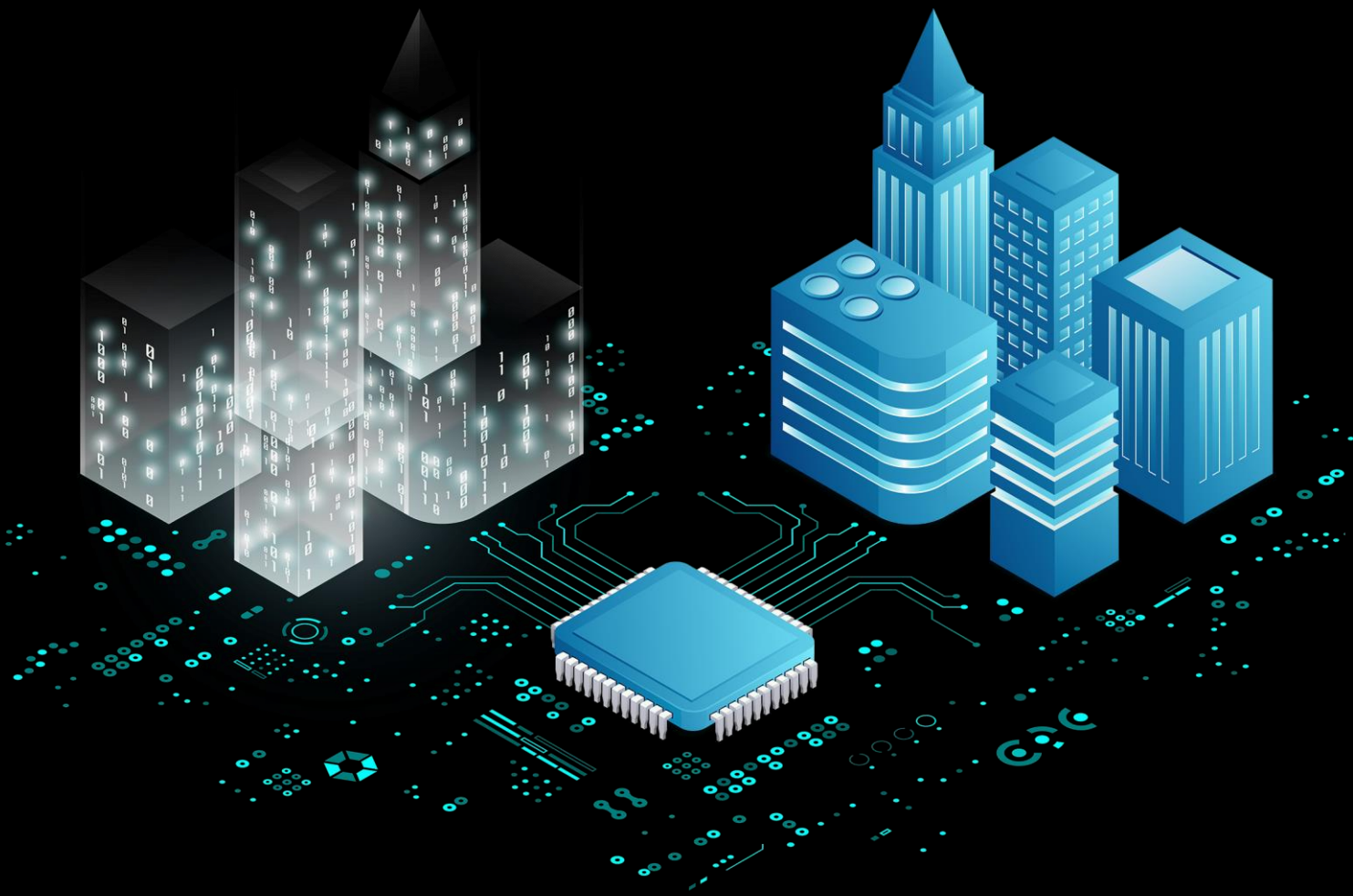
After completing this Learning, you will be able to understand:



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).

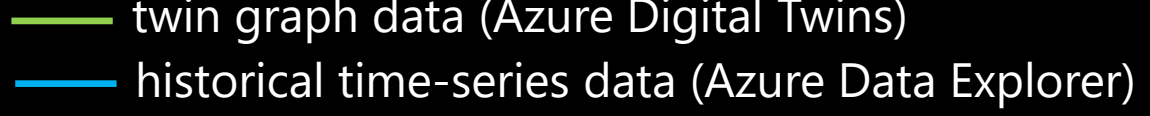


Azure Digital Twins and time-series data

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?



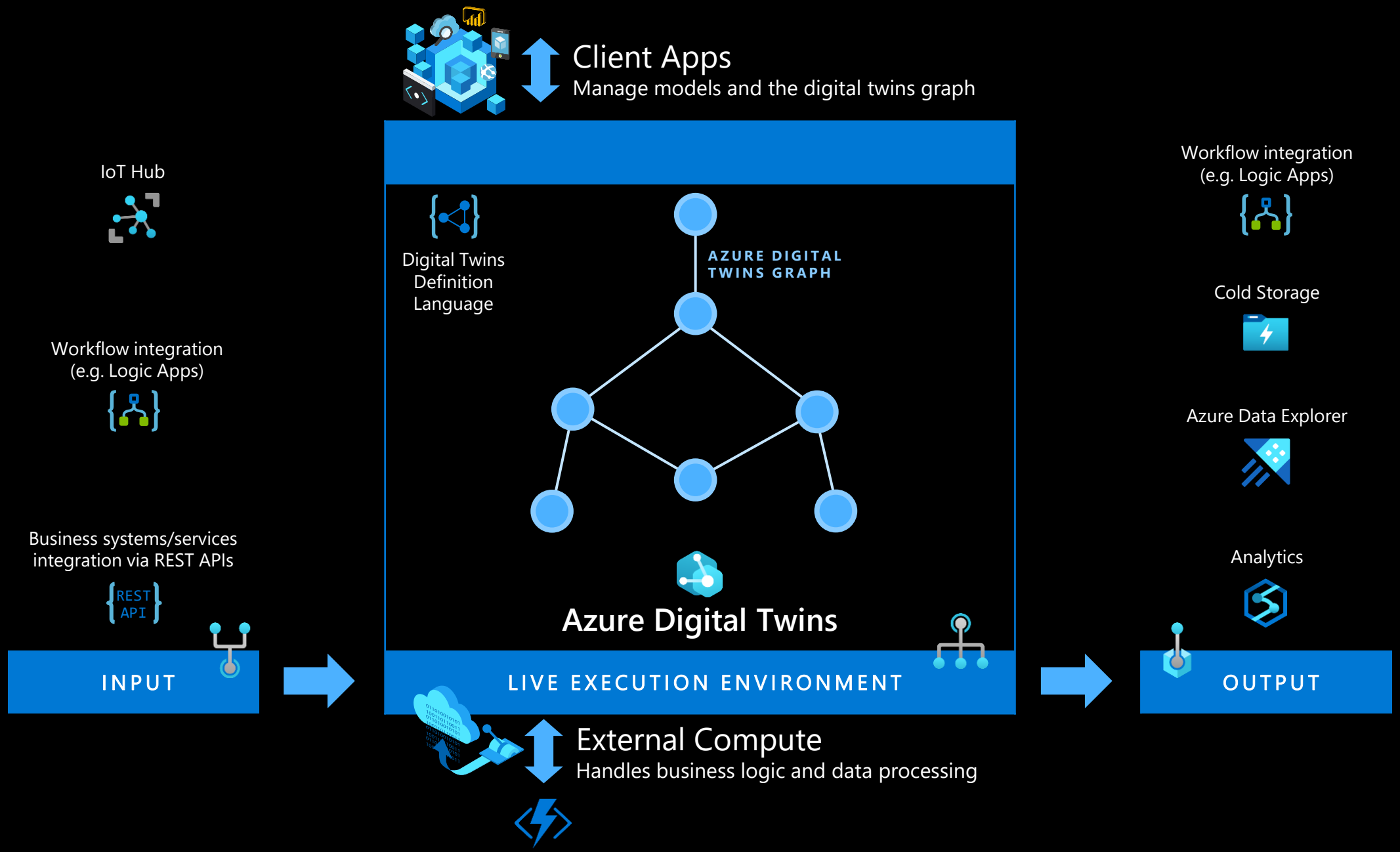
Use the twin graph to *contextualize* time series data

Adding historical analytics to Azure Digital Twins

Service	Data Type	State
Azure Digital Twins	Twin Graph	Last Known Values
Azure Data Explorer	Time Series Data	Historical Values

Best of both worlds: Azure Digital Twins + Azure Data Explorer

Create next generation IoT solutions that model the real world




BOSCH
The Power of Digital Twins and Semantic Data Explorations for Building Operations
Project physical structures & systems to semantical models powered process executions to gain analytical insights for highly efficient and transparent operations

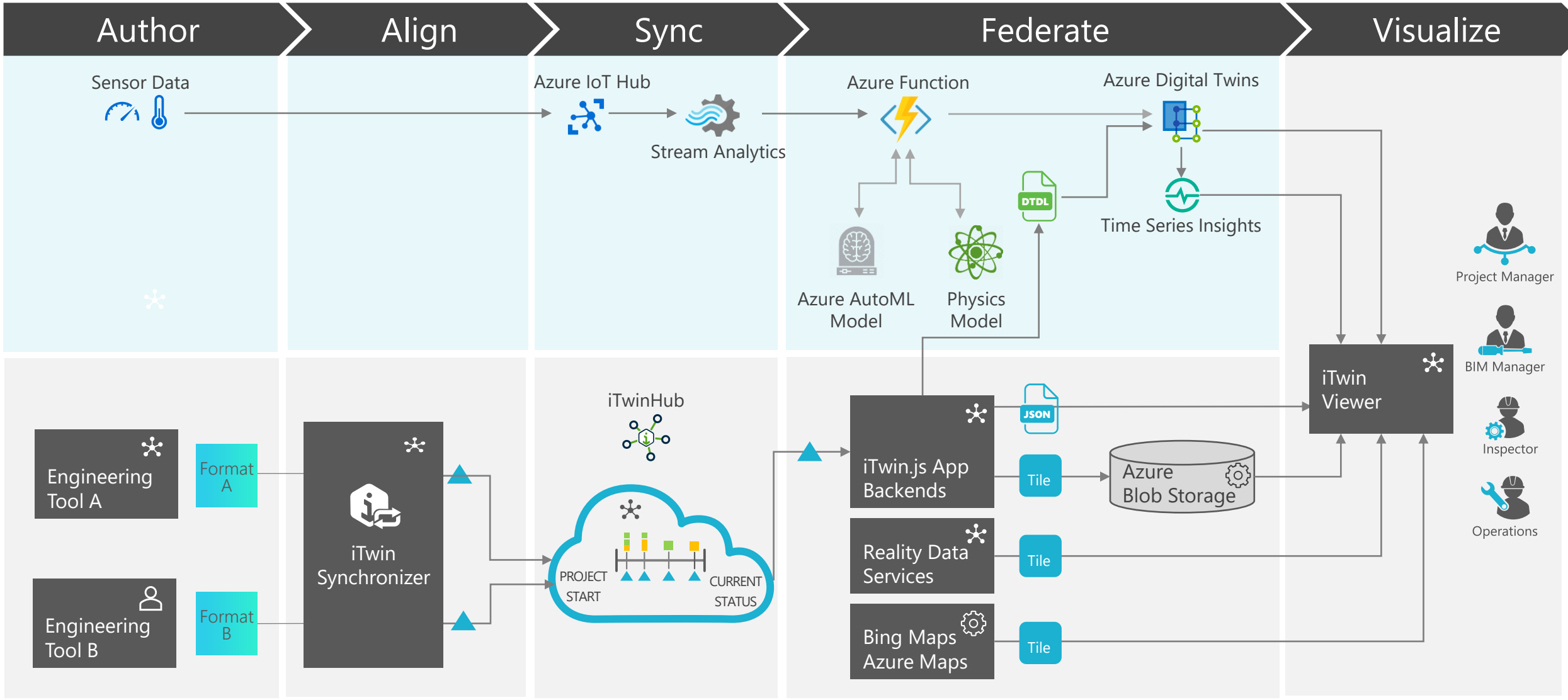
Bentley®
Advancing Infrastructure

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

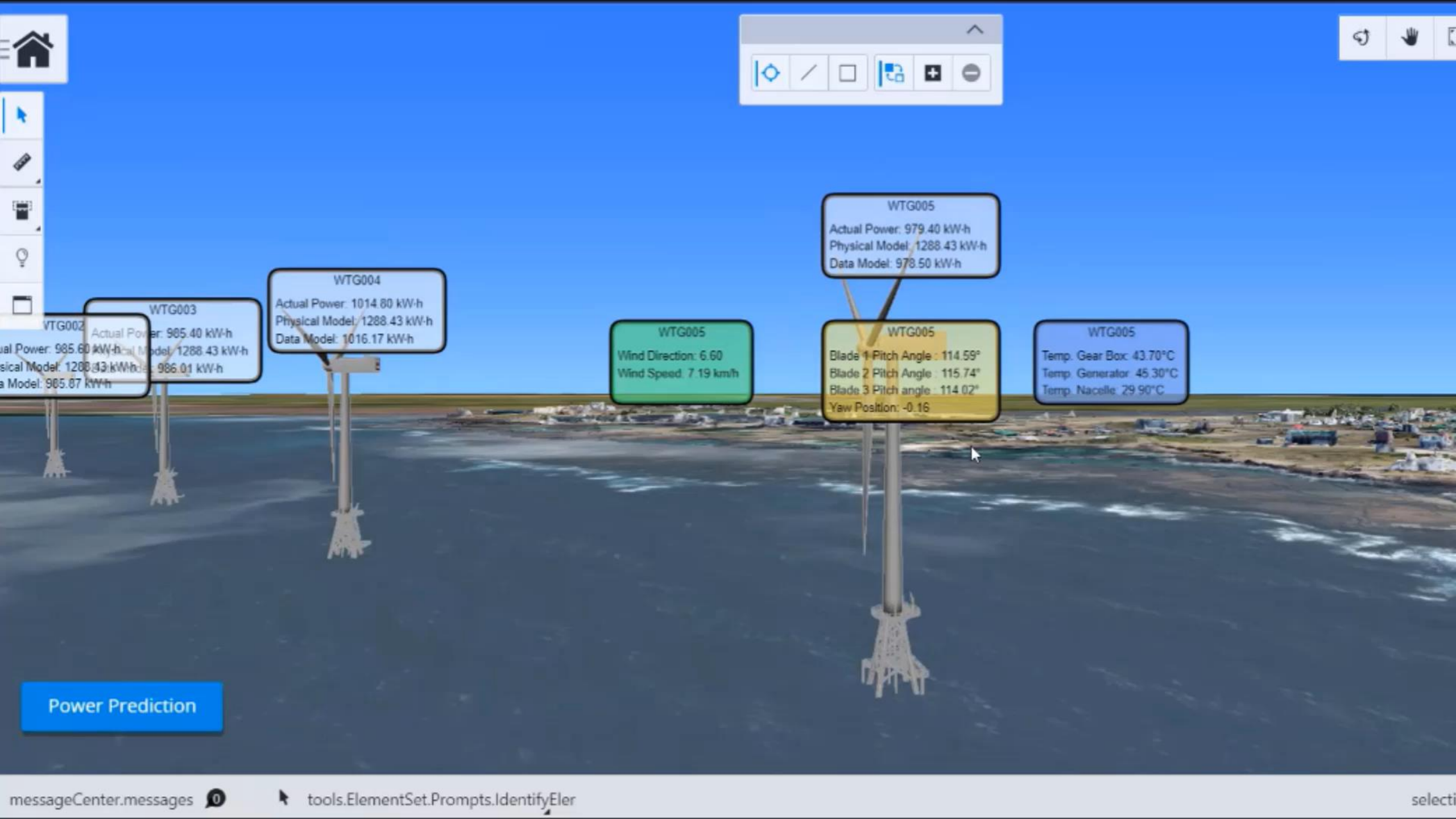
Benefiting from Microsoft Azure Digital Twins and Azure IoT Hub. Bentley's users can rapidly process and make sense of huge amounts of sensor data, produce critical insights, and gain quick decision-making capabilities.

Digital Twins - Viewer



Bentley


localhost:3000
Sign In
Sign Out
WTG008
Actual Power: 951.40 kW·h
Physical Model:
Data Model: 947
WTG009
Actual Power: 929.20 kW·h
Physical Model: 1288.43 kW·h
Data Model: 925.70 kW·h
WTG010
Actual Power: 1049.70 kW·h
Physical Model:
Data Model: 1053.08 kW·h
messageCenter.messages
0
tools.ElementSet.Prompts.IdentifyEler
selectionScopeField.label:
Top Assembly


WTG005
Actual Power: 979.40 kW·h
Physical Model: 1288.43 kW·h
Data Model: 978.50 kW·h
WTG004
Actual Power: 1014.80 kW·h
Physical Model: 1288.43 kW·h
Data Model: 1016.17 kW·h
WTG003
Actual Power: 985.40 kW·h
WTG005
Wind Direction: 6.60
Wind Speed: 7.19 km/h
WTG005
Blade 1 Pitch Angle : 114.59°
Blade 2 Pitch Angle : 115.74°
Blade 3 Pitch angle : 114.02°
Yaw Position: -0.16
WTG005
Temp. Gear Box: 43.70°C
Temp. Generator: 45.30°C
Temp. Nacelle: 29.90°C
Power Prediction
messageCenter.messages
tools.ElementSet.Prompts.IdentifyEler

Product Links

Product

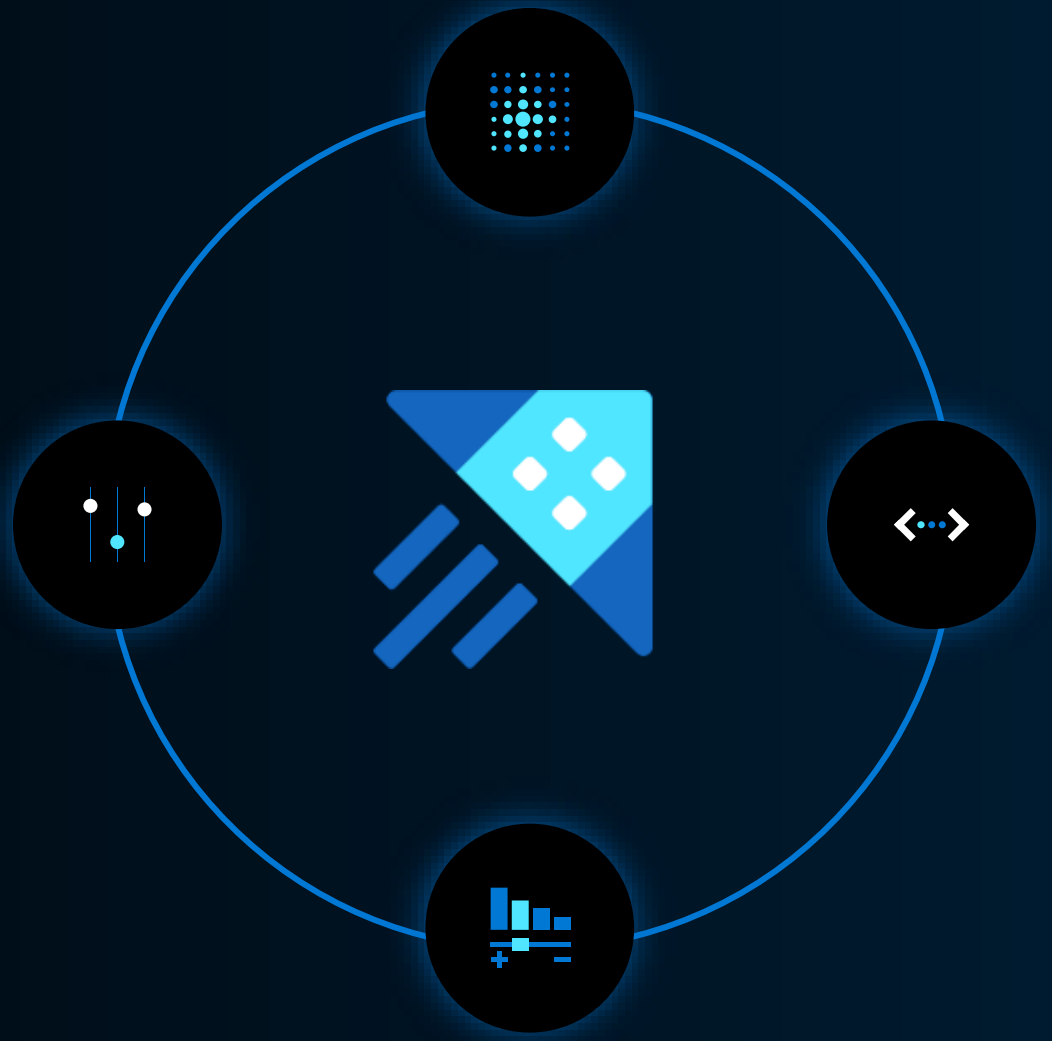
- Product Page: http://aka.ms/DigitalTwins
- Documentation: https://aka.ms/digital-twins
- Content: https://aka.ms/ADTConent
- IoT Show: **Bring your Digital Twins solutions to production with Azure**
- Quickstart: Find available spaces with good air quality
- IoT Learn: MS Learn Azure IoT
- Certification: Azure IoT Developer Specialty
- POSTMAN for Management API
- Customer stories

Social and Community

- Tech Community Blog: https://aka.ms/adx.blog
- Tech Community Forum: https://techcommunity.microsoft.com/t5/Azure-IoT/bd-p/Azure-IoT
- YouTube Channel: https://aka.ms/iotshow

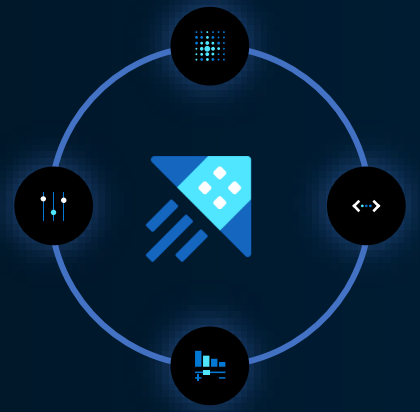
Kusto Query

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.

