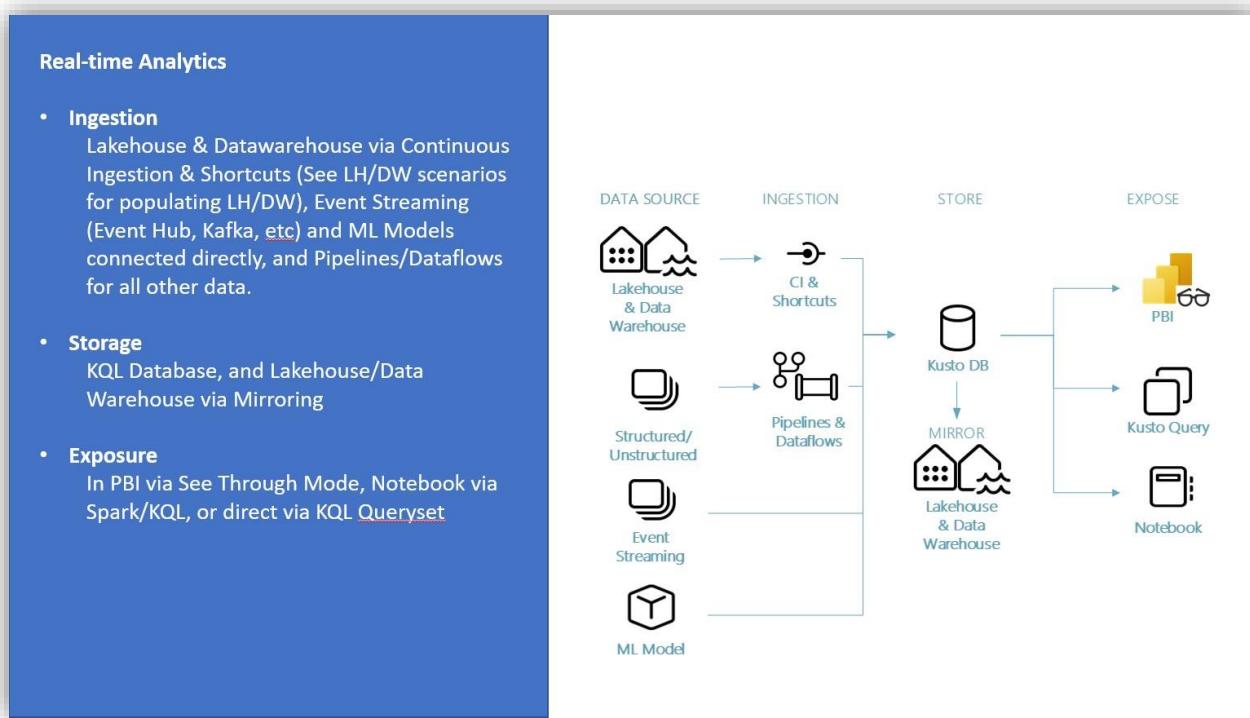


# End-to-End Scenario Tutorial

## Real-time Analytics

Published: July 2023

Updated : Feb 2024



## Contents

Introduction.....	3
Real-time Analytics.....	3
Scenario .....	4
Prerequisites .....	5
Module 1: Build your first Real-time Analytics solution in Fabric.....	6
Create a KQL Database.....	6
Create an Eventstream.....	8
Stream data from Eventstream to KQL Database.....	9
Explore data and build Power BI report.....	17
Module 2: Extending the real-time analytics solution.....	20
Get dimension data from Azure Storage .....	20
Query data .....	23
Explore data further in the KQL Queryset .....	24
Build Power BI report .....	27
Create OneLake shortcut.....	30

# Introduction

## What is Fabric?

Fabric provides a one-stop shop for all the analytical needs for every enterprise. It covers the complete spectrum of services including data movement, data lake, data engineering, data integration and data science, real time analytics, and business intelligence. With Fabric, there is no need to stitch together different services from multiple vendors. Instead, the customer enjoys an end-to-end, highly integrated, single comprehensive product that is easy to understand, onboard, create and operate. There is no other product on the market that offers the breadth, depth, and level of integration that Fabric offers. Additionally, Microsoft Purview is included by default in every tenant to meet compliance and governance needs.

To get an overview over the components and concepts of Fabric read [Fabric - Overview and Concepts](#).

## The purpose of this tutorial

While many concepts in Fabric may be familiar to data and analytics professionals it can be challenging to apply those concepts in a new environment. This tutorial has been designed to walk step-by-step through an end-to-end scenario from data acquisition to data consumption to build a basic understanding of the Fabric UX, the various workloads and their integration points, and the Fabric professional and citizen developer experiences.

The tutorials are not intended to be a reference architecture, an exhaustive list of features and functionality, or a recommendation of specific best practices.

## Real-time Analytics

Real-time Analytics is a portfolio of capabilities that provides an end-to-end analytics streaming solution across Fabric experiences. It supplies high velocity, low latency data analysis, and is optimized for time-series data, including automatic partitioning and indexing of any data format and structure, such as structured data, semi-structured (JSON), and free text.

Real-time Analytics delivers high performance when it comes to your increasing volume of data. It accommodates datasets as small as a few gigabytes or as large as several petabytes and allows you to explore data from different sources and a variety of data formats.

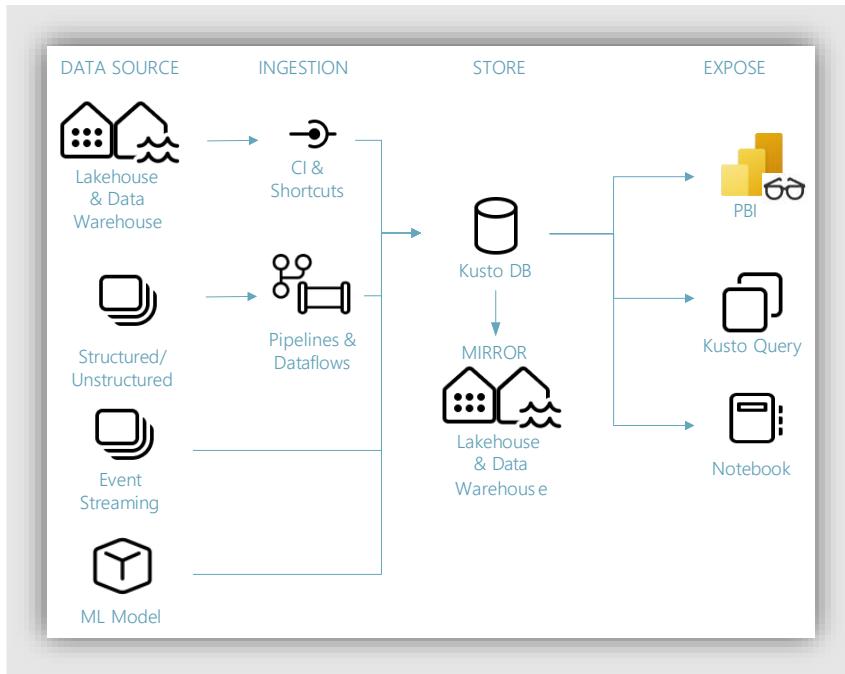


Figure 1: End-to-end Scenario - Real-time Analytics

Real-time Analytics includes integration with other Fabric experiences such as Lakehouse, Data Warehouse, Pipelines, Dataflows and Event Streaming on data sources and ingestion side. On the data exposition and consumption, Real-time analytics integrates with Power BI, and Notebooks.

You can use Real-time Analytics for a range of solutions, such as IoT analytics and log analytics, and in a number of scenarios including manufacturing operations, oil and gas, automotive, and more.

In this tutorial, you learn how to:

- Create a KQL Database
- Create Eventstream
- Stream data from Eventstream to KQL Database
- Check your data with sample queries
- Save queries as a KQL Queryset
- Create a Power BI report
- Create a OneLake shortcut

## Scenario

This tutorial is based on a *sample on New York Yellow Taxi trip data*. The dataset contains trip records of New York's yellow taxis. The yellow taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. You'll use the streaming and query capabilities of Real-time Analytics to answer key questions about the trip statistics, taxi demand in the boroughs of New York and related insights.

## Prerequisites

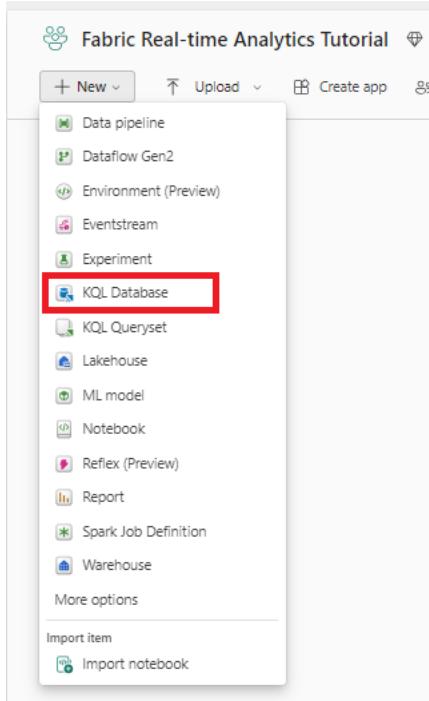
- Power BI Premium subscription. For more information, see [How to purchase Power BI Premium](#).
- Workspace

## Module 1: Build your first Real-time Analytics solution in Fabric

The intent of this module is to quickly build end to end journey of building a real-time analytics solution, ingesting streaming data from Eventstream and then using the KQL Database for creating a real-time refreshing Power BI report.

### Create a KQL Database

1. Go to your Fabric workspace.
2. In the upper left corner of the Fabric Workspace home page, select **New > KQL Database**

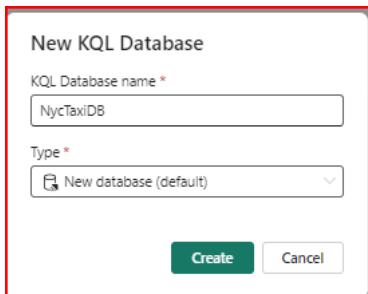


3. Or In the **Real-time Analytics** section, select **KQL Database**.

A screenshot of the Real-time Analytics section. The title 'Real-time Analytics' is at the top, followed by the sub-instruction 'Find insights, track progress, and make decisions faster.' Below this are three cards: 'KQL Database' (selected and highlighted with a red box), 'KQL Queryset', and 'Eventstream'.

