# Microsoft Fabric: Crypto Currency Case Study

By Krishan Patel

# Map out the product

## Goal:

To develop a Live Cryptocurrency Tracker using Microsoft Fabric.

## Essential Features:

- **Real-Time Price Updates:** Fetch the latest price of each cryptocurrency with a latency of under 1 minute.
- **Flexible Time Slicing:** Users should be able to select the time period they want to view.
- **Adaptive Candle Aggregation:** For larger time ranges, the tracker should automatically adjust the candle aggregation, so users can see clear trends without overcrowding the display.

## Key Design Principals:

- **Upstream Transformations:** Ensure any data transformations are performed as far upstream as possible.
- **Scalability:** Design the solution with the ability to efficiently handle increasing amounts of data without compromising performance.
- **Optimize Performance:** Ensure the report is fast and responsive

# Data source

To build a Cryptocurrency Tracker, we need a **REST API** to fetch real-time market data.

For this we will use the Binance Kline REST API: https://binance-docs.github.io/apidocs/spot/en/#kline-candlestick-data.

### Kline/Candlestick Data

`GET /api/v3/klines`

Kline/candlestick bars for a symbol.
Klines are uniquely identified by their open time.

**Weight(IP):** 2

**Parameters:**

| Name | Type | Mandatory | Description |
|---|---|---|---|
| symbol | STRING | YES | |
| interval | ENUM | YES | |
| startTime | LONG | NO | |
| endTime | LONG | NO | |
| timeZone | STRING | NO | Default: 0 (UTC) |
| limit | INT | NO | Default 500; max 1000. |

- If startTime and endTime are not sent, the most recent klines are returned.
- Supported values for `timeZone`:
  - Hours and minutes (e.g. `-1:00`, `05:45`)
  - Only hours (e.g. `0`, `8`, `4`)
  - Accepted range is strictly [-12:00 to +14:00] inclusive
- If `timeZone` provided, kline intervals are interpreted in that timezone instead of UTC.
- Note that `startTime` and `endTime` are always interpreted in UTC, regardless of `timeZone`.

```
[
  [
    1499040000000,      // Kline open time
    "0.01634790",       // Open price
    "0.80000000",       // High price
    "0.01575800",       // Low price
    "0.01577100",       // Close price
    "148976.11427815",  // Volume
    1499644799999,      // Kline Close time
    "2434.19055334",    // Quote asset volume
    308,                // Number of trades
    "1756.87402397",    // Taker buy base asset volume
    "28.46694368",      // Taker buy quote asset volume
    "0"                 // Unused field, ignore.
  ]
]
```

To produce the tracker, we will use the following key columns:

- Open, High, Low, and Close prices
- Kline open time
- Kline close time*
- Volume and Quote asset Volume

Other columns will be disregarded to optimise data usage.

* Each kline has a designated *EndTime*, but the latest candle remains open until its period ends. To get the most accurate *EndTime* for the latest candle, we will need to manually calculate it by storing the time the API was called.

# Option 1. Incremental Refresh

## Approach:

### Write Historical Data (Fabric Notebook) into a Delta Table

Retrieve 1-minute trading data from Binance in batches of 1,000 rows, from the earliest to the most recent timestamp, and write this data into a Delta table within a Lakehouse.

