

<u>Real-Time Intelligence System for Event-Driven Data Analysis</u>	1
<u>Executive Summary</u>	1
<u>Project Scope</u>	1
<u>Business Context</u>	1
<u>Project Overview</u>	1
<u>Technical Requirements</u>	2
<u>System Prerequisites</u>	2
<u>Project Deliverables</u>	2
<u>Setting Up Real-Time Data Infrastructure</u>	3
<u>Implementation Steps</u>	3
1. <u>Workspace Configuration</u>	3
2. <u>Eventhouse Deployment</u>	3
3. <u>Provisioning Verification</u>	3
<u>Technical Architecture</u>	4
<u>Skills Demonstrated</u>	5
<u>Get data in the Real-Time Hub</u>	5
<u>Implementation Components</u>	5
1. <u>Event Stream Configuration</u>	5
2. <u>Event Transformation Logic</u>	8
3. <u>Destination Configuration</u>	12
4. <u>Pipeline Deployment</u>	16
<u>Technical Architecture</u>	16
<u>Key Features Demonstrated</u>	16
<u>Results</u>	16
<u>Setting Up Real-Time Alerts for Bike Availability Monitoring</u>	17
<u>Implementation Steps</u>	17
1. <u>Accessing the Event Stream</u>	17
2. <u>Alert Configuration</u>	18
3. <u>Alert Rule Definition</u>	19
4. <u>Alert Activation</u>	20
<u>Functionality</u>	22
<u>Technical Outcome</u>	23
<u>Data Transformation Implementation in KQL Database</u>	24
<u>Implementation Steps</u>	24
1. <u>Data Organization - Bronze Layer Setup</u>	24
2. <u>Target Table Creation - Silver Layer</u>	26
3. <u>Transformation Logic - Function Development</u>	27
4. <u>Update Policy Configuration</u>	29

<u>5. Validation and Verification</u>	30
<u>Technical Achievements</u>	31
<u>Key Insights</u>	32
<u>Query Streaming Data Using KQL</u>	33
<u>1. Time-Series Data Visualization</u>	33
<u>Implementation</u>	33
<u>Key Features</u>	34
<u>2. Materialized View for Real-Time Aggregation</u>	34
<u>Materialized View Creation</u>	34
<u>Technical Details</u>	35
<u>Visualization Query</u>	35
<u>3. Cross-Language Query Support: T-SQL Integration</u>	36
<u>T-SQL Query Implementation</u>	36
<u>4. Query Translation: SQL to KQL Conversion</u>	37
<u>Translation Method</u>	37
<u>Process</u>	37
<u>Learning Value</u>	38
<u>Project Skills Demonstrated</u>	38
<u>Real-Time Dashboard Implementation for Streaming Data Visualization</u>	39
<u>Implementation Steps</u>	39
<u>1. Dashboard Query Development</u>	39
<u>2. Dashboard Creation</u>	40
<u>3. Geographic Visualization Implementation</u>	41
<u>Dashboard Features</u>	44
<u>Interactive Capabilities</u>	44
<u>Data Insights Provided</u>	44
<u>Technical Implementation Notes</u>	45
<u>Outcome</u>	45
<u>Anomaly Detection Implementation for Real-Time Data Monitoring</u>	46
<u>Implementation Steps</u>	46
<u>1. Data Source Configuration</u>	46
<u>2. Anomaly Detector Setup</u>	47
<u>3. Attribute Selection and Model Parameters</u>	47
<u>4. Model Training and Analysis</u>	48
<u>5. Results Validation</u>	49
<u>6. Model Optimization</u>	49
<u>Technical Outcomes</u>	50
<u>Key Learnings</u>	50
<u>Future Enhancements</u>	50

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:

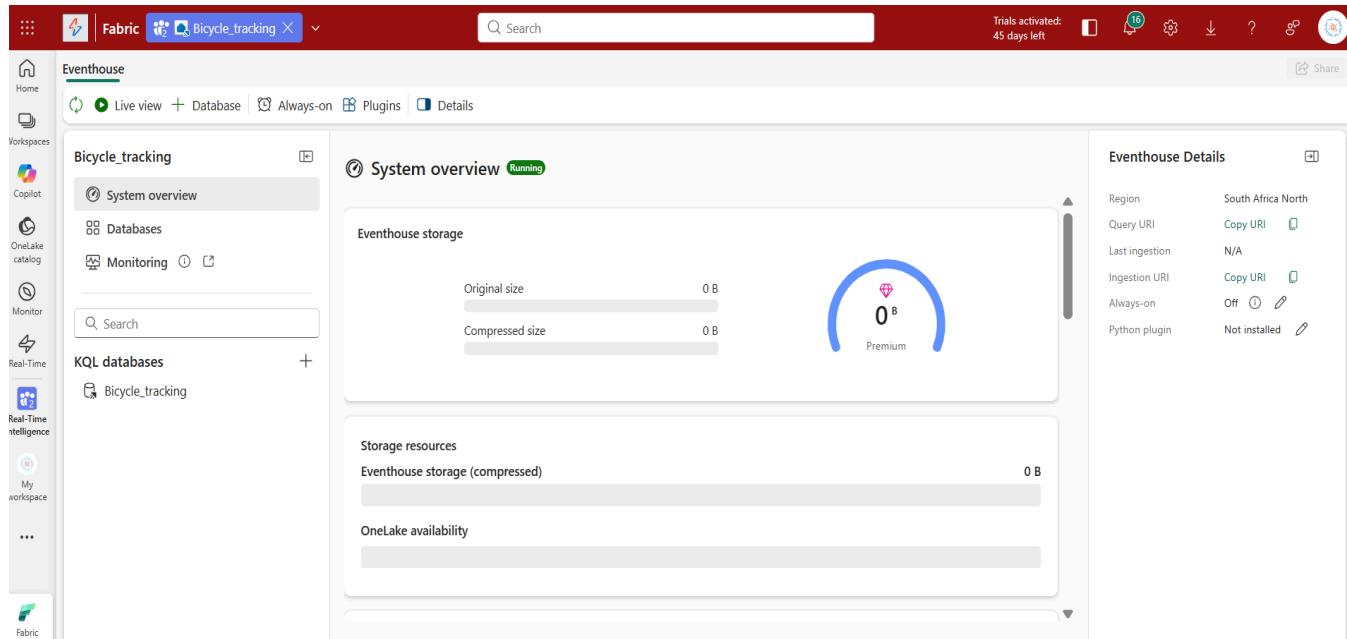
- **Resource Name:** Bicycle_tracking
- **Service Type:** Eventhouse
- **Auto-provisioned Component:** KQL Database (inherits parent name)

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*

Discover, analyze, and act on data-in-motion

Recent streaming data

1 Data	Source item	Item owner	Workspace	Endorsement	Sensi
Automotive	Bicycle_tracking	Robert Ssebambulidde	Real-Time Intelligence	—	—
ev1	eventhouse1	Robert Ssebambulidde	Data Engineering	—	—
eventstream1-stream	eventstream1	Robert Ssebambulidde	Data Engineering	—	—
onelake_events_eventstream-stream	onelake_events_eventstream	Robert Ssebambulidde	Datadepartment	—	—

Filter by: Data type, Item owner, Item, Workspace

Search for streaming data | Add data

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

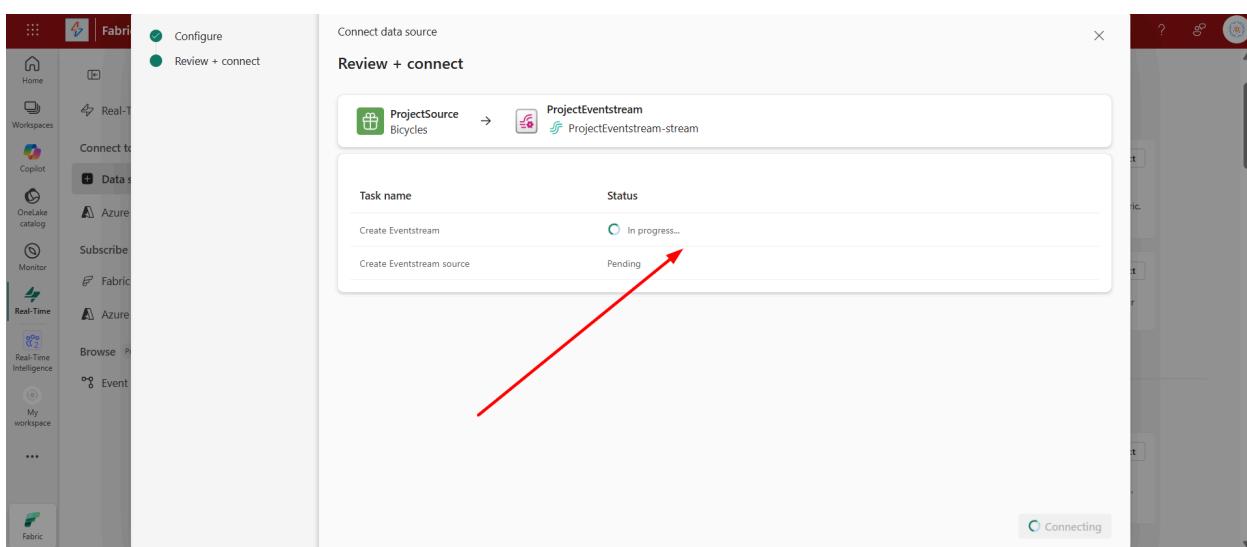
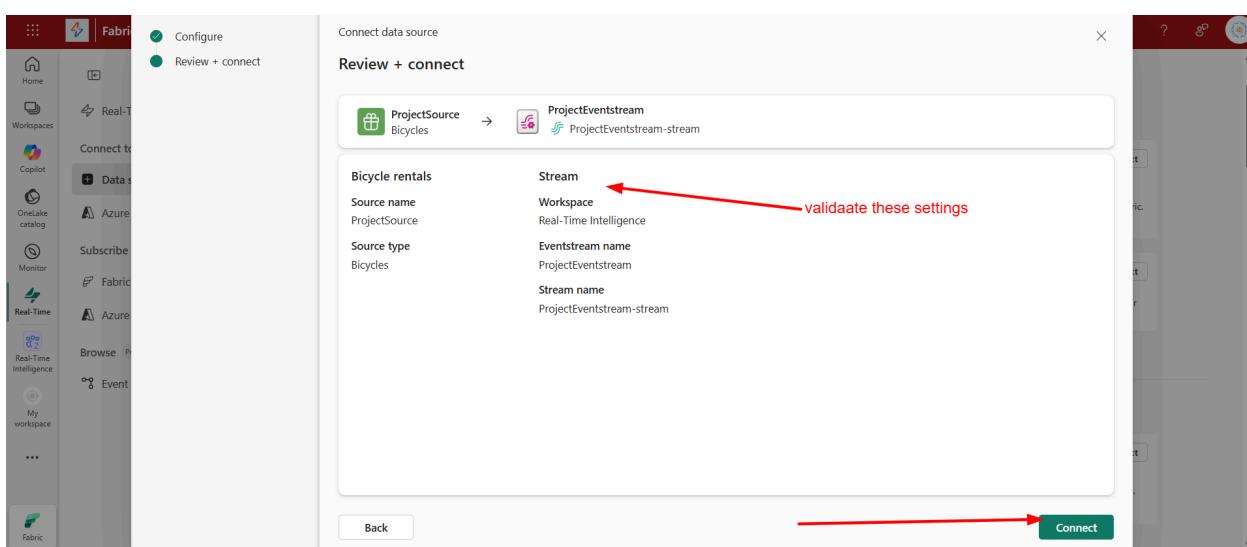
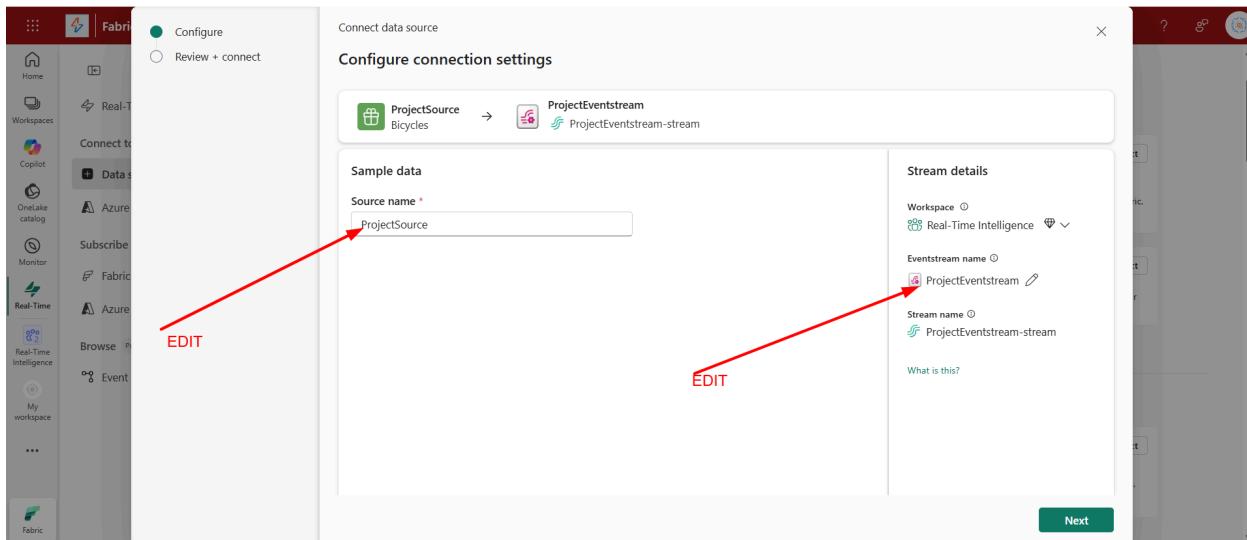
Discover, analyze, and act on data-in-motion

