Databricks / Photon

ROBIN SCHLAURI – NINO ZANITTI – LINUS SZOKODY

Inhalt

- Übersicht Databrick
- Databrick optimizations
- Photon
- Photon Tests

Databricks

2013 von Apache Spark entwicklern gegründete amerikanische Firma

“Eine einheitliche Platform mit welcher Data Scientists, Data Engineers und Analysten zusammenarbeiten können, um Wertschöpfung aus Daten zu generieren”

Basierend auf Apache Spark mit vielen Open Source Projekten

- Apache Spark
- Delta Lake
- Mlflow

Databricks Features

Hauptpunkte:

- Erstellung und Konfiguration von Server-Clustern
- Verbindung zu verschiedenen Dateisystemen
- Programmierschnittstellen für Python, Scala und SQL
- Interaktive und kollaborative Workspaces
- Sehr schneller start möglich durch managed umgebungen von Cloud providern (lokale installation nicht möglich)

User Interface

Workspace

- Notebooks und andere Files für die Zusammenarbeit

Catalog

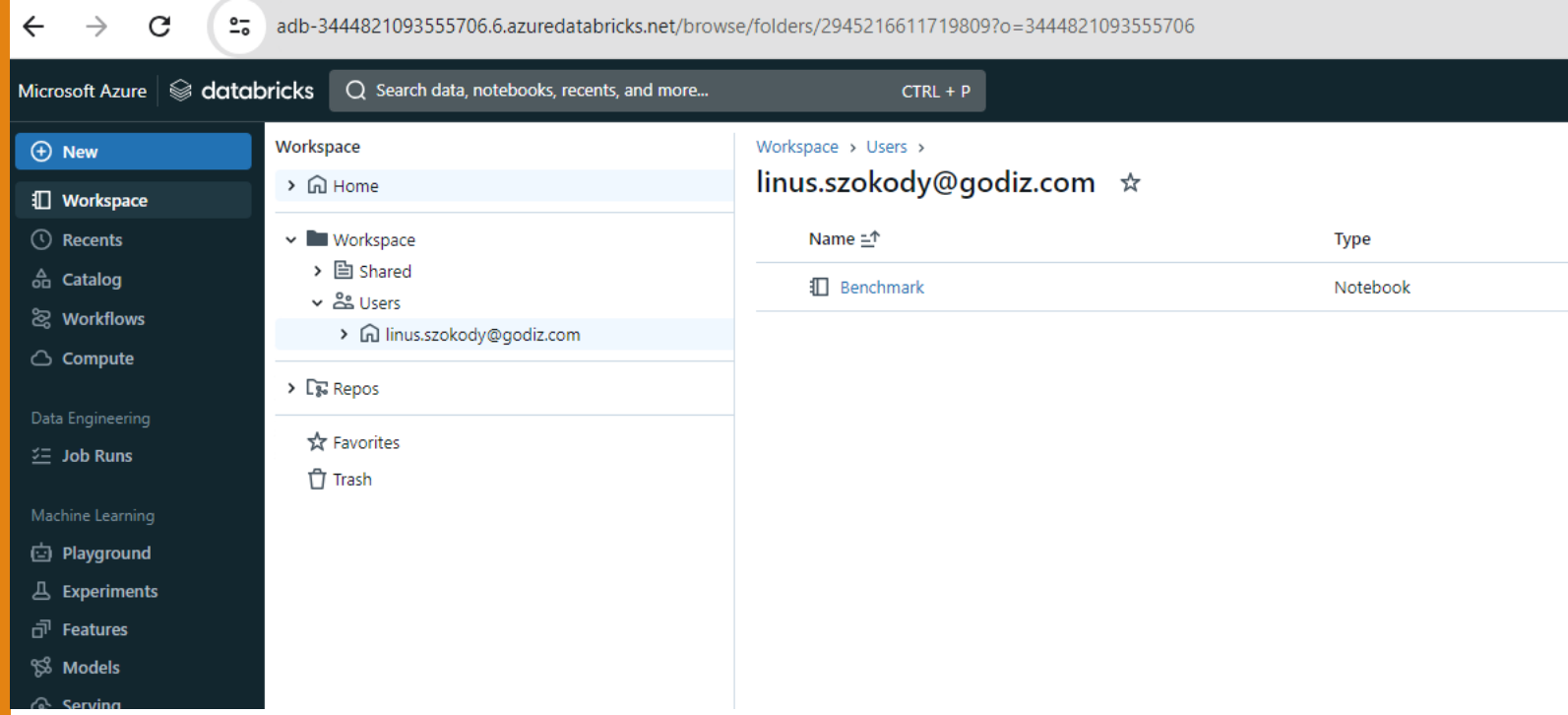
- Delta Lake

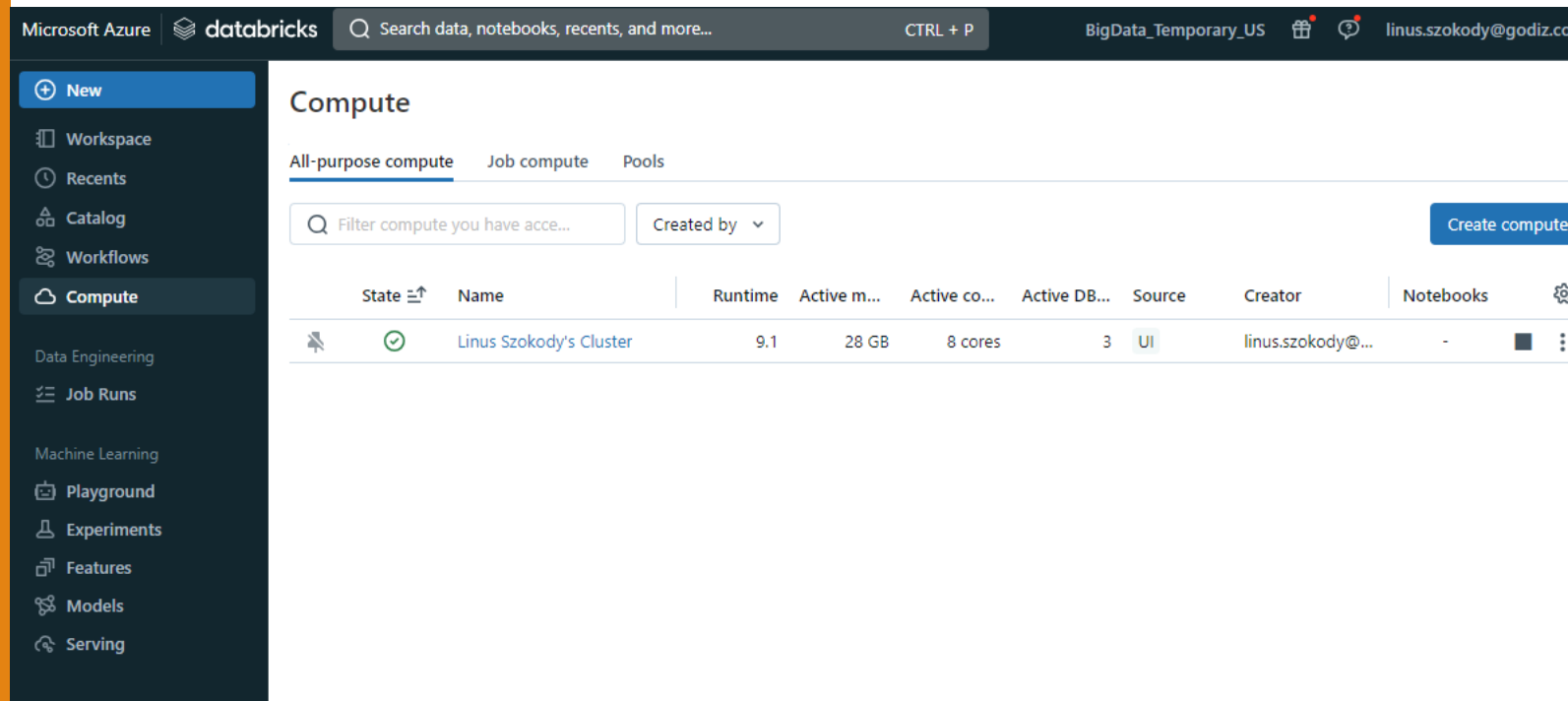
Workflows

- Automations -> z.B. Scheduled Jobs

Compute

- Cluster und Jobs Konfigurieren





Databricks optimizations

Starke integration in Cloud (Microsoft Azure, Amazon AWS, Google Cloud)

