Open Source Lakehouse Container (mittels DuckDB)

Seminararbeit

vorgelegt am 19. Januar 2025

Fakultät Wirtschaft und Gesundheit

Studiengang Wirtschaftsinformatik

Kurs WWI2022F

von

David Kreismann

DHBW Stuttgart:

Andreas Buckenhofer
Dozent für Data Management

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACID	Atomarität, Konsistenz, Isolation und Dauerhaftigkeit
BI	Business Intelligence
CRM	Customer Relationship Management
DBMS	Datenbankmanagementsystem
DL	Data Lake
DW	Data Warehouse
ERP	Enterprise Resource Planning
ETL	Extract, Transform, Load
HDFS	Hadoop Distributed File System
LH	Data Lakehouse
ML	Machine Learning
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
SQL	Structured Query Language

Abbildungsverzeichnis

1 Grundlagen

Mit dem Wachstum der Datenmengen setzen Unternehmen zunehmend auf Data Lakehouse (LH) Architekturen, um strukturierte und unstrukturierte Daten zu verwalten.[1] Weder Data Lake (DL) noch Data Warehouse (DW) Systeme gelten als ideal für moderne Anwendungsfälle, insbesondere bei fortschrittlichen Analysen wie Machine Learning (ML) Anwendungen, da führende ML-Systeme nur eingeschränkt mit DWs kompatibel sind.[2] Im Gegensatz zu Business Intelligence (BI)-Abfragen, die kleine Datenmengen verarbeiten, benötigen ML-Systeme große Datensätze und komplexen Code, der über Structured Query Language (SQL) hinausgeht.[3] Dies verdeutlicht die Herausforderungen der aktuellen Datenarchitekturen. Obwohl Cloud-basierte DL und DW-Lösungen durch die Trennung von Speicher (z.B. Objektspeicher-Dienste) und Rechenressourcen (z.B. Data Warehouse Engines) kosteneffizient wirken, führen sie zu erheblicher Komplexität.[4] Moderne Architekturen erfordern oft einen mehrstufigen Extract, Transform, Load (ETL)-Prozess, bei dem Daten zunächst roh im DL und anschließend im DW gespeichert werden. Dieser Prozess ist zeitaufwendig, komplex und anfällig für Fehler. LH-Architekturen lösen diese Probleme, indem sie offene Speicherformate mit Funktionen von DW-Systemen kombinieren, wie Abrageoptimierungen und Atomarität, Konsistenz, Isolation und Dauerhaftigkeit (ACID)-Transaktionen.[5]

Im Rahmen dieser Arbeit wird eine Open-Source-Umsetzung einer LH-Architektur realisiert, die Komponenten wie MinIO, DuckDB und Apache Superset integriert, um die Vorteile dieser Architektur zu demonstrieren.

1.1 Entstehung der Data Lakehouse Architektur

Traditionelle Datenbanken, sogenannte Online Transaction Processing (OLTP)-Systeme, wurden entwickelt, um tägliche Transaktionen effizient zu verarbeiten und schnellen sowie konsistenten Zugriff auf Daten zu gewährleisten.[6] OLTP-Systeme sind somit optimiert für hohe Transaktionslasten und verwenden normalisierte Datenstrukturen, um Anomalien bei Updates zu vermeiden.[7] Diese starke Normalisierung macht sie jedoch ineffizient für komplexe Analysen, bei denen große Datenmengen verarbeitet oder mehrere Tabellen verknüpft werden müssen.[8]

Aus diesem Grund wurden Online Analytical Processing (OLAP)-Systeme entwickelt, die auf Datenanalyse und Entscheidungsunterstützung ausgerichtet sind. OLAP-Queries erfordern oft

[1] Vgl. Armbrust u. a. 2021, S. 1
[2] Vgl. Mazumdar/Hughes/Onofre 2023, S. 5
[3] Vgl. Armbrust u. a. 2021, S. 1
[4] Vgl. Mazumdar/Hughes/Onofre 2023, S. 5
[5] Vgl. Armbrust u. a. 2021, S. 1
[6] Vgl. Vaisman/Zimnyi 2014, S. 45
[7] Vgl. Vaisman/Zimnyi 2014, 45 ff.
[8] Vgl. Vaisman/Zimnyi 2014, 45 ff.

vollständige Tabellenscans und Aggregationen, wofür OLTP-Systeme ungeeignet sind.[9]

Aus dieser Notwendigkeit entstanden Datenbanken für analytische Zwecke, sogenannte DWs, als zentrale Speicherorte für strukturierte Daten, die aus verschiedenen Quellen über ETL-Prozesse integriert werden, siehe Abbildung 1.[10]

Die Abbildung verdeutlicht den typischen ETL-Prozess eines DW. Daten werden aus verschiedenen Quellen wie Customer Relationship Management (CRM)- und Enterprise Resource Planning (ERP)-Systemen extrahiert, validiert, bereinigt und transformiert, bevor sie in das DW geladen werden. Dort werden sie in Data Marts organisiert und für Reporting, Visualisierung und BI-Anwendungen genutzt. Diese Daten werden häufig in Modellen wie Data Vault[11] oder Starschemata[12] organisiert, um eine effiziente Abfrage und Berichterstellung zu ermöglichen.[13] DWs sind ideal für BI und historische Analysen, jedoch oft teuer und mit modernen Open-Source- oder Cloud-basierten Tools schwer kompatibel.[14]

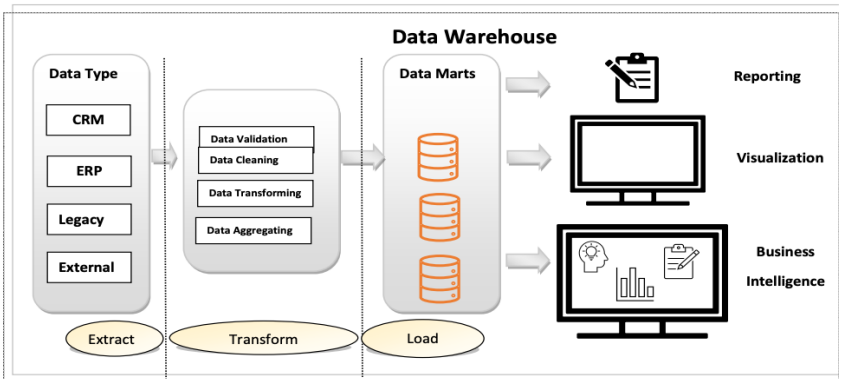


Abb. 1: Data Warehouse Architektur (DW).[15]

DLs wurden von Dixon 2010 als flexible Alternative zu DWs entwickelt, siehe Abbildung 2, um große Mengen an unstrukturierten, semi-strukturierten und strukturierten Daten zu speichern.[16] DL verzichten auf ein vorab definiertes Schema und speichern Daten in ihrer Rohform, was eine hohe Flexibilität bietet. Daten können nach Bedarf organisiert werden, beispielsweise in „Daten-Teiche“ (Data Ponds) für spezifische Datentypen wie rohe Daten, Anwendungsdaten oder Textdaten.[17]

Abbildung 2 illustriert die grundlegende Architektur eines Data Lakes. Daten unterschiedlicher Typen (strukturiert, semi-strukturiert und unstrukturiert) werden extrahiert und geladen, bevor sie in Big Data Systemen wie Hadoop oder SQL/NoSQL-Datenbanken verarbeitet werden. Anschließend durchlaufen sie Prozesse wie die Datenvorbereitung, das Metadatenmanagement und

[9] Vgl. Vaisman/Zimnyi 2014, S. 46
[10] Vgl. Harby/Zulkernine 2022, S. 390
[11] Vgl. Kimball/Ross 2013, S. 1
[12] Vgl. Linstedt/Olschimke 2015, S. 1
[13] Vgl. Vaisman/Zimnyi 2014, S. 6
[14] Vgl. Harby/Zulkernine 2022, S. 390
[15] Enthalten in: Harby/Zulkernine 2022, p. 389
[16] Vgl. Harby/Zulkernine 2022, S. 390
[17] Vgl. Harby/Zulkernine 2022, S. 390

die Governance. Diese Schritte gewährleisten eine effiziente Verwaltung und Bereitstellung der Daten für Analysen und Machine-Learning-Anwendungen.