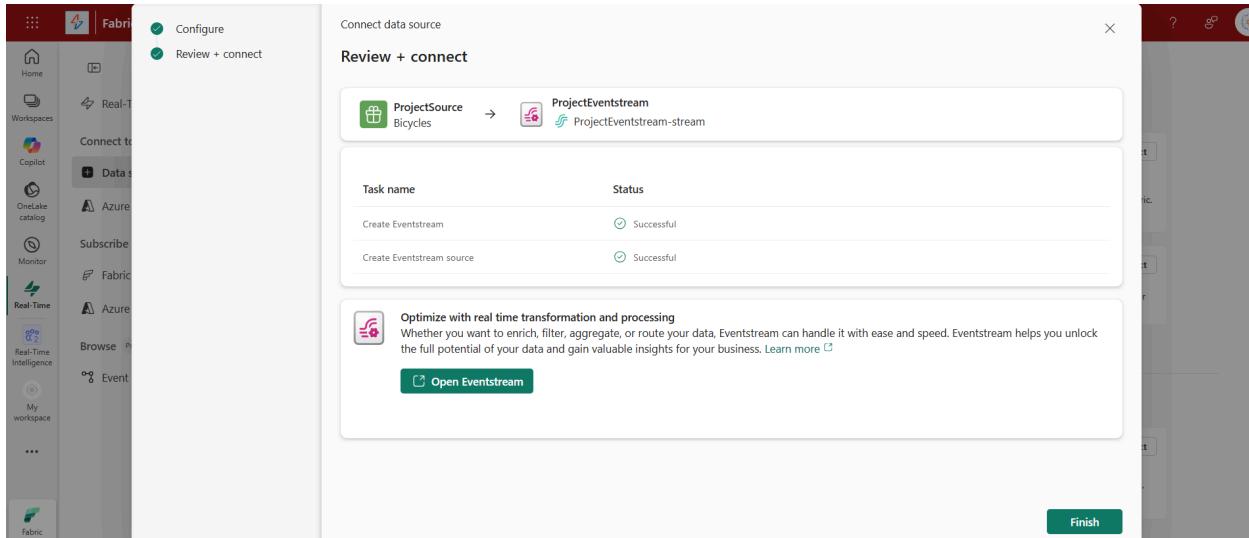
Recommended

Sample scenarios

Bicycle rentals	Stock market	Azure diagnostics logs	Azure Event Hubs
Streaming sample data of bicycle rentals, including rental station locations (longitude and latitude), number of bikes, number of empty docks.	Eventstream of stock market data including bids, prices, volume, event times, and more.	Pull diagnostics logs and metrics data from your Azure resources and stream it into Fabric.	Pull data from your Azure Event Hubs and stream it into Fabric.
Yellow Taxi	Stock market	OneLake events	Fabric Workspace Item events
Eventstream of taxi data including pickup and drop off times, passenger counts, distances, fares, and more.	Eventstream of stock market data including bids, prices, volume, event times, and more.	Route your Fabric OneLake events, such as creation or deletion, to Fabric.	Route your Fabric Workspace Item events, such as creation or deletion, to Fabric.

All | Microsoft sources | Database CDC | Azure events | Fabric events | Sample scenarios | Public Feeds





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

2. Event Transformation Logic

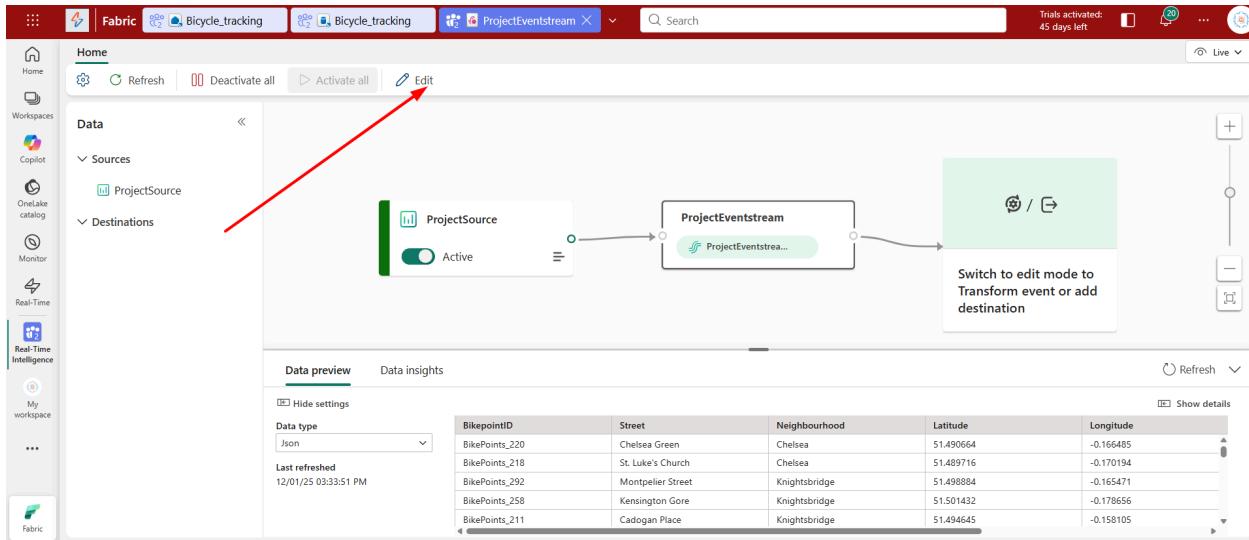
Implement data transformation to enrich incoming events with timestamp metadata.

Technical Implementation:

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

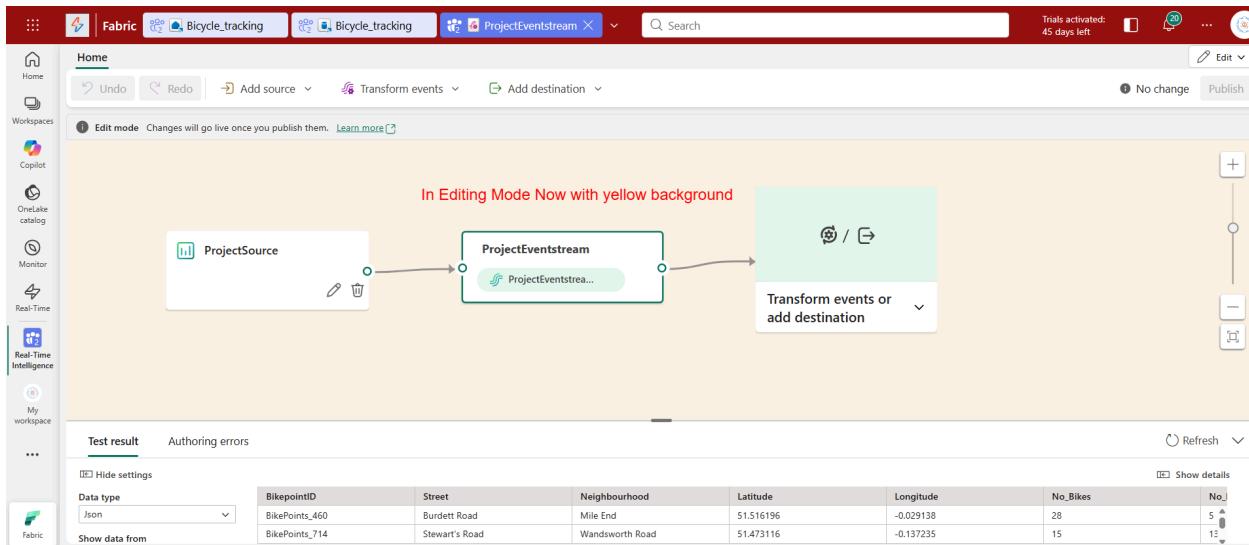
1. Transformation Setup

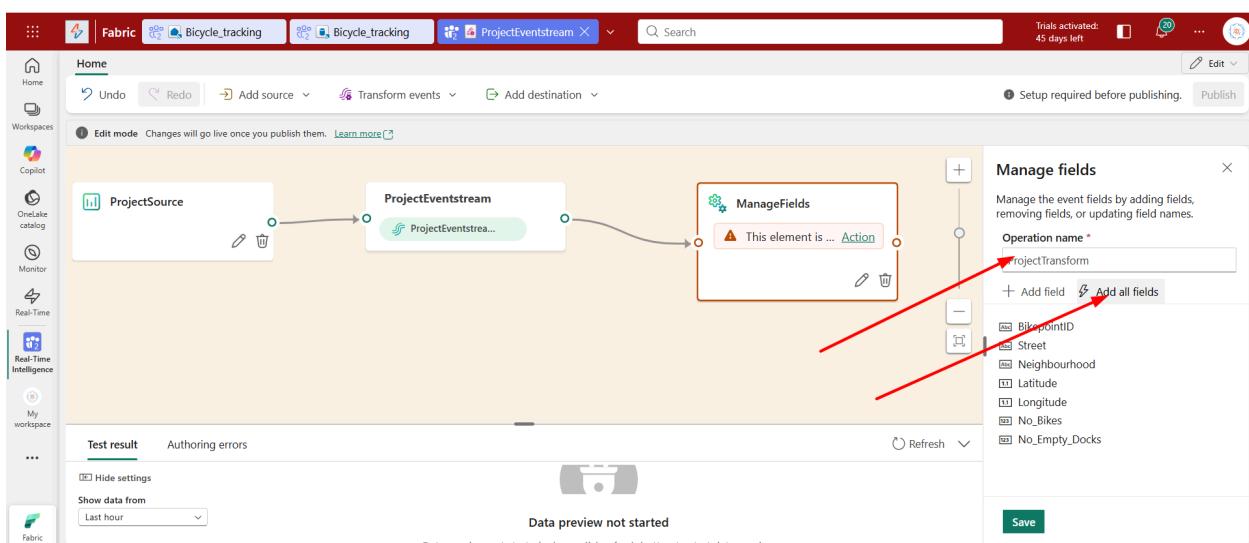
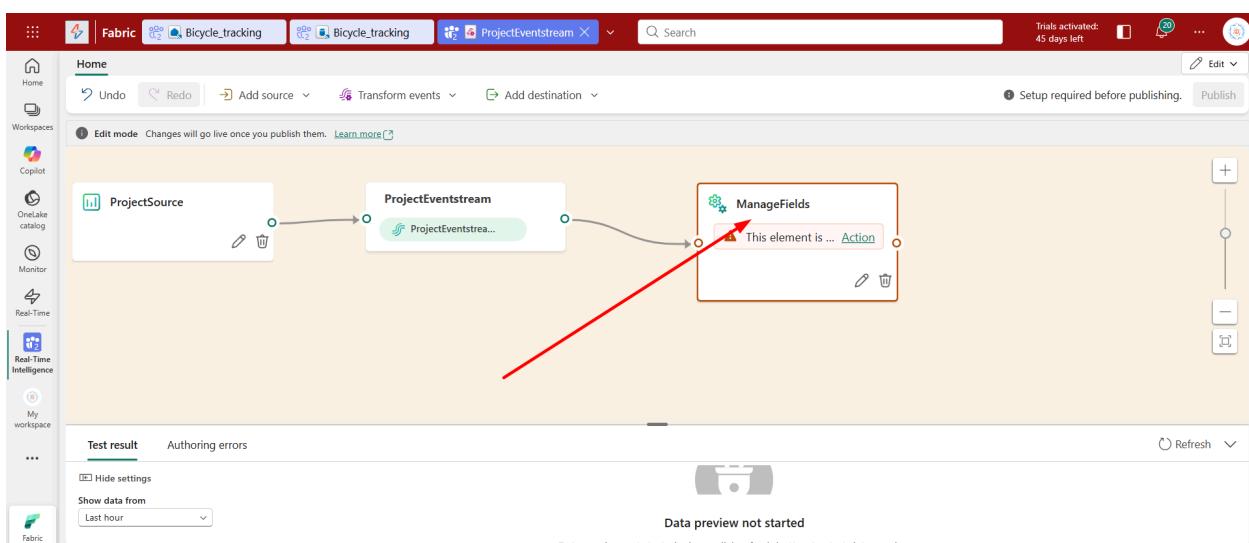
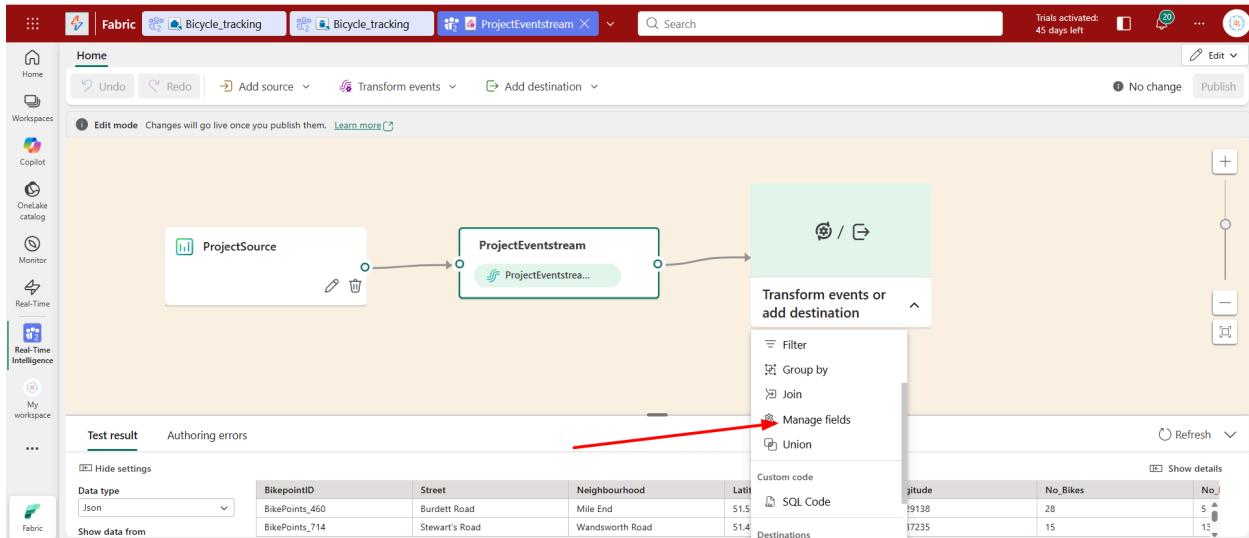
- Activated the editing mode in the event stream authoring interface
- Added a field management transformation component named ProjectTransform



