Real-Time Intelligence System for Event-Driven Data Analysis

Event-Driven Data Analysis with Microsoft Fabric

Executive Summary

This project implements a comprehensive real-time data intelligence system using Microsoft Fabric's Real-Time Intelligence capabilities. The solution processes streaming event data, performs data transformations, and delivers actionable insights through visualization and automated alerting mechanisms.

Project Scope

This implementation focuses on building an end-to-end real-time analytics solution capable of handling event-driven scenarios and streaming data workflows. The system is designed to process, transform, and visualize data in motion, enabling immediate insight extraction and decision-making.

Business Context

The project addresses the need for real-time monitoring and analysis of operational data streams. Using a bicycle tracking dataset as the implementation case study, the system processes continuous streams of location-based telemetry data including vehicle identification, geographic coordinates, timestamps, and operational metadata.

Project Overview

The implementation delivers the following key capabilities:

- ***Infrastructure Setup*** *– Configure the necessary environment and resources for real-time data processing*
- ***Data Ingestion Pipeline*** *– Establish data capture mechanisms through Real-Time hub integration*

- ***Stream Processing*** *– Implement event transformation logic for data enrichment and standardization*
- ***Data Publishing*** *– Deploy eventstream functionality for downstream consumption*
- ***Advanced Transformations*** *– Utilize update policies for in-flight data transformation within Eventhouse*

- ***Custom Analytics*** *– Develop KQL (Kusto Query Language) queries for specific analytical requirements*
- ***Proactive Monitoring*** *– Configure query-based alerting for threshold detection and anomaly response*
- ***Interactive Visualization*** *– Build Real-Time dashboards for operational monitoring*
- ***Visual Data Exploration*** *– Enable ad-hoc analysis through interactive dashboard components*
- ***Anomaly Detection*** *– Implement automated pattern recognition on Eventhouse data tables*
- ***Geospatial Analysis*** *– Create location-based visualizations using geographic coordinates*
- ***Stream-Level Alerts*** *– Establish immediate notification mechanisms at the eventstream layer*

Technical Requirements

System Prerequisites

- *Active workspace with Microsoft Fabric-enabled capacity allocation*
- *Administrator-level configuration enabling:*
    - *Maps visualization preview feature*
    - *Anomaly detector preview functionality*
    - *Access configured through the Fabric admin portal*

Project Deliverables

This implementation produces a fully operational real-time intelligence system with integrated monitoring, alerting, visualization, and analytical capabilities suitable for production deployment and enterprise-scale event processing scenarios.

Setting Up Real-Time Data Infrastructure

This involves establishing a real-time data analytics infrastructure using Microsoft Fabric's Eventhouse service. The implementation focuses on creating a centralized data repository capable of handling streaming data and real-time queries.

- *Deploy a scalable Eventhouse environment for real-time data processing*
- *Establish a KQL (Kusto Query Language) database for analytical operations*
- *Configure the foundational infrastructure for time-series data management*

Implementation Steps

1. Workspace Configuration

To ensure proper resource organization and access management, all project components were consolidated within a single workspace. This approach:

- *Maintains logical grouping of related resources*
- *Simplifies permission management*
- *Ensures consistent networking and security policies*

2. Eventhouse Deployment

Navigation and Resource Creation:

- *Accessed the designated project workspace*
- *Initiated resource creation via the **+ New item** option*
- *Filtered available services using the search term "Eventhouse"*

Configuration Details:

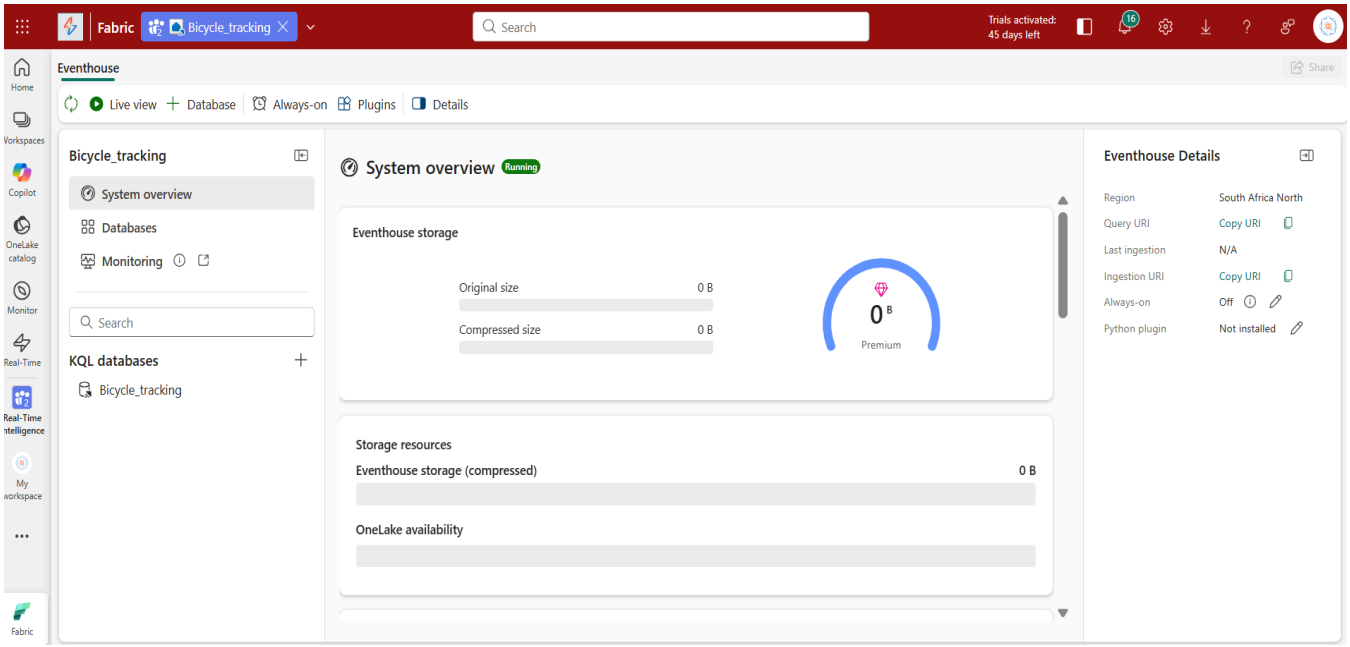
- ***Resource Name:** Bicycle_tracking*
- ***Service Type:** Eventhouse*
- ***Auto-provisioned Component:** KQL Database (inherits parent name)*

Key Technical Note: The Eventhouse service automatically provisions a child KQL database with an identical naming convention, establishing a parent-child relationship that streamlines data organization.

3. Provisioning Verification

Upon successful deployment:

- *The system displayed the Eventhouse **System Overview** page*
- *This dashboard provides operational metrics and management capabilities*
- *Confirmed that both the Eventhouse and its associated KQL database were active*



Technical Architecture

The implemented solution consists of:

- ***Eventhouse:** High-level container for real-time analytics resources*
- ***KQL Database:** Query-optimized database for time-series and streaming data analysis*

Skills Demonstrated

- *Cloud resource provisioning in Microsoft Fabric*
- *Understanding of real-time data platform architecture*
- *Knowledge of KQL database infrastructure*
- *Workspace management and resource organization*

Get data in the Real-Time Hub

This demonstrates the implementation of a real-time data pipeline using event stream processing capabilities. The solution involves setting up a data ingestion pipeline that captures, transforms, and routes streaming data to a database for analysis.

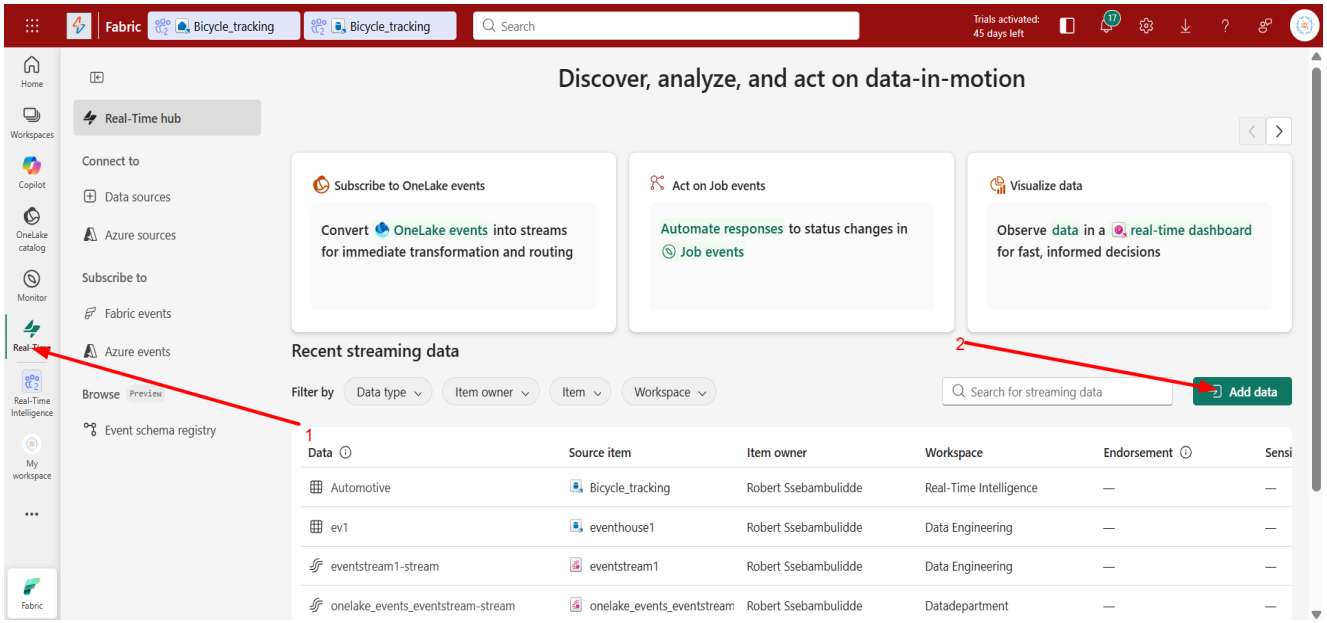
Implementation Components

1. Event Stream Configuration

Establish a real-time data ingestion pipeline for continuous data flow.

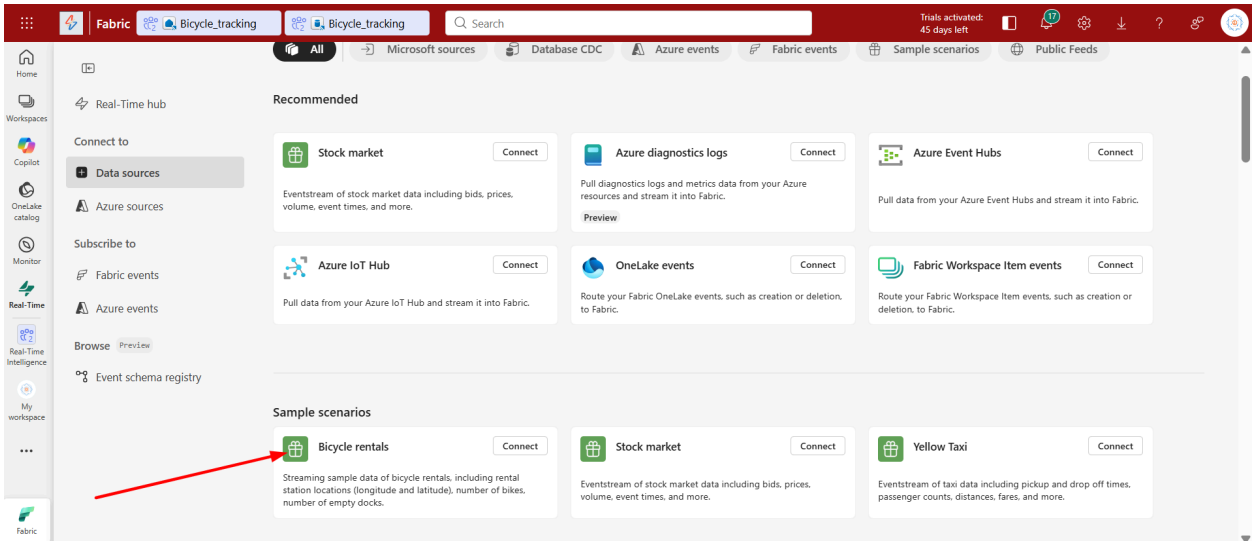
Implementation Steps:

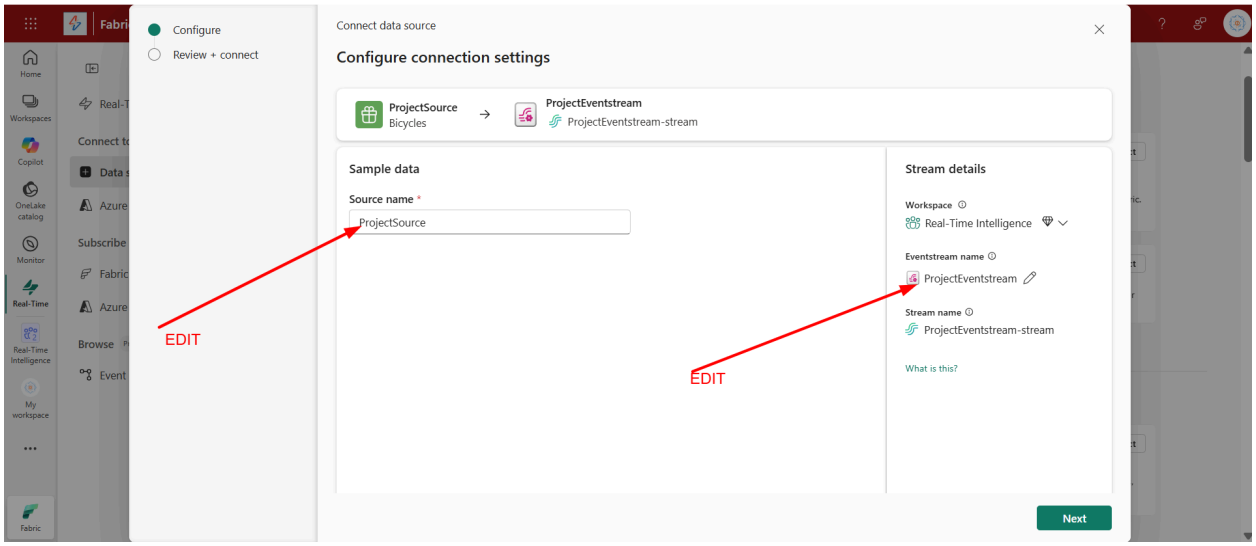
1. **Access Real-Time Hub**
    - *Navigate to the Real-Time section from the main navigation panel*
    - *Initialize data connection by selecting the add data option*

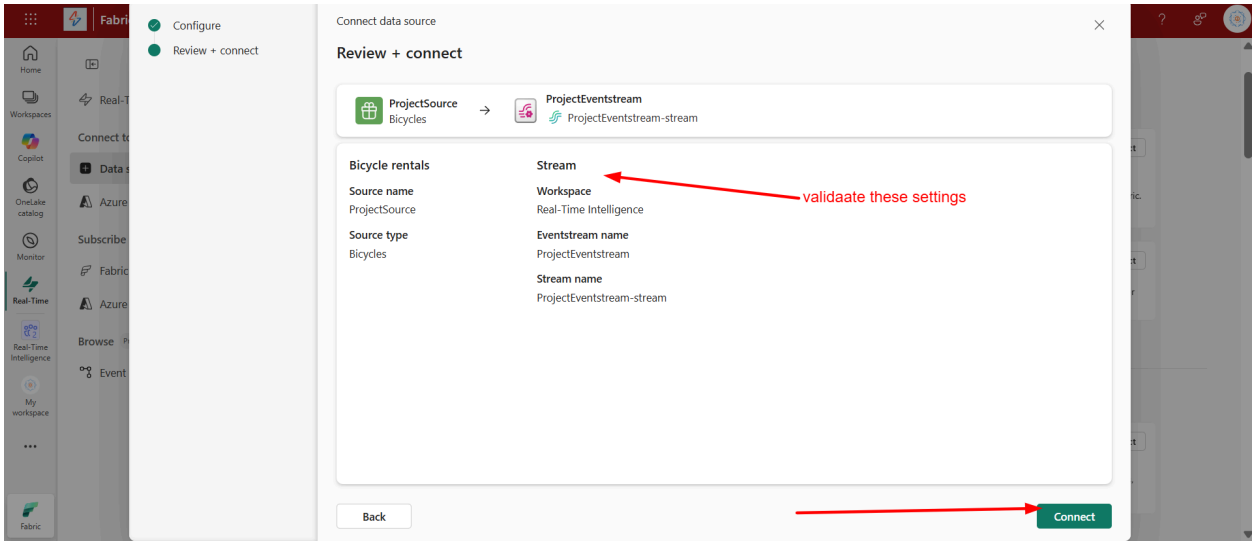


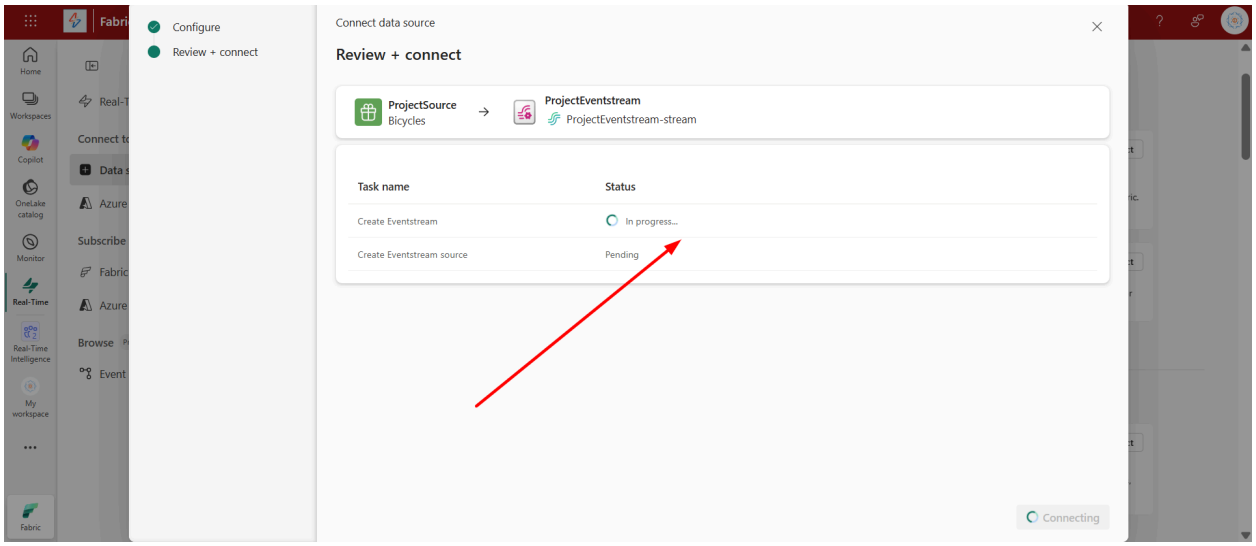
2. **Source Configuration**

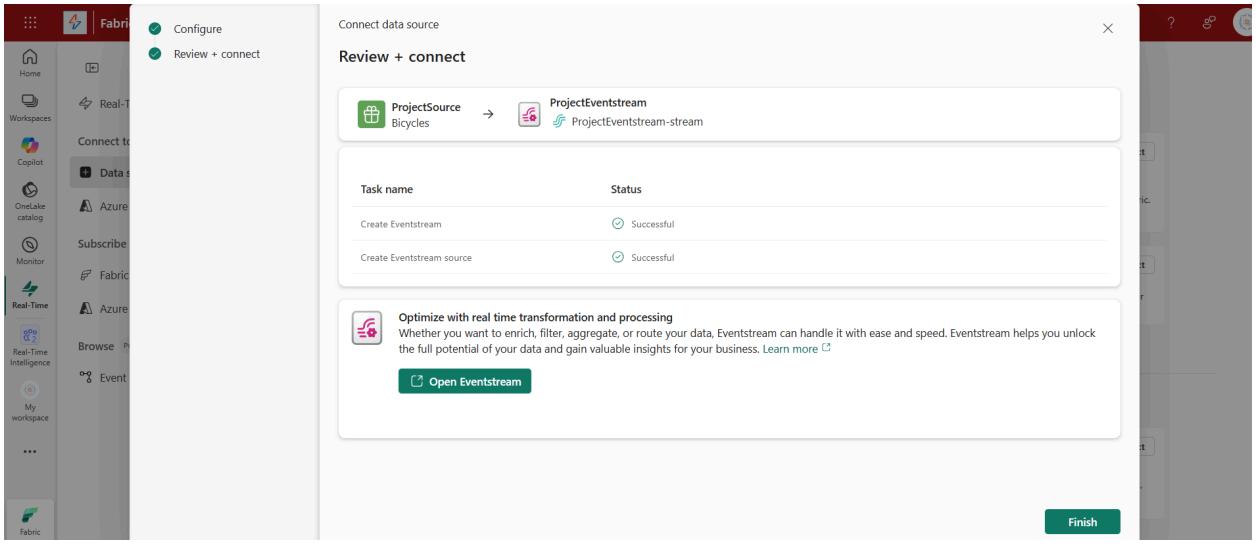
- *Selected the Bicycle rentals sample scenario as the data source*
- *Configured source parameters:*
  - *Source identifier: ProjectSource*
  - *Stream name: ProjectEventstream*
- *Validated configuration settings before establishing connection*




Configure
Review + connect
Connect data source
Configure connection settings
ProjectSource
Bicycles
ProjectEventstream
ProjectEventstream-stream
Sample data
Source name *
ProjectSource
Stream details
Workspace
Real-Time Intelligence
Eventstream name
ProjectEventstream
Stream name
ProjectEventstream-stream
What is this?
EDIT
EDIT
Next


Configure
Review + connect
Connect data source
Review + connect
ProjectSource
Bicycles
ProjectEventstream
ProjectEventstream-stream
Bicycle rentals
Source name
ProjectSource
Source type
Bicycles
Stream
Workspace
Real-Time Intelligence
Eventstream name
ProjectEventstream
Stream name
ProjectEventstream-stream
validaate these settings
Back
Connect


Configure
Review + connect
Connect data source
Review + connect
ProjectSource
Bicycles
ProjectEventstream
ProjectEventstream-stream
Task name
Status
Create Eventstream
In progress...
Create Eventstream source
Pending
Connecting



Outcome: Successfully established a continuous data stream from the selected source.