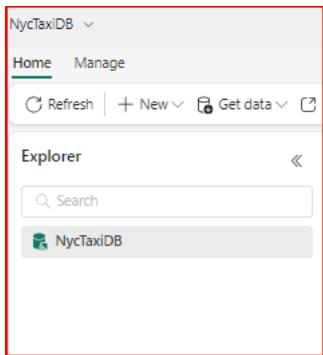
- KQL Database**: Rapidly load structured, unstructured, and streaming data for querying.
- KQL Queryset**: Run queries on your data to produce shareable tables and visuals.
- Eventstream**: Capture, transform, and route real-time event stream to various destinations in desired format with no-code experience.

4. On the **New KQL Database** dialog, enter a *unique name*.
5. Select **Create**.



6. When provisioning is complete the KQL database editor landing page will be shown.

7. Select the **Database** in the Object tree.



8. Select **Explore your data**



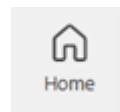
9. Enable availability in Onelake (replace **NycTaxiDB** with your database's name).  
In the **Database details** card, select the **pencil** icon

The screenshot shows the Azure Data Explorer interface. On the left, there's a 'Data tree' sidebar with a 'Database' tab selected. Under 'Database', there's a search bar and a list of databases: 'NycTaxiDB' (selected), 'Tables', 'Shortcuts', 'Materialized views', 'Functions', and 'Data streams'. On the right, the main pane displays 'Database: NycTaxiDB' with 'Database details'. The details include: Created by: Cris Barros, Region: EastUS2, Created on: 4/27/23, 13:45, Last ingestion: Today, 1m ago. To the right of these details are three buttons: 'Query URI', 'Ingestion URI', and 'OneLake folder'. Below these is a 'Copy path' button, which is also highlighted with a red box. The status 'Inactive' is shown next to the copy path button.

Toggle the button to **Active** and select **Done**

## Create an Eventstream

1. Return to the Real-Time Analytics home page. The **Home** icon directs you to the home page of the experience you're currently using.



2. In the **Real-time Analytics** section, select **Eventstream**

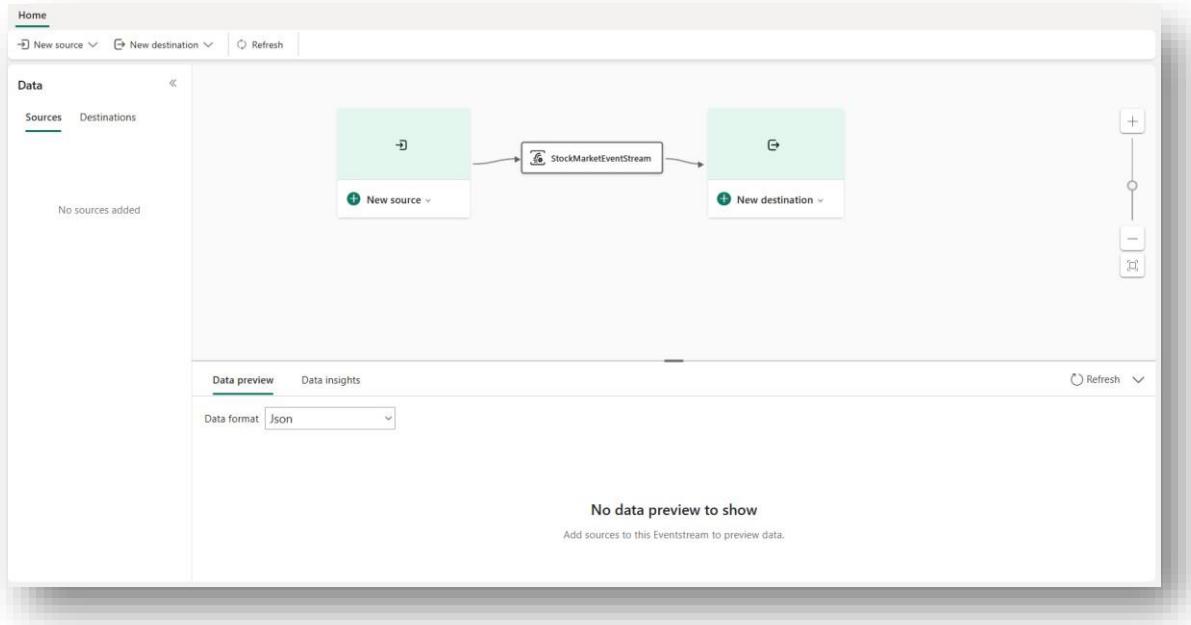
The screenshot shows a 'New' dialog in the Real-time Analytics section. It has a 'Current workspace: Fabric Real-time Analytics Tutorial' header. Below it are four options: 'KQL Database', 'KQL Queryset', 'Eventstream' (which is highlighted with a red box), and 'Use a sample'.

3. On the New Eventstream dialog, enter NyTaxiTripsEventstream as the name.

The screenshot shows the 'New Eventstream' dialog. It has a 'Name \*' field containing 'NyTaxiTripsEventstream'. At the bottom are two buttons: 'Create' (highlighted with a red box) and 'Cancel'.

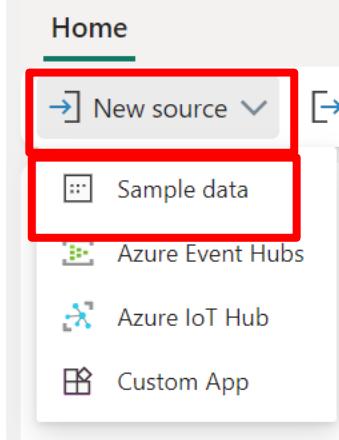
4. Select **Create**.

- When provisioning is complete, Eventstream landing page will be shown.



## Stream data from Eventstream to KQL Database

- In the Eventstream authoring area, select **New source** and choose **Sample data**.



- Enter **nytaxitripsdatasource** as the Source Name on the right window, choose **Yellow Taxi(high data-rate)** and click on **Add**.

## Sample data

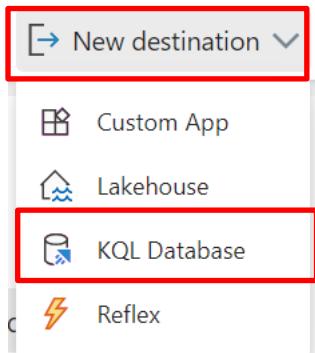
Source name \*

nytaxitripsdatasource

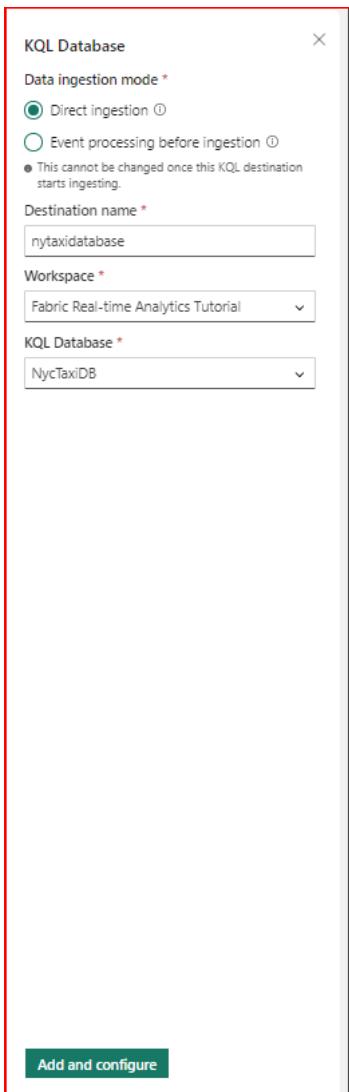
Sample data \*

Yellow Taxi (high data-rate)