Ermöglicht Features und Verbesserungen die sonst nicht möglich wären:

- I/O von anderen Services können perfekt abgestimmt werden
  - z.B. Azure Databricks + Azure Data Lake
- optimiertes Caching

- Sowas wie “**Photon**” ist möglich

Photon

"Photon is a high-performance Databricks-native vectorized query engine that runs your SQL workloads and DataFrame API calls faster to reduce your total cost per workload." (Databricks Dokumentation)

Funktionsweise:

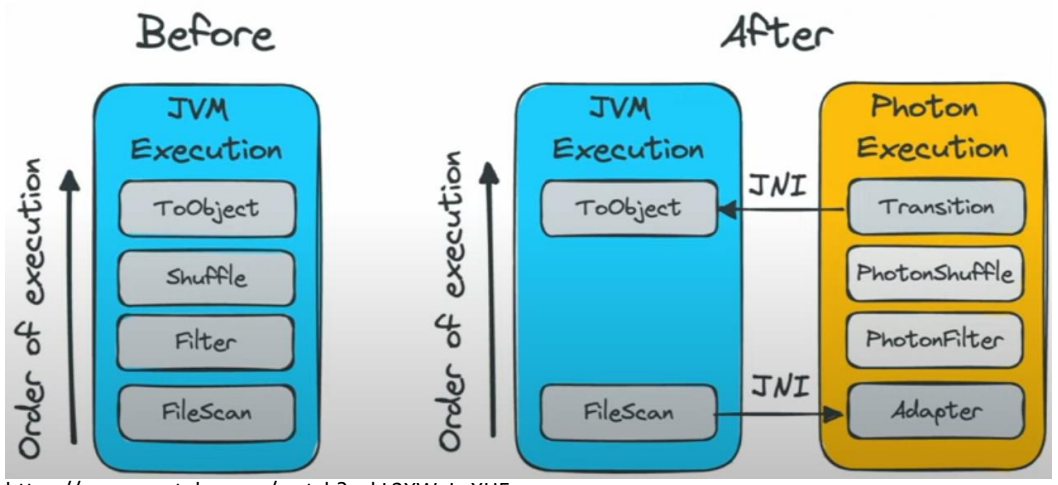
- Photon aktiviert -> Photon bearbeitet die Abfrage (wenn schneller)
- Von Photon nicht unterstützt/langsamer mit Photon -> Spark übernimmt

Grund:

- Spark engine läuft auf einer JVM (Java Virtual Machine) (Scala)
- Durch C++ soll JVM Overhead verringert werden
- Photon arbeitet immer Vektorisiert

Integration:

- Standardmässig in Databricks aktiviert
- Kompatibel mit Apache Spark-APIs (**funktioniert mit bestehendem Code**)



https://www.youtube.com/watch?v=kL2XWgLeXHE

Photon – Operatoren, Ausdrücke und Datentypen

Operators

- Scan, Filter, Project
- Hash Aggregate/Join/Shuffle
- Nested-Loop Join
- Null-Aware Anti Join
- Union, Expand, ScalarSubquery
- Delta/Parquet Write Sink
- Sort
- Window Function

Expressions

- Comparison / Logic
- Arithmetic / Math (most)
- Conditional (IF, CASE, etc.)
- String (common ones)
- Casts
- Aggregates(most common ones)
- Date/Timestamp

Data types

- Byte/Short/Int/Long
- Boolean
- String/Binary
- Decimal
- Float/Double
- Date/Timestamp
- Struct
- Array
- Map

https://learn.microsoft.com/de-de/azure/databricks/compute/photon#operators-expressions-and-data-types

Limitationen

- Queries die unter zwei Sekunden dauern werden nicht beeinflusst
- User defined functions und RDD APIs werden nicht unterstützt
- Structured Streaming:
  - Stateless streaming mit Delta, Parquet, CSV und JSON werden unterstützt
  - Stateless Kafka und Kinesis streaming ist supported, wenn nach Delta oder Parquet geschrieben wird

Features die nicht von Photon supported sind, funktionieren normal mit Sparks weiter