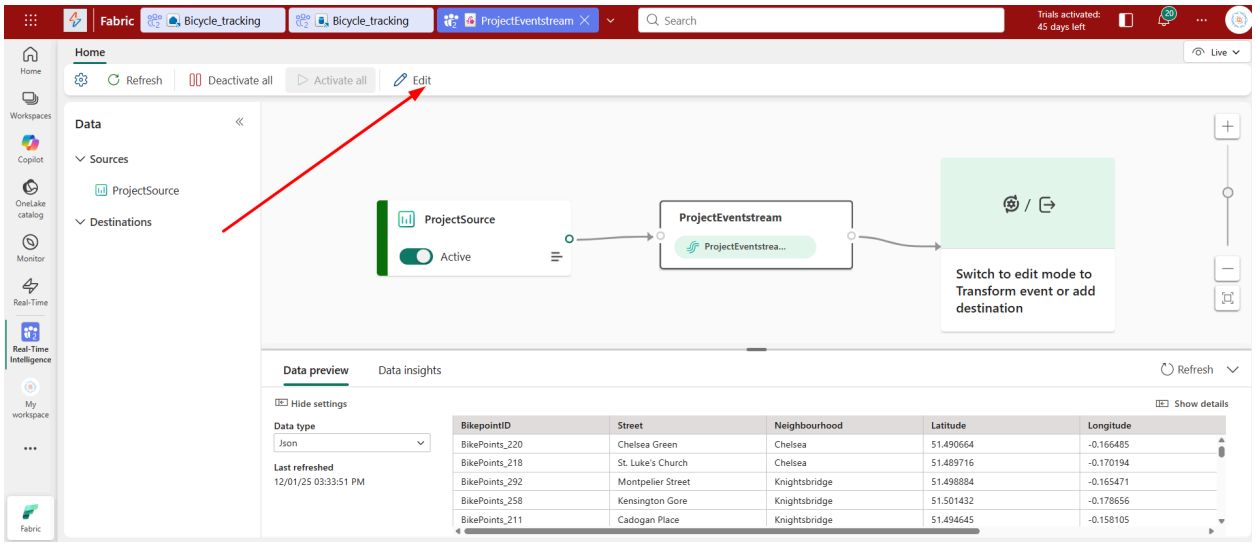
2. Event Transformation Logic

Implement data transformation to enrich incoming events with timestamp metadata.

Technical Implementation:

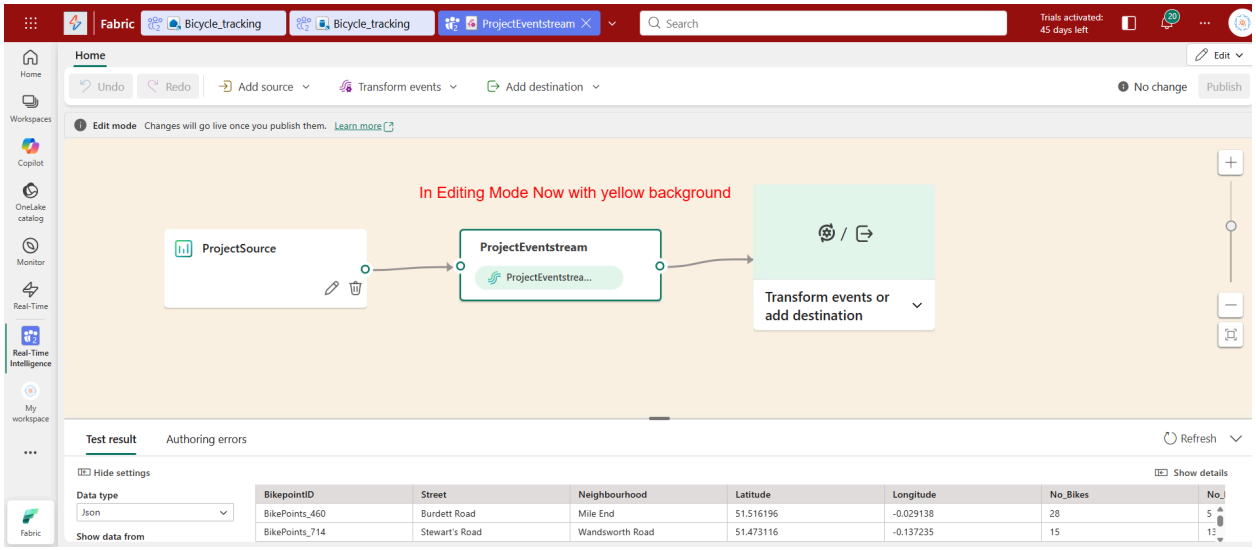
The transformation layer was configured to add temporal context to each event:

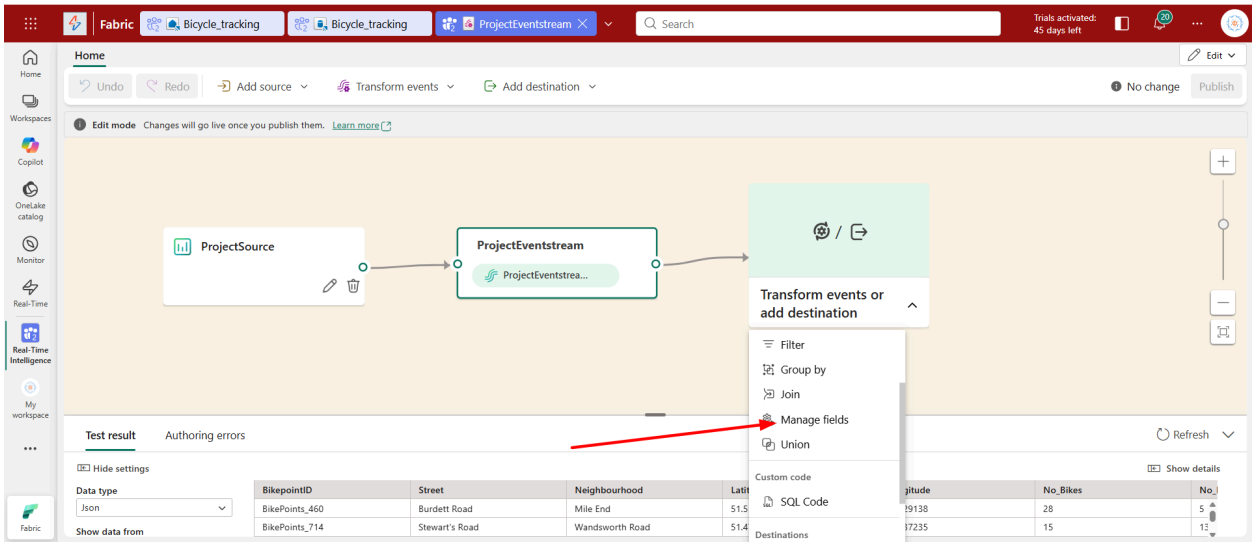
1. **Transformation Setup**
    - *Activated the editing mode in the event stream authoring interface*
    - *Added a field management transformation component named* ProjectTransform

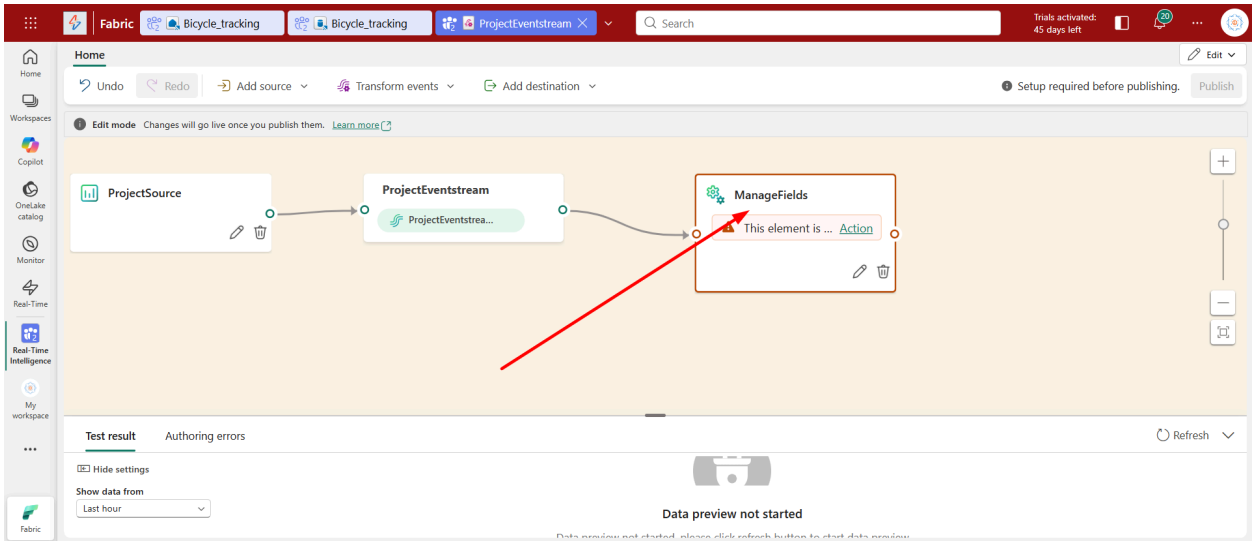


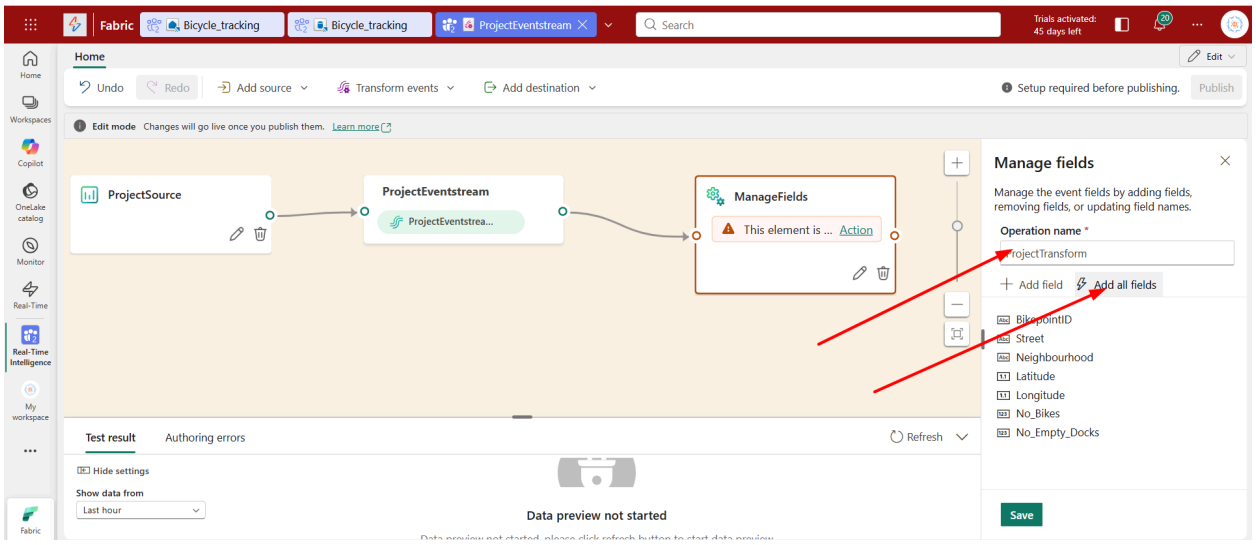
2. **Field Mapping Configuration**

- *Preserved all existing fields from the source data*
- *Implemented custom field addition using built-in functions*



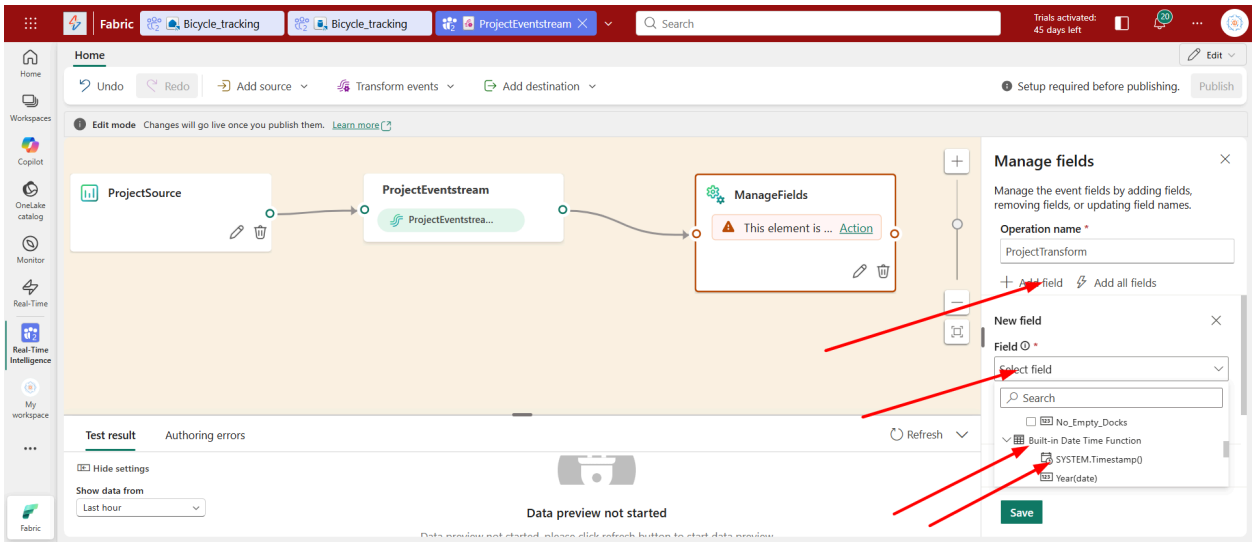

Fabric
Bicycle_tracking
Bicycle_tracking
ProjectEventstream
Search
Trials activated: 45 days left
Home
Edit
Undo
Redo
Add source
Transform events
Add destination
No change
Publish
Edit mode
Changes will go live once you publish them.
Learn more
ProjectSource
ProjectEventstream
ProjectEventstrea...
Transform events or add destination
Filter
Group by
Join
Manage fields
Union
Custom code
SQL Code
Destinations
Test result
Authoring errors
Refresh
Hide settings
Show details
Data type
Json
Show data from
BikepointID
Street
Neighbourhood
No_Bikes
BikePoints_460
Burdett Road
Mile End
28
BikePoints_714
Stewart's Road
Wandsworth Road
15

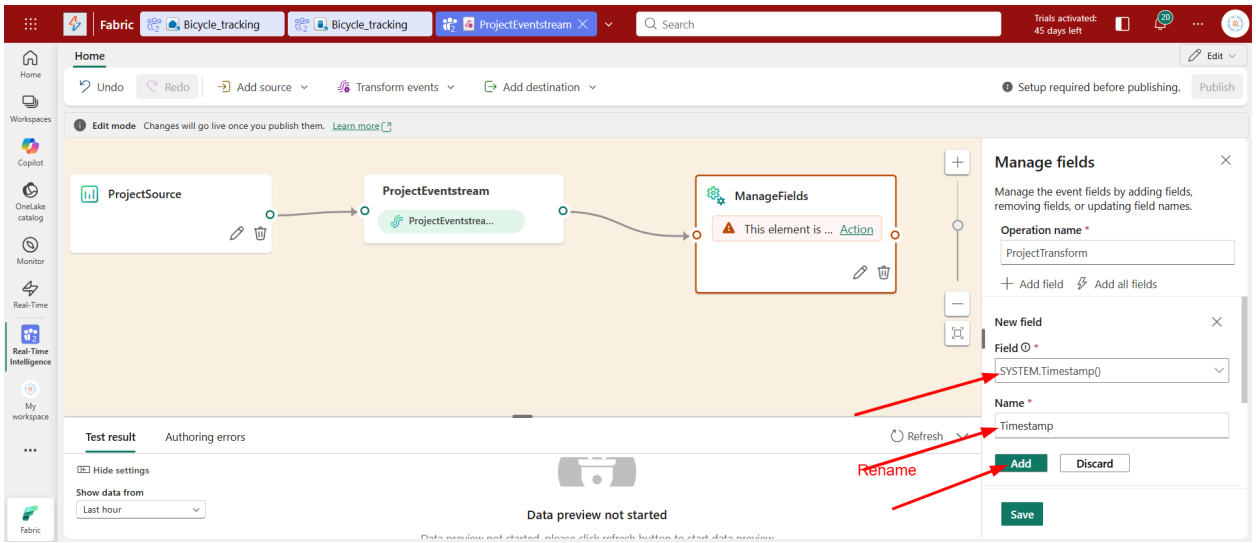

Fabric
Bicycle_tracking
Bicycle_tracking
ProjectEventstream
Search
Trials activated: 45 days left
Home
Edit
Undo
Redo
Add source
Transform events
Add destination
Setup required before publishing.
Publish
Edit mode
Changes will go live once you publish them.
Learn more
ProjectSource
ProjectEventstream
ProjectEventstrea...
ManageFields
This element is ... Action
Test result
Authoring errors
Refresh
Hide settings
Show data from
Last hour
Data preview not started

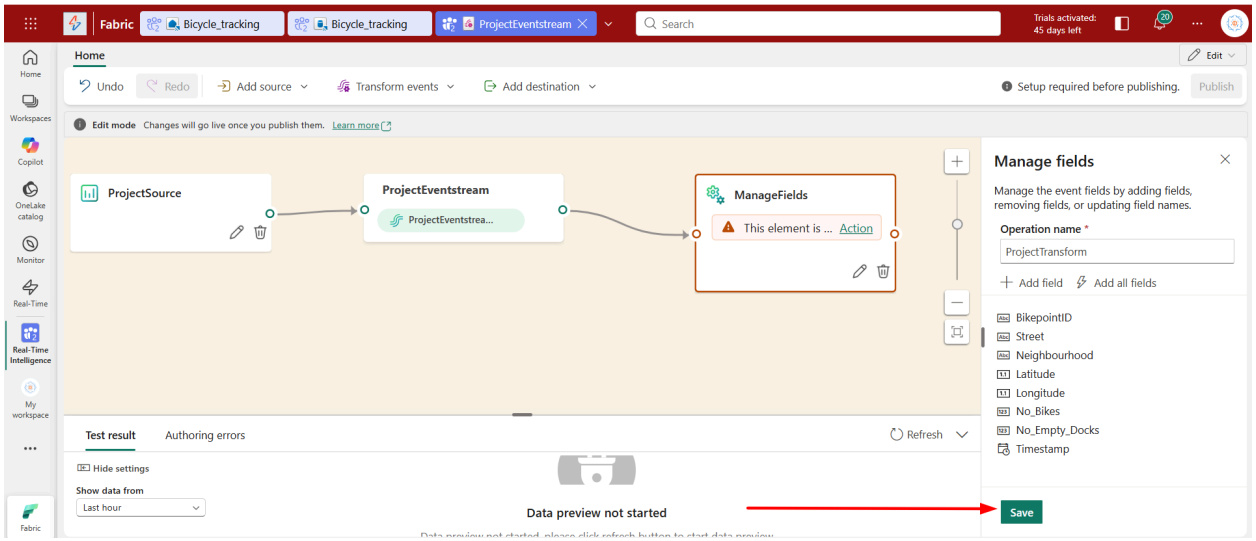

Fabric
Bicycle_tracking
Bicycle_tracking
ProjectEventstream
Search
Trials activated: 45 days left
Home
Edit
Undo
Redo
Add source
Transform events
Add destination
Setup required before publishing.
Publish
Edit mode
Changes will go live once you publish them.
Learn more
ProjectSource
ProjectEventstream
ProjectEventstrea...
ManageFields
This element is ... Action
Manage fields
Manage the event fields by adding fields, removing fields, or updating field names.
Operation name *
ProjectTransform
Add field
Add all fields
BikepointID
Street
Neighbourhood
Latitude
Longitude
No_Bikes
No_Empty_Docks
Save
Test result
Authoring errors
Refresh
Hide settings
Show data from
Last hour
Data preview not started

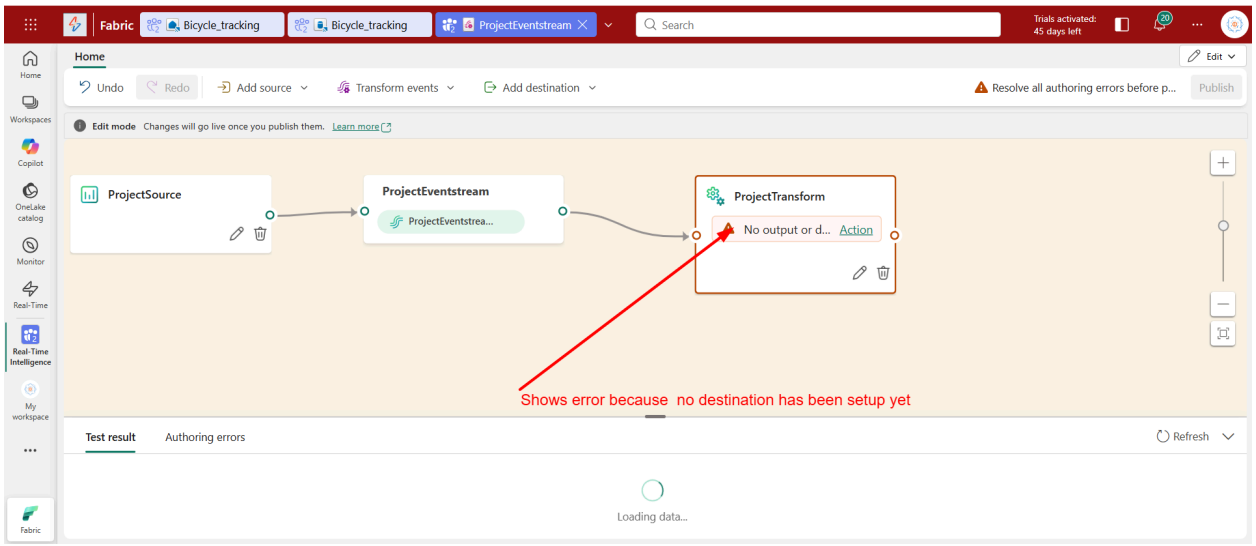
3. **Timestamp Enrichment**

- *Function utilized: SYSTEM.Timestamp()*
- *Field name: Timestamp*
- *Purpose: Capture the exact moment each event enters the processing pipeline*









Transformation Logic:

Operation: Manage Fields
- Retain: All source fields
- Add: Timestamp field (SYSTEM.Timestamp())
- Output: Enriched event payload with temporal metadata

3. Destination Configuration

Route transformed events to a persistent storage layer for downstream analysis.