Diese Strukturierung erleichtert die Handhabung großer Datenmengen, doch Data Lakes stehen vor Herausforderungen wie mangelnder Datenqualität und der Gefahr von „Data Swamps“, in denen unorganisierte und schwer auffindbare Daten die Effektivität einschränken.[18]

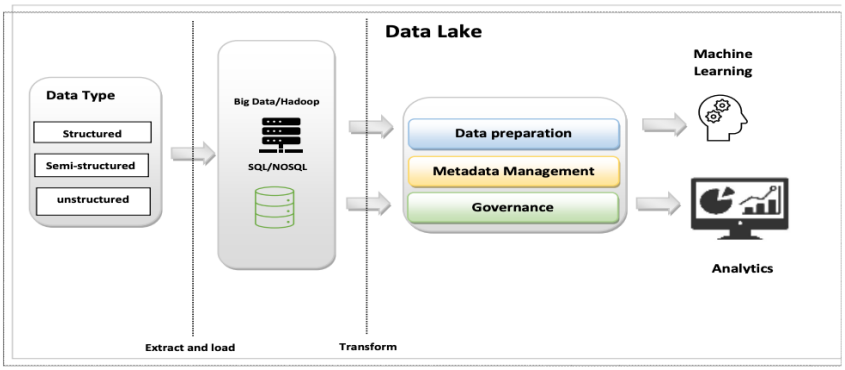


Abb. 2: Data Lake Architektur (DL).[19]

1.2 Definition und Beschreibung von einem Data Lakehouse

Um die Stärken von DWs und DLs zu vereinen, wurde die LH-Architektur entwickelt. Diese kombiniert die skalierbare, flexible Speicherfähigkeit von DLs mit den strukturierten und integrierten Analysefunktionen von DWs.[20]

Armbrust u. a. 2021 definiert ein LH als ein Datenmanagementsystem, das kostengünstigen, direkt zugänglichen Speicher mit den traditionellen Verwaltungs- und Leistungsmerkmalen eines analytischen Datenbankmanagementsystem (DBMS) kombiniert.[21] Zu diesen Merkmalen zählen ACID-Transaktionen, Datenversionierung, Auditierung, Indexierung, Caching und Abfrageoptimierung.

Ein LH bietet eine kostengünstige Speicherung in einem offenen Format, das von verschiedenen Systemen zugänglich ist, während leistungsstarke Verwaltungs- und Optimierungsfunktionen bereitgestellt werden. Die Architektur ist besonders geeignet für Cloud-Umgebungen mit getrennter Verarbeitung und Speicherung.[22] Anwendungen wie ML-Modelle können flexibel auf separaten Rechenknoten ausgeführt werden, während sie auf denselben Speicher zugreifen. Gleichzeitig ist auch die Implementierung in lokalen Speicherumgebungen wie Hadoop Distributed File System (HDFS) möglich.[23]

[18] Vgl. Inmon 2016, S. 46 ff.
[19] Enthalten in: Harby/Zulkernine 2022, p. 389
[20] Vgl. Harby/Zulkernine 2022, S. 391
[21] Vgl. Armbrust u. a. 2021, S. 3
[22] Vgl. Armbrust u. a. 2021, S. 3
[23] Vgl. Armbrust u. a. 2021, S. 3

Ein zentraler Aspekt der LH-Architektur ist die Nutzung moderner technologischer Ansätze, die flexible Datenzugänglichkeit mit leistungsstarker Abfrage- und Analyseoptimierung kombinieren. Diese hybride Lösung vereint die Vorteile von DLs und DWs. Offene Speicherformate wie Apache Parquet oder Delta Lake sowie Cloud-Objektspeicher wie Amazon S3 oder MinIO bilden dabei die Grundlage.[24]

Im Gegensatz zu herkömmlichen DWs, die Rechenleistung und Speicher eng koppeln und dadurch in ihrer Skalierbarkeit eingeschränkt sind, entkoppeln Lakehouses diese Komponenten. Dadurch können Abfragen unabhängig von der Datenhaltung in separaten Engines verarbeitet werden, was verteilte Abfragen über verschiedene Datenquellen ermöglicht.[25] Zudem vermeiden LH redundante ETL-Prozesse und physische Datenkopien, indem sie direkt auf semi-strukturierte Speicher wie S3-Objektspeicher zugreifen.[26]

Die Konsistenz der Daten wird durch Metadatenkataloge wie Apache Iceberg gewährleistet, die ACID-Transaktionen ermöglichen.[27] Optimierungen wie Indexerstellung und effiziente Datenlayouts erhöhen die Abfragegeschwindigkeit erheblich. Dank dieser Eigenschaften etabliert sich die LH-Architektur zunehmend als Standard für die Verarbeitung großer Datenmengen.[28]

Abbildung 3 zeigt den Aufbau eines LH. Die Architektur umfasst eine Extraction- und Ingestion-Schicht, die Daten aus verschiedenen Quellen extrahiert und verarbeitet. Im DL werden die Daten zunächst in der Landing/Stage Area in ihrer Rohform gespeichert und in der Foundation Area weiter aufbereitet. Anschließend werden sie in das DW-Modell überführt, das aus der Base Layer für konsistente Datenspeicherung und der Performance- und Analytics-Layer für schnelle Abfragen, Berichte und Analysen besteht.

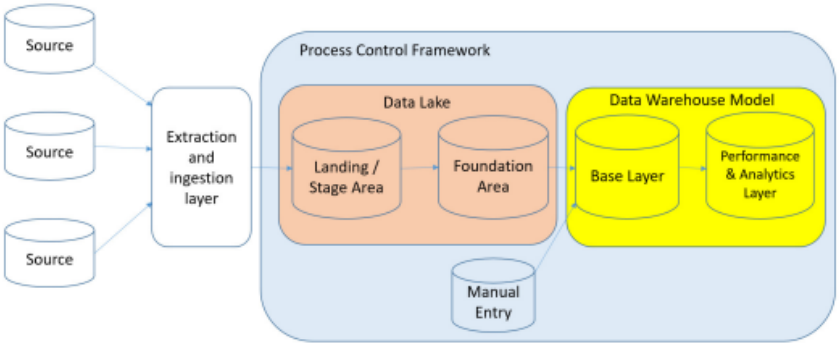


Abb. 3: Data Lakehouse Architektur (LH).[29]

Im Rahmen dieser Arbeit wird die theoretische Grundlage der LH-Architektur durch eine praxisorientierte Open Source Umsetzung aufgezeigt. Die einzelnen Komponenten der Architektur werden mithilfe von Containern realisiert. MinIO fungiert als Objektspeicher für Daten im Delta- und

[24] Vgl. Mazumdar/Hughes/Onofre 2023, S. 6
[25] Vgl. Armbrust u. a. 2021, S. 2
[26] Vgl. Armbrust u. a. 2021, S. 2
[27] Vgl. Armbrust u. a. 2021, S. 2
[28] Vgl. Armbrust u. a. 2021, S. 2
[29] Enthalten in: Oreščanin/Hlupić 2021, p. 1243

Parquetformat, während Apache Spark die Zerlegung der Beispieldaten in Delta-Tabellen und deren Speicherung übernimmt. Für die Verarbeitung und Analyse der Daten werden DuckDB, Pandas und Ibis verwendet. Die Ergebnisse werden in Apache Superset visualisiert, das Dashboards und Diagramme erstellt, um die analysierten Daten übersichtlich darzustellen.

Die Zerlegung und Speicherung der Daten in MinIO bildet die Extraction- und Ingestion-Schicht, während die Foundation Area durch die Datenaufbereitung und Analysen mit DuckDB und Ibis umgesetzt wird. Die Performance- und Analytics-Layer wird durch die Visualisierung in Superset realisiert. Diese Implementierung demonstriert die Flexibilität, Skalierbarkeit und Effizienz der Lakehouse-Architektur in einer Open-Source-Umgebung.

Die LH Architektur wird durch ein Process-Control-Framework gesteuert, das eine konsistente Datenverwaltung sicherstellt. Zusätzlich ermöglicht die manuelle Eingabe die Integration spezifischer Daten. Dadurch können LH-Systeme komplexe Abfragen und Analysen effizient unterstützen und sind besonders geeignet für Anwendungen wie ML und Analytics. Trotz ihrer zahlreichen Vorteile erfordern LH-Architekturen eine sorgfältige Integration der Speicher- und Rechenkomponenten sowie eine Optimierung der Abfragen und Datenformate. Technologien wie Delta Lake, Apache Parquet und Metadatenkataloge wie Apache Iceberg spielen hierbei eine zentrale Rolle.[30]

[30] Vgl. Mazumdar/Hughes/Onofre 2023, S. 6

2 Installation

In diesem Kapitel wird die lokale Installation der Container erläutert, welche die Basis für die in dieser Arbeit entwickelte Open Source Lakehouse Umgebung darstellt. Die Bereitstellung erfolgt mithilfe einer `docker-compose.yml` Datei, die die Konfiguration der einzelnen Architekturkomponenten, einschließlich Speicher, Verarbeitung und Visualisierung, sowie deren Aufbau und Orchestrierung definiert.