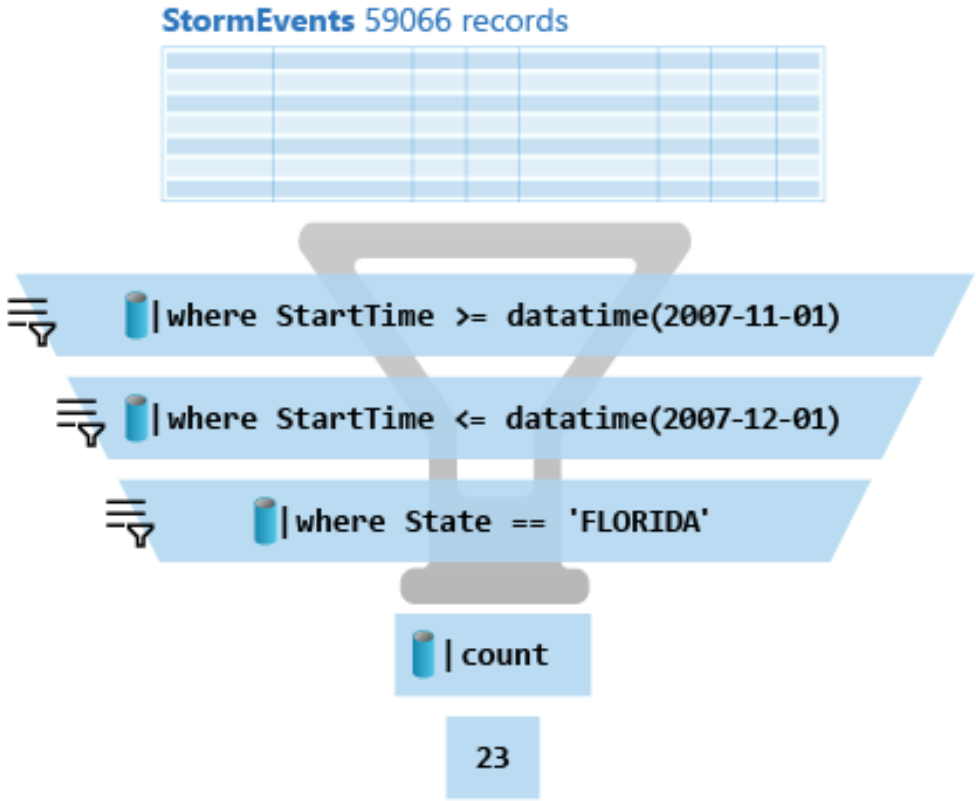
What is a tabular statement?

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
```

PDF: https://techcommunity.microsoft.com/t5/azure-data-explorer-blog/azure-data-explorer-kql-cheat-sheets/ba-p/1057404

Common KQL for IoT Analytics

- Function
- JSON
- Summarize
- Make-series
- Render
- Series_fill_linear
- Series_decompose_forecast
- Series_decomponse_anomalies
- Materialized-view
- Arg-max
- External-table
- Invoke
- Project-away
- Evaluate
- Join

Kusto (sample & function)

```
// cluster('igniteadxsource.eastus2').database('Occupancy')

StageIoTRawData
| sample 10

.create-or-alter function ExtractThermostatData {
    StageIoTRawData
    | where EventType has 'Thermostat' and Body has 'temp'
    | project
    EnqueuedTimeUTC,
    SubEventType,
    DeviceID,
    BatteryLevel = tolong(Body.['BatteryLevel']),
    Temp = toreal(Body.['temp']),
    Humidity = toreal(Body.['humidity'])
}

Thermostats
| sample 10
```

Kusto (sum, series & render)

```
Thermostats
| count

//What is the average temp every 1 min
Thermostats
| where EnqueuedTimeUTC > ago(7d)
| where DeviceId =='637086755190714287'
| summarize avg(Temp) by bin(EnqueuedTimeUTC,1m)
| render timechart


//Is there any missing data?
//make-series
//Create series of specified aggregated values along specified axis.
Thermostats
| where EnqueuedTimeUTC > ago(6h)
| where DeviceId =='637086755190714287'
| make-series AvgTemp=avg(Temp) on EnqueuedTimeUTC from ago(6h) to now() step 1m
| render timechart
```

Kusto (series fill)

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

Kusto (forecast)

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

Kusto (anomalies)

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

Kusto (expand)

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

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

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

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

Ref: 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

```
.show function GetDevicesbyStore

.create-or-alter function with (folder = "Analytics/IoT", skipvalidation = "true")
GetDevicesbyStore(Office:string) {
let ADTendpoint = "https://digitaltwinpm10774.api.eus.digitaltwins.azure.net";
let ADTquery = strcat("SELECT T.$dtId as Office, F.$dtId as Floor, D.$dtId as
DeviceId FROM DIGITALTWINS T JOIN F RELATED T.officecontainsfloors JOIN D
RELATED F.floorcontainsdevices where T.$dtId='", Office, "'");
evaluate azure_digital_twins_query_request(ADTendpoint, ADTquery)
| project Office=tostring(Office), Floor=tostring(Floor),
DeviceId=tostring(DeviceId)
}
```

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

```
customer
| join hint.strategy = shuffle
orders
on $left.c_custkey == $right.o_custkey
| 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 `has` operator	Don't use `contains`	When looking for full tokens, `has` works better, since it doesn't look for substrings
Case-sensitive	Use `==`, `in`, `contains_cs`	Don't use `=~`, `in~`	Use case-sensitive operators when possible
Searching text	Look in a specific column	Don't use `*`	Does a full text search across all columns
New queries	Use `limit [small number]` or `count` 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	`hint.strategy=shuffle` 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 `hint.strategy=broadcast`		Small means <= 100,000 records

See more: 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

KQL Resources

Learn more

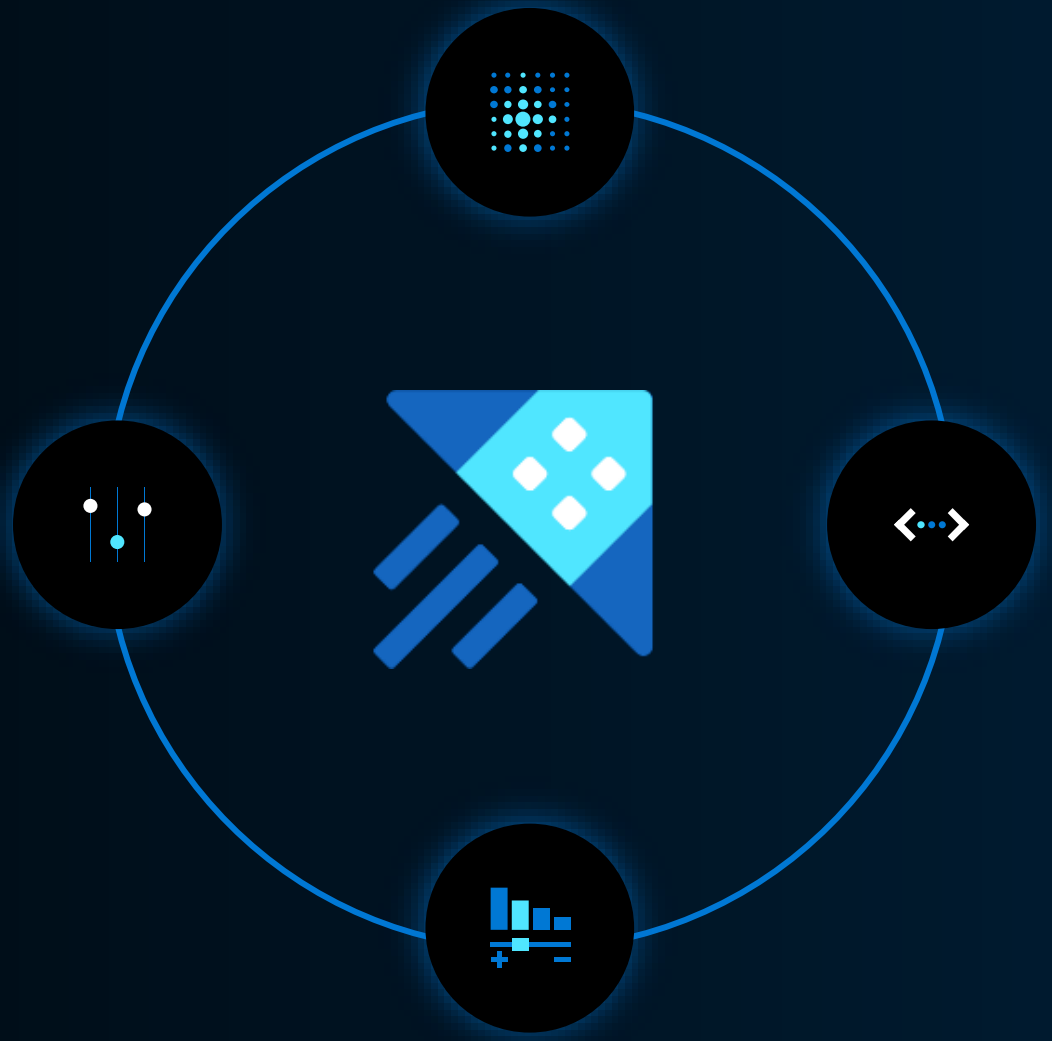
- Docs: https://aka.ms/adx.docs
- Learn: Write your first query with Kusto Query Language
- Free online Courses: KQL from Scratch, Advanced KQL
- Lab: https://aka.ms/adx.lab
- Trial Cluster: http://aka.ms/adx.try
- ADX Video: KQL for beginners
- KQL Samples: Kusto queries
- Tutorial: Use Kusto queries
- ADX KQL Playlist: YouTube Channel

Social and Community

- Twitter: @AzDataExplorer
- Tech Community Blog: https://aka.ms/adx.blog
- Tech Community Forum: http://aka.ms/adx.techcommunity
- Stack overflow: https://aka.ms/adx.sof
- YouTube Channel: https://aka.ms/adx.youtube

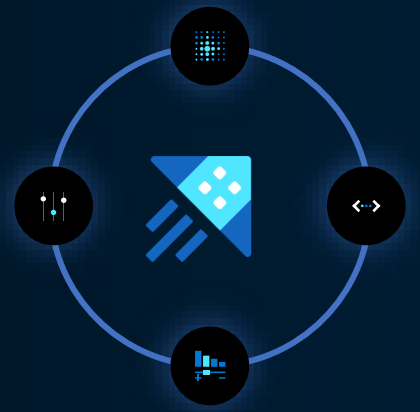
ML & Time Series

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

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

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

```
demo_series2
| extend series_fit_2lines(y), series_fit_line(y)
| 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

```
demo_series3
| render timechart;

demo_series3
| project (periods, scores) = series_periods_detect(num, 0., 14d/2h, 2)
| mv-expand periods, scores
| 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

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

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

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

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

```
let min_t = datetime(2017-01-05);
let max_t = datetime(2017-02-03 22:00);
let dt = 2h;
let horizon = 7d;
demo_make_series2
| make-series num=avg(num) on TimeStamp from min_t to max_t+horizon step dt
by sid
| where sid == 'TS1'
| extend forecast = series_decompose_forecast(num, toint(horizon/dt))
| 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

```
let min_t = datetime(2017-01-05);
let max_t = datetime(2017-02-03 22:00);
let dt = 2h;
let horizon = 7d;
demo_make_series2
| make-series num=avg(num) on TimeStamp from min_t to max_t+horizon step dt
by sid
| extend offset=case(sid=='TS3', 4000000, sid=='TS2', 2000000, 0)
| extend num=series_add(num, offset)
| extend forecast = series_decompose_forecast(num, toint(horizon/dt))
| 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

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

```
let min_t=datetime(2016-08-23 11:00);
demo_clustering1
| make-series num=count() on PreciseTimeStamp from min_t to min_t+8h step 1m
| 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

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
demo_clustering1 | where PreciseTimeStamp between(min_peak_t..max_peak_t)
| count

//take a sample 20 of 972:
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
demo_clustering1 | where PreciseTimeStamp between(min_peak_t..max_peak_t)
| take 20
```

Use autocluster() for single record set clustering

- Instantly extract a short list of common segments.

[Click to run query]

Kusto

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
demo_clustering1 | where PreciseTimeStamp between(min_peak_t..max_peak_t)
| 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

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
demo_clustering1 | where PreciseTimeStamp between(min_peak_t..max_peak_t)
| 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

```
let min_peak_t=datetime(2016-08-23 15:00);
let max_peak_t=datetime(2016-08-23 15:02);
let min_baseline_t=datetime(2016-08-23 14:50);
let max_baseline_t=datetime(2016-08-23 14:58);
let splitime=(max_baseline_t+min_peak_t)/2.0;
demo_clustering1
| where (PreciseTimeStamp between(min_baseline_t..max_baseline_t)) or
        (PreciseTimeStamp between(min_peak_t..max_peak_t))
| extend AB=iff(PreciseTimeStamp > splitime, 'Anomaly', 'Baseline')
| 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

```
let min_t = toscalar(demo_clustering1 | summarize min(PreciseTimeStamp));
let max_t = toscalar(demo_clustering1 | summarize max(PreciseTimeStamp));
demo_clustering1
| extend seg = iff(Region == "eau" and ScaleUnit == "su7" and DeploymentId ==
"b5d1d4df547d4a04ac15885617edba57"
and ServiceHost == "e7f60c5d-4944-42b3-922a-92e98a8e7dec", "Problem", "Normal")
| make-series num=count() on PreciseTimeStamp from min_t to max_t step 10m by seg
| 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 3.6* 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.

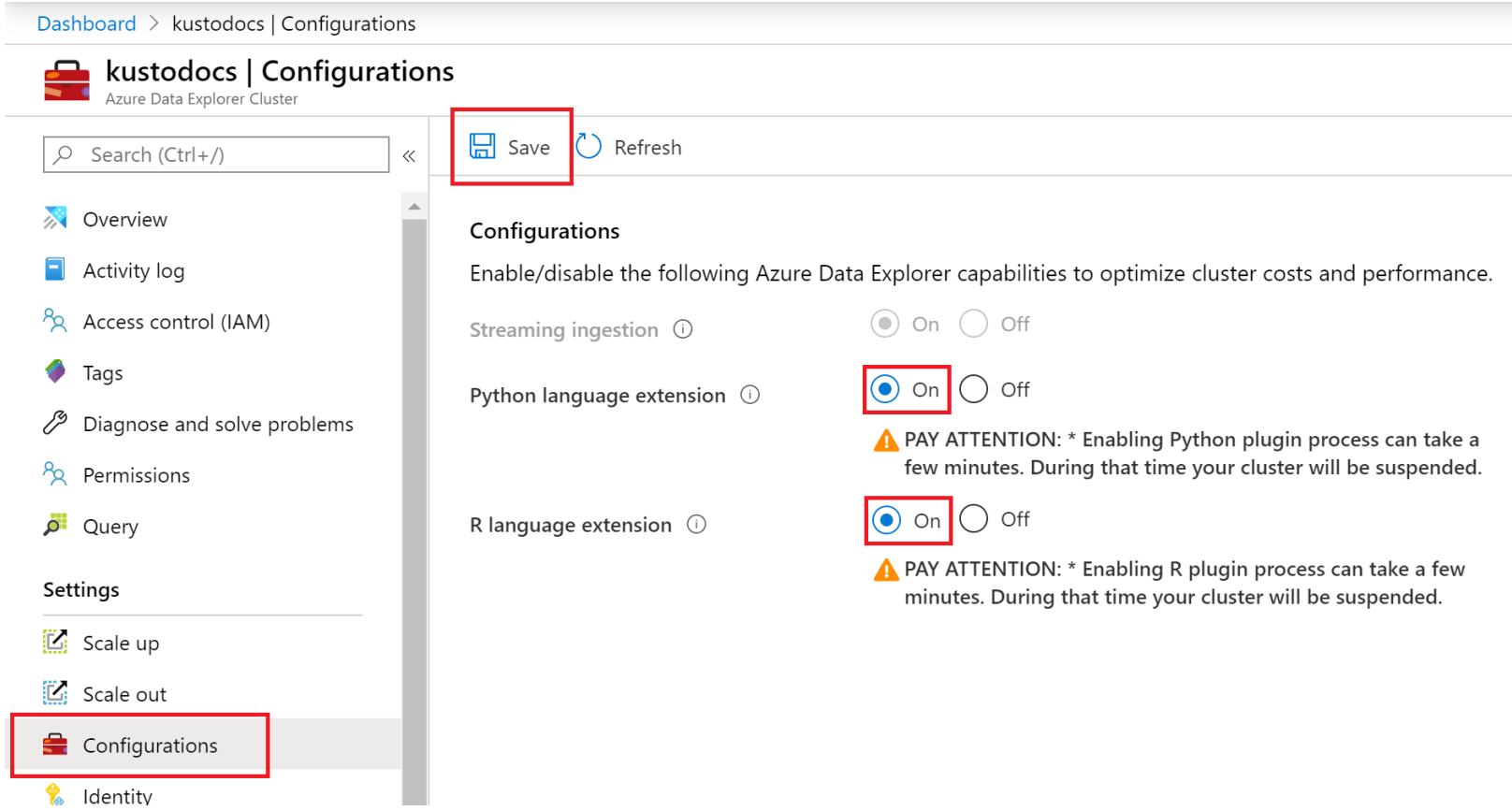
Python Examples

Kusto

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



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_\n';
16     tbl
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

```
// Custom UDF to score based on pre-trained model
.create-or-alter function with (folder = "Python", skipvalidation = "true")
classify_sf_OccupDetc(
    samples:(*),
    models_tbl:(name:string,timestamp:datetime,model:string),
    model_name:string,
    features_cols:dynamic,
    pred_col:string)
{
    let model_str = toscalar(
    ML_Models
    | where name == model_name
    | top 1 by timestamp desc
    | project model
    );
    let kwargs = pack('smodel', model_str, 'features_cols', features_cols,
'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

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

```
.show cluster policy callout

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

Note: This change requires AllDatabasesAdmin permissions.

Example

Install the Faker package that generates fake data.

Kusto

```
range ID from 1 to 3 step 1
| extend Name=''
| evaluate python(typeof(*), ```if 1:
    from sandbox_utils import Zipackage
    Zipackage.install("Faker.zip")
    from faker import Faker
    fake = Faker()
    result = df
    for i in range(df.shape[0]):
        result.loc[i, "Name"] = fake.name()
    ```,
    external_artifacts=pack('faker.zip',
'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

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

```
.show operations
| where StartedOn > ago(1d)
| project d_seconds = Duration / 1s
| evaluate hint.distribution = per_node r(
    typeof(*, d2:double),
    'result <- df\n'
    'result$d2 <- df$d_seconds\n'
  )
| 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

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

ADX: ML & TS Resources for IoT

Learn more

- Docs: https://aka.ms/adx.docs
- Kusto Analytics Lib: github.com/Azure/azure-kusto-analytics-lib
- Trial Cluster: http://aka.ms/adx.try

Social and Community

- Twitter: @AzDataExplorer
- Tech Community Blog: https://aka.ms/adx.blog
- Tech Community Forum: http://aka.ms/adx.techcommunity
- Stack overflow: https://aka.ms/adx.sof
- YouTube Channel: https://aka.ms/adx.youtube

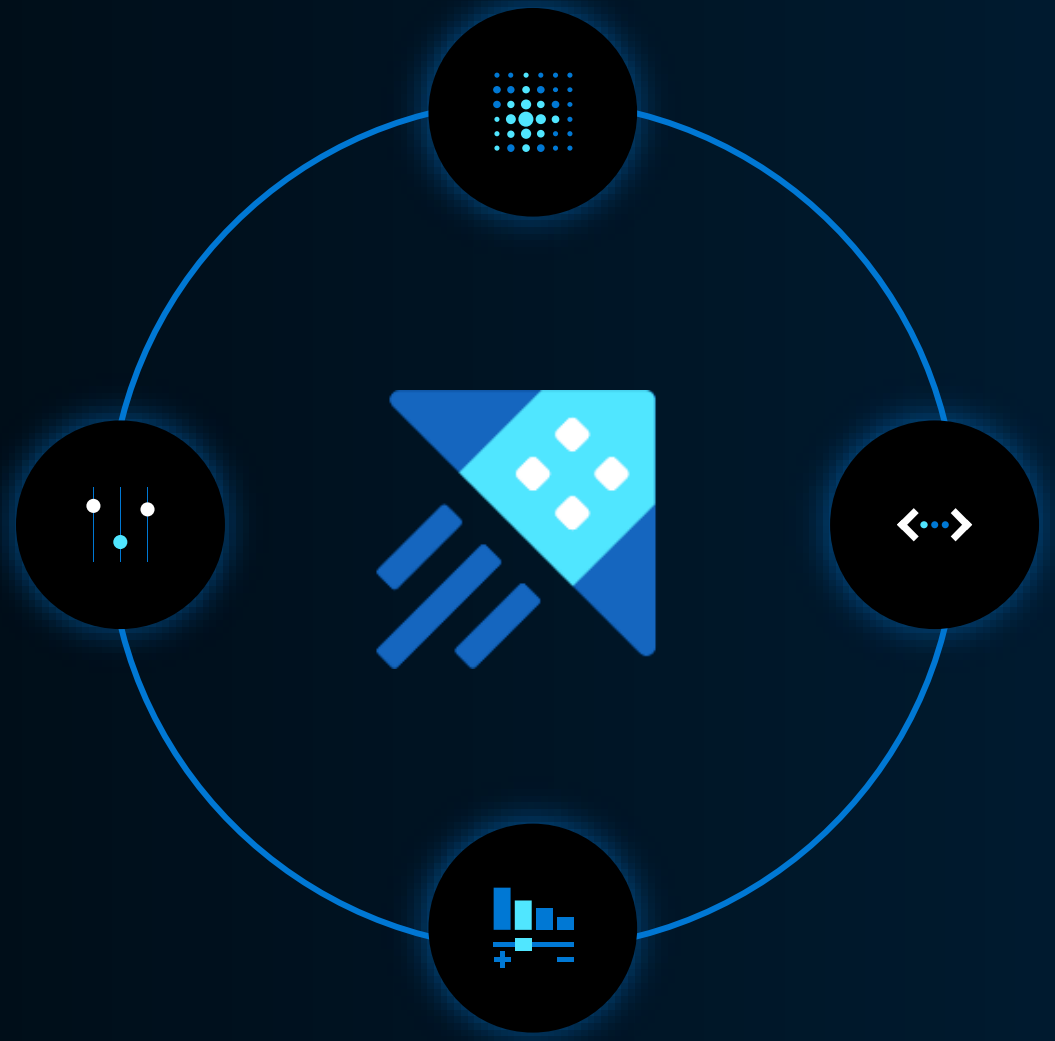
Demo*

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

* Estimate 5 minutes

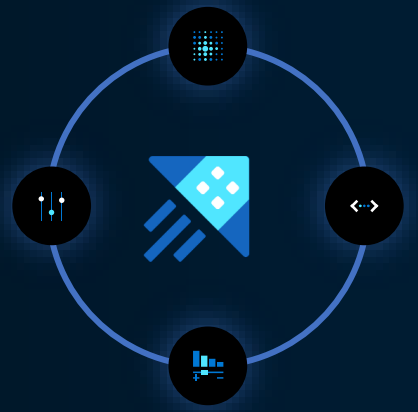
Visualize Data

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. anomalychart
2. areachart
3. barchart
4. card
5. columnchart
6. ladderchart*
7. linechart
8. piechart
9. pivotchart*
10. scatterchart
11. stackedareachart
12. table
13. timechart
14. timepivot*