2. Field Mapping Configuration

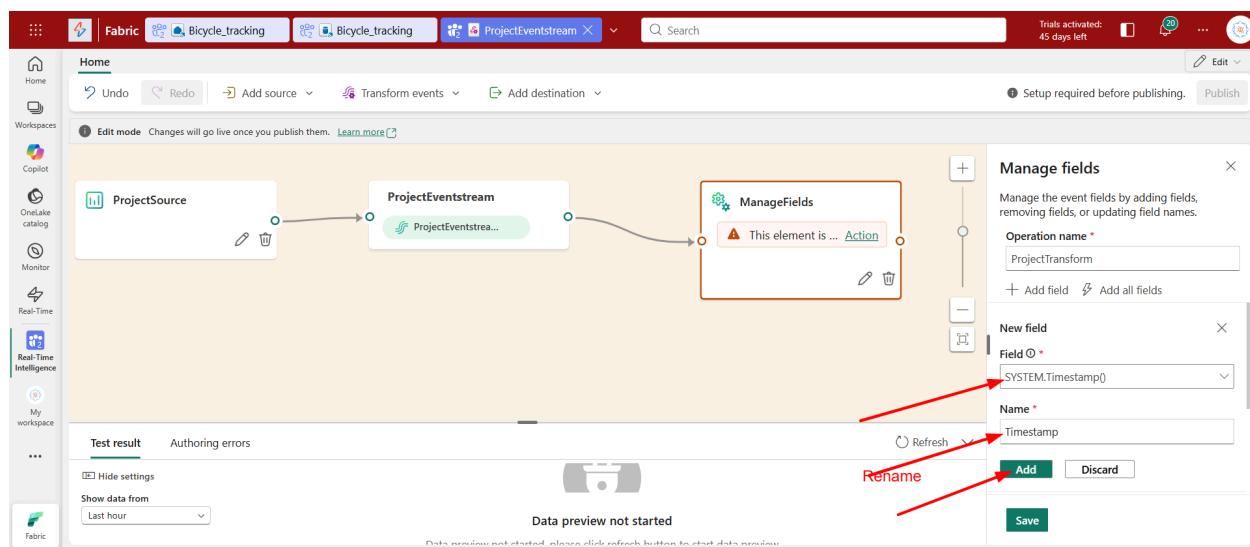
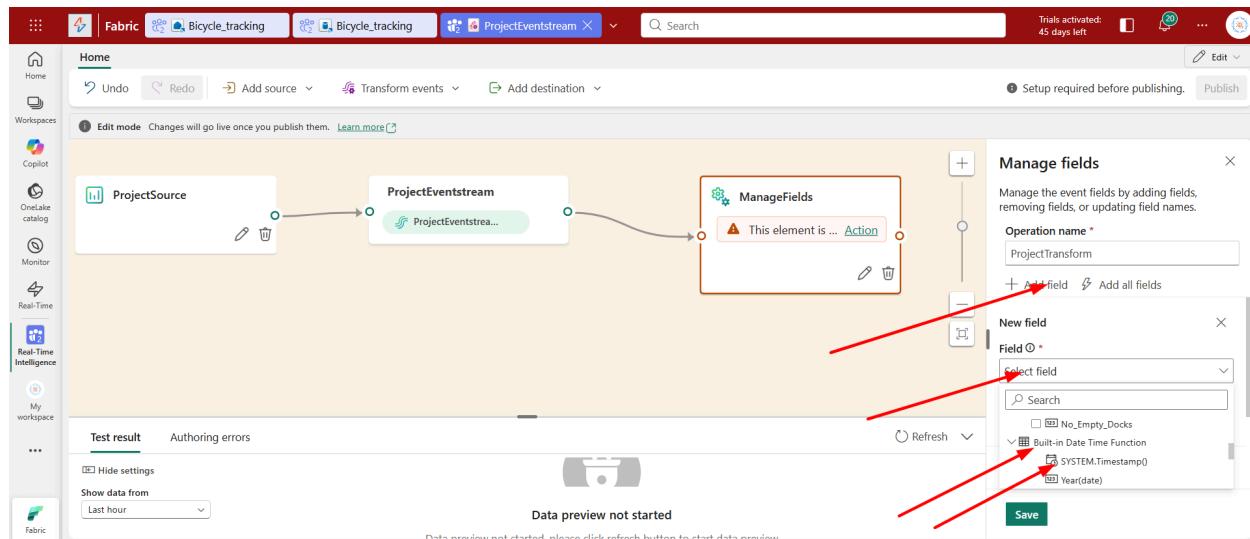
- Preserved all existing fields from the source data
- Implemented custom field addition using built-in functions

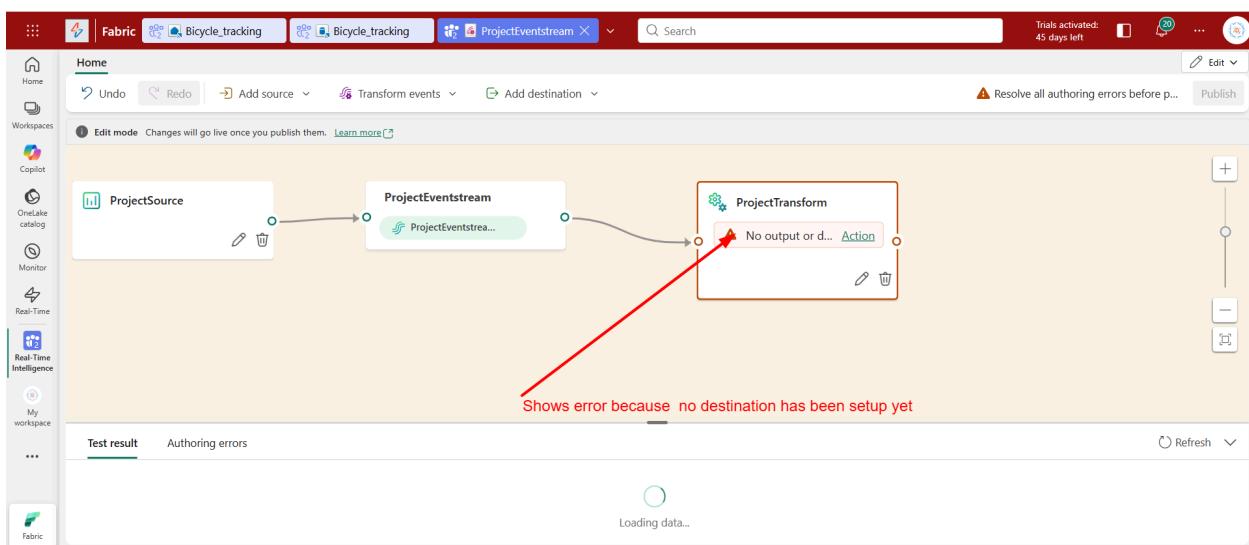
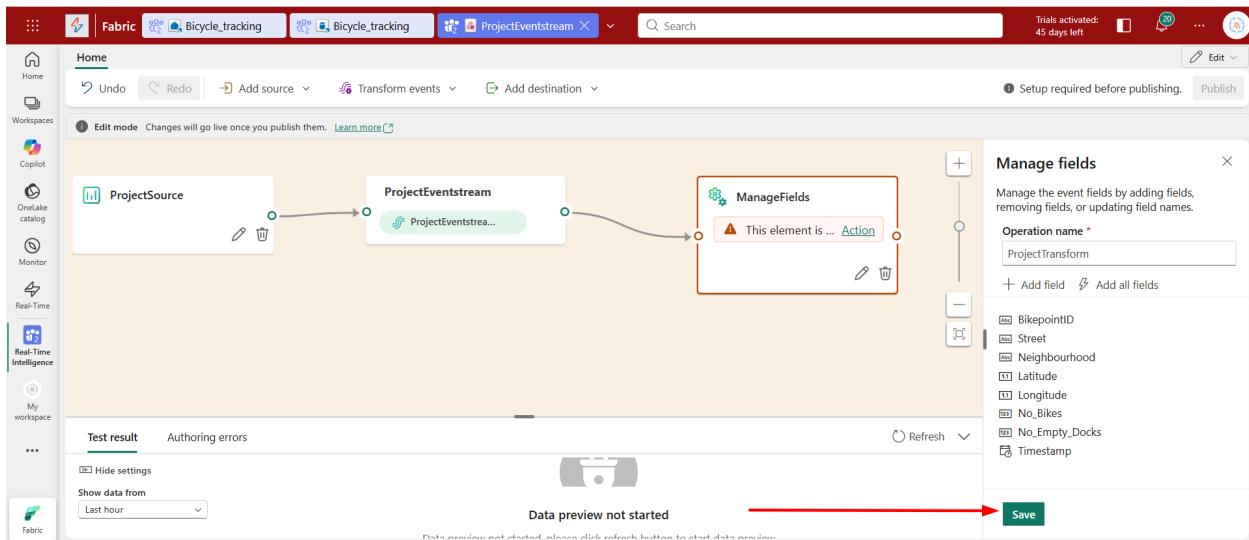




3. Timestamp Enrichment

- 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.

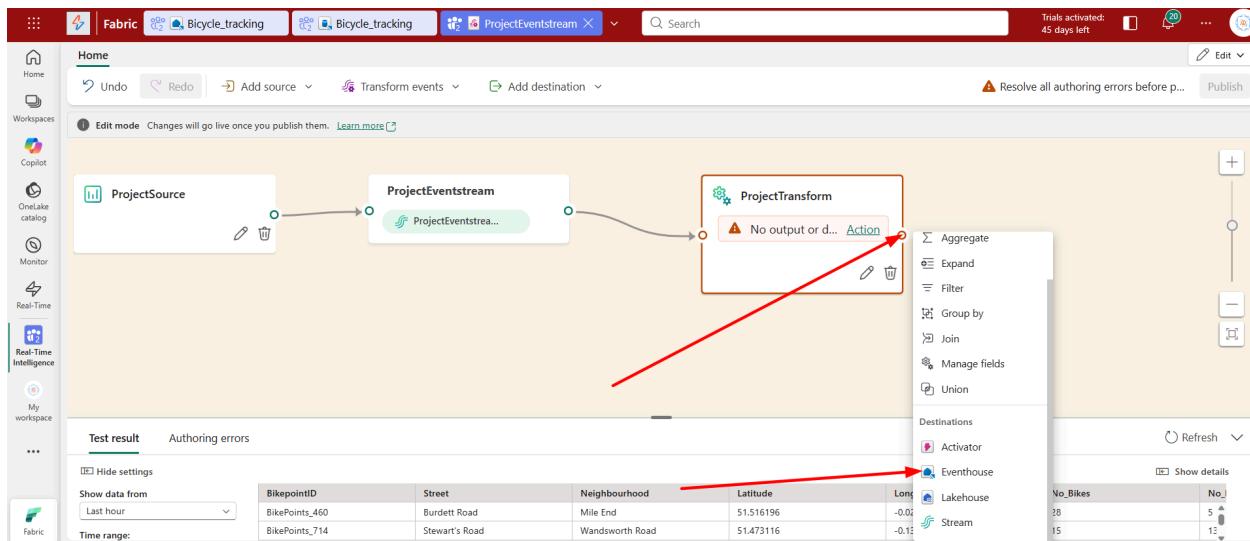
Configuration Details:

Established connection to an Eventhouse database with the following specifications:

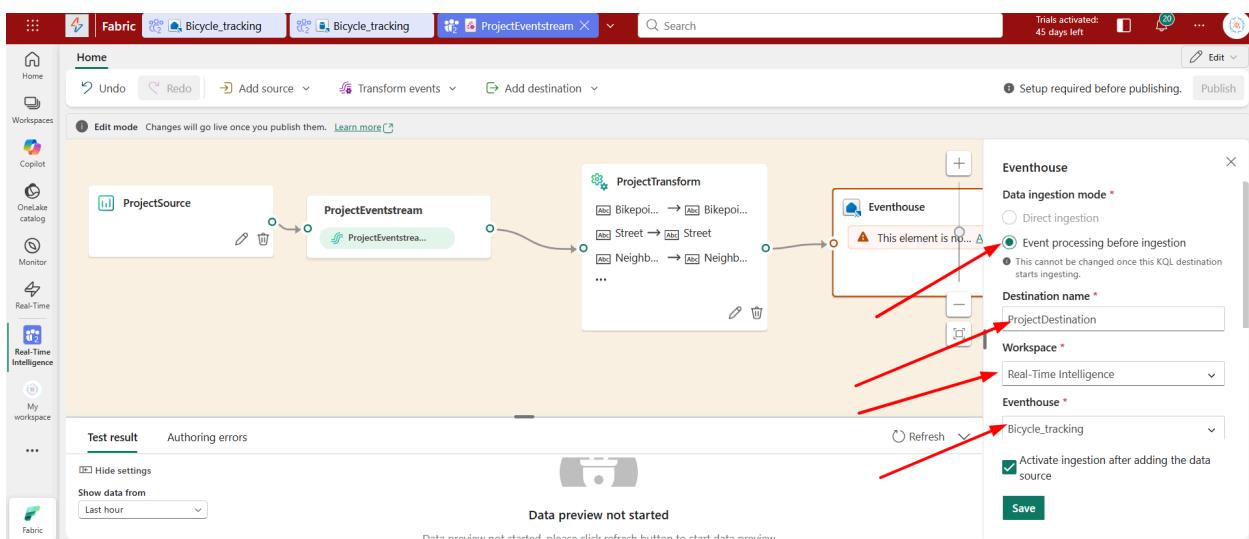
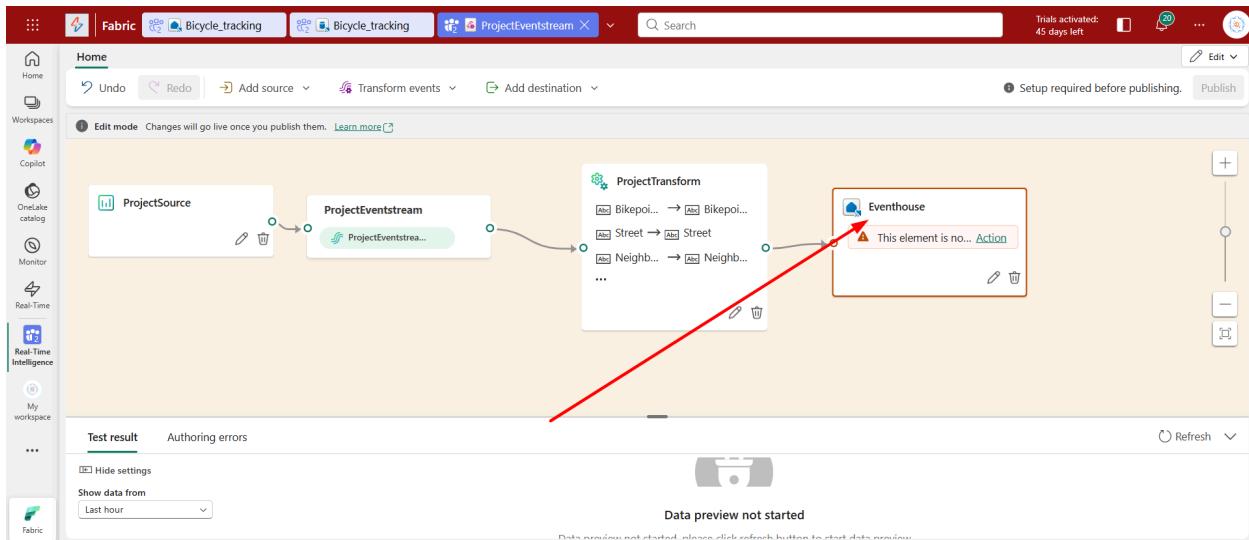
Parameter	Value
Ingestion Mode	Event processing before ingestion
Destination Name	ProjectDestination
Target Database	Project(KQL Database)
Destination Table	RawData (newly created)
Data Format	JSON
Auto-activation	Enabled

Implementation Process:

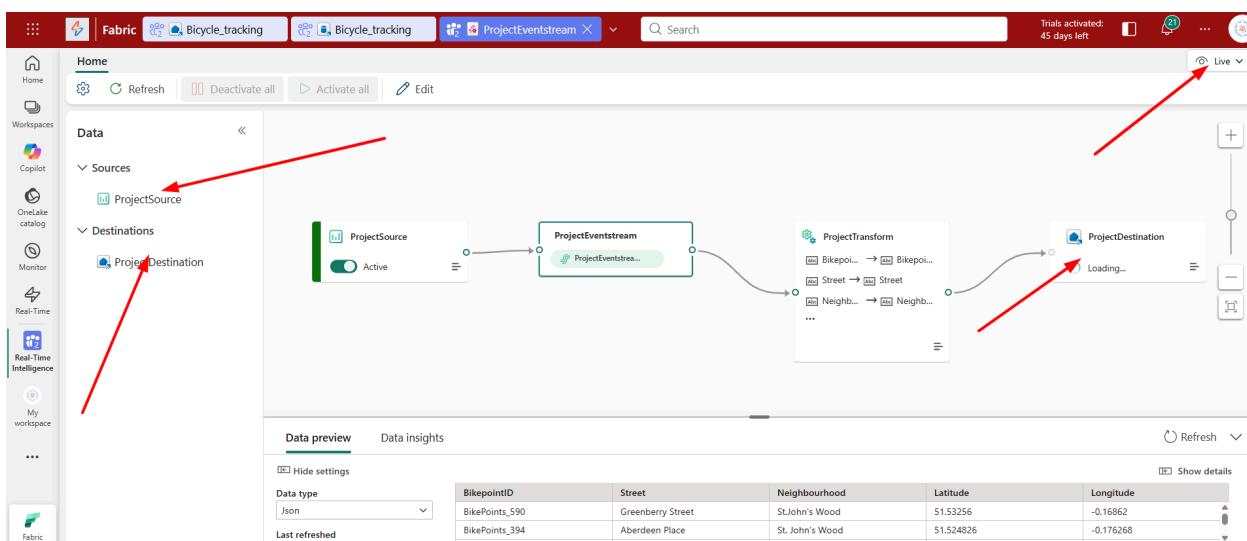
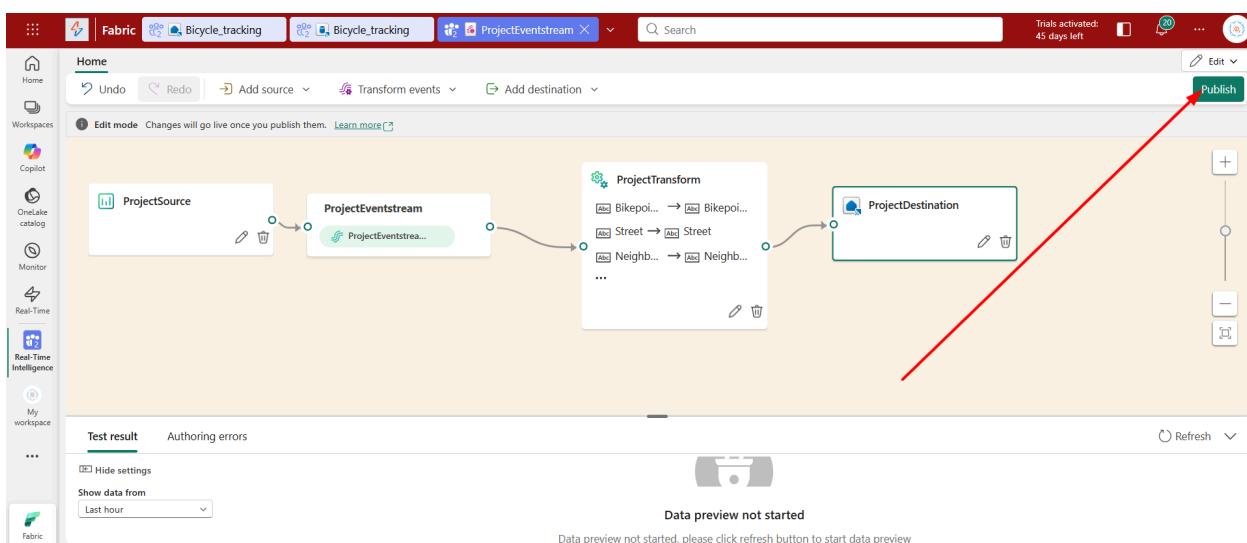
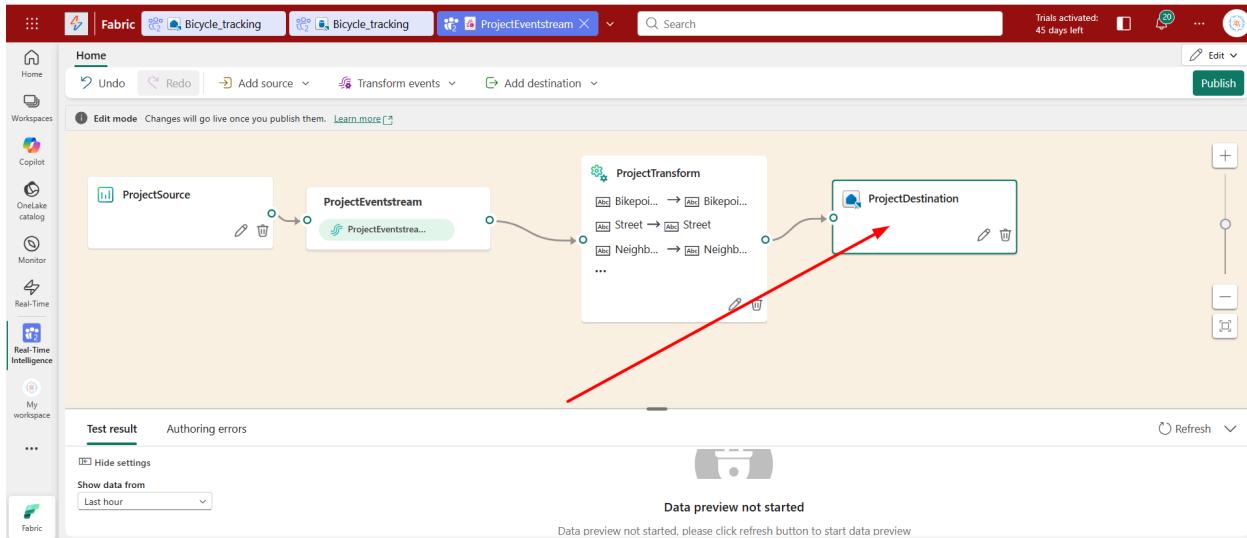
1. Connected transformation output to Eventhouse destination



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



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.

The screenshot shows the DataInMotion Real-Time interface. On the left sidebar, under the 'Real-Time' section, the 'Real-Time events' option is highlighted with a red arrow. The main area displays a hub with three cards: 'Subscribe to OneLake events', 'Act on Job events', and 'Visualize data'. Below the hub, a table titled 'Recent streaming data' lists two entries:

Data	Source item	Item owner	Workspace	Endorsement	Sensi
ProjectEventstream-stream	ProjectEventstream	Robert Ssebambulidde	Real-Time Intelligence	—	—
RawData	Bicycle_tracking	Robert Ssebambulidde	Real-Time Intelligence	—	—

The screenshot shows the Fabric Real-Time hub interface. On the left, there's a sidebar with icons for Home, Workspaces, Copilot, OneLake catalog, Monitor, Real-Time Intelligence, My workspace, and Fabric. The main area has tabs for 'Fabric', 'Bicycle_tracking', 'Bicycle_tracking', and 'ProjectEventstream'. A search bar and a 'Trials activated: 44 days left' message are at the top right. The central part displays the 'ProjectEventstream-stream' details, including its owner (Robert Ssebambulidde) and type (Stream). It also features a 'Stay on top of your business events' section with a 'Set alert' button. Below this, a table lists existing components: ProjectEventstream (Eventstream, Parent), Bicycle_tracking (KQL Database, Downstream), and ProjectSource ([Connector] SampleData, Upstream). To the right is a 'Stream profile' section with a chart titled 'Insights' showing message counts over the last 6 hours. The chart has two series: 'IncomingMessages' (blue line) and 'OutgoingMessages' (purple line), both starting at 0 and ending near 24.15k.

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.

This screenshot is identical to the one above, but it includes a red arrow pointing specifically to the 'Set alert' button in the 'Stay on top of your business events' section of the central panel.

The screenshot shows the Fabric Real-Time Intelligence interface. On the left, there's a sidebar with various icons for Home, Workspaces, Copilot, OneLake catalog, Monitor, Real-Time Intelligence, and My workspace. The main area displays a ProjectEventstream-stream named 'ProjectEventstream' owned by Robert Ssebambulidde. Below this, a table lists existing components: ProjectEventstream (Eventstream, Parent), Bicycle_tracking (KQL Database, Downstream), and ProjectSource ([Connector] SampleData, Upstream). On the right, an 'Add rule' dialog box is open. It has tabs for 'Details', 'Monitor', 'Condition', and 'Action'. In the 'Details' tab, the 'Rule name' field is empty. In the 'Monitor' tab, the 'Source' is set to 'ProjectEventstream'. In the 'Condition' tab, the 'Check' is set to 'On each event'. In the 'Action' tab, the 'Select action' is 'Message to individuals' and the 'To' field contains 'Rssebambulidde'. A red arrow points from the 'Condition' tab towards the 'Set alert' button located at the bottom left of the main interface.

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

The screenshot shows the Fabric Real-Time Intelligence interface. The left sidebar and main project details are identical to the previous screenshot. The 'Add rule' dialog box is open and filled with configuration. In the 'Condition' tab, the 'Check' is 'On each event when', 'Field' is 'No_Bikes', 'Operation' is 'Is less than', and 'Value' is '5'. A red arrow points from the 'Condition' tab towards the 'Create' button at the bottom right of the dialog box.

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)*

The screenshot shows the SamaBrains Real-Time Intelligence platform. On the left, there's a sidebar with various icons for Home, Workspaces, Copilot, OneLake catalog, Monitor, Real-Time Intelligence, My workspace, and Fabric. The main area displays a project named 'ProjectEventstream-stream' under 'Real-Time Intelligence'. It shows the owner as 'Owner Robert Ssebambulidde' and a 'Type Stream'. Below this, a section titled 'See what already exists' lists items: 'ProjectEventstream' (Eventstream, Parent), 'Bicycle_tracking' (KQL Database, Downstream), and 'ProjectSource' ([Connector] SampleData, Upstream). To the right, a modal window titled 'Add rule' is open. It contains fields for 'To' (with recipients 'Rssebambulidde' and 'SamaBrains Admin'), 'Subject' ('Activator alert ProjectRule'), 'Headline' ('The condition for 'ProjectRule' has been met'), 'Notes' (empty), 'Context' ('No_Bikes'), and a 'Save location' section where 'Real-Time Intelligence' is selected as the workspace, 'Create a new item' is chosen for the item, and the new item name is 'projectalert'. A 'Create' button is at the bottom of the modal.

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.