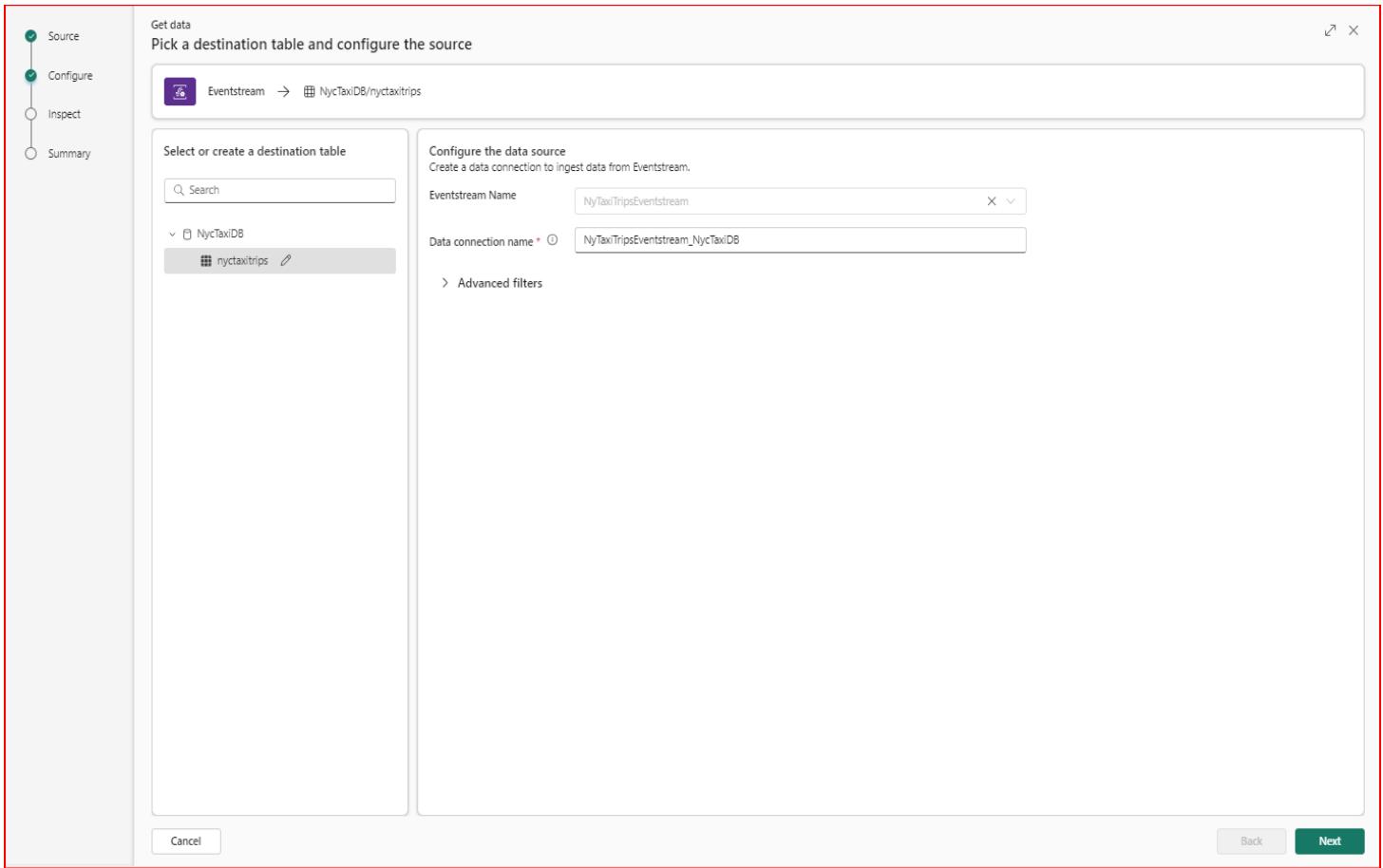
3. In the Eventstream authoring area, select **New destination** and choose **KQL Database**.



4. Select Data ingestion Mode as **Direct Ingestion**, Enter **nytaxidatabase** as the destination name, choose your Fabric workspace from the Workspace dropdown and then choose your KQL Database that you created above and Click on **Add and Configure**



5. In the Get Data Screen , select **New table** and enter **nytaxitrips** as the table name.



6. In the **Inspect** tab, choose **JSON** as the Data format dropdown.

The screenshot shows the 'Inspect' tab interface. On the left, there are four tabs: Source, Configure, Inspect (which is selected and highlighted in green), and Summary. The main area is titled 'Get data' and 'Inspect the data'. It shows an 'Eventstream' source connected to 'NyCTaxiDB/nyctaxitrips'. Below this, there are two tabs: 'Preview data' (selected) and 'Command viewer'. The 'Preview data' tab displays a table of data with 23 rows. The columns are labeled: VendorID (long), tpep\_pickup\_datetime (datetime), tpep\_dropoff\_datetime (datetime), passenger\_count (real), trip\_distance (real), RatecodeID (real), store\_and\_fwd\_flag (string), PULocationID (long), DOLocationID (long), and payment\_type (long). The data shows various taxi trips with their details like pickup and dropoff times, distance, and fare type. To the right of the table, there is a 'Format' dropdown set to 'JSON' with other options like CSV, TSV, etc. Below the table, there are buttons for 'more data', 'Discard and fetch new data', and a dropdown menu.

7. After choosing JSON as the Data format, data preview will refresh and show the data in strongly typed columns.

This screenshot shows the same 'Inspect' tab interface as the previous one, but the data preview has been refreshed. The columns are now explicitly typed as strongly typed columns, indicated by the '(long)', '(datetime)', '(real)', and '(string)' suffixes. The table content remains the same, showing taxi trip data. The 'Format' dropdown is still set to 'JSON'. At the bottom right of the preview area, there are 'Back' and 'Finish' buttons.

8. In this step, we will change data types of multiple columns.

- Select **Edit Columns** on the top of screen For the column **VendorID** column name, **Change data type**, and then choose **int**.

- b. choose datatype as **long** for the columns passenger\_count, PULocationID, DOLocationID and payment\_type.
- c. choose datatype as **real** for the following columns: extra, mta\_tax, tolls\_amount, improvement\_surcharge, congestion\_charge, airport\_fee, trip\_distance, fare\_amount, tip\_amount, total\_amount

After changing the datatypes of the above columns , Click on **Apply**

The screenshot shows the 'Edit columns' interface with a red border around the main content area. On the left, there's a sidebar with four tabs: 'Source' (selected), 'Configure', 'Inspect', and 'Summary'. The main area has a header 'Edit columns' with a back arrow and a close button. Below the header is a table with the following columns:

Column name	Type	Source	Mapping transformation	Sample data
VendorID	int	VendorID		2
tpep_pickup_datetime	datetime	tpep_pickup_datetime		2022-06-01T00:59:15Z
tpep_dropoff_datetime	datetime	tpep_dropoff_datetime		2022-06-01T01:06:50Z
passenger_count	long	passenger_count		1
trip_distance	real	trip_distance		1.93
RatecodeID	real	RatecodeID		1
store_and_fwd_flag	string	store_and_fwd_flag		N
PULocationID	long	PULocationID		90
DOLocationID	long	DOLocationID		231
payment_type	long	payment_type		1
fare_amount	real	fare_amount		8
extra	real	extra		0.5
mta_tax	real	mta_tax		0.5
tip_amount	real	tip_amount		2.95
tolls_amount	real	tolls_amount		6.55
improvement_surcharge	real	improvement_surcharge		0.3
total_amount	real	total_amount		14.75
congestion_surcharge	real	congestion_surcharge		2.5

At the bottom right of the interface are 'Back' and 'Apply' buttons.

9. Click on Finish

Get data

Source Configure Inspect Summary

### Inspect the data

Eventstream → NycTaxiDB/nyctaxitrips

Preview data Command viewer Format: JSON ▾ Edit columns Advanced ▾

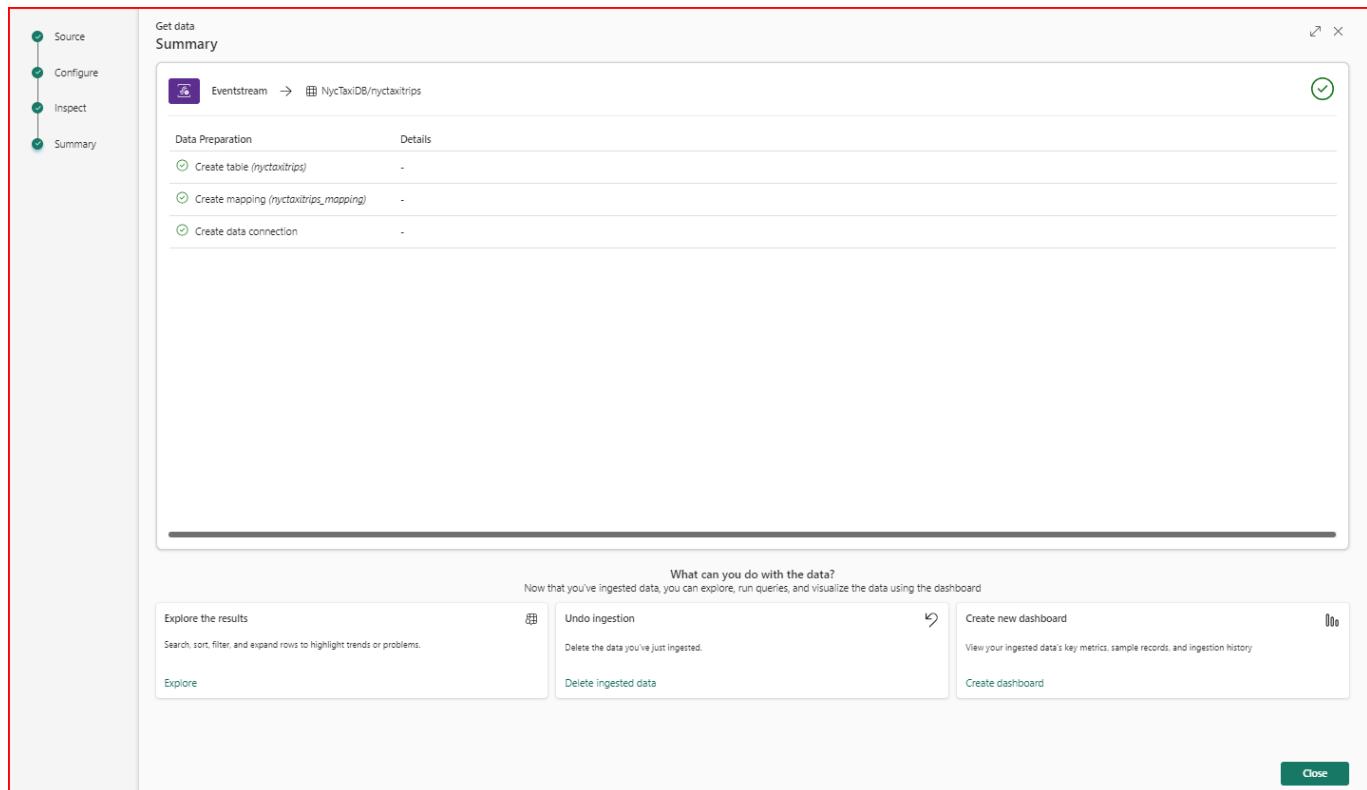
Data sample found. Fetch more data Discard and fetch new data

23 events found. 23 events match the selected settings. Open for more details.