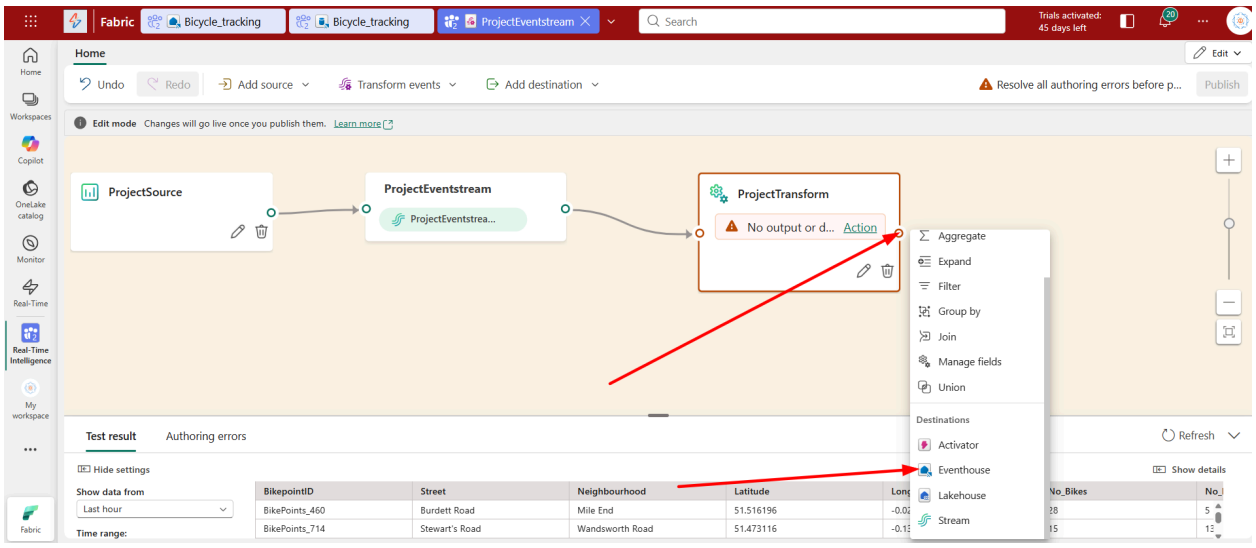
Configuration Details:

Established connection to an Eventhouse database with the following specifications:

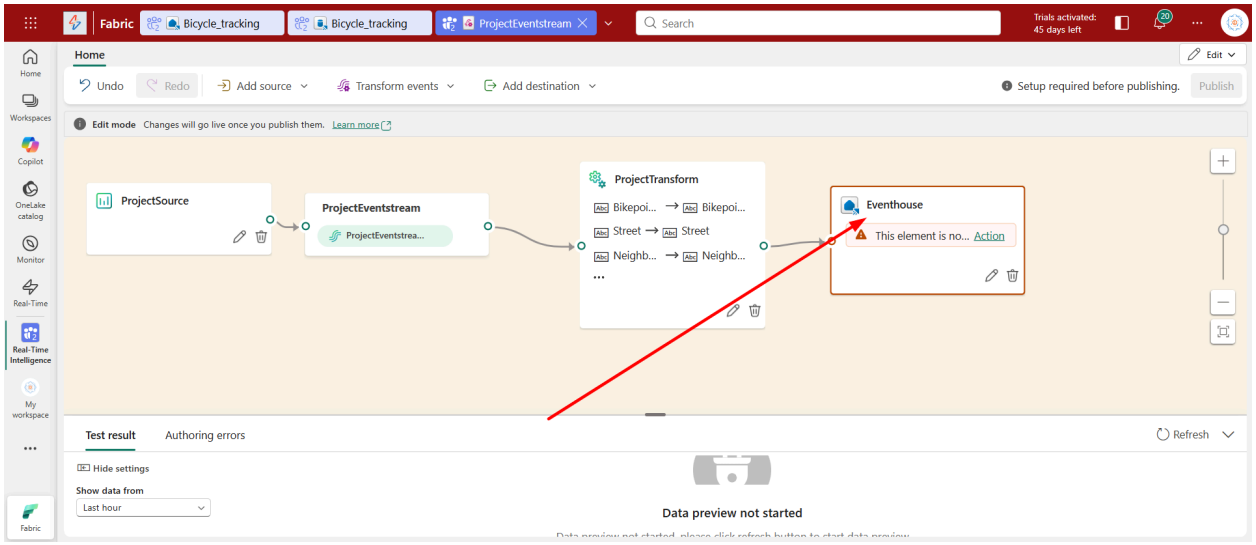
Parameter	Value
Ingestion Mode	Event processing before ingestion
Destination Name	ProjectDestination
Target Database	Project(KQL Database)
Destination Table	RawData (newly created)
Data Format	JSON
Auto-activation	Enabled

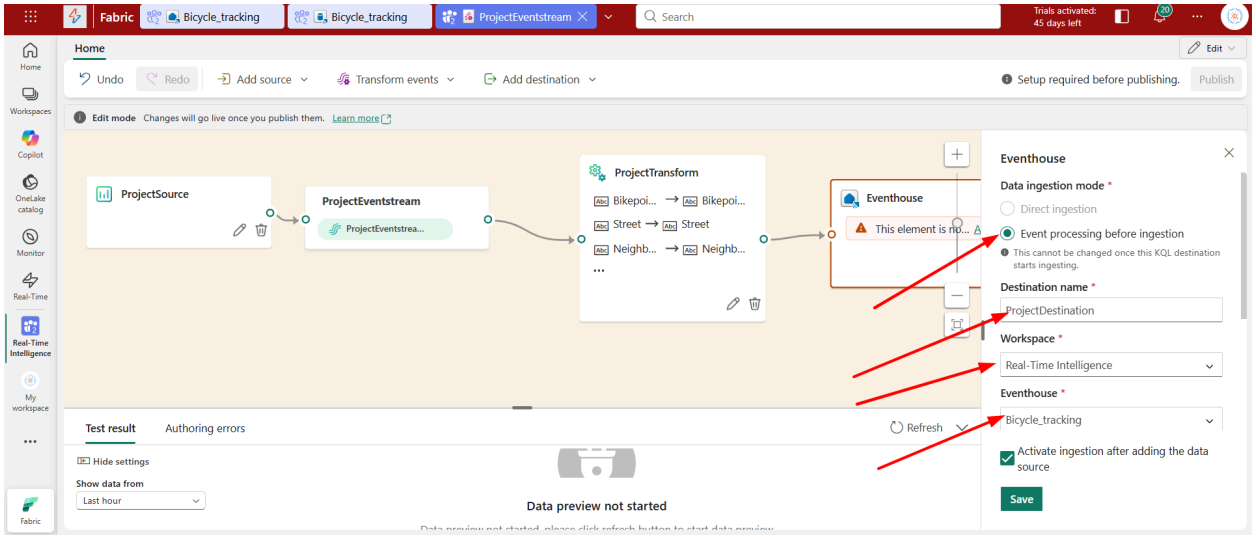
Implementation Process:

1. *Connected transformation output to Eventhouse destination*

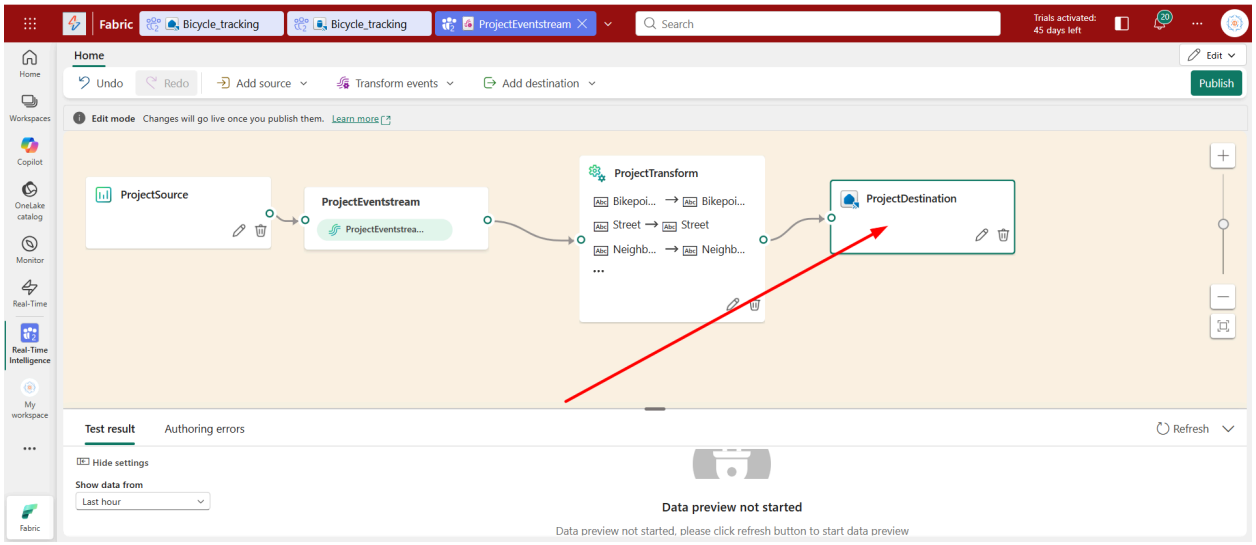


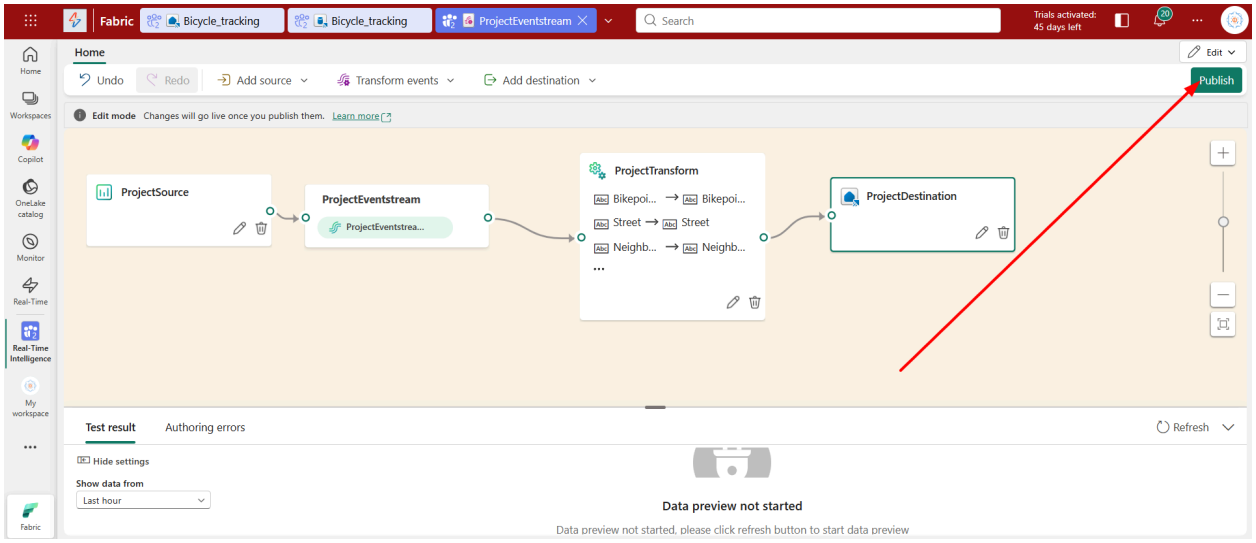
2. *Configured data ingestion parameters for optimal performance*

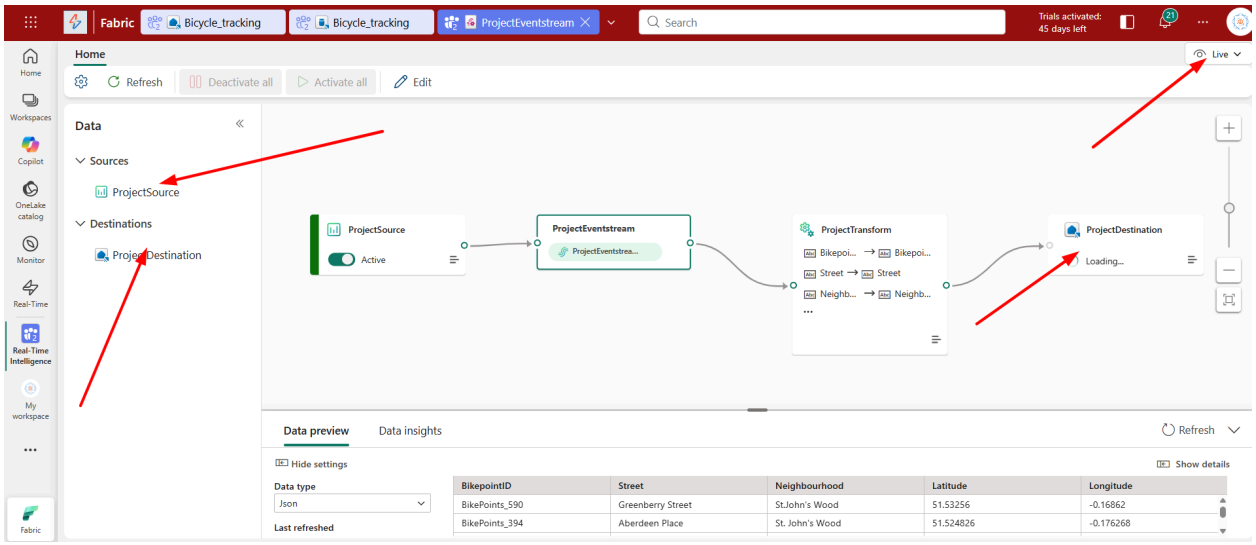




3. *Created destination table schema to accommodate enriched event structure*
4. *Enabled automatic ingestion activation upon deployment*


Fabric
Bicycle_tracking
Bicycle_tracking
ProjectEventstream
Search
Trials activated: 45 days left
Home
Edit
Undo
Redo
Add source
Transform events
Add destination
Publish
Edit mode Changes will go live once you publish them. Learn more
ProjectSource
ProjectEventstream
ProjectEventstrea...
ProjectTransform
Bikepoi... → Bikepoi...
Street → Street
Neighb... → Neighb...
ProjectDestination
Test result
Authoring errors
Refresh
Hide settings
Show data from
Last hour
Data preview not started
Data preview not started, please click refresh button to start data preview


Fabric
Bicycle_tracking
Bicycle_tracking
ProjectEventstream
Search
Trials activated: 45 days left
Home
Edit
Undo
Redo
Add source
Transform events
Add destination
Publish
Edit mode Changes will go live once you publish them. Learn more
ProjectSource
ProjectEventstream
ProjectEventstrea...
ProjectTransform
Bikepoi... → Bikepoi...
Street → Street
Neighb... → Neighb...
ProjectDestination
Test result
Authoring errors
Refresh
Hide settings
Show data from
Last hour
Data preview not started
Data preview not started, please click refresh button to start data preview


Fabric
Bicycle_tracking
Bicycle_tracking
ProjectEventstream
Search
Trials activated: 45 days left
Home
Live
Refresh
Deactivate all
Activate all
Edit
Data
Sources
ProjectSource
Destinations
ProjectDestination
ProjectSource
Active
ProjectEventstream
ProjectEventstrea...
ProjectTransform
Bikepoi... → Bikepoi...
Street → Street
Neighb... → Neighb...
ProjectDestination
Loading...
Data preview
Data insights
Refresh
Hide settings
Show details
Data type
Json
Last refreshed
BikepointID
Street
Neighbourhood
Latitude
Longitude
BikePoints_590
Greenberry Street
St.John's Wood
51.53256
-0.16862
BikePoints_394
Aberdeen Place
St. John's Wood
51.524826
-0.176268

4. Pipeline Deployment

Final Steps:

- Validated complete pipeline configuration
- Published the event stream to production environment
- Verified end-to-end data flow from source through transformation to destination

Technical Architecture

Data Flow:

Source (Bicycle Rentals) → Event Stream → Transformation Layer → Eventhouse Database

(Add Timestamp) (RawData Table)

Key Features Demonstrated

- ***Real-time data ingestion*** *from streaming sources*
- ***Dynamic data transformation*** *using built-in functions*
- ***Schema management*** *for evolving data structures*
- ***Database integration*** *with automatic ingestion*
- ***Pipeline orchestration*** *and deployment*

Results

Successfully implemented a production-ready real-time data pipeline capable of:

- *Capturing streaming events continuously*
- *Enriching data with temporal context*
- *Persisting transformed events to a queryable database*
- *Supporting downstream analytics and reporting requirements*

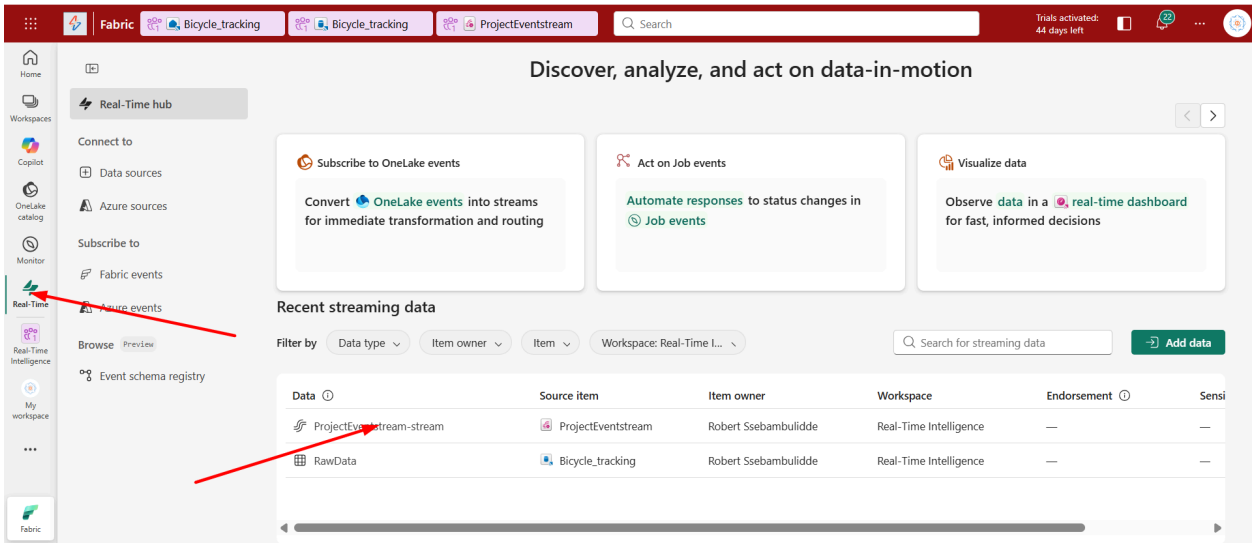
Setting Up Real-Time Alerts for Bike Availability Monitoring

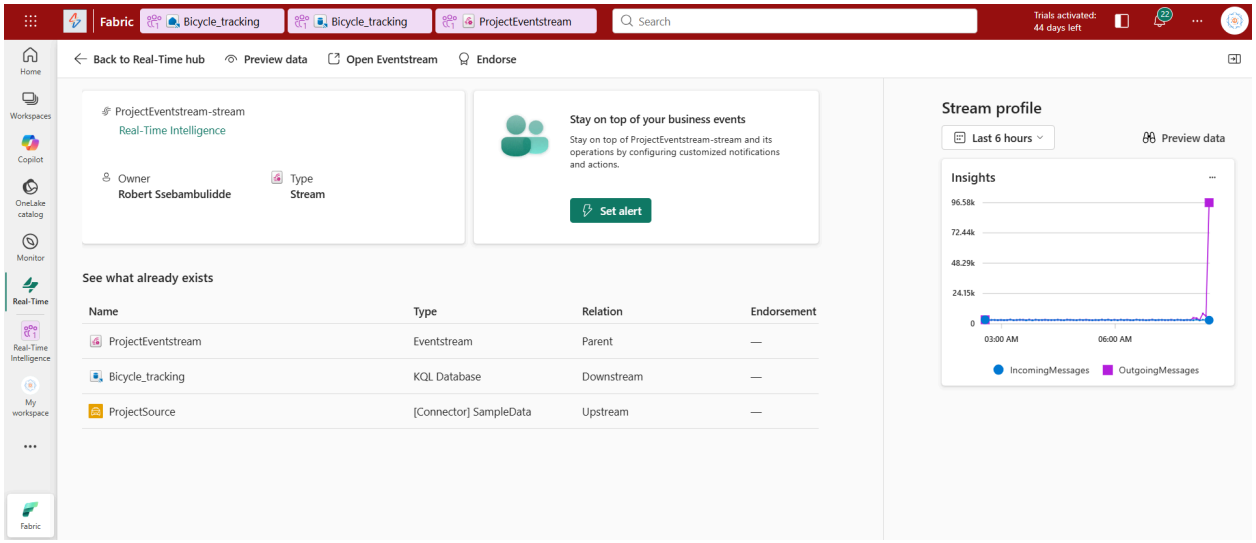
This component of the project implements an intelligent alerting system that monitors bike availability in real-time and sends automated notifications when inventory levels fall below critical thresholds. The system ensures proactive management of bike resources by triggering immediate alerts through Email integration.