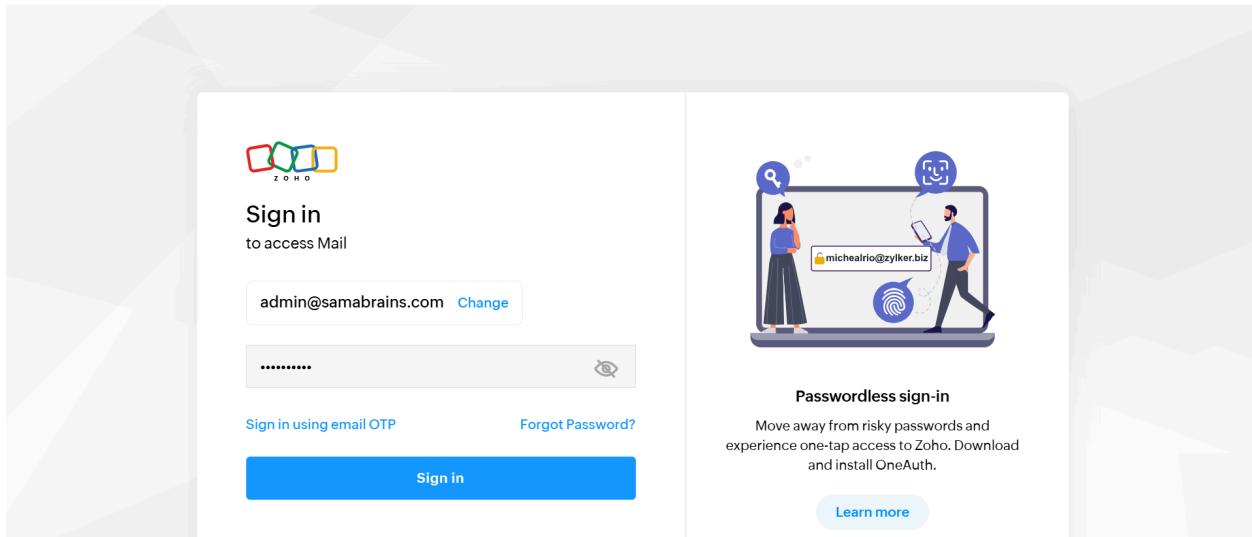
The screenshot shows the Fabric Real-Time Intelligence interface. On the left, there's a sidebar with icons for Home, Workspaces, Copilot, OneLake catalog, Monitor, Real-Time Intelligence, and My workspace. The main area has tabs for Fabric, Bicycle_tracking, and ProjectEventstream. A search bar at the top right shows 'Trials activated: 44 days left'. The central part displays a 'ProjectEventstream-stream' card with an owner named Robert Ssebambulidde and a 'Type Stream' section. Below this is a table titled 'See what already exists' listing 'ProjectEventstream' (Eventstream, Parent), 'Bicycle_tracking' (KQL Database, Downstream), and 'ProjectSource' ([Connector] SampleData, Upstream). To the right, a modal window titled 'Add rule' is open, containing fields for 'To' (Rssebambulidde, SamaBrains Admin), 'Subject' (Activator alert ProjectRule), 'Headline' (The condition for 'ProjectRule' has been met), 'Notes' (No_Bikes), and 'Context' (No_Bikes). A 'Save location' section shows 'Workspace' set to 'Real-Time Intelligence' and 'Item' set to 'Create a new item'. A red arrow points from the 'Create' button at the bottom right of the modal to the 'Create' button at the bottom right of the main interface.

This screenshot shows the same interface after the rule has been created. The 'Alert created' message is displayed on the right, stating: '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.' The 'Save location' section shows 'projectalert' selected. The 'Source' section shows 'ProjectEventstream'. The 'Event types' section lists 'ProjectEventstream'. The 'Condition' section says 'On each event when'. The 'Action' section says 'Send me an email'. A red arrow points from the 'Open' button at the bottom right of the 'Alert created' message back to the 'Create' button at the bottom right of the main interface.

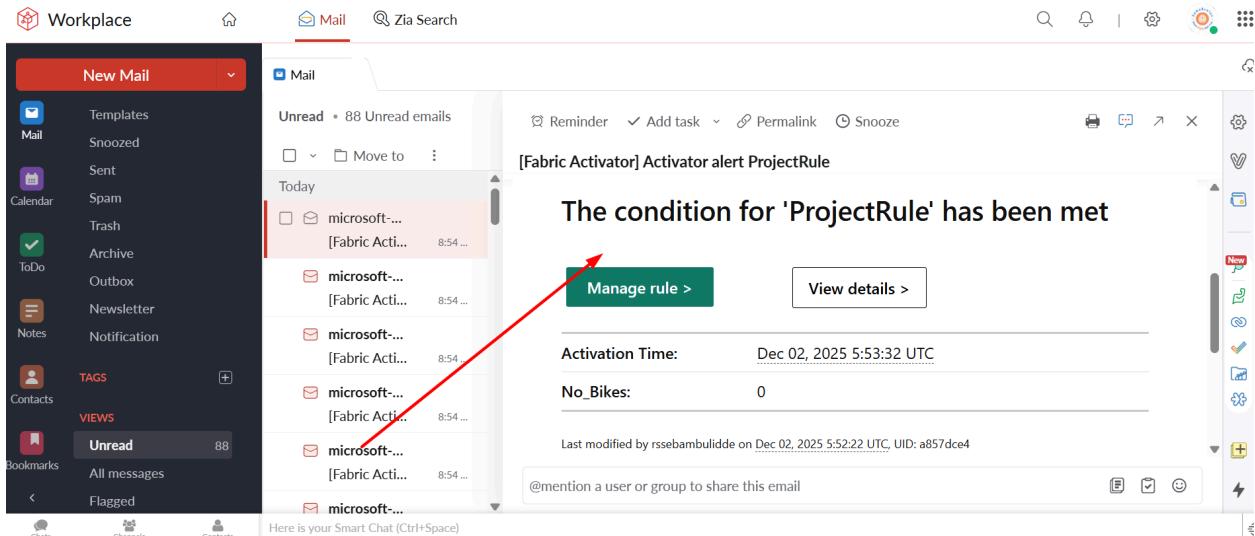
This screenshot shows the 'Rules' tab in the Fabric interface. The sidebar includes icons for Home, Create, Browse, OneLake catalog, Workspaces, Real-Time Intelligence, and a 'projectalert' workspace. The main area has tabs for 'Delete', 'Edit details', 'Start', 'Stop', and 'Send me a test action'. An 'Explorer' sidebar shows a tree structure with 'ProjectEventstream' expanded, showing 'ProjectEventstream-stream event' and 'ProjectRule'. A red arrow points from the 'Running' status indicator in the 'ProjectRule' card to the 'Live feed' tab in the main content area. The 'Live feed' tab shows a table of 'Activations' with 1076 rows, filtered by 'Time' (Last 24 hours) and 'No_Bikes' (System.Action.Email.Headline). The table includes columns for Time, No_Bikes, and System.Action.Email.Headline. A red arrow points from the bottom of the 'Activations' table to the 'Definition' panel on the right. The 'Definition' panel shows a 'Condition' section with 'Condition 1' (Operation: Is less than, Column: No_Bikes, Value: 5, Default type: None, Default: Enter value for Default) and an 'Action' section (Action: Send me an email). Buttons for 'Save' and 'Save and update' are at the bottom.

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.



A screenshot of the Zoho Workplace Mail inbox. The sidebar on the left includes links for Workplace, Home, Mail (which is selected), Zia Search, and various productivity tools like Calendar, ToDo, Notes, Contacts, and Bookmarks. The main area shows an 'Unread' folder with 89 unread emails. A red arrow points to the second email in the list, which is from 'microsoft-noreply@microsoft.com' with the subject '[Fabric Activator] Activator alert ProjectRule'. The inbox also lists other messages from the same sender with similar subjects. A red banner at the bottom of the list reads 'Incoming Email Alerts'.



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:

The screenshot shows the Microsoft Fabric Data Lake interface. On the left, the navigation bar includes 'Eventhouse', 'Database', 'Copilot', 'OneLake catalog', 'Monitor', 'Real-Time Intelligence', and 'My workspace'. The main area displays the 'Bicycle_tracking' database details. A red arrow points to the 'Bicycle_tracking_queryset' entry under 'Tables'. The interface includes a 'Data Activity Tracker' section with a bar chart showing ingestion and query activity over time, and a 'Database details' panel on the right providing metrics like size and OneLake availability.

The screenshot shows the Databricks Queryset interface. On the left, there's a sidebar with various workspace options like Home, Copilot, OneLake catalog, Monitor, Real-Time Intelligence, and My workspace. The main area shows a database named 'Bicycle_tracking' with a 'Queryset' tab selected. A red arrow points from the text 'Edit THE KQL code here' to the KQL code editor. The code editor contains the following KQL:

```

1 // Here are two articles to help you get started with KQL:
2 // KQL reference guide - https://aka.ms/KQLguide
3 // SQL - KQL conversions - https://aka.ms/salcheatsheet
4
5
6
7 // Use "take" to view a sample number of records in the table and check the data.
8 YOUR_TABLE_HERE
9 | take 100
10
11 // See how many records are in the table.
12 YOUR_TABLE_HERE

```

At the bottom of the code editor, it says 'Run a query and explore the results here'. The top right corner shows 'Trials activated: 44 days left'.

This screenshot is similar to the first one but shows a different KQL command. A red arrow points from the text 'Select the all code and run' to the KQL code editor. The code editor contains the following KQL:

```

1 .alter table RawData (BikepointID:string,Street:string,Neighbourhood:string,Latitude:real,Longitude:real,
2 No_Bikes:long,No_Empty_Docks:long,Timestamp:datetime) with
3 (folder="Bronze")
4

```

At the bottom of the code editor, it says 'Run a query and explore the results here'. The top right corner shows 'Trials activated: 44 days left'.

```
.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.

The screenshot shows the Databricks Query Editor interface. On the left, the sidebar displays 'Bicycle_tracking' under 'Databases' and 'Bronze' under 'Tables'. A red arrow points from the 'Bronze' folder to the 'Table_0' entry in the main query editor window. The query editor contains the following KSQL code:

```

1 .alter table RawData (BikepointID:string,Street:string,Neighbourhood:string,Latitude:real,Longitude:real,//
2 No_Bikes:long,No_Empty_Docks:long,Timestamp:datetime) with (folder="Bronze")
3
4
5
6

```

2. Target Table Creation - Silver Layer

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

Implementation:

The screenshot shows the Databricks Query Editor interface. On the left, the sidebar displays 'Bicycle_tracking' under 'Databases' and 'Bronze' under 'Tables'. A red arrow points from the 'Bronze' folder to the newly added KSQL code in the main query editor window. The query editor contains the following KSQL code:

```

1 .alter table RawData (BikepointID:string,Street:string,Neighbourhood:string,Latitude:real,Longitude:real,//
2 No_Bikes:long,No_Empty_Docks:long,Timestamp:datetime) with (folder="Bronze")
3
4
5 //A destination table was created to store the transformed data
6
7 .create table TransformedData (BikepointID: int, Street: string, Neighbourhood: string, Latitude: real,//
8 Longitude: real, No_Bikes: long, No_Empty_Docks: long, Timestamp: datetime, BikesToBeFilled: long, Action: string) with (folder="Silver")

```

The status bar at the bottom right shows the timestamp: 2025-12-02 07:35 (UTC).

```
.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")
```

The screenshot shows the Eventhouse interface with the 'Queryset' tab selected. The left sidebar has a 'Bicycle_tracking' section with 'System overview', 'Databases', and 'Monitoring'. Under 'Tables', it shows 'Bronze', 'Silver', and 'TransformedData'. A red arrow points from the sidebar to the 'TransformedData' table in the main pane. The main pane displays a KQL query:

```

1 .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")
2
3
4
5 //A destination table was created to store the transformed data
6
7 .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")

```

The 'TransformedData' table is selected in the 'Table' list. The 'Schema' table details are shown in the bottom right:

TableName	Schema	DatabaseName	Folder	DocString
TransformedData	{"Name": "TransformedData", "Folder": "Silver", "OrderedColumns": [{"Name": "BikepointID", "Type": "System.Int"}, {"Name": "Street", "Type": "String"}, {"Name": "Neighbourhood", "Type": "String"}, {"Name": "Latitude", "Type": "Real"}, {"Name": "Longitude", "Type": "Real"}, {"Name": "No_Bikes", "Type": "Long"}, {"Name": "No_Empty_Docks", "Type": "Long"}, {"Name": "Timestamp", "Type": "Datetime"}, {"Name": "BikesToBeFilled", "Type": "Long"}, {"Name": "Action", "Type": "String"}]}	4ca552d2-bf2c-4c43-8dd5-23a5cabab220	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:

The screenshot shows the Eventhouse interface with the 'Database' tab selected. The left sidebar has a 'Database' section with 'Live view', 'New', 'Get data', 'Query with code', 'KQL Queryset', 'Notebook', 'Real-Time Dashboard', 'Data policies', and 'OneLake'. A red arrow points to the 'New' button. Another red arrow points to the 'Function' option in the dropdown menu. A third red arrow points to the 'Function' entry in the list. The main pane shows the 'Bicycle_tracking' database details and an 'Overview' section with a bar chart of data activity.

Fabric | Bicycle_tracking | Bicycle_tracking | ProjectEventstream | Search

Eventhouse Database **Queryset**

Copilot

Bicycle_tracking

- System overview
- Databases
- Monitoring

Search

SQL databases

- Bicycle_tracking
 - Bicycle_tracking_queryset
 - Tables
 - Bronze
 - Silver

edit

Run a query and explore the results here

Fabric | Bicycle_tracking | Bicycle_tracking | ProjectEventstream | Search

Eventhouse Database **Queryset**

Copilot

Bicycle_tracking

- System overview
- Databases
- Monitoring

Search

SQL databases

- Shortcuts
- Materialized views
- Functions
 - TransformRawData**
- Data streams

Created function

Run the code

