

# Lab: Performing Real-Time Intelligence using Microsoft Fabric

## Objective

- Create **KQL database** in your Fabric enabled Power BI workspace
- Run powerful KQL queries to explore the data via **KQL Queryset**
- Visualize data in Fabric **Real Time Dashboards**

## Exercise 1: Database Creation, Data Ingestion and Exploration

## Exercise 2: Ingest data from Azure Storage Account

## Exercise 3: Starting with the basics of KQL

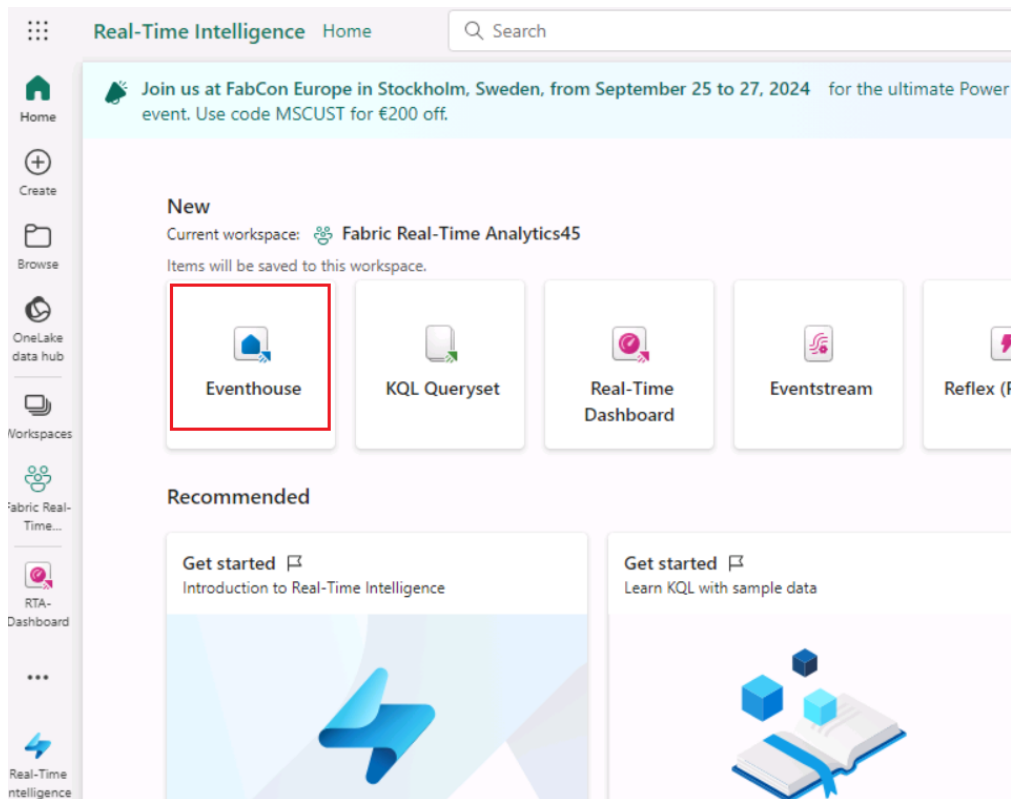
## Exercise 4: Explore and Transform Data

## Exercise 5: Eventstreams

## Exercise 1: Database Creation, Data Ingestion and Exploration

### Task 1: Create a KQL Database

Kusto query language (KQL) is used to query static or streaming data in a table that is defined in a KQL database. To analyze the sales data, you must create a table in a KQL database and ingest the data from the file.



In the **New Eventhouse** dialog box, enter the **Eventhouse name** as **FabricRTA** (or new database with a name of your choice) and click on **Create** button.

### New Eventhouse

Eventhouse name

Create

Cancel

### Task 2: Create a KQL queryset

The KQL Queryset exists within the context of a workspace. A new KQL queryset is always associated with the workspace you're using when you create it.

1. Select **Your workspace** in the left navigation pane.

2. Select the **New** drop-down, and from there select **KQL Queryset**
3. In the **New KQL Queryset** dialog box, enter the **KQL Query name** as querysetXX (or new database with a name of your choice) and click on **Create** button.

## New KQL Queryset

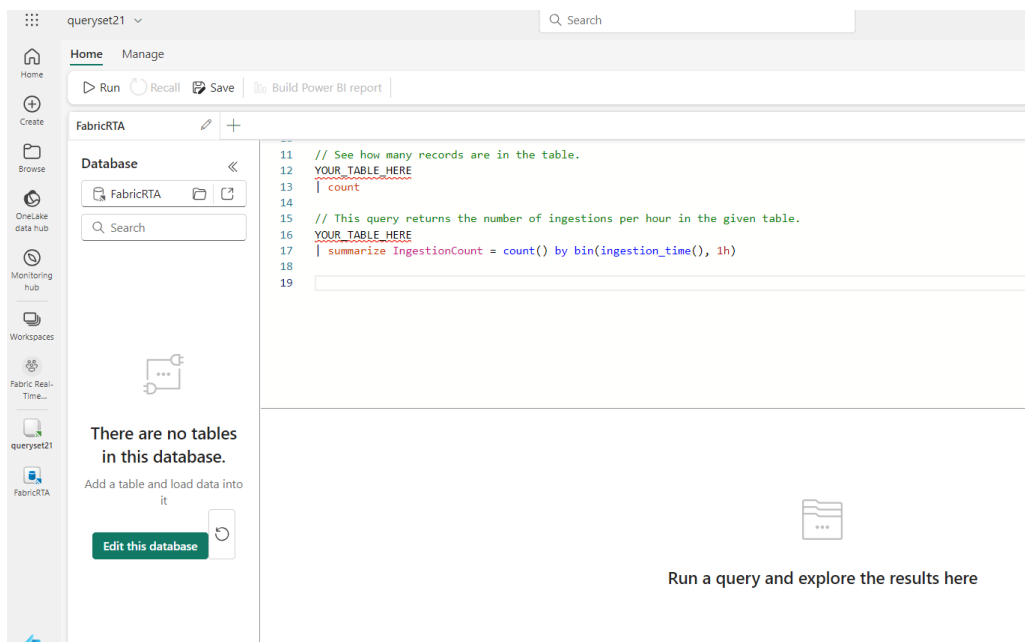
KQL Queryset name \*

queryset21 **1**

Create

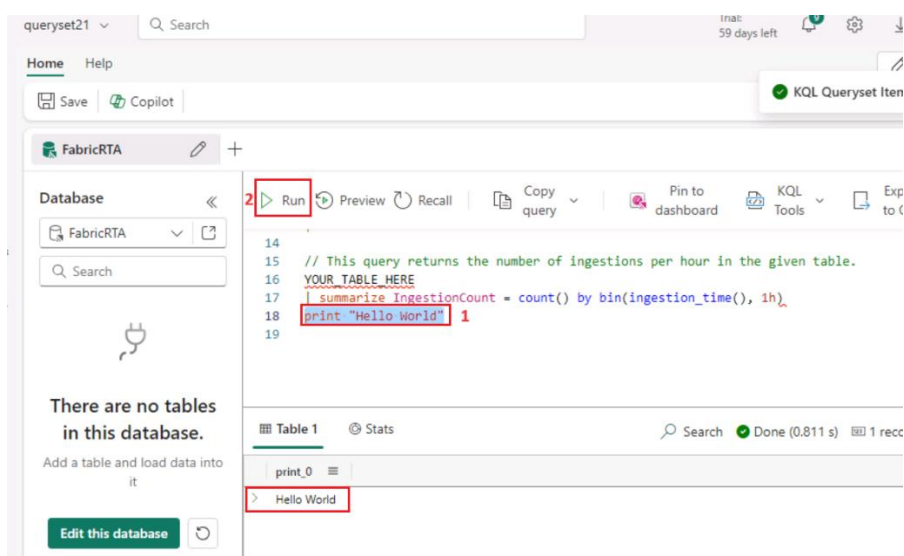
**2** Cancel

4. In **OneLake data hub** pane ,select **FabricRTA** KQL Database and click on **Select** button.



5. Now you can write a simple KQL query:

```
print "Hello World"
```



6. Highlight the line with the code and hit the **Run** button. The query will be executed and its result can be seen in the result grid at the bottom of the page.

queryset21

Search

Inat: 59 days left

Home

Help

Save

Copilot

KQL Queryset Item

FabricRTA

+

Database

FabricRTA

Search

There are no tables in this database.