VendorID (int)	tpep_pickup_datetime (datetime)	tpep_dropoff_datetime (datetime)	passenger_count (long)	trip_distance (real)	RatecodeID (real)	store_and_fwd_flag (string)	PULocationID (long)	DOLocationID (long)	payment_type (long)
2	2022-06-01 00:20:25.0000	2022-06-01 00:26:29.0000	2.04	1	N	162	237	1	
2	2022-06-01 00:20:25.0000	2022-06-01 00:44:11.0000	6.67	1	N	234	89	1	
2	2022-06-01 00:22:01.0000	2022-06-01 00:28:18.0000	2.36	1	N	142	166	1	
2	2022-06-01 00:22:37.0000	2022-06-01 00:32:55.0000	3.09	1	N	148	209	1	
1	2022-06-01 00:23:02.0000	2022-06-01 00:48:28.0000	7	1	N	249	61	1	
2	2022-06-01 00:26:20.0000	2022-06-01 00:35:52.0000	2.57	1	N	144	66	1	
2	2022-06-01 00:27:09.0000	2022-06-01 00:31:58.0000	1.41	1	N	137	79	1	
2	2022-06-01 00:28:18.0000	2022-06-01 00:31:25.0000	3.83	1	N	113	163	1	
1	2022-06-01 00:28:24.0000	2022-06-01 00:30:45.0000	0.4	1	N	230	230	1	
2	2022-06-01 00:31:16.0000	2022-06-01 00:40:22.0000	1.83	1	N	151	152	1	
2	2022-06-01 00:33:50.0000	2022-06-01 00:48:51.0000	4.78	1	Y	146	79	1	
1	2022-06-01 00:38:30.0000	2022-06-01 00:42:06.0000	1.7	1	N	239	230	1	
2	2022-06-01 00:36:18.0000	2022-06-01 00:40:27.0000	0.52	1	N	263	262	1	
2	2022-06-01 00:38:54.0000	2022-06-01 01:03:38.0000	4.59	1	N	231	230	1	
2	2022-06-01 00:39:51.0000	2022-06-01 01:02:57.0000	16.54	2	N	132	233	1	
1	2022-06-01 00:46:31.0000	2022-06-01 01:08:27.0000	6.5	1	N	234	7	2	
1	2022-06-01 00:46:49.0000	2022-06-01 01:03:23.0000	4.2	1	N	230	7	2	
2	2022-06-01 00:50:10.0000	2022-06-01 01:07:11.0000	6.35	1	N	79	74	1	
2	2022-06-01 00:56:37.0000	2022-06-01 01:05:01.0000	2.31	1	N	163	262	1	
2	2022-06-01 00:57:16.0000	2022-06-01 01:12:10.0000	9.19	1	N	264	107	1	
1	2022-06-01 00:58:37.0000	2022-06-01 01:11:55.0000	3.1	1	N	158	233	1	
2	2022-06-01 00:59:15.0000	2022-06-01 01:06:50.0000	1.93	1	N	90	231	1	

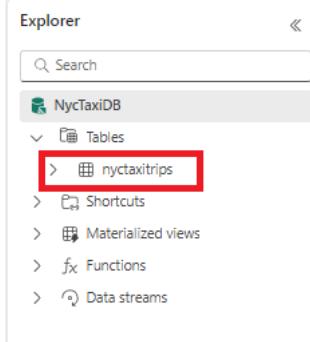
Cancel Back Finish

10. In the **Summary Tab, Continuous ingestion from Event Stream established** window, all steps will be marked with green check marks when the data connection is successfully created. The data from Eventstream will begin streaming automatically into your table. Click on **Close**

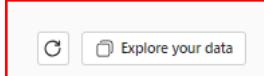


## Explore data and build Power BI report

1. Navigate to your Fabric workspace, select your KQL Database.
2. In the object tree, select the table **nytaxitrips**



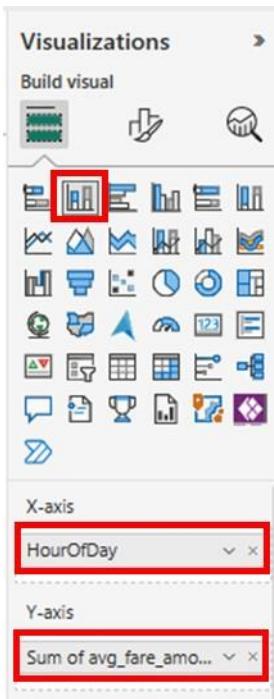
3. In the top right corner, select **Explore your data**.



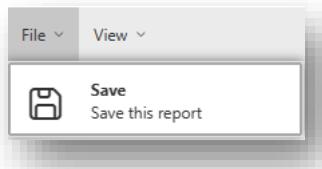
4. Paste the following query in the query editor and select **Run**

```
//Calculate average fare amount by the hour of the day.  
nytaxitrips  
| summarize avg(fare_amount) by HourOfDay = hourofday(tpep_dropoff_datetime)
```

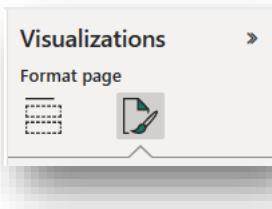
5. Select **Build Power BI Report**. Empty PowerBI report editing window will open.
6. Select **Stacked Column Chart** in the Visualizations pane. Drag **HourOfDay** field to X-axis and **avg\_fare\_amount** to Y-axis



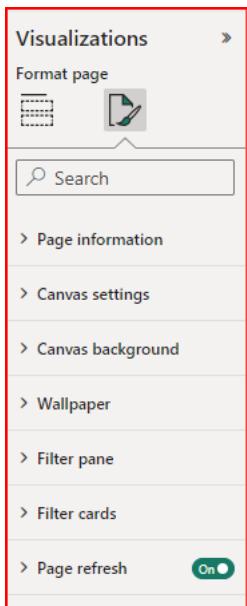
7. Select **File** menu and then **Save** in the top left corner



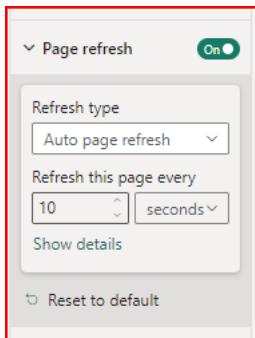
8. Enter **nyctaxitripstats** in the **Name your file in Power BI** field, choose your workspace.
9. Once the report is saved, click on the link **Open the file in Power BI to view, edit, and get a shareable link.**
10. Click on **Edit** button to edit the Power BI report.
11. Click on an Empty Space in the Canvas , Choose **Format page**



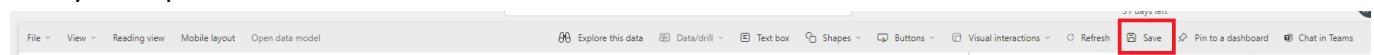
12. Toggle **Page Refresh** to **On**



13. Expand **Page Refresh** and set the refresh interval to 10 seconds. Please note: Refresh interval will be limited by the Admin interval. Refresh interval can only be greater than or equal to the Admin interval.



14. Save your report.



With this tutorial, you have now built an auto-refreshing Power BI report that is querying streaming data arriving in KQL Database from Eventstream.

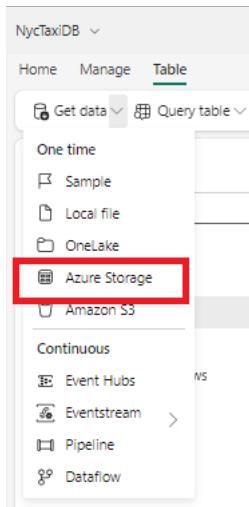
## Module 2: Extending the real-time analytics solution

### Get dimension data from Azure Storage

In this module, you are going to ingest Location available in a Azure storage. This data contains additional information on the pick-up locations and drop-off locations used in the trips dataset. Real-time analytics reads and ingests data directly from the blob storage without requiring any other intermediary service.

1. Navigate to KQL Database **NycTaxiDB**

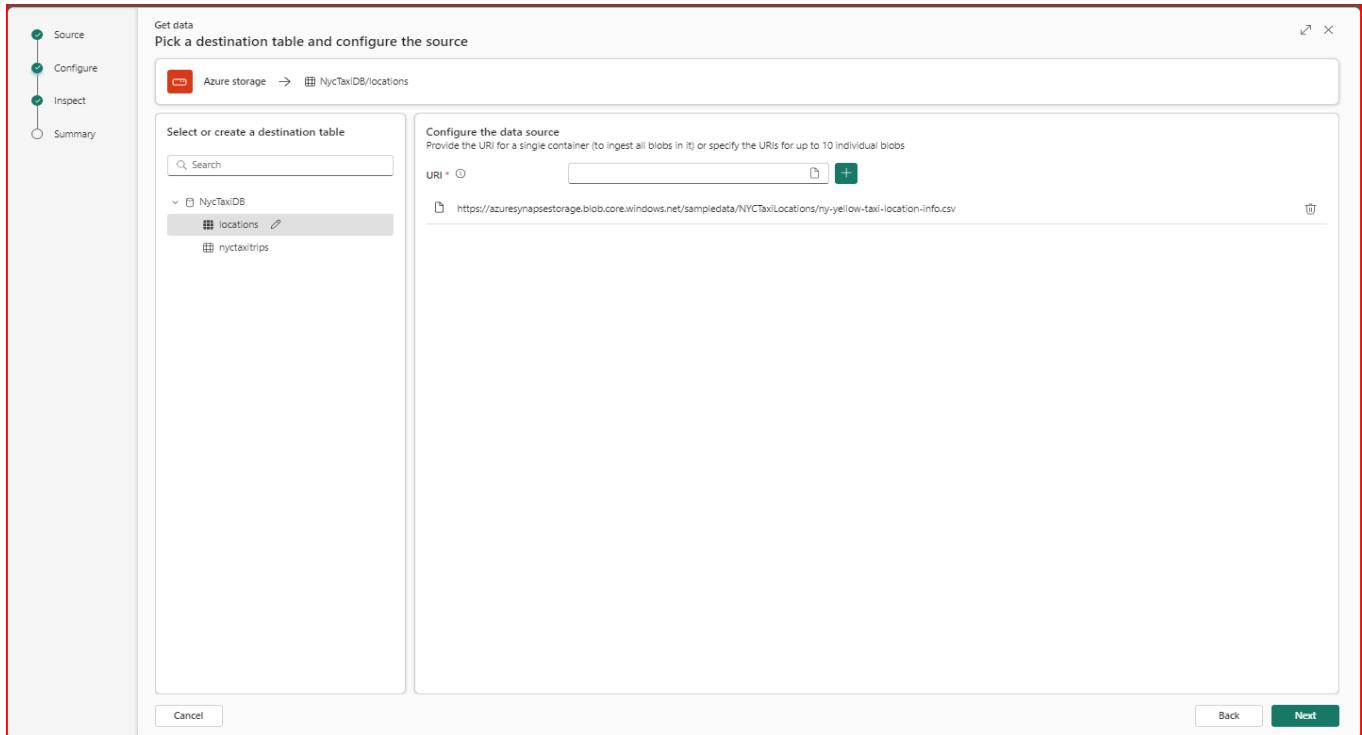
- From within the KQL Database, select **Get Data > Azure Storage**.