2.1 Voraussetzungen

Für eine erfolgreiche Installation müssen bestimmte Voraussetzungen erfüllt sein:

- Vorhandensein einer installierten Container-Laufzeitumgebung wie Docker[31] oder Podman[32] mit Unterstützung für Compose-Dateien.
- Eine korrekt konfigurierte `.env`-Datei mit den notwendigen Umgebungsvariablen. Im Rahmen der Arbeit sind diese Werte in der bereitgestellten `.env`-Datei bereits gesetzt.

Die Umgebungsvariablen definieren unter anderem die Zugangsdaten für MinIO und Apache Superset und bilden die Grundlage für eine funktionierende Lakehouse-Umgebung.

Nachfolgend der Inhalt der `.env`-Datei:

```
# MinIO
MINIO_ROOT_USER=
MINIO_ROOT_PASSWORD=
MINIO_URL=
MINIO_BUCKET=
MINIO_ACCESS_KEY=
MINIO_SECRET_KEY=

# Superset
SUPERSET_ADMIN_USERNAME=
SUPERSET_ADMIN_PASSWORD=
SUPERSET_SECRET_KEY=
```

[31] Docker Inc. 2025
[32] Podman 2025

2.2 Installation der Container Infrastruktur

Für die Installation ist ein Terminal zu öffnen und in das Root-Verzeichnis des Ordners zu navigieren, in dem sich die `docker-compose.yml`-Datei befindet. Die Installation erfolgt in den folgenden Schritten:

1. Die `docker-compose.yml`-Datei, siehe Anhang 1, wird ausgeführt, wodurch alle Container gestartet werden.

   Für die Nutzung von Podman lautet der entsprechende Befehl:

```
podman-compose build
podman-compose up
```

   Für die Nutzung von Docker lautet der entsprechende Befehl:

```
docker-compose build
docker-compose up
```

Die Infrastruktur umfasst mehrere Container, die verschiedene Komponenten der Lakehouse-Architektur implementieren:

MinIO (storage und storage-init) MinIO dient als Objektspeicher und stellt den zentralen Speicherort für die Daten im Delta- und Parquet-Format bereit. Der Container `storage-init` ist für die Initialisierung und Konfiguration des MinIO-Speichers zuständig.

Compute (duckdb-ibis-python) Dieser Container enthält DuckDB und Ibis, die für die Analyse und Verarbeitung der in MinIO gespeicherten Daten verantwortlich sind. Die Umgebung ist darauf ausgelegt, Daten direkt aus MinIO zu lesen, zu verarbeiten und für weiterführende Analysen vorzubereiten.

Visualisierung (apache-superset) Apache Superset dient als Visualisierungstool und ermöglicht die Erstellung von Dashboards und Berichten auf Basis der von DuckDB analysierten Daten.

Apache Spark Cluster (spark-master, spark-worker, spark-submit) Apache Spark ist für die Verarbeitung der Ursprungsdaten (`ds_salaries.csv`) zuständig. Der Spark-Master und Spark-Worker bilden den Cluster, während der `spark-submit`-Container periodisch Jobs zur Verarbeitung ausführt.

Scraper (scraper) Der Scraper-Container ist für das Abrufen externer Gehälter von `https://www.glassdoor.de/Job/Data-Scientist-jobs-SRCH_KO0,10.htm` vorgesehen. Die extrahierten Daten können direkt in den Speicher oder die Verarbeitungs-Pipeline eingespeist werden.

2.3 Überprüfung der Installation

Nach dem erfolgreichen Ausführen der Befehle werden die einzelnen Container gestartet und sind unter den in der `docker-compose.yml` definierten Ports und URLs erreichbar. Die folgenden Komponenten sollten anschließend zur Verfügung stehen:

- **MinIO:** `http://localhost:9000`.
- **Apache Superset:** `http://localhost:8088`.
- **Apache Spark:** `http://localhost:8080`.
- **DuckDB:** Verarbeitet die Daten direkt aus dem Objektspeicher und führt Analysen durch.

Die Installation erfolgt lokal und wurde im Rahmen der Entwicklung erfolgreich auf verschiedenen Endgeräten mit Podman und Docker als Container-Laufzeitumgebungen getestet.

3 Umsetzung Beispiel

Dieses Kapitel präsentiert die praktische Umsetzung der in dieser Arbeit entwickelten Open Source Lakehouse Architektur. Dabei werden die zentralen Features, die verwendeten Skripte sowie die Integration und Funktionalität der einzelnen Komponenten näher erläutert.

Für die Analyse, Verarbeitung und Visualisierung dient der Datensatz `ds_salaries.csv` als Grundlage. Dieser Datensatz umfasst insgesamt 11 Spalten, die detaillierte Informationen zu Gehältern, Arbeitsbedingungen und Unternehmensstrukturen enthalten. Er ist online unter folgender URL verfügbar: `https://www.kaggle.com/datasets/arnabchaki/data-science-salaries-2023` und in der bereitgestellten Abgabe im Ordner `spark/data` hinterlegt.

Die 11 Spalten des Datensatzes beinhalten die folgenden Informationen:

1. work_year: The year the salary was paid.
2. experience_level: The experience level in the job during the year
3. employment_type: The type of employment for the role
4. job_title: The role worked in during the year.
5. salary: The total gross salary amount paid.
6. salary_currency: The currency of the salary paid as an ISO 4217 currency code.
7. salaryinusd: The salary in USD
8. employee_residence: Employee's primary country of residence in during the work year as an ISO 3166 country code.
9. remote_ratio: The overall amount of work done remotely
10. company_location: The country of the employer's main office or contracting branch
11. company_size: The median number of people that worked for the company during the year

Abb. 4: Beschreibung der Spalten des Datensatzes.

Um den Datensatz weiter anzureichern und die Möglichkeiten der Echtzeitverarbeitung zu demonstrieren, wurde zusätzlich ein Web-Scraper entwickelt. Dieser sammelt Gehaltsdaten von der Webseite Glassdoor unter folgender URL: `https://www.glassdoor.de/Job/Data-Scientist-jobs-SRCH_KO0,10.htm` und integriert die gesammelten Informationen direkt in den bestehenden Datensatz. Auf diese Weise werden die Analysen um zusätzliche Aspekte ergänzt, die eine Echtzeitverarbeitung sowie eine erweiterte Visualisierung der Daten ermöglichen. Der Web-Scraper befindet sich im Ordner `scraper` und kann durch das Starten des entsprechenden Containers ausgeführt werden.

3.1 Demonstration von Features

Die Umsetzung der Lakehouse-Architektur umfasst mehrere Schritte und Features. Zu Beginn wird ein `minio`-Container gestartet, der als zentraler Objektspeicher dient. Anschließend erfolgt die Einrichtung durch einen `minio-init`-Container. Während dieses Prozesses wird ein Administrator-Konto erstellt, dessen Zugangsdaten aus der `.env`-Datei ausgelesen werden. Zudem wird ein Bucket mit dem Namen `lakehouse-storage` konfiguriert, der als Speicherort für die verarbeiteten Daten dient.

Abbildung 5 zeigt den Login-Bildschirm von MinIO, der den Zugang zur Verwaltungsoberfläche ermöglicht.