Add a table and load data into it

Edit this database

Run

Preview

Recall

Copy query

Pin to dashboard

KQL Tools

Export

14

15

16

17

18

19

// This query returns the number of ingestions per hour in the given table.

YOUR\_TABLE\_HERE

| summarize IngestionCount = count() by bin(ingestion\_time(), 1h)

print "Hello world" 1

Table 1

Stats

Search

Done (0.811 s)

1 rec

print\_0

>Hello World

## Exercise 2: Ingest data from Azure Storage Account

Data ingestion to KQL Database is the process used to load data records from one or more sources into a table. Once ingested, the data becomes available for query.

KQL Database supports several ingestion methods, including Eventstream, Fabric Pipeline, and Fabric Dataflow. Also available with Azure Data Factory and Event Hubs.

### Task 1: Create the raw table - logsRaw

Ingest data using one-click ingestion from Azure Blob Storage to your KQL Database.

1. In KQL queryset pane, replace all the code in the **cell** with the following code and click on ► **Run** button to create a table

Copy

```
.create table logsRaw(  
    Timestamp:datetime,  
    Source:string,  
    Node:string,  
    Level:string,  
    Component:string,  
    ClientRequestId:string,  
    Message:string,  
    Properties:dynamic  
)
```

The screenshot shows the Azure Data Explorer interface. The top bar includes a search bar, a 'Run' button (highlighted with a red box and labeled '2'), and other navigation icons. The main pane displays a KQL query (lines 1-10) to create a table named 'logsRaw' with columns: Timestamp (datetime), Source (string), Node (string), Level (string), Component (string), ClientRequestId (string), Message (string), and Properties (dynamic). The query is highlighted with a red box and labeled '1'. Below the query, the results pane shows 'Table 1' with 1 record. The table structure is displayed as follows:

TableName	Schema	DatabaseName	Folder	DocString
logsRaw	{["Name":"logsRaw","OrderedColumns":[{"Name":"Timestamp","Type	34fee477-2d48-468c-		

2. Click on refresh the page to see the new table on the left.

## Task 2: Use the “One-click” User Interface to ingest data from Azure blob storage

You need to analyze the system logs for Contoso, which are stored in Azure blob storage.

1. Select **FabricRTA** database in the left navigation pane.

The screenshot shows the Microsoft Fabric query editor interface. The left navigation pane contains a list of items: Home, Create, Browse, Workspaces, Fabric Real-Time..., queryset21, and **FabricRTA** (highlighted with a red box). The main area displays the **FabricRTA** database with a search bar and a list of tables including **logsRaw**. The right pane shows a SQL query being edited:

```
1 .create table log
2     Timestamp:dat
3     Source:string
4     Node:string,
5     Level:string,
6     Component:str
7     ClientRequest
8     Message:stin
9     Properties:dy
10 )
```

Below the query editor, a table structure is shown:

TableName	Schema
logsRaw	{"Name": "I

← ↻ 🔒 <https://app.fabric.microsoft.com/groups/a7d4e9d1-87ce-44a8-a8b3-59690fe83>

FabricRTA ▾ 🔍 Search

Home  
Create  
Browse  
Workspaces  
Fabric Real-Time...  
queryset21  
FabricRTA  
...  
Real-Time

**Eventhouse**

↻ Refresh + New database ⌚ Minimum consumption

**FabricRTA** ⓘ

🕒 System overview

🗄 Databases

📊 Monitoring Coming soon

🔍 Search

**KQL databases** +

🗄 FabricRTA

**System overview**

✅ Status: Running

**Storage**

❄️  
0 B  
OneLake standard storage

**OneLake cache storage**

OneLake standard storage

2. In **FabricRTA** database, under the **Home** tab, navigate and click on **Get data** in the command bar, then select **Azure Storage**.

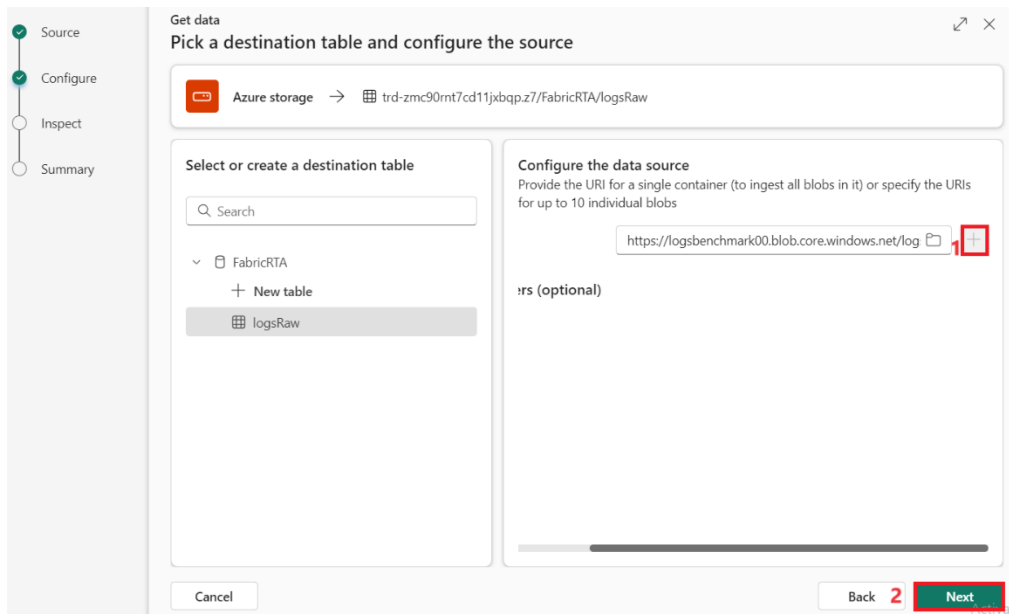
The screenshot shows the Microsoft Fabric interface. On the left is a navigation pane with icons for Home, Create, Browse, OneLake data hub, Monitoring hub, Workspaces, and Fabric Real-Time... The main area has tabs for 'Home' and 'Manage'. Under 'Home', there are buttons for 'Refresh', '+ New', 'Get data' (highlighted with a red box and a red '1'), and 'New related item'. A dropdown menu for 'Get data' is open, showing options: 'One time' (Sample, Local file, OneLake, Azure Storage (highlighted with a red box and a red '2'), Amazon S3) and 'Continuous' (Event Hubs, Eventstream, Pipeline, Dataflow). On the right, the 'Database: FabricRTA' details panel is visible, showing 'Created by: MOD Administrator', 'Region: westus3', 'Created on: Today, 5m ago', 'Last ingestion: -', 'Query URI: Copy URI', 'Ingestion URI: Copy URI', and 'OneLake folder: Inactive'.

3. Then use the wizard to import the data into a new table by selecting the following options:
4. In the **Get data** tab, select the existing table as **logsRaw**. Under the Configure the data source tab enter the **URI** : <https://rtainaday.blob.core.windows.net/logsbenchmark-onegb/2014?sp=rl&st=2025-03-12T19:25:59Z&se=2099-03-13T03:25:59Z&spr=https&sv=2022-11-02&sr=c&sig=XxQnOYzmgarwNi8xhRF7zbRMuY8TOWXmz5CVCJMiWOM%3D>