```
.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:

The screenshot shows the Fabric interface with the 'Database' tab selected. In the left sidebar, under 'Databases', the 'Bicycle_tracking' database is selected. A red arrow points from step 1 to the 'New' button in the top navigation bar. Another red arrow points from step 2 to the 'Functions' section in the sidebar. Step 3 is indicated by a red arrow pointing to the search bar at the bottom of the sidebar. The main panel displays the KQL code for the 'TransformRawData' function:

```

1 .create-or-alter function TransformRawData()
2 RawData
3 | parse BikepointID with * "BikePoints_"
4 | extend BikesToBeFilled = No_Empty_Docks - No_Bikes
5 | extend Action = iff(BikesToBeFilled > 0, tostring(BikesToBeFilled), "NA")
6 |

```

The screenshot shows the Fabric interface with the 'Database' tab selected. In the left sidebar, under 'Tables', the 'Bicycle_tracking' table is selected. A red arrow points from step 1 to the 'Tables' section in the sidebar. Step 2 is indicated by a red arrow pointing to the 'Policy' section in the table details panel. The main panel displays the KQL code for the 'UpdatePolicy' and its configuration:

```

14 // the source table, such as extent tags and creation time, apply to the target table
15 // read more - https://aka.ms/updatepolicy
16
17 .alter table TransformedData policy update
18   [
19     {
20       "IsEnabled": true,
21       "Source": "RawData",
22       "Query": "TransformRawData()",
23       "IsTransactional": false,
24       "PropagateIngestionProperties": false
25     }
26   ]

```

PolicyName	EntityName	Policy	ChildEntities	EntityType
UpdatePolicy	[4ca552d2-bf2c-4c43-8dd5-23a5cabab220].[TransformedData]	[{"IsEnabled": true, "Source": "RawData", "Query": "TransformRawData()", "IsTransactional": false, "PropagateIngestionProperties": false}]		Table

.alter table TransformedData policy update

```

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

```

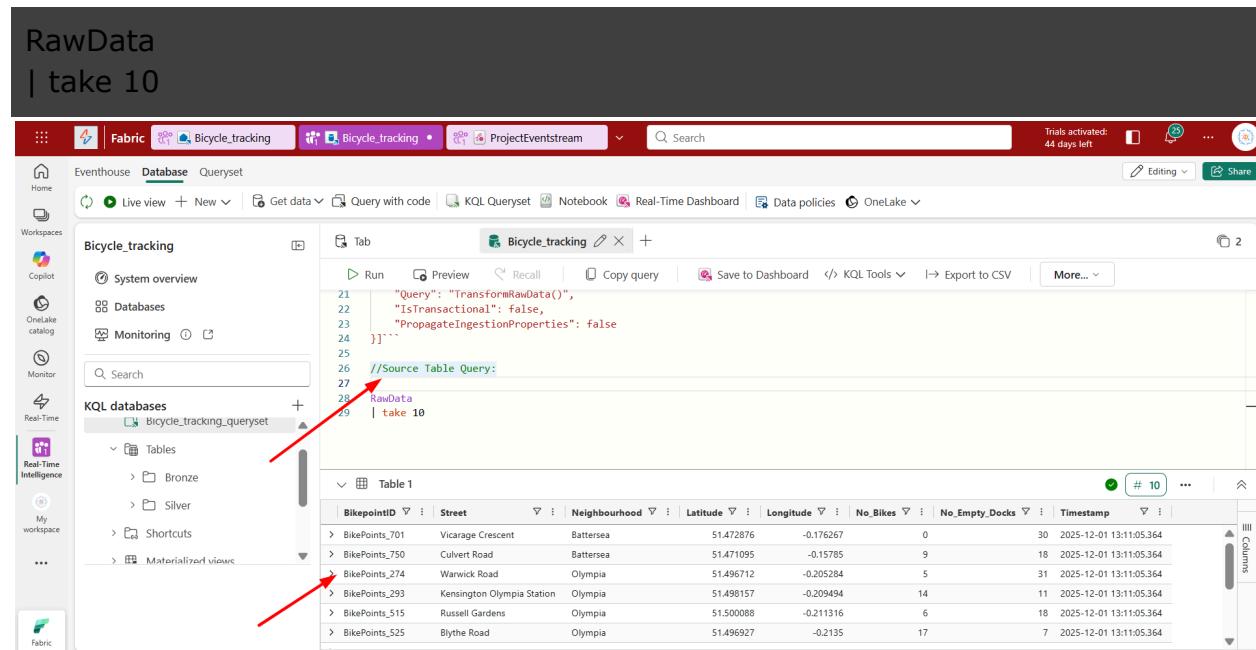
## Configuration Details:

- **.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:



The screenshot shows the DataFabric interface with the following details:

- RawData | take 10**: The title of the query.
- Fabric**, **Bicycle\_tracking**: The workspace and database selected.
- Eventhouse Database Queryset**: The type of view.
- Bicycle\_tracking**: The table being queried.
- System overview**: A link to system information.
- Databases**: A list of databases: **Bronze**, **Silver**, **Shortcuts**, and **Materialized views**.
- Monitoring**: A link to monitoring tools.
- Search bar**: A search bar for the interface.
- KQL Query** (Code View):
 

```

21 "Query": "TransformRawData()",
22 "IsTransactional": false,
23 "PropagateIngestionProperties": false
24 }]}
25
26 //Source Table Query
27
28 RawData
29 | take 10

```
- Table View** (Results):
 

BikepointID	Street	Neighbourhood	Latitude	Longitude	No_Bikes	No_Empty_Docks	Timestamp
BikePoints_701	Vicarage Crescent	Battersea	51.472876	-0.176267	0	30	2025-12-01 13:11:05.364
BikePoints_750	Culvert Road	Battersea	51.471095	-0.15785	9	18	2025-12-01 13:11:05.364
BikePoints_274	Warwick Road	Olympia	51.496712	-0.205284	5	31	2025-12-01 13:11:05.364
BikePoints_293	Kensington Olympia Station	Olympia	51.498157	-0.209494	14	11	2025-12-01 13:11:05.364
BikePoints_515	Russell Gardens	Olympia	51.500088	-0.211316	6	18	2025-12-01 13:11:05.364
BikePoints_525	Blythe Road	Olympia	51.496927	-0.2135	17	7	2025-12-01 13:11:05.364

## Transformed Table Query:

The screenshot shows the Databricks interface with a query editor and a results table.

**Query Editor:**

```
TransformedData
| take 10

25
26 //Source Table Query:
27
28 RawData
29 | take 10
30
31
32 //Transformed Table Query
33
34 TransformedData
35 | take 10
```

**Results Table:**

BikepointID	Street	Neighbourhood	Latitude	Longitude	No_Bikes	No_Empty_Docks	Timestamp	BikesToBeFilled	Action
701	Vicarage Crescent	Battersea	51.472876	-0.176267	1	29	2025-12-02 08:02:05.510	-28	
750	Culvert Road	Battersea	51.471095	-0.15785	10	17	2025-12-02 08:02:05.510	7	7
274	Warwick Road	Olympia	51.496712	-0.205284	14	22	2025-12-02 08:02:05.510	8	8
293	Kensington Olympia Station	Olympia	51.498157	-0.209494	16	9	2025-12-02 08:02:05.510	-7	NA
515	Russell Gardens	Olympia	51.500088	-0.211316	7	17	2025-12-02 08:02:05.510	10	10
525	Blythe Road	Olympia	51.496927	-0.2135	19	5	2025-12-02 08:02:05.510	-14	NA

## 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

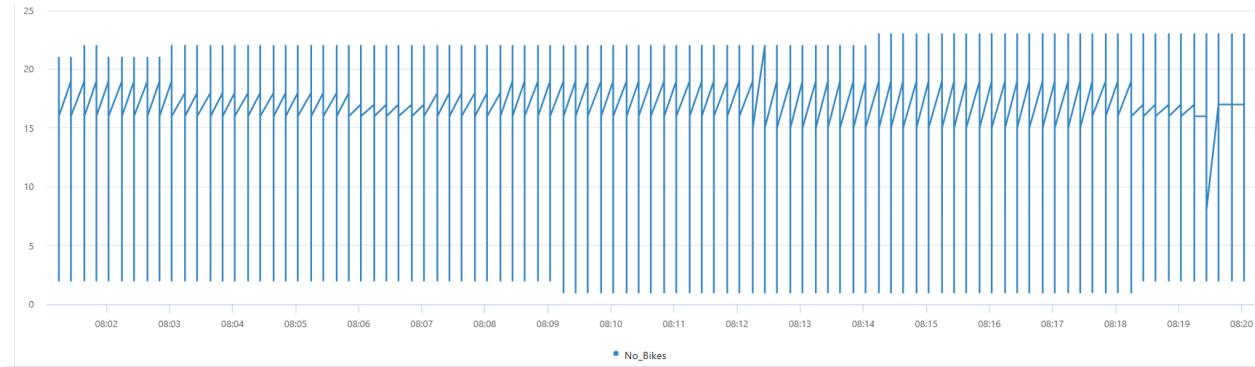
The screenshot shows the Kusto Query Editor interface. On the left, there's a sidebar with icons for Home, Workspaces, Copilot, OneLake catalog, Monitor, Real-Time, and My workspace. The main area has tabs for Eventhouse, Database, and Queryset. A tab titled "Bicycle\_tracking" is active. The query pane contains the following KQL code:

```
37 | take 10
38
39
40
41 //a time chart that shows the number of bikes in the Chelsea neighborhood
42
43 | TransformaData
44 | where BikepointID > 100 and Neighbourhood == "Chelsea"
45 | project Timestamp, No_Bikes
46 | render timechart
```

A red arrow points to the line "44 | TransformaData". Below the query pane is a visual representation of the data. It's a timechart titled "Table 1" showing the number of bikes ("No\_Bikes") over time from 08:02 to 08:20. The Y-axis ranges from 0 to 30. The data shows a high-frequency, low-amplitude oscillation between approximately 10 and 20 bikes per timestamp. The timestamp axis is labeled with minutes from 08:02 to 08:20. The bottom right of the visualization shows "2025-12-02 08:23 (UTC)" and "Done (1.214 s) # 1,045 records".

#### 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

The screenshot shows the Microsoft Fabric Data Explorer interface. On the left, the sidebar shows "Eventhouse" selected under "Workspaces". In the center, a tab for "Bicycle\_tracking" is open, showing a KQL query editor with the following code:

```

45 | project Timestamp, No_Bikes
46 | render timechart
47
48
49 //a materialized view
50
51 .create-or-alter materialized-view with (folder="Gold") AggregatedData on table TransformedData
52 {
53 TransformedData
54 | summarize arg_max(Timestamp,No_Bikes) by BikepointID
55 }

```

Red arrows point to the "Gold" folder icon in the sidebar and the "AggregatedData" table in the "Materialized views" section of the sidebar. Below the query editor, a table titled "Table\_0" shows the details of the materialized view:

Name	SourceTable	Query	MaterializedTo	LastRun	LastRunResult	IsHealthy
AggregatedData	TransformedData	TransformedData   summarize arg_max(Timestamp,No_Bikes) by BikepointID	2025-12-02 08:27:12.1361973	2025-12-02 08:28:06.5275504	Completed	true

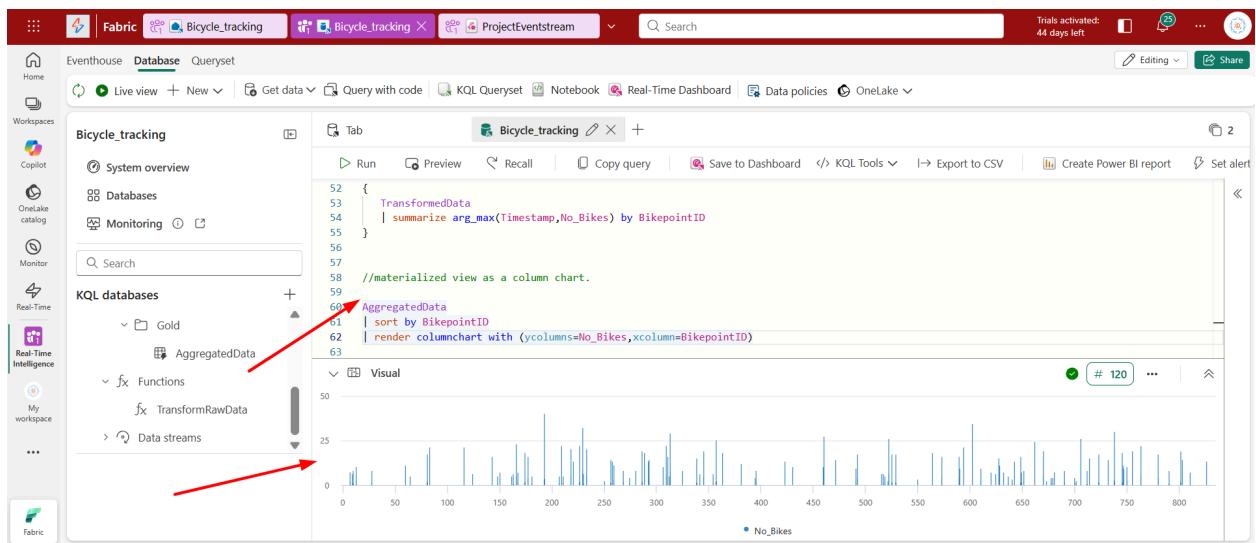
.create-or-alter materialized-view with (folder="Gold") AggregatedData on table  
TransformedData

