Time Series Exploration mit Azure Data Explorer

Tim Niklas Vinkemeier

Senior Software Engineer @ prodot

- Cloud, IoT, Web

@TimVinkemeier

github.com/TimVinkemeier

Was ist Azure Data Explorer?

aka „ADX“ aka „Kusto“

Was ist Azure Data Explorer?

Überblick

- Gemanageter, horizontal skalierbarer Dienst zur explorativen Datenanalyse
- Startete vor mehr als 8 Jahren als MS-internes Projekt „Kusto“
- Optimiert für große Mengen von Zeitreihendaten (strukturiert, semistrukturiert, unstrukturiert)
  - Telemetrie
  - Sensordaten
  - etc.
- Üblicherweise für append-only Daten mit begrenztem Zeithorizont

Was ist Azure Data Explorer?

Prinzip

- Trennung von Storage und Compute
- Daten werden automatisch in Shards aufgeteilt
  - Column-Store basiert, komprimiert, immutable
  - Persistente Datenablage in Azure Storage
  - Verteilung in SSD- und Memory-Caches der Knoten
- Logische Datenstrukturierung ist relational angelehnt
  - Datenbanken > Tabellen > Spalten mit Datentypen
- Automatische Indexierung aller Daten
- Automatisches Cache-Management mit Bevorzugung aktuellerer Daten

Was ist Azure Data Explorer?

Nutzungsszenarien und Beispielanwendungen

- IoT, Big Data Logging, SaaS-Anwendungen
- Explorative Datenanalyse inkl. Visualisierung und Dashboards
- Verarbeitung von Echtzeit-Streamingdaten mit geringer Latenz
- Datenhaltung für Reportingzwecke mit anderen Visualisierungstools (Excel, PowerBI, Grafana, Kibana, etc.)

- Etliche Azure-Dienste basieren auf Data Explorer
  - Log Analytics/Application Insights
  - Monitor
  - Time Series Insights
  - Defender ATP
  - Sentinel

Was ist Azure Data Explorer *nicht*?

Abgrenzung zu anderen Nutzungsszenarien

- Data-Warehouse (wie z.B. Azure Synapse Analytics oder DataBricks)
  - DW folgt üblicherweise einem klassischen ETL-Ablauf mit relativ langer Latenz
  - ADX nutzt Schema-on-read, optimiert für Echtzeit-Analysen ohne Datenvorbereitung
- Geeignet für transaktionale Datenhaltung oder Speicherung von Applikationsdaten
  - ADX ist nicht transaktional und nicht für Updates ausgelegt
- Plattform für Machine Learning
  - ADX ist nicht als Trainingsplattform ausgelegt, kann aber genutzt werden, um Daten gegen trainierte Modelle zu evaluieren
- Optimiert für die reine Echtzeitberechnung von Aggregierungen, etc. (wie z.B. Apache Spark)
  - ADX evaluiert frei definierte Abfragen statt vordefinierter Aggregierungen

Schritte zur Datenanalyse

Data Ingestion

Überblick

- Möglich über verschiedene Integrationen
  - Pipelines (Event Grid, Event Hub, IoT Hub, Data Factory)
  - Connectors and Plugins (Logstash, Kafka, Power Automate, Apache Spark)
  - SDKs
  - Tools (One-Click Ingestion, LightIngest CLI)
  - KQL (Inline, from query, from storage)
- Passiert parallel auf allen Nodes, jede Node schreibt unabhängige Shards
  - Horizontale Skalierung der Ingestion-Performance
- Batch vs. Streaming-Ingestion

Querying

Tools

- Nutzung der proprietären Abfragesprache „KQL“ (Kusto Query Language)
- Verfügbar über diverse Schnittstellen und Tools
  - Web UI
  - Kusto.Explorer
  - Azure Data Studio
  - SDKs
- Automatischer Scale-Out auf alle Nodes

Demo

Zusammenfassung

- ADX ermöglicht die einfache Analyse großer, unstrukturierter Datenmengen
- Data-Ingestion ist aus diversen Quellen möglich
- KQL stellt vielfältige Funktionen zur Analyse und Voraussage bereit
    - Datenschnitte
    - Zeitreihenanalyse
    - Trendanalyse und Anomaly Detection
    - Geo-Analyse

Insgesamt einfach ein mächtiges und cooles Werkzeug :D

Links

- Azure Data Explorer Dokumentation
  - What is Azure Data Explorer? | Microsoft Docs
- Technical White Paper
  - Azure Data Explorer technical white paper | Microsoft Azure
- Einführung von Vincent Lauzon
  - Azure Data Explorer (Kusto) | Vincent-Philippe Lauzon's (vincentlauzon.com)
- On-Time Dataset bei Kaggle
  - Airline Delay and Cancellation Data, 2009 - 2018 | Kaggle
- Ergänzende Daten zu Airlines, etc.
  - OpenFlights: Airport and airline data

Vielen Dank!