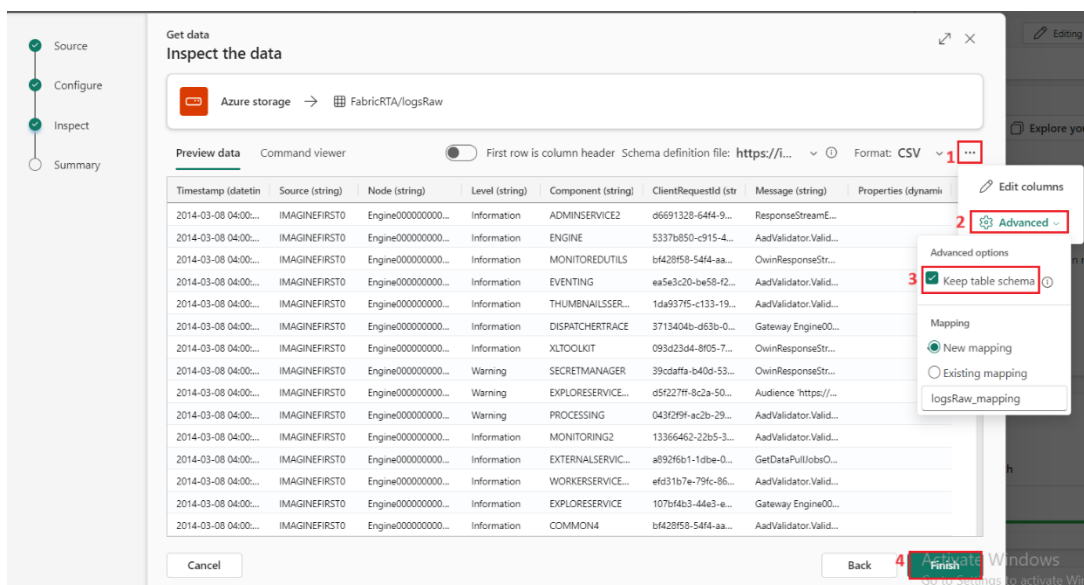
The screenshot shows the 'Get data' wizard in Microsoft Fabric. The left sidebar has a progress indicator with steps: Source, Configure (active), Inspect, and Summary. The main area is titled 'Get data' and 'Pick a destination table and configure the source'. It shows 'Azure storage' selected as the source, with the path 'trd-zmc90mt7cd11jxbqp.z7/FabricRTA/logsRaw'. Under 'Select or create a destination table', 'logsRaw' is selected (highlighted with a red box and a red '1'). Under 'Configure the data source', the URI 'https://logsbenchmark00.blob.core.windows.net/' is entered (highlighted with a red box and a red '2'). The bottom of the wizard has 'Cancel', 'Back', and 'Next' buttons.

5. In the **Get data** tab, click on the + and click on **Next** button.

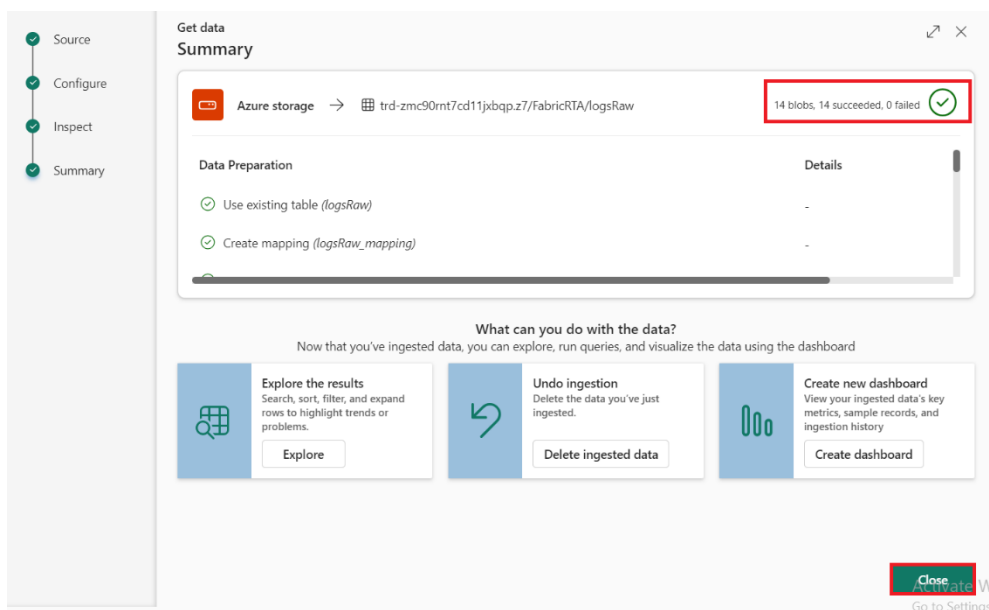




6. In Get data tab under the Inspect the data click on the horizontal ellipses (...) beside Edit columns, select the **Advanced** and check the **Keep table schema**. Click on the **Finish** button\*\*.\*\*



7. Wait for the Data ingestion to be completed, and click **Close**.



8. Select **querysetXX** KQL queryset in the left navigation pane.

The screenshot shows the Microsoft Fabric interface. In the left navigation pane, the 'queryset21' KQL queryset is highlighted with a red box. The main area displays the 'Database: FabricRTA' details, including 'Created by', 'Region', 'Created on', 'Last ingestion', 'Query URI', 'Ingestion URI', and 'OneLake folder'. The 'Data tree' on the left shows the 'FabricRTA' database with a search bar and a list of items: Tables, logsRaw, Shortcuts, Materialized views, Functions, and Data streams.

9. Replace all the code in the **cell** with the following code and click on ► **Run** button

logsRaw

| count

The screenshot shows the Microsoft Fabric interface with the 'queryset89' KQL queryset selected. The code 'logsRaw | count' is entered in the query editor. The 'Run' button is highlighted with a red box. Below the query editor, the results are displayed in a table with the following data:

Count
3,834,012

The logsRaw table should have 3,834,012 records.

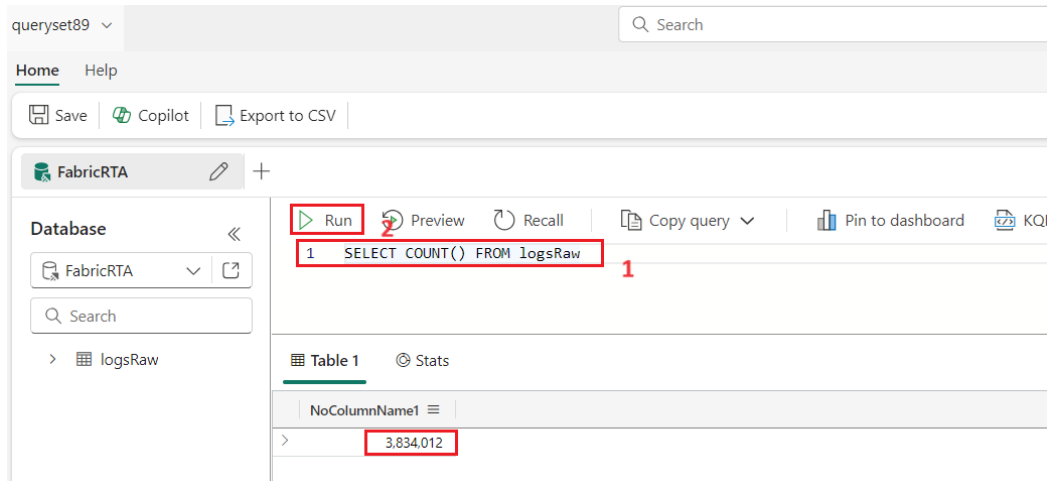
### Exercise 3: Starting with the basics of KQL

A *Kusto query* is a read-only request to process data and return results. The request is stated in plain text that's easy to read, author, and automate. A Kusto query has one or more query statements and returns data in a tabular or graph format.