Implementation Steps

1. Accessing the Event Stream

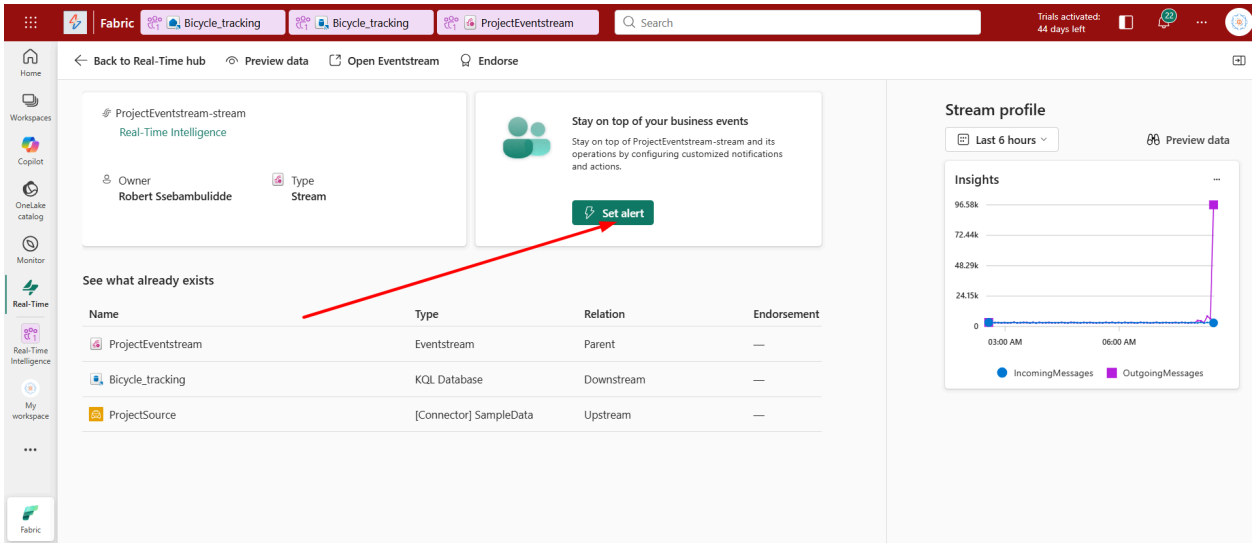
I navigated to the Real-Time section from the application's main navigation panel and selected the previously configured eventstream named ProjectEventstream. This opened the detailed management interface for the stream.

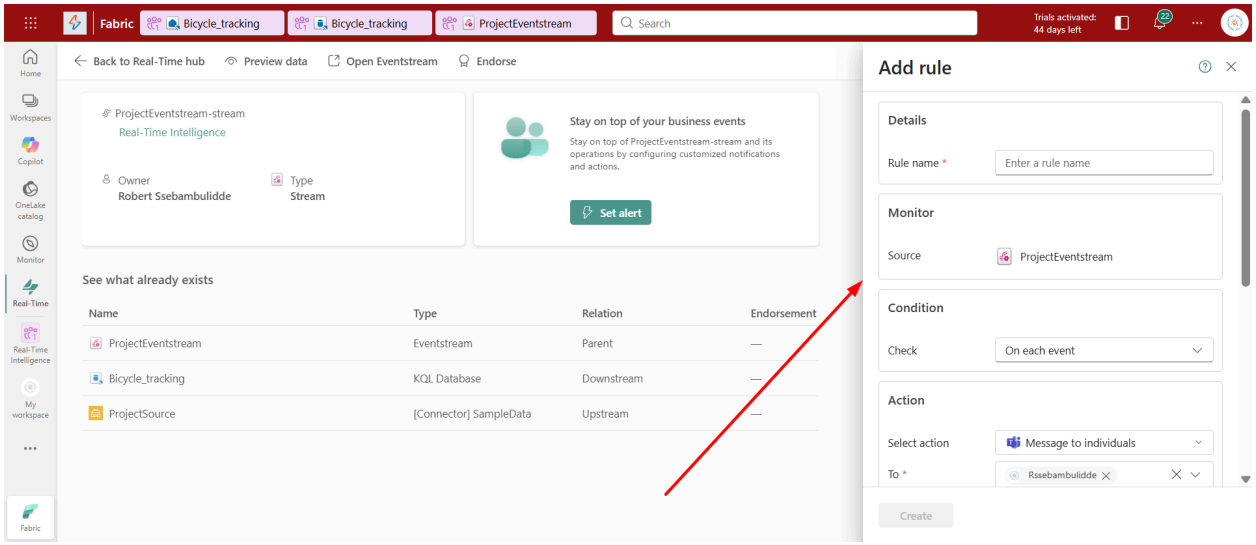




2. Alert Configuration

From the eventstream details page, I initiated the alert setup process by selecting the **Set alert** option, which opened the configuration panel.





3. Alert Rule Definition

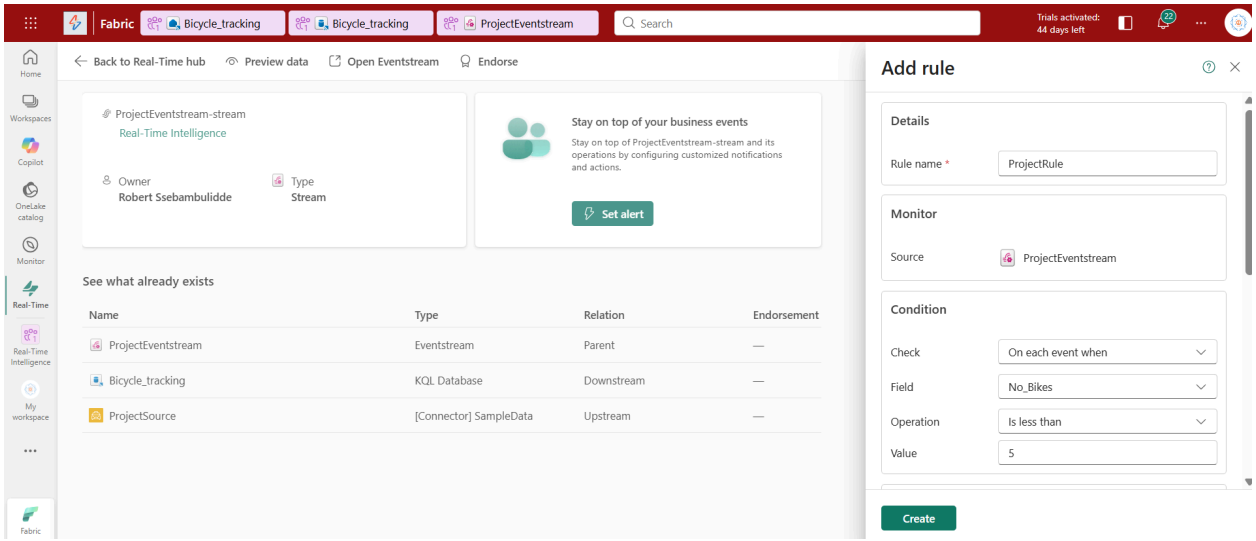
I configured the alert parameters with the following specifications:

Rule Identification

- *Rule Name*: ProjectRule

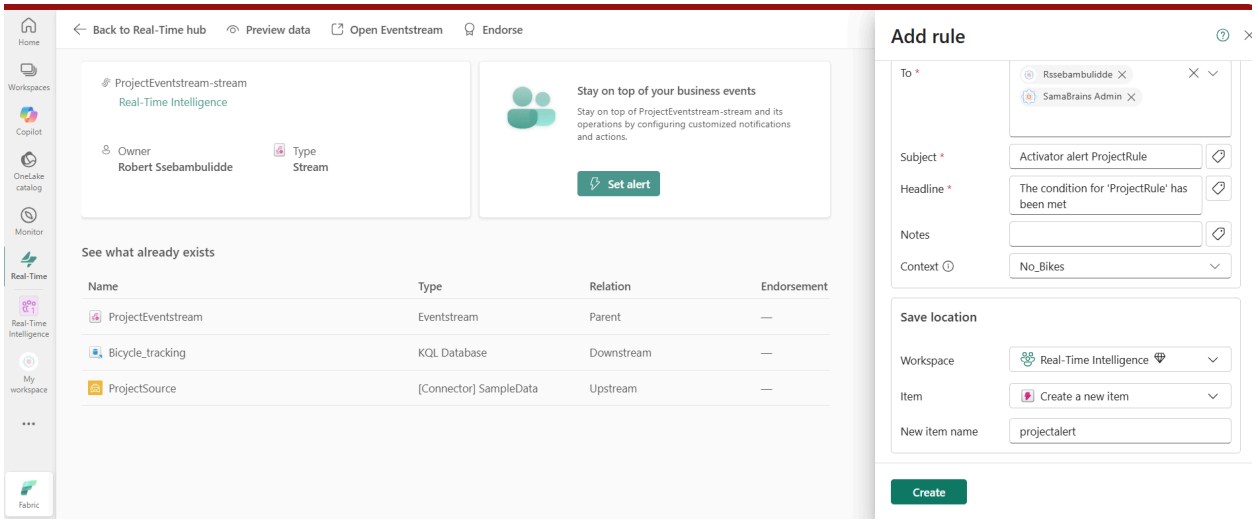
Trigger Conditions

- *Monitoring Frequency: Real-time evaluation (on each event when)*
- *Monitored Field:* No_Bikes
- *Condition Logic: Less than* (<)
- *Threshold Value:* 5



Alert Action

- *Notification Method: Direct message to individuals via Email*
- *Recipients: Email Accounts*
- *Message Headline: "Activator alert"*
- *Message Body: "The condition has been met"*
- *Contextual Data:* No_Bikes *(current bike count included in notification)*

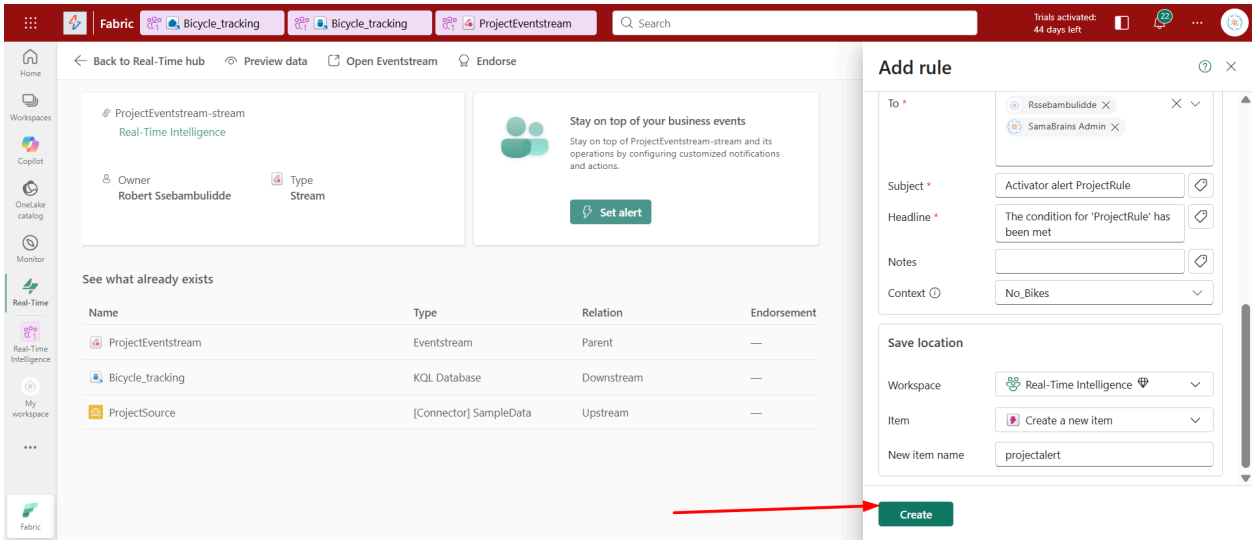


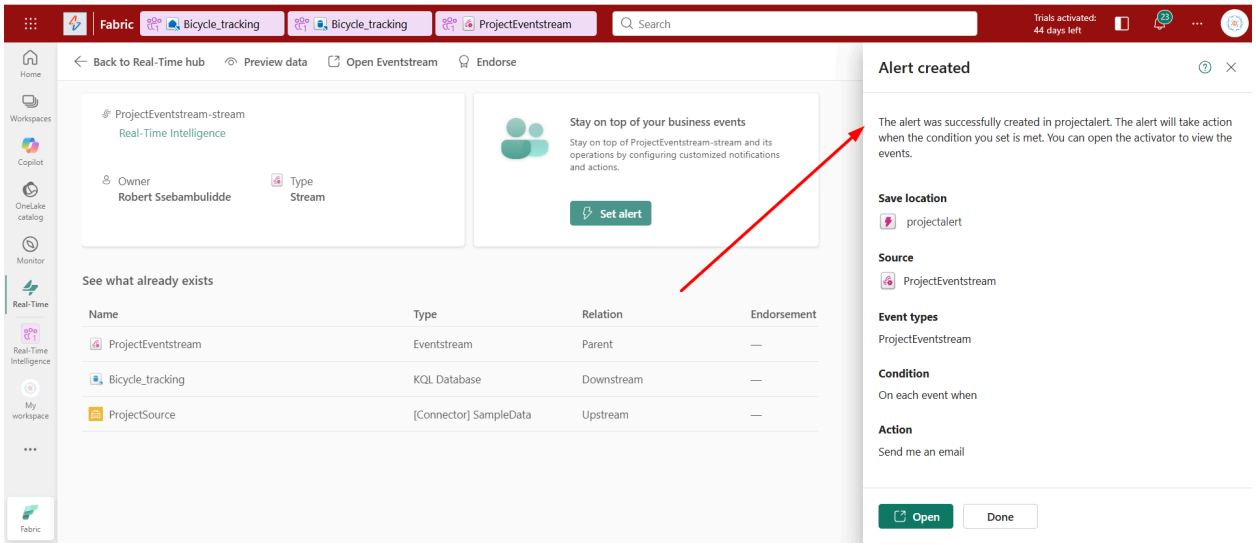
Storage Configuration

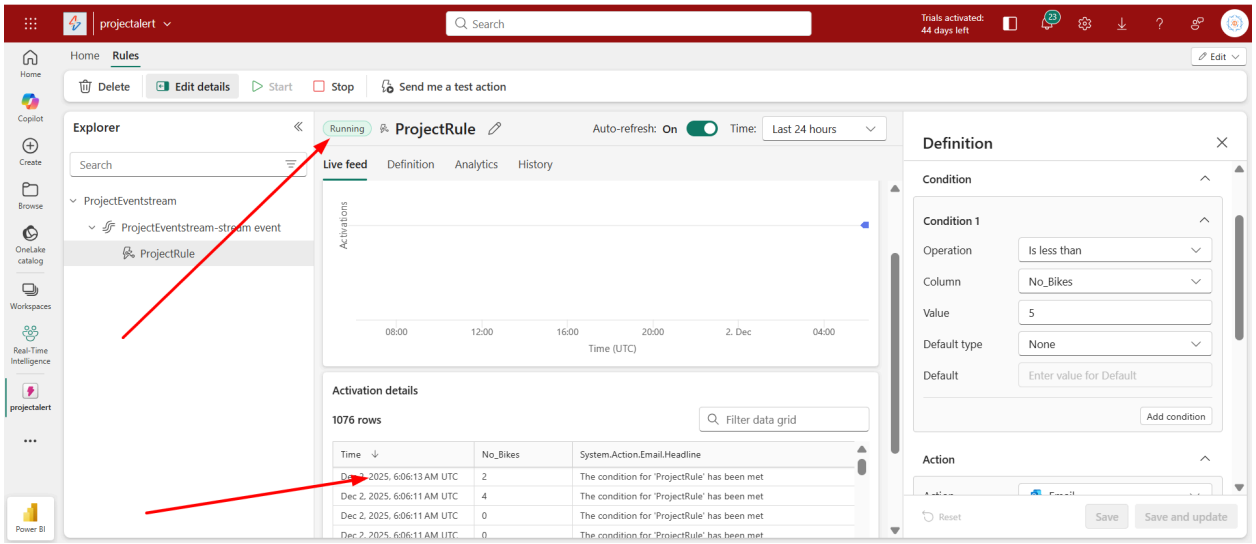
- *Workspace: Selected the project workspace containing all related resources*
- *Item Creation: New item*
- *Item Name*: projectalert

4. Alert Activation

After verifying all configuration parameters, I finalized the setup by selecting **Create**. The alert system is now active and operational.

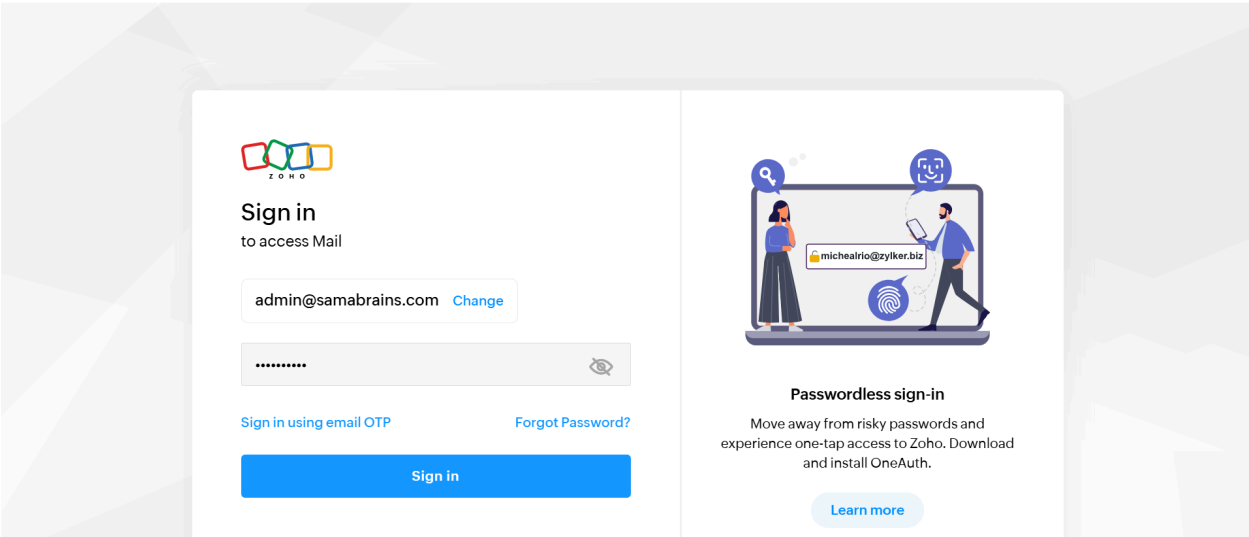

Fabric
Bicycle_tracking
Bicycle_tracking
ProjectEventstream
Search
Trials activated: 44 days left
Back to Real-Time hub
Preview data
Open Eventstream
Endorse
ProjectEventstream-stream
Real-Time Intelligence
Owner
Robert Ssebambulidde
Type
Stream
Stay on top of your business events
Stay on top of ProjectEventstream-stream and its operations by configuring customized notifications and actions.
Set alert
See what already exists
Name
Type
Relation
Endorsement
ProjectEventstream
Eventstream
Parent
Bicycle_tracking
KQL Database
Downstream
ProjectSource
[Connector] SampleData
Upstream
Add rule
To *
Rssebambulidde
SamaBrains Admin
Subject *
Activator alert ProjectRule
Headline *
The condition for 'ProjectRule' has been met
Notes
Context
No_Bikes
Save location
Workspace
Real-Time Intelligence
Item
Create a new item
New item name
projectalert
Create

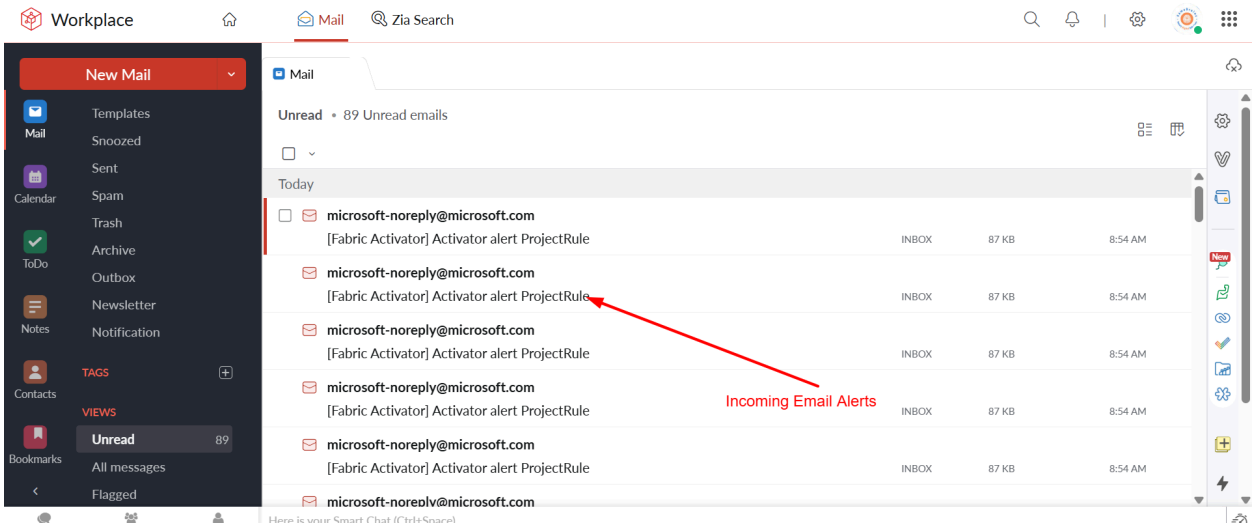

Fabric
Bicycle_tracking
Bicycle_tracking
ProjectEventstream
Search
Trials activated: 44 days left
Back to Real-Time hub
Preview data
Open Eventstream
Endorse
ProjectEventstream-stream
Real-Time Intelligence
Owner
Robert Ssebambulidde
Type
Stream
Stay on top of your business events
Stay on top of ProjectEventstream-stream and its operations by configuring customized notifications and actions.
Set alert
See what already exists
Name
Type
Relation
Endorsement
ProjectEventstream
Eventstream
Parent
Bicycle_tracking
KQL Database
Downstream
ProjectSource
[Connector] SampleData
Upstream
Alert created
The alert was successfully created in projectalert. The alert will take action when the condition you set is met. You can open the activator to view the events.
Save location
projectalert
Source
ProjectEventstream
Event types
ProjectEventstream
Condition
On each event when
Action
Send me an email
Open
Done

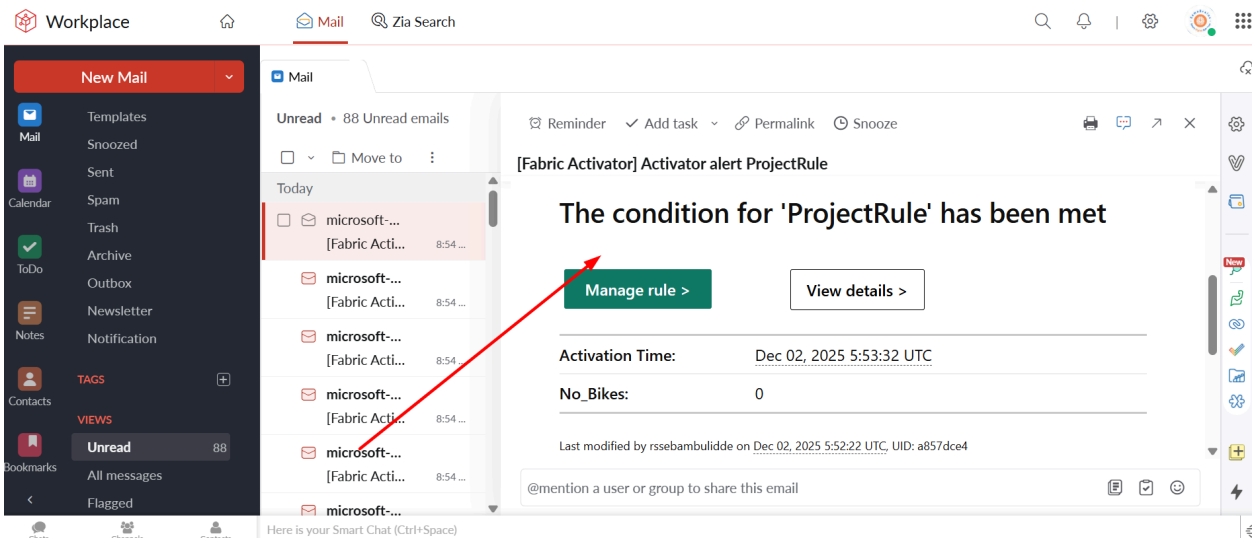

projectalert
Search
Trials activated: 44 days left
Home
Rules
Edit
Delete
Edit details
Start
Stop
Send me a test action
Explorer
Search
ProjectEventstream
ProjectEventstream-stream event
ProjectRule
Running
ProjectRule
Auto-refresh: On
Time:
Last 24 hours
Live feed
Definition
Analytics
History
Activations
08:00
12:00
16:00
20:00
2. Dec
04:00
Time (UTC)
Activation details
1076 rows
Filter data grid
Time
No_Bikes
System.Action.Email.Headline
Dec 2, 2025, 6:06:13 AM UTC
2
The condition for 'ProjectRule' has been met
Dec 2, 2025, 6:06:11 AM UTC
4
The condition for 'ProjectRule' has been met
Dec 2, 2025, 6:06:11 AM UTC
0
The condition for 'ProjectRule' has been met
Definition
Condition
Condition 1
Operation
Is less than
Column
No_Bikes
Value
5
Default type
None
Default
Enter value for Default
Add condition
Action
Reset
Save
Save and update

Functionality

The implemented alert system continuously monitors the bike availability data stream. When the number of available bikes drops below 5 units, the system automatically generates and sends a notification through Email, enabling immediate response to low inventory situations.