### Destination tab

In the **Destination** tab, **Database** is auto populated with the name of the selected KQL database.

- Under **Table**, make sure that **New table** is selected, and enter ***locations*** as your table name.



- URI : <https://azuresynaptestorage.blob.core.windows.net/sampleddata/NYCTaxiLocations/ny-yellow-taxi-location-info.csv> click on + icon and click **Next** :
- In the **Inspect** tab, click on **Finish**

Get data  
Inspect the data

Azure storage → NycTaxiDB/locations

Schema definition file: <https://a...>

Format: CSV

Edit columns Advanced

	LocationID (long)	Borough (string)	Zone (string)	service_zone (string)	Latitude (real)	Longitude (real)
1	EWR	Newark Airport	EWR	40.6895	-74.1745	
2	Queens	Jamaica Bay	Boro Zone	40.6097	-73.8233	
3	Bronx	Allerton/Pelham Gardens	Boro Zone	40.8667	-73.8478	
4	Manhattan	Alphabet City	Yellow Zone	40.7738	-73.9795	
5	Staten Island	Arden Heights	Boro Zone	40.5569	-74.1737	
6	Staten Island	Arrochar/Fort Wadsworth	Boro Zone	40.5964	-74.0671	
7	Queens	Astoria	Boro Zone	40.7644	-73.9235	
8	Queens	Astoria Park	Boro Zone	40.7779	-73.9229	
9	Queens	Auburndale	Boro Zone	40.757	-73.7898	
10	Queens	Baisley Park	Boro Zone	40.6781	-73.7655	
11	Brooklyn	Bath Beach	Boro Zone	40.594224	-74.006131	
12	Manhattan	Battery Park	Yellow Zone	40.70382	-74.017027	
13	Manhattan	Battery Park City	Yellow Zone	40.711932	-74.016869	
14	Brooklyn	Bay Ridge	Boro Zone	40.653801	-74.030621	
15	Queens	Bay Terrace/Fort Totten	Boro Zone	40.791211	-73.776978	
16	Queens	Bayridge	Boro Zone	40.76314	-73.771125	
17	Brooklyn	Bedford	Boro Zone	40.687216	-73.941776	
18	Bronx	Bedford Park	Boro Zone	40.8701	-73.885691	
19	Queens	Bellerose	Boro Zone	40.733014	-73.722938	
20	Bronx	Belmont	Boro Zone	40.85232	-73.88601	
21	Brooklyn	Bensonhurst East	Boro Zone	40.610019	-73.992507	
22	Brooklyn	Bensonhurst West	Boro Zone	40.611011	-74.002925	
23	Staten Island	Bloomfield/Emerson Hill	Boro Zone	40.605883	-74.105138	
24	Manhattan	Bloomingdale	Yellow Zone	40.801181	-73.964546	
25	Brooklyn	Boerum Hill	Boro Zone	40.685683	-73.984501	
26	Brooklyn	Borough Park	Boro Zone	40.633993	-73.996938	
27	Queens	Breezy Point/Fort Tilden/Rili Beach	Boro Zone	40.557154	-73.925042	

Cancel Back Finish

## 6. Summary tab

In the **Data ingestion is in progress** window, all steps will be marked with green check marks when the data has been successfully ingested. The data from Azure Storage will begin streaming automatically into your table.

Get data  
Summary

Azure storage → NycTaxiDB/locations

1 blobs, 1 succeeded, 0 failed

Blob name	Status	Details
https://azuresynapsestorage.blob.core.windows...	Successfully ingested	-

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

Close

Now that you've got data in your database, click on **Close**. You're going to check your data with sample queries.

## Query data

In the following step, you'll use the advanced data analysis capabilities of Kusto Query language to query your telemetry data.

1. Select **Explore your data** on the right-hand side of your database editor.

The screenshot shows the Azure Data Explorer interface for the 'nyctaxitrips' table. It displays various metrics such as Row count (62,900), Last ingestion (Today, this minute), and Size (3.2 MB Compressed size, 10.97 MB Original size). A 'Mappings' section shows no items. The 'Explore your data' button is located at the top right of the table details panel.

Note: The numbers in the screen capture above may look different in your database editor page.

2. Let's take a look at the data itself. Paste the following query in **Explore your data** window to take 10 random records from your data.

Query to be pasted →

*nyctaxitrips  
/ take 10*

The screenshot shows the 'Explore your data' window with the following KQL query:

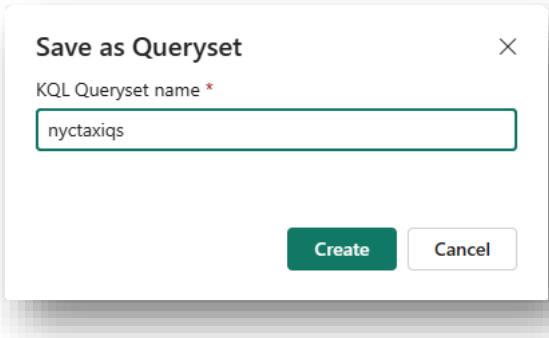
```
nyctaxitrips  
/ take 10
```

3. Select **Run**.

Table 1 Stats

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag
> 2	2022-06-06 18:00:04.0000	2022-06-06 18:09:47.0000	1	2.21	1.0	N
> 2	2022-06-06 18:05:30.0000	2022-06-06 18:25:48.0000	1	3.46	1.0	N
> 2	2022-06-06 18:26:37.0000	2022-06-06 18:33:04.0000	1	1.02	1.0	N
> 2	2022-06-06 18:33:18.0000	2022-06-06 18:37:46.0000	2	0.93	1.0	N
> 2	2022-06-06 18:35:16.0000	2022-06-06 18:55:46.0000	1	6.42	1.0	N
> 2	2022-06-06 18:37:57.0000	2022-06-06 18:53:17.0000	1	9.95	1.0	N
> 2	2022-06-06 18:41:01.0000	2022-06-06 18:49:22.0000	1	1.17	1.0	N
> 2	2022-06-06 18:51:12.0000	2022-06-06 19:04:42.0000	1	4.35	1.0	N
> 2	2022-06-06 18:55:50.0000	2022-06-06 18:56:59.0000	2	0.27	1.0	N
> 2	2022-06-06 18:55:50.0000	2022-06-06 18:56:59.0000	2	0.27	1.0	N

4. Select **Save as KQL Query Set** to save this and future queries for later use.
5. Under **KQL Queryset name**, enter *nyctaxiqs*, then select **Create**.



**'Explore your data'** enables you to run some quick queries to understand your data. This query can be saved as a KQL Queryset and persisted in the workspace as an item. Query set autosaves the queries as you type them and lets you resume from the point where you had stopped. In the next module, you will work with the KQL Queryset. Saving the quick query as KQL Query Set will automatically open your **KQL Queryset** with the queries that you wrote in the query editor.

## Explore data further in the KQL Queryset

In this module, you are going to write queries using *Kusto Query Language* to explore the data that you have ingested from the Event hub and blob storage. 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 SQLs: databases, tables, and columns. 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. You are going to write some simple Kusto queries to get familiar with the language and discover its power and simplicity.

Run the following queries in the new KQL Queryset you have created. Copy/paste each query into your environment, select the query and then select **Run**.

1. The following query returns the top 10 pickup locations in New York City for Yellow Taxis.

Query to be  
pasted →

```
//Top 10 pickup locations
nyctaxitrips
| summarize count() by PULocationID
| top 10 by count_
```

PULocationID	count_
> 237	61,883
> 236	55,365
> 132	55,057
> 161	47,341
> 162	40,191
> 142	40,128
> 186	37,716
> 170	37,533
> 230	35,370

**Note:** Result of the query may not exactly match the screenshot provided as you are ingesting streaming data.

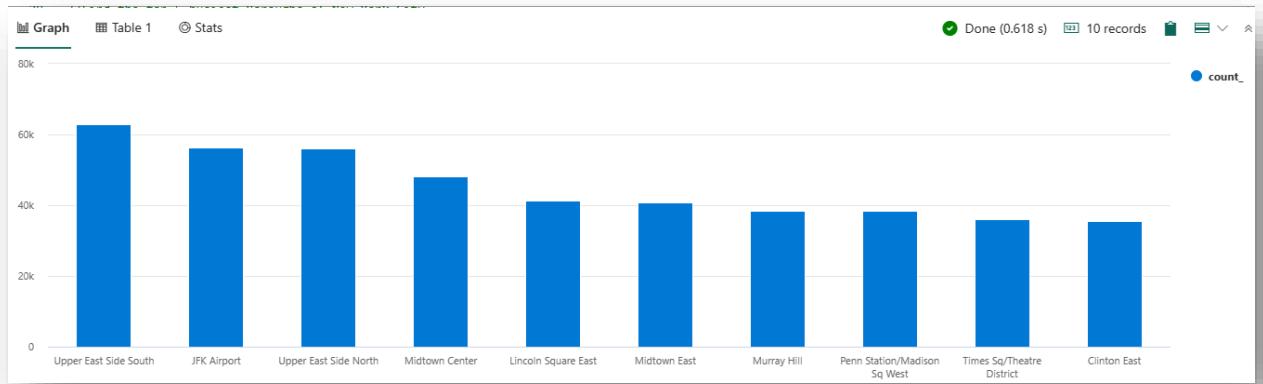
2. We will run the same query as in the previous step with an addition of looking up the corresponding zones of the top 10 pickup locations by using the ‘locations’ table.