#### Task 1: Journey from SQL to KQL!

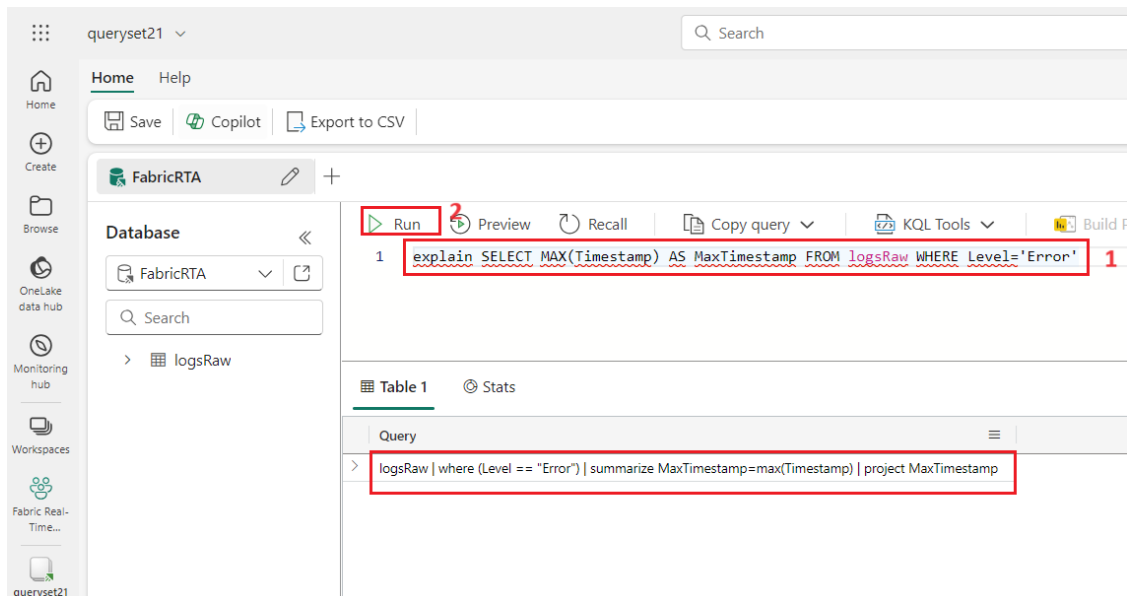
1. For all the SQL pros out there, KQL Database allows a subset of TSQL queries. Try running the following SQL query in web UI. Replace all the code in the **cell** with the following code and click on ► **Run** button.

SELECT COUNT() FROM logsRaw



2. The primary language to interact with Kusto is KQL (Kusto Query Language). To make the transition and learning experience easier, you can use the explain operator to translate SQL queries to KQL.
3. Replace all the code in the **cell** with the following code and click on ► **Run** button.

explain SELECT MAX(Timestamp) AS MaxTimestamp FROM logsRaw WHERE Level='Error'



Output of the above query will be a corresponding KQL query

logsRaw

| where (Level == "Error")

| summarize MaxTimestamp=max(Timestamp)

| project MaxTimestamp

## Task 2: Basic KQL queries - explore the data

In this task, you will see some KQL examples. For this task, we will use the table logsRaw, which has data we loaded in previous challenge from storage account.

1. Execute the queries and view the results. KQL queries can be used to filter data and return specific information. Now, you'll learn how to choose specific rows of data. The **where** operator filters results that satisfy a certain condition.
2. Replace all the code in the **cell** with the following code and click on ► **Run** button.

logsRaw

| where Level=="Error"

| take 10

3. Find out how many records are in the table. Replace all the code in the **cell** with the following code and click on ► **Run** button.

logsRaw

| summarize count()

4. Find out the minimum and maximum Timestamp. Replace all the code in the **cell** with the following code and click on ► **Run** button.

logsRaw

| summarize min(Timestamp), max(Timestamp)

5. Our dataset has trace records written by Contoso's DOWNLOADER program, which downloads files from blob storage as part of its business operations. Replace all the code in the **cell** with the following code and click on ► **Run** button.

logsRaw

| where Component == "DOWNLOADER"

| take 10

6. Select the **Properties** column is dynamic. The dynamic data type is special in that it can take on any value of other data types, as well as arrays and property bags (dictionaries).

The screenshot shows the Microsoft Fabric query editor interface. At the top, there's a search bar and a 'queryset21' dropdown. Below that, the 'Home' and 'Manage' tabs are visible. The main area displays a KQL query in a code editor:

```
1 logsRaw
2 | where Component == "DOWNLOADER"
3 | take 10
4
```

The query results are shown in a table with columns: Timestamp, Source, Node, Level, Component, ClientRequestid, and Properties. The first row is highlighted, showing a download record. The 'Properties' column is expanded, revealing a JSON object with fields like 'compressedSize', 'originalSize', and 'downloadDuration'.

Timestamp	Source	Node	Level	Component	ClientRequestid	Properties
2014-03-08 03:00:03.5650	IMAGINEFIRST0	Engine0000000000769	Inform...	DOWNLOADER	bd6192fc-c33d-569d-ec96-163809897aad	["compressedSize":1009991164,"Ori...
2014-03-08 03:00:00.1070	IMAGINEFIRST0	Engine0000000000086	Inform...	DOWNLOADER	4be1dc96-7bba-5b16-cc9-862398269a3f	["compressedSize":1017160773,"Ori...
2014-03-08 03:00:00.4740	IMAGINEFIRST0	Engine0000000000148	Inform...	DOWNLOADER	f5aa8306-a8f1-ca55-9f64-ef06486c881d	["compressedSize":1895002443,"Ori...

7. The dynamic type is extremely beneficial when it comes to storing JSON data, since KQL makes it simple to access fields in JSON and treat them like an independent column: just use either the dot notation (dict.key) or the bracket notation (dict["key"]).
8. The extend operator adds a new calculated column to the result set, during query time. This allows for the creation of new standalone columns to the result set, from the JSON data in dynamic columns. Replace all the code in the **cell** with the following code and click on ► **Run** button.

logsRaw

| where Component == "DOWNLOADER"

| take 100

| extend originalSize=Properties.OriginalSize, compressedSize=Properties.compressedSize

### Task 3: Explore the table and columns

After subscribing a dynamic object, it is necessary to cast (convert) the value to a simple type in order to utilize them (for example, if you want to summarize the sizes of all the OriginalSize, you should convert the dynamic type to a numeric type, like long).

1. This below query to get the table that is shown in the image below (we want to convert the OriginalSize and CompressedSize columns to long)
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

logsRaw

| extend originalSize=Properties.OriginalSize, compressedSize=Properties.compressedSize

| getschema

The screenshot shows the Microsoft Fabric query editor interface. At the top, there's a search bar and navigation tabs like 'Home' and 'Help'. Below that are buttons for 'Save', 'Copilot', and 'Export to CSV'. The main workspace is divided into a left sidebar and a right pane. The sidebar shows a 'Database' section with 'FabricRTA' selected and a search bar. Below it, a table icon indicates the 'logsRaw' table. The right pane contains a KQL query editor with a red box highlighting the query code: 

```
logsRaw
| extend originalSize=Properties.OriginalSize, compressedSize=Properties.compressedSize
| getschema
```

 Above the query editor are buttons for 'Run', 'Preview', 'Recall', 'Copy query', 'KQL Tools', and 'Build PowerBI report'. The 'Run' button is highlighted with a red box. Below the query editor, the 'getschema' output is displayed as a table with columns: ColumnName, ColumnOrdinal, DataType, and ColumnType. The table lists 10 columns, with the last three being 'originalSize', 'compressedSize', and 'compressedSize' (likely a typo for 'originalSize' in the original image), all with 'dynamic' data types.

ColumnName	ColumnOrdinal	DataType	ColumnType
> Timestamp	0	System.DateTime	datetime
> Source	1	System.String	string
> Node	2	System.String	string
> Level	3	System.String	string
> Component	4	System.String	string
> ClientRequestId	5	System.String	string
> Message	6	System.String	string
> Properties	7	System.Object	dynamic
> originalSize	8	System.Object	dynamic
> compressedSize	9	System.Object	dynamic

## Task 4: Keep the columns of your interest

You are investigating an incident and wish to review only several columns of the dataset.

1. A query to get only specific desired columns: Timestamp, ClientRequestId, Level, Message. Take arbitrary 10 records.
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

logsRaw

| project Timestamp, ClientRequestId, Level, Message

| take 10

The screenshot shows the Microsoft Fabric Data Explorer interface. At the top, there's a search bar and a 'queryset21' label. Below that, a 'Database' sidebar on the left shows 'FabricRTA' and 'logsRaw'. The main area contains a KQL query editor with the following code:

```
1 logsRaw
2 | project Timestamp, ClientRequestId, Level, Message
3 | take 10
4
```

The 'Run' button is highlighted with a red box. Below the query editor, the results are displayed in a table titled 'Table 1'. The table has four columns: Timestamp, ClientRequestId, Level, and Message. It shows 10 records, with the first three highlighted in yellow.