Technical Outcome

Successfully established an automated, event-driven alerting mechanism that provides real-time visibility into critical resource levels and facilitates rapid intervention when predefined thresholds are breached.

Data Transformation Implementation in KQL Database

This demonstrates the implementation of an automated data transformation pipeline using update policies in a KQL (Kusto Query Language) Database. The solution transforms raw bike-sharing data into a structured, analytics-ready format without requiring manual orchestration.

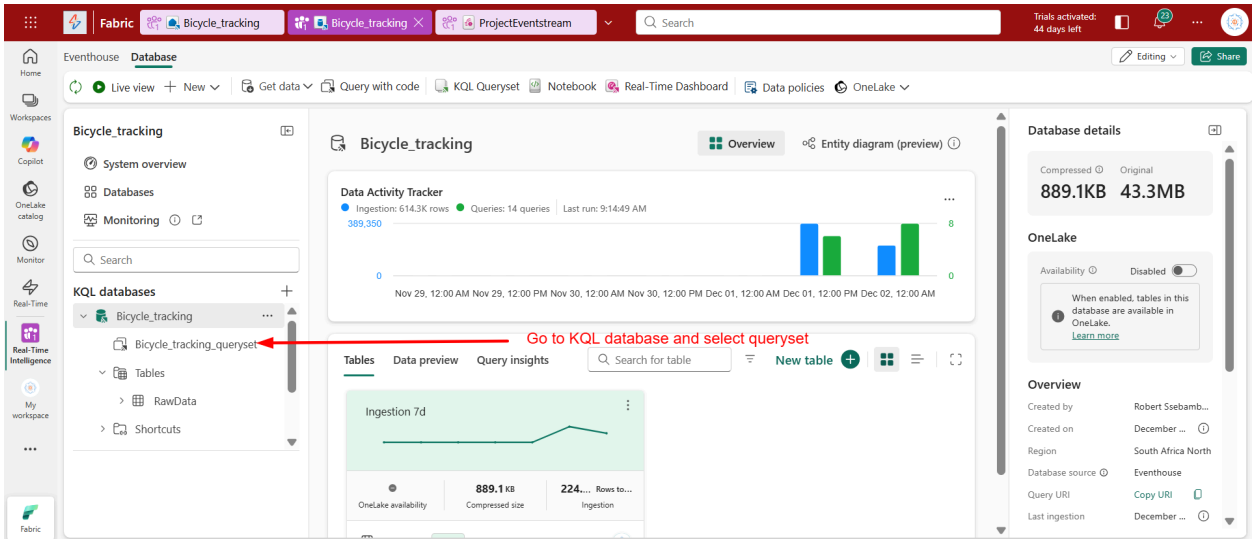
- *Implemented automated data transformation using update policies*
- *Organized data using a medallion architecture (Bronze & Silver layers)*
- *Parsed and enriched raw data with calculated fields*
- *Created a scalable transformation pipeline for real-time data processing*

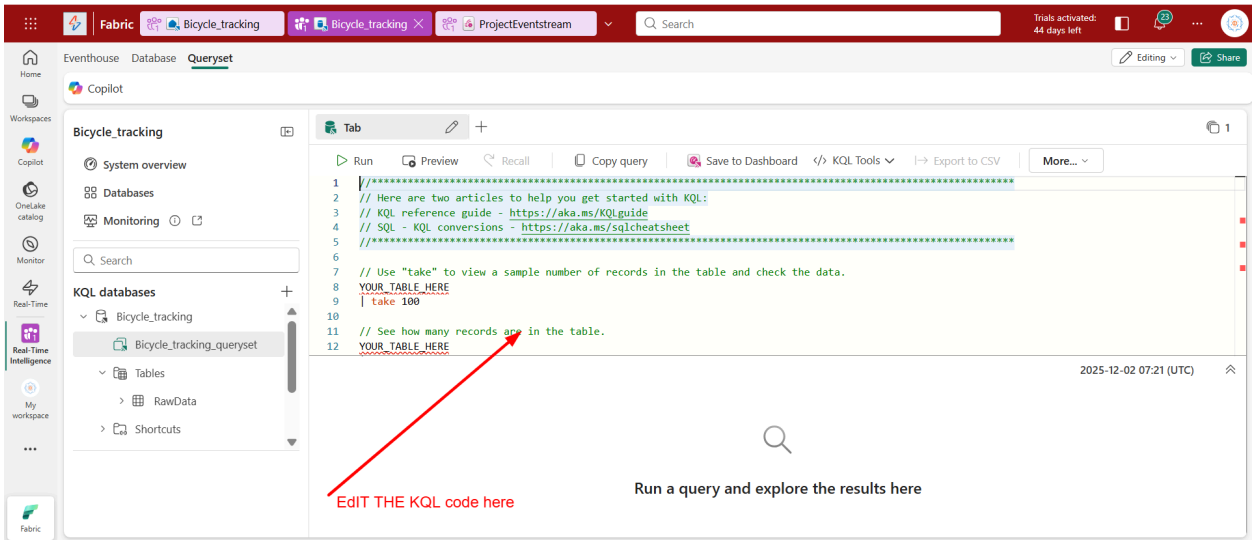
Implementation Steps

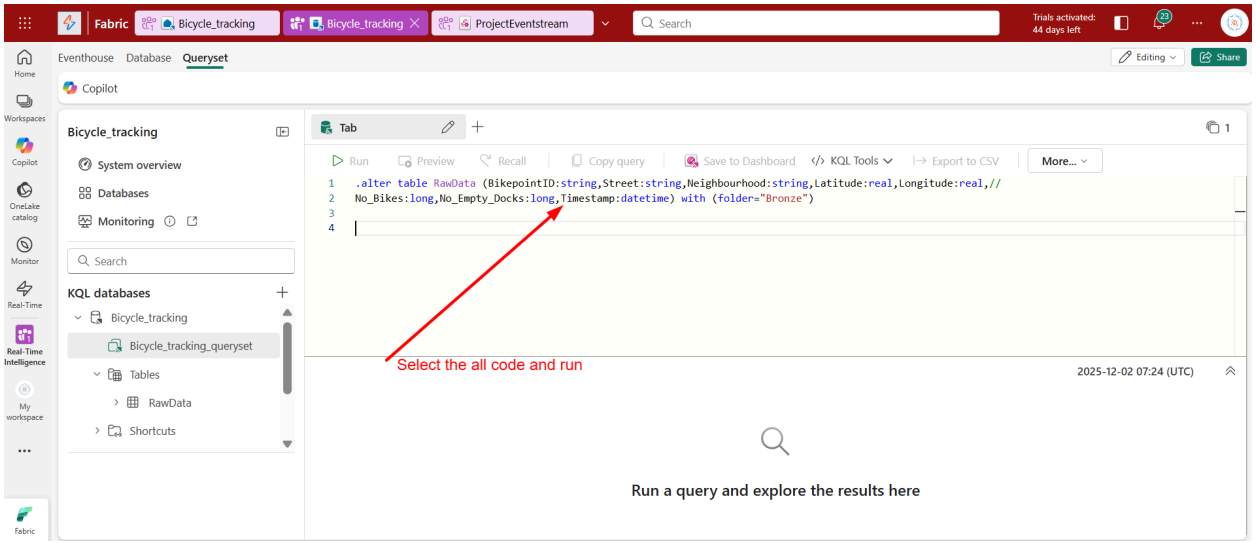
1. Data Organization - Bronze Layer Setup

The raw data table was organized into a Bronze folder to establish a clear data lake architecture pattern.

Implementation:

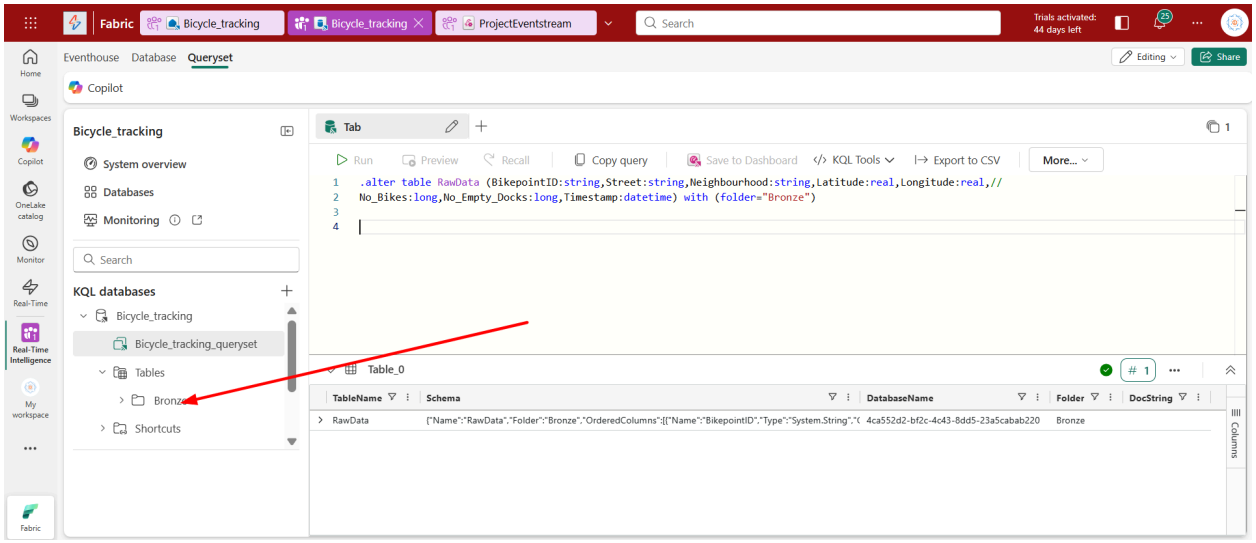







```
.alter table RawData (BikepointID:string,Street:string,Neighbourhood:string,Latitude:real,Longitude:real,No_Bikes:long,No_Empty_Docks:long,Timestamp:datetime) with (folder="Bronze")
```

This established the Bronze layer as the landing zone for raw, unprocessed data, maintaining the original data structure for audit and reprocessing capabilities.

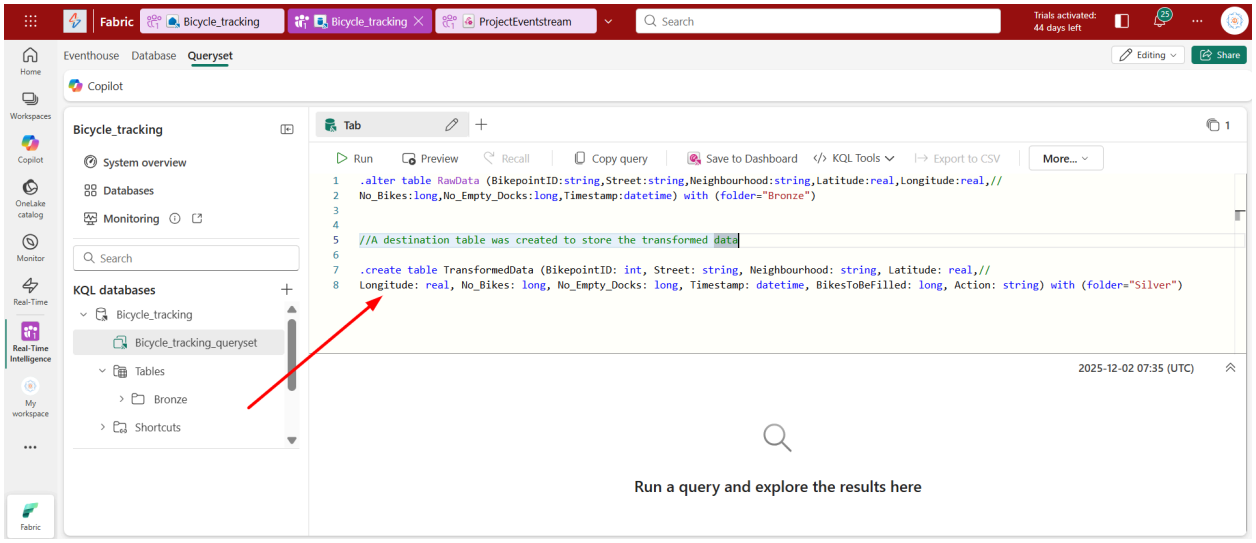


---

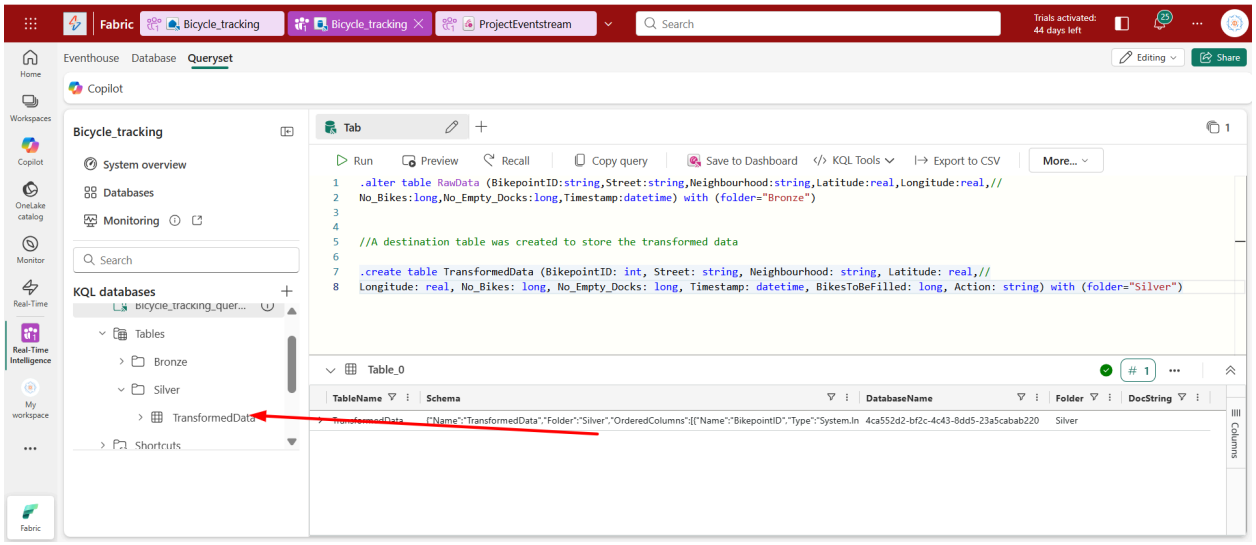
2. Target Table Creation - Silver Layer

A destination table was created to store the transformed data with an enhanced schema including calculated fields.

Implementation:




```
.create table TransformedData (BikepointID: int, Street: string, Neighbourhood: string, Latitude: real, Longitude: real, No_Bikes: long, No_Empty_Docks: long, Timestamp: datetime, BikesToBeFilled: long, Action: string) with (folder="Silver")
```



Schema Enhancements:

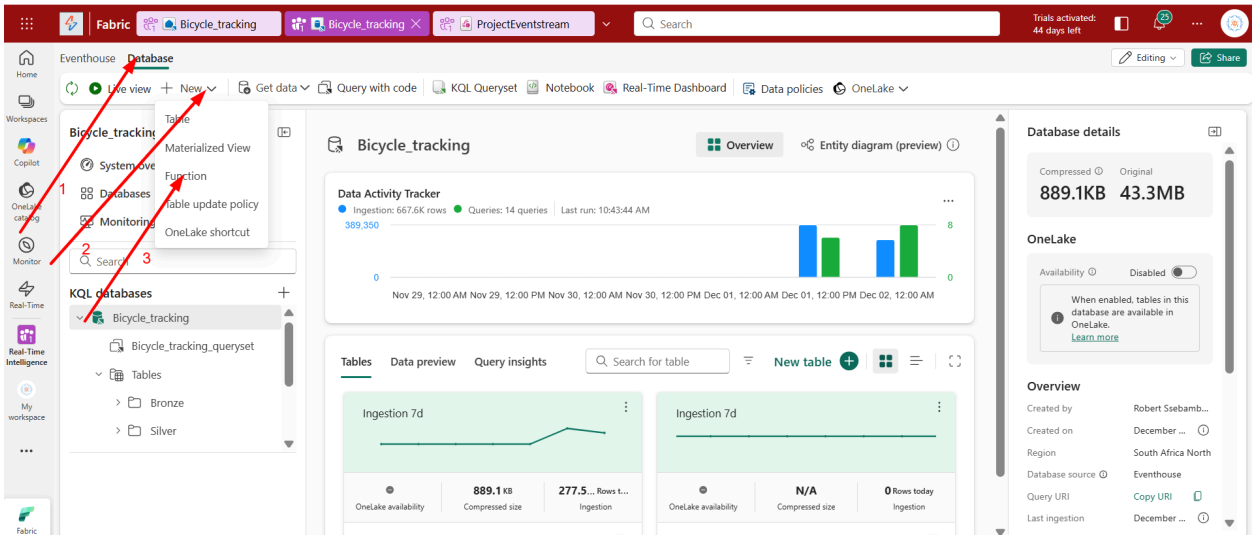
- *BikepointID: Changed from string to integer for better query performance*
- *BikesToBeFilled: New calculated field for capacity analysis*
- *Action: New field for operational recommendations*

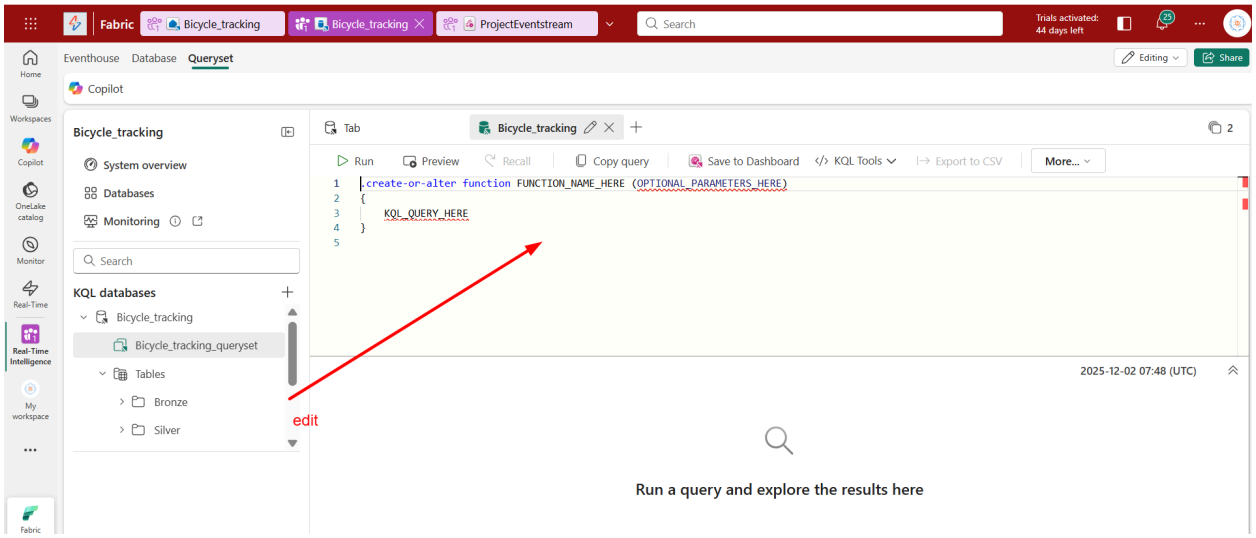
---

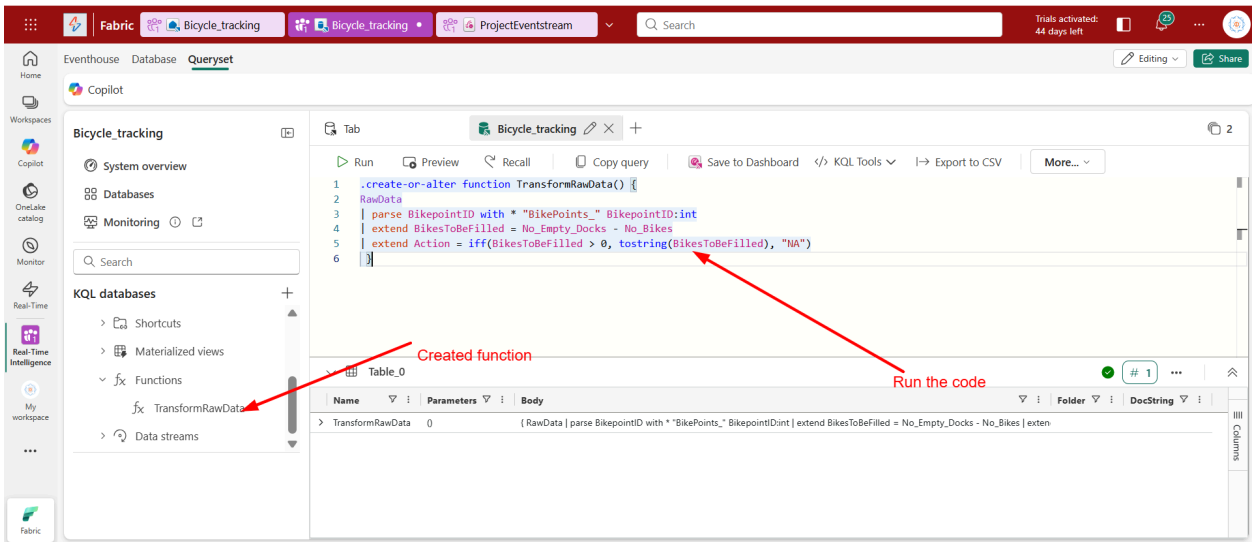
3. Transformation Logic - Function Development

A stored function was created to encapsulate the transformation logic, making it reusable and maintainable.

