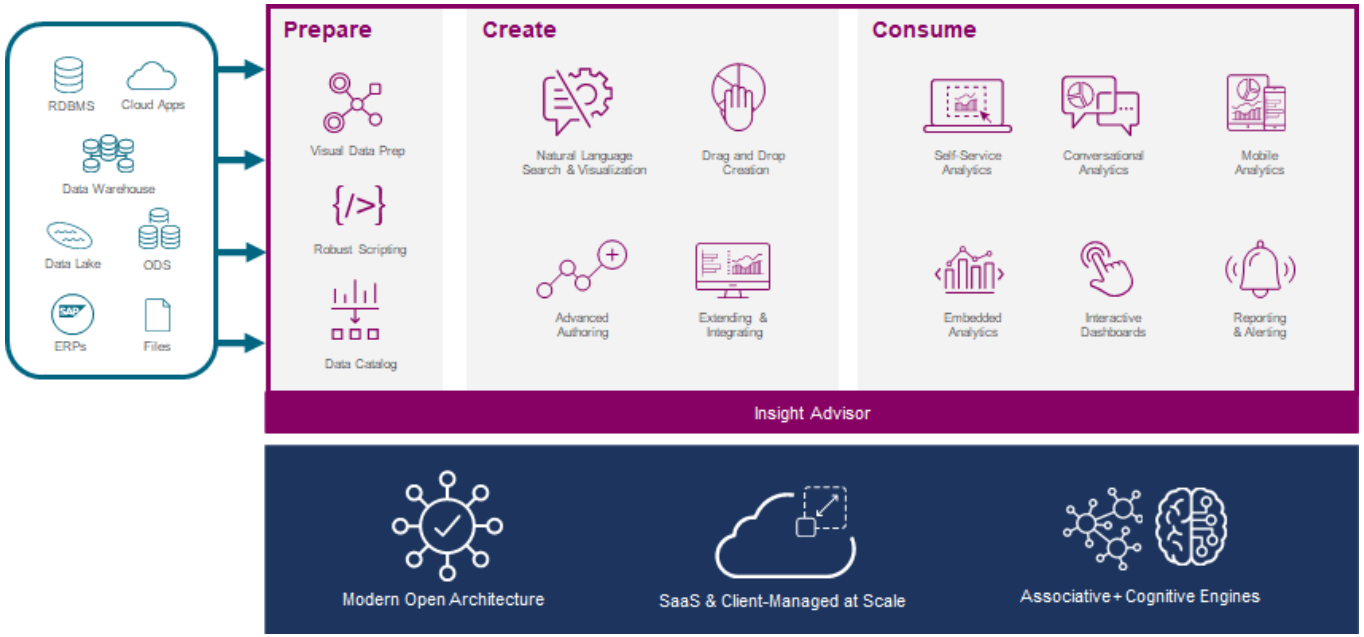
heyde

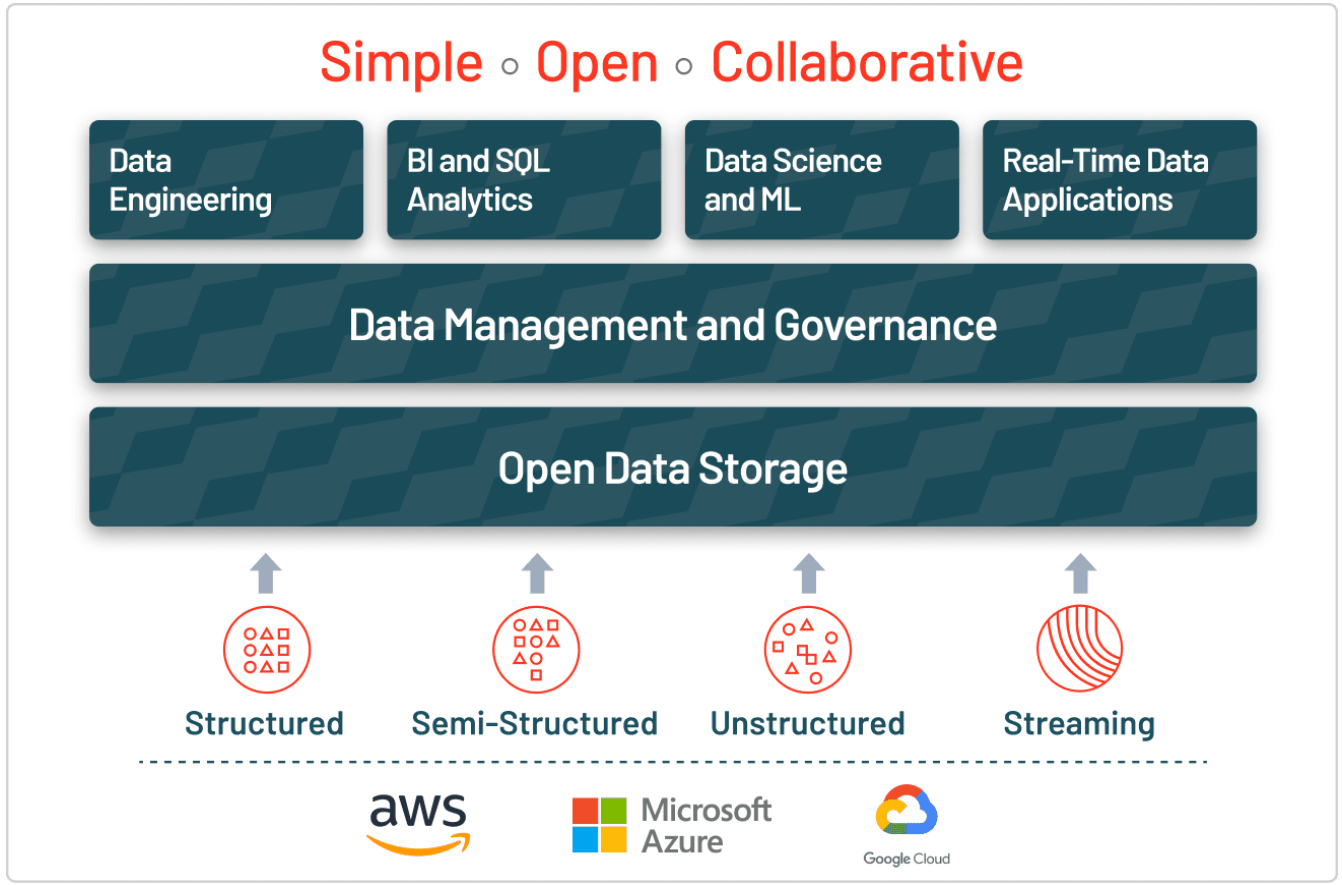
databricks®

QlikSense



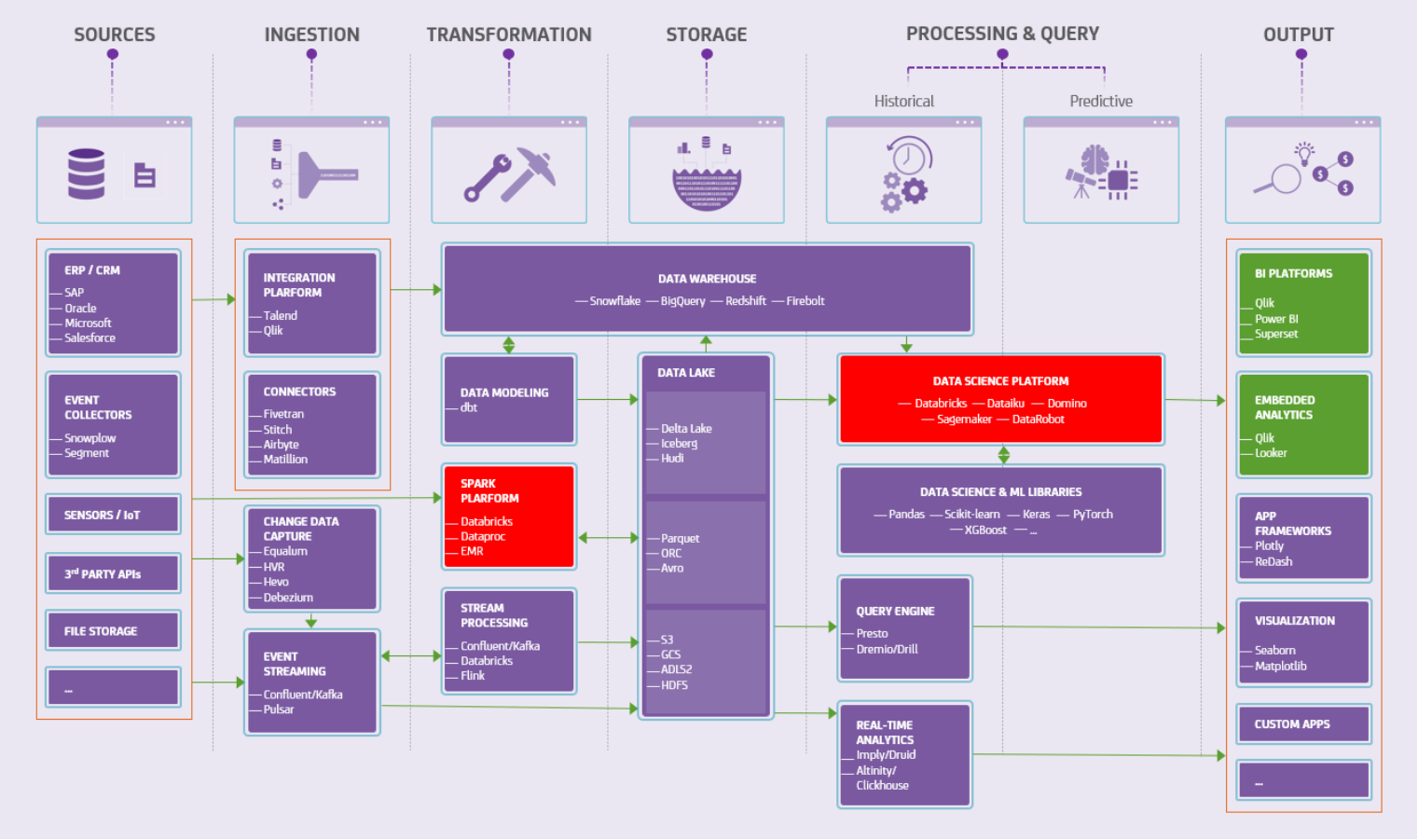
- Qlik Data Analytics Plattform
- Mehr als nur Business Intelligence
- In-Memory Technologie