Timestamp	ClientRequestId	Level	Message
2014-03-08 04:00:00.0060	d6691328-64f4-9721-1fa0-5c45f903859	Information	ResponseStreamEncoder Engine000000000677: 677: State='enabled' Reason='Accept-Encoding set to gzip' of 'https://Engine000000000677.ADMIL...
2014-03-08 04:00:00.0210	5337b850-c915-42ff-58ba-5f53c1ee6dd3	Information	AadValidator.ValidateIssuerImpl.Engine000000000123: Start 123 Issuer='https://Engine000000000123.ENGINE.com/'
2014-03-08 04:00:00.0230	b4428f59-54f4-aa5b-b0b0-bfb0970a2e12	Information	OwinResponseStreamCompressor: State='enabled' Reason='Accept-Encoding set to deflate'
2014-03-08 04:00:00.0350	ea5e3c20-be58-f2bb-d3ee-9f0d82cf32e3	Information	AadValidator.ValidateIssuerImpl.Engine000000000603: Start 603 Issuer='https://Engine000000000603.EVENTING.com/'
2014-03-08 04:00:00.0370	1da937f5-c133-195d-3a30-8af934b6ada5	Information	AadValidator.ValidateAudienceImpl.Engine000000000761: 761 Audiences='https://Engine000000000761.THUMBNAILSERVICETRACE.com
2014-03-08 04:00:00.0460	3713404b-d63b-0923-c238-a9aefab982ea	Information	Gateway Engine000000000580 resolved primary 580 for service 'fabric://management.admin.svc' - '1' net.tcp://Engine000000000580.DISPATCHER...
2014-03-08 04:00:00.0490	093c23d4-8f05-76ad-17b7-055da12d269a	Information	OwinResponseStreamCompressor: State='enabled' Reason='Accept-Encoding set to deflate'
2014-03-08 04:00:00.0500	39cdaffa-b40d-5339-577c-ec23ad66d02c	Warning	OwinResponseStreamCompressor: State='enabled' Reason='Accept-Encoding set to deflate'
2014-03-08 04:00:00.0620	d5f227ff-8c2a-500b-29ab-2f6b9fe57d76	Warning	Audience 'https://Engine000000000055.EXPLORESERVICEWATCHDOG.com/' matches the valid Engine000000000055 audience regex '^https://[w]...
2014-03-08 04:00:00.0640	043f2f9f-ac2b-2997-39f2-16ca203010ff	Warning	AadValidator.ValidateAudienceImpl.Engine000000000082: 82 Audiences='https://Engine000000000082.PROCESSING.com

## Task 5: Filter the output

You are investigating an incident that occurred within a specific time frame.

1. Write a query to get only specific desired columns: Timestamp, ClientRequestId, Level, Message. Take all the records between 2014-03-08 01:00 and 2014-03-08 10:00.
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

logsRaw

| where Timestamp between (datetime(2014-03-08 01:00).. datetime(2014-03-08 10:00))

| project Timestamp, ClientRequestId, Level, Message

| take 10

## Task 6: Sorting the results

Your system generated an alert indicating a significant decrease in incoming data. You want to check the traces of the "INGESTOR\_EXECUTER" [sic] component of the program.

1. Write a query that returns 20 sample records in which the Component column equals the word "INGESTOR\_EXECUTER" [sic].
2. Once done, rewrite the query to take the top 1 records by the value of rowCount (for the "INGESTOR\_EXECUTER" [sic] records).
3. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

```
logsRaw
| where Component == 'INGESTOR_EXECUTER'
| extend rowCount=toint(Properties.rowCount)
| where isnotempty(rowCount)
| sort by rowCount
| top 10 by rowCount desc
```

### Task 7: Data profiling

1. As part of the incident investigation, you want to extract format and rowCount from INGESTOR\_EXECUTER [sic] component. Rename the calculated fields to fileFormat and rowCount respectively. Also, Make Sure Timestamp, fileFormat and rowCount are the first 3 columns.
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

```
logsRaw
| where Component == 'INGESTOR_EXECUTER'
| extend rowCount=toint(Properties.rowCount), fileFormat=tostring(Properties.format)
| project Timestamp, fileFormat, rowCount, ClientRequestId, Component, Level, Message
| take 10
```

### Task 8: Total number of records

The system comprises of several "components", but you don't know their names or how many records were generated by each.

1. Write a query to find out how many records were generated by each component. Use the Component column.
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

Copy

```
logsRaw
| summarize count() by Component
```

### Task 9: Aggregations and string operations

You assume that the incident being investigated has a connection to the ingestion process run by Contoso's program.

1. Write a query to find out how many records contain the string 'ingestion' in the Message column. Aggregate the results by **Level**.
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

```
logsRaw
| where Message has "ingestion"
| summarize count() by Level
```

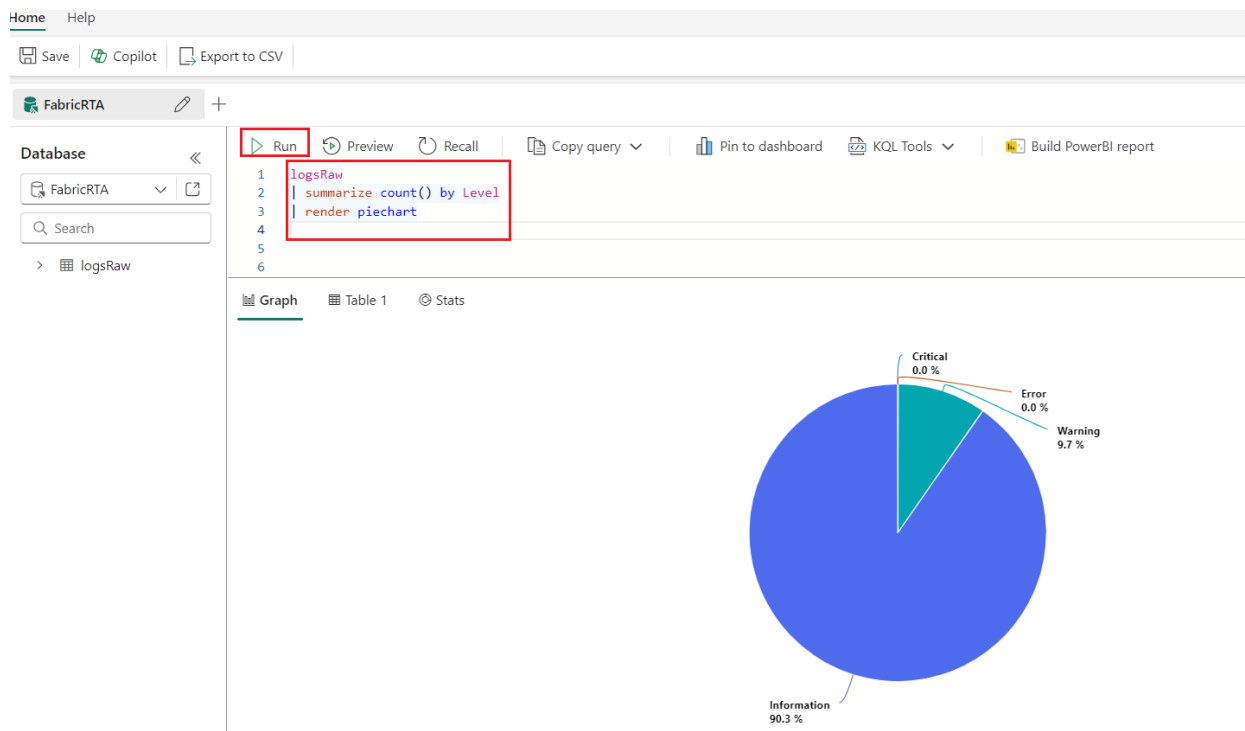
## Task 10: Render a chart

1. Write a query to find out how many total records are present per Level (aggregated by Level) and render a piechart.
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