Implementation:








```
.create-or-alter function TransformRawData() {
RawData
| parse BikepointID with * "BikePoints_" BikepointID:int
| extend BikesToBeFilled = No_Empty_Docks - No_Bikes
| extend Action = iff(BikesToBeFilled > 0, tostring(BikesToBeFilled), "NA")
 }
```

Transformation Logic:

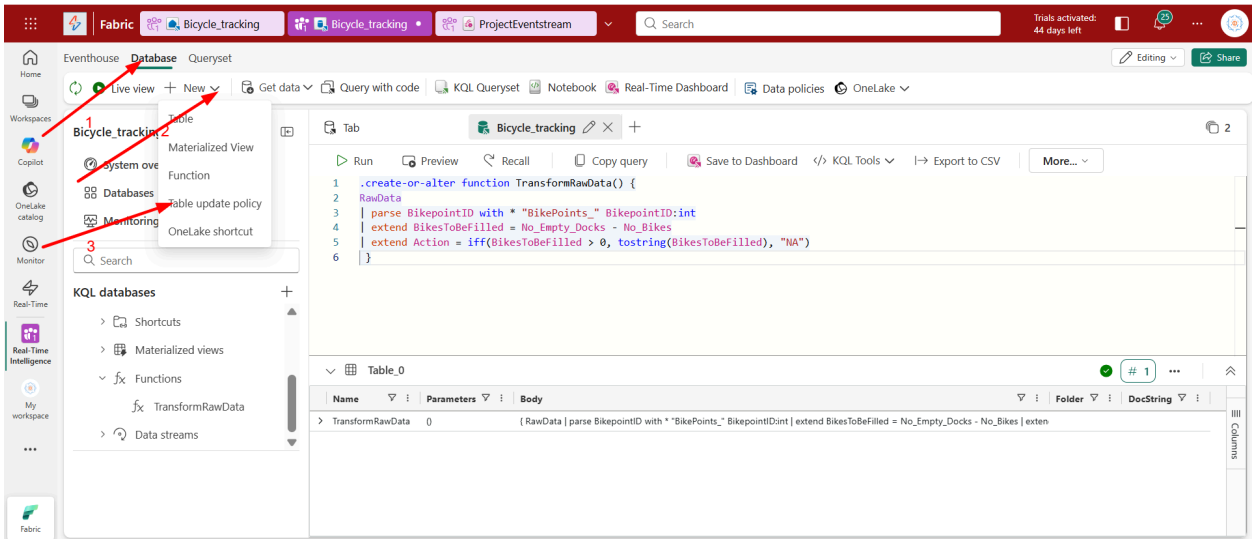
- ***Data Parsing:*** *Extracts numeric ID from the "BikePoints_" prefixed string*
- ***Capacity Calculation:*** *Computes available capacity (empty docks minus current bikes)*

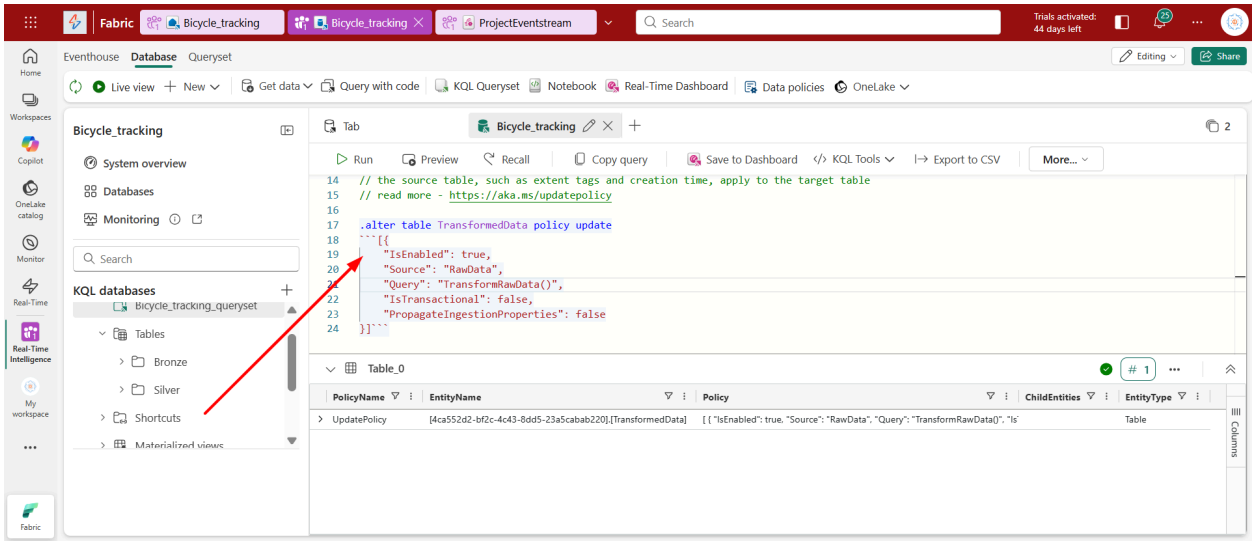
- ***Action Recommendation:*** *Generates actionable insights based on capacity analysis*

4. Update Policy Configuration

An update policy was applied to automate the transformation process whenever new data arrives in the source table.

Implementation:






```
.alter table TransformedData policy update
```

```
```[{
   "IsEnabled": true,
   "Source": "RawData",
   "Query": "TransformRawData()",
   "IsTransactional": false,
   "PropagateIngestionProperties": false
}]```
```

**Configuration Details:**

- ***IsEnabled:*** *Activates the policy for automatic execution*
- ***Source:*** *Links to the RawData table as the trigger*
- ***Query:*** *References the transformation function*
- ***IsTransactional:*** *Set to false for better performance in streaming scenarios*
- ***PropagateIngestionProperties:*** *Disabled to prevent metadata propagation*

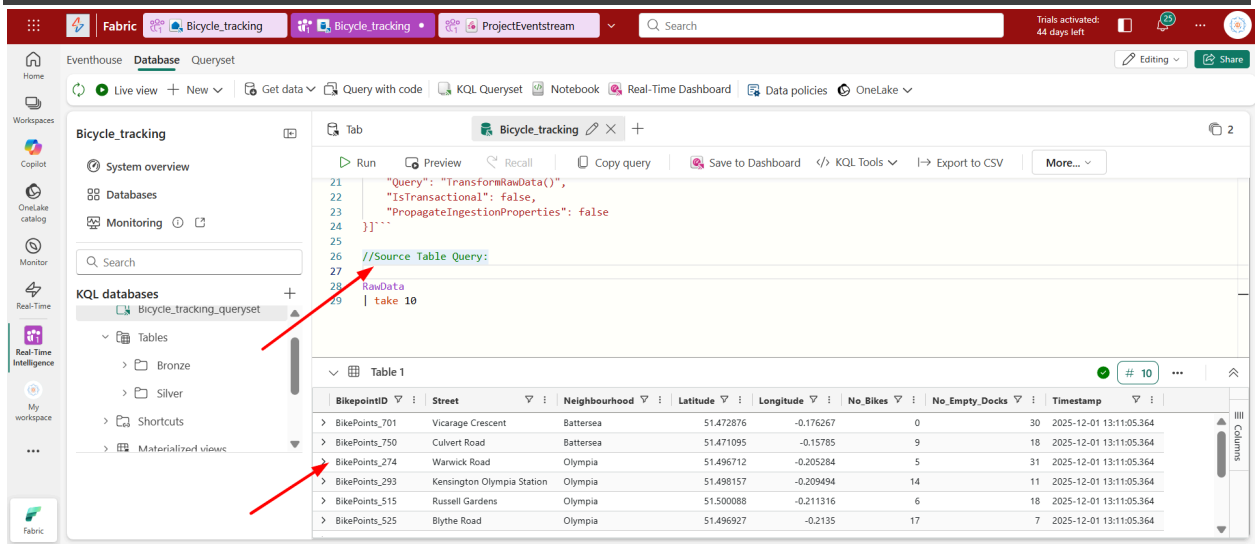
---

## 5. Validation and Verification

The transformation was validated by comparing sample records from both source and target tables.

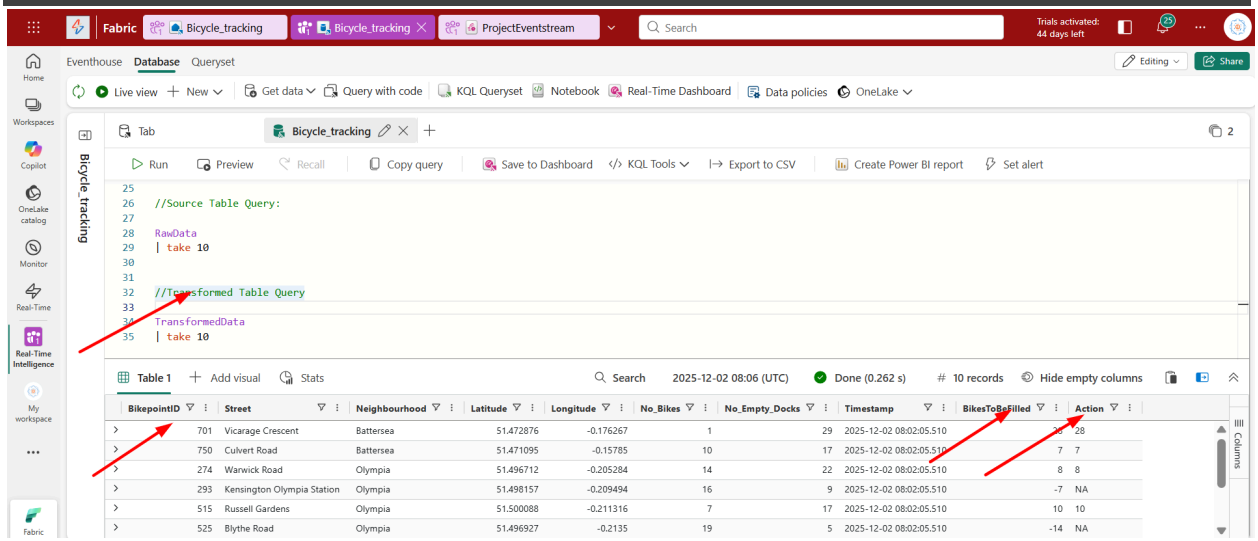
**Source Table Query:**

```
RawData
| take 10
```


```

Transformed Table Query:

```
TransformedData
| take 10
```



Validation Results:

- *BikepointID prefix successfully removed and converted to integer*
- *Calculated fields (BikesToBeFilled, Action) populated correctly*
- *All source data fields preserved in target table*
- *Data transformation occurs automatically for new ingested records*

Technical Achievements

- ***Automated Pipeline:*** *Eliminated manual data transformation processes*
- ***Scalable Architecture:*** *Medallion architecture supports multiple transformation layers*
- ***Performance Optimization:*** *Non-transactional update policy enables high-throughput processing*
- ***Data Quality:*** *Structured validation ensures transformation accuracy*
- ***Maintainability:*** *Encapsulated transformation logic in reusable functions*

Key Insights

The update policy mechanism provides a powerful automation framework for real-time data transformation. By separating raw and transformed data into different layers, the solution maintains data lineage while enabling multiple downstream consumers with different schema requirements. The function-based approach ensures consistency and simplifies maintenance of transformation logic.

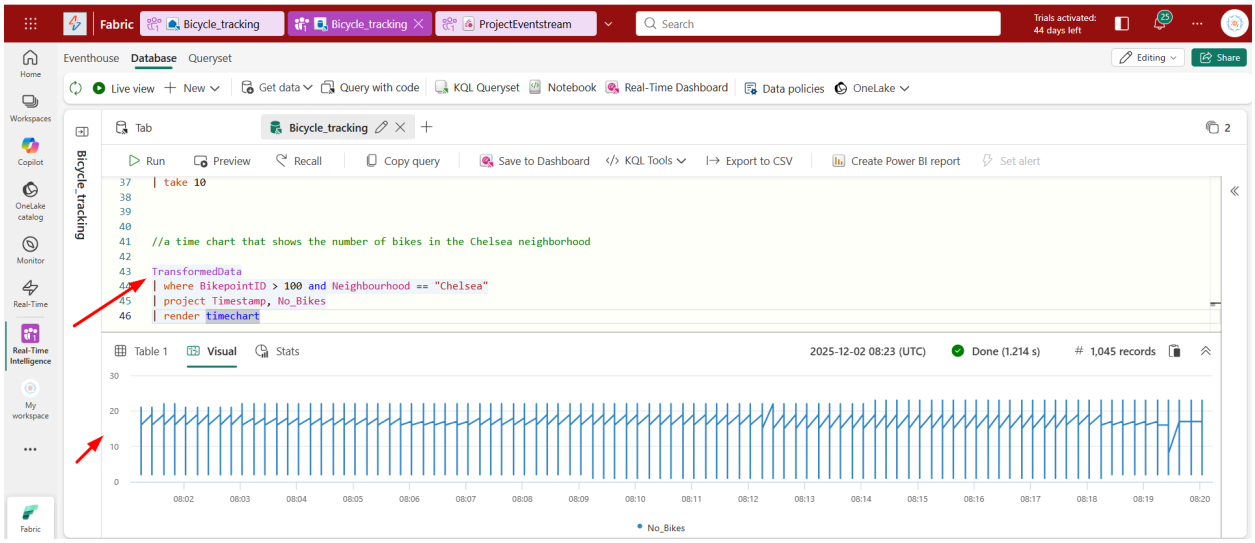
Query Streaming Data Using KQL

This demonstrates advanced querying techniques applied to streaming bike-sharing data. Showcases proficiency in Kusto Query Language (KQL), query optimization through materialized views, cross-language query translation.

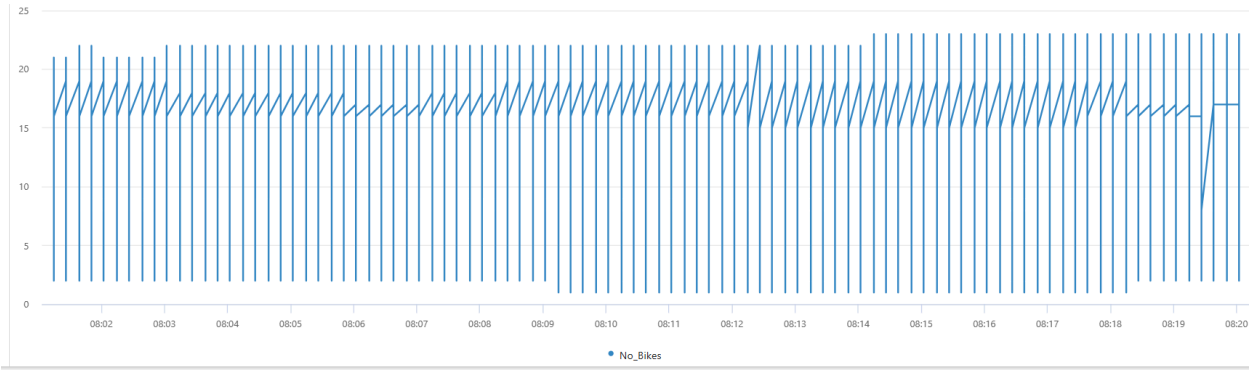
1.Time-Series Data Visualization

Analyzed bike availability patterns in the Chelsea neighborhood using time-series visualization.

Implementation




```
TransformedData
| where BikepointID > 100 and Neighbourhood == "Chelsea"
| project Timestamp, No_Bikes
| render timechart
```



Key Features

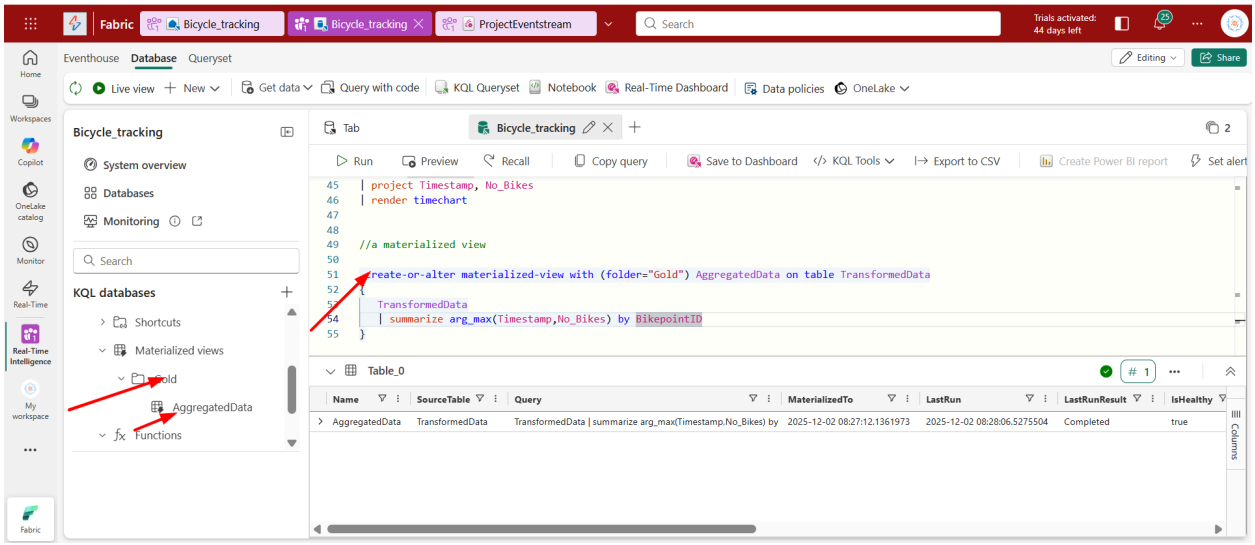
- *Filters data for specific geographic area (Chelsea) and bike point criteria*
- *Projects only relevant fields for analysis*
- *Renders results as an interactive time chart showing bike availability trends over time*

---

2. Materialized View for Real-Time Aggregation

Optimized query performance by pre-aggregating the most recent bike availability data for each station.