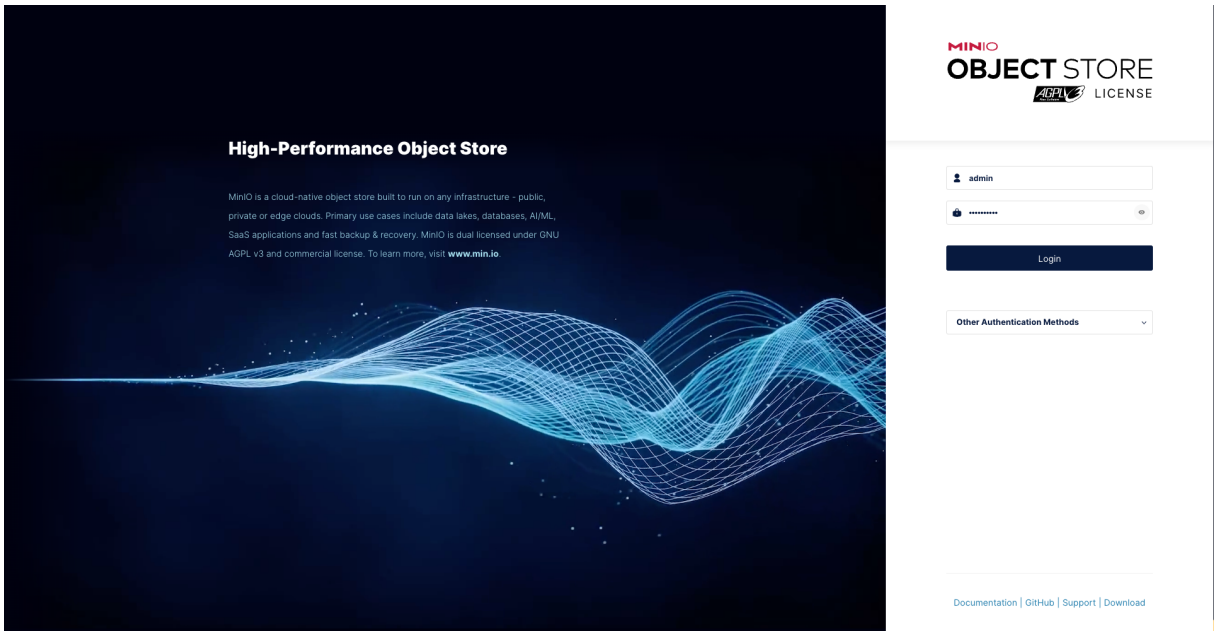


Abb. 5: MinIO - Login Screen.

Nach erfolgreicher Einrichtung zeigt die MinIO-Oberfläche den konfigurierten Bucket `lakehouse-storage`, wie in Abbildung 6 dargestellt. Dieser Bucket dient als zentraler Speicherort für Daten im Delta- und Parquet-Format, die im Rahmen der Lakehouse-Architektur verarbeitet und analysiert werden.

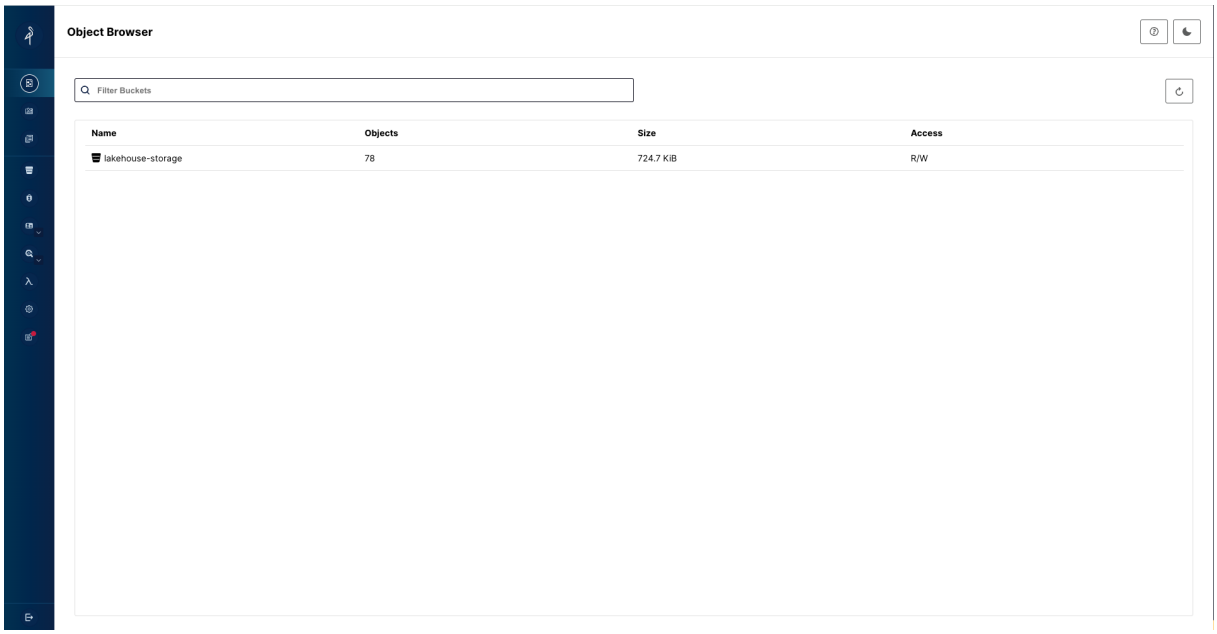
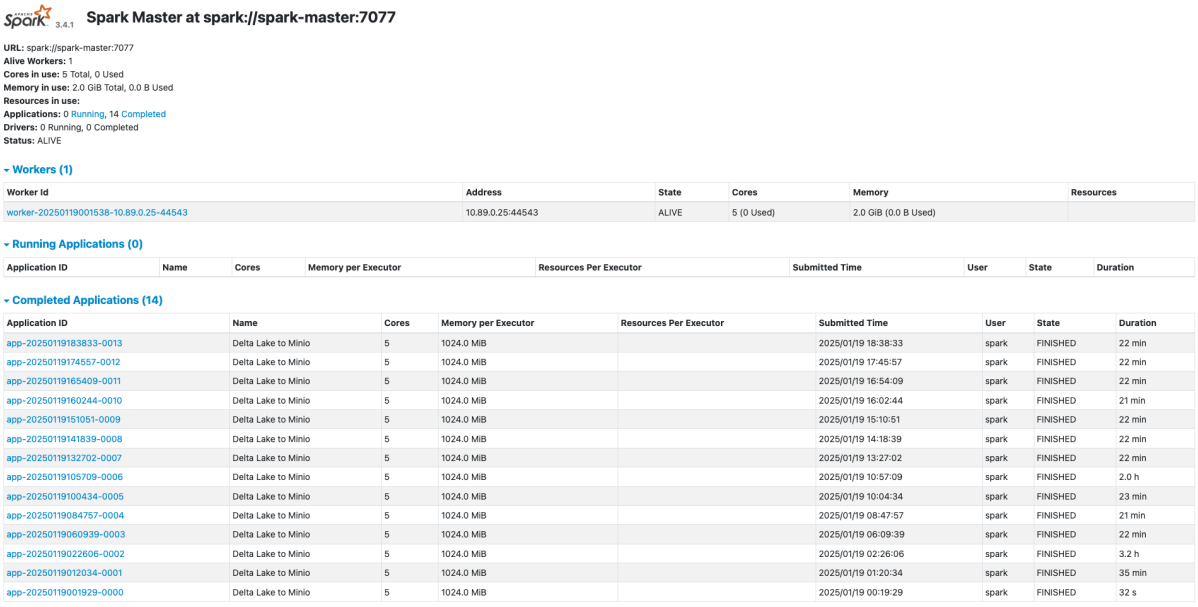


Abb. 6: MinIO mit erstelltem `lakehouse-storage` Bucket.

Anschließend wird ein Spark-Cluster gestartet, bestehend aus einem `spark-master`-Container, einem `spark-worker`-Container und einem `spark-submit`-Container. Dieser Cluster übernimmt die Verarbeitung der Ursprungsdaten aus der Datei `ds_salaries.csv`. Dabei werden die Daten in Delta-Tabellen zerlegt und in den MinIO-Speicher hochgeladen, um eine effiziente Speicherung und Weiterverarbeitung zu gewährleisten. Dies markiert den ersten Schritt innerhalb einer Medallion-Architektur, in dem die "Raw"Daten verarbeitet, in Delta-Tabellen überführt und im gemeinsamen Speicher organisiert werden. Gleichzeitig erfolgen erste Transformationen, die die Grundlage für nachfolgende Verarbeitungsschritte bilden. Zudem wird dadurch eine Simulationsumgebung geschaffen, die demonstriert, wie Daten aus verschiedenen Systemen in ein Speicher geladen werden können.

Abbildung 7 zeigt die Oberfläche des Apache Spark Clusters, die den Status und die laufenden Jobs visualisiert.