logsRaw

| summarize count() by Level

| render piechart



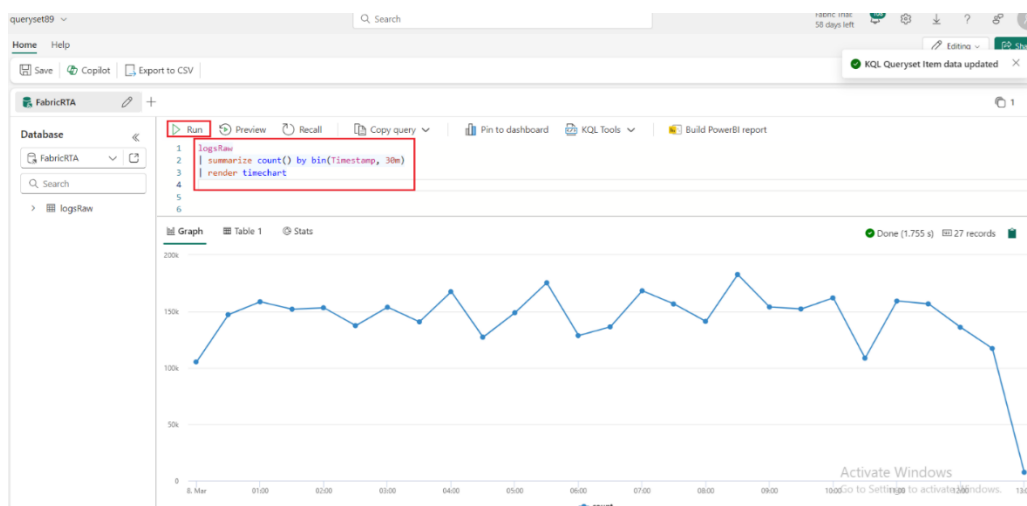
## Task 11: Create bins and visualize time series

1. Write a query to show a timechart of the number of records in 30 minute bins (buckets). Each point on the timechart represent the number of logs in that bucket.
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

logsRaw

| summarize count() by bin(Timestamp, 30m)

| render timechart





## Exercise 4: Explore and Transform Data

In this exercise we will explore 3 capabilities of Data Explorer

- **User-defined functions** are reusable KQL subqueries that can be defined as part of the query itself (ad-hoc functions), or persisted as part of the database metadata (stored functions - reusable KQL query, with the given name). Stored functions are invoked through a name, are provided with zero or more input arguments (which can be scalar or tabular), and produce a single value (which can be scalar or tabular) based on the function body.
- **Update Policy** is like an internal ETL. It can help you manipulate or enrich the data as it gets ingested into the source table (e.g. extracting JSON into separate columns, creating a new calculated column, joining the newly ingested records with a static dimension table that is already in your database, etc). For these cases, using an update policy is a very common and powerful practice.

Each time records get ingested into the source table, the update policy's query (which we'll define in the update policy) will run on them (**only on newly ingested records** - other existing records in the source table aren't visible to the update policy when it runs), and the results of the query will be appended to the target table. This function's output schema and target table schema should exactly match.

### Task 1: User defined Function (Stored Functions)

1. Create a stored functions, named ManipulatelogsRaw, that will contain the code below. Make sure the function works.
2. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

```
logsRaw
| where Component in (
'INGESTOR_EXECUTER',
'INGESTOR_GATEWAY',
'INTEGRATIONDATABASE',
'INTEGRATIONSERVICEFLOWS',
'INTEGRATIONSERVICETRACE')
```

### Task 2: Create an update policy

In this task, we will use an update policy to filter the raw data in the logsRaw table (the source table) for ingestion logs, that will be ingested into the new table ingestionLogs that we'll create.

1. Build the target table, replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

```
.create table ingestionLogs (
Timestamp: datetime,
Source: string,
Node: string,
Level: string,
Component: string,
ClientRequestId: string,
Message: string,
Properties: dynamic)
```

2. Create a function for the update policy, replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

```
.create function ingestionComponents(){  
  
logsRaw  
  
| where Component has_any ('INGESTOR_EXECUTER', 'INGESTOR_GATEWAY',  
'INTEGRATIONDATABASE','INTEGRATIONSERVICEFLOWS', 'INTEGRATIONSERVICETRACE', 'DOWNLOADER')  
  
}
```

3. Create the update policy(Fill in the blanks), replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.

```
.alter table ingestionLogs  
  
policy update @[{"IsEnabled": true,"Source": "logsRaw", "Query": "ingestionComponents()",  
"IsTransactional": true, "PropagateIngestionProperties": false}]'
```

4. Update policy can transform and move the data from source table from the time it is created. It cannot look back at already existing data in source table. We will ingest new data into logsraw table and see new data flowing into ingestionLogs table
5. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output.
6. copy **operationIds** and paste them in a notepad , and then Save the notepad to use the information in the upcoming step.

// Note, the following .ingest commands set creationTime to 2014 as you may notice in the file path.

// This param allows to backfill the table with historical data and index it according ot the creationTime setting.

```
.execute database script <|  
  
.ingest async into table logsRaw  
(h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-  
onegb/2014/03/08/00/data.csv.gz') with (format='csv', creationTime='2024-03-08T00:00:00Z');  
  
.ingest async into table logsRaw  
(h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-  
onegb/2014/03/08/01/data.csv.gz') with (format='csv', creationTime='2024-03-08T01:00:00Z');  
  
.ingest async into table logsRaw  
(h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-  
onegb/2014/03/08/02/data.csv.gz') with (format='csv', creationTime='2024-03-08T02:00:00Z');  
  
.ingest async into table logsRaw  
(h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-  
onegb/2014/03/08/03/data.csv.gz') with (format='csv', creationTime='2024-03-08T03:00:00Z');  
  
.ingest async into table logsRaw  
(h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-  
onegb/2014/03/08/04/data.csv.gz') with (format='csv', creationTime='2024-03-08T04:00:00Z');
```

Run

Preview

Recall

Copy query

Pin to dashboard

KQL Tools

Export to CSV

Power BI

Set alert

```

1 // Note, the following .ingest commands set creationTime to 2014 as you may notice in the file path.
2 // This param allows to backfill the table with historical data and index it according to the creationTime settings
3
4 .execute database script <|
5 .ingest async into table logsRaw (h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-
6 .ingest async into table logsRaw (h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-
7 .ingest async into table logsRaw (h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-
8 .ingest async into table logsRaw (h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-
9 .ingest async into table logsRaw (h'https://adxsamplefiles.blob.core.windows.net/publiccsvsamples/logsbenchmark-
10

```

Table 1

Search Done (4.299 s) 5 records

OperationId	CommandType	CommandText	Result	Reason
> b8de507f-85f3-494c-9d0...	DataIngestPull	.ingest async into table logsRaw (*****) ...	Completed	
> 10a18d64-d515-496f-a4...	DataIngestPull	.ingest async into table logsRaw (*****) ...	Completed	
> 76191c9d-8743-4a57-b6...	DataIngestPull	.ingest async into table logsRaw (*****) ...	Completed	
> 6031b4e5-65a5-478d-96...	DataIngestPull	.ingest async into table logsRaw (*****) ...	Completed	
> d5ed7302-9b4e-440c-8a...	DataIngestPull	.ingest async into table logsRaw (*****) ...	Completed	

**Note:** The above command does not complete immediately. Because we used the async parameter, the output of the above query will be **operationIds**. The progress of the query can be checked by used the below command

- Check progress of the commands, replace all the code in the **cell** with the following code and click on **Run** cell button and review the output.

.show operations

| where OperationId == 'OperationId '

**Note:** Replace the operationIds which you have saved in Step 6

Run

Preview

Recall

Copy query

Pin to dashboard

KQL Tools

Export to CSV

Power BI

Set alert

```

1
2 .show operations
3 | where OperationId == 'b8de507f-85f3-494c-9d0f-8283d14110cb'

```

Table 1

Search Done (0.155 s) 2 records

OperationId	Operation	NodeId	StartedOn	LastUpdatedOn	Duration
> b8de507f-85f3-494c-9d0f-8283d14110cb	DataIngestPull		2024-08-22 05:53:17.6850	2024-08-22 05:53:17.6850	00:00:00.0002789
> b8de507f-85f3-494c-9d0f-8283d14110cb	DataIngestPull		2024-08-22 05:53:17.6850	2024-08-22 05:53:17.7580	00:00:00.0727414

**Note:** If the count is not matching for ingestionLogs table, it means that one of the above .ingest commands have throttled or failed.