Photon Testen

Tests sind in Präsentation genauer beschrieben

- Sortieren / Aggregieren aus generierten Daten (25Mio Records)
  - Mit Photon: 5.252s (Sort); 4.607s (Agg)
  - Ohne Photon: 4.151s (Sort); 4.342s (Agg)
- Parquet erstellen
  - Mit Photon: 65.255 Sekunden
  - Ohne Photon: 63.757 Sekunden
- 1GB Ints sortieren (aus Azure Blob Storage)
  - Mit Photon: 58.482 Sekunden
  - Ohne Photon: 51.046 Sekunden

Mögliche Erklärung:

- Zu kleine Datenmenge
  - Photon overhead durch mehr möglichkeiten für Catalyst?

→Neuer Test mit komplexen JOINs

Test mit JOINs

1 Abfrage mit:

- 3 Inner Joins
- 1 Left Join auf Subquery
- Gruppierung
- Durchschnitt
- Summe
- Min/Max
- Count
- Sortiert

Durchschnitt von 50 Runs auf grösseres Dataset

Cmd 5

Join Data

Cmd 6

```python
import time

num_iterations = 50
durations = []

# Example of a complex join query across databases
for i in range(num_iterations):
    start = time.time()
    result_df = spark.sql("""
        SELECT
            c.name AS CustomerName,
            c.age AS CustomerAge,
            o.order_year,
            o.order_month,
            COUNT(DISTINCT o.order_id) AS TotalOrders,
            SUM(od.quantity) AS TotalProductsOrdered,
            AVG(o.order_total) AS AverageOrderValue,
            SUM(o.order_total) AS TotalOrderValue,
            MAX(p.price) AS MaxProductPrice,
            MIN(p.price) AS MinProductPrice,
            p_top.product_name AS MostExpensiveProductOrdered
        FROM
            db1.customers c
        JOIN
            (SELECT order_id, customer_id, YEAR(order_date) AS order_year, MONTH(order_date) AS order_month, order_total
            FROM db1.orders) o ON c.customer_id = o.customer_id
        JOIN
            db1.order_details od ON o.order_id = od.order_id
        JOIN
            db2.products p ON od.product_id = p.product_id
        LEFT JOIN
            (SELECT product_id, product_name FROM db2.products ORDER BY price DESC LIMIT 1) p_top ON od.product_id = p_top.product_id
        GROUP BY
            c.name, c.age, o.order_year, o.order_month, p_top.product_name
        ORDER BY
            TotalOrderValue DESC, TotalProductsOrdered DESC
        LIMIT 10
    """)
    # Show the result
    result_df.show()
    end = time.time()
    duration = end - start
    durations.append(duration)
    print(duration)

average_duration = sum(durations) / num_iterations
print(f"Average query time over {num_iterations} runs: {average_duration} seconds.")
```

Resultat

Ohne Photon:

Average query time over 50 runs: 11.28707139968872 seconds.

Mit Photon:

Average query time over 50 runs: 3.189061794281006 seconds.