Query to be  
pasted →

```
//For the same top 10 locations, lookup the NYC zones --> Top 10 zones
nyctaxitrips
| lookup (locations) on $left.PULocationID == $right.LocationID
| summarize count() by Zone
| top 10 by count_
| render columnchart
```

Zone	count_
> Upper East Side South	62,942
> JFK Airport	56,417
> Upper East Side North	56,052
> Midtown Center	48,263
> Lincoln Square East	41,236
> Midtown East	40,895
> Murray Hill	38,479
> Penn Station/Madison Sq West	38,324
> Times Sq/Theatre District	36,167
> Clinton East	35,622

**Note:** Result of the query may not exactly match the screenshot provided as you are ingesting streaming data.

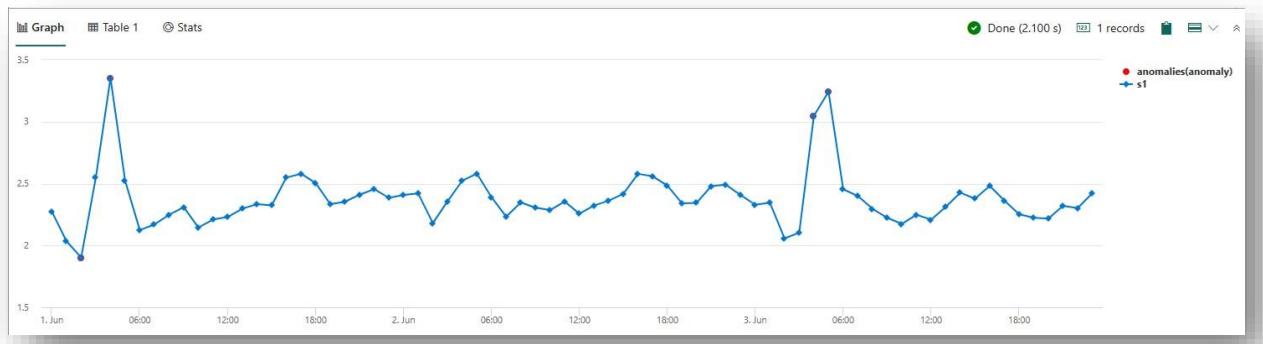


3. Let's check anomalies in the tips that have been given by the customers in the Manhattan borough. Hover over the red dots to see the values.

Query to be pasted →

```
//Find anomalies in the tips given by the customers
nyctaxitrips
| lookup (locations) on $left.PULocationID==$right.LocationID
| where Borough == "Manhattan"
| make-series s1 = avg(tip_amount) on tpep_pickup_datetime from datetime(2022-06-01) to datetime(2022-06-04) step 1h
| extend anomalies = series_decompose_anomalies(s1)
| render anomalychart with (anomalycolumns=anomalies)
```

**Note:** Result of the query may not exactly match the screenshot provided as you are ingesting streaming data.

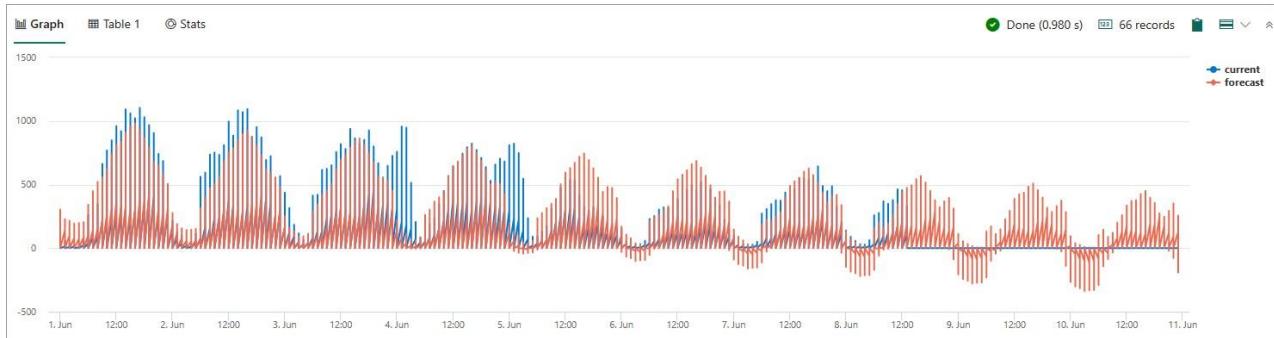


- To ensure that the sufficient taxis are plying in the Manhattan borough, let's forecast the number of taxis needed per hour.

*//Forecast the number of trips that will begin from Manhattan to line up the taxis in that borough*

```
nyctaxitrips
| lookup (locations) on $left.PULocationID==$right.LocationID
| where Borough == "Manhattan"
| make-series s1 = count() on tpep_pickup_datetime from datetime(2022-06-01) to datetime(2022-06-08)+3d step 1h by PULocationID
| extend forecast = series_decompose_forecast(s1, 24*3)
| render timechart
```

**Note:** Result of the query may not exactly match the screenshot provided below as you are ingesting streaming data.



## Build Power BI report

A Power BI report is a multi-perspective view into a dataset, with visuals that represent findings and insights from that dataset.

- Continuing the in same queryset, paste the following query. The output of this query will be used as the dataset for building the Power BI report.

*//Find the total number of trips that started and ended at the same location*

```
nyctaxitrips
| where PULocationID == DOLocationID
| lookup (locations) on $left.PULocationID==$right.LocationID
| summarize count() by Borough, Zone, Latitude, Longitude
```

- Select the query and then select **Build Power BI report**.

The screenshot shows the Kusto Query Editor interface. At the top, there's a navigation bar with 'Home' and 'Manage' buttons, followed by 'Run', 'Recall', 'Save', 'Copy query', and a button labeled 'Build Power BI report' which is highlighted with a red box. Below the navigation is a database dropdown set to 'NycTaxiDB' with a plus sign icon. A search bar is present. On the left, a tree view shows 'Database' expanded, with 'NycTaxiDB' selected, and two tables: 'locations' and 'nyctaxitrips'. The main area displays a Kusto query:

```

28: nyctaxitrips
29: | where PUlocationID == $OlocationID
30: | lookup (locations) on $left.PUlocationID==$right.LocationID
31: | summarize count() by Borough, Zone, Latitude, Longitude
32:
33:

```

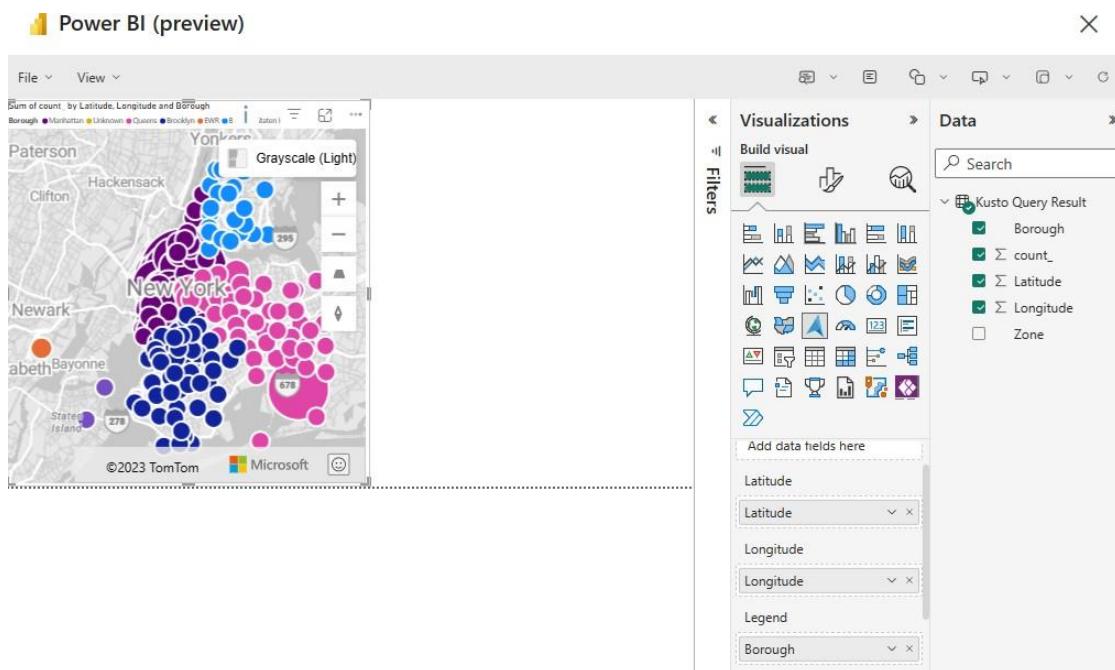
Power BI report editor will open with the result of the query available as a table with the name Kusto Query Result.