## Exercise 5: Eventstreams

### Task 1: Create Eventstream

1. Select **your workspace** in the left navigation pane.
2. Select the **New** drop-down, and from there select **Eventstream**. Name the Eventstream **RTA\_EventStream** and click on the **Create** button.

### New Eventstream

Name \*

RTA\_Eventstream

☐ Enhanced Capabilities (preview)



Enhancements, currently in preview, enable you to reuse the events, route events based on the content, etc. [Learn more](#)

Create

Cancel

3. On the Eventstream, select **New source** and select **Custom App**.
4. On the **Custom App** configuration page, enter the source name as **RTAcustom** and click on **Add** button and publish the app.

### Custom App

Source name \*

RTAcustom

5. On the **Eventstream** pane, select the **keys** under the Details ,copy the **connection strings-primarykey** and paste them on a notepad, as you need them in the upcoming task

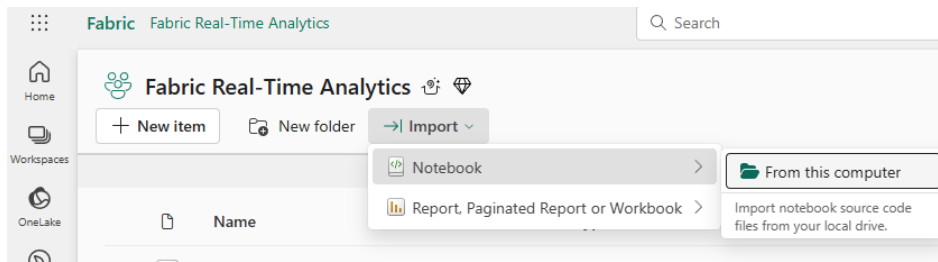
The screenshot shows the Azure portal interface for an Eventstream. On the left, the 'Sources' section is expanded, showing 'RTAcustom'. The main area displays a diagram of the 'RTAcustom' source connected to the 'RTA\_Eventstream' destination. Below the diagram, the 'Details' tab is selected, showing the 'Keys' section. The 'Keys' section contains the following information:

Property	Value
Event hub name	es_64488f47-3202-4b12-8ea5-fff50018cbd7
Shared access key name	key_32a232dc-5c13-c3c2-42e7-a2d8f0ac4e08
Primary key	.....
Secondary key	.....
Connection string-primary key	Endpoint=sb://esehusw38o2fy1dsmj9s2xr8.servicebus.....
Connection string-secondary key	.....

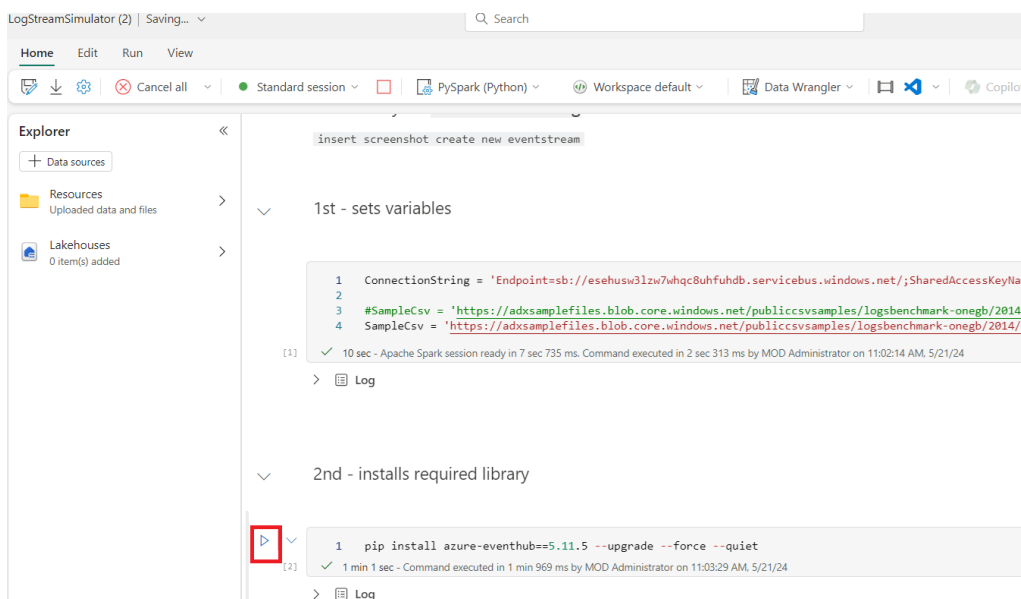
The primary key and the connection string-primary key are highlighted with red boxes in the original image.

## Task 2: Import LogStreamSimulator notebook

1. Download **LogStreamSimulator** notebooks.
2. Now, select **your workspace** in the left navigation pane
3. Select **Import notebook**.



4. Select **Upload** from the **Import status** pane that appears on the right side of the screen.
5. You will see a notification stating **Imported successfully**.
6. *Open*, the **LogStreamSimulator** notebook.
7. To start the notebook, in the 1st cell paste the **connection string of your custom app source**(the value that you have saved in your notepad in the **Exercise 5>Task 1>Step 5**), select the **Run** icon that appears on the left side of the cell.
8. To install required library, select and run the 2nd , 3rd cells.



9. To send the data, select and run the 4th ,5th cells.

**Note:** If you get `AnalysisException: [PATH_NOT_FOUND] Path does not exist`, replace code cell 1 with **backup URL location** <https://github.com/microsoft/FabricRTA-in-a-Day/blob/main/assets/blobURLbackup.md#lab-3> for SampleCsv,

**Important Note:** If either of the last code cells in step 4 fail with error file does not exist, then refresh the browser page and click stop session to get a new session (not reconnect to existing session), then click Run All again.

Errors with pip's dependency resolver are okay, proceeding code cells should run successfully.

⚠ If you get `ImportError: cannot import name 'Buffer' from 'typing_extensions'`, replace code cell 2 with: `pip install azure-eventhub==5.11.5 --upgrade --force` and click Run All again.

10. Select **RTA-Eventstream** in the left navigation pane.
11. Open eventstream artifact, preview the data stream

The screenshot shows the Real-Time Intelligence dashboard. In the top navigation bar, there's a search bar and a 'Home' button. Below the navigation bar, there's a 'Data' section with a list of artifacts. The 'RTA-EventStream' artifact is highlighted with a red box. Below the artifact list, there's a 'Data preview' tab, also highlighted with a red box. The 'Data preview' tab shows a table of data with columns: 'Type', 'Source', 'Node', 'Level', 'Component', and 'ClientRe'. The table contains five rows of data, all with 'Information' level and 'IMAGINEFIRSTO' source. The 'Component' column shows 'INGESTOR\_GATEWAY', 'DOWNLOADER', 'COMMON3', 'GATEWAY', and 'METRICCOLLECTORSERVICE'. The 'ClientRe' column shows 'ff7855i' for all rows. The 'Last refreshed' time is '07/25/24 04:14:21 AM' and the 'Last event time' is '07/25/24 04:14:14 AM'.

### Task 3: Send data from the Eventstream to the KQL database

1. Our data should be arriving into our Eventstream, and we'll now configure the data to be ingested into the KQL database we created in the above task. On the Eventstream, click on **New destination**, then navigate and click on **Eventhouse**.
2. On the KQL settings, select **Direct ingestion**. While we have the opportunity to process event data at this stage, for our purposes, we will ingest the data directly into the KQL database. Set the destination name to **RTA-Destination**, then select your **workspace**, **Eventhouse**, and KQL database created in the above task, then click on **Save** button and **Publish**.

The screenshot shows the 'Eventhouse' configuration form. The 'Data ingestion mode' section has two radio buttons: 'Direct ingestion' (selected and highlighted with a red box) and 'Event processing before ingestion'. Below this, there's a red-bordered box containing four input fields: 'Destination name' (with placeholder text 'Enter a destination name'), 'Workspace' (with placeholder text 'Type to select a workspace'), 'Eventhouse' (with placeholder text 'Type to select a eventhouse'), and 'KQL Database' (with placeholder text 'Type to select a KQL database'). Below the red-bordered box, there's a warning message: 'Activation for KQL database - direct ingestion mode requires further actions after publish.' At the bottom, there's a 'Save' button highlighted with a red box.