Sample:

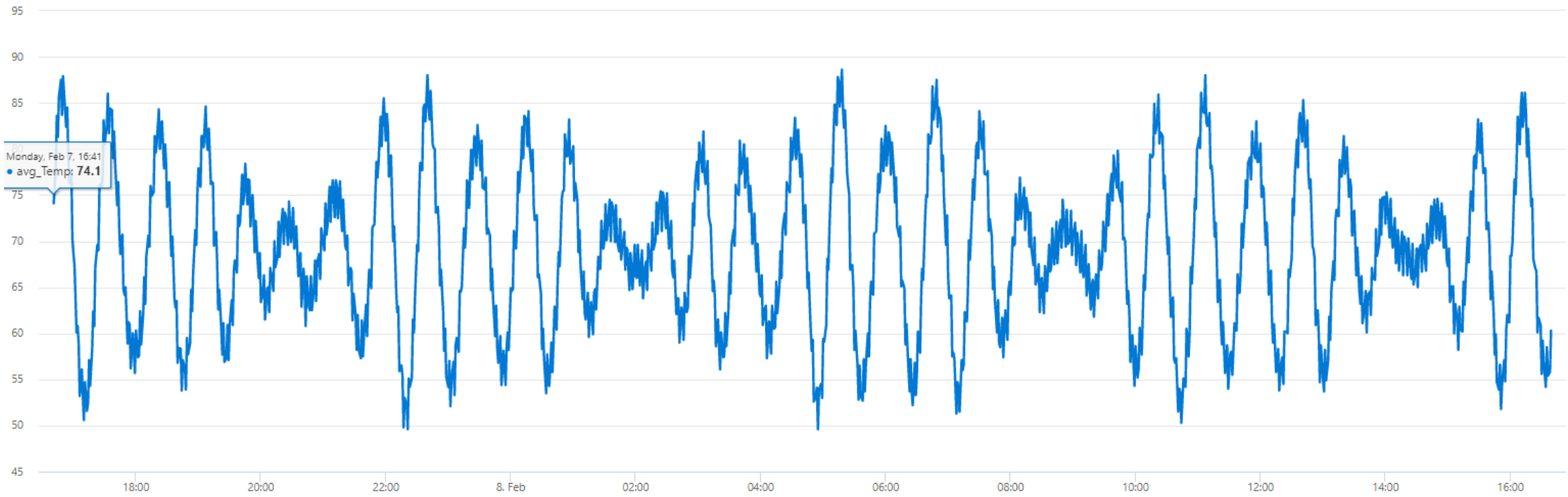
```
range x from 0.0 to 2*pi() step 0.01 | extend y=sin(x) | render linechart
```

* not yet supported

render operator example:

timechart

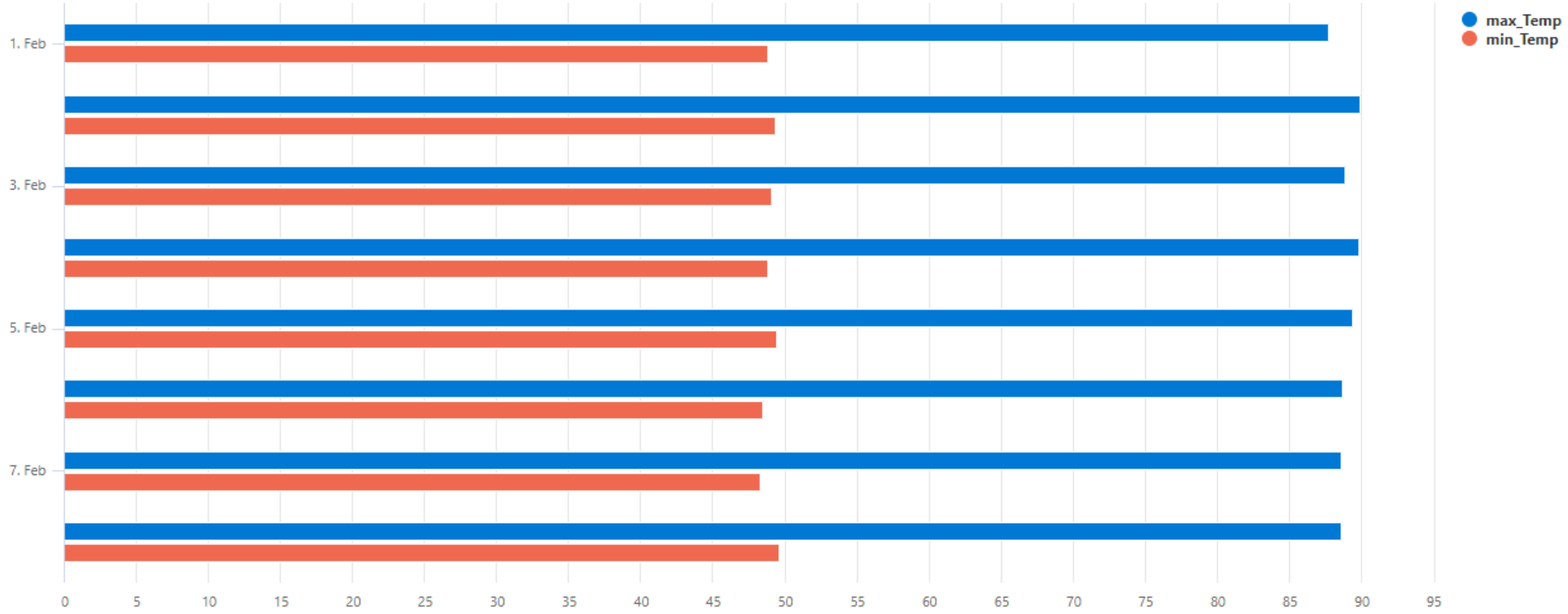
```
109 Thermostats
110 | where EnqueuedTimeUTC >= ago(7d)
111 | summarize avg(Temp) by bin(EnqueuedTimeUTC,1m)
112 | render timechart
```



render operator example:

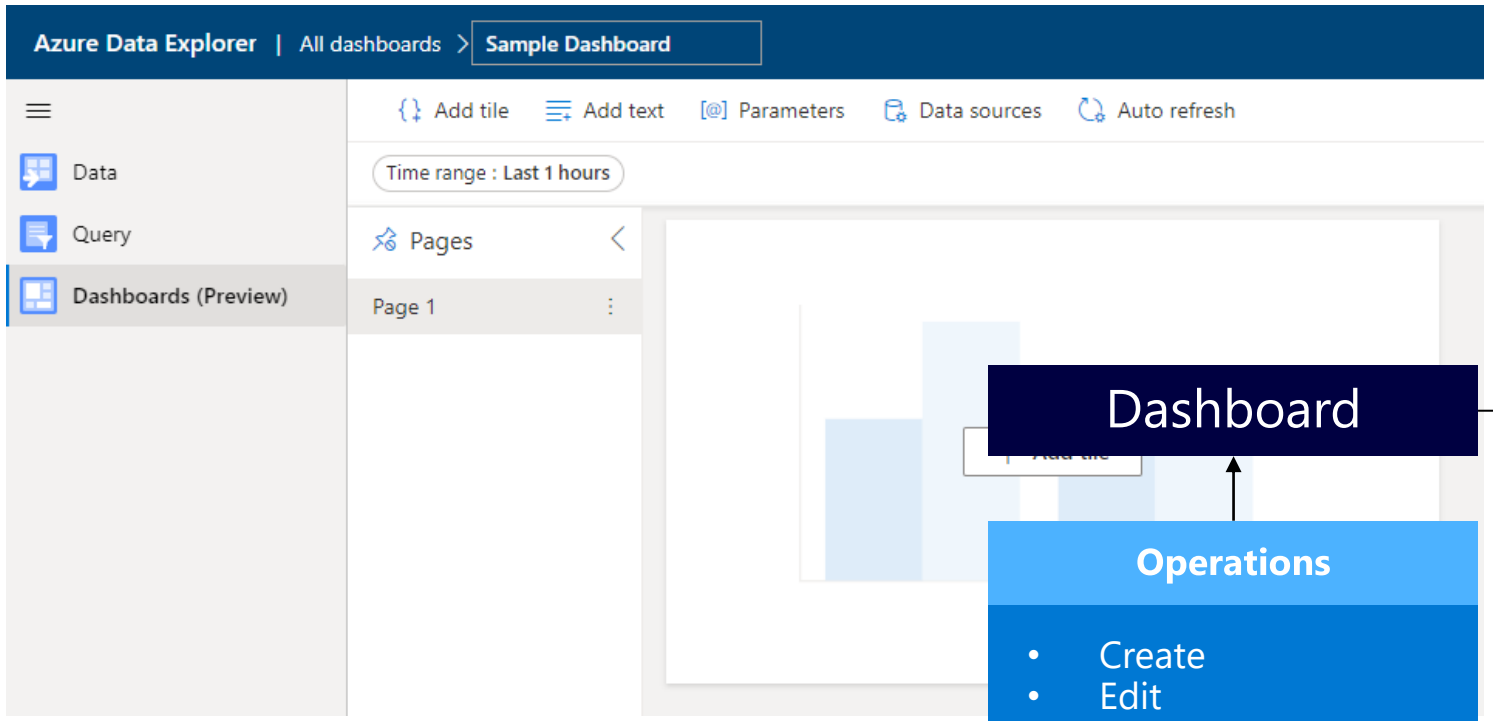
barchart

```
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



Dashboard

1...n

Page

1...n

Tile

Operations

- Create
- Edit
- Copy link
- Manage Permissions
- Duplicate
- Delete
- Forget
- Export to File
- Replace with File
- Share

Operations

- Create
- Edit
- Add tile
- Add text
- Data source(s)
- Auto refresh
- Parameters
- Rename
- Duplicate
- Delete
- Move

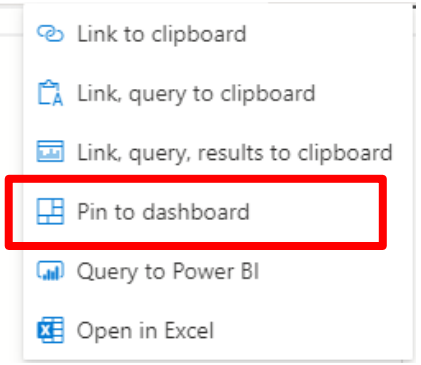
Operations

- Create
- Edit
- Rename
- Duplicate
- Delete
- Move

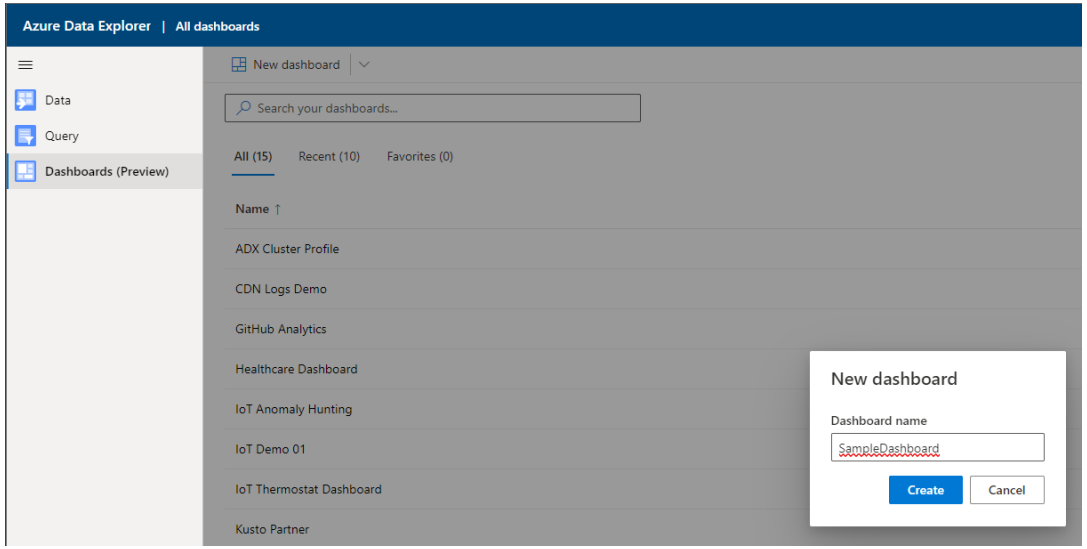
Creating Dashboards

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

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

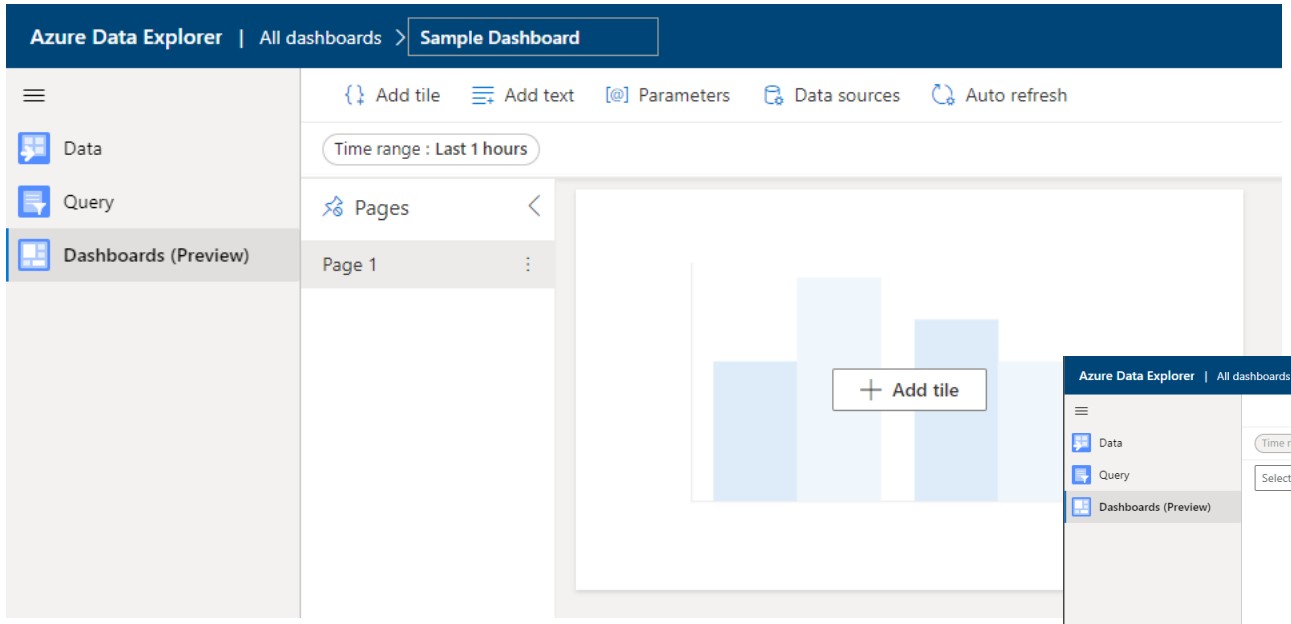


2. Vanilla approach: Create a new Dashboard

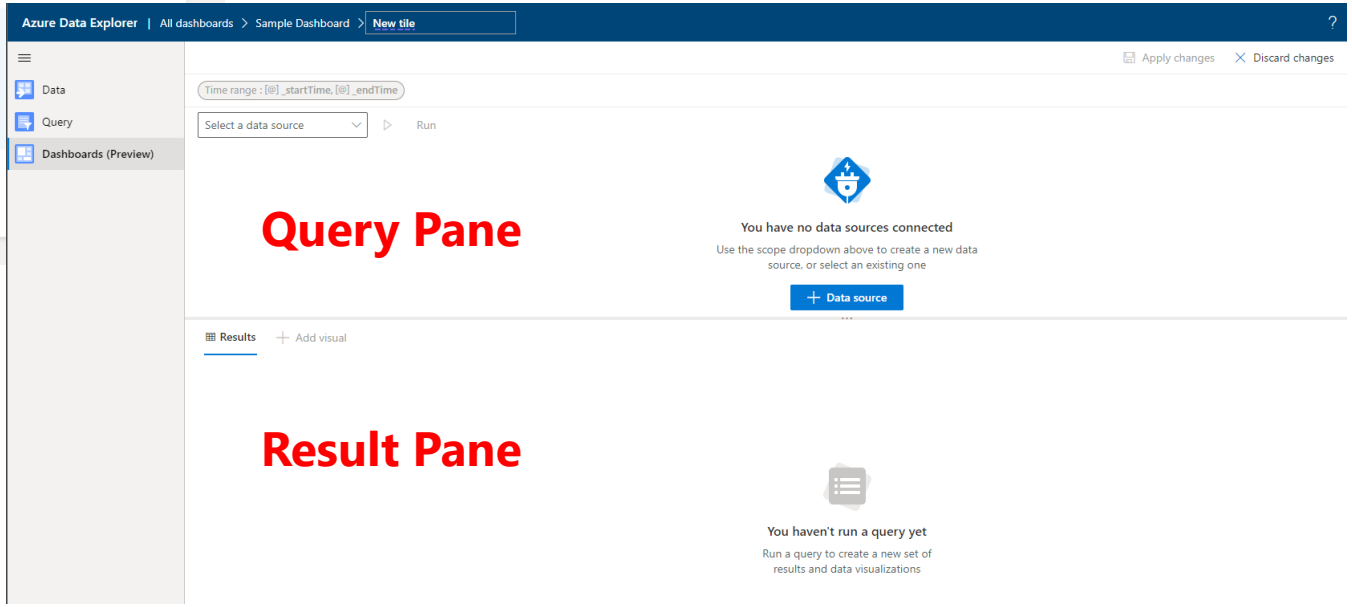


Creating pages and tiles

1. 1st Page is automatically created. Add a new tile.

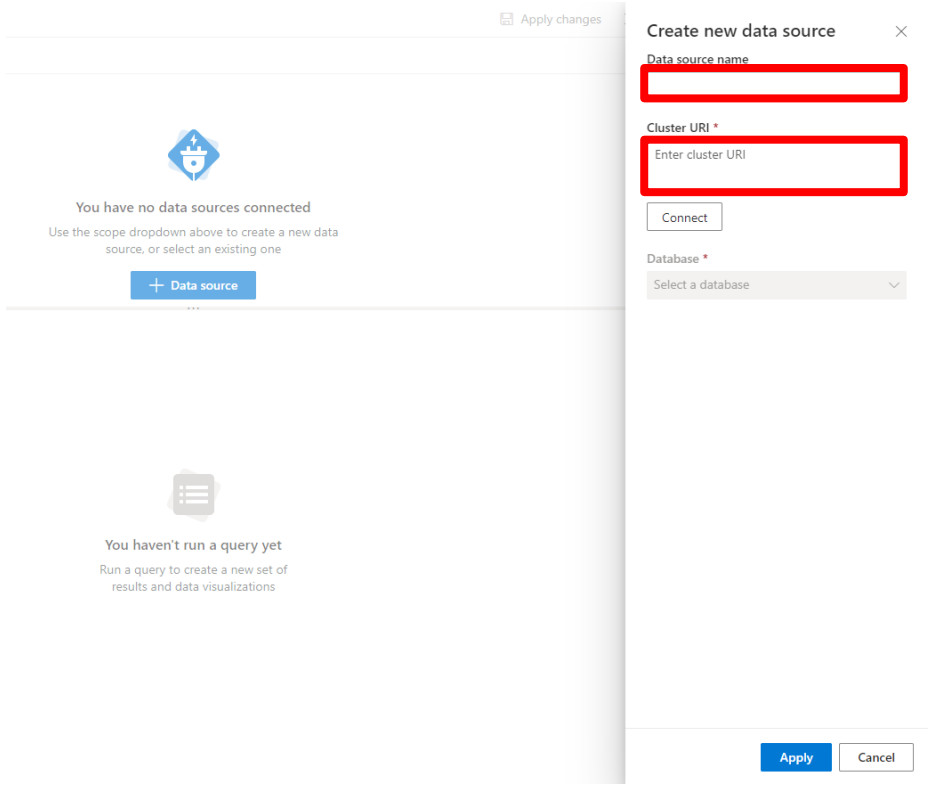


2. On adding a new tile, familiar KQL editor experience is provided

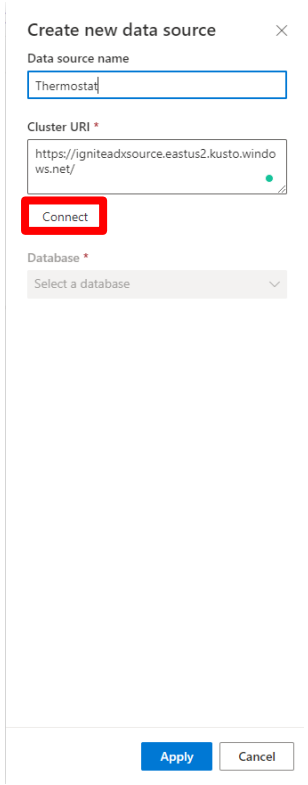


Adding data sources

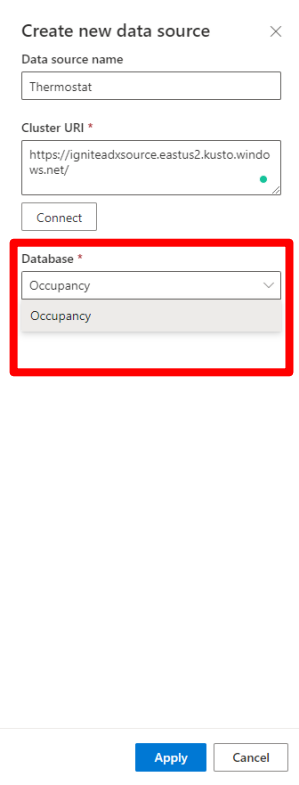
1. Add data source. Add your cluster's URI. You can find it on the Overview page in Azure Portal



2. Connect to your cluster to retrieve the databases

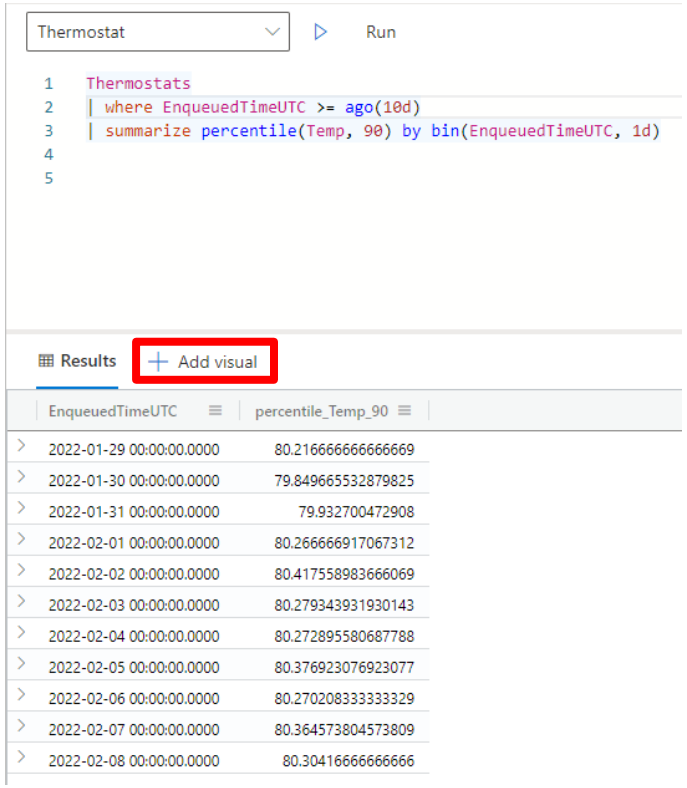


3. Select the database that you want to query

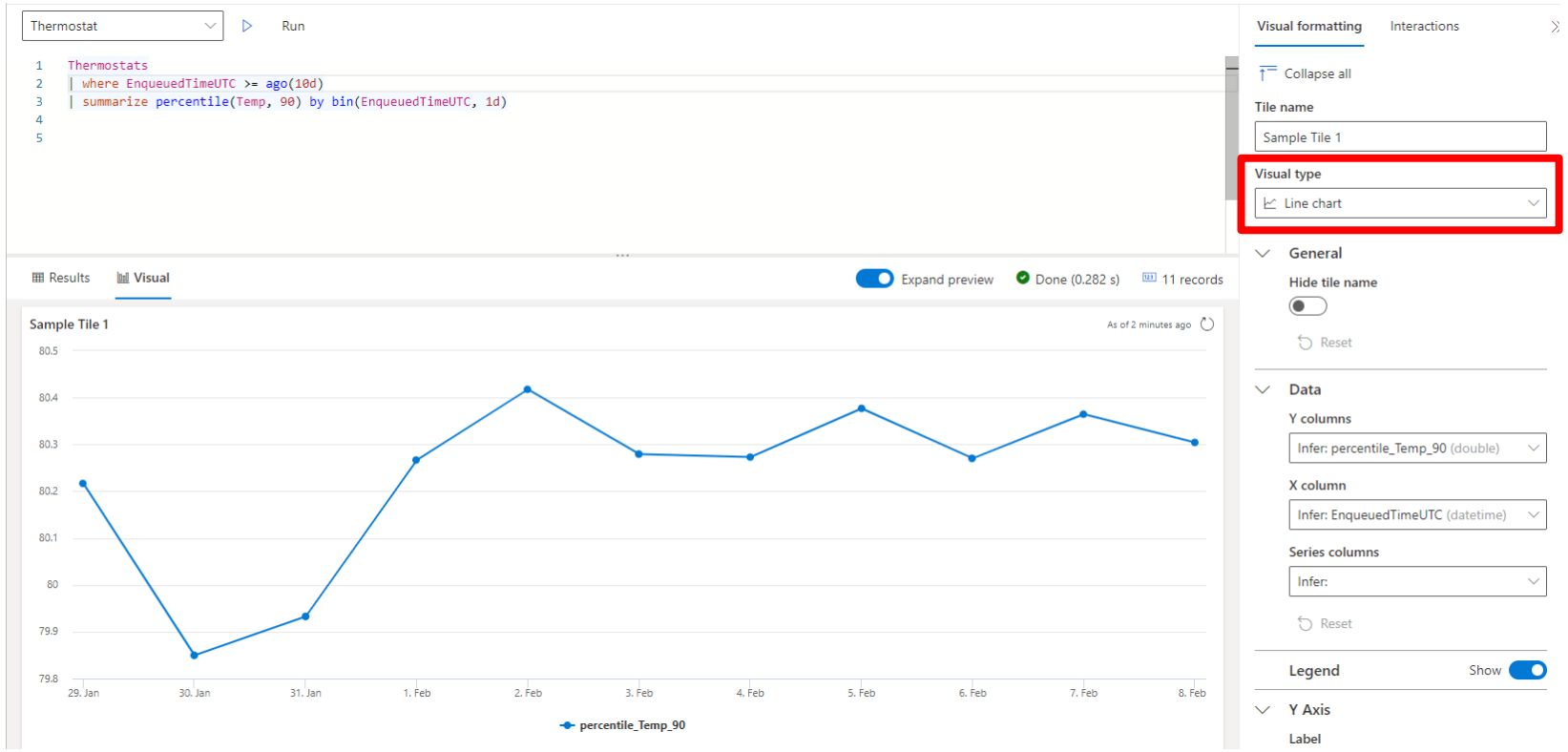


Creating a chart

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

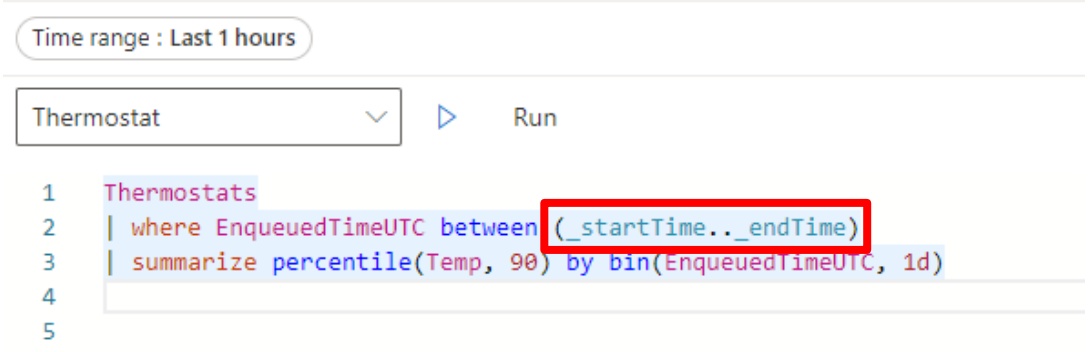


2. Choose your preferred chart type. Generally, ADX will infer x & y axes from the table. Use formatting options (conditional, legends, titles, etc.)

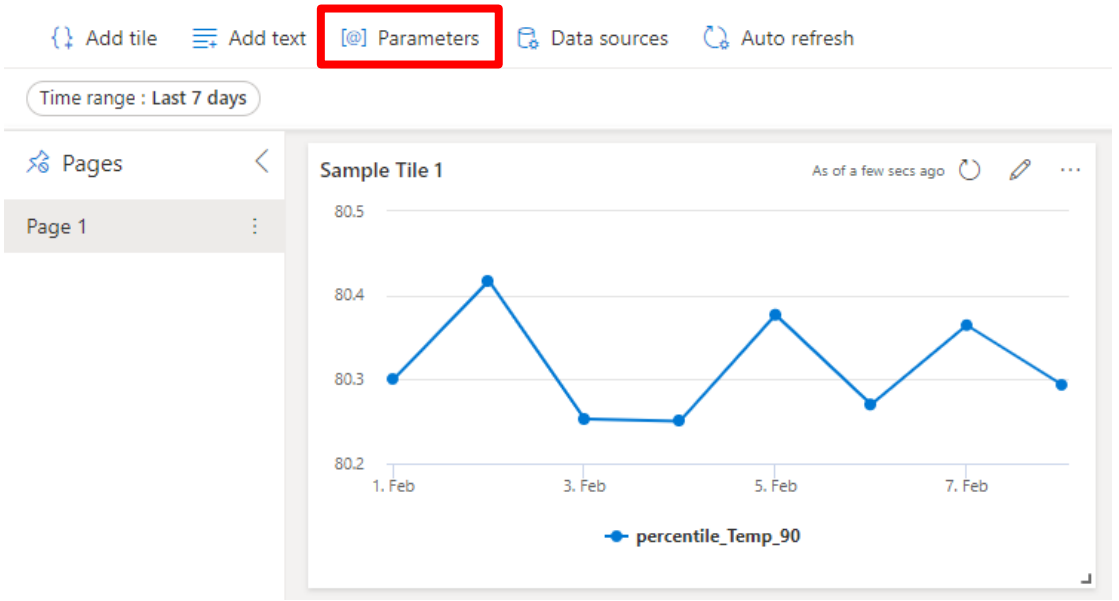


Parameters

1. Add parameters to your query

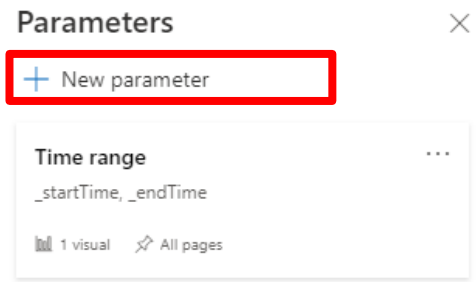


2. Add more parameters for your page (applies to tiles that have appropriately parametrized queries

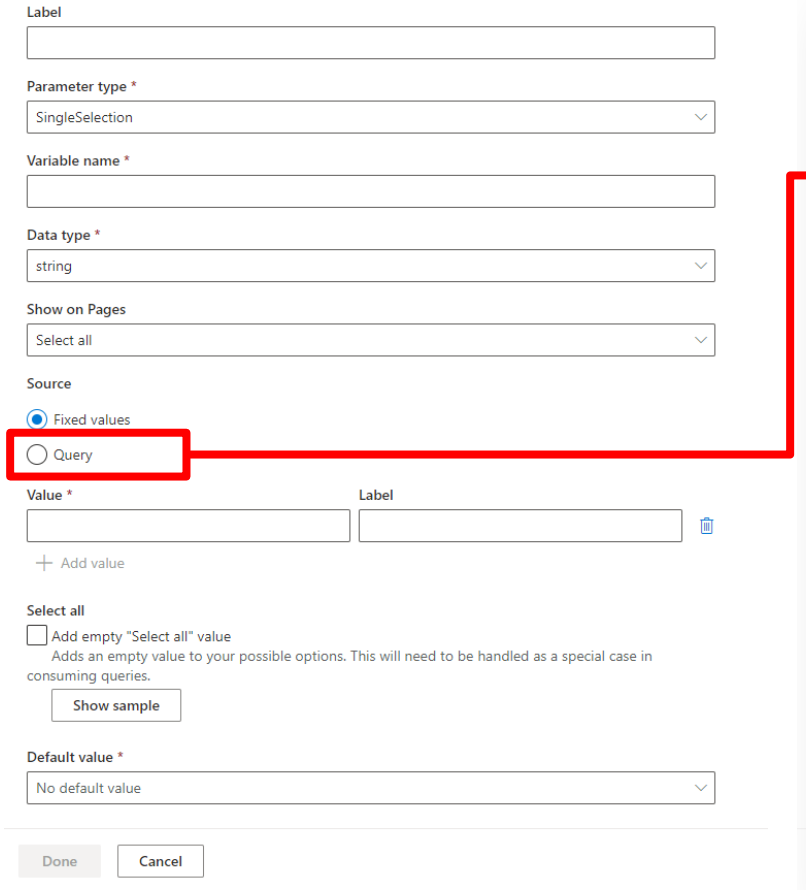


Parameters (cont'd…)

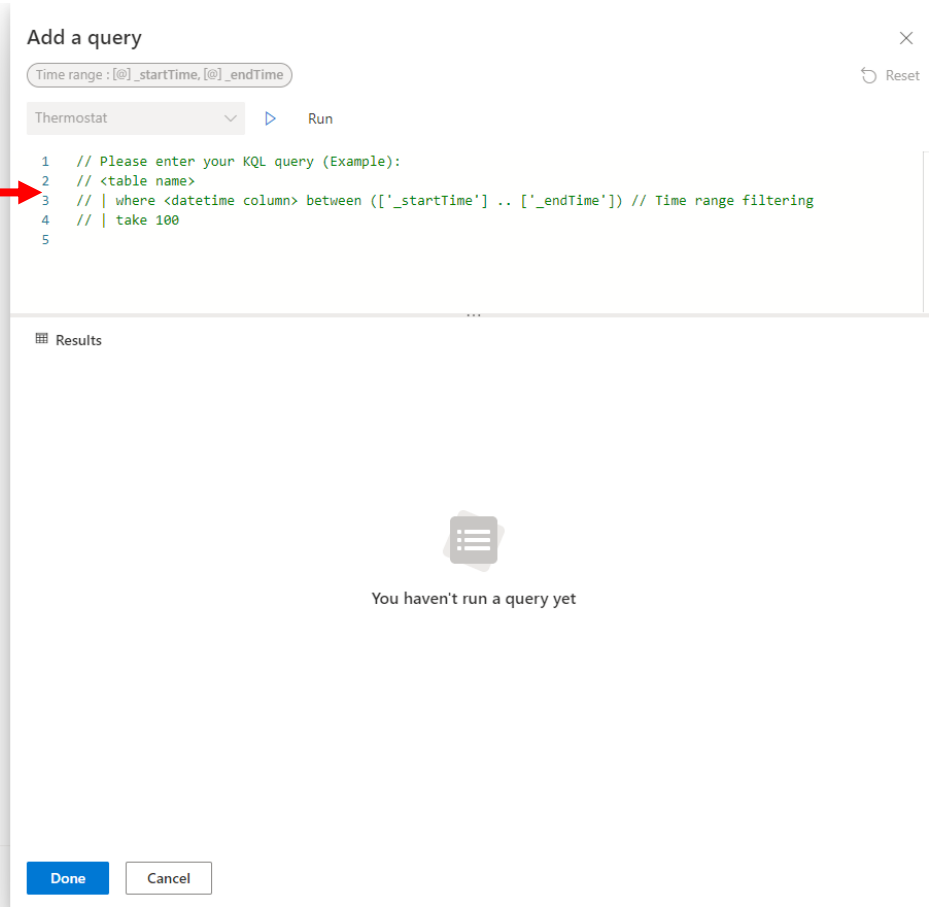
1. Define new parameter



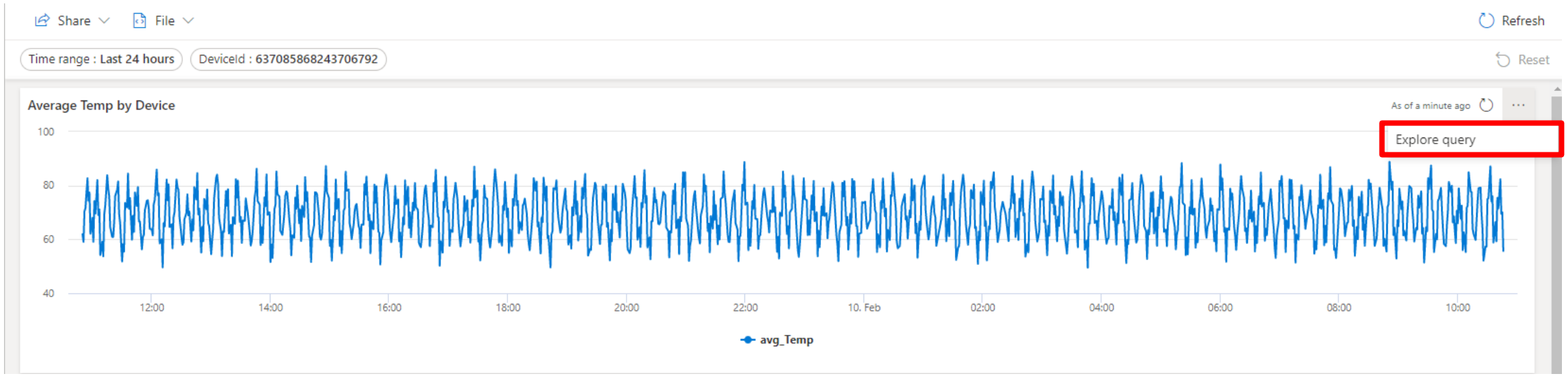
2. Enter values for the new parameter



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



Switching between reporting and exploration



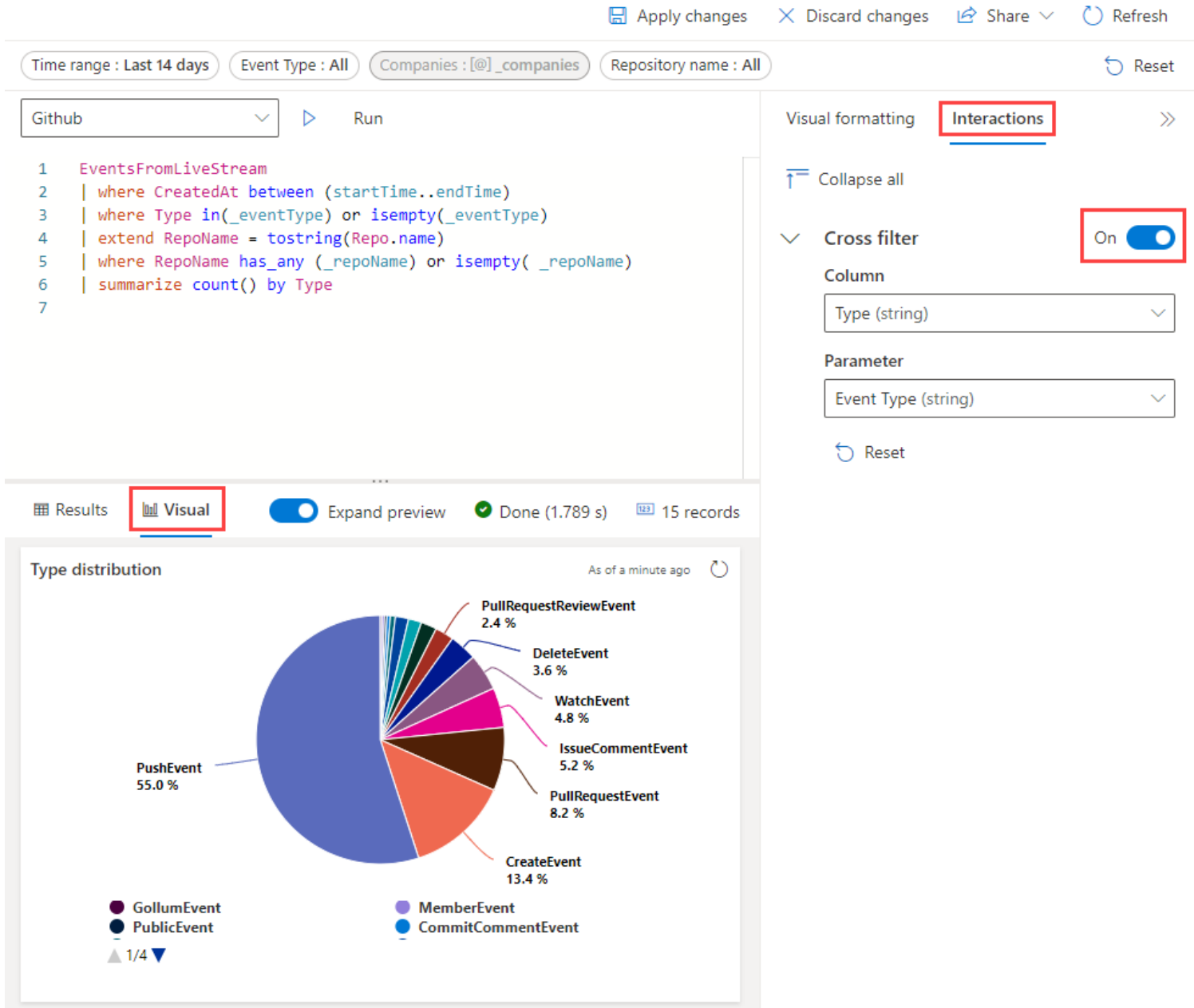
Run Recall **Scope:** @igniteadxsource.eastus2/Occupancy

```
let ['startTime']=datetime('2022-02-09T10:49:14Z');
let ['endTime']=datetime('2022-02-10T10:49:14Z');
let ['Device_Id']='637085868243706792';
// Please enter your KQL query (Example):
// TableName | take 10
Thermostats
| where EnqueuedTimeUTC between (startTime .. endTime)
| where DeviceId == Device_Id
| summarize avg(Temp) by bin(EnqueuedTimeUTC,1m)
| 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.