- Proprietäre QVD File für die persistente Datenspeicherung
- Eigene SQL ähnliche Scriptsprache
- Qlik Applikation (QVF) für Code/Analyse

Databricks



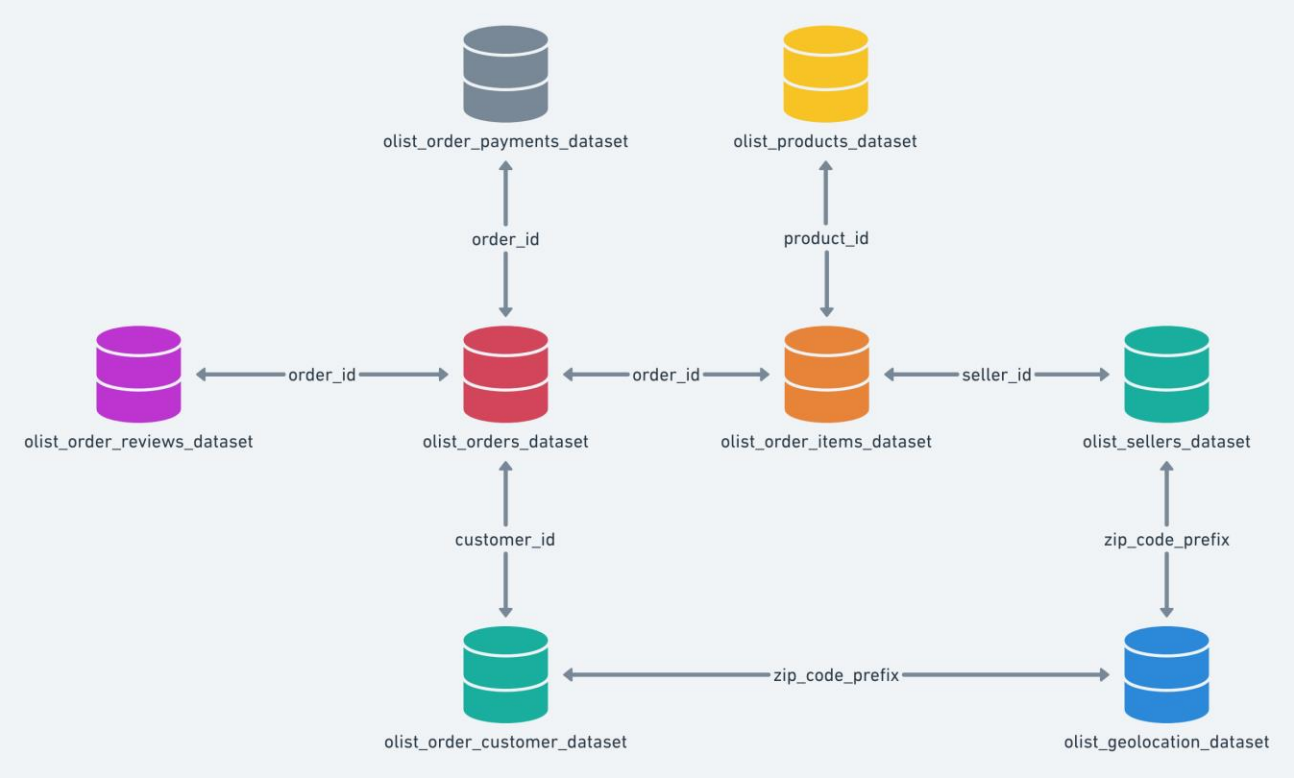
- Vollständige Lakehouse Plattform für Data Engineering, Analytics, Data Science
- Apache Spark, Delta Lake, MLflow
- In-Memory und Cluster-Computing
- Offene Delta Lake Format
- Scala, Python, SQL und R als mögliche Sprachen
- Notebook für Code, Output und Dokumentation

Vergleich bei der Datenaufbereitung



- QlikSense (Business Intelligence) vs Databricks (Data Platform)
- Unterschiedliche Produkte mit unterschiedlichem Hauptanwendungsgebiet
- Gemeinsamkeiten (Datenhaltung, In-Memory)
- Unterschiede (Scriptsprache, Datei Format, Geschwindigkeit resp. Overhead)

Dataset



https://www.kaggle.com/olistbr/brazilian-ecommerce

- Brazilian E-Commerce Public Dataset
- Die Tabelle *order_items* enthält 112’650 Datensätze
- Datensätze der Tabellen *order_items, orders, customers, products, sellers* vervielfachen um Faktor 100
- Speichern in CSV, QVD und Parquet
- Der Schlüssel *zip_code_prefix* für die *geolocation* ist nicht eindeutig

Transformation

```sql
CREATE TABLE retail_details AS
SELECT
  ...
  oi.price * 100 AS price,
  oi.freight_value * 100 as freight_value,
  o.customer_id,
  ...
FROM
  order_items AS oi
  INNER JOIN orders AS o ON oi.order_id = o.order_id
  LEFT JOIN products AS p ON oi.product_id = p.product_id
  INNER JOIN sellers AS s ON oi.seller_id = s.seller_id
  INNER JOIN customers AS c ON o.customer_id = c.customer_id
WHERE
  oi.price >= 5
  AND o.order_purchase_timestamp >= make_date(2016, 11, 1)
  AND s.seller_state <> 'MT'
  AND c.customer_state <> 'AC';
```

1. Erstellen der `retail_details` Tabelle und in QVD / Parquet speichern
2. Aggregieren der neuen detail Tabelle und als `retail_aggr` Tabelle in QVD / Parquet speichern

```sql
CREATE TABLE retail_aggr AS
SELECT
  year(order_purchase_timestamp) as year,
  month(order_purchase_timestamp) as period,
  order_status,
  product_category_name,
  product_weight_g,
  seller_city,
  seller_state,
  customer_city,

  sum(price) AS price,
  sum(freight_value) AS freight_value
FROM
  retail_details
WHERE
  seller_zip_code_prefix <> 3813
GROUP BY
  year(order_purchase_timestamp),
  month(order_purchase_timestamp),
  order_status,
  product_category_name,
  product_weight_g,
  seller_city,
  seller_state,
  customer_city;
```

File Format

File	Records	csv	qvd	parquet	qvd/csv	parquet/csv	parquet/qvd
customers	10'043'541	1'023'886'370	938'602'412	851'290'429	-8.33%	-16.86%	-9.30%
geolocation	1'000'163	59'027'802	51'023'265	19'929'274	-13.56%	-66.24%	-60.94%
order_items	11'377'650	1'833'309'026	744'460'233	1'114'669'773	-59.39%	-39.20%	49.73%
orders	10'043'541	2'166'785'685	1'043'638'384	893'185'519	-51.83%	-58.78%	-14.42%
products	3'328'051	263'005'888	179'597'508	161'239'187	-31.71%	-38.69%	-10.22%
sellers	312'595	18'992'790	15'014'731	13'715'741	-20.95%	-27.78%	-8.65%
retail_details	11'336'665	5'959'170'893	1'909'706'472	2'310'332'763	-67.95%	-61.23%	20.98%
retail_aggr	655'639	61'269'235	7'929'489	8'664'517	-87.06%	-85.86%	9.27%
TOTAL	**48'097'845**	**11'385'447'689**	**4'889'972'494**	**5'373'027'203**	**-57.05%**	**-52.81%**	**9.88%**

- Filegrösse in bytes
- QVD und Parquet sind CSV zu bevorzugen und komprimieren die Daten um rund 50%
- Kleinere Filegrösse bedeutet weniger Kosten (ADLS2, GCS, S3, …)
- QVD = proprietäres Format (Vendor lock-in)
- Parquet = Open-Source Format (Databricks, Snowflake, BigQuery, Amazon Athena, …)

QlikSense

- QlikSense February 2021 Patch 4
- Azure Windows Server 2019
- Standard_DS3_v2 (14GB RAM, 4 Cores)

Task Name	App Name	Reloa...	Reload Failures	Failure Rate	Avg Reload Duration	Max Duration	Last Reload	Last Reload Failure
retail	-	4	0	0%	00:03:43	00:03:51	2021-04-27 15:09:44	-
Totals		**4**	**0**	**0%**	**00:03:43**	**00:03:51**	-	-

- Die durchschnittliche Laufzeit = **3m 40s**

CREATE TABLE retail_details	00:01:29
CREATE TABLE qlik.retail_aggr	00:02:22
TOTAL	**00:03:51**

1/3 -> Join, Where
2/3 -> Aggregation

Databricks

Run	Start Time	Launched	Duration	Spark	Status
View Details	Apr 27 2021, 16:00 PM CEST	By scheduler	8m 45s	Spark UI / Logs / Metrics	Succeeded - Delete
View Details	Apr 27 2021, 15:00 PM CEST	By scheduler	7m 58s	Spark UI / Logs / Metrics	Succeeded - Delete
View Details	Apr 27 2021, 14:35 PM CEST	Manually	8m 35s	Spark UI / Logs / Metrics	Succeeded - Delete
View Details	Apr 27 2021, 14:26 PM CEST	Manually	8m 13s	Spark UI / Logs / Metrics	Succeeded - Delete
View Details	Apr 27 2021, 14:18 PM CEST	Manually	7m 56s	Spark UI / Logs / Metrics	Succeeded - Delete

- Single Node Mode und Databricks Runtime Version 8.1
- Node Type: Standard_DS3_v2 (14GB RAM, 4 Cores)
- Die durchschnittliche Laufzeit = **8m 20s**
- 4 Minuten um den Cluster aufzusetzen

CREATE OR REPLACE TEMPORARY VIEW	00:00:10
CREATE TABLE retail_details	00:03:37
CREATE TABLE qlik.retail_aggr	00:00:19
TOTAL	**00:04:06**

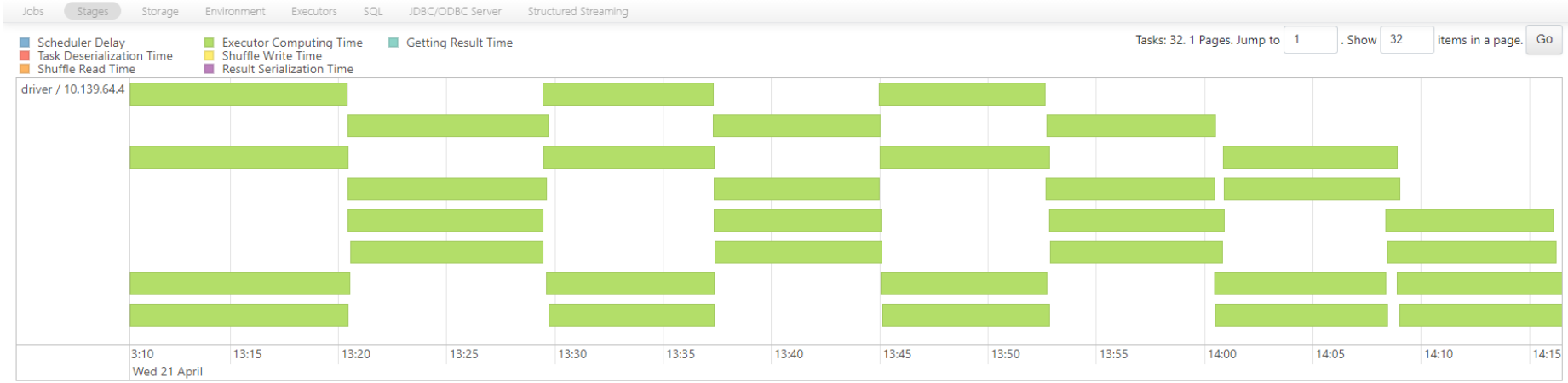
9/10 -> Join, Where
1/10-> Aggregation

Geolocation

```sql
CREATE TABLE retail_details AS
SELECT
  ...
  g.geolocation_lat AS customer_lat,
  g.geolocation_lng AS customer_lng
FROM
  order_items AS oi
  INNER JOIN orders AS o ON oi.order_id = o.order_id
  LEFT JOIN products AS p ON oi.product_id = p.product_id
  INNER JOIN sellers AS s ON oi.seller_id = s.seller_id
  INNER JOIN customers AS c ON o.customer_id = c.customer_id
  LEFT JOIN geolocation AS g ON (c.customer_zip_code_prefix = g.geolocation_zip_code_prefix
  AND c.customer_city = g.geolocation_city
  AND c.customer_state = g.geolocation_state)
WHERE
  oi.price >= 5
  AND o.order_purchase_timestamp >= make_date(2016, 11, 1)
  AND s.seller_state <> 'MT'
  AND c.customer_state <> 'AC';
```

- Geolocation mit `zip_code, city` und `state` nicht eindeutig
- QlikSense konnte die Datenmenge nicht bearbeiten
- Die Tabelle retail_details hat anschliessend 1’621’387’367 Datensätze

Geolocation



Summary Metrics for 32 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	7.6 min	7.7 min	7.9 min	8.9 min	10 min
GC Time	14 s	14 s	16 s	17 s	18 s
Output Size / Records	876.9 MiB / 50379988	880.3 MiB / 50590171	882.4 MiB / 50688977	884.1 MiB / 50787844	888.5 MiB / 50939622
Shuffle Read Size / Records	108.5 MiB / 666451	108.7 MiB / 667505	108.8 MiB / 668397	108.9 MiB / 668898	109.1 MiB / 669973

- Berechnung wird aufgeteilt
- Rechenzeit von 1 Stunde
- 32 Parquet Files und Total 30 GB

Schlussfolgerung

- QlikSense kann weiterhin bei der Datenaufbereitung eingesetzt werden und muss sich gegenüber modernen cloudbasierten Produkte wie Databricks nicht verstecken
- Beim Wechsel auf QlikSense SaaS oder grösseren/komplexeren QlikSense Architekturen sollte folgendes bedenkt werden
  - Die Datenhaltung (QVD) und Scriptsprache sind proprietär
  - Die Server Hardware- und Unterhaltskosten fallen rund um die Uhr an und ist nicht abhängig von der Nutzung (Zeit/Volumen)
  - Skalierung von Hardware ist schwieriger
  - Nutzen von Machine-/Deep-Learning Libraries oder Natural Language Processing ist nur umständlich möglich
  - Stream Processing ist nicht möglich
  - Semistrukturierte Daten sind teilweise und unstrukturierte Daten gar nicht möglich
  - Qlik kennt kein Tabellen Schema und Historisierung der Tabellen