- Once publish, select configure on destination. On the first settings page, select **logsRaw** table and click on the **Next** button.

Get data  
Pick a destination table and configure the source

Eventstream → FabricRTA/logsRaw

Select or create a destination table

Search

FabricRTA

+ New table

logsRaw

Configure the data source  
Create a data connection to ingest data from Eventstream.

Eventstream Name: RTA\_Eventstream

Data connection name: RTA\_Eventstream\_FabricRTA

> Advanced filters

Cancel Back **Next** Go to Settings

- The next page allows us to inspect and configure the schema. Be sure to change the format from TXT to **JSON**, if necessary. The default columns of *symbol*, *price*, and *timestamp* should be formatted as shown in the below image; then click on the **Finish** button.

Get data  
Inspect the data

Eventstream → FabricRTA/logsRaw

Preview data Command viewer Format: JSON Edit columns Advanced

Data sample found. Fetch more data Discard and fetch new data

50 events found, 50 events match the selected settings. Open for more details.

Timestamp (datetime)	Source (string)	Node (string)	Level (string)	Component (string)	ClientRequestId (string)	Message (string)
2014-03-08 00:00:45.7090	IMAGINEFIRST0	Engine0000000000940	Information	EXECUTIONLOG		AadValidator.ValidateAudienceId
2014-03-08 00:00:45.7120	IMAGINEFIRST0	Engine0000000000025	Warning	COMPANIONPROVIDER		Gateway Engine000000000025 r
2014-03-08 00:00:45.7360	IMAGINEFIRST0	Engine00000000000274	Information	COMPANIONPROVIDER		ResponseStreamEncoder Engine
2014-03-08 00:00:45.7390	IMAGINEFIRST0	Engine00000000000139	Information	MODELOPERATOR		ResponseStreamEncoder Engine
2014-03-08 00:00:45.7600	IMAGINEFIRST0	Engine00000000000525	Information	CONTENTPROVIDER		AadValidator.ValidateIssuerImpl
2014-03-08 00:00:45.7630	IMAGINEFIRST0	Engine00000000000234	Information	DATAEXTENSION		AadValidator.ValidateAudienceId
2014-03-08 00:00:45.7870	IMAGINEFIRST0	Engine00000000000896	Information	NODEAGENT		AadValidator.ValidateAudienceId
2014-03-08 00:00:45.7900	IMAGINEFIRST0	Engine00000000000253	Information	CACHE		GetDataPullJobsOperation.Oper
2014-03-08 00:00:45.8050	IMAGINEFIRST0	Engine00000000000089	Warning	BROKERTRACE		ResponseStreamEncoder Engine

Cancel Back **Finish** Go to Settings

- On the **Summary** page, if there are no errors, you'll see a **green checkmark** as shown in the below image, then click on the **Close** button to complete the configuration.

Source

Configure

Inspect

Summary

Get data

Summary

Eventstream → FabricRTA/logsRaw

Data Preparation

Use existing table (*logsRaw*)

Create mapping (*logsRaw\_mapping*)

Create data connection

Details

-

-

-

What can you do with the data?

Now that you've ingested data, you can explore, run queries, and visualize the data using the dashboard

Explore the results

Search, sort, filter, and expand rows to highlight trends or problems.

Explore

Undo ingestion

Delete the data you've just ingested.

Delete ingested data

Create new dashboard

View your ingested data's key metrics, sample records, and ingestion history

Create dashboard

RTA\_Eventstream

Search

Fabric Trial: 55 days left

Home

Settings

New source

New destination

Refresh

Activate all

Deactivate all

Data

Sources

RTACustom

Destinations

RTA\_destination

RTACustom

RTA\_Eventstream

RTA\_destination

Details

Data preview

Name

RTA\_destination

Type

KQL Database

Status

Active

Related item

Kusto item: FabricRTA

Destination Saved  
"RTA\_destination, KQL Database" is successfully saved.  
Copy

6. Now, select **your workspace** in the left navigation pane and open **KQL Queryset**

7. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output

logsRaw

| take 10



queryset21

Home Help

Save Copilot

FabricRTA

Database

FabricRTA

Search

Functions

ingestionLogs

logsRaw

Run Preview Recall Copy query Pin to dashboard KQL Tools Export to CSV

1 logsRaw  
2 | take 10  
3

Table 1 Stats

Search Done (0.516 s) 10 records

Timestamp	Source	Node	Level	Component	ClientRe
> 2014-03-08 03:00:00.0050	IMAGINEFIRST0	Engine00000000000236	Information	DATASHAPEQUERYTRANSLATION	2fb064f
> 2014-03-08 03:00:00.0160	IMAGINEFIRST0	Engine00000000000727	Information	FABRICINTEGRATOR	a8f57ec
> 2014-03-08 03:00:00.0210	IMAGINEFIRST0	Engine00000000000344	Information	ECSCIENTTRACE	04a1b7
> 2014-03-08 03:00:00.0520	IMAGINEFIRST0	Engine00000000000407	Information	RUNNER3	f5aa83c
> 2014-03-08 03:00:00.0800	IMAGINEFIRST0	Engine00000000000274	Information	CLOUDCACHE	cc58c7e
> 2014-03-08 03:00:00.1070	IMAGINEFIRST0	Engine00000000000086	Information	DOWNLOADER	4be1dc
> 2014-03-08 03:00:00.1120	IMAGINEFIRST0	Engine00000000000424	Information	ADMINSERVICE1	f5aa83c

8. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output

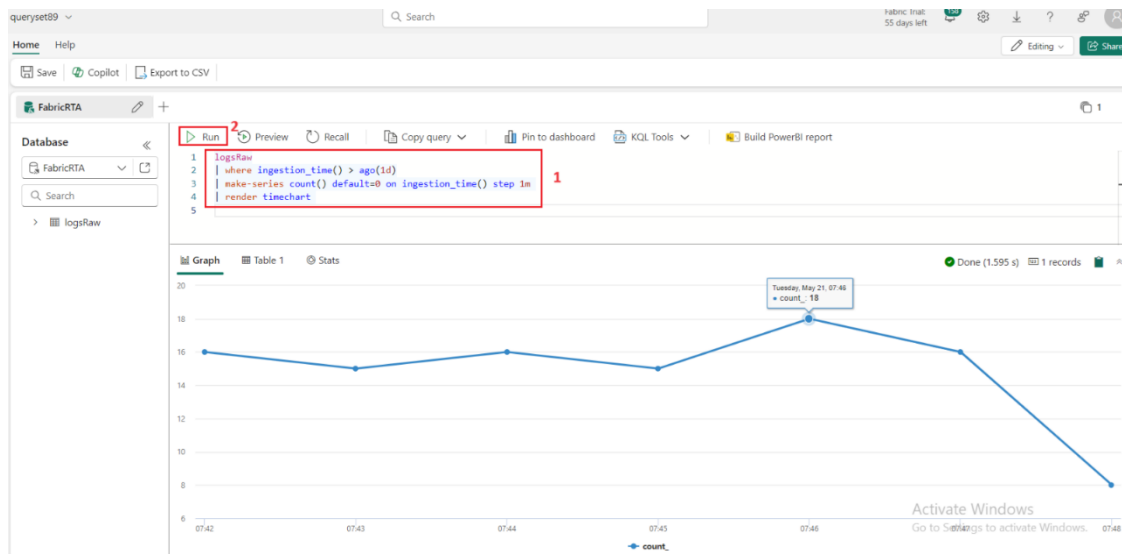
```
//rate
```

```
logsRaw
```

```
| where ingestion_time() > ago(1d)
```

```
| make-series count() default=0 on ingestion_time() step 1m
```

```
| render timechart
```



9. Replace all the code in the **cell** with the following code and click on ► **Run cell** button and review the output

```
//lag
```

```
logsRaw
```

```
| where ingestion_time() > ago(1m)
```

```
| summarize m=max(ingestion_time())
```

```
| project lag=now()-m
```

```
1 //lag
2 logsRaw
3 | where ingestion_time() > ago(1m)
4 | summarize m=max(ingestion_time())
5 | project lag=now()-m
6
```

&gt; 00:00:23.5553565