ADX + Power BI

+

3 ways of connecting Power BI to Azure Data Explorer

Direct Query

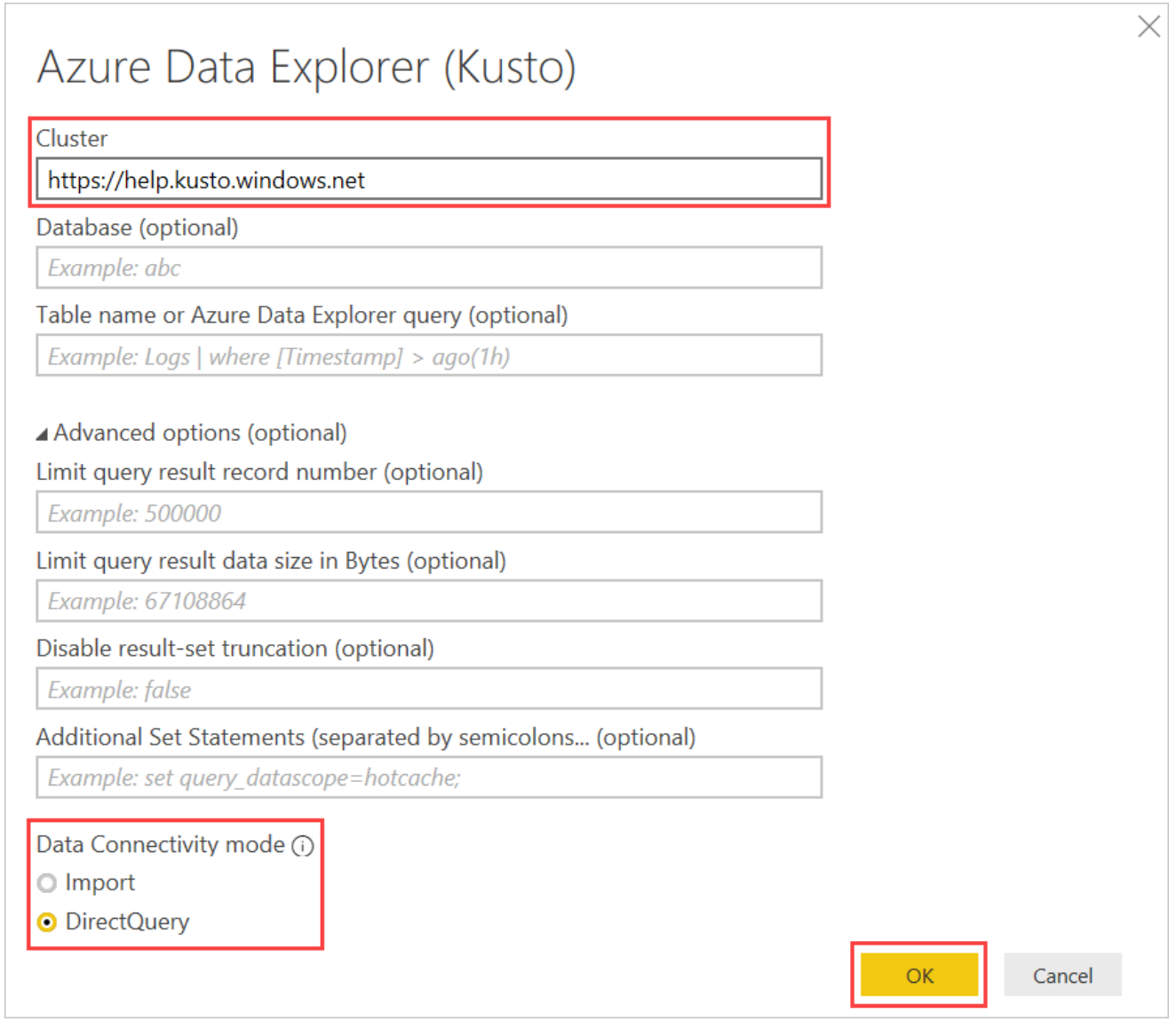
- Use for **large**, frequently updated datasets
- For creating Fact tables as they are large and change often
- Need near real-time data

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

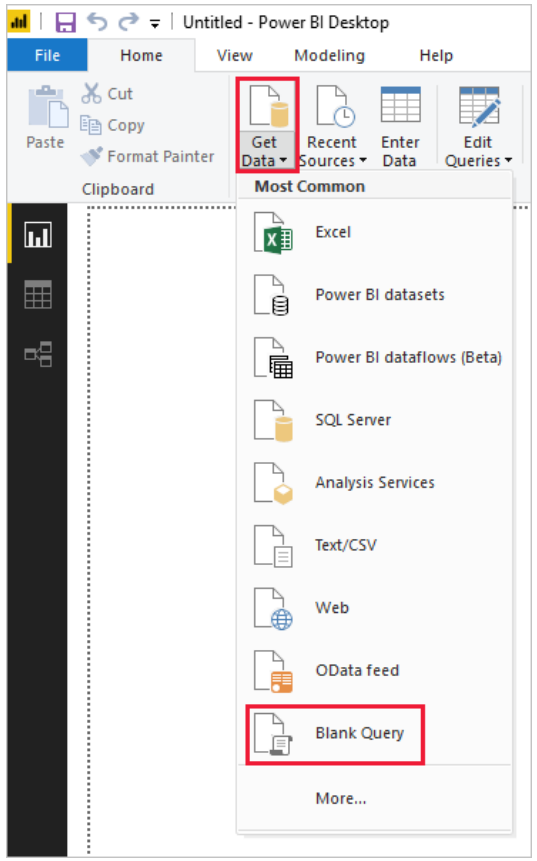
SQL Query

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

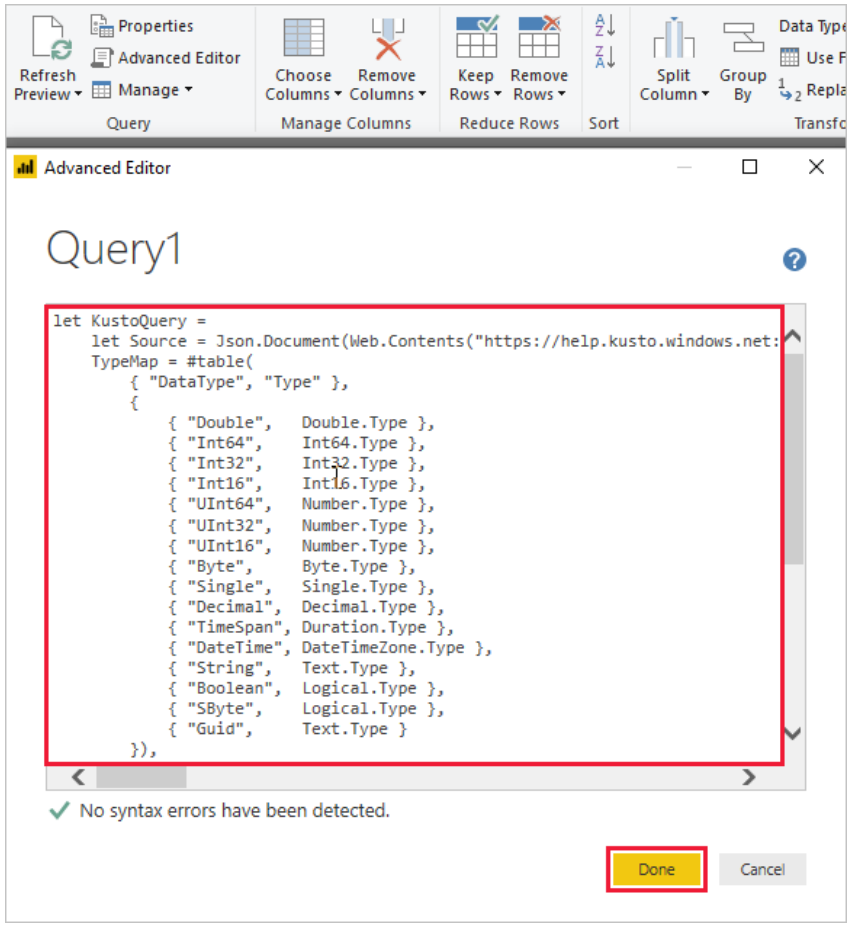


Imported query

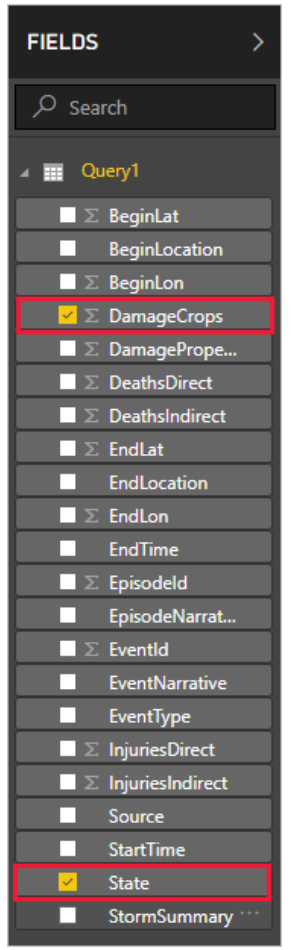
1. Use “Blank Query” to Get Data

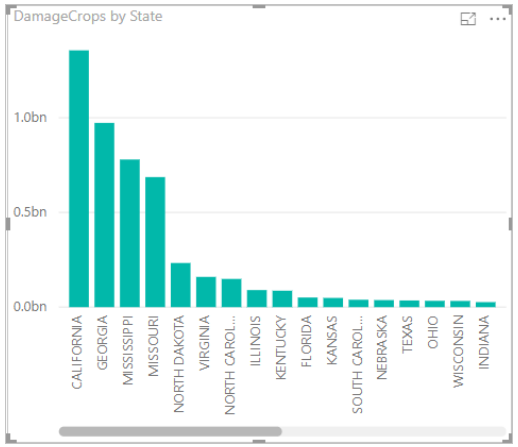


2. Paste KQL as shown below with the required parameters



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.

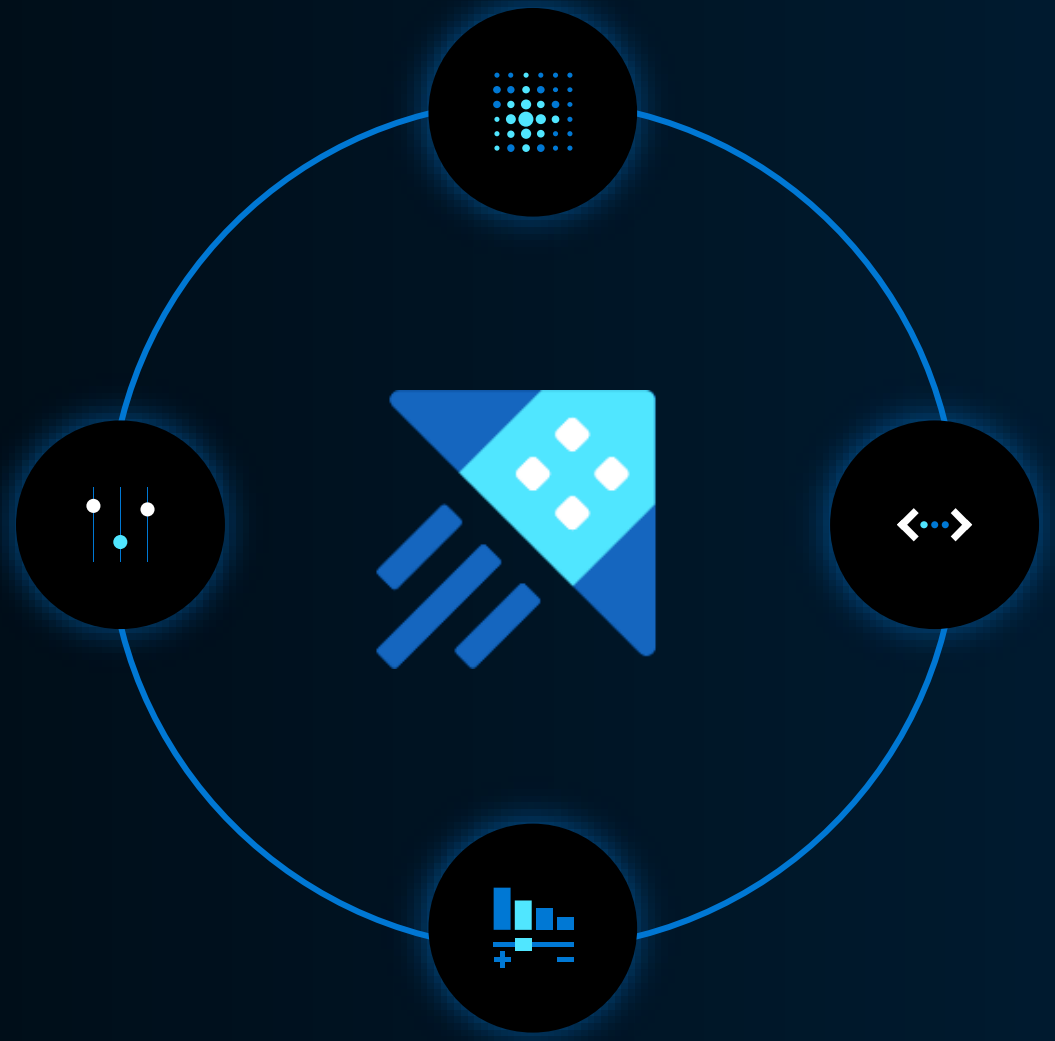
Demo*

- ADX Dashboard: IoT Demo 01
- Power BI: Thermostat
- Grafana: IoT Demo Dashboard

* Estimate 5 minutes

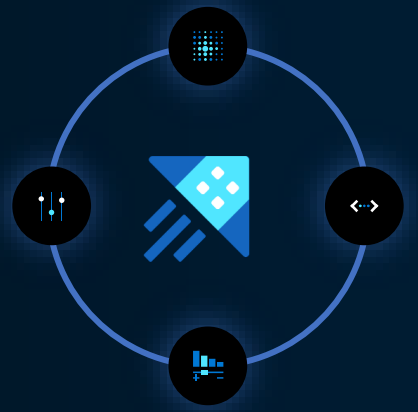
Ops & Mgmt

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

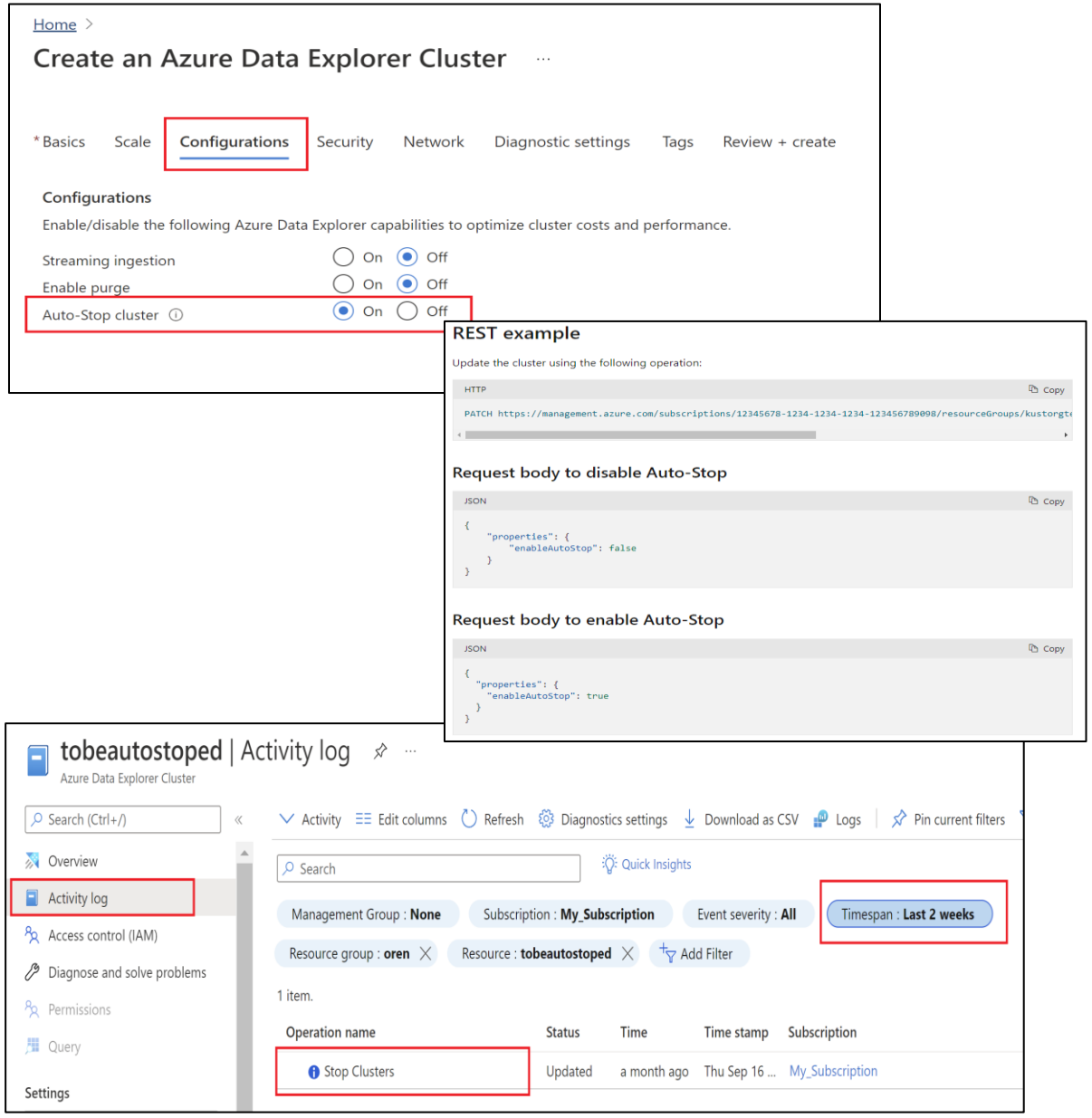
- Larger (1-4 TBs) per node.
- Suited for large volume of data with less intensive compute.
- Optional premium storage (managed disk) attached for hot storage.

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.



Scaling

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

Horizontal (in & out)

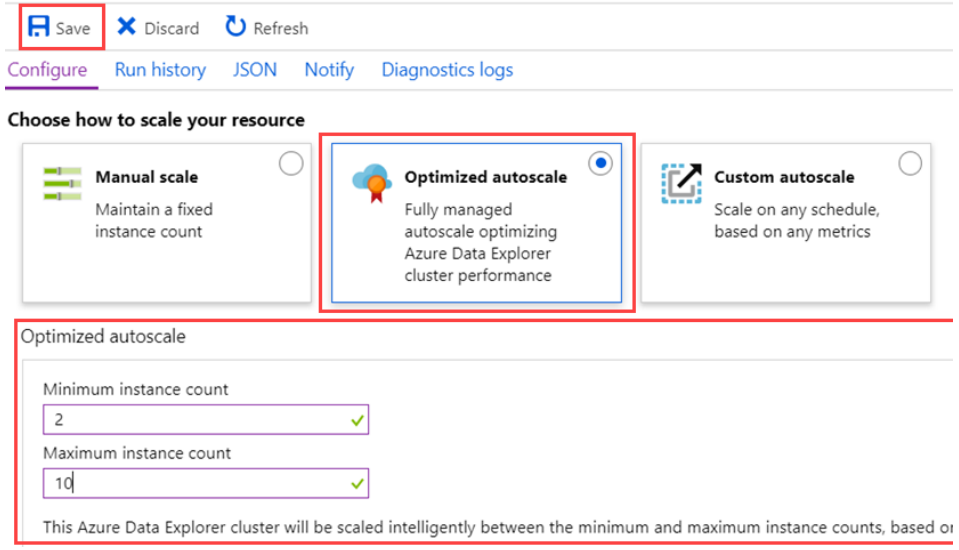
- Manual (default) static capacity.