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

## Technical Details

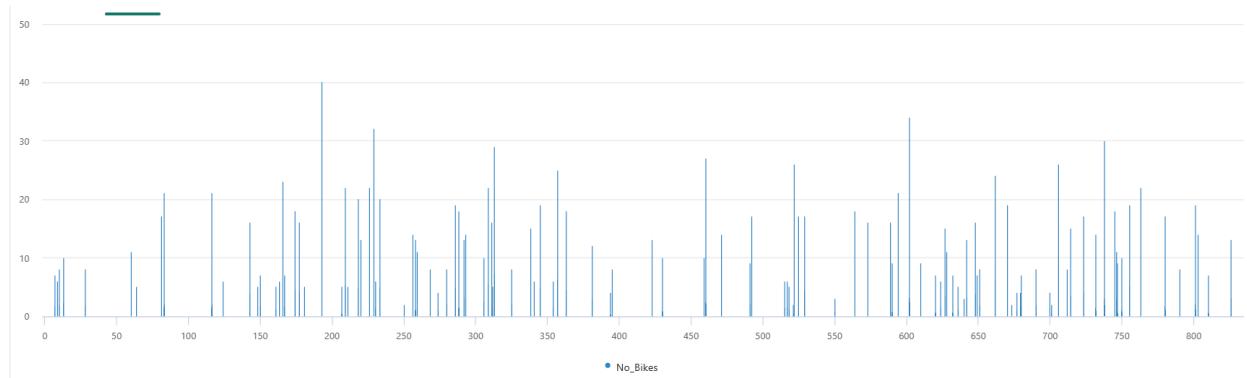
- **View Name:** AggregatedData
- **Storage Folder:** Gold (follows data lake organization principles)
- **Aggregation Logic:** Captures the latest bike count per station using arg\_max()
- **Performance Benefit:** Eliminates need to scan full dataset for current state queries

## 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
```

The screenshot shows the Databricks interface with the following details:

- Header:** Fabric, Bicycle\_tracking, ProjectEventstream, Search, Trials activated: 44 days left, Editing, Share.
- Sidebar:** Eventhouse, Database, Queryset, Workspaces (Copilot, OneLake catalog, Monitor, Real-Time Intelligence, My workspace), Real-Time.
- Current Workspace:** Bicycle\_tracking
- Code Editor:**

```
59
60 AggregatedData
61 | sort by BikepointID
62 | render columnchart with (ycolumns=No_Bikes,xcolumn=BikepointID)
63
64
65 //T-SQL Integration
66
67 SELECT top(10) *
68 FROM AggregatedData
69 ORDER BY No_Bikes DESC
```
- Result View:** Shows a table named 'Table 1' with columns BikepointID, Timestamp, and No\_Bikes. The data is as follows:

BikepointID	Timestamp	No_Bikes
193	2025-12-02 08:39:59.108	38
602	2025-12-02 08:40:04.861	34
738	2025-12-02 08:40:08.362	32
229	2025-12-02 08:40:08.362	32
313	2025-12-02 08:40:04.861	30
460	2025-12-02 08:40:04.861	27

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
```

The screenshot shows the Databricks interface with the following details:

- Top Bar:** Shows the project name "Fabric", workspace "Bicycle\_tracking", and a search bar.
- Left Sidebar:** Includes sections for Home, Workspaces (Copilot, OneLake catalog, Monitor, Real-Time, Real-Time Intelligence), and My workspace.
- Middle Panel:** A "Bicycle\_tracking" database is selected. The left sidebar shows the schema: System overview, Databases (AggregatedData), Monitoring, and a search bar. The right sidebar shows KQL databases: Gold (AggregatedData), Functions, TransformRawData, and Data streams.
- Right Panel:** The main area displays a KQL query editor with the following code:

```
68 FROM AggregatedData
69 ORDER BY No_Bikes DESC
70
71
72 //Convert a SQL query to KQL
73
74 explain
75
76 SELECT top(10) *
77 FROM AggregatedData
78 ORDER BY No_Bikes DESC
```

A red arrow points from the "Data streams" section in the sidebar to the "explain" keyword in the code.
- Output Pane:** Shows a table named "Table 1" with a single row:

```
Query
AggregatedData | project BikepointID, Timestamp, No_Bikes | sort by No_Bikes desc nulls last | take int(10)
```

A red arrow points from the "Table 1" row to the "Query" column.
- Bottom Right:** A JSON output pane displays the generated KQL:

```
1 "Query": AggregatedData
| project BikepointID, Timestamp,
| No_Bikes
3 | sort by No_Bikes desc nulls last
4 | take int(10)
5
```

A red arrow points from the "Query" label to the first line of the JSON.

### 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

The screenshot shows the Eventhouse interface with the 'Database' tab selected. On the left, there's a sidebar with various workspace options like Home, Workspaces, Copilot, OneLake catalog, Monitor, Real-Time, Real-Time Intelligence, and My workspace. The main area displays a 'Bicycle\_tracking' database. A red arrow points from the sidebar's 'Functions' section to the 'AggregatedData' function in the KQL code editor. Another red arrow points from the code editor to the resulting table view.

```

74
75 explain
76 SELECT top(10) *
77 FROM AggregatedData
78 ORDER BY No_Bikes DESC
79
80
81
82 AggregatedData
83 | project BikepointID, Timestamp, No_Bikes
84 | sort by No_Bikes desc nulls last
85 | take int(10)
86

```

BikepointID	Timestamp	No_Bikes
193	2025-12-02 09:32:35.423	39
602	2025-12-02 09:32:29.673	35
738	2025-12-02 09:32:31.907	31
460	2025-12-02 09:32:29.673	29
313	2025-12-02 09:32:29.673	28

## 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

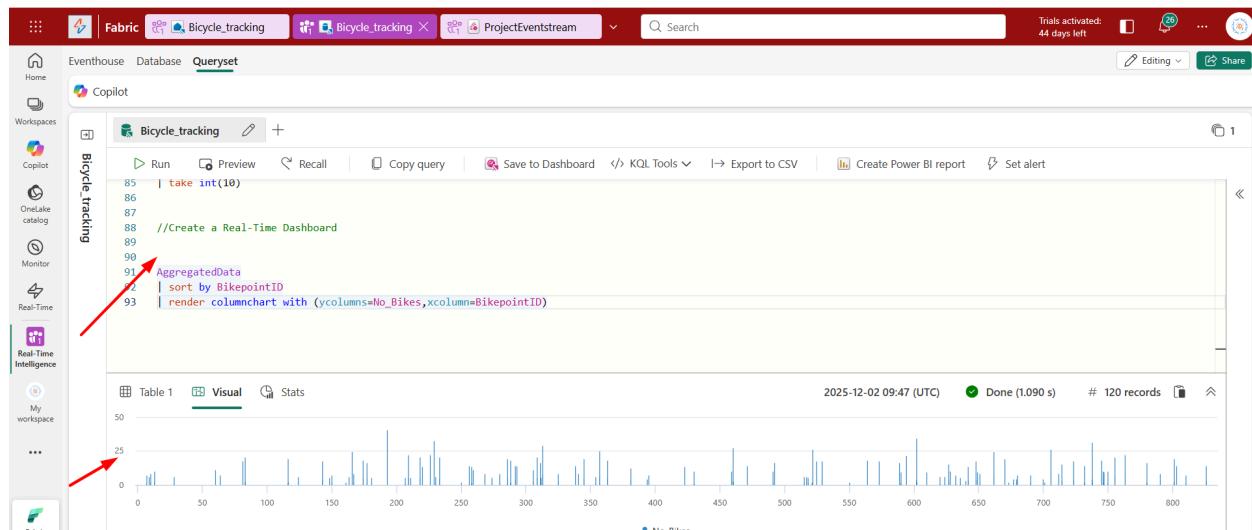
# Real-Time Dashboard Implementation for Streaming Data Visualization

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:



A screenshot of the Kusto Query Editor interface. The top navigation bar shows 'Fabric' and 'Bicycle\_tracking' as the current database. The left sidebar includes 'Eventhouse', 'Database', 'Queryset' (which is selected), 'Copilot', 'OneLake catalog', 'Monitor', 'Real-Time', 'Real-Time Intelligence', and 'My workspace'. The main area displays a KQL query:`85 | take int(10)
86
87
88 //Create a Real-Time Dashboard
89
90
91 AggregatedData
92 | sort by BikepointID
93 | render columnchart with (ycolumns=No_Bikes,xcolumn=BikepointID)`

Below the query, there's a preview section showing 'Table 1' with a column chart. The chart has 'No\_Bikes' on the y-axis (ranging from 0 to 50) and 'BikepointID' on the x-axis (ranging from 0 to 800). A red arrow points to the 'AggregatedData' part of the query, and another red arrow points to the column chart in the preview.

```
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)*

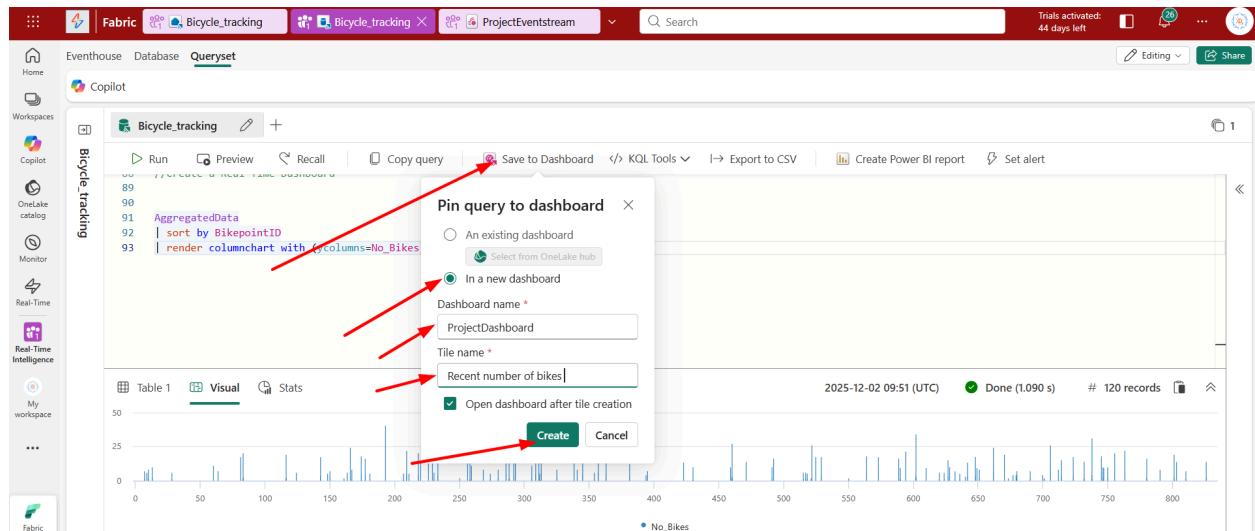
## 2. Dashboard Creation

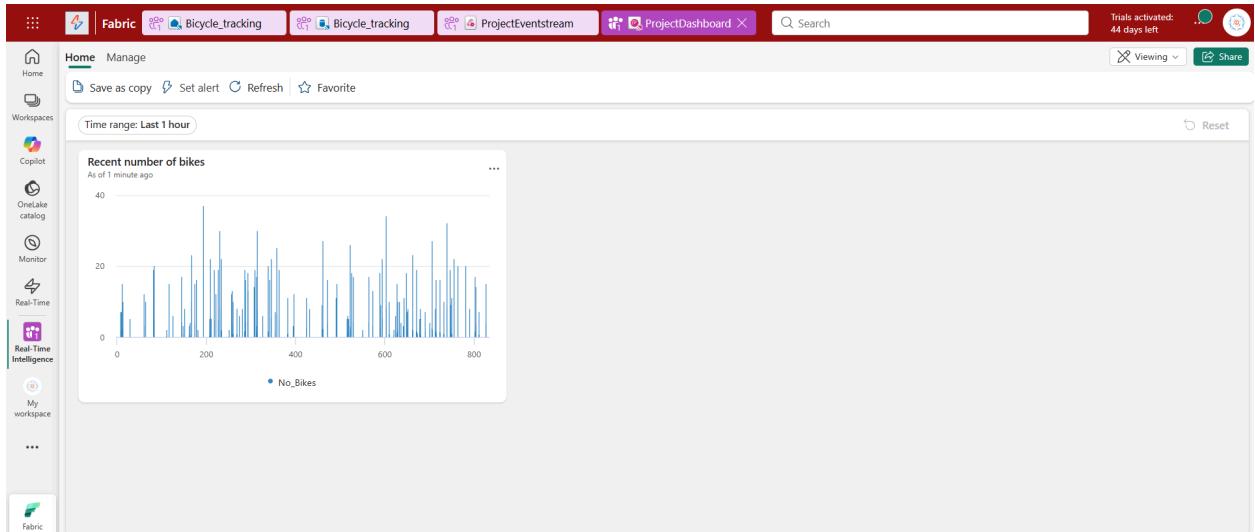
### Configuration Parameters:

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

### Creation Process:

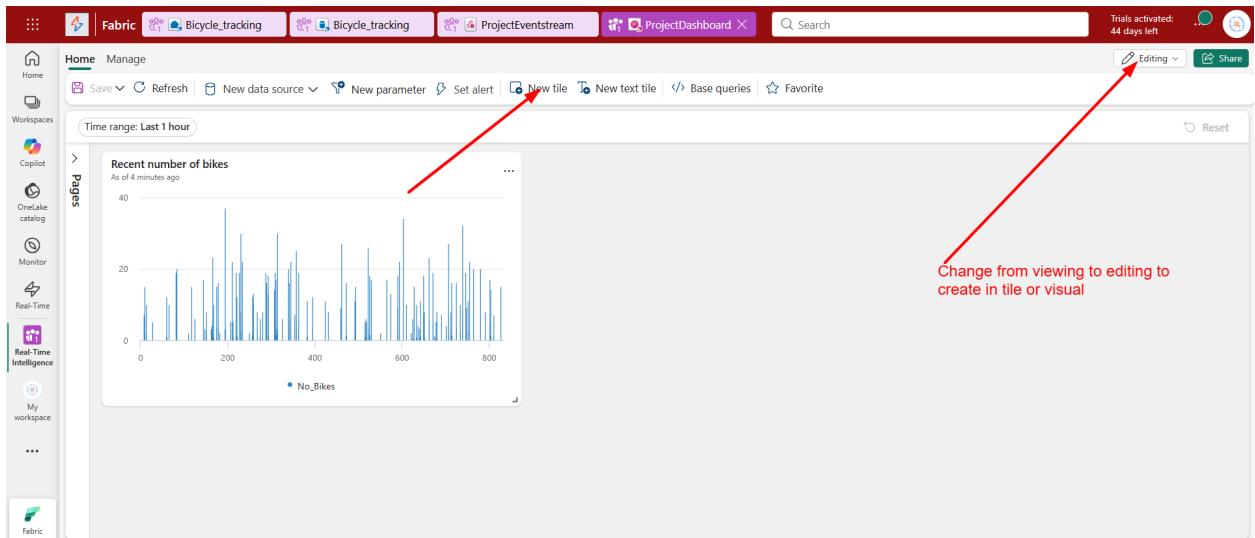
1. Execute the visualization query in the KQL queryset
2. Select "Save to dashboard"
3. Configure the dashboard name and Tile name
4. Confirm creation to initialize the dashboard





### 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)
```

The screenshot shows the Fabric Data Explorer interface. On the left, there's a sidebar with various workspace and real-time intelligence icons. The main area has tabs for 'Fabric', 'Bicycle\_tracking', and 'ProjectEventstream'. A search bar at the top right shows 'Trials activated: 44 days left'. Below the tabs is a 'Run' section with two numbered steps: 1. RawData and 2. | where Timestamp > ago(1h). A red arrow points from the number 2 to the 'Run' section. Another red arrow points from the number 2 to the results table below. The results table displays data for 36,000 records, with columns including BikepointID, Street, Neighbourhood, Latitude, Longitude, No\_Bikes, No\_Empty\_Docks, and Timestamp. The timestamp for the first record is 2025-12-02 09:05:29.479.