Fragen? Gern ☺

Abfragen zum Nachmachen I

```
// Total data count
flights
| count

// Data Count by year
flights
| where datetime_part("year", Date) > 2000
| summarize count() by datetime_part("year", Date)

// Example Data
flights
| take 100

// Flights by month
flights
| where datetime_part("year", Date) > 2000
| summarize count() by datetime_part("month", Date)
| render barchart

// Flights by airline with full names
flights
| join kind=leftouter airlines on $left.CarrierCode == $right.IATA
| summarize count() by Name
| render piechart

// Cancelled flights by year
flights
| where datetime_part("year", Date) > 2000 and IsCancelled == 1
| summarize count() by datetime_part("year", Date)
| render barchart

// Cancelled flights by month
flights
| where datetime_part("year", Date) > 2000 and IsCancelled == 1
| summarize count() by datetime_part("month", Date)
| render barchart
```

Abfragen zum Nachmachen II

```
// Flights with more than 20 minutes arrival delay vs all by airline
flights
| extend IsSignificantlyDelayed = ArrivalDelayMinutes > 20
| summarize NotDelayed = countif(IsSignificantlyDelayed == 0), Delayed = countif(IsSignificantlyDelayed == 1) by CarrierCode
| render columnchart

// Flights with more than 20 minutes arrival delay as percentage by airline
flights
| extend IsSignificantlyDelayed = ArrivalDelayMinutes > 20
| summarize Percentage = (countif(IsSignificantlyDelayed == 1)/toreal(countif(IsSignificantlyDelayed == 0))) * 100 by CarrierCode
| order by CarrierCode asc
| render columnchart

// Time Chart
let start = datetime(2018-01-01);
let end = datetime(2019-01-01);
flights
| make-series numberOfFlights = count() on Date from start to end step 1d
| render timechart

// Period detection
let start = datetime(2017-01-01);
let end = datetime(2019-01-01);
flights
| make-series numberOfFlights = count() on Date from start to end step 1d
| project series_periods_detect(numberOfFlights,0,10,5)

// Trend Analysis 2017-2018
let start = datetime(2017-01-01);
let end = datetime(2019-01-01);
flights
| make-series numberOfFlights = count() on Date from start to end step 1d
| extend (baseline, seasonal, trend, residual) = series_decompose(numberOfFlights,-1,"linefit")
| render timechart
```

Abfragen zum Nachmachen III

```
// Anomaly detection for 2018
let start = datetime(2018-01-01);
let end = datetime(2019-01-01);
flights
| make-series numberOfFlights = count() on Date from start to end step 1d
| extend (anomalies, score, baseline) = series_decompose_anomalies(numberOfFlights, 2)
| render anomalychart with(anomalycolumns=anomalies)

// Forecast for first month of 2019
let start = datetime(2018-01-01);
let end = datetime(2019-01-01);
let horizon=31d;
flights
| make-series numberOfFlights=count() on Date from start to end+horizon step 1d
| extend forecast = series_decompose_forecast(numberOfFlights, toint(horizon/1d))
| render timechart

// Geo functions (Kusto.Explorer only)
airports
| where Country == "United States"
| summarize count() by hash=geo_point_to_geohash(Longitude, Latitude, 3)
| extend cell=geo_geohash_to_central_point(hash)
| project cell, hash, count_
| render piechart with (kind=map)

airports
| where Country == "United States"
| summarize count() by hash=geo_point_to_s2cell(Longitude, Latitude, 6)
| extend cell=geo_s2cell_to_central_point(hash)
| project cell, hash, count_
| render piechart with (kind=map)

airports
| where Country == "United States"
| summarize count() by hash=geo_point_to_h3cell(Longitude, Latitude, 3)
| extend cell=geo_h3cell_to_central_point(hash)
| project cell, hash, count_
| render piechart with (kind=map)
```