Materialized View Creation



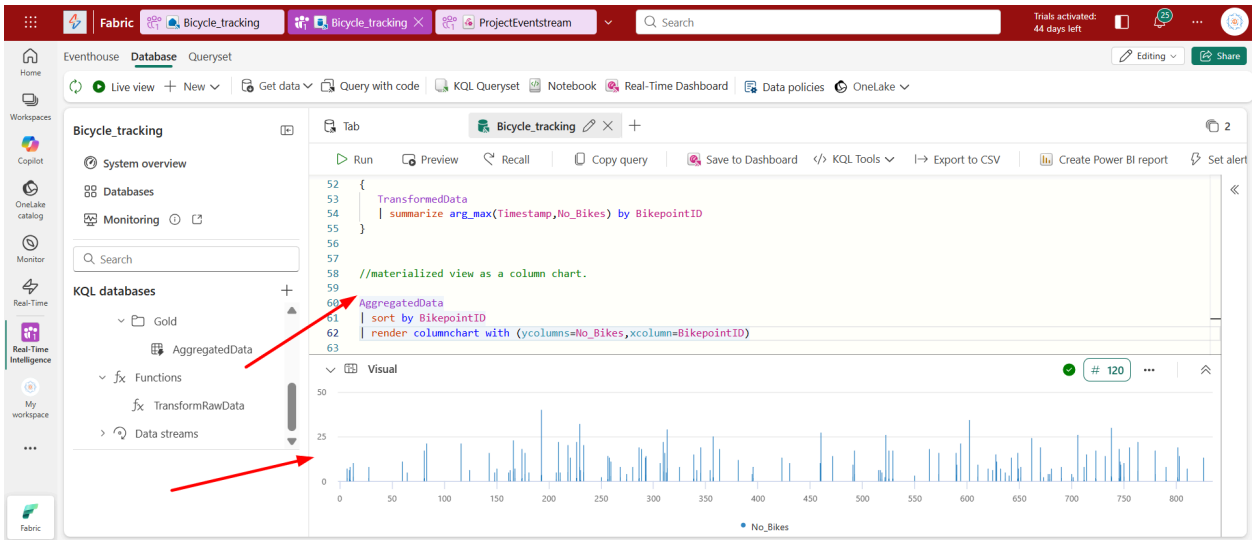

```
.create-or-alter materialized-view with (folder="Gold") AggregatedData on table TransformedData
```

```
{
  TransformedData
  | summarize arg_max(Timestamp, No_Bikes) by BikepointID
}
```

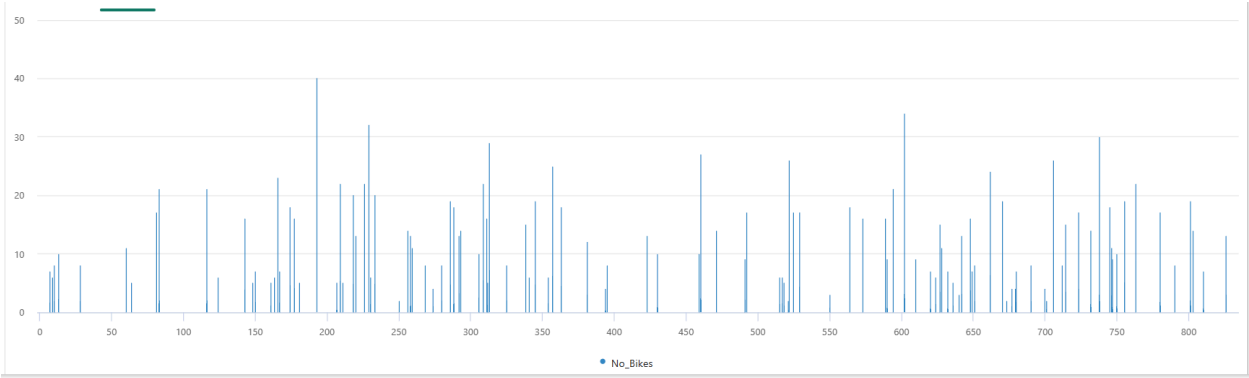
Technical Details

- ***View Name**: AggregatedData*
- ***Storage Folder**: Gold (follows data lake organization principles)*
- ***Aggregation Logic**: Captures the latest bike count per station using arg_max()*
- ***Performance Benefit**: Eliminates need to scan full dataset for current state queries*

Visualization Query




```
AggregatedData
| sort by BikepointID
| render columnchart with (ycolumns=No_Bikes, xcolumn=BikepointID)
```



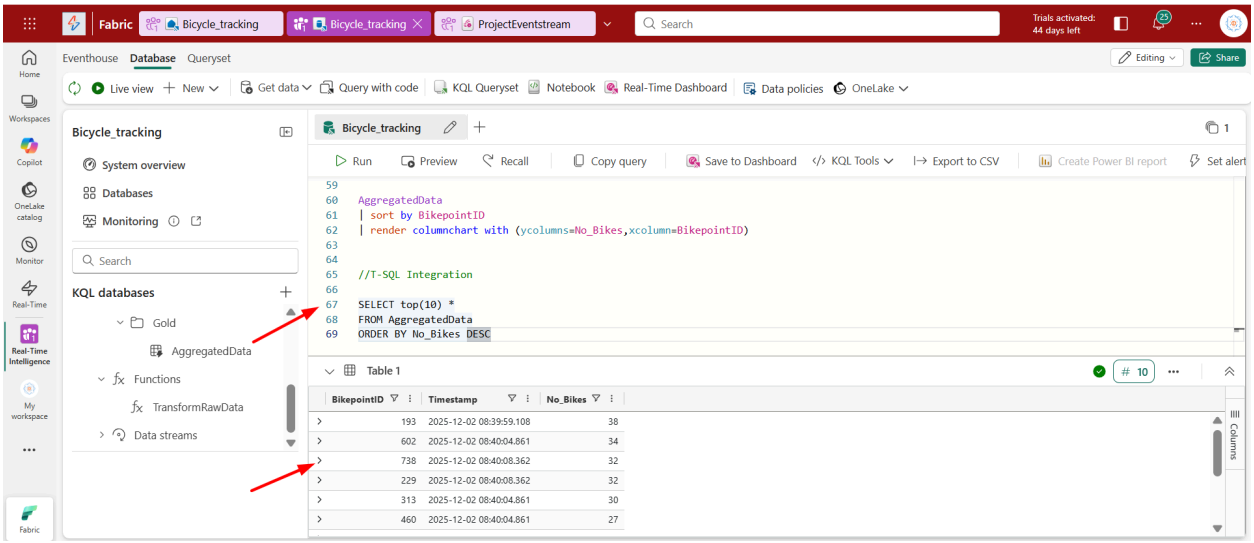
This query generates a column chart displaying current bike availability across all stations, sorted by station ID.

3. Cross-Language Query Support: T-SQL Integration

Demonstrate interoperability between SQL and KQL within the same analytics environment.

T-SQL Query Implementation

```
SELECT top(10) *
FROM AggregatedData
ORDER BY No_Bikes DESC
```



Retrieves the top 10 bike stations with highest bike availability, demonstrating that traditional SQL syntax is supported for users familiar with relational database querying.

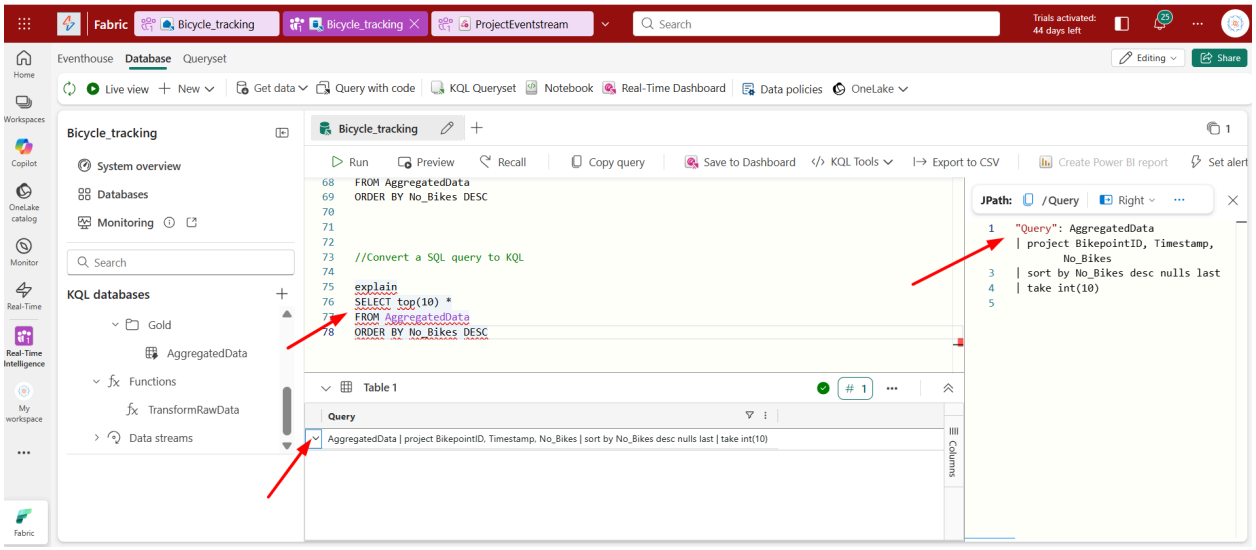
---

4. Query Translation: SQL to KQL Conversion

Understand KQL equivalents of SQL queries for optimization and learning purposes.

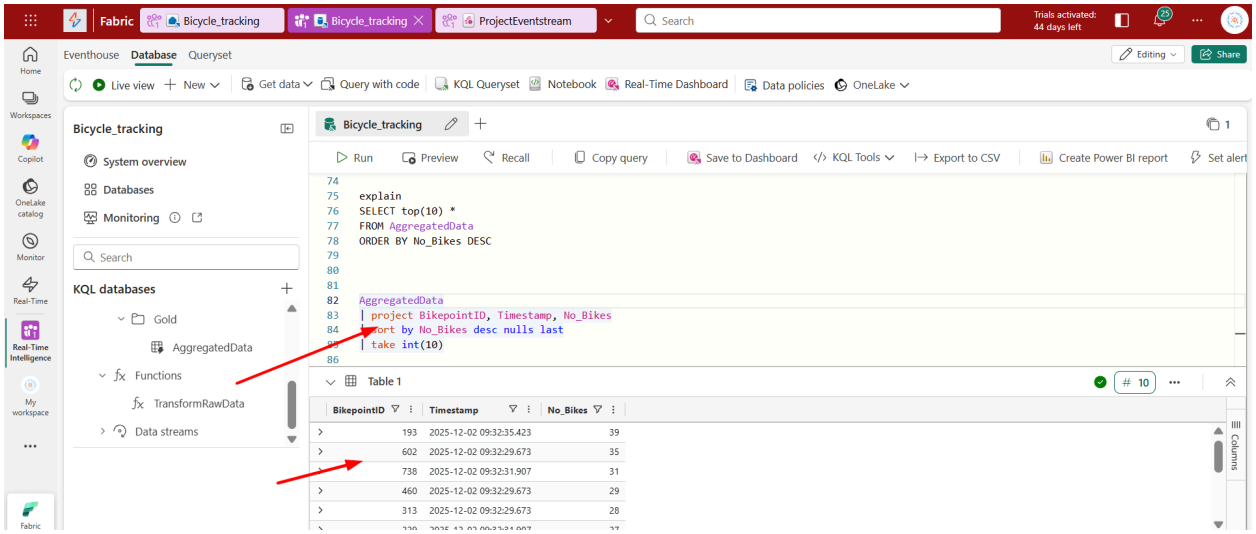
Translation Method

```
explain
SELECT top(10) *
FROM AggregatedData
ORDER BY No_Bikes DESC
```



Process

1. *Prefix any T-SQL query with explain keyword*
2. *Execute the query*
3. *Review the KQL equivalent in the output pane*
4. *Copy and use the generated KQL for further optimization*



Learning Value

This approach bridges the knowledge gap between SQL and KQL, helping developers understand KQL syntax patterns and idiomatic query construction.

Project Skills Demonstrated

- ***KQL Query Writing**: Filtering, projection, and rendering techniques*
- ***Performance Optimization**: Materialized view design for real-time analytics*
- ***Cross-Platform Querying**: T-SQL integration in KQL environments*
- ***Query Translation**: Understanding language equivalencies for optimization*
- ***Data Visualization**: Creating meaningful time-series and distribution charts*

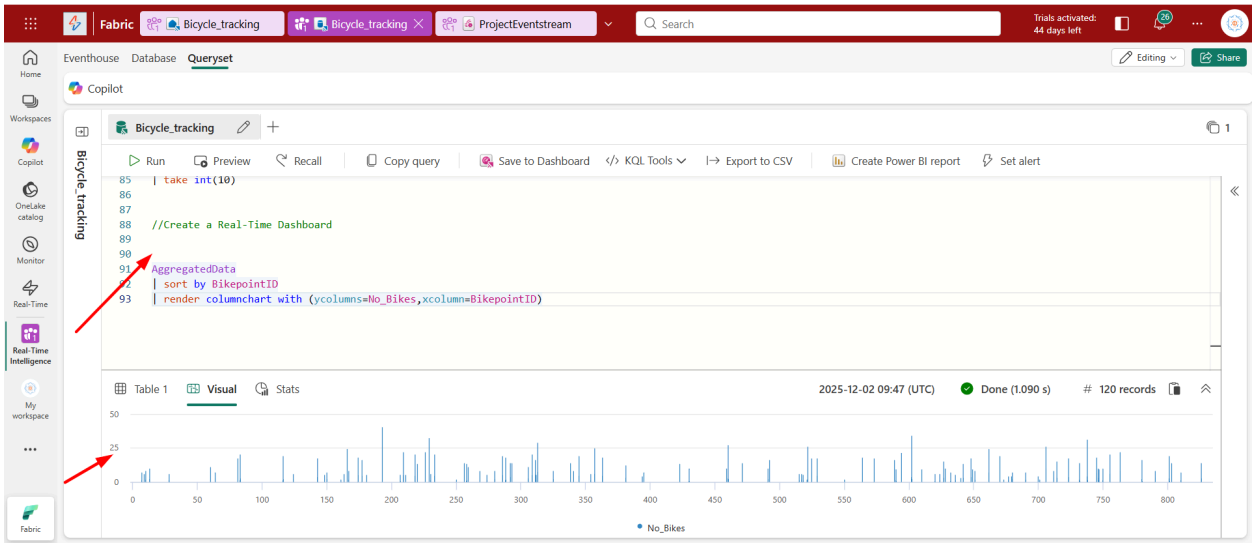
Real-Time Dashboard Implementation for Streaming Data Visualization

This section documents the implementation of an interactive Real-Time Dashboard to visualize streaming bicycle availability data. The dashboard provides real-time insights through multiple visual representations, including column charts and geographic maps.

Implementation Steps

1. Dashboard Query Development

The foundation of the dashboard relies on a KQL query that aggregates and visualizes bicycle availability across different locations:




```
AggregatedData
| sort by BikepointID
| render columnchart with (ycolumns=No_Bikes, xcolumn=BikepointID)
```

Query Logic:

- *Retrieves data from the AggregatedData table*
- *Sorts records by BikepointID to ensure consistent visualization*
- *Renders a column chart displaying the number of available bikes (No_Bikes) for each bike station (BikepointID)*

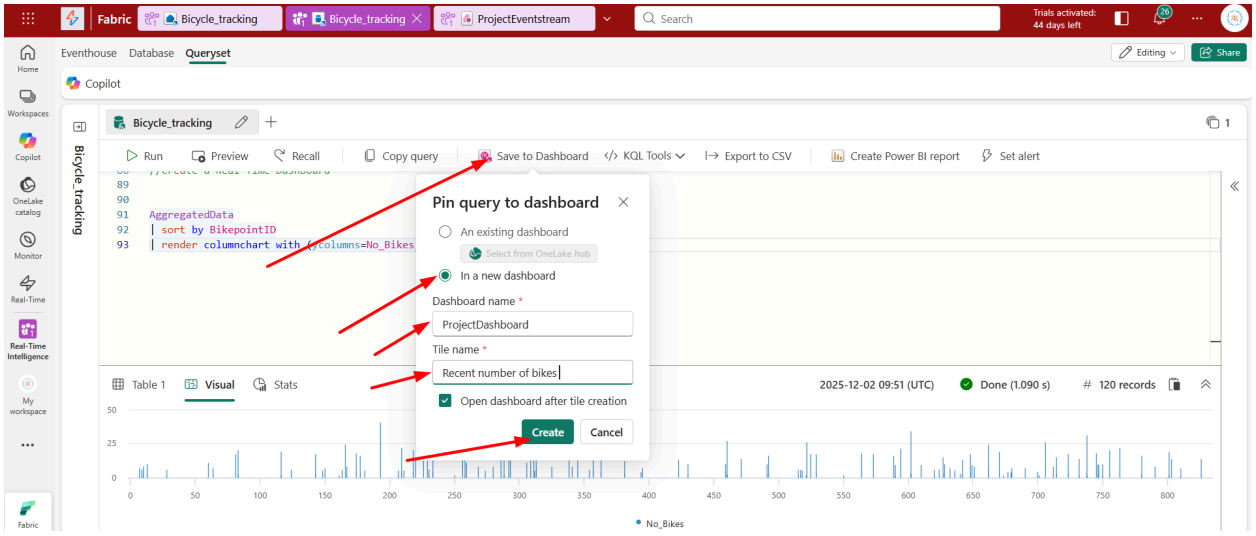
2. Dashboard Creation

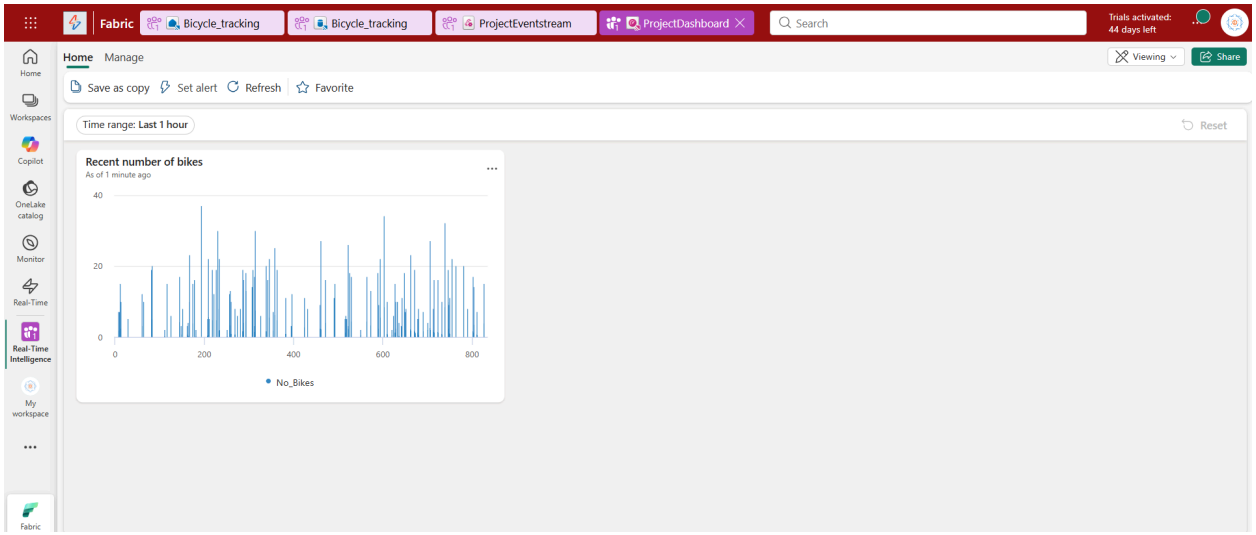
Configuration Parameters:

Parameter	Value	Purpose
Name	ProjectDashboard	Unique identifier for the dashboard
Location	Current workspace	Defines where the dashboard resource is stored

Creation Process:

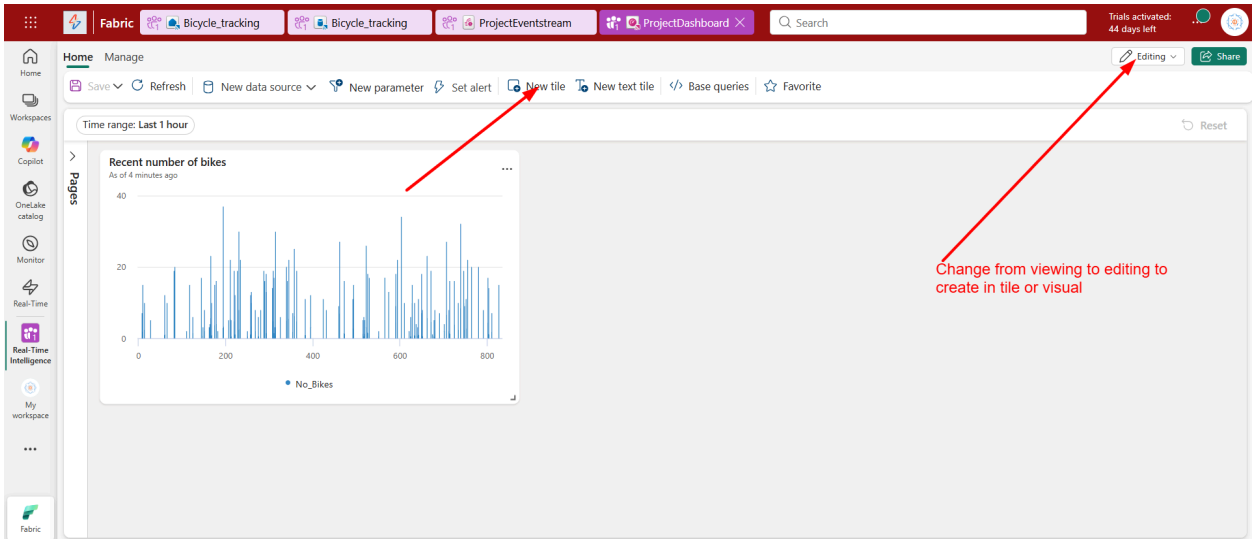
1. *Execute the visualization query in the KQL queryset*
2. *Select "Save to dashboard"*
3. *Configure the dashboard name and Tile name*
4. *Confirm creation to initialize the dashboard*





3. Geographic Visualization Implementation

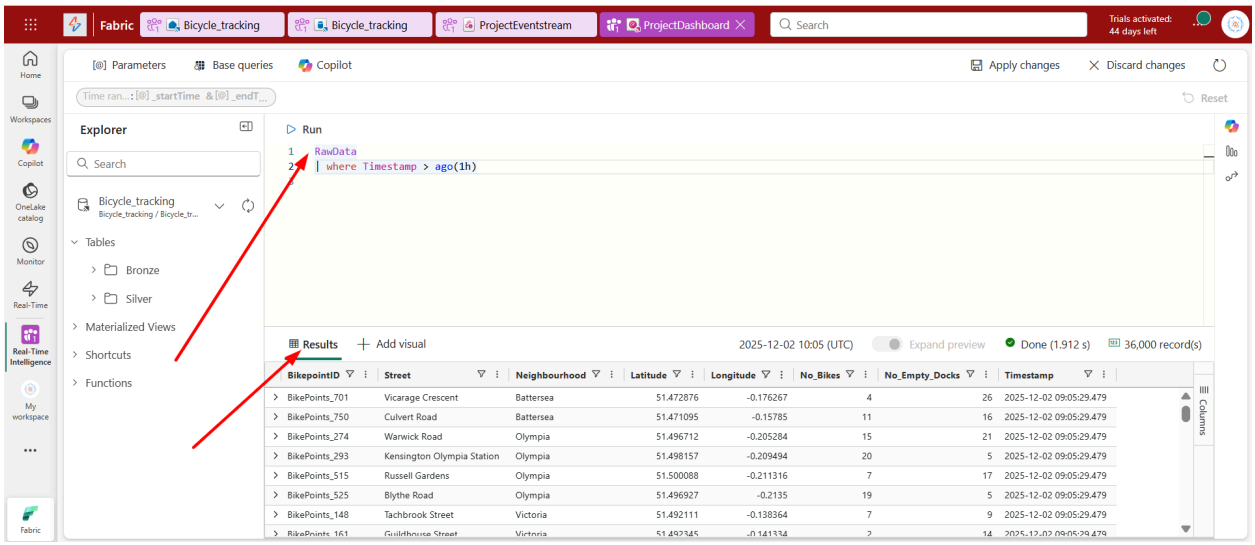
To enhance spatial analysis capabilities, a map-based visualization was added showing the geographic distribution of bike stations.




```
RawData
| where Timestamp > ago(1h)
```