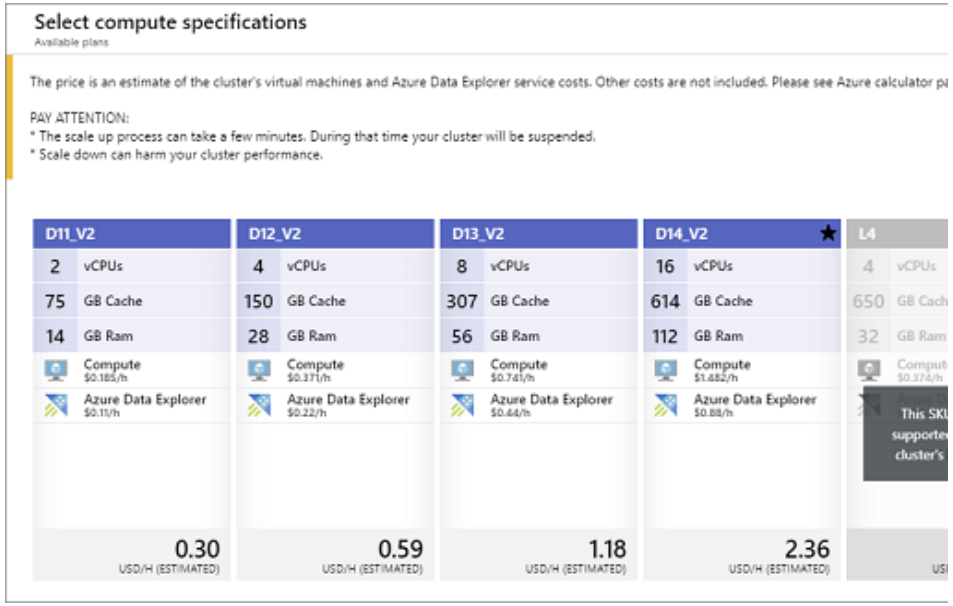
- Optimized autoscale (**recommended**).
  - **Predictive** scaling based on your usage pattern.
  - Will scale in-time to handle the load.
  - Takes **6 days** of data to take affect.
  - If under-utilized will scale in and out if over-utilized.

- Custom autoscale is dynamic based on metrics you specify.

Vertical (up & down).

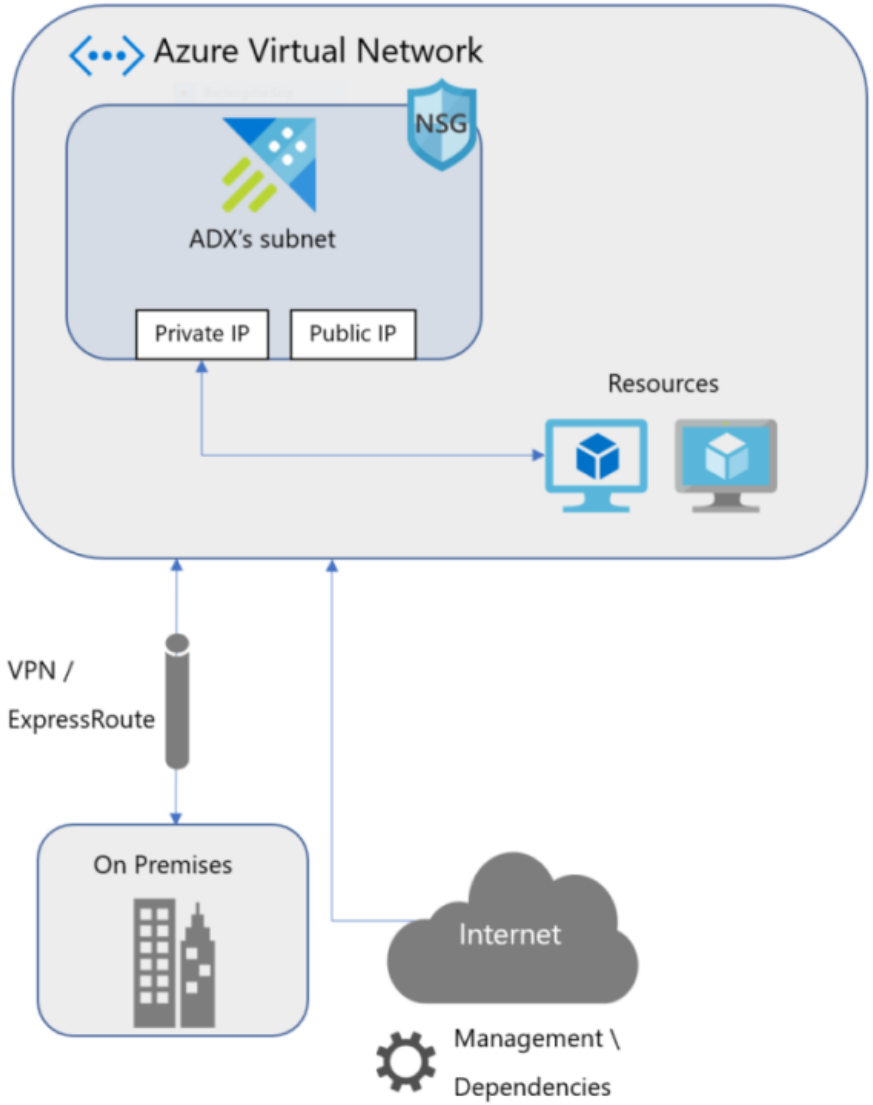
- Scale the SKU.





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 Data Explorer cluster's subnet.
- Secure your data connection sources (event hub and event grid) with service endpoints.