### Append Incremental Data (Fabric Notebook)

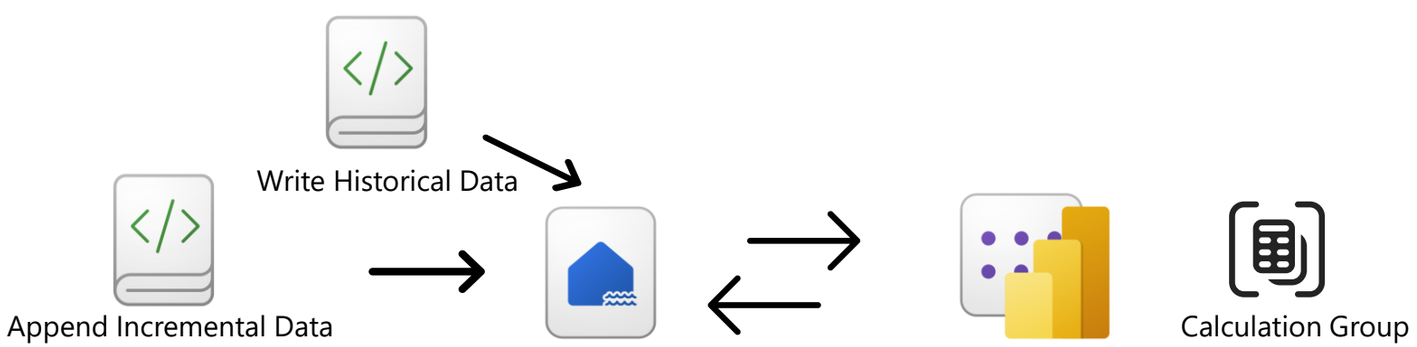
Continuously append new 1-minute interval data for each cryptocurrency, ensuring that the dataset stays up-to-date.

### Manage Small File Problem

A consequence of running frequent appends is the creation of many small files, which can degrade query performance and storage efficiency; using the OPTIMIZE command with V-order at regular intervals helps compact these small files and mitigate the issue.

### Use Power BI Techniques

Use Field Parameters to let users dynamically switch between date dimensions (e.g., daily, weekly, monthly) and then deploy Calculation Groups to adjust measures based on the selected time period.



# Pros and Cons of Incremental Refresh

## Pros:

### Improved Notebook Performance

Faster data refresh by processing only new or modified data, reducing resource consumption.

### Other use cases

The 1-minute trading data can be used outside of Power BI for data science applications, such as predictive modelling, anomaly detection, and advanced analytics.

## Cons:

### Complex DAX Aggregations:

Aggregations must be performed in DAX measures, which can increase report complexity and may impact performance, especially with large datasets.

### Data Loading Challenge:

We need to load 60 million rows of data, as Publish to Web doesn't support Direct Lake, leading to potential performance issues and increased load times.

# Option 2. Pre-Aggregated Time Periods

| Time Period | Aggregation Level |
|---|---|
| 1H | 1m |
| 4H | 5m |
| 1D | 15m |
| 1W | 1H |
| 1M | 4H |
| YTD | YTD |
| 1Y | 1D |
| MAX | 1W |

Instead of using DAX for complex calculations, we can import data that's already aggregated (e.g., hourly, daily) for a specific period.

## Approach:

### Data Fetching

Using the Binance API, we define Time Periods and set the appropriate Aggregation Level parameters to fetch coin data, adjusting the data frequency based on the recency of the time period, as shown in the table.

### Concurrent Retrieval

We call the API simultaneously for each time period of a coin using a thread pool, allowing multiple API requests to run in parallel. This ensures that the close prices across time periods are consistent, as even a slight delay in seconds can result in different prices.

### Data Processing

The results are processed into DataFrames, combined into a single dataset, and organized for easy filtering by time period and coin, eliminating the need for complex DAX queries.

# Pros and Cons of Pre-Aggregated Time Periods

## Pros:

### Improved Report Performance

By importing pre-aggregated data, we eliminate the need for complex DAX calculations, leading to faster report rendering and smoother user experience.

### Reduced Resource Consumption

Pre-aggregating data minimizes the computational load during report generation by handling calculations in advance, reducing memory and CPU usage and improving overall efficiency.

## Cons:

### Loss of Granularity

Aggregating historical data can completely obscure critical market trends, hiding significant price shifts and distorting the true picture of long-term market behavior.

In my opinion, the Pre-Aggregated option is the best fit, as it meets the requested features and stays true to the core design principles.