Map Tile Query:

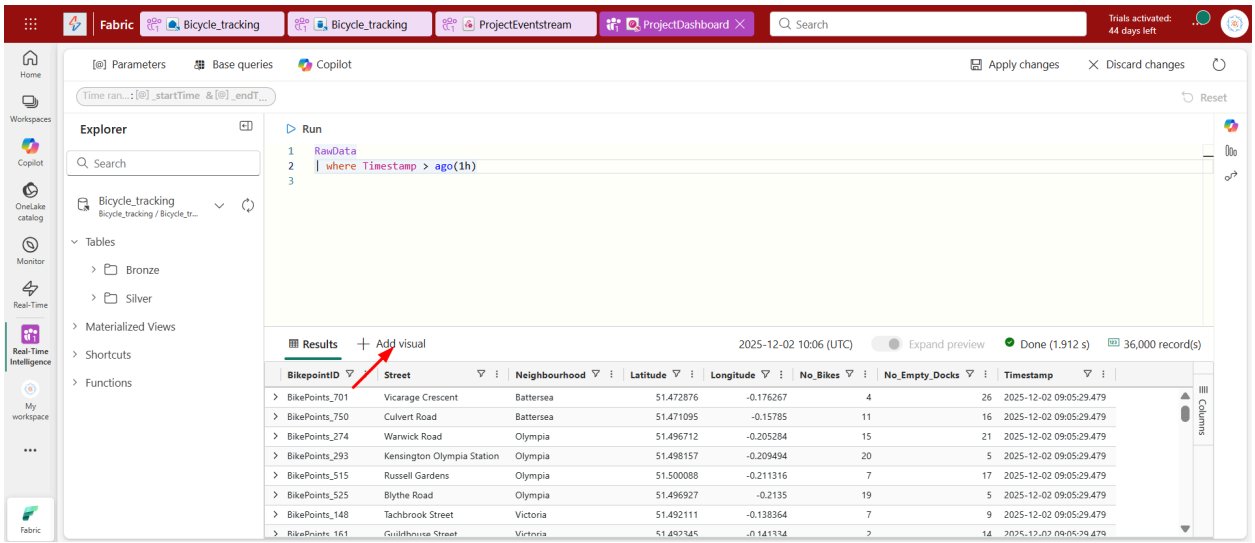
```
RawData
| where Timestamp > ago(1h)
```

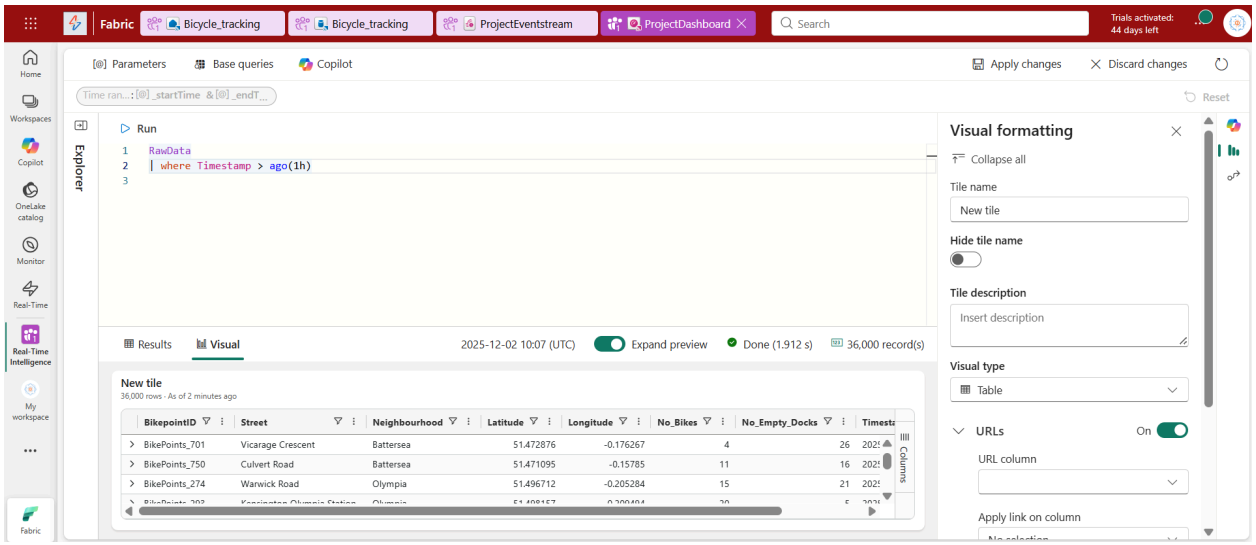


Query Purpose:

- *Filters data to show only records from the last hour*
- *Ensures the map displays current, real-time information*
- *Reduces data volume for optimal performance*

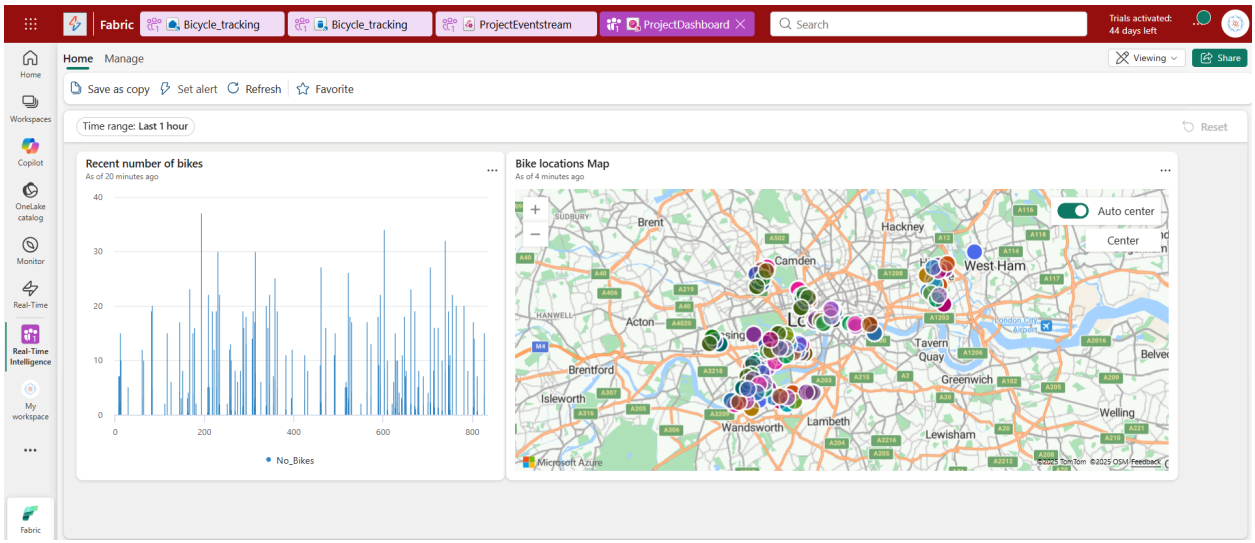
Visual Configuration:





Property	Value	Description
Tile name	Bike locations Map	Display name for the visualization
Visual type	Map	Geographic representation format
Location definition	Latitude and longitude	Coordinate-based positioning
Latitude column	Latitude	Vertical coordinate field
Longitude column	Longitude	Horizontal coordinate field
Label column	BikepointID	Station identifier displayed on map

Dashboard Features



Interactive Capabilities

- ***Real-time updates**: Dashboard refreshes automatically to display current data*
- ***Visual customization**: Tiles can be resized and repositioned according to user preferences*
- ***Map interaction**: Users can zoom and pan to explore different geographic regions*
- ***Multi-perspective analysis**: Combines quantitative (column chart) and spatial (map) views*

Data Insights Provided

1. ***Availability trends**: Column chart reveals bike availability patterns across stations*
2. ***Geographic distribution**: Map visualization shows the physical location of bike stations*
3. ***Temporal filtering**: One-hour window ensures relevance of displayed information*

Technical Implementation Notes

- *Dashboard state is persisted through the save functionality in the top-left corner*
- *Multiple tiles can coexist, enabling comprehensive data analysis*
- *The dashboard can be accessed directly from the workspace item list*
- *Visual formatting is applied before finalizing tile configuration to ensure proper rendering*

Outcome

The completed Real-Time Dashboard provides a comprehensive monitoring solution with three integrated visualization tiles, combining aggregated statistics and real-time geographic data to support data-driven decision-making for bicycle sharing operations.

Anomaly Detection Implementation for Real-Time Data Monitoring

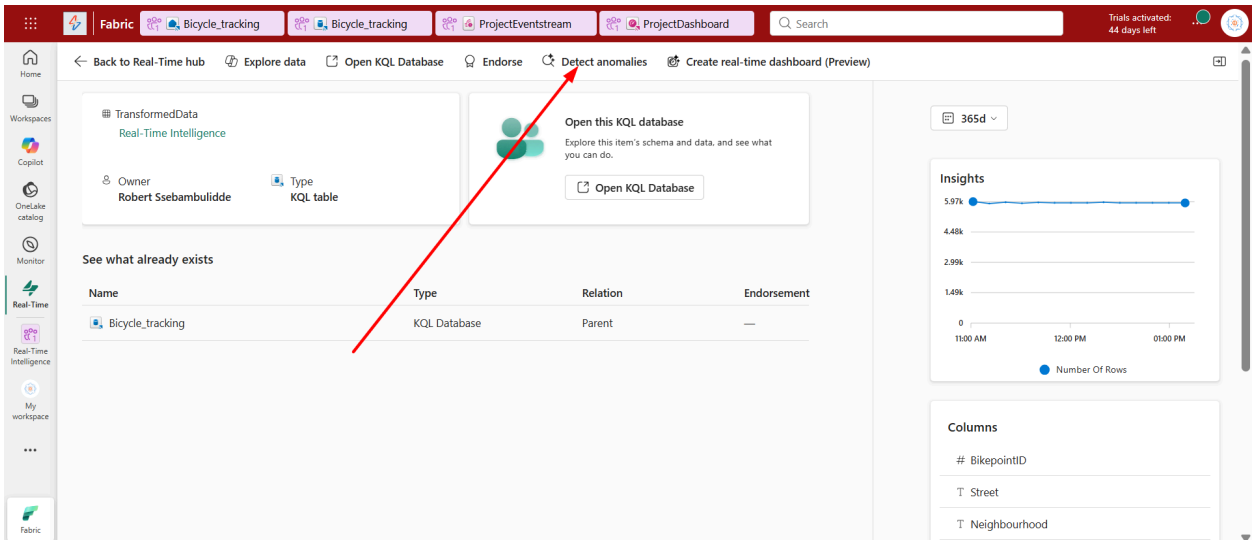
This implements an anomaly detection system to monitor and identify irregular patterns in bike-sharing station data, specifically focusing on unusual fluctuations in the availability of empty docking spaces across different station locations.

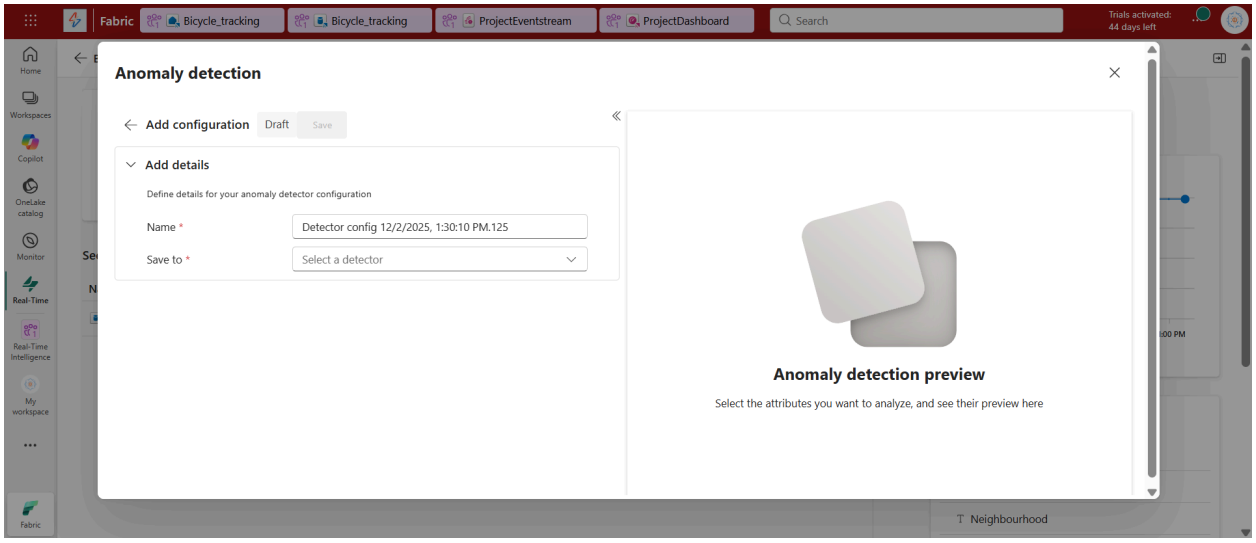
- *Implement real-time anomaly detection on streaming data infrastructure*
- *Monitor empty dock availability patterns across multiple bike stations*
- *Create automated alerts for unusual operational conditions*
- *Build scalable anomaly detection models for time-series data analysis*

Implementation Steps

1. Data Source Configuration

Navigate to the Real-Time hub from the workspace navigation panel and access the TransformedData eventhouse table created in the data transformation phase. From the table details interface, initiate the anomaly detection feature.





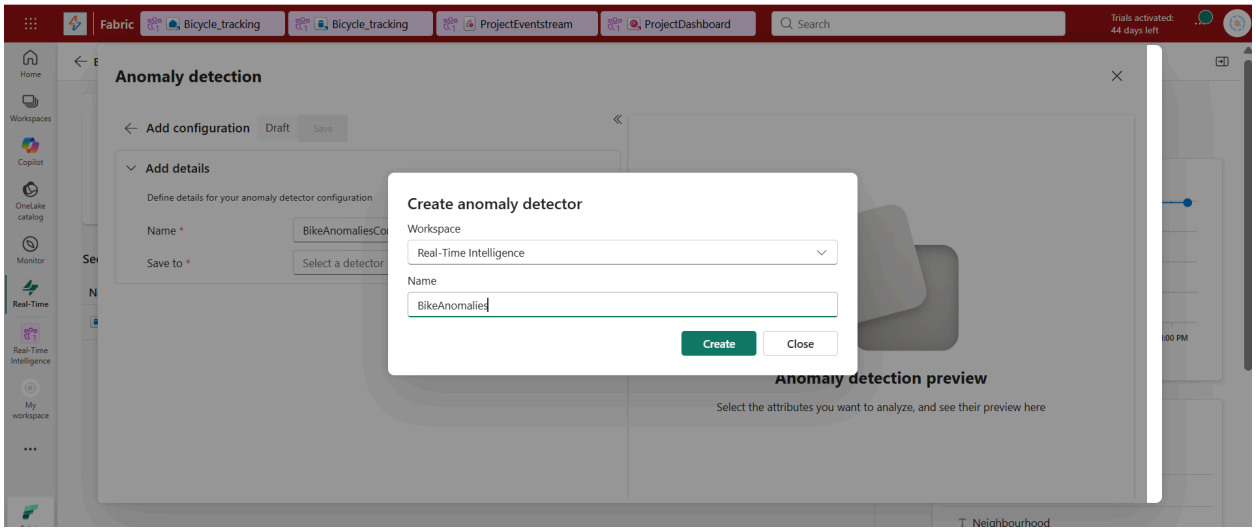
2. Anomaly Detector Setup

Configure a new anomaly detector with the following specifications:

Configuration Name: BikeAnomaliesConfiguration

Detector Item Name: BikeAnomalies

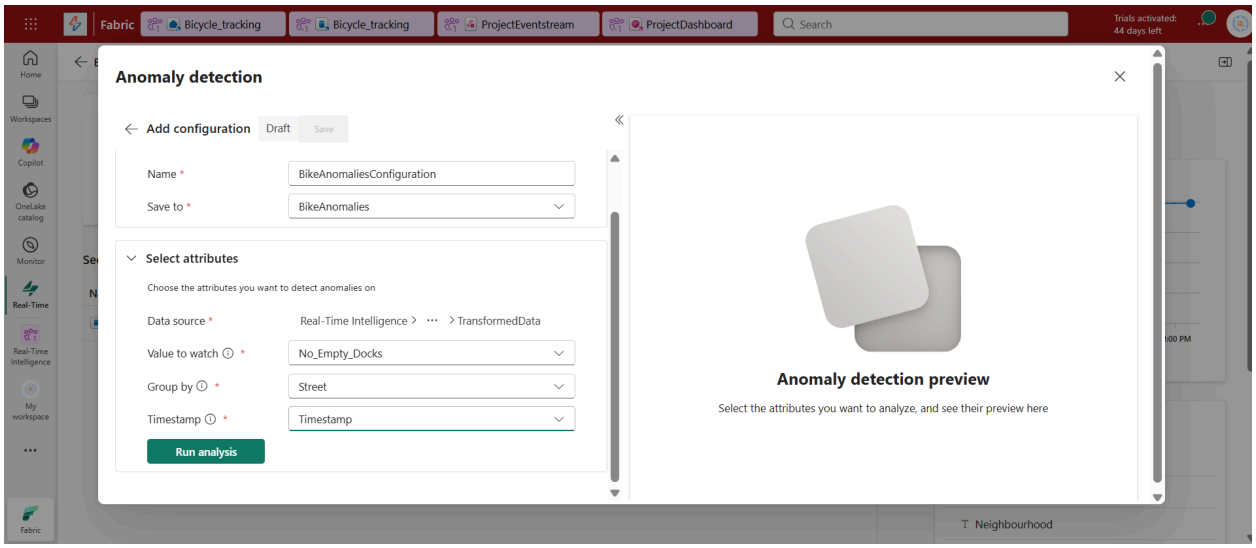
The detector is created and saved within the designated workspace for future monitoring and analysis.



3. Attribute Selection and Model Parameters

Define the following key attributes for anomaly detection:

Parameter	Configuration Value	Purpose
Value to watch	No_Empty_Docks	Primary metric being monitored for anomalies
Group by	Street	Enables per-location anomaly detection
Timestamp	Timestamp	Time-series alignment for pattern recognition



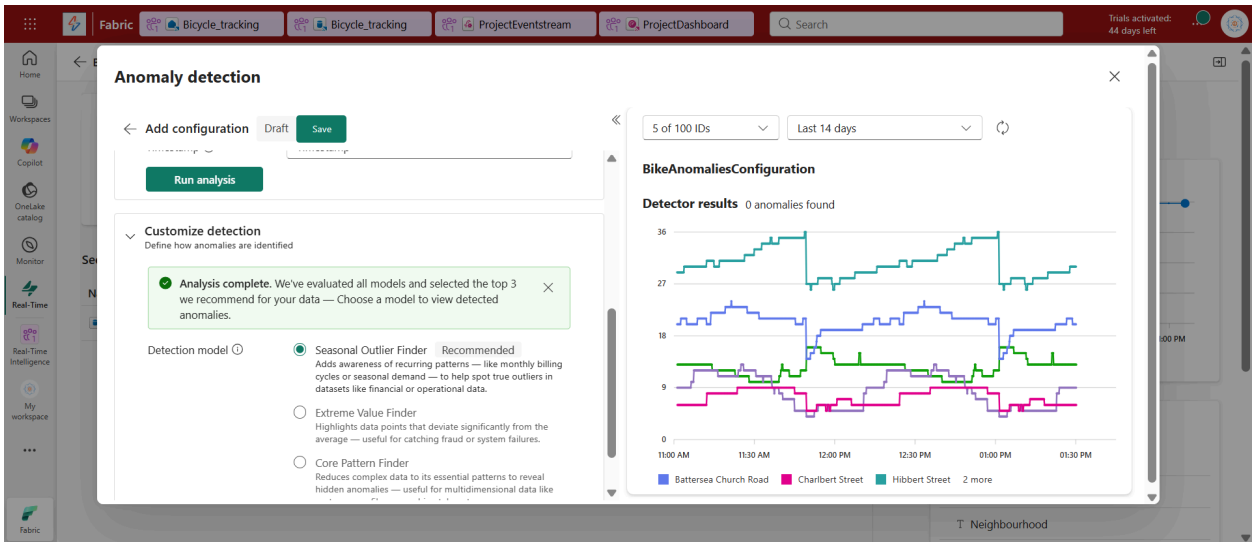
4. Model Training and Analysis

Execute the analysis process to train the anomaly detection model on historical data patterns. The analysis duration varies based on dataset size, typically completing within 4 to 30 minutes depending on data volume and complexity.

Data Requirements:

- Sufficient historical data is essential for accurate model training
- High-frequency data (per-second readings) requires several days of history

- Low-frequency data (daily readings) requires several months for optimal results



5. Results Validation

Upon completion, the system displays:

- Detected anomalies highlighted in the time-series visualization
- Tabular data showing anomaly details and timestamps
- Statistical metrics and confidence scores

6. Model Optimization

Fine-tune detection accuracy through:

- ***Detection Model Customization:*** *Adjust sensitivity thresholds and algorithm parameters*
- ***Timestamp Range Selection:*** *Refine the analysis window for improved pattern recognition*
- ***Iterative Testing:*** *Evaluate different configurations to optimize detection performance*

Technical Outcomes

- *Successfully deployed automated anomaly detection on real-time streaming data*
- *Established baseline patterns for normal operational behavior*
- *Created a scalable monitoring solution capable of multi-location analysis*
- *Implemented time-series analysis techniques for pattern recognition*

Key Learnings

- *Anomaly detection model performance improves significantly with adequate historical data*
- *Grouping by location (Street) enables granular monitoring across different stations*
- *Model customization allows for balancing sensitivity between false positives and missed anomalies*
- *Real-time processing capabilities enable proactive operational monitoring*

Future Enhancements

- *Integration with alerting systems for immediate notification of detected anomalies*
- *Expansion to additional metrics beyond empty dock availability*
- *Implementation of predictive models to forecast potential anomalies*
- *Development of automated response workflows triggered by anomaly detection*