Note:

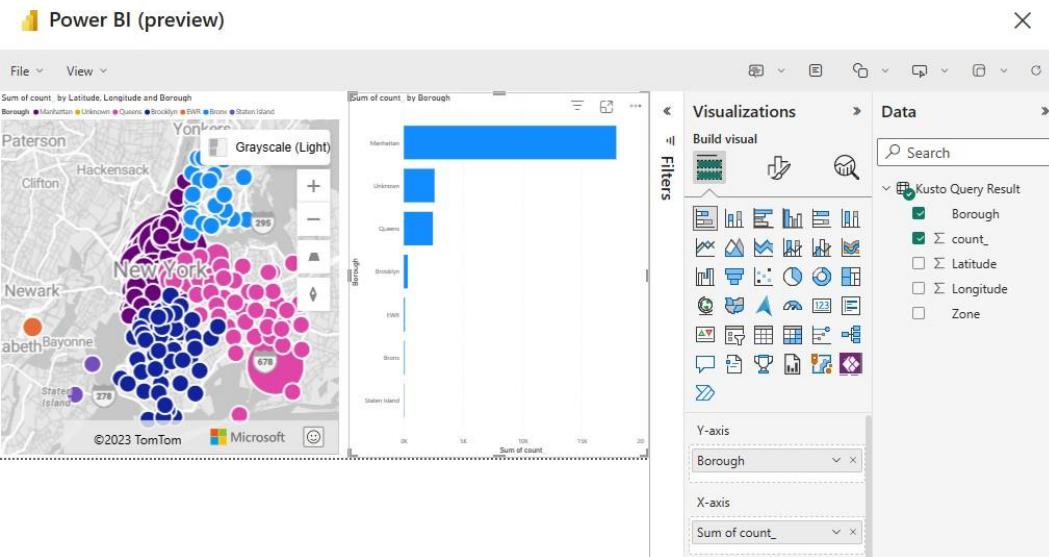
(a) When you build a report, a dataset is created and saved in your workspace. You can create multiple reports from a single dataset. (b) Result of the query may not exactly match the screenshot provided below as you are ingesting streaming data.

If you delete the dataset, your reports will also be removed.

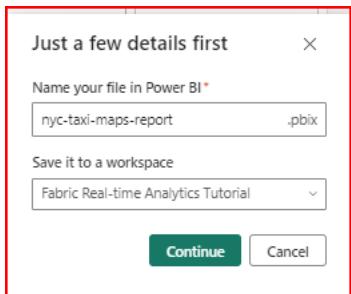
- In the report editor, choose **Maps** as the visual, drag **Latitude** field to Latitude, **Longitude** field to Longitude, **Borough** field to Legend, and **count\_** field to Size.



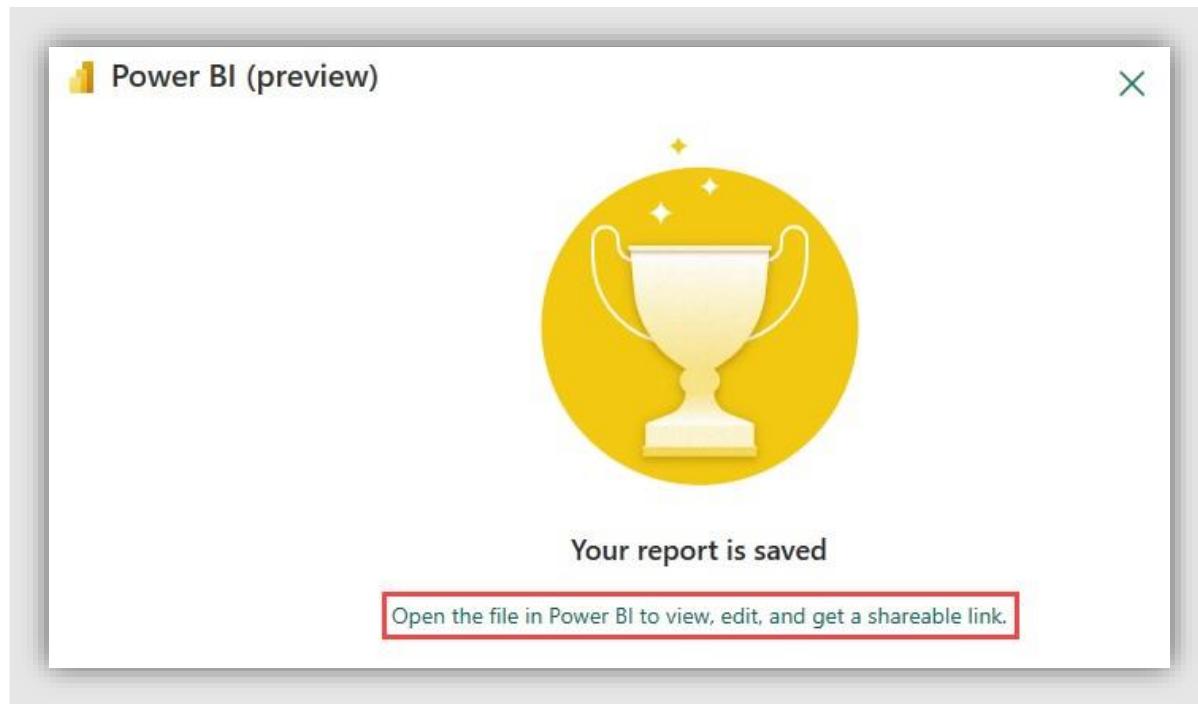
- Add a **Stacked Bar Chart** to the canvas. Drag **Borough** field to Y-Axis and **count\_** to the X-axis



5. Click File > Save
6. Under **Name your file in Power BI**, enter *nyc-taxi-maps-report*.



7. Select the workspace in which you want to save this report. The report can be saved in a different workspace than the one you started in.
8. Select **Open the file in Power BI to view, edit, and get a shareable link** to view and edit your report.



## Create OneLake shortcut

Now that you have finished exploring your data, you may want to access the underlying data from other Fabric experiences.

OneLake is a single, unified, logical data lake for Fabric to store lakehouses, warehouses and other items. Shortcuts are embedded references within OneLake that point to other files' store locations. The embedded reference makes it appear as though the files and folders are stored locally but in reality; they exist in another storage location. Once you create a shortcut, you can access your data in all of Fabric's experiences. Shortcuts can be updated or removed from your items, but these changes will not affect the original data and its source.

1. Select **Create** in the **Navigation pane**.

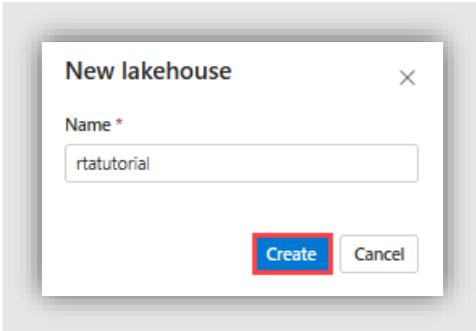


2. Under **Data engineering**, select **Lakehouse**.

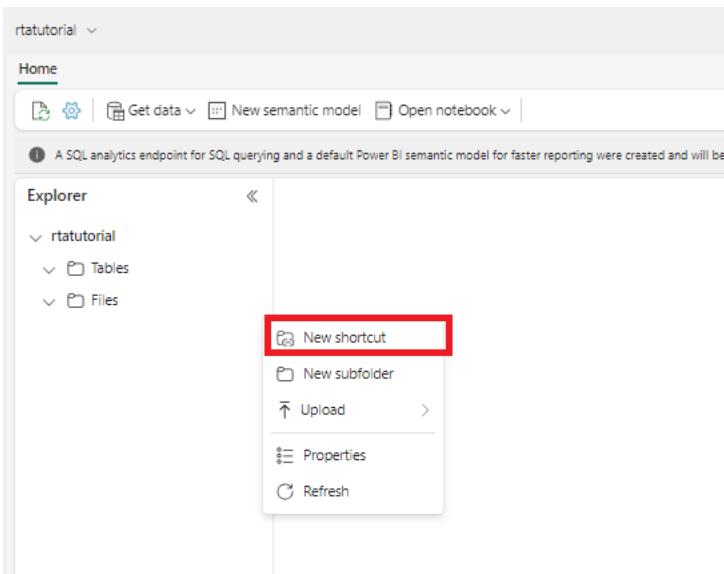
**Data Engineering**  
Create a lakehouse and operationalize your workflow to build, transform, and share your data estate.

- Lakehouse** Store big data for cleaning, querying, reporting, and sharing.
- Notebook** Explore data and build machine learning solutions with Apache Spark applications.
- Environment (Preview)** Set up shared libraries, Spark compute settings, and resources for notebooks and Spark job definitions.
- Spark Job Definition** Define, schedule, and manage your Apache Spark jobs for big data processing.

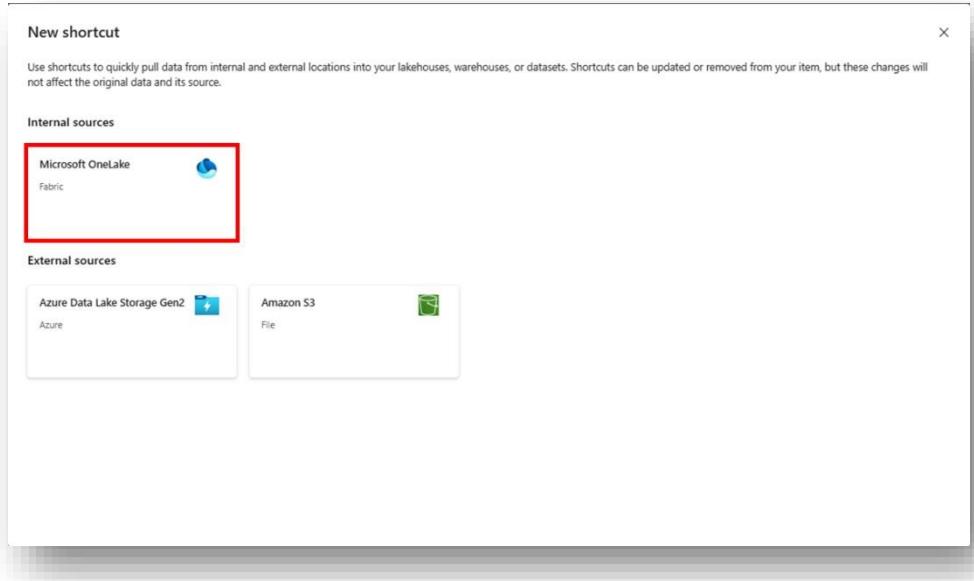
- Enter *rstatutorial* as your Lakehouse name, then select **Create**.