Important: Default NSG rules block access to public IP addresses outside the VNet. Subnet size must be planned in advanced, 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

```
.add database Test users ('aaduser=imikeoein@fabrikam.com') 'Test user (AAD)'

.add database Test admins ('aadgroup=SGDisplayName;fabrikam.com') 'Test group
@fabrikam.com (AAD)'

.add database Test viewers ('aadapp=4c7e82bd-6adb-46c3-b413-
fdd44834c69b;9752a91d-8e15-44e2-aa72-e9f8e12c3ec5') 'Test app on another
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		Use .clear table
Routinely delete old data	Routinely delete old data	Use a retention policy
Bulk delete specific data by extents	Only use if you are an expert user	Use .drop extents
Delete records based on their content	- Storage artifacts that contain the deleted records aren't necessarily deleted - Deleted records **can't** be recovered (regardless of any retention or recoverability settings) - Quick way to delete records	Use soft delete
Delete records based on their content	- Storage artifacts that contain the deleted records are deleted - Deleted records **can't** be recovered (regardless of any retention or recoverability settings) - Requires significant system resources and time to complete	Use purge

Duplicates

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

Kusto

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

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

Business Continuity and Disaster Recovery (BCDR)

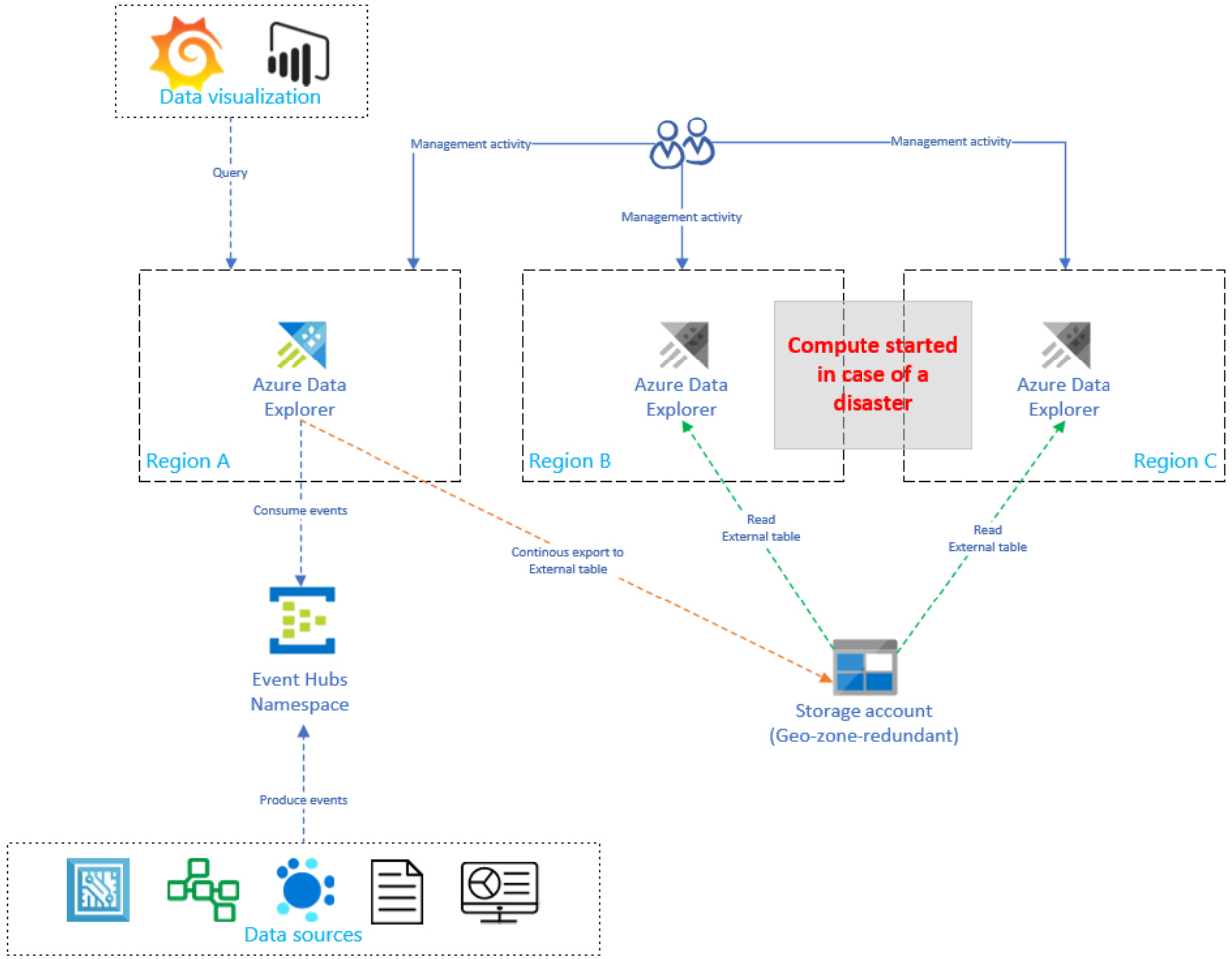
BCDR enables your business to continue operating in the face of a disruption and mitigate events.

- 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

Monitoring capabilities

Metrics

Numerical values.
Example: CPU of the cluster

Diagnostic Logs

Variety of datatypes. Detailed.
Example: details on queries (user details, application details, duration)

Advisor

Actionable recommendations.
Example: enable auto-scale

Resource Health

Notify about service problems.
Example: Is the cluster available or not.

ADX Insights

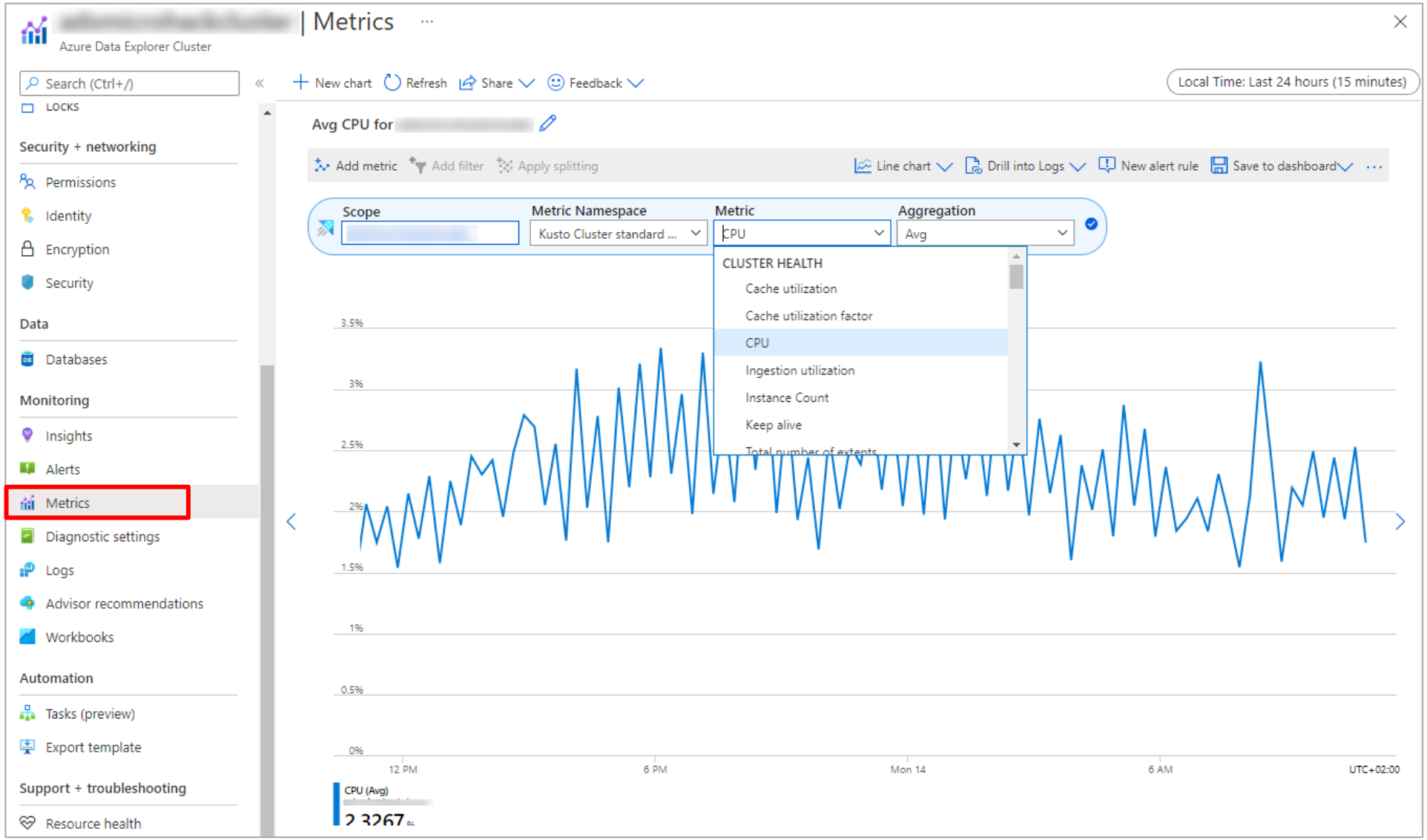
Out-ot-the box monitoring dashboard. Surface data of metrics, logs, Advisor and resource health

Azure monitor metrics

Azure Monitor Metrics is a feature of Azure Monitor that collects **numeric data,** at regular intervals, from monitored resources (like ADX cluster).

Azure Data Explorer metrics provide key indicators as to the health and performance of the cluster resources.

Azure monitor metrics



Azure monitor metrics

Supported ADX metrics (partial list)

Cluster metrics – CPU, instance (node) count, cache utilization
Ingestion metrics - Ingestion result, Ingestion volume,
Streaming ingest metrics - Streaming ingest result, streaming ingest request rate
Query metrics – query duration, total number of concurrent queries
Materialized view metrics – materialized view result, materialized view age seconds

Azure monitor diagnostic logs

Azure Monitor diagnostic logs provide detailed information about the operation of Azure resources.

Azure Data Explorer uses diagnostic logs to store detailed information on ingestion, commands, query, tables, and changes.

You can save the logs in Azure Storage, Event Hub, or Log Analytics workspace. Unlike metrics, diagnostic logs are not enabled by default.
(Normal usage charges for the destination will occur.)

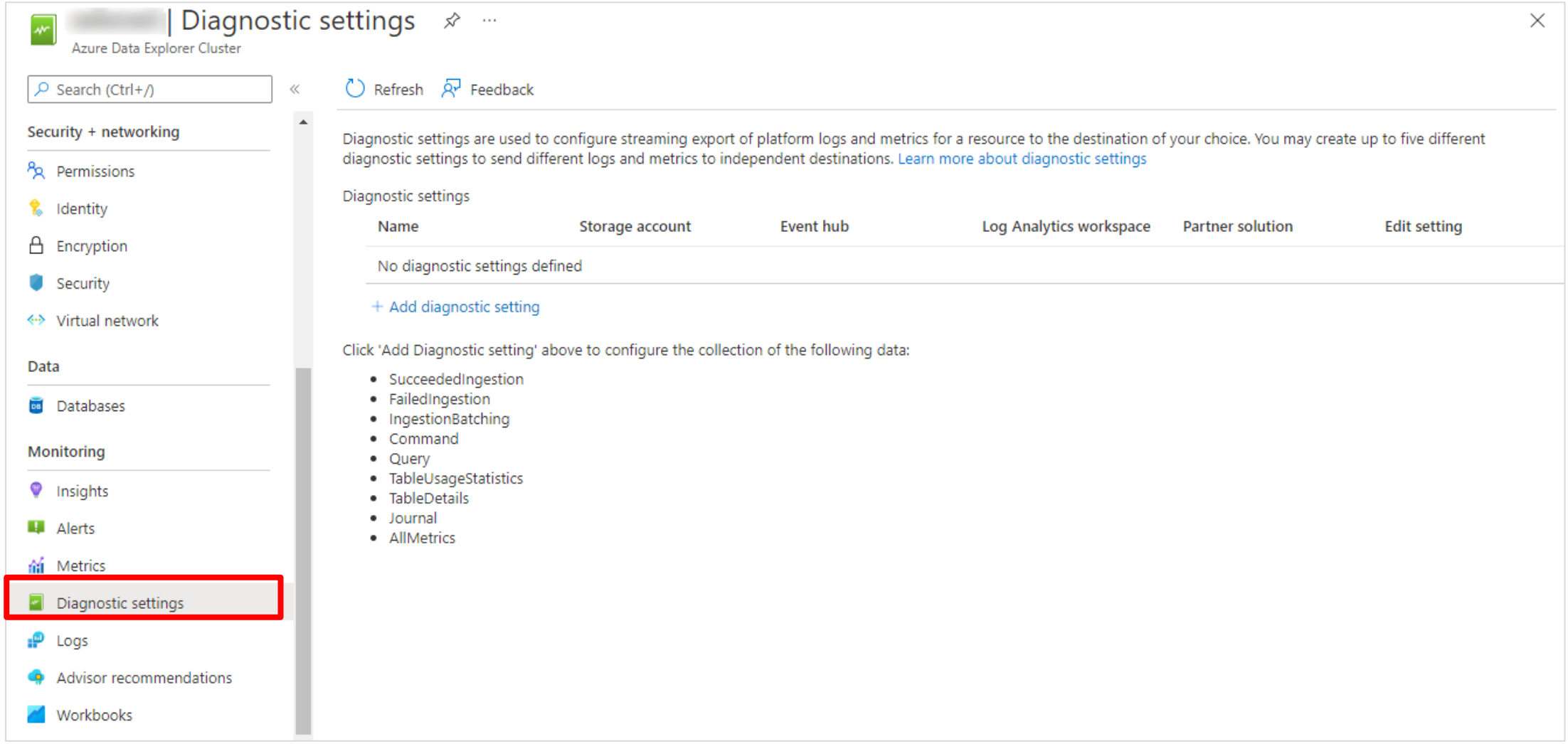
To allow easy data exploration and to make the most of the ADX Insights experience (will be discussed later), the Log Analytics workspace is recommended destination.

Azure monitor diagnostic logs

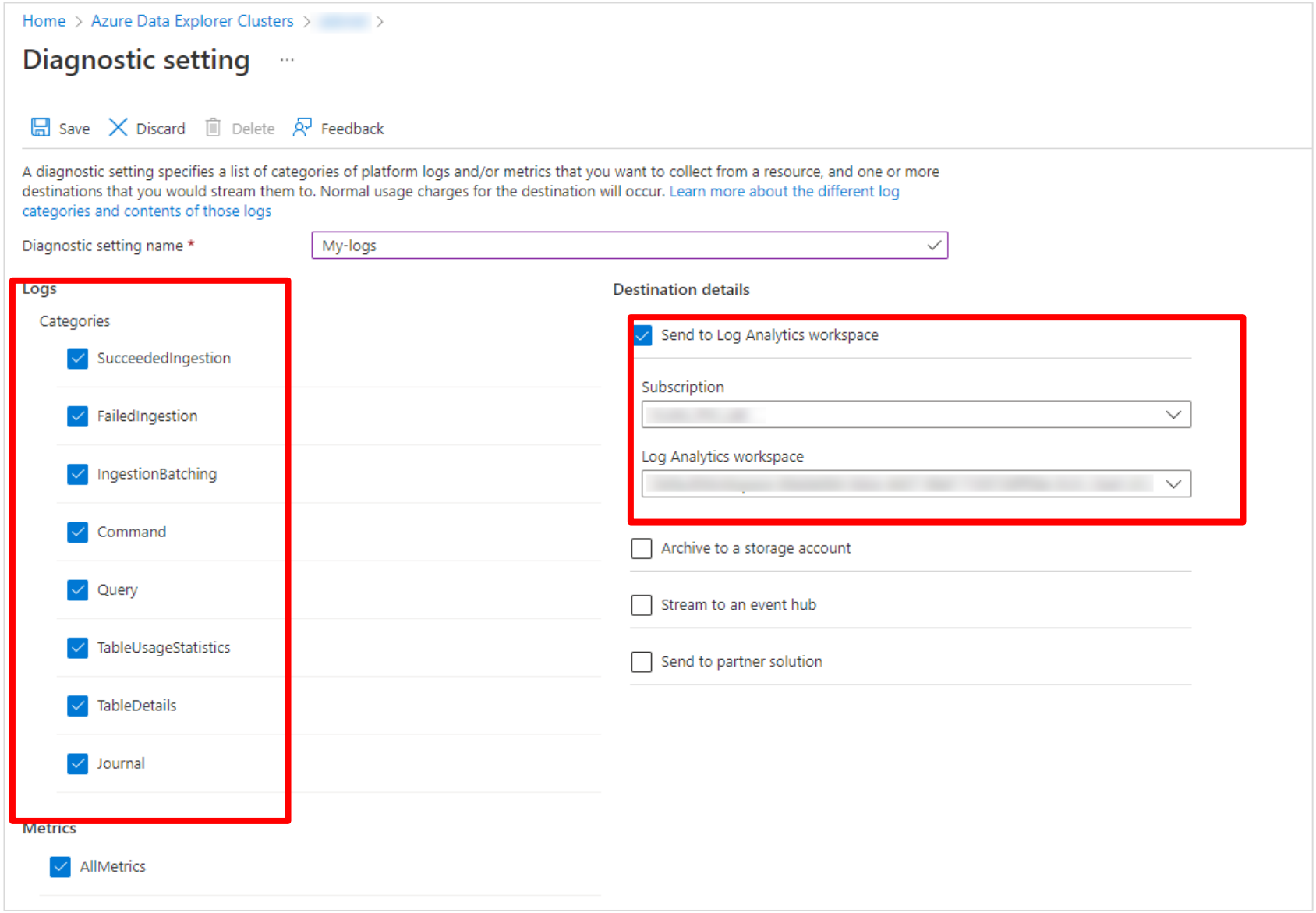
Supported logs

Ingestion - Successful ingestion, failed ingestions, ingestion batching process
User interaction – commands, queries, journal
Tables - usage statistics per table, table details

Azure monitor diagnostic logs



Azure monitor diagnostic logs



Metrics vs Logs

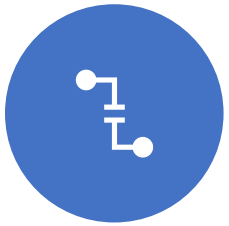
Attribute	Metrics	Logs
Data	Numerical values only	Text or numeric data
Benefits	Analyze with time charts. Lightweight and capable of near-real time scenarios.	Analyze with KQL. Ideal for deep analysis and identifying root cause.
Structure	Standard set of properties including sample time, resource being 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

Informs you of the health of your Azure Data Explorer

Updated every 1-2 minutes and reports the current and past health of your resources

Resource health



Engine not answering (Cluster not available for queries)

Query failures (429, 520, timeout) with high cluster data capacity

Query Failures (Failed / throttled / cancelled)

Low ingestion success rate

Update Policy failures

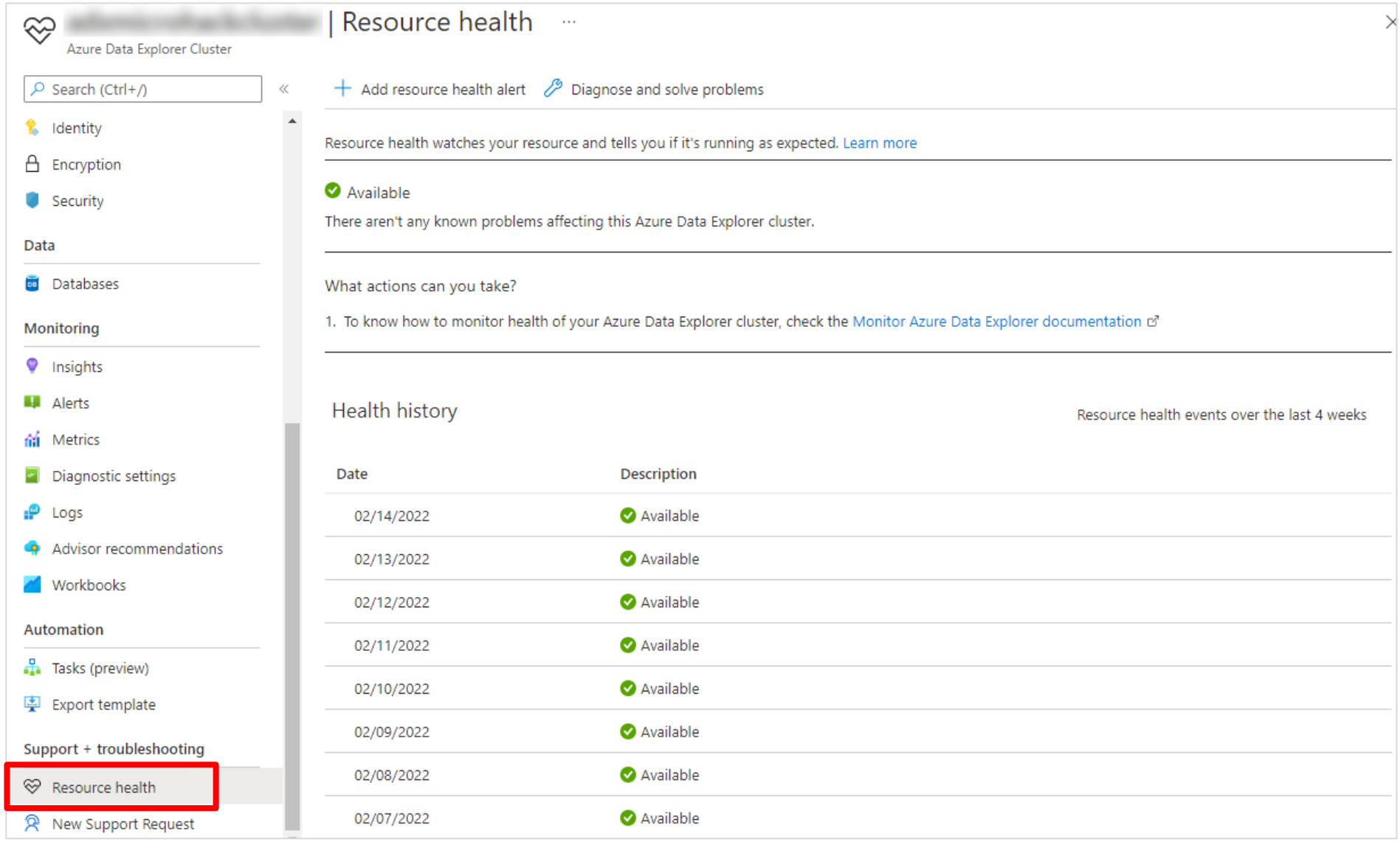
High Ingestion latency (Event Age)

Materialized view degraded/unhealthy

data connection stage latency

Data connections critical attention

Resource health



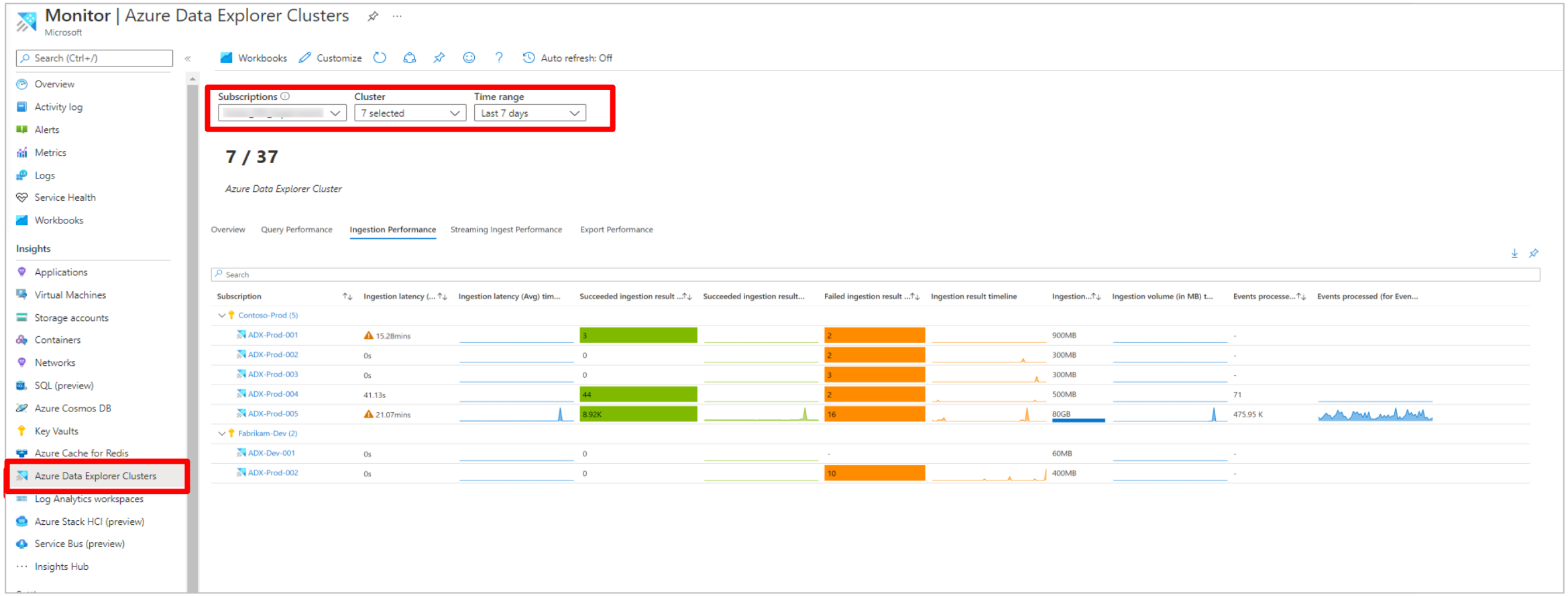
Azure Monitor insights

Comprehensive monitoring of your clusters by delivering a unified view of the cluster performance, operations, ingestions and usage.

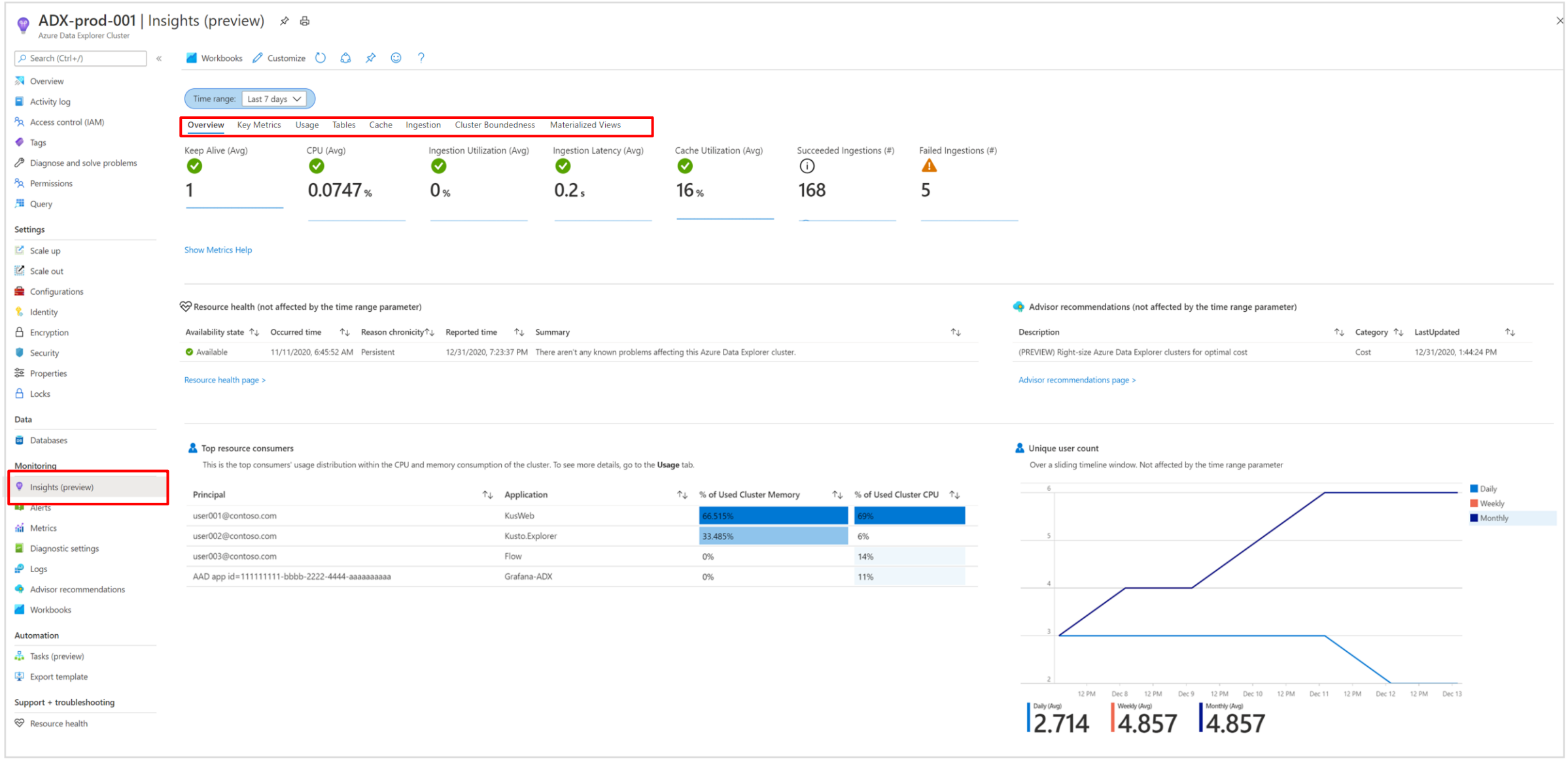
Built on-top of metrics, logs, resource health and Advisor.

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.



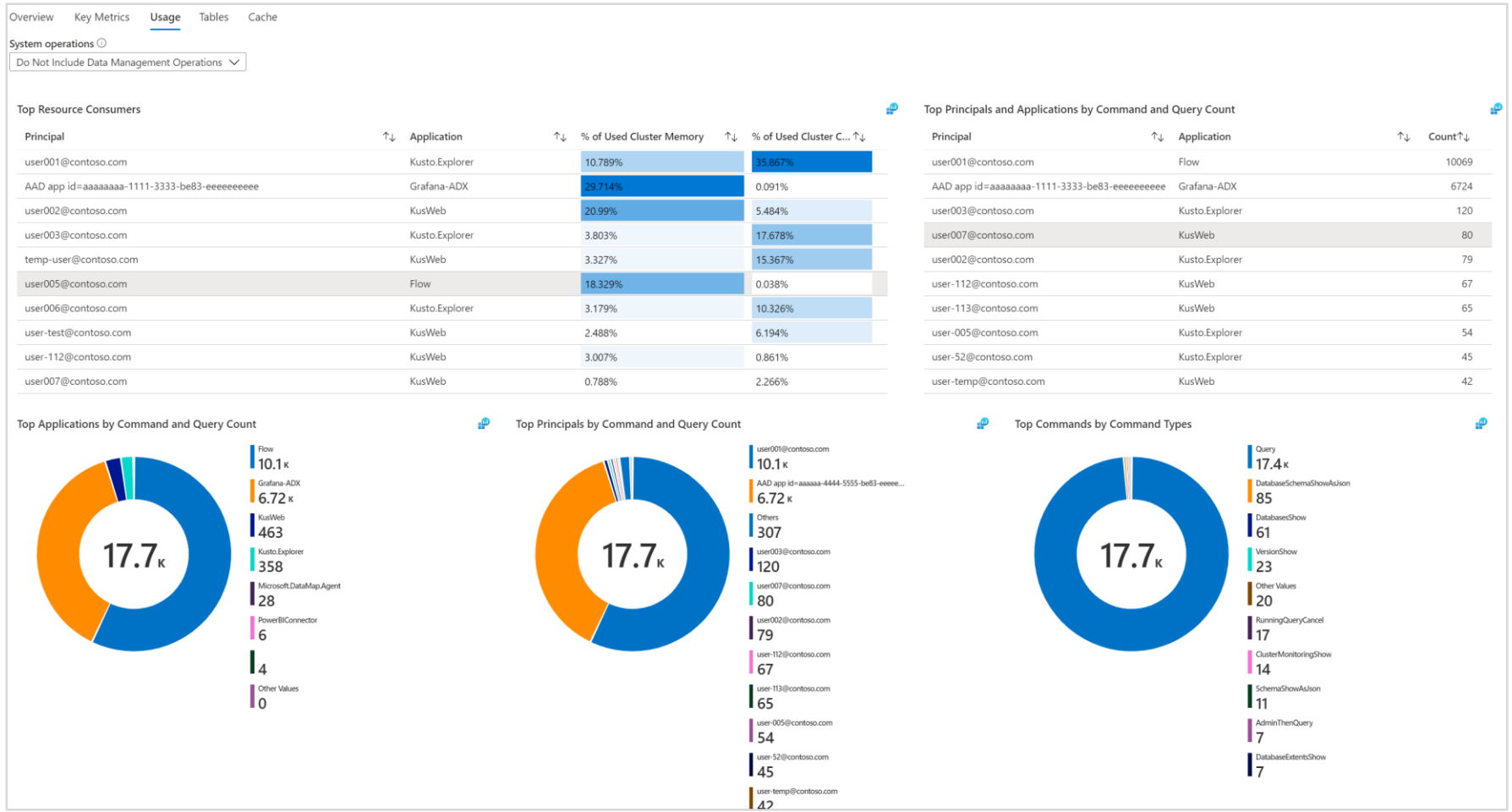
Azure Monitor insights



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:

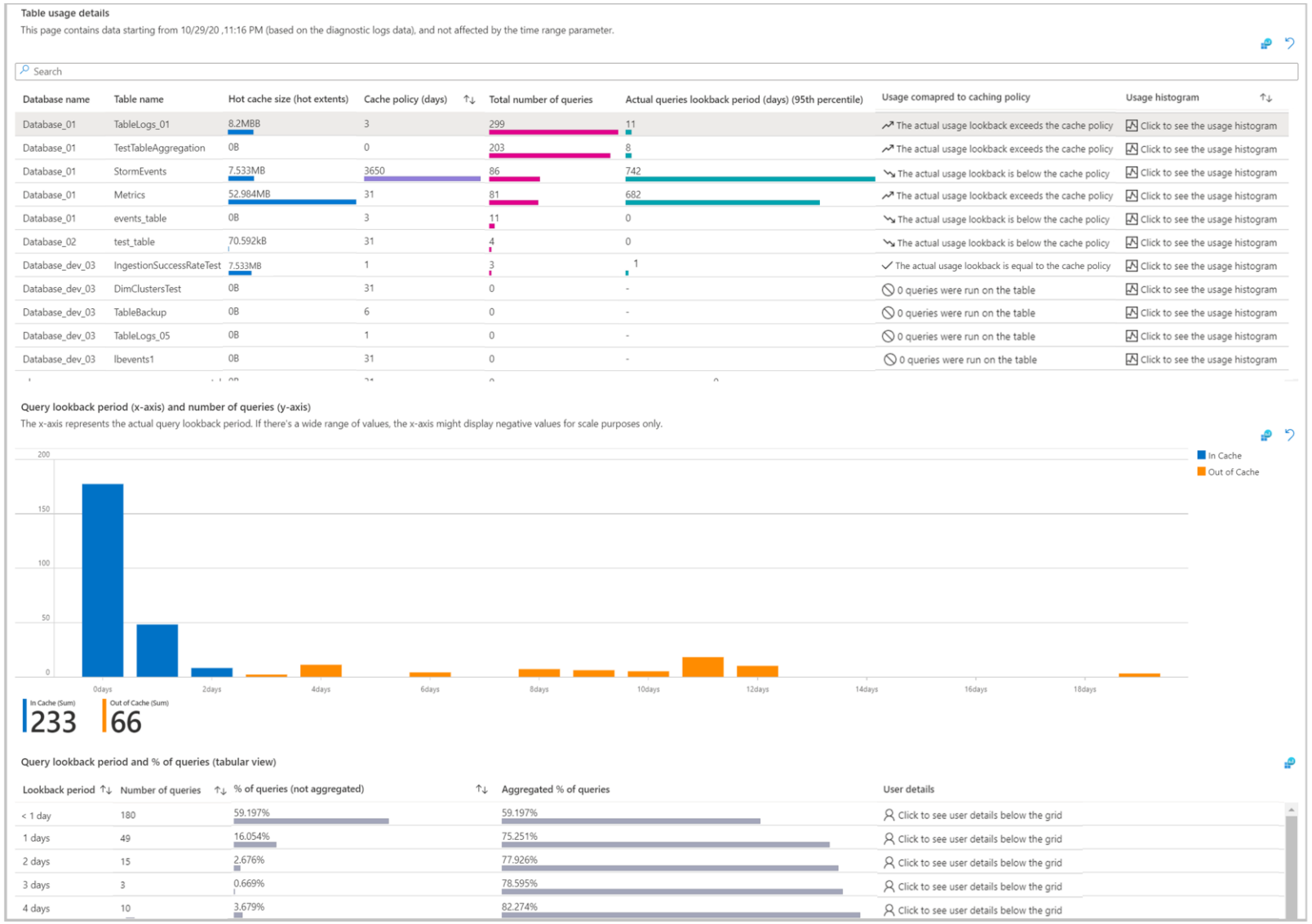
- See which workload groups, users, and applications are sending the most queries or consuming the most CPU and memory.
- Identify users and applications by failed queries.
- Identify recent changes in the number of queries, compared to the historical daily average
- Identify trends and peaks in the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

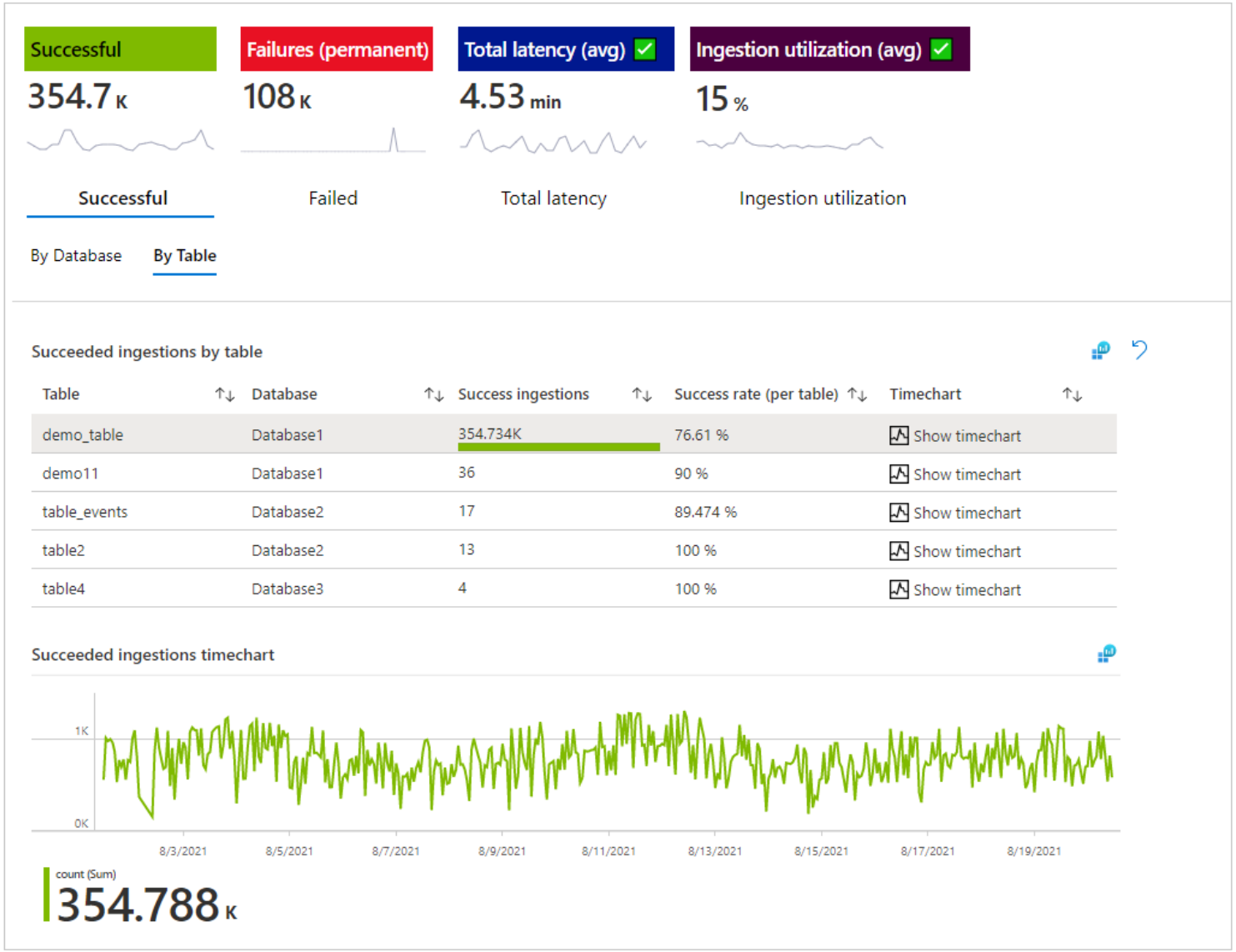
You can identify tables used by the most queries and tables that are not queried at all and adapt the cache policy accordingly.



Azure Monitor insights

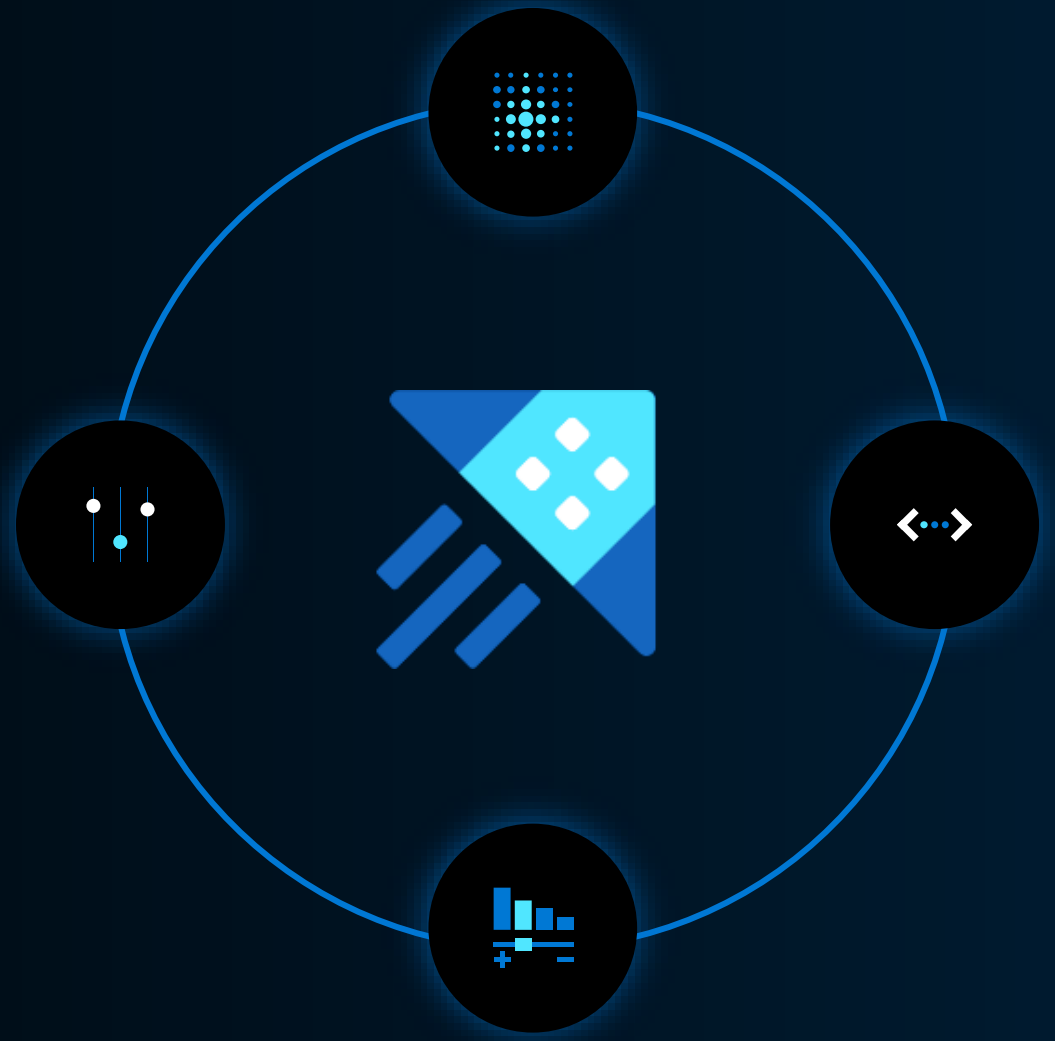
The **Ingestion tab** allows you to monitor the ingestion process.
Here are some questions you can get answers to:

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?



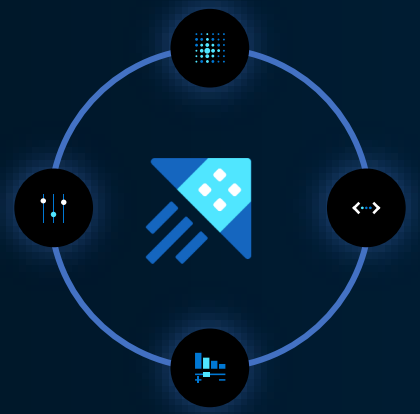
Advanced

Module 9



Objectives

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

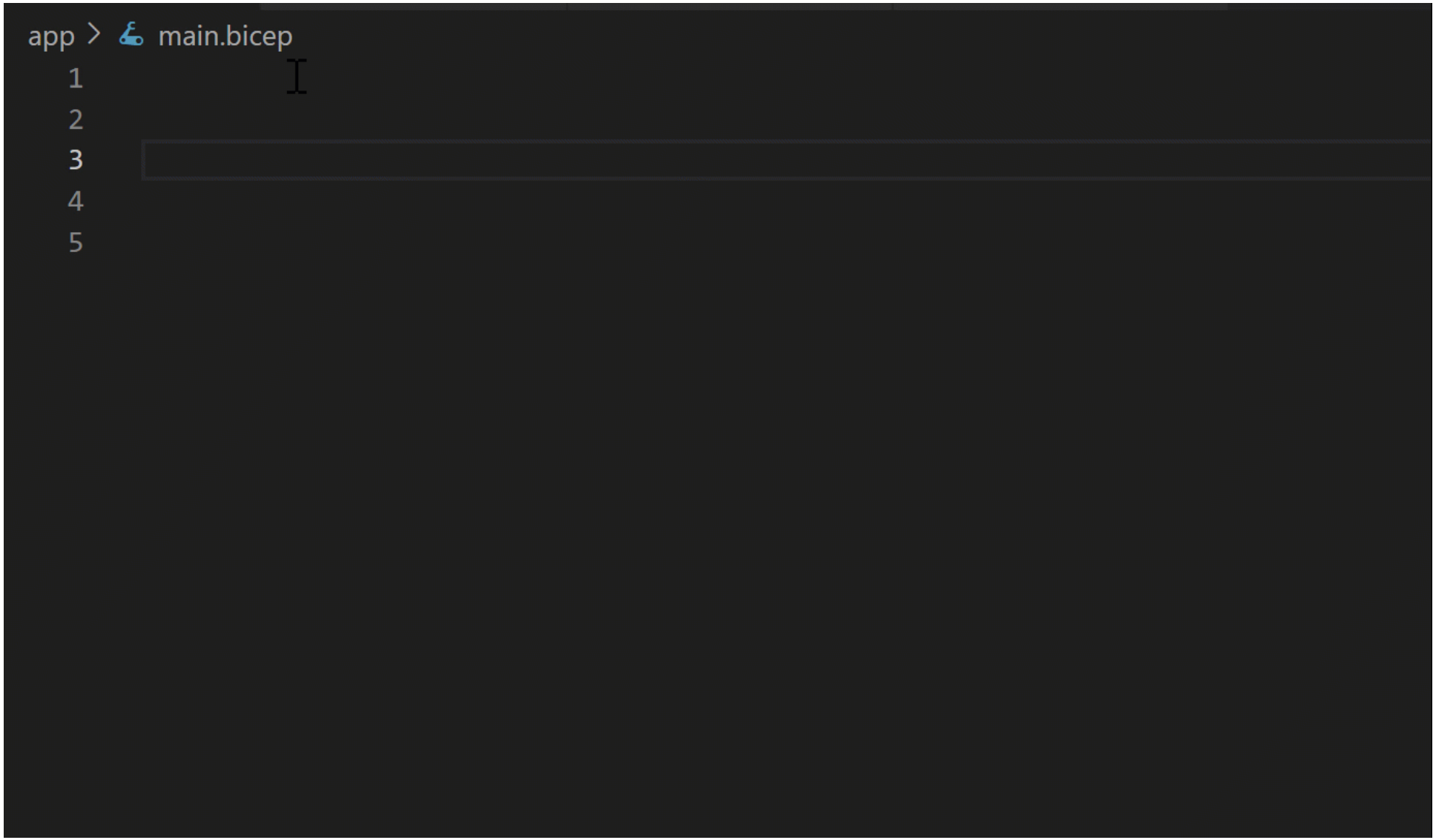


1. Bicep
2. Management Commands
3. Policies
4. Caching
5. Partitioning
6. REST APIs
7. SDKs
8. Optimize for high-concurrency

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



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. Using pipe: Count how many tables are in the database-in-scope:
.show tables
| count

// 2. Using semicolon: Count how many tables are in the database-in-scope:
.show tables;
$command_results
| count

// 3. Using semicolon, and including a let statement:
.show tables;
let useless=(n:string){strcat(n,'-','useless')};
$command_results | extend LastColumn=useless(TableName)
```