-> Query fast 4x schneller mit Photon enabled

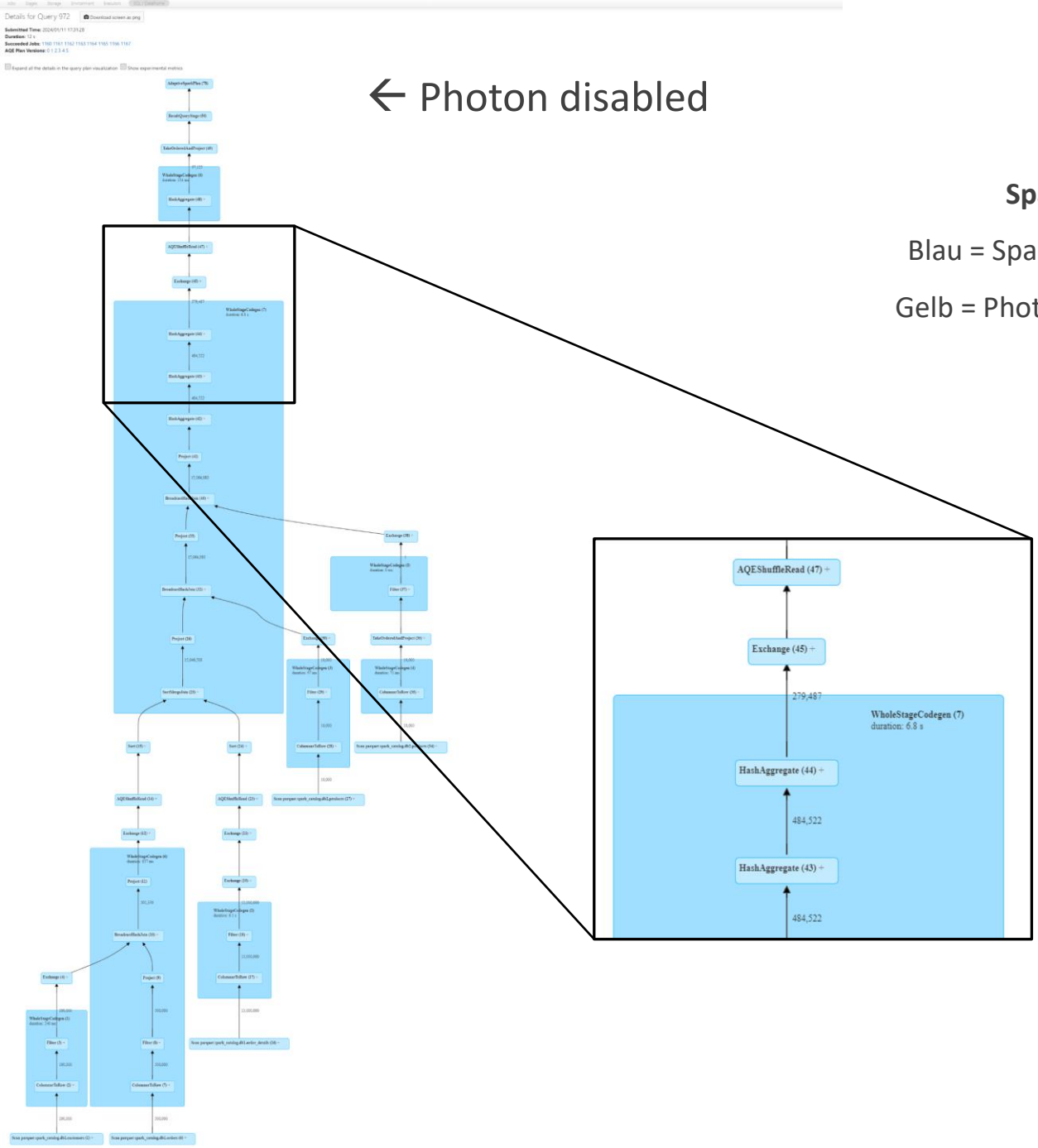
← Photon disabled

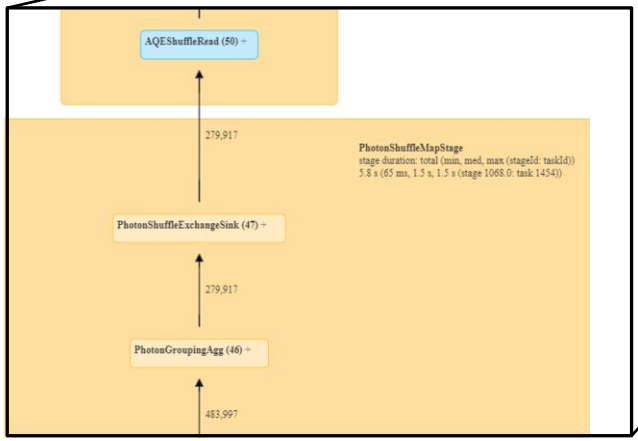
Photon enabled →

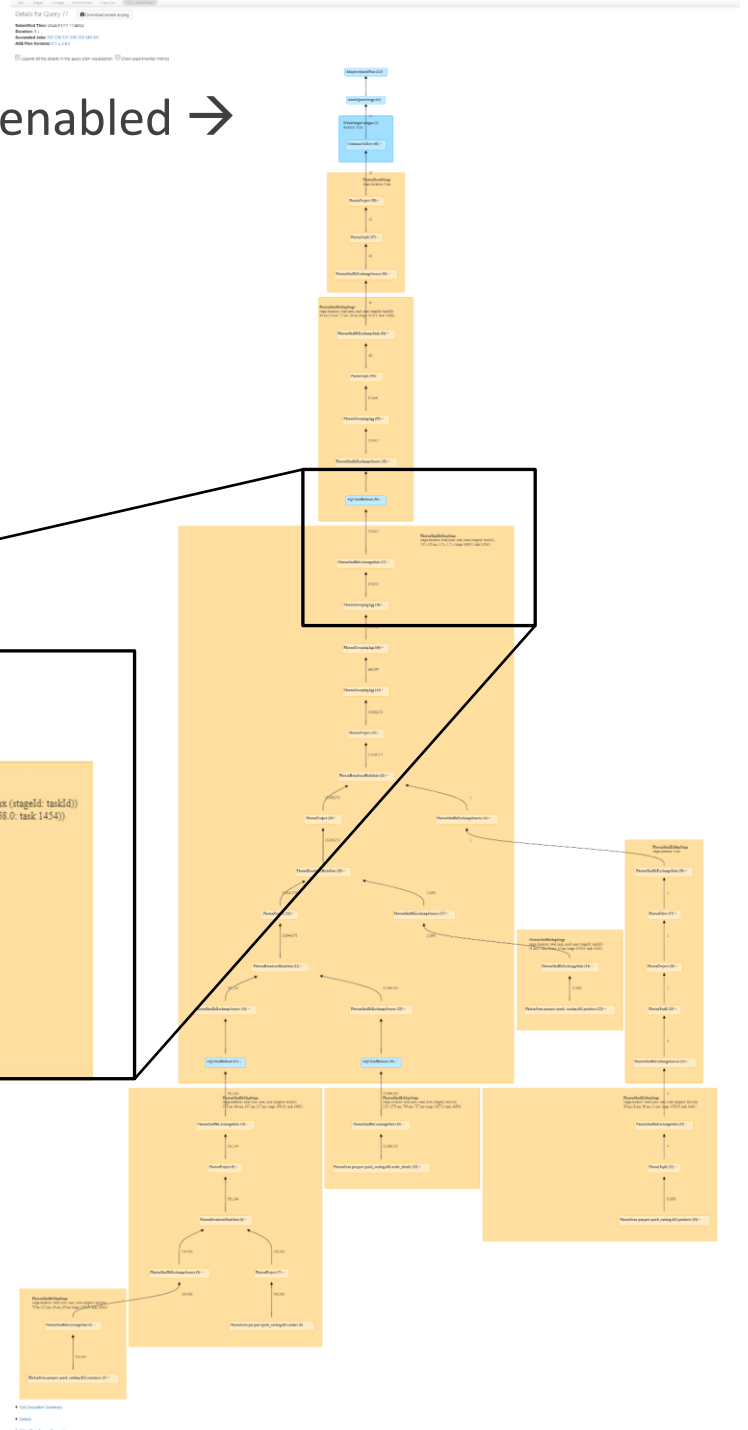
Spark UI

Blau = Spark ausführung

Gelb = Photon ausführung







Abschluss

Photon kann ein Query schneller machen aber nicht bei kleinen Datenmengen.

Wenn Photon eingeschaltet wird, kosten die Cluster auch mehr

→ Abwägung nötig, ob höhere Kosten und dafür schneller verarbeitung lohnenswert sind

Github mit Logs und Folien: https://github.com/Szokody/ZHAW_BigData_Photon

Wer selbst mit Databricks spielen will:

-> PPTX Anleitung wie man auf Azure selbst eine Databricks Umgebung aufbaut

Links

Databrick on Azure:

- https://learn.microsoft.com/en-us/azure/databricks

Databrick Benchmarks:

- https://docs.databricks.com/en/index.html

Databrick Dokumentation:

- https://docs.databricks.com/en/index.html

Photon Kosten vs. Nutzen:

- https://synccomputing.com/databricks-photon-and-graviton-instances-worth-it/

Photon Whitepaper:

- https://people.eecs.berkeley.edu/~matei/papers/2022/sigmod_photon.pdf

Quellen

https://datasolut.com/was-ist-databricks

https://learn.microsoft.com/en-us/azure/databricks/compute/photon

https://learn.microsoft.com/en-us/azure/databricks/spark/

https://docs.databricks.com/en/compute/photon.html