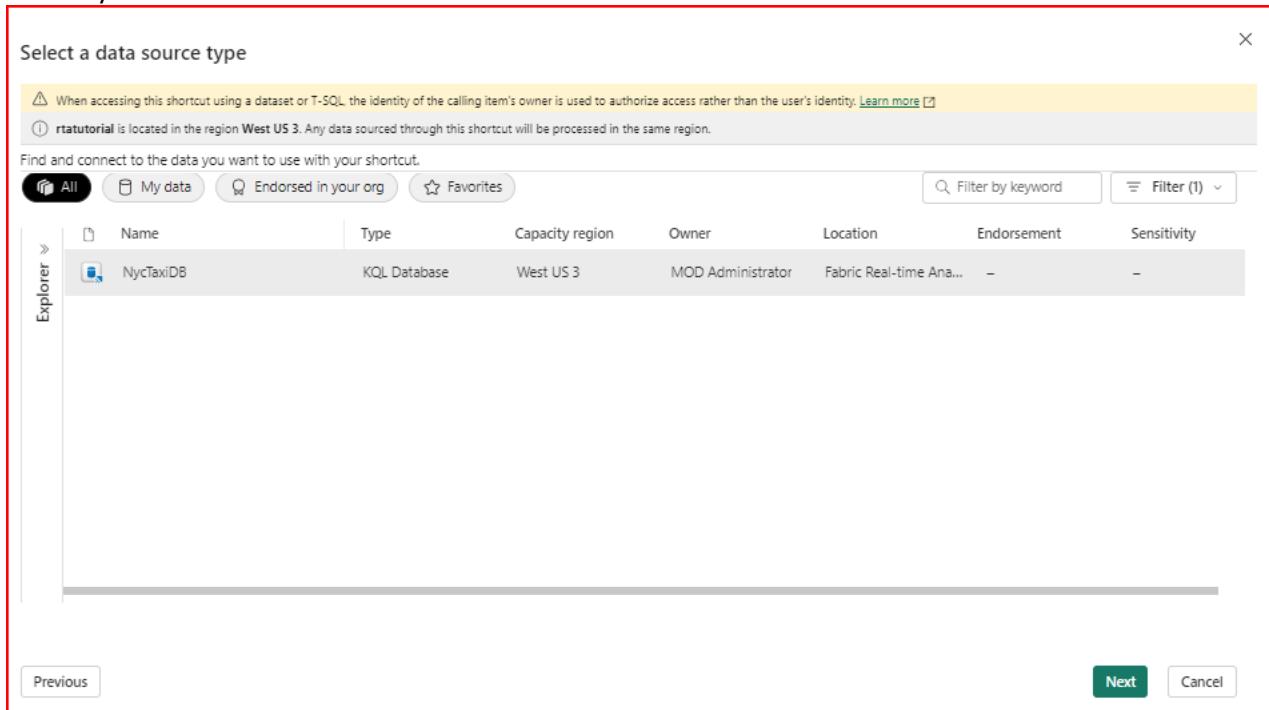
- In Lakehouse Explorer, navigate to **Files**, select "...", select **New Shortcut** from the menu.



- Under **Internal sources**, select **OneLake**.



- In **Select a data source type**, Filter KQL Database and select **your KQL Database**, then select **Next** to connect the data to your shortcut.



- To connect the table with the data from Eventstream, select > to expand the tables in the left-hand pane, then select the table titled **nytaxitrips**.

New shortcut

When accessing this shortcut using a dataset or T-SQL, the identity of the calling item's owner is used to authorize access rather than the user's identity. [Learn more](#)

rtatutorial is located in the region West US 3. Any data sourced through this shortcut will be processed in the same region.

Find and connect to the data you want to use with your shortcut.

Select the table and files that you want to include in your shortcut.

OneLake

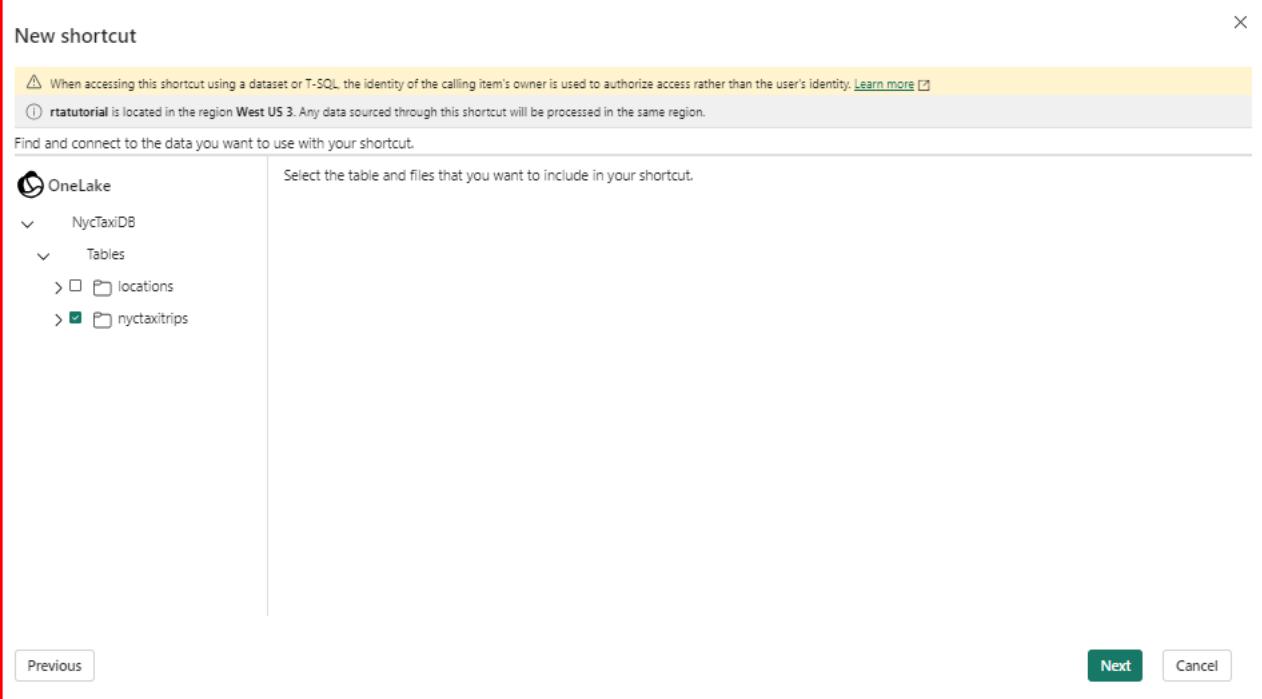
NycTaxiDB

Tables

locations

nyctaxitrips

Previous Next Cancel



4. Select **Create** to create the shortcut. The Lakehouse will automatically refresh. Click on Close

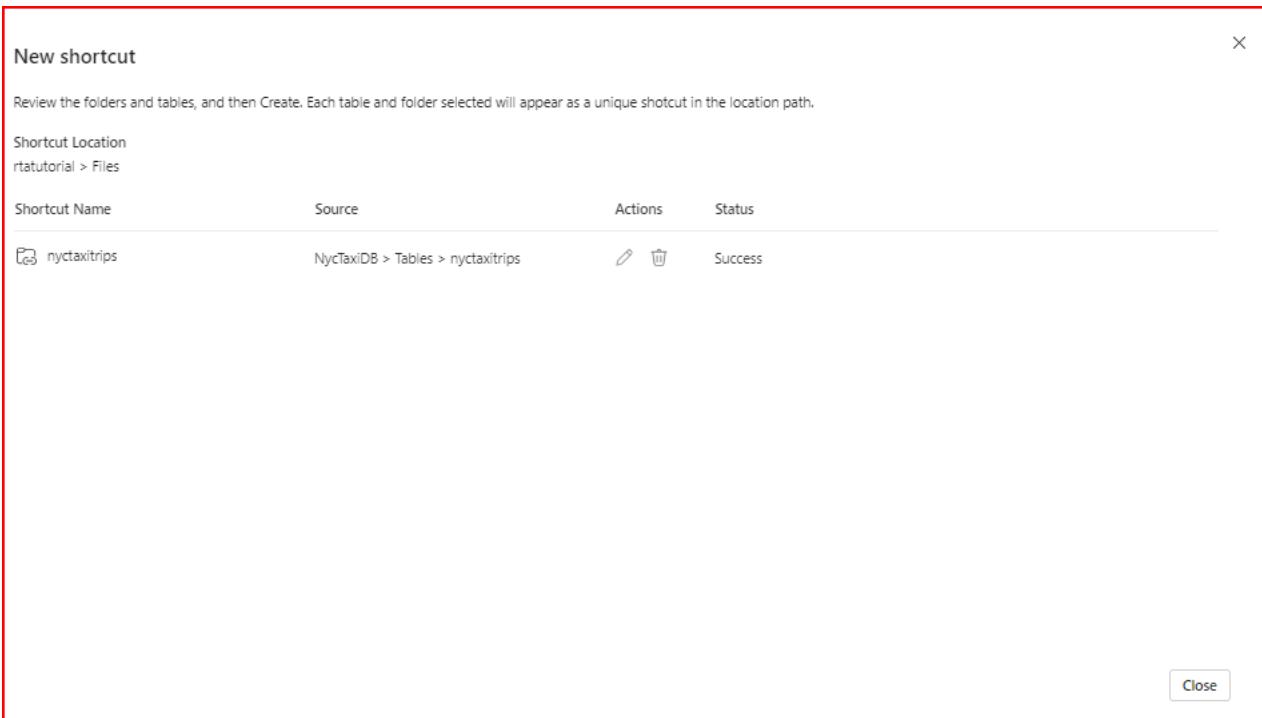
New shortcut

Review the folders and tables, and then Create. Each table and folder selected will appear as a unique shortcut in the location path.

Shortcut Location  
rtatutorial > Files

Shortcut Name	Source	Actions	Status
nyctaxitrips	NycTaxiDB > Tables > nyctaxitrips		Success

Close



The Lakehouse shortcut has been created. Without additional management, you now have one logical copy of your data in the Lakehouse that you can use in other Fabric experiences, such as Notebooks and Spark jobs.