Combining queries and control commands

Example of **AdminFromQuery**:

Kusto

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

Drop all `drop-by` tags from extents in table `MyTable`:

Kusto

```
.ingest ... with @'{"tags":"[\"drop-by:2016-02-17\"]"}'

.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

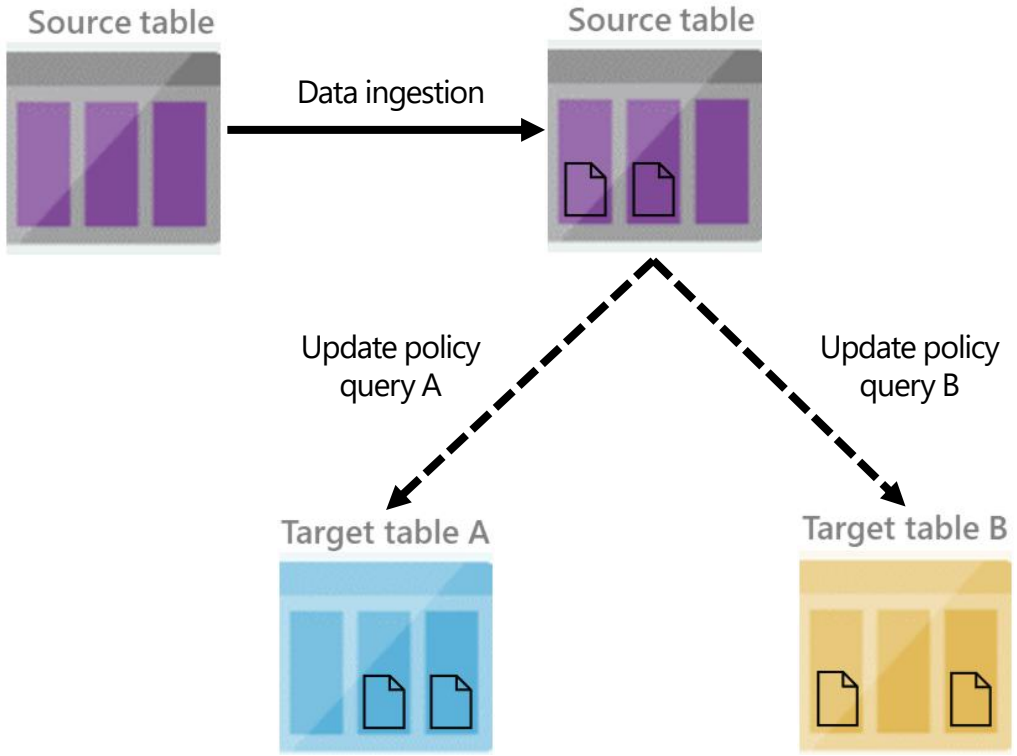
Update policies

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

```
.alter-merge table SourceTable policy retention softdelete = 0sec
recoverability = disabled

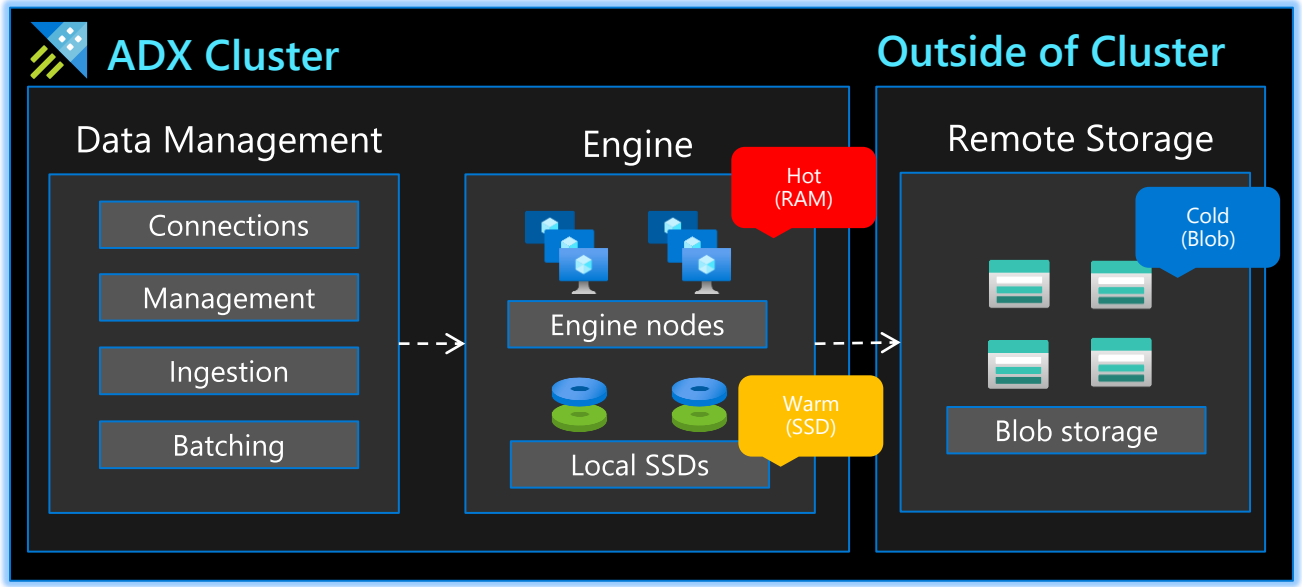
.create function TransformData() {
    SourceTable | extend Counter = Counter * 2 //your business logic
}

.alter table TargetTable policy update
@'[{ "IsEnabled": true, "Source": "SourceTable", "Query": "TransformData()",
"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

```
set query_datascope="hotcache"; MyTable | limit 10

.alter cluster policy caching hot = 30d

.alter cluster policy caching
    hot = 30d,
    hot_window = datetime(2021-01-01) .. datetime(2021-02-01)
```

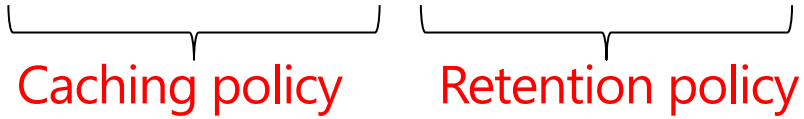
Retention Policy

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



- Recoverability=enabled allows to un-delete data for 14 days.

Kusto

```
.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

```
//Example of RLS statement
.create-or-alter function RLSFunction() {
  let IsManager=current_principal_is_member_of("aadgroup=managers@company.com");
  let AllData=MyTable | where IsManager;
  let PartialData=MyTable | where not(IsManager) and UserColumn==current_principal();
  union AllData, PartialData
}

//and applied to a table
.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

```python
from azure.kusto.data import KustoConnectionStringBuilder, DataFormat
from azure.kusto.ingest import QueuedIngestClient, IngestionProperties, FileDescriptor,
BlobDescriptor

ip = IngestionProperties(database="IoTAnalytics", table="iotingest",
data_format=DataFormat.CSV)
client =
QueuedIngestClient(KustoConnectionStringBuilder.with_interactive_login("https://adxiot.eastus.
kusto.windows.net"))

# ingest from file. ie. the raw (uncompressed) size of the data is (15 KB)
fd = FileDescriptor("iotingest.csv", 15360)
client.ingest_from_file(fd, ingestion_properties=ip)
client.ingest_from_file("iotingest.csv", ingestion_properties=ip)

# ingest from blob. ie. the raw (uncompressed) size of the data is (50 KB)
bd = BlobDescriptor("https://adlsg2.blob.core.windows.net/cntr/iotingest.csv.gz?sas", 51200)
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

* Estimate 5-10 minutes

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

Key Takeaways

Use Azure Data Explorer for building your IoT Analytics:

- Near-Real-Time ingestion from IoT Hub, Event Hub, Kafka, and Storage
- Built-in Time Series and Machine Learning
- Fast queries
- Big Data and Partner integrations
- Gives you deep control of data sources, data schema, data life cycle, and cost!

Connecting data & unlocking insights is key

Distributed Query Engine

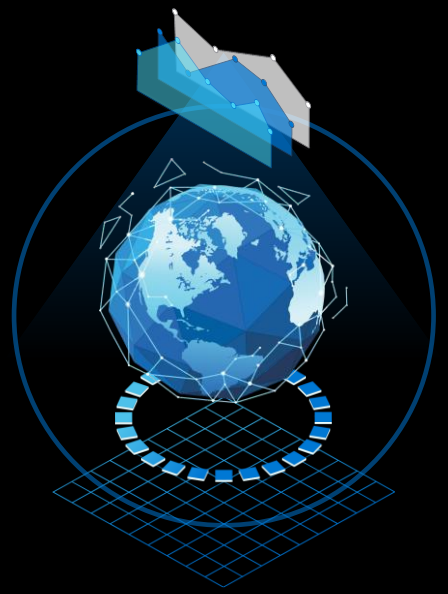
- Storage and Compute isolated
- Auto compression, indexing, optimization
- Semi-structure (JSON) and unstructured data (free text) indexing
- Data is sharded, distributed, cached across nodes on local SSDs and persisted on cloud storage
- Ingestion time-based partitioning + custom partitioning
- Hot 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

Azure **Data Explorer**

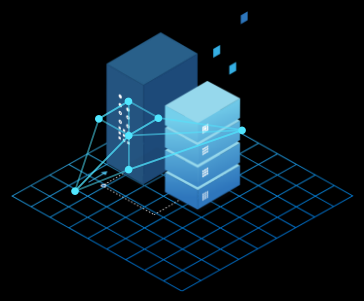
- Tailored for IoT use cases –
  Time series analysis, telemetry, log and event data
- Designed for data exploration –
  Run ad-hoc queries on TBs of data in seconds
- Complex log processing –
  Large scale logfile processing, ELK replacement
- Proven technology –
  Used MS-internally since 2015
- Fully managed PaaS service
- Optimized for streaming data –
  Up to 200MB ingestion per second per node
- Structured, semi-structured and free-text data –
  Relational tables with dynamic schema support

Pricing

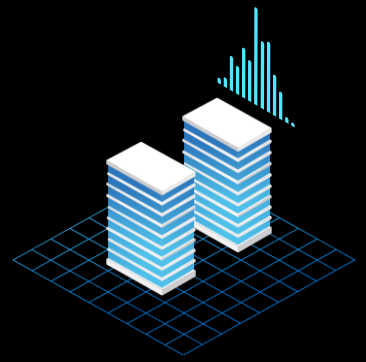
Dedicated capacity

- Freely explore your data - not bound by cost per query
- Predictable cost
- Decoupled storage and compute
- Easy to switch between SKUs after provisioning
- Ability to pause the compute

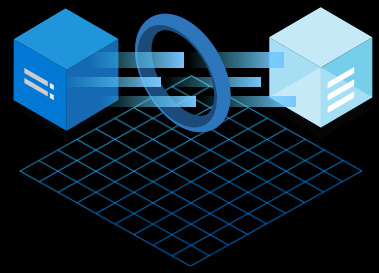
Three main components



Compute
vCores meters



Storage
Data Stored meters



Network
Passthrough meters

Resources

- Documentation: aka.ms/adx.docs
- Reference Architecture: aka.ms/adx.iotrefarch
- Tech Blogs: aka.ms/adx.techblogs
- PG Ask me Anything: aka.ms/ama/azuredataexplorer
- Free personal cluster: aka.ms/adx.free
- Cost Estimator: aka.ms/adx.cost or aka.ms/adx.cost.old

Microsoft

Thank you

aka.ms/adx.iot.eval