## 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:

This screenshot is similar to the previous one, showing the Fabric Data Explorer interface. It features the same sidebar, tabs, and search bar. The 'Run' section shows the same two steps: 1. RawData and 2. | where Timestamp > ago(1h). A red arrow points from the 'Add visual' button in the results table header to the table itself. The results table shows the same 36,000 records with the same columns and timestamp values as the previous screenshot.

The screenshot shows the Fabric Data Explorer interface. At the top, there are tabs for 'Fabric', 'Bicycle\_tracking', 'ProjectEventstream', and 'ProjectDashboard'. The main area displays a query run titled 'Bicycle\_tracking' with the following code:

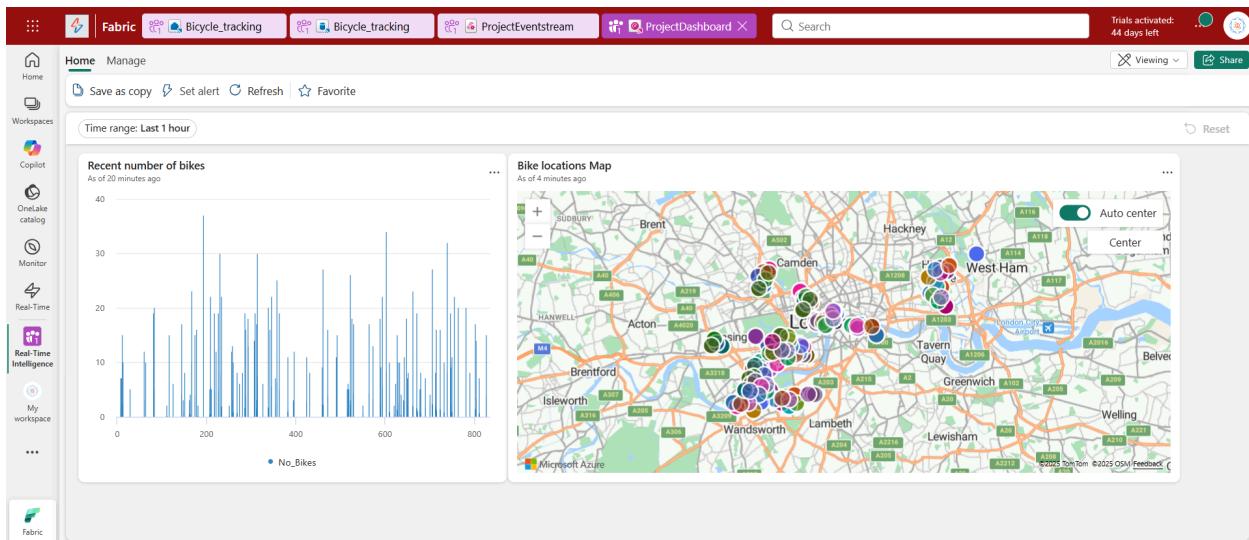
```
1 RawData
2 | where Timestamp > ago(1h)
3
```

The results table shows data from 36,000 rows, with columns including BikepointID, Street, Neighbourhood, Latitude, Longitude, No\_Bikes, No\_Empty\_Docks, and Timestamp. The table includes sorting and filtering options.

To the right, a 'Visual formatting' panel is open, allowing configuration for a new tile. The tile name is set to 'New tile', and the visual type is selected as 'Table'. Other options like 'URLs' and 'Apply link on column' are also visible.

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.

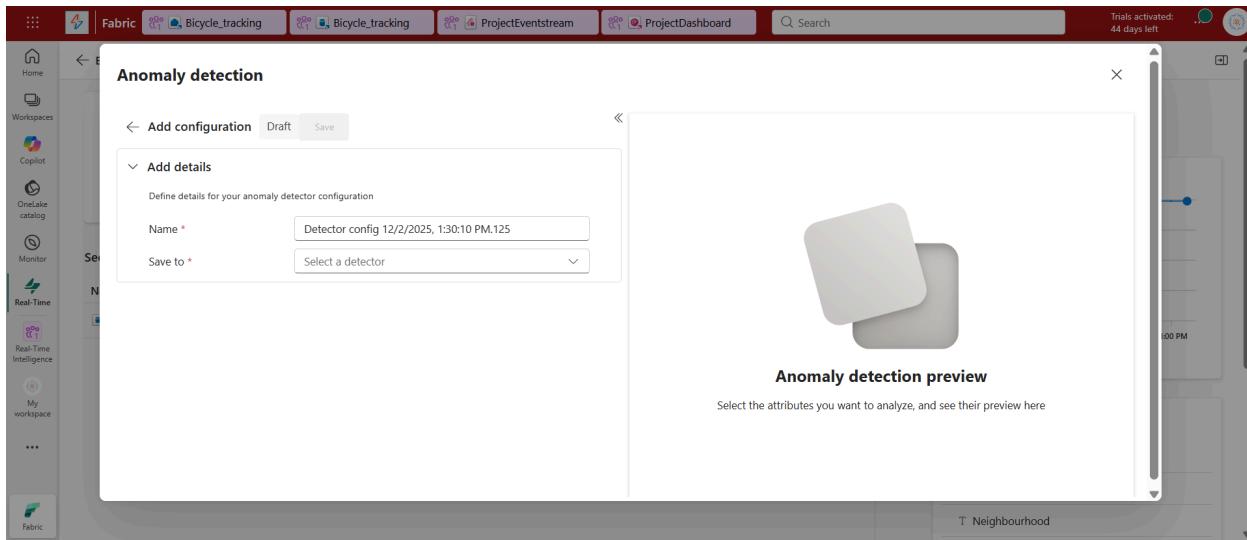
- *Implement real-time anomaly detection on streaming data infrastructure*
- *Monitor empty dock availability patterns across multiple bike stations*
- *Create automated alerts for unusual operational conditions*
- *Build scalable anomaly detection models for time-series data analysis*

## 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.

The screenshot shows the Databricks Real-Time hub interface. In the top navigation bar, there are several tabs: Fabric, Bicycle\_tracking, ProjectEventstream, ProjectDashboard, and a search bar. Below the tabs, there are buttons for Back to Real-Time hub, Explore data, Open KQL Database, Endorse, Detect anomalies (which is highlighted by a red arrow), and Create real-time dashboard (Preview). The main content area displays the 'TransformedData' table details. It includes sections for 'Owner' (Robert Ssebambulidde) and 'Type' (KQL table). There is also a 'See what already exists' section showing a table with one row for 'Bicycle\_tracking'. To the right of the table, there is an 'Insights' section with a line chart showing the number of rows over time, and a 'Columns' section listing '# BikepointID', 'T Street', and 'T Neighbourhood'.



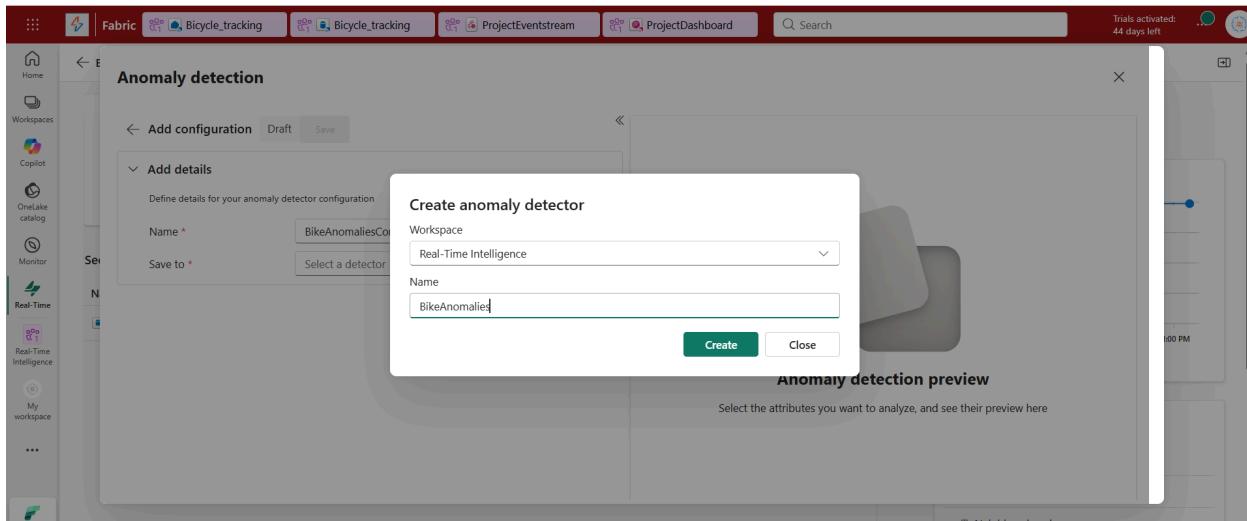
## 2. Anomaly Detector Setup

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
<b>Value to watch</b>	No_Empty_Docks	Primary metric being monitored for anomalies
<b>Group by</b>	Street	Enables per-location anomaly detection
<b>Timestamp</b>	Timestamp	Time-series alignment for pattern recognition

The screenshot displays the 'Anomaly detection' configuration screen within a real-time intelligence application. The main configuration panel includes fields for 'Name' (set to 'BikeAnomaliesConfiguration'), 'Save to' (set to 'BikeAnomalies'), and 'Select attributes'. Under 'Select attributes', the 'Value to watch' is set to 'No\_Empty\_Docks', 'Group by' is set to 'Street', and 'Timestamp' is selected. A prominent green 'Run analysis' button is located at the bottom of this panel. To the right, a preview window titled 'Anomaly detection preview' shows a visual representation of the data being analyzed, consisting of two overlapping gray squares.

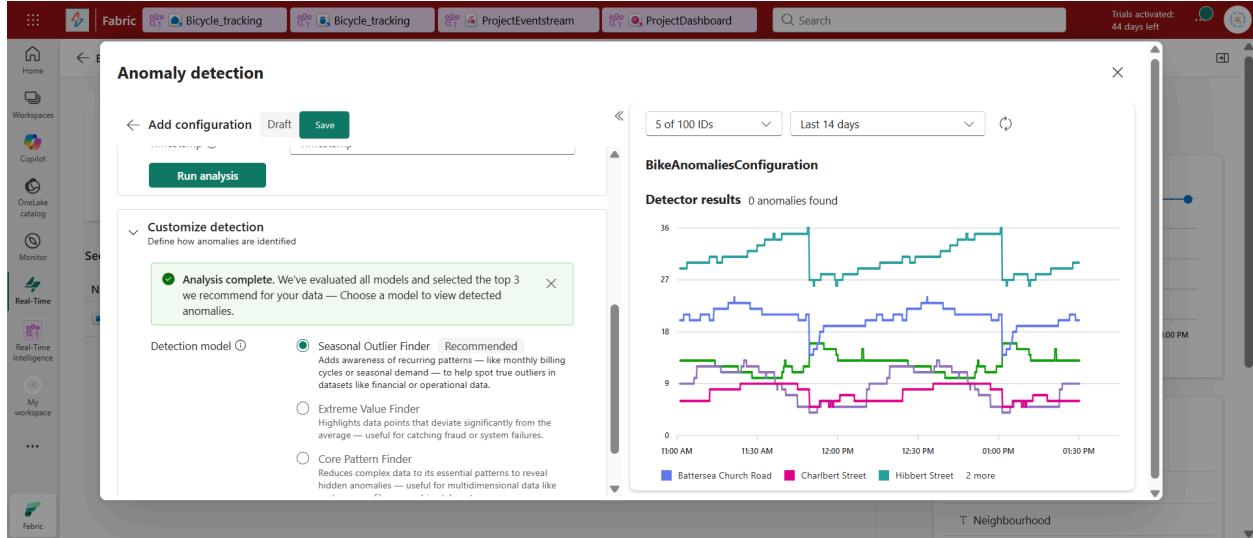
## 4. Model Training and Analysis

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