Spark 3.4.1 **Spark Master at spark://spark-master:7077**

URL: spark://spark-master:7077
Alive Workers: 1
Cores in use: 5 Total, 0 Used
Memory in use: 2.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 14 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20250119001538-10.89.0.25-44543	10.89.0.25:44543	ALIVE	5 (0 Used)	2.0 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (14)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20250119183833-0013	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 18:38:33	spark	FINISHED	22 min
app-20250119174557-0012	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 17:45:57	spark	FINISHED	22 min
app-20250119165409-0011	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 16:54:09	spark	FINISHED	22 min
app-20250119160244-0010	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 16:02:44	spark	FINISHED	21 min
app-20250119151051-0009	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 15:10:51	spark	FINISHED	22 min
app-20250119141839-0008	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 14:18:39	spark	FINISHED	22 min
app-20250119132702-0007	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 13:27:02	spark	FINISHED	22 min
app-20250119105709-0006	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 10:57:09	spark	FINISHED	2.0 h
app-20250119100434-0005	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 10:04:34	spark	FINISHED	23 min
app-20250119084757-0004	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 08:47:57	spark	FINISHED	21 min
app-20250119060939-0003	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 06:09:39	spark	FINISHED	22 min
app-20250119022606-0002	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 02:26:06	spark	FINISHED	3.2 h
app-20250119012034-0001	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 01:20:34	spark	FINISHED	35 min
app-20250119001929-0000	Delta Lake to Minio	5	1024.0 MiB		2025/01/19 00:19:29	spark	FINISHED	32 s

Abb. 7: Oberfläche des Apache Spark Clusters.

Zur Anreicherung des Datensatzes mit zusätzlichen Informationen wird der `scraper`-Container eingesetzt. Dieser sammelt Gehaltsdaten von der Webseite Glassdoor, verarbeitet sie mithilfe der Python-Bibliothek `pandas` und der Bibliothek `Selenium` und fügt sie dem bestehenden Datensatz hinzu (siehe Abbildung 8). Um eine regelmäßige Aktualisierung der Daten zu gewährleisten, sind sowohl der `scraper`-Container als auch der `spark-submit`-Container so konfiguriert, dass sie alle 20 Minuten automatisch ausgeführt werden. Dadurch bleibt der MinIO-Speicher stets auf dem neuesten Stand.

```
;Job Title: Remote Sensing Scientist– Atmosphere Products
/Salary Estimate: 7500 € (Arbeitgeber-Schtz.)
Job Description: Experience with operational data processing software would be an
'Kenntnisse und Fähigkeiten: C, Python
Rating: 3.6
Company Name: EUMETSAT
Location: Darmstadt
```

Abb. 8: Ein Ausschnitt einer gescrapten Jobstelle von Glassdoor.

Die verarbeiteten Daten werden im MinIO-Speicher in Form von Delta Lake Tabellen gespeichert, wie in Abbildung 9 dargestellt. Diese Tabellen sind in verschiedene logische Partitionen unterteilt, was eine flexible Abfrage und Analyse ermöglicht.

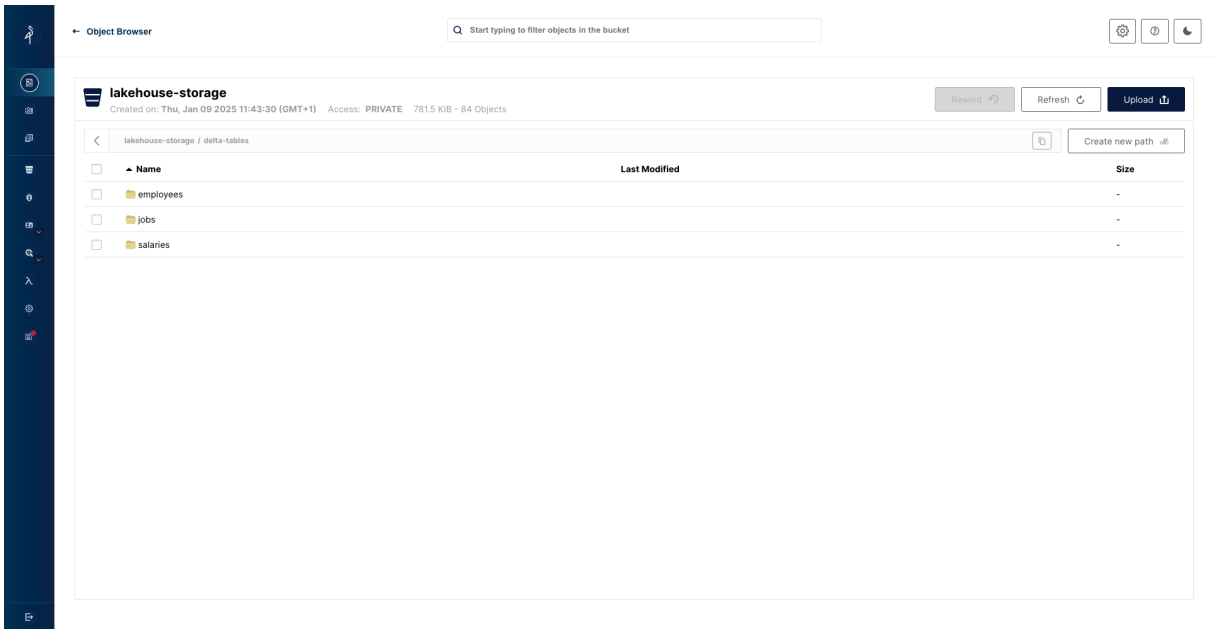


Abb. 9: Zerlegte CSV-Datei in Delta Lake Tabellen.

Eine detaillierte Ansicht der gespeicherten Daten zeigt die Employee-Tabelle, die aus mehreren Parquet-Dateien besteht. Diese Dateien ermöglichen eine effiziente Speicherung und Bereitstellung großer Datenmengen, wie in Abbildung 11 illustriert.

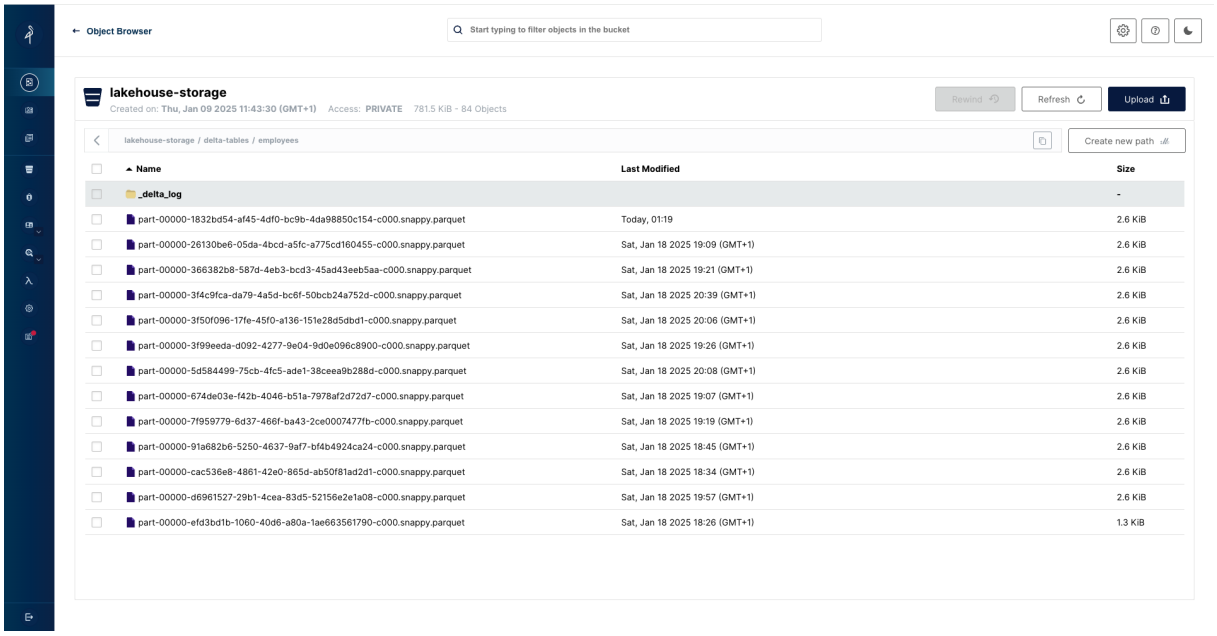


Abb. 10: Delta Lake Employee Tabelle mit Parquet-Dateien.

Die drei Delta-Tabellen werden von DuckDB eingelesen und anschließend mit Ibis zusammengeführt, verarbeitet und analysiert. Die dabei erstellten Analysen bieten wertvolle Einblicke für geschäftliche Anwendungen und repräsentieren das Gold-Layer der Medallion-Architektur.

```
Average Salary by Job Title:
                          job_title    avg_salary
0         Head of Machine Learning  6.000000e+06
1         Principal Data Architect  3.000000e+06
2   Lead Machine Learning Engineer  2.548667e+06
3              Lead Data Scientist  9.284853e+05
4              Data Analytics Lead  9.225000e+05
..                             ...           ...
88                 Insight Analyst  3.850000e+04
89         Compliance Data Analyst  3.000000e+04
90   Autonomous Vehicle Technician  2.627750e+04
91              Staff Data Analyst  1.500000e+04
92          Product Data Scientist  8.000000e+03

[93 rows x 2 columns]

Salary Distribution by Experience Level:
  experience_level     avg_salary  count
0               EN  192141.971537   3689
1               MI  191818.586011   3703
2               SE  176793.350136   3690
3               EX  174740.141116   3529

Salary Trends Over Years:
   work_year     avg_salary  count
0       2020  386352.750000     76
1       2021  544163.252174    230
2       2022  165421.016827   1664
3       2023  160381.480672   1785

Average Salary by Company Size:
  company_size     avg_salary  count
0            S  197081.792378  33248
1            L  191012.805115  64874
2            M  187013.825886  69219

Average Salary by Remote Ratio:
   remote_ratio     avg_salary  count
0           0.0  193779.568751  41723
1          50.0  193496.319889  50602
2         100.0  186798.608830  75016
```

Abb. 11: Analysen mittels DuckDB und Ibis zur Visualisierung in Apache Superset.

Die Visualisierung der Analyseergebnisse wird in Apache Superset realisiert, wo Dashboards und Diagramme erstellt werden, um detaillierte Einblicke in Gehaltsstrukturen und Arbeitsbedingungen zu bieten. Dieser Schritt bildet den Abschluss der Medallion-Architektur und demonstriert die vollständige Umsetzung einer Lakehouse-Architektur.

Aus den erstellten Charts und Dashboards lassen sich wertvolle Erkenntnisse ableiten, die eine fundierte Grundlage für strategische Entscheidungen bieten. Das erste Diagramm (Abbildung 12) zeigt die durchschnittlichen Gehälter nach Jobtiteln und verdeutlicht, welche Rollen innerhalb des Bereichs Data Science am höchsten vergütet werden. Diese Analyse kann dazu genutzt werden, strategische Personalentscheidungen zu treffen oder Gehaltspakete wettbewerbsfähig zu gestalten.

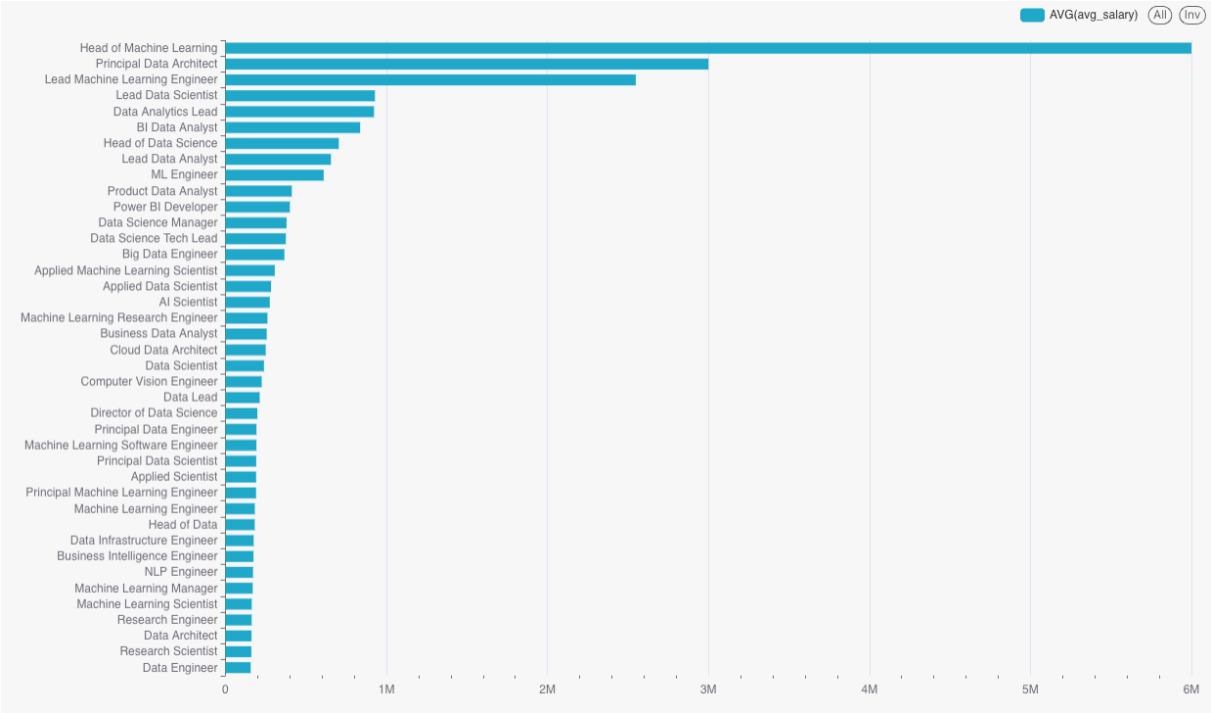


Abb. 12: Durchschnittsgehälter nach Jobpositionen.

Das zweite Diagramm (Abbildung 13) zeigt die Gehaltsentwicklung über die Jahre. Es wird ersichtlich, wie sich die durchschnittlichen Gehälter im Zeitverlauf verändert haben. Besonders auffällig ist der Höhepunkt im Jahr 2021, gefolgt von einem deutlichen Rückgang in den darauffolgenden Jahren, dies könnte jedoch auch auf fehlende Daten in diesem Jahr zurückzuführen sein.

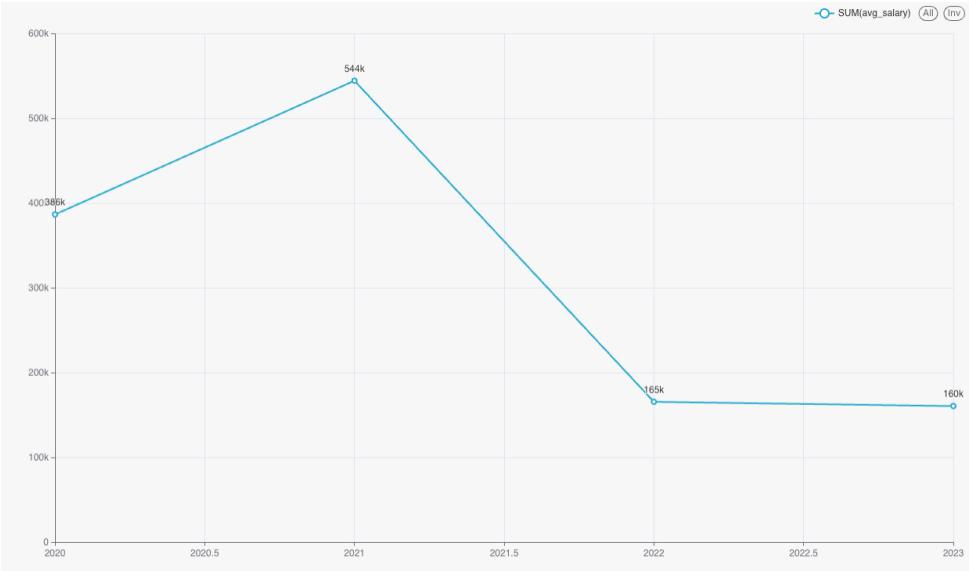


Abb. 13: Durchschnittliche Gehaltsentwicklung bei Data Science Jobs über die Jahre.

Im Anhang sind drei weitere Diagramme aus den Analysen mit DuckDB, Ibis und Apache Superset dargestellt (siehe Anhang 2). Diese Visualisierungen verdeutlichen die Vielseitigkeit und den Mehrwert der in der Lakehouse-Architektur implementierten Datenanalyse.

Abschließend bleibt anzumerken, dass die Konfiguration von Apache Superset den einzigen manuellen Schritt innerhalb der Umsetzung darstellt. Dabei müssen Datenquellen und Dashboards eingerichtet werden, um benutzerspezifische Visualisierungen zu ermöglichen. Während dieses Prozesses trat ein nicht behebbarer Fehler auf. Das Hinzufügen von Charts zu einem Dashboard führte wiederholt zu einem Core Dump, siehe Anhang 3. Weder die Log-Analyse noch Tests auf unterschiedlichen Endgeräten konnten das Problem lösen, was auf einen möglichen Fehler in Apache Superset hinweist. Dennoch konnte die Funktionalität der Lakehouse-Architektur erfolgreich demonstriert werden, da die Visualisierung der Daten weiterhin über die Erstellung von Charts möglich war.

Anhang

Anhangverzeichnis

Anhang 1: Docker-Compose Datei

```
services:
  storage:
    image: minio/minio:latest
    container_name: minio
    command: server --console-address ":9101" /data
    ports:
      - "9000:9000" # MinIO API
      - "9101:9101" # MinIO Console
    environment:
      MINIO_ROOT_USER: "${MINIO_ROOT_USER}"
      MINIO_ROOT_PASSWORD: "${MINIO_ROOT_PASSWORD}"
    volumes:
      - ./minio/data:/data
      - ./minio/config:/root/.minio
    networks:
      - lakehouse-network

  storage-init:
    image: minio/mc:latest
    container_name: minio-init
    depends_on:
      - storage
    entrypoint: >
      /bin/sh -c "
      sleep 5 &&
      mc alias set myminio http://minio:9000 ${MINIO_ROOT_USER} ${
         MINIO_ROOT_PASSWORD} &&
      if ! mc ls myminio/lakehouse-storage >/dev/null 2>&1; then
        mc mb myminio/lakehouse-storage;
      fi
      "
    networks:
      - lakehouse-network

  compute:
    build:
      context: ./compute
      dockerfile: Dockerfile
    container_name: duckdb-ibis-python
```

```
39      depends_on:
40        - storage
41      working_dir: /app
42      volumes:
43        - ./compute/src:/app/src
44        - ./compute/data:/app/data
45        - ./compute/duckdb-file:/app/duckdb-file
46      environment:
47        MINIO_URL: "http://minio:9000"
48        MINIO_ACCESS_KEY: "${MINIO_ACCESS_KEY}"
49        MINIO_SECRET_KEY: "${MINIO_SECRET_KEY}"
50      networks:
51        - lakehouse-network
52
53    visualisation:
54      image: apache/superset:latest
55      container_name: apache-superset
56      ports:
57        - "8088:8088"
58      depends_on:
59        - compute
60      environment:
61        SUPERSET_LOAD_EXAMPLES: "yes"
62        SUPERSET_ADMIN_USERNAME: "${SUPERSET_ADMIN_USERNAME}"
63        SUPERSET_ADMIN_PASSWORD: "${SUPERSET_ADMIN_PASSWORD}"
64        SUPERSET_SECRET_KEY: "${SUPERSET_SECRET_KEY}"
65      volumes:
66        - ./superset:/var/lib/superset
67        - ./compute/duckdb-file:/app/duckdb-file
68      networks:
69        - lakehouse-network
70      command: >
71        /bin/sh -c "
72        pip install duckdb duckdb-engine &&
73        superset db upgrade &&
74        superset fab create-admin --username ${SUPERSET_ADMIN_
            USERNAME} --password ${SUPERSET_ADMIN_PASSWORD} --
            firstname Admin --lastname User --email admin@example.com
            &&
75        superset init &&
76        superset run -h 0.0.0.0 -p 8088
```

```
77        "
78      deploy:
79        resources:
80          limits:
81            cpus: "2.0"    # Allocate 2 CPU cores
82            memory: "4g"   # Allocate 4 GB of RAM
83
84    spark-master:
85      build:
86        context: ./spark
87        dockerfile: Dockerfile
88      container_name: spark-master
89      environment:
90        - SPARK_MODE=master
91        - SPARK_MASTER_HOST=spark-master
92        - SPARK_MASTER_PORT=7077
93        - SPARK_WORKER_MEMORY=2G
94        - AWS_ACCESS_KEY_ID=${MINIO_ROOT_USER}
95        - AWS_SECRET_ACCESS_KEY=${MINIO_SECRET_KEY}
96        - MINIO_URL=http://minio:9000
97      ports:
98        - "8080:8080"   # Master web UI
99        - "7077:7077"   # Master RPC port
100       - "4040:4040"   # Driver web UI
101     volumes:
102       - ./spark/src:/app/src
103       - ./spark/data:/app/data
104       - ./spark/delta-data:/tmp/delta-table
105       - ./spark/logs:/opt/spark/logs
106     networks:
107       - lakehouse-network
108
109   spark-worker:
110     build:
111       context: ./spark
112       dockerfile: Dockerfile
113     container_name: spark-worker
114     depends_on:
115       - spark-master
116       - storage
117     environment:
```

```
118       - SPARK_MODE=worker
119       - SPARK_MASTER_URL=spark://spark-master:7077
120       - SPARK_WORKER_MEMORY=2G
121       - AWS_ACCESS_KEY_ID=${MINIO_ROOT_USER}
122       - AWS_SECRET_ACCESS_KEY=${MINIO_SECRET_KEY}
123       - MINIO_URL=http://minio:9000
124     volumes:
125       - ./spark/src:/app/src
126       - ./spark/data:/app/data
127       - ./spark/delta-data:/tmp/delta-table
128       - ./spark/logs:/opt/spark/logs
129     networks:
130       - lakehouse-network
131
132   spark-submit:
133     build:
134       context: ./spark
135       dockerfile: Dockerfile
136     container_name: spark-submit
137     volumes:
138       - ./spark/src:/app/src
139       - ./spark/data:/app/data
140       - ./spark/delta-data:/tmp/delta-table
141       - ./spark/logs:/opt/spark/logs
142     depends_on:
143       - spark-master
144       - spark-worker
145       - storage
146     environment:
147       - MINIO_URL=http://minio:9000
148       - MINIO_ACCESS_KEY=${MINIO_ACCESS_KEY}
149       - MINIO_SECRET_KEY=${MINIO_SECRET_KEY}
150     restart: always
151     networks:
152       - lakehouse-network
153     command: >
154       /bin/bash -c "
155       while true; do
156         echo 'Starting Spark job...';
157         spark-submit --master spark://spark-master:7077 \
158           --packages io.delta:delta-core_2.12:2.4.0,org.apache.
```

```
              hadoop:hadoop-aws:3.4.0,com.amazonaws:aws-java-sdk-
              bundle:1.12.277 \
            /app/src/main.py;
          echo 'Sleeping for 30 minutes...';
          sleep 1800;
        done
        "

  scraper:
    platform: linux/amd64
    build:
      context: ./scraper
      dockerfile: Dockerfile
    container_name: scraper
    volumes:
      - ./scraper/src:/app/src
      - ./spark/data:/app/data
    depends_on:
      - storage
    environment:
      - MINIO_ACCESS_KEY=${MINIO_ACCESS_KEY}
      - MINIO_URL=http://minio:9000
      - MINIO_SECRET_KEY=${MINIO_SECRET_KEY}
      - SELENIUM_SERVER_URL=http://selenium:4444/wd/hub
      - XDG_CACHE_HOME=/tmp/.cache
    restart: always
    networks:
      - lakehouse-network
    command: >
      /bin/bash -c "
      while true; do
        echo 'Starting scraper...';
        python3 /app/src/scrape_glasdoor.py;
        echo 'Sleeping for 30 minutes...';
        sleep 1800;
      done
      "

networks:
  lakehouse-network:
    driver: bridge
```

Anhang 2: Zusätzliche Diagramm-Analysen mit Apache Superset

Anhang 2/1: Gehaltsverteilung nach Erfahrungslevel

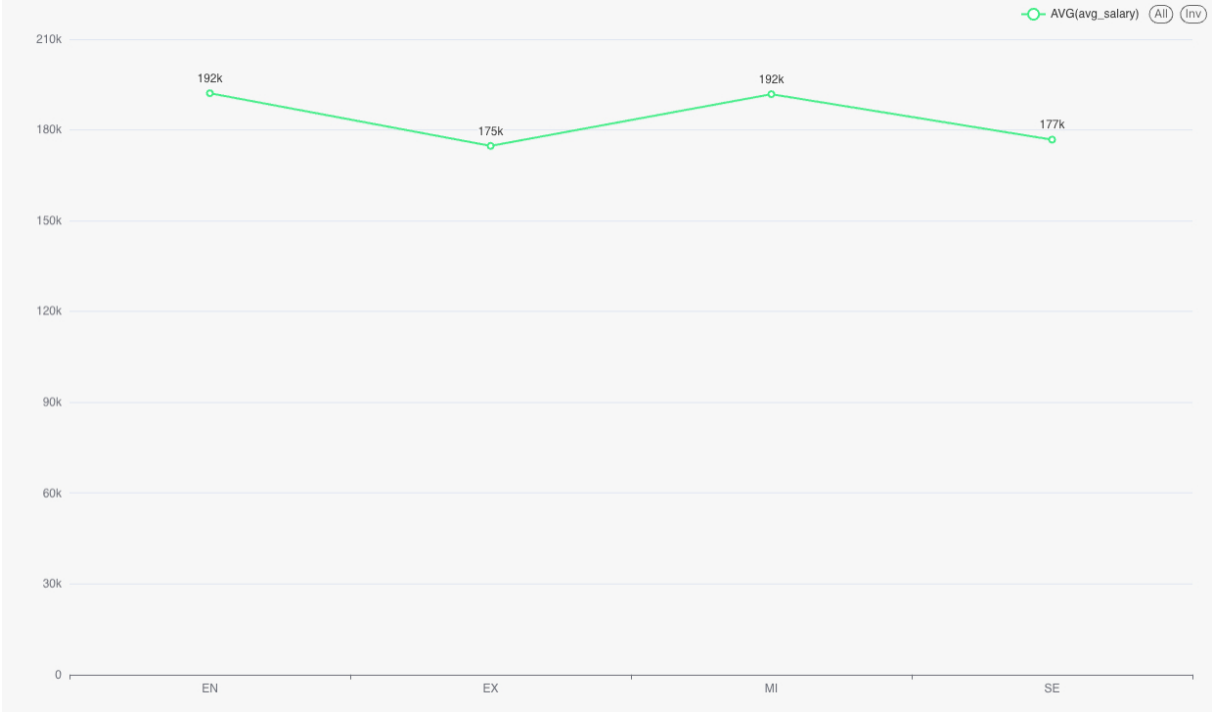


Abb. 14: Gehaltsverteilung nach Erfahrungslevel.

Anhang 2/2: Gehalt im Verhältnis zur prozentualen Remote-Arbeit

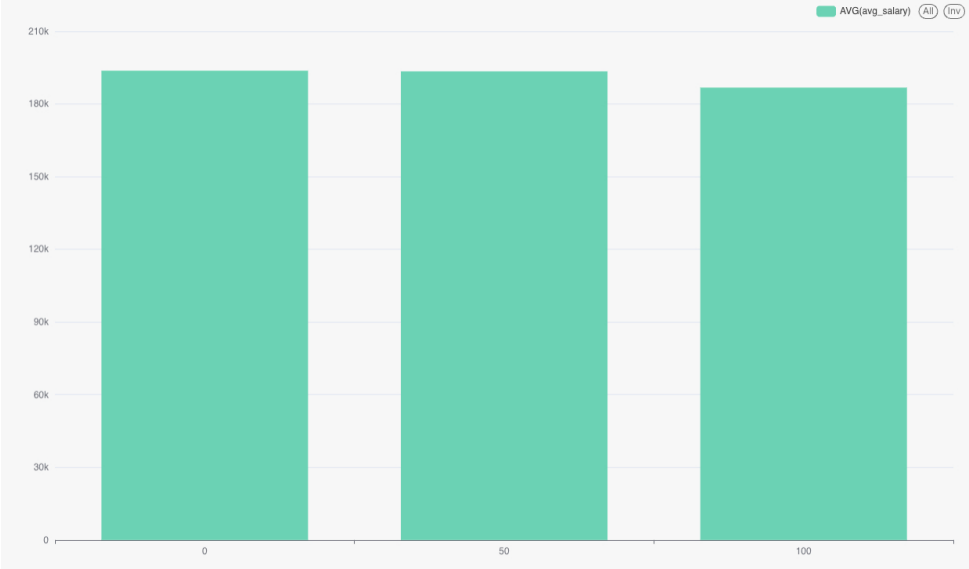


Abb. 15: Gehalt im Verhältnis zur prozentualen Remote-Arbeit.

Anhang 2/3: Representation der Arbeitnehmer nach Unternehmensgröße

Abb. 16: Representation der Arbeitnehmer nach Unternehmensgröße aus dem Datensatz.

Anhang 3: Core Dump beim erstellen von einem Dashboard in Apache Superset

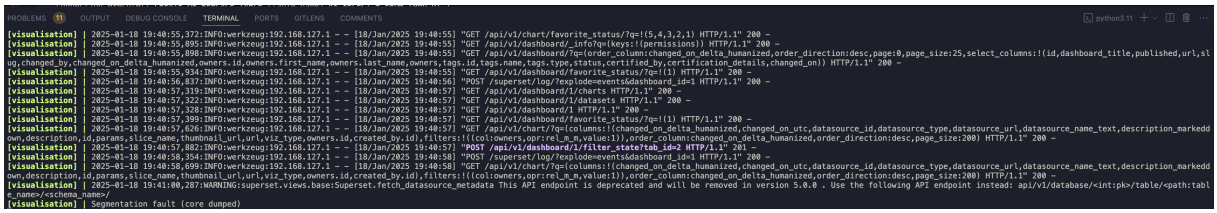


Abb. 17: Core Dump beim erstellen von Dashboard in Apache Superset. Als Fehlermeldung wird Segmentation fault (core dumped) angezeigt.

Literaturverzeichnis

Armbrust, M./Ghodsi, A./Xin, R./Zaharia, M. (2021): Lakehouse: A New Generation of Open Platforms That Unify Data Warehousing and Advanced Analytics. In.

Dixon, J. (2010): Pentaho, Hadoop, and Data Lakes. (Abruf: 17. 10. 2024).

Docker Inc. (2025): Develop faster. Run anywhere. Docker Inc. URL: https://www.docker.com/ (Abruf: 19. 01. 2025).

Harby, A. A./Zulkernine, F. (2022): From Data Warehouse to Lakehouse: A Comparative Review. In: *2022 IEEE International Conference on Big Data (Big Data)*, S. 389–395. DOI: 10.1109/BigData55660.2022.10020719.

Inmon, B. (2016): Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump. Technics Publications, LLC. ISBN: 1-63462-117-4.

Kimball, R./Ross, M. (2013): The data warehouse toolkit: The definitive guide to dimensional modeling. John Wiley & Sons.

Linstedt, D./Olschimke, M. (2015): Building a scalable data warehouse with data vault 2.0. Morgan Kaufmann.

Mazumdar, D./Hughes, J./Onofre, J. B. (2023): The Data Lakehouse: Data Warehousing and More. arXiv: 2310.08697 [cs]. (Abruf: 25. 07. 2024).

Oreščanin/Hlupić (2021): Data Lakehouse - a Novel Step in Analytics Architecture. In: *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, S. 1242–1246. ISBN: 2623-8764. DOI: 10.23919/MIPRO52101.2021.9597091.

Podman (2025): The best free & open source container tools. Red Hat. URL: https://podman.io/ (Abruf: 19. 01. 2025).

Vaisman, A./Zimnyi, E. (2014): Data Warehouse Systems: Design and Implementation. Springer Publishing Company, Incorporated. ISBN: 3-642-54654-4.

Erklärung zur Verwendung generativer KI-Systeme

Bei der Erstellung der eingereichten Arbeit habe ich die nachfolgend aufgeführten auf künstlicher Intelligenz (KI) basierten Systeme benutzt:

1. Consensus
2. ChatGPT-4o

Ich erkläre, dass ich

- mich aktiv über die Leistungsfähigkeit und Beschränkungen der oben genannten KI-Systeme informiert habe,[33]
- die aus den oben angegebenen KI-Systemen direkt oder sinngemäß übernommenen Passagen gekennzeichnet habe,
- überprüft habe, dass die mithilfe der oben genannten KI-Systeme generierten und von mir übernommenen Inhalte faktisch richtig sind,
- mir bewusst bin, dass ich als Autorin bzw. Autor dieser Arbeit die Verantwortung für die in ihr gemachten Angaben und Aussagen trage.

Die oben genannten KI-Systeme habe ich wie im Folgenden dargestellt eingesetzt:

Arbeitsschritt in der wissenschaftlichen Arbeit	**Eingesetzte(s) KI-System(e)**	**Beschreibung der Verwendungsweise**
Literaturrecherche	Consensus	Unterstützung bei der Suche nach wissenschaftlicher Literatur.
Korrektur der Arbeit	ChatGPT-4	Das erste Kapitel der Arbeit wurde ChatGPT zum Korrigieren auf Rechtschreibfehler gegeben.
Fehleranalyse von Programmcode	ChatGPT-4	Einige Programmzeilen wurden an ChatGPT übergeben, um sie auf Fehler zu prüfen.

[33] U.a. gilt es hierbei zu beachten, dass an KI weitergegebene Inhalte ggf. als Trainingsdaten genutzt und wiederverwendet werden. Dies ist insb. für betriebliche Aspekte als kritisch einzustufen.

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: *Open Source Lakehouse Container (mittels DuckDB)* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 19. Januar 2025
(Ort, Datum)